Package: pliman (via r-universe)

October 16, 2026

Title Tools for Plant Image Analysis

Version 3.1.1

Description Tools for both single and batch image manipulation and analysis (Olivoto, 2022 <doi:10.1111/2041-210X.13803>) and phytopathometry (Olivoto et al., 2022 <doi:10.1007/S40858-021-00487-5>). The tools can be used for the quantification of leaf area, object counting, extraction of image indexes, shape measurement, object landmark identification, and Elliptical Fourier Analysis of object outlines (Claude (2008) <doi:10.1007/978-0-387-77789-4>). The package also provides a comprehensive pipeline for generating shapefiles with complex layouts and supports high-throughput phenotyping of RGB, multispectral, and hyperspectral orthomosaics. This functionality facilitates field phenotyping using UAV- or satellite-based imagery.

License GPL (>= 3)

URL https://nepem-ufsc.github.io/pliman/, https://github.com/nepem-ufsc/pliman

BugReports https://github.com/nepem-ufsc/pliman/issues

Depends R (>= 4.1)

Imports cli, dplyr, exactextractr, methods, mirai, purrr, Rcpp, sf, terra

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

LinkingTo Rcpp, RcppArmadillo

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 7.3.2

NeedsCompilation yes

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

Maintainer Tiago Olivoto <tiagoolivoto@gmail.com>

Repository https://cran.r-universe.dev

Date/Publication 2025-08-23 23:10:02 UTC

RemoteUrl https://github.com/cran/pliman

RemoteRef HEAD

RemoteSha 49fda6a611a8ea6bf1242b989ae8bd6909aeec8f

Contents

analyze_objects 5
analyze_objects_minimal 18
analyze_objects_shp 24
apply_fun_to_imgs 28
as_image 29
calibrate 30
ccc 31
clear_pliman_cache 32
contours 33
custom_palette 34
dist_transform 34
efourier 35
efourier_coefs 37
efourier_error 37
efourier_inv 39
efourier_norm 40
efourier_power 41
efourier_shape 42
ellipse 44
entropy 45
get_pliman_viewer 45
get_uuid 46
ggplot_color 46
image_align 47
image_alpha 48
image_augment 49
image_binary 50
image_canny_edge 53
image_combine 54
image_contour_line 55
image_create 55
image_expand 56
image_index 57
image_label 60
image_line_segment 61

image_prepare . . . 62
image_segment . . . 63
image_segment_kmeans . . . 67
image_segment_manual . . . 68
image_segment_mask . . . 69
image_shp . . . 71
image_square . . . 72
image_thinning_guo_hall . . . 73
image_to_mat . . . 74
image_view . . . 75
landmarks . . . 76
landmarks_add . . . 78
landmarks_angle . . . 79
landmarks_dist . . . 80
landmarks_regradi . . . 81
leading_zeros . . . 82
line_on_halfplot . . . 83
make_brush . . . 83
make_mask . . . 84
measure_disease . . . 85
measure_disease_byl . . . 91
measure_disease_shp . . . 95
measure_injury . . . 97
mosaic_aggregate . . . 99
mosaic_analyze . . . 100
mosaic_analyze_iter . . . 106
mosaic_chm . . . 108
mosaic_chm_extract . . . 110
mosaic_chm_mask . . . 111
mosaic_classify . . . 112
mosaic_clip . . . 113
mosaic_crop . . . 114
mosaic_draw . . . 116
mosaic_epsg . . . 118
mosaic_extract . . . 119
mosaic_hist . . . 119
mosaic_index . . . 120
mosaic_index2 . . . 122
mosaic_input . . . 123
mosaic_interpolate . . . 124
mosaic_lonlat2epsg . . . 125
mosaic_plot . . . 126
mosaic_plot_rgb . . . 127
mosaic_prepare . . . 127
mosaic_project . . . 129
mosaic_resample . . . 130
mosaic_rotate . . . 130
mosaic_segment . . . 131

mosaic_segment_pick . . . 133
mosaic_to_pliman . . . 134
mosaic_to_rgb . . . 135
mosaic_vectorize . . . 136
mosaic_view . . . 138
object_bbox . . . 140
object_edge . . . 141
object_export . . . 142
object_export_shp . . . 145
object_label . . . 147
object_map . . . 149
object_mark . . . 150
object_rgb . . . 151
object_scatter . . . 152
object_split . . . 154
object_split_shp . . . 156
object_to_color . . . 157
otsu . . . 159
palettes . . . 159
pipe . . . 161
pixel_index . . . 162
pliman_images . . . 163
pliman_indexes_ican_compute . . . 164
pliman_viewer . . . 165
plot.image_shp . . . 165
plot_bbox . . . 166
plot_id . . . 167
plot_index . . . 168
plot_index_shp . . . 170
plot_line_segment . . . 172
plot_lw . . . 172
poly_apex_base_angle . . . 173
poly_pcv . . . 174
poly_width_at . . . 175
prepare_to_shp . . . 177
random_color . . . 177
sad . . . 178
sentinel_to_tif . . . 179
separate_col . . . 180
set_pliman_viewer . . . 181
shapefile_build . . . 181
shapefile_edit . . . 184
shapefile_interpolate . . . 185
shapefile_measures . . . 186
shapefile_operations . . . 187
shapefile_plot . . . 188
shapefile_surface . . . 189
summary_index . . . 190

utils_colorspace . . . 191
utils_dpi . . . 192
utils_file . . . 195
utils_image . . . 197
utils_indexes . . . 199
utils_measures . . . 199
utils_objects . . . 202
utils_pca . . . 205
utils_pick . . . 207
utils_polygon . . . 210
utils_polygon_plot . . . 216
utils_rows_cols . . . 217
utils_shapefile . . . 218
utils_shapes . . . 220
utils_stats . . . 222
utils_transform . . . 223
utils_wd . . . 230
uuid . . . 231
watershed2 . . . 232

Index **233**

`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_img = NULL,
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,
smooth = FALSE,
pcv = FALSE,
pcv_niter = 100,
resize = FALSE,
trim = FALSE,
fill_hull = FALSE,
erode = FALSE,
dilate = FALSE,
opening = FALSE,
closing = 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,
  show_bbox = FALSE,
  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"),
  ...
)

## S3 method for class 'anal_obj_ls'
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, reference_img	A color palette for the foregrond, background, and reference object, 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. Other allowed value is `har_band = "GRAY"`.

`smooth` whether the object contours should be smoothed with `poly_smooth()`. Defaults to `FALSE`. To smooth use a numeric value indicating the number of interactions used to smooth the contours.

`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.

`opening`, `closing`, `filter`, `erode`, `dilate`

Morphological operations (brush size)

- `dilate` puts the mask over every background pixel, and sets it to foreground if any of the pixels covered by the mask is from the foreground.
- `erode` puts the mask over every foreground pixel, and sets it to background if any of the pixels covered by the mask is from the background.
- `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects.
- `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects.
- `filter` performs median filtering in the binary image. 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. 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_size = 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`.

`show_bbox`
: Show the bounding box around the objects? Defaults to `FALSE`.

`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. Additionaly, har_band = "GRAY" can be used. In this case, a grayscale (0.299 * R + 0.587 * G + 0.114 * B) is used.
  - 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

```
if (interactive() && requireNamespace("EBImage")) {
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 ob
ject
# 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)
}

if (interactive() && requireNamespace("EBImage")) {
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_minimal` *Analyzes objects in an image*

Description

A lighter option to `analyze_objects()`

Usage

```
analyze_objects_minimal(
  img,
  segment_objects = TRUE,
  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,
  fill_hull = FALSE,
  opening = FALSE,
  closing = FALSE,
  filter = FALSE,
  erode = FALSE,
  dilate = FALSE,
  invert = FALSE,
  object_size = "medium",
  index = "NB",
  r = 1,
  g = 2,
  b = 3,
  re = 4,
  nir = 5,
  threshold = "Otsu",
  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,
  plot = TRUE,
  show_original = TRUE,
  show_contour = TRUE,
  contour_col = "red",
  contour_size = 1,
  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_minimal'
```

```
plot(
  x,
  which = "measure",
  measure = "area",
  type = c("density", "histogram"),
  ...
)

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

Arguments

`img`	The image to be analyzed.
`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.
`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.

`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.

`opening`, `closing`, `filter`, `erode`, `dilate`

Morphological operations (brush size)

- `dilate` puts the mask over every background pixel, and sets it to foreground if any of the pixels covered by the mask is from the foreground.
- `erode` puts the mask over every foreground pixel, and sets it to background if any of the pixels covered by the mask is from the background.
- `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects.
- `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects.
- `filter` performs median filtering in the binary image. 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. 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.

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.

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_size = 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).

`plot`
: Show image after processing?

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

`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`.

`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()`.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Examples

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
img <- image_pliman("soybean_touch.jpg")
obj <- analyze_objects(img)
obj$statistics

}

if (interactive() && requireNamespace("EBImage")) {
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_minimal(img)
# density of area
plot(rgb)

# histogram of area
plot(rgb, type = "histogram") # or 'hist'
}
```

`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,
  opening = FALSE,
  closing = FALSE,
  filter = FALSE,
  erode = FALSE,
  dilate = 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.

`opening`, `closing`, `filter`, `erode`, `dilate`

Morphological operations (brush size)

- `dilate` puts the mask over every background pixel, and sets it to foreground if any of the pixels covered by the mask is from the foreground.
- `erode` puts the mask over every foreground pixel, and sets it to background if any of the pixels covered by the mask is from the background.
- `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects.
- `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects.
- `filter` performs median filtering in the binary image. 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.

`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() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
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
}
```

---

ccc *Lin's Concordance Correlation Coefficient (CCC)*

Description

Computes Lin's Concordance Correlation Coefficient (CCC) between observed and predicted values. Also returns Pearson's correlation coefficient and root mean squared error (RMSE). If the input is a grouped data frame (`grouped_df`), the function will return results for each group.

Usage

```
ccc(data, real, predito)
```

Arguments

`data`	A data frame containing the columns for observed and predicted values.
`real`	The column name (unquoted) corresponding to the observed values.
`predito`	The column name (unquoted) corresponding to the predicted values.

Details

The CCC is defined as:

$$\rho_c = \frac{2 \cdot \text{Cov}(x, y)}{\text{Var}(x) + \text{Var}(y) + (\bar{x} - \bar{y})^2}$$

where:

- $\text{Cov}(x, y)$ is the covariance between observed and predicted values
- $\text{Var}(x)$ and $\text{Var}(y)$ are the variances of the observed and predicted values
- $\bar{x}$ and $\bar{y}$ are the means of the observed and predicted values

Value

A data frame with the following columns:

- `r`: Pearson correlation coefficient
- `ccc`: Lin's Concordance Correlation Coefficient
- `rmse`: Root mean squared error

Examples

```
library(dplyr)
library(pliman)
df <- data.frame(
  group = rep(c("A", "B"), each = 5),
  real = c(1:5, 2:6),
  predicted = c(1.1, 2, 2.9, 4.1, 5, 2.2, 3.1, 4, 4.8, 6.1)
)

# Without grouping
ccc(df, real, predicted)

# With grouping
df |>
  group_by(group) |>
  ccc(real, predicted)
```

---

clear_pliman_cache *Clear cached files created by pliman*

Description

Deletes cached `.rds` files used by functions such as `object_scatter()`. You can either remove the entire cache directory or only files older than a given number of days.

Usage

```
clear_pliman_cache(all = TRUE, days = NULL)
```

Arguments

`all`	Logical. If `TRUE` (default), deletes the entire cache directory.
`days`	Integer (optional). If provided, removes only files older than `days`. Ignored if `all = TRUE`.

Value

Invisibly returns TRUE if the operation was successful.

Examples

```
if(interactive()){
# Clear everything
clear_pliman_cache()

# Clear only files older than 7 days
clear_pliman_cache(all = FALSE, days = 7)
}
```

---

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("yellow", "#53CC67", "#009B95", "#00588B", "#4B0055"),
  n = 5
)
```

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

```
if (interactive() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
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
- `min_harm` The minimum number of harmonics to achieve a given power.

References

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

Examples

```
if (interactive() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
ellipse(contours)
}
```

`entropy` *Compute Shannon Entropy*

Description

This function calculates the Shannon entropy of a numeric vector.

Usage

```
entropy(x)
```

Arguments

`x`	A numeric vector containing the values for which entropy will be computed.

Value

A numeric value representing the entropy.

Examples

```
library(pliman)
x <- c(1, 2, 2, 3, 3, 3, 4, 4, 4, 4)
entropy(x)
```

`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.

get_uuid *Extract UUID from filenames*

Description

This function extracts a UUID (Universal Unique Identifier) from the filename, using a regular expression that specifically identifies the standard UUID format.

Usage

```
get_uuid(filename)
```

Arguments

filename	A character vector containing filenames or strings

Value

A character vector with extracted UUIDs (or NA if none found)

Examples

```
library(pliman)
file <- "Grãos - contagem_f68bca60-c8cf-4272-9448-3f28891a97cd.jpg"
file2 <- "Grãos - contagem_f68bca60-c8cf-4272-9448-3f8891a97cd.jpg"
get_uuid(file)
```

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 < 90 degrees 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() && requireNamespace("EBImage")) {
library(pliman)
flax <- image_pliman("flax_leaves.jpg", plot = TRUE)
aligned <- image_align(flax)
}
```

`image_alpha` *Add Alpha Layer to an RGB Image*

Description

This function adds an alpha (transparency) layer to an RGB image using the EBImage package. The alpha layer can be specified as a single numeric value for uniform transparency or as a matrix/array matching the dimensions of the image for varying transparency.

Usage

```
image_alpha(img, mask)
```

Arguments

`img`	An RGB image of class `Image` from the EBImage package. The image must be in RGB format (color mode 2).
`mask`	A numeric value or matrix/array specifying the alpha layer: * If `mask` is a single numeric value, it sets a uniform transparency level (0 for fully transparent, 1 for fully opaque). * If `mask` is a matrix or array, it must have the same dimensions as the image channels, allowing for varying transparency.

Value

An `Image` object with an added alpha layer, maintaining the RGBA format.

Examples

```
if (interactive() && requireNamespace("EBImage")) {
# Load the EBImage package
library(pliman)

# Load a sample RGB image
img <- image_pliman("soybean_touch.jpg")

# 50% transparency
image_alpha(img, 0.5) |> plot()

# transparent background
mask <- image_binary(img, "NB")[[1]]
img_tb <- image_alpha(img, mask)
```

```
plot(img_tb)

}
```

`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,
  workers = NULL
)
```

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.
`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.

Value

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

Examples

```
if (interactive() && requireNamespace("EBImage")) {
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,
  erode = FALSE,
  dilate = FALSE,
  opening = FALSE,
  closing = 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`.
`erode`, `dilate`, `opening`, `closing`, `filter`	**Morphological operations (brush size)** • `dilate` puts the mask over every background pixel, and sets it to foreground if any of the pixels covered by the mask is from the foreground. • `erode` puts the mask over every foreground pixel, and sets it to background if any of the pixels covered by the mask is from the background. • `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects.

- `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects.
- `filter` performs median filtering in the binary image. 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.

Hierarchically, the operations are performed as opening > closing > filter. The value declared in each argument will define the brush size.

`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

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
img <- image_pliman("soybean_touch.jpg")
image_binary(img, index = c("R, G"))
}
```

image_canny_edge *Canny Edge Detector*

Description

Canny Edge Detector for Images. Adapted from https://github.com/bnosac/image/tree/master/image.CannyEdges.

Usage

```
image_canny_edge(img, index = "GRAY", s = 5, low_thr = 10, high_thr = 20)
```

Arguments

img	An Image object.
index	A character string with the index to be used. Defaults to "GRAY".
s	sigma, the Gaussian filter variance. Defaults to 5.
low_thr	lower threshold value of the algorithm. Defaults to 10.
high_thr	upper threshold value of the algorithm. Defaults to 20

Value

a list with an Image object with values 0 or 255, and the number of pixels which have value 255 (pixels_nonzero).

Examples

```
if(interactive()){
library(pliman)
img <- image_pliman("sev_leaf.jpg")
conts <- image_canny_edge(img, index = "B")
par(mfrow = c(1, 2))
plot(img)
plot(conts$edges)
par(mfrow = c(1, 1))
}
```

`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

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
img1 <- image_pliman("sev_leaf.jpg")
img2 <- image_pliman("sev_leaf_nb.jpg")
image_combine(img1, img2)
}
```

image_contour_line *Smooth Contour Line Detection*

Description

Smooth Contour Line Detection

Usage

```
image_contour_line(img, index = "GRAY", Q = 2)
```

Arguments

img	An Image object.
index	A character string with the index to be used. Defaults to "GRAY".
Q	numeric value with the pixel quantization step

Value

A list with the contour lines.

Examples

```
if(interactive()){
library(pliman)
img <- image_pliman("sev_leaf.jpg")
conts <- image_contour_line(img, index = "B")
plot(img)
plot_contour(conts, col = "black")
}
```

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

```
if (interactive() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
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. See this page for a detailed list of available indexes.

The S3 method `plot()` can be used to generate 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

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

Karcher, D.E., and M.D. Richardson. 2003. Quantifying Turfgrass Color Using Digital Image Analysis. Crop Science 43(3): 943–951. doi:10.2135/cropsci2003.9430

Bannari, A., D. Morin, F. Bonn, and A.R. Huete. 1995. A review of vegetation indices. Remote Sensing Reviews 13(1–2): 95–120. doi:10.1080/02757259509532298

Examples

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
img <- image_pliman("soybean_touch.jpg")
image_index(img, index = c("R, NR"))
```

```
}
if (interactive() && requireNamespace("EBImage")) {
# 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_label` *Label Connected Components in a Binary Image*

Description

This function labels connected components in a binary image while allowing for a specified maximum gap between pixels to still be considered part of the same object.

Usage

```
image_label(img, max_gap = 0)
```

Arguments

`img`	A binary image matrix where `1` represents foreground pixels and `0` represents background pixels. This should be compatible with the EBImage package.
`max_gap`	An integer specifying the maximum allowable gap (in pixels) between connected components to be considered as part of the same object. Default is `1`.

Value

An object of class `Image` (from the EBImage package), where each connected component is assigned a unique integer label.

Examples

```
if(interactive()){
library(pliman)
img <- matrix(c(
  1, 1, 0, 0, 0, 1, 1, 1, 0,
  0, 0, 0, 0, 0, 1, 0, 0, 0,
  1, 1, 0, 0, 1, 1, 1, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 1
), nrow = 4, byrow = TRUE)
```

```
image_label(img, max_gap = 1)
image_label(img, max_gap = 2)
image_label(img, max_gap = 3)
}
```

`image_line_segment` *Line Segment Detection in an Image*

Description

Detects line segments in a digital image using the Line Segment Detector (LSD), a linear-time method that controls false detections and requires no parameter tuning. Based on Burns, Hanson, and Riseman's method with an a-contrario validation approach.

Usage

```
image_line_segment(
  img,
  index = "GRAY",
  scale = 0.8,
  sigma_scale = 0.6,
  quant = 2,
  ang_th = 22.5,
  log_eps = 0,
  density_th = 0.7,
  n_bins = 1024,
  union = FALSE,
  union_min_length = 5,
  union_max_distance = 5,
  union_ang_th = 7,
  union_use_NFA = FALSE,
  union_log_eps = 0
)
```

Arguments

`img`	An `Image` object.
`index`	A character string with the index to be used. Defaults to `"GRAY"`.
`scale`	A positive numeric value. Scales the input image before detection using Gaussian filtering. A value <1 downscales, >1 upscales. Default is 0.8.
`sigma_scale`	A positive numeric value determining the Gaussian filter sigma. If scale <1, sigma = sigma_scale / scale; otherwise, sigma = sigma_scale. Default is 0.6.
`quant`	A positive numeric value controlling gradient quantization error. Default is 2.0.
`ang_th`	A numeric value (0-180) defining the gradient angle tolerance in degrees. Default is 22.5.

log_eps	A numeric detection threshold. Larger values make detection stricter. Default is 0.0.
density_th	A numeric value (0-1) defining the minimum proportion of supporting points in a rectangle. Default is 0.7.
n_bins	A positive integer specifying the number of bins for pseudo-ordering gradient modulus. Default is 1024.
union	Logical. If TRUE, merges close line segments. Default is FALSE.
union_min_length	Numeric. Minimum segment length to merge. Default is 5.
union_max_distance	Numeric. Maximum distance between segments to merge. Default is 5.
union_ang_th	Numeric. Angle threshold for merging segments. Default is 7.
union_use_NFA	Logical. If TRUE, uses NFA in merging. Default is FALSE.
union_log_eps	Numeric. Detection threshold for merging. Default is 0.0.

Value

A list of class `lsd` containing:

- `n` - Number of detected line segments.
- `lines` - A matrix with detected segments (columns: x1, y1, x2, y2, width, p, -log_nfa).
- `pixels` - A matrix assigning each pixel to a detected segment (0 = unused pixels).

References

Grompone von Gioi, R., Jakubowicz, J., Morel, J.-M., & Randall, G. (2010). LSD: A Fast Line Segment Detector with a False Detection Control. IEEE Transactions on Pattern Analysis and Machine Intelligence, 32(4), 722-732.doi:10.5201/ipol.2012.gjmrlsd

Examples

```
library(pliman)
```

---

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(),
  downsample = NULL,
  max_pixels = 1e+06
)
```

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.
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.
max_pixels	integer > 0. Maximum number of cells to use for the plot. If max_pixels < npixels(img), regular sampling is used before plotting.

Value

The alighed/cropped image for further visualization or analysis.

Examples

```
# Example usage:
if (interactive() && requireNamespace("EBImage")) {
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,
  na_background = FALSE,
  has_white_bg = FALSE,
  fill_hull = FALSE,
  erode = FALSE,
  dilate = FALSE,
  opening = FALSE,
  closing = 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).
`na_background`	Consider the background as NA? Defaults to FALSE.
`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`.
`erode`, `dilate`, `opening`, `closing`, `filter`	**Morphological operations (brush size)** • `dilate` puts the mask over every background pixel, and sets it to foreground if any of the pixels covered by the mask is from the foreground. • `erode` puts the mask over every foreground pixel, and sets it to background if any of the pixels covered by the mask is from the background. • `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects. • `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects.

- `filter` performs median filtering in the binary image. 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.

Hierarchically, the operations are performed as opening > closing > filter. The value declared in each argument will define the brush size.

`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

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
img <- image_pliman("soybean_touch.jpg", plot = TRUE)
image_segment(img, index = c("R, G, B"))
}
```

`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,
  opening = FALSE,
  closing = FALSE,
  filter = FALSE,
  erode = FALSE,
  dilate = 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.
`erode`, `dilate`, `opening`, `closing`, `filter`	**Morphological operations (brush size)** • `dilate` puts the mask over every background pixel, and sets it to foreground if any of the pixels covered by the mask is from the foreground. • `erode` puts the mask over every foreground pixel, and sets it to background if any of the pixels covered by the mask is from the background. • `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects. • `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects. • `filter` performs median filtering in the binary image. 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.

	Hierarchically, the operations are performed as opening > closing > filter. The value declared in each argument will define the brush size.
`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

```
if (interactive() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
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,
  edit = FALSE,
  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),
  ...
)
```

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`.

edit	If TRUE enable editing options using mapedit::editMap().
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. Set to c(0, 1)
...	Additional arguments to be passed to downsample_fun.

Value

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

Examples

```
if (interactive() && requireNamespace("EBImage")) {
# 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)
```

line_on_halfplot *Extract mid-lines from half-plots*

Description

For each polygon in an `sf` object, computes the line segment joining the midpoints of the longer pair of opposite edges (the "half-plot line").

Usage

```
line_on_halfplot(shapefile)
```

Arguments

`shapefile`	An `sf` object of polygons. Each geometry must be closed (first and last coordinate coincide) so that `st_coordinates(...)` yields a repeating start point.

Value

A `SpatVector` (from the **terra** package) of line geometries representing the half-plot midlines.

Examples

```
if(interactive()){
library(pliman)
shp <- shapefile_input( paste0(image_pliman(), "/soy_shape.rds"))
mosaic <- mosaic_input( paste0(image_pliman(), "/soy_dsm.tif"))
mosaic_plot(mosaic)
half <- line_on_halfplot(shp)
shapefile_plot(half, add = TRUE, col = "blue")

}
```

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

```
if (interactive() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
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,
  opening = c(10, 0),
  closing = c(0, 0),
  filter = c(0, 0),
  erode = c(0, 0),
  dilate = c(0, 0),
  parallel = FALSE,
  workers = NULL,
  resize = FALSE,
  fill_hull = TRUE,
  index_lb = NULL,
  index_dh = "GLI",
  has_white_bg = FALSE,
  threshold = NULL,
  invert = FALSE,
```

```
  lower_noise = 0.1,
  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 = 3,
  by_leaf = FALSE,
  viewer = get_pliman_viewer(),
  opening = c(10, 0),
  closing = c(0, 0),
  filter = c(0, 0),
  erode = c(0, 0),
  dilate = c(0, 0),
  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.
`erode`, `dilate`, `opening`, `closing`, `filter`	**Morphological operations (brush size)** • `dilate` puts the mask over every background pixel, and sets it to foreground if any of the pixels covered by the mask is from the foreground. • `erode` puts the mask over every foreground pixel, and sets it to background if any of the pixels covered by the mask is from the background. • `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects. • `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects. • `filter` performs median filtering in the binary image. 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. Hierarchically, the operations are performed as opening > closing > filter. The value declared in each argument will define the brush size.
`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_noise — By default, lesions with lesser than 10% of the mean area of all lesions are removed (lower_noise = 0.1). Increasing this value will remove larger lesions. To define an explicit lower or upper size (in pixel unit), use the lower_size and upper_size arguments.

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.

`by_leaf` Compute the severity by leaf? If `TRUE`, `measure_disease_byl()` is called internallty and the severity is computed for each object (leaf) in the image. The background segmentation is then controlled by the argument `index`.

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

```
if (interactive() && requireNamespace("EBImage")) {
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,
  opening = c(10, 0),
  closing = c(0, 0),
  filter = c(0, 0),
```

```
  erode = c(0, 0),
  dilate = c(0, 0),
  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.

`opening`, `closing`, `filter`, `erode`, `dilate`

Morphological operations (brush size)

- `dilate` puts the mask over every background pixel, and sets it to foreground if any of the pixels covered by the mask is from the foreground.
- `erode` puts the mask over every foreground pixel, and sets it to background if any of the pixels covered by the mask is from the background.
- `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects.
- `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects.
- `filter` performs median filtering in the binary image. 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.

`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

```
if (interactive() && requireNamespace("EBImage")) {
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() && requireNamespace("EBImage")) {
# 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
}
```

`measure_injury` *Measures Injury in Images*

Description

The `measures_injury` function calculates the percentage of injury in images by performing binary segmentation and identifying lesions. It processes either a single image or a batch of images specified by a pattern in a directory.

Usage

```
measure_injury(
  img = NULL,
  pattern = NULL,
  index = "GRAY",
  threshold = "Otsu",
  invert = FALSE,
  opening = 5,
  closing = FALSE,
  filter = FALSE,
  erode = FALSE,
  dilate = FALSE,
  plot = TRUE,
  dir_original = NULL,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE
)
```

Arguments

`img`	The image to be analyzed.
`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).
`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"`
`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.
`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).
`opening`, `closing`, `filter`, `erode`, `dilate`	**Morphological operations (brush size)** • `dilate` puts the mask over every background pixel, and sets it to foreground if any of the pixels covered by the mask is from the foreground. • `erode` puts the mask over every foreground pixel, and sets it to background if any of the pixels covered by the mask is from the background. • `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects. • `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects. • `filter` performs median filtering in the binary image. 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.
`plot`	Show image after processing?

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

Details

The function processes each image by reading it, applying binary segmentation to detect lesions, filling the segmented areas, calculating the injury percentage, and optionally saving the processed image with highlighted lesions. In batch mode, it uses the provided pattern to identify images in the specified directory and can utilize parallel processing for efficiency.

Value

A numeric value representing the injury percentage for a single image, or a data frame with injury percentages for batch processing.

---

mosaic_aggregate *SpatRaster aggregation*

Description

Aggregate a SpatRaster to create a new SpatRaster with a lower resolution (larger cells), using the GDAL's gdal_translate utility https://gdal.org/programs/gdal_translate.html

Usage

```
mosaic_aggregate(mosaic, pct = 50, fun = "nearest", in_memory = TRUE)
```

Arguments

mosaic	SpatRaster
pct	The size as a fraction (percentage) of the input image size. Either a scalar (eg., 50), or a length-two numeric vector. In the last, different percentage reduction/expansion can be used for columns, and rows, respectively.

fun	The resampling function. Defaults to nearest, which applies the nearest neighbor (simple sampling) resampler. Other accepted values are: 'average', 'rms', 'bilinear', 'cubic', 'cubicspline', 'lanczos', and 'mode'. See Details for a detailed explanation.
in_memory	Wheter to return an 'in-memory' SpatRaster. If FALSE, the aggregated raster will be returned as an 'in-disk' object.

Value

SpatRaster

Examples

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
library(terra)
r <- rast()
values(r) <- 1:ncell(r)
r2 <- mosaic_aggregate(r, pct = 10)
opar <- par(no.readonly = TRUE)
par(mfrow=c(1,2))
mosaic_plot(r)
mosaic_plot(r2)
par(opar)
}
```

---

mosaic_analyze *Analyze a mosaic of remote sensing data*

Description

This function analyzes a mosaic of remote sensing data (UVAs or satellite imagery), extracting information from specified regions of interest (ROIs) defined in a shapefile or interactively drawn on the mosaic. It allows counting and measuring individuals (eg., plants), computing canopy coverage, and statistical summaries (eg., mean, coefficient of variation) for vegetation indices (eg, NDVI) at a block, plot, individual levels or even extract the raw results at pixel level.

Usage

```
mosaic_analyze(
  mosaic,
  r = 3,
  g = 2,
  b = 1,
  re = NA,
  nir = NA,
  swir = NA,
  tir = NA,
```

```
  crop_to_shape_ext = TRUE,
  grid = TRUE,
  nrow = 1,
  ncol = 1,
  plot_width = NULL,
  plot_height = NULL,
  layout = "lrtb",
  indexes = NULL,
  shapefile = NULL,
  basemap = NULL,
  build_shapefile = TRUE,
  check_shapefile = TRUE,
  buffer_edge = 1,
  buffer_col = 0,
  buffer_row = 0,
  segment_plot = FALSE,
  segment_individuals = FALSE,
  segment_pick = FALSE,
  mask = NULL,
  dsm = NULL,
  dsm_lower = 0.2,
  dsm_upper = NULL,
  dsm_window_size = c(5, 5),
  simplify = FALSE,
  map_individuals = FALSE,
  map_direction = c("horizontal", "vertical"),
  watershed = TRUE,
  tolerance = 1,
  extension = 1,
  include_if = "centroid",
  plot_index = "GLI",
  segment_index = NULL,
  threshold = "Otsu",
  opening = FALSE,
  closing = FALSE,
  filter = FALSE,
  erode = FALSE,
  dilate = FALSE,
  lower_noise = 0.15,
  lower_size = NULL,
  upper_size = NULL,
  topn_lower = NULL,
  topn_upper = NULL,
  summarize_fun = "mean",
  summarize_quantiles = NULL,
  attribute = NULL,
  invert = FALSE,
  color_regions = rev(grDevices::terrain.colors(50)),
```

```
  alpha = 1,
  max_pixels = 2e+06,
  downsample = NULL,
  quantiles = c(0, 1),
  plot = TRUE,
  verbose = TRUE
)
```

Arguments

mosaic	A mosaic of class SpatRaster, generally imported with mosaic_input().
r, g, b, re, nir, swir, tir	The red, green, blue, red-edge, near-infrared, shortwave Infrared, and thermal infrared bands of the image, respectively. By default, the function assumes a BGR as input (b = 1, g = 2, r = 3). If a multispectral image is provided up to seven bands can be used to compute built-in indexes. There are no limitation of band numbers if the index is computed using the band name.
crop_to_shape_ext	Crop the mosaic to the extension of shapefile? Defaults to TRUE. This allows for a faster index computation when the region of the built shapefile is much smaller than the entire mosaic extension.
grid	Logical, indicating whether to use a grid for segmentation (default: TRUE).
nrow	Number of rows for the grid (default: 1).
ncol	Number of columns for the grid (default: 1).
plot_width, plot_height	The width and height of the plot shape (in the mosaic unit). It is mutually exclusiv with buffer_col and buffer_row.
layout	Character: one of • 'tblr' for top/bottom left/right orientation • 'tbrl' for top/bottom right/left orientation • 'btlr' for bottom/top left/right orientation • 'btrl' for bottom/top right/left orientation • 'lrtb' for left/right top/bottom orientation • 'lrbt' for left/right bottom/top orientation • 'rltb' for right/left top/bottom orientation • 'rlbt' for right/left bottom/top orientation
indexes	An optional SpatRaster object with the image indexes, computed with mosaic_index().
shapefile	An optional shapefile containing regions of interest (ROIs) for analysis.
basemap	An optional basemap generated with mosaic_view().
build_shapefile	Logical, indicating whether to interactively draw ROIs if the shapefile is NULL (default: TRUE).
check_shapefile	Logical, indicating whether to validate the shapefile with an interactive map view (default: TRUE). This enables live editing of the drawn shapefile by deleting or changing the drawn grids.

`buffer_edge` Width of the buffer around the shapefile (default: 5).

`buffer_col, buffer_row`
Buffering factor for the columns and rows, respectively, of each individual plot'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 plot by 25% on each side.

`segment_plot` Logical, indicating whether to segment plots (default: FALSE). If `TRUE`, the `segment_index` will be computed, and pixels with values below the `threshold` will be selected.

`segment_individuals`
Logical, indicating whether to segment individuals within plots (default: FALSE). If `TRUE`, the `segment_index` will be computed, and pixels with values below the `threshold` will be selected, and a watershed-based segmentation will be performed.

`segment_pick` When `segment_plot` or `segment_individuals` are `TRUE`, `segment_pick` allows segmenting background (eg., soil) and foreground (eg., plants) interactively by picking samples from background and foreground using `mosaic_segment_pick()`

`mask` An optional mask (SpatRaster) to mask the mosaic.

`dsm` A SpatRaster object representing the digital surface model. Must be a single-layer raster. If a DSM is informed, a mask will be derived from it using `mosaic_chm_mask()`.

`dsm_lower` A numeric value specifying the lower height threshold. All heights greater than this value are retained.

`dsm_upper` An optional numeric value specifying the upper height threshold. If provided, only heights between lower and upper are retained.

`dsm_window_size`
An integer (meters) specifying the window size (rows and columns, respectively) for creating a DTM using a moving window. Default is c(5, 5).

`simplify` Removes vertices in polygons to form simpler shapes. The function implementation uses the Douglas-Peucker algorithm using `sf::st_simplify()` for simplification.

`map_individuals`
If `TRUE`, the distance between objects within plots is computed. The distance can be mapped either in the horizontal or vertical direction. The distances, coefficient of variation (CV), and mean of distances are then returned.

`map_direction` The direction for mapping individuals within plots. Should be one of `"horizontal"` or `"vertical"` (default).

`watershed` If `TRUE` (default), performs watershed-based object detection. This will detect objects even when they are touching one another. 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. A higher value smooths out small objects.

`include_if` Character vector specifying the type of intersection. Defaults to "centroid" (individuals in which the centroid is included within the drawn plot will be included in that plot). Other possible values include `"covered"`, `"overlap"`, and `"intersect"`. See Details for a detailed explanation of these intersecting controls.

`plot_index` The index(es) to be computed for the drawn plots. Either a single vegetation index (e.g., `"GLAI"`), a vector of indexes (e.g., `c("GLAI", "NGRDI", "HUE")`), or a custom index based on the available bands (e.g., `"(R-B)/(R+B)"`). See `pliman_indexes()` and `image_index()` for more details.

`segment_index` The index used for segmentation. The same rule as `plot_index`. Defaults to `NULL`

`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 provided, this value will be used as a threshold.

`opening`, `closing`, `filter`, `erode`, `dilate`

Morphological operations (brush size)

- `dilate` puts the mask over every background pixel, and sets it to foreground if any of the pixels covered by the mask is from the foreground.
- `erode` puts the mask over every foreground pixel, and sets it to background if any of the pixels covered by the mask is from the background.
- `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects.
- `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects.
- `filter` performs median filtering in the binary image. 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.

`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_size = 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.

summarize_fun
: The function to compute summaries for the pixel values. Defaults to "mean," i.e., the mean value of the pixels (either at a plot- or individual-level) is returned.

summarize_quantiles
: quantiles to be computed when 'quantile' is on summarize_fun.

attribute
: The attribute to be shown at the plot when plot is TRUE. Defaults to the first summary_fun and first segment_index.

invert
: Logical, indicating whether to invert the mask. Defaults to FALSE, i.e., pixels with intensity greater than the threshold values are selected.

color_regions
: The color palette for regions (default: rev(grDevices::terrain.colors(50))).

alpha
: opacity of the fill color of the raster layer(s).

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.

quantiles
: the upper and lower quantiles used for color stretching.

plot
: Logical, indicating whether to generate plots (default: TRUE).

verbose
: Logical, indicating whether to display verbose output (default: TRUE).

Details

Since multiple blocks can be analyzed, the length of arguments grid, nrow, ncol, buffer_edge, , buffer_col, buffer_row, segment_plot, segment_i, ndividuals, includ_if, threshold, segment_index, invert, filter, threshold, lower_size, upper_size, watershed, and lower_noise, can be either an scalar (the same argument applied to all the drawn blocks), or a vector with the same length as the number of drawn. In the last, each block can be analyzed with different arguments.

When segment_individuals = TRUE is enabled, individuals are included within each plot based on the include_if argument. The default value ('centroid') includes an object in a given plot if the centroid of that object is within the plot. This makes the inclusion mutually exclusive (i.e., an individual is included in only one plot). If 'covered' is selected, objects are included only if their entire area is covered by the plot. On the other hand, selecting overlap is the complement of covered; in other words, objects that overlap the plot boundary are included. Finally, when intersect is chosen, objects that intersect the plot boundary are included. This makes the inclusion ambiguous (i.e., an object can be included in more than one plot).

Value

A list containing the following objects:

- result_plot: The results at a plot level.

- result_plot_summ: The summary of results at a plot level. When segment_individuals = TRUE, the number of individuals, canopy coverage, and mean values of some shape statistics such as perimeter, length, width, and diameter are computed.
- result_individ: The results at an individual level.
- map_plot: An object of class mapview showing the plot-level results.
- map_individual: An object of class mapview showing the individual-level results.
- shapefile: The generated shapefile, with the drawn grids/blocks.

Examples

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
url <- "https://github.com/TiagoOlivoto/images/raw/master/pliman/rice_field/rice_ex.tif"
mosaic <- mosaic_input(url)
# Draw a polygon (top left, top right, bottom right, bottom left, top left)
# include 8 rice lines and one column
res <-
 mosaic_analyze(mosaic,
                r = 1, g = 2, b = 3,
                segment_individuals = TRUE,     # segment the individuals
                segment_index = "(G-B)/(G+B-R)",# index for segmentation
                filter = 4,
                nrow = 8,
                map_individuals = TRUE)
# map with individual results
res$map_indiv
}
```

---

mosaic_analyze_iter *Analyze mosaics iteratively*

Description

High-resolution mosaics can take a significant amount of time to analyze, especially when segment_individuals = TRUE is used in mosaic_analyze(). This is because the function needs to create in-memory arrays to segment individual using the watershed algorithm. This process utilizes a for-loop approach, iteratively analyzing each shape within the mosaic one at a time. To speed up processing, the function crops the original mosaic to the extent of the current shape before analyzing it. This reduces the resolution for that specific analysis, sacrificing some detail for faster processing.

Usage

```
mosaic_analyze_iter(
  mosaic,
  shapefile,
  basemap = NULL,
  r = 3,
```

```
  g = 2,
  b = 1,
  re = NA,
  nir = NA,
  swir = NA,
  tir = NA,
  plot = TRUE,
  verbose = TRUE,
  max_pixels = 3e+06,
  attribute = NULL,
  summarize_fun = "mean",
  segment_plot = FALSE,
  segment_individuals = FALSE,
  segment_index = "VARI",
  plot_index = "VARI",
  color_regions = rev(grDevices::terrain.colors(50)),
  alpha = 0.75,
  quantiles = c(0, 1),
  parallel = FALSE,
  workers = NULL,
  ...
)
```

Arguments

mosaic	A mosaic of class SpatRaster, generally imported with mosaic_input().
shapefile	An optional shapefile containing regions of interest (ROIs) for analysis.
basemap	An optional basemap generated with mosaic_view().
r, g, b, re, nir, swir, tir	The red, green, blue, red-edge, near-infrared, shortwave Infrared, and thermal infrared bands of the image, respectively. By default, the function assumes a BGR as input (b = 1, g = 2, r = 3). If a multispectral image is provided up to seven bands can be used to compute built-in indexes. There are no limitation of band numbers if the index is computed using the band name.
plot	Logical, indicating whether to generate plots (default: TRUE).
verbose	Logical, indicating whether to display verbose output (default: TRUE).
max_pixels	Maximum number of pixels to render in the map or plot (default: 500000).
attribute	The attribute to be shown at the plot when plot is TRUE. Defaults to the first summary_fun and first segment_index.
summarize_fun	The function to compute summaries for the pixel values. Defaults to "mean," i.e., the mean value of the pixels (either at a plot- or individual-level) is returned.
segment_plot	Logical, indicating whether to segment plots (default: FALSE). If TRUE, the segment_index will be computed, and pixels with values below the threshold will be selected.
segment_individuals	Logical, indicating whether to segment individuals within plots (default: FALSE). If TRUE, the segment_index will be computed, and pixels with values below the

	threshold will be selected, and a watershed-based segmentation will be performed.
segment_index	The index used for segmentation. The same rule as plot_index. Defaults to NULL
plot_index	The index(es) to be computed for the drawn plots. Either a single vegetation index (e.g., "GLAI"), a vector of indexes (e.g., c("GLAI", "NGRDI", "HUE")), or a custom index based on the available bands (e.g., "(R-B)/(R+B)"). See pliman_indexes() and image_index() for more details.
color_regions	The color palette for regions (default: rev(grDevices::terrain.colors(50))).
alpha	opacity of the fill color of the raster layer(s).
quantiles	the upper and lower quantiles used for color stretching.
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.
...	Further arguments passed on to mosaic_analyze()

Value

A list containing the following objects:

- result_plot: The results at a plot level.
- result_plot_summ: The summary of results at a plot level. When segment_individuals = TRUE, the number of individuals, canopy coverage, and mean values of some shape statistics such as perimeter, length, width, and diameter are computed.
- result_individ: The results at an individual level.
- map_plot: An object of class mapview showing the plot-level results.
- map_individual: An object of class mapview showing the individual-level results.

---

mosaic_chm *Calculate Canopy Height Model and Volume*

Description

This function calculates the canopy height model (CHM) and the volume for a given digital surface model (DSM) raster layer. Optionally, a digital terrain model (DTM) can be provided or interpolated using a set of points or a moving window.

Usage

```
mosaic_chm(
  dsm,
  dtm = NULL,
  points = NULL,
  interpolation = c("Tps", "Kriging"),
  window_size = c(5, 5),
  ground_quantile = 0,
  mask = NULL,
  mask_soil = TRUE,
  verbose = TRUE
)
```

Arguments

dsm	A SpatRaster object representing the digital surface model. Must be a single-layer raster.
dtm	(optional) A SpatRaster object representing the digital terrain model. Must be a single-layer raster. If not provided, it can be interpolated from points or created using a moving window.
points	(optional) An sf object representing sample points for DTM interpolation. If provided, dtm will be interpolated using these points.
interpolation	(optional) A character string specifying the interpolation method to use when points are provided. Options are "Kriging" (default) or "Tps" (Thin Plate Spline).
window_size	An integer (meters) specifying the window size (rows and columns, respectively) for creating a DTM using a moving window. Default is c(10, 10).
ground_quantile	Numeric value between 0 and 1 indicating the quantile threshold for ground point selection in the CHM computation. Lower values (e.g., 0) retain the lowest ground points, while higher values (e.g., 1) consider higher ground elevations. Default is 0, which uses the lowest points within each window.
mask	(optional) A SpatRaster object used to mask the CHM and volume results. Default is NULL.
mask_soil	Is mask representing a soil mask (eg., removing plants)? Default is TRUE.
verbose	Return the progress messages. Default is TRUE.

Details

The function first checks if the input dsm is a valid single-layer SpatRaster object. If dtm is not provided, The function generates a Digital Terrain Model (DTM) from a Digital Surface Model (DSM) by downsampling and smoothing the input raster data. It iterates over the DSM matrix in windows of specified size, finds the minimum value within each window, and assigns these values to a downsampled matrix. After downsampling, the function applies a mean filter to smooth the matrix, enhancing the visual and analytical quality of the DTM. Afterwards, DTM is resampled with the original DSM.

If both dsm and dtm are provided, the function ensures they have the same extent and number of cells, resampling dtm if necessary. The CHM is then calculated as the difference between dsm and dtm, and the volume is calculated by multiplying the CHM by the pixel size. The results are optionally masked using the provided mask.

Value

A SpatRaster object with three layers: dtm (digital terrain model), height (canopy height model), and volume.

---

mosaic_chm_extract *Extracts height metrics and plot quality from a Canopy Height Model (CHM)*

Description

This function extracts height-related summary statistics from a CHM using a given shapefile.

Usage

```
mosaic_chm_extract(chm, shapefile, chm_threshold = NULL)
```

Arguments

chm	An object computed with mosaic_chm().
shapefile	An sf object containing the polygons over which height metrics are extracted.
chm_threshold	A numeric value representing the height threshold for calculating coverage. If NULL, coverage is not computed.

Value

An sf object containing height summary statistics for each plot, including:

- min: Minimum height value.
- q05: 5th percentile height value.
- q50: Median height value.
- q95: 95th percentile height value.
- max: Maximum height value.
- mean: Mean height value.
- volume: Total sum of heights multiplied by CHM resolution.
- coverage: If a mask is used in mosaic_chm() or chm_threshold is informed, returns the proportion of pixels covered within the plot. Otherwise, returns 1.

`mosaic_chm_mask` *Apply a height mask to CHM data*

Description

This function applies a height-based mask to a Canopy Height Model (CHM), focusing on areas with heights above a specified `lower` threshold and, optionally, below an `upper` threshold.

Usage

```
mosaic_chm_mask(
  dsm,
  lower,
  upper = NULL,
  window_size = c(5, 5),
  interpolation = "Tps"
)
```

Arguments

`dsm`	A `SpatRaster` object representing the digital surface model. Must be a single-layer raster.
`lower`	A numeric value specifying the lower height threshold. All heights greater than this value are retained.
`upper`	An optional numeric value specifying the upper height threshold. If provided, only heights between `lower` and `upper` are retained.
`window_size`	An integer (meters) specifying the window size (rows and columns, respectively) for creating a DTM using a moving window. Default is c(10, 10).
`interpolation`	(optional) A character string specifying the interpolation method to use when `points` are provided. Options are "Kriging" (default) or "Tps" (Thin Plate Spline).

Details

The `mosaic_chm` function, used internally, generates the DTM from the DSM by downsampling and smoothing raster data, applying a moving window to extract minimum values and then interpolating the results. The CHM is computed as the height difference between the DSM and DTM. This function calculates and applies a mask based on height thresholds.

Value

An `SpatRaster` object representing the masked CHM.

mosaic_classify *Classify a Mosaic Based on Index Breaks*

Description

This function classifies a given raster mosaic based on user-defined breaks. It provides an option to calculate the frequency and area of each class, as well as plot the classified mosaic.

Usage

```
mosaic_classify(mosaic, breaks, frequency = TRUE, plot = TRUE)
```

Arguments

mosaic	A SpatRaster object representing the mosaic to be classified.
breaks	A numeric vector specifying the breakpoints for classification.
frequency	Logical. If TRUE, computes the class frequency and area (in hectares).
plot	Logical. If TRUE, plots the classified mosaic.

Value

A list with two elements:

- classified: A SpatRaster object containing the classified mosaic.
- class_freq: A data frame containing class frequencies, areas (ha), and percentages (if frequency = TRUE).

Examples

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

# Create an example raster
r <- terra::rast(matrix(runif(100, min = 0, max = 1), nrow=10, ncol=10))

# Classify the raster
result <- mosaic_classify(r, breaks = c(0.3, 0.6))

# View results
result$classified
result$class_freq
}
```

`mosaic_clip` *Clip Raster Mosaic by Polygons*

Description

Quickly partition a large raster mosaic into individual tiles using a polygon layer. Each tile is clipped by either the polygon's bounding box or (optionally) the exact feature geometry, and written to disk as a separate GeoTIFF named by the feature's `unique_id`.

Usage

```
mosaic_clip(
  mosaic,
  shapefile,
  unique_id = "unique_id",
  out_dir = NULL,
  overwrite = TRUE,
  verbose = TRUE,
  exact = FALSE,
  parallel = FALSE,
  workers = NULL
)
```

Arguments

`mosaic`	A terra::SpatRaster object or a file path pointing to a raster. In-memory rasters are first written to a temporary GeoTIFF.
`shapefile`	An sf::sf, terra::SpatVector, or path to a vector file. Must contain a column named `unique_id` for naming each output tile.
`unique_id`	A column present in `shapefile` that uniquely identifies the plots to be clipped.
`out_dir`	Directory where clipped rasters will be saved. Defaults to the current working directory. Created recursively if it does not exist.
`overwrite`	Logical; if `TRUE` (the default), existing files in `out_dir` with the same name will be overwritten.
`verbose`	Logical; if `TRUE` (default), progress bars and status messages will be shown.
`exact`	Logical; if `FALSE` (default), tiles are cropped by each feature's bounding box. If `TRUE`, the function extracts each polygon as a cutline for an exact crop (slower, but shape-accurate).
`parallel`	Logical; if `TRUE` (default), processing is parallelized using `mirai`. Set to `FALSE` for purely sequential execution.
`workers`	Integer; number of parallel daemons to launch when `parallel = TRUE`. Defaults to 70% of available cores.

Details

Clip a Raster Mosaic by Polygons

This function wraps GDAL's `warp` utility for efficient raster clipping. When `parallel = TRUE`, it will spawn multiple workers via `mirai` and process tiles in batches. Use `exact = TRUE` to clip to the true polygon shape (at some extra cost), or leave `exact = FALSE` for a faster bounding-box crop.

Value

Invisibly returns a character vector of file paths to all clipped GeoTIFFs.

`mosaic_crop` *Crop or Mask a Mosaic Raster*

Description

This function allows cropping of a raster mosaic interactively or programmatically:

- **Interactive Mode**: If neither `shapefile` nor `mosaic2` is provided, an interactive map is shown via `mosaic_view()`, allowing users to draw a rectangle to define the cropping area.
- **Shapefile Mode**: If a `SpatVector` is provided in `shapefile`, cropping or masking is performed based on its extent or exact shape, optionally with a buffer.
- **Raster Mode**: If `mosaic2` is provided, `mosaic` will be cropped to match the extent of `mosaic2`.

For disk-based mosaics, cropping with shapefiles uses GDAL (`sf::gdal_utils()`) to improve efficiency.

Usage

```
mosaic_crop(
  mosaic,
  r = 3,
  g = 2,
  b = 1,
  re = 4,
  nir = 5,
  shapefile = NULL,
  in_memory = FALSE,
  mosaic2 = NULL,
  buffer = 0,
  show = c("rgb", "index"),
  index = "R",
  max_pixels = 5e+05,
  downsample = NULL,
  type = c("crop", "mask"),
  ...
)
```

Arguments

mosaic	A SpatRaster object to be cropped.
r, g, b, re, nir	Integer indices representing the red, green, blue, red-edge, and near-infrared bands of the input mosaic. Default assumes BGR format (b = 1, g = 2, r = 3).
shapefile	An optional SpatVector (or sf object) to use as cropping/masking geometry. Can be created interactively with shapefile_input().
in_memory	Logical. If TRUE, raster processing will occur entirely in memory using terra. If FALSE (default), disk-based processing with GDAL will be used when appropriate.
mosaic2	A second SpatRaster whose extent will be used to crop mosaic.
buffer	A numeric value indicating a buffer (in CRS units) to apply around the shapefile geometry.
show	A character value indicating what to display in the interactive viewer. Either "rgb" or "index".
index	The index to show if show = "index". Default is "R".
max_pixels	Maximum number of pixels to render in the interactive viewer.
downsample	Optional downsampling factor for display purposes.
type	Either "crop" (default) or "mask": • "crop" crops the mosaic to the bounding box of the shapefile. • "mask" sets pixels outside the shapefile geometry to NA (recommended when using exact shapes).
...	Additional arguments passed to mosaic_view().

Details

Crop or mask a SpatRaster object (mosaic) based on user input from an interactive map or by using a provided shapefile or another raster.

Value

A cropped or masked SpatRaster object.

Examples

```
if (interactive() && requireNamespace("EBImage")) {
  library(pliman)
  # Load a sample raster
  mosaic <- mosaic_input(system.file("ex/elev.tif", package = "terra"))

  # Interactive cropping with drawn rectangle
  cropped <- mosaic_crop(mosaic)

  # View result
  mosaic_view(cropped)
}
```

mosaic_draw *Drawing Lines or Polygons with Raster Information*

Description

Drawing Lines or Polygons with Raster Information

Usage

```
mosaic_draw(
  mosaic,
  r = 3,
  g = 2,
  b = 1,
  re = 4,
  nir = 5,
  index = "NGRDI",
  show = "rgb",
  segment = FALSE,
  viewer = c("mapview", "base"),
  threshold = "Otsu",
  invert = FALSE,
  summarize_fun = NULL,
  buffer = 2,
  color_regions = rev(grDevices::terrain.colors(50)),
  alpha = 1,
  max_pixels = 1e+06,
  downsample = NULL,
  quantiles = c(0, 1),
  plot = TRUE,
  plot_layout = c(1, 2, 3, 3)
)
```

Arguments

mosaic	A mosaic of class SpatRaster, generally imported with mosaic_input().
r, g, b, re, nir	The red, green, blue, red-edge, and near-infrared bands of the image, respectively. By default, the function assumes a BGR as input (b = 1, g = 2, r = 3). If a multispectral image is provided up to seven bands can be used to compute built-in indexes. There are no limitation of band numbers if the index is computed using the band name.
index	The index to use for the index view. Defaults to "B".
show	The display option for the map view. Options are "rgb" for RGB view and "index" for index view.
segment	Should the raster object be segmented? If set to TRUE, pixels within each polygon/rectangle will be segmented based on the threshold argument.

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.
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.
invert	Inverts the mask if desired. Defaults to FALSE.
summarize_fun	An optional function or character vector. When summarize_fun = "mean", the mean values of index are calculated within each object. For more details on available functions, refer to exactextractr::exact_extract().
buffer	Adds a buffer around the geometries of the SpatVector created. Note that the distance unit of buffer will vary according to the CRS of mosaic.
color_regions	The color palette for displaying index values. Defaults to rev(grDevices::terrain.colors(50)).
alpha	opacity of the fill color of the raster layer(s).
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.
quantiles	the upper and lower quantiles used for color stretching.
plot	Plots the draw line/rectangle? Defaults to TRUE.
plot_layout	The de plot layout. Defaults to plot_layout = c(1, 2, 3, 3). Ie., the first row has two plots, and the second row has one plot.

Details

The mosaic_draw function enables you to create mosaic drawings from remote sensing data and compute vegetation indices.

- If a line is drawn using the "Draw Polyline" tool, the profile of index is displayed on the y-axis along the line's distance, represented in meter units. It is important to ensure that the Coordinate Reference System (CRS) of mosaic has latitude/longitude units for accurate distance representation.
- If a rectangle or polygon is drawn using the "Draw Rectangle" or "Draw Polygon" tools, the index values are calculated for each object. By default, the raw data is returned. You can set the summarize_fun to compute a summary statistic for each object.

Value

An invisible list containing the mosaic, draw_data, distance, distance_profile, geometry, and map.

Examples

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

# draw a polyline to see the elevation profile along the line
mosaic_draw(mosaic, buffer = 1500)
}
```

`mosaic_epsg` *Determine EPSG Code for a Mosaic*

Description

This function calculates the EPSG code for a given mosaic based on its geographic extent.

Usage

```
mosaic_epsg(mosaic)
```

Arguments

`mosaic`	A raster object representing the mosaic for which the EPSG code is to be determined.

Details

The function calculates the centroid of the mosaic's extent, determines the UTM zone based on the centroid's longitude, and identifies the hemisphere based on the centroid's latitude. The EPSG code is then constructed accordingly.

Value

A character string representing the EPSG code corresponding to the UTM zone and hemisphere of the mosaic's centroid. If the mosaic is not in the lon/lat coordinate system, a warning is issued.

Examples

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
library(terra)

# Create a sample mosaic
mosaic <- rast(nrow=10, ncol=10, xmin=-120, xmax=-60, ymin=30, ymax=60)

# Get the EPSG code for the mosaic
mosaic_epsg(mosaic)
}
```

mosaic_extract *Extract Values from a Raster Mosaic Using a Shapefile*

Description

This function extracts values from a raster mosaic based on the regions defined in a shapefile using `exactextractr::exact_extract()`.

Usage

```
mosaic_extract(mosaic, shapefile, fun = "median", ...)
```

Arguments

`mosaic`	A `SpatRaster` object representing the raster mosaic from which values will be extracted.
`shapefile`	A shapefile, which can be a `SpatVector` or an `sf` object, defining the regions of interest for extraction.
`fun`	A character string specifying the summary function to be used for extraction. Default is `"median"`.
`...`	Additional arguments to be passed to `exactextractr::exact_extract()`.

Value

A data frame containing the extracted values for each region defined in the shapefile.

mosaic_hist *A wrapper around terra::hist()*

Description

Create a histogram of the values of a `SpatRaster`.

Usage

```
mosaic_hist(mosaic, layer, ...)
```

Arguments

`mosaic`	SpatRaster
`layer`	positive integer or character to indicate layer numbers (or names). If missing, all layers are used
`...`	Further arguments passed on to `terra::hist()`.

Value

A NULL object

Examples

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
r <- mosaic_input(system.file("ex/elev.tif", package="terra"))
mosaic_hist(r)
}
```

`mosaic_index` *Mosaic Index*

Description

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

Usage

```
mosaic_index(
  mosaic,
  index = "NGRDI",
  r = 3,
  g = 2,
  b = 1,
  re = NA,
  nir = NA,
  swir = NA,
  tir = NA,
  plot = TRUE,
  in_memory = TRUE,
  output = c("memory", "disk"),
  workers = 1,
  verbose = TRUE
)
```

Arguments

mosaic	A mosaic of class SpatRaster, generally imported with mosaic_input().
index	A character value (or a vector of characters) specifying the target mode for conversion to a binary image. Use pliman_indexes_rgb() and pliman_indexes_me() to see the available RGB and multispectral indexes, respectively. Users can also calculate their own index using R, G, B, RE, NIR, SWIR, and TIR bands (eg., index = "R+B/G") or using the names of the mosaic's layers (ex., "(band_1 + band_2) / 2").

r, g, b, re, nir, swir, tir
: The red, green, blue, red-edge, near-infrared, shortwave Infrared, and thermal infrared bands of the image, respectively. By default, the function assumes a BGR as input (b = 1, g = 2, r = 3). If a multispectral image is provided up to seven bands can be used to compute built-in indexes. There are no limitation of band numbers if the index is computed using the band name.

plot
: Plot the computed index? Defaults to TRUE.

in_memory
: Logical, indicating whether the indexes should be computed in memory. Defaults to TRUE. In most cases, this is 2-3 times faster, but errors can occur if mosaic is a large SpatRaster. If FALSE, raster algebra operations are performed on temporary files.

output
: Character(1), either "memory" or "disk". If "memory", the function returns a terra::SpatRaster object assembled in memory. If "disk", each index layer is written out to a temporary GeoTIFF and the function returns a terra::SpatRaster object that points to those rasters. Default is "memory".

workers
: numeric. The number of workers you want to use for parallel processing when computing multiple indexes.

verbose
: Whether to display progress messages.

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

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))
names(mosaic)
elev2 <- mosaic_index(mosaic, "elevation * 5", plot = FALSE)
oldpar <- par(no.readonly=TRUE)
par(mfrow=c(1,2))

mosaic_plot(mosaic)
mosaic_plot(elev2)

# return the original parameters
par(oldpar)
}
```

mosaic_index2 *Mosaic Index with GDAL*

Description

Compute or extract an index layer from a multi-band mosaic raster using gdal_calc.py (https://gdal.org/programs/gdal_calc.ht
This requires a Python and GDAL installation.

Usage

```
mosaic_index2(
  mosaic,
  index = "B",
  r = 3,
  g = 2,
  b = 1,
  re = 4,
  nir = 5,
  plot = TRUE,
  python = Sys.which("python.exe"),
  gdal = Sys.which("gdal_calc.py")
)
```

Arguments

mosaic	A mosaic of class SpatRaster, generally imported with mosaic_input().
index	A character value (or a vector of characters) specifying the target mode for conversion to a binary image. Use pliman_indexes_rgb() and pliman_indexes_me() to see the available RGB and multispectral indexes, respectively. Users can also calculate their own index using R, G, B, RE, NIR, SWIR, and TIR bands (eg., index = "R+B/G") or using the names of the mosaic's layers (ex., "(band_1 + band_2) / 2").
r, g, b, re, nir	The red, green, blue, red-edge, and near-infrared bands of the image, respectively. By default, the function assumes a BGR as input (b = 1, g = 2, r = 3). If a multispectral image is provided up to seven bands can be used to compute built-in indexes. There are no limitation of band numbers if the index is computed using the band name.
plot	Plot the computed index? Defaults to TRUE.
python	The PATH for python.exe
gdal	The PATH for gdal_calc.py

Value

An index layer extracted/computed from the mosaic raster.

Examples

```
if(interactive() & (Sys.which('python.exe') != '' ) & (Sys.which('gdal_calc.py') != '' )){
library(pliman)
mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))
names(mosaic) <- "R"
elev2 <- mosaic_index2(mosaic, "R * 5", plot = FALSE)
oldpar <- par(no.readonly=TRUE)
mosaic_plot(mosaic)
mosaic_plot(elev2)
par(mfrow=c(1,2))
}
```

`mosaic_input` *Create and Export mosaics*

Description

Create and Export mosaics

Usage

```
mosaic_input(
  mosaic,
  mosaic_pattern = NULL,
  info = TRUE,
  check_16bits = FALSE,
  check_datatype = FALSE,
  ...
)

mosaic_export(mosaic, filename, datatype = NULL, 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.
`mosaic_pattern`	A pattern name to import multiple mosaics into a list.
`info`	Print the mosaic informations (eg., CRS, extent). Defaults to `TRUE`
`check_16bits`	Checks if mosaic has maximum value in the 16-bits format (65535), and replaces it by NA. Defaults to `FALSE`.
`check_datatype`	Logical. If `TRUE`, checks and suggests the appropriate data type based on the raster values.
`...`	Additional arguments passed to `terra::rast()` (`mosaic_input()`) or `terra::writeRaster()` (`mosaic_output()`)

filename	character. The Output filename.
datatype	The datatype. By default, the function will try to guess the data type that saves more memory usage and file size. See terra::writeRaster() and terra::datatype() for more details.
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

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)

# create an SpatRaster object based on a matrix
x <- system.file("ex/logo.tif", package="terra")
rast <- mosaic_input(x)
mosaic_plot(rast)

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

---

mosaic_interpolate *Mosaic interpolation*

Description

Performs the interpolation of points from a raster object.

Usage

```
mosaic_interpolate(mosaic, points, method = c("bilinear", "loess", "idw"))
```

Arguments

mosaic	An SpatRaster object
points	An sf object with the points for x and y coordinates, usually obtained with shapefile_build(). Alternatively, an external shapefile imported with shapefile_input() containing the x and y coordinates can be used. The function will handle most used shapefile formats (eg., .shp, .rds) and convert the imported shapefile to an sf object.
method	One of "bilinear" (default), "loess" (local regression) or "idw" (Inverse Distance Weighting).

Value

An SpatRaster object with the same extent and crs from mosaic

---

mosaic_lonlat2epsg *Project a Mosaic from Lon/Lat to EPSG-based CRS*

Description

This function projects a given mosaic from the lon/lat coordinate system to an EPSG-based CRS determined by the mosaic's extent.

Usage

```
mosaic_lonlat2epsg(mosaic)
```

Arguments

mosaic	A raster object representing the mosaic to be projected. The mosaic must be in the lon/lat coordinate system.

Value

A raster object representing the projected mosaic. If the mosaic is not in the lon/lat coordinate system, a warning is issued.

Examples

```
if (interactive() && requireNamespace("EBImage")) {
library(terra)
library(pliman)

# Create a sample mosaic
mosaic <- rast(nrow=10, ncol=10, xmin=-120, xmax=-60, ymin=30, ymax=60)

# Project the mosaic to the appropriate UTM zone
mosaic_lonlat2epsg(mosaic)
}
```

`mosaic_plot` *A wrapper around terra::plot()*

Description

Plot the values of a SpatRaster

Usage

```
mosaic_plot(
  mosaic,
  col = custom_palette(c("red", "yellow", "forestgreen"), n = 200),
  smooth = TRUE,
  ...
)
```

Arguments

`mosaic`	SpatRaster
`col`	character vector to specify the colors to use. Defaults to `custom_palette(c("red", "yellow", "forestgreen"))`.
`smooth`	logical. If TRUE (default) the cell values are smoothed (only if a continuous legend is used).
`...`	Further arguments passed on to `terra::plot()`.

Value

A `NULL` object

Examples

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
r <- mosaic_input(system.file("ex/elev.tif", package="terra"))
mosaic_plot(r)
}
```

mosaic_plot_rgb *A wrapper around terra::plotRGB()*

Description

Plot the RGB of a SpatRaster

Usage

```
mosaic_plot_rgb(mosaic, ...)
```

Arguments

mosaic	SpatRaster
...	Further arguments passed on to terra::plotRGB().

Value

A NULL object

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 = TRUE,
  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, g, b, re, nir	The red, green, blue, red-edge, and near-infrared bands of the image, respectively. By default, the function assumes a BGR as input (b = 1, g = 2, r = 3). If a multispectral image is provided up to seven bands can be used to compute built-in indexes. There are no limitation of band numbers if the index is computed using the band name.
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() && requireNamespace("EBImage")) {
library(pliman)
mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))
mosaic_prepare(mosaic)
}
```

`mosaic_project` *Project a Mosaic to a New Coordinate Reference System (CRS)*

Description

This function projects a given mosaic to a specified CRS.

Usage

```
mosaic_project(mosaic, y, ...)
```

Arguments

`mosaic`	A raster object representing the mosaic to be projected.
`y`	The target CRS to which the mosaic should be projected. This can be specified in various formats accepted by the `terra::project()` function.
`...`	Additional arguments passed to the `terra::project()` function.

Value

A raster object representing the projected mosaic.

Examples

```
if (interactive() && requireNamespace("EBImage")) {
library(terra)
library(pliman)

# Create a sample mosaic
mosaic <- rast(nrow=10, ncol=10, xmin=-120, xmax=-60, ymin=30, ymax=60)
mosaic
# Define target CRS (EPSG code for WGS 84 / UTM zone 33N)
target_crs <- "EPSG:32633"

# Project the mosaic
projected_mosaic <- mosaic_project(mosaic, "EPSG:32633")
projected_mosaic
}
```

mosaic_resample *A wrapper around terra::resample()*

Description

Transfers values between SpatRaster objects that do not align (have a different origin and/or resolution). See `terra::resample()` for more details

Usage

```
mosaic_resample(mosaic, y, ...)
```

Arguments

`mosaic`	SpatRaster to be resampled
`y`	SpatRaster with the geometry that x should be resampled to
`...`	Further arguments passed on to `terra::resample()`.

Value

SpatRaster

Examples

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
library(terra)
r <- rast(nrows=3, ncols=3, xmin=0, xmax=10, ymin=0, ymax=10)
values(r) <- 1:ncell(r)
s <- rast(nrows=25, ncols=30, xmin=1, xmax=11, ymin=-1, ymax=11)
x <- mosaic_resample(r, s, method="bilinear")
opar <- par(no.readonly =TRUE)
par(mfrow=c(1,2))
plot(r)
plot(x)
par(opar)
}
```

mosaic_rotate *Rotate a mosaic image by specified angles*

Description

This function rotates a mosaic image by 90, 180, or 270 degrees.

Usage

```
mosaic_rotate(mosaic, angle, direction = "clockwise")
```

Arguments

mosaic	A SpatRaster object representing the mosaic image.
angle	An integer specifying the rotation angle. Must be one of 90, 180, or 270.
direction	A string specifying the rotation direction. Must be either "clockwise" or "anti-clockwise".

Value

A SpatRaster object with the rotated mosaic image.

Examples

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
# Convert a mosaic raster to an Image object
mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))
r90 <- mosaic_rotate(mosaic, 90)
r180 <- mosaic_rotate(mosaic, 180)
r270 <- mosaic_rotate(mosaic, 270)
# Plot all rotations side by side
par(mfrow = c(2, 2))
mosaic_plot(mosaic, main = "Original")
mosaic_plot(r90, main = "90 Degrees")
mosaic_plot(r180, main = "180 Degrees")
mosaic_plot(r270, main = "270 Degrees")
par(mfrow = c(1, 1))

}
```

---

mosaic_segment *Segment a mosaic*

Description

Segment a SpatRaster using a computed image index. By default, values greater than threshold are kept in the mask.

Usage

```
mosaic_segment(
  mosaic,
  index = "R",
  r = 3,
  g = 2,
```

```
  b = 1,
  re = NA,
  nir = NA,
  swir = NA,
  tir = NA,
  threshold = "Otsu",
  invert = FALSE,
  return = c("mosaic", "mask")
)
```

Arguments

mosaic	A mosaic of class SpatRaster, generally imported with mosaic_input().
index	A character value (or a vector of characters) specifying the target mode for conversion to a binary image. Use pliman_indexes_rgb() and pliman_indexes_me() to see the available RGB and multispectral indexes, respectively. Users can also calculate their own index using R, G, B, RE, NIR, SWIR, and TIR bands (eg., index = "R+B/G") or using the names of the mosaic's layers (ex., "(band_1 + band_2) / 2").
r, g, b, re, nir, swir, tir	The red, green, blue, red-edge, near-infrared, shortwave Infrared, and thermal infrared bands of the image, respectively. By default, the function assumes a BGR as input (b = 1, g = 2, r = 3). If a multispectral image is provided up to seven bands can be used to compute built-in indexes. There are no limitation of band numbers if the index is computed using the band name.
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 provided, this value will be used as a threshold.
invert	Logical, indicating whether to invert the mask. Defaults to FALSE, i.e., pixels with intensity greater than the threshold values are selected.
return	The output of the function. Either 'mosaic' (the segmented mosaic), or 'mask' (the binary mask).

Value

The segmented mosaic (SpatRaster object)

Examples

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))
seg <-
mosaic_segment(mosaic,
               index = "elevation",
               threshold = 350)
mosaic_plot(seg)
}
```

`mosaic_segment_pick` *Segments a mosaic interactively*

Description

The function segments a mosaic using an interative process where the user picks samples from background (eg., soil) and foreground (eg., plants).

Usage

```
mosaic_segment_pick(
  mosaic,
  basemap = NULL,
  g = 2,
  r = 3,
  b = 1,
  max_pixels = 2e+06,
  downsample = NULL,
  quantiles = c(0, 1),
  return = c("mosaic", "mask")
)
```

Arguments

`mosaic`	A mosaic of class `SpatRaster`, generally imported with `mosaic_input()`.
`basemap`	An optional `mapview` object.
`r`, `g`, `b`	The layer for the Red, Green and Blue band, respectively. Defaults to `1`, `2`, and `3`.
`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`.
`quantiles`	the upper and lower quantiles used for color stretching.
`return`	The output of the function. Either 'mosaic' (the segmented mosaic), or 'mask' (the binary mask).

Value

An `SpatRaster` object with the segmented `mosaic` (if `return = 'mosaic'`) or a mask (if `return = 'mask'`).

Examples

```
if (interactive() && requireNamespace("EBImage")) {
 mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))
 seg <- mosaic_segment_pick(mosaic)
 mosaic_plot(seg)
}
```

`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, g, b, re, nir`	The red, green, blue, red-edge, and near-infrared bands of the image, respectively. By default, the function assumes a BGR as input (b = 1, g = 2, r = 3). If a multispectral image is provided up to seven bands can be used to compute built-in indexes. There are no limitation of band numbers if the index is computed using the band name.
`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

```
if (interactive() && requireNamespace("EBImage")) {
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, g, b	The red, green, blue bands.
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

```
if (interactive() && requireNamespace("EBImage")) {

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_vectorize` *Vectorize a* `SpatRaster` *mask to an* `sf` *object*

Description

Converts a raster mask into a vectorized `sf` object, with various options for morphological operations and filtering.

Usage

```
mosaic_vectorize(
  mask,
  aggregate = NULL,
  watershed = TRUE,
  tolerance = 1,
  extension = 1,
  opening = FALSE,
  closing = FALSE,
  filter = FALSE,
  erode = FALSE,
  dilate = FALSE,
  fill_hull = FALSE,
  lower_size = NULL,
  upper_size = NULL,
  topn_lower = NULL,
  topn_upper = NULL,
  smooth = FALSE
)
```

Arguments

`mask`	An optional mask (SpatRaster) to mask the mosaic.

`aggregate` The size as a fraction (percentage) of the input image size. Either a scalar (eg., 50), or a length-two numeric vector. In the last, different percentage reduction/expansion can be used for columns, and rows, respectively.

`watershed` If `TRUE` (default), performs watershed-based object detection. This will detect objects even when they are touching one another. 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. A higher value smooths out small objects.

`opening`, `closing`, `filter`, `erode`, `dilate`

Morphological operations (brush size)

- `dilate` puts the mask over every background pixel, and sets it to foreground if any of the pixels covered by the mask is from the foreground.
- `erode` puts the mask over every foreground pixel, and sets it to background if any of the pixels covered by the mask is from the background.
- `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects.
- `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects.
- `filter` performs median filtering in the binary image. 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.

`fill_hull` Fill holes in the binary image? Defaults to `FALSE`.

`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_size = 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.

`smooth` Smoothes the contours using a moving average filter. Default is `FALSE`.

Value

An `sf` object containing vectorized features from the raster mask, with added area measurements.

Examples

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
mask <- image_pliman("mask.tif")
shp <- mosaic_vectorize(mask, watershed = FALSE)
mosaic_plot(mask)
shapefile_plot(shp, add = TRUE, lwd = 3)
}
```

`mosaic_view` *Mosaic View*

Description

Mosaic View

Usage

```
mosaic_view(
  mosaic,
  r = 3,
  g = 2,
  b = 1,
  edit = FALSE,
  title = "",
  shapefile = NULL,
  attribute = NULL,
  viewer = c("mapview", "base"),
  show = c("rgb", "index"),
  index = "B",
  max_pixels = 1e+06,
  downsample = NULL,
  downsample_fun = "nearest",
  alpha = 1,
  quantiles = c(0, 1),
  color_regions = custom_palette(c("red", "yellow", "forestgreen")),
  axes = FALSE,
  ...
)
```

Arguments

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

edit	If TRUE enable editing options using mapedit::editMap().
title	A title for the generated map or plot (default: "").
shapefile	An optional shapefile of class sf to be plotted over the mosaic. It can be, for example, a plot-level result returned by mosaic_analyze().
attribute	The attribute name(s) or column number(s) in shapefile table of the column(s) to be rendered.
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.
downsample_fun	The resampling function. Defaults to nearest. See further details in mosaic_aggregate().
alpha	opacity of the fill color of the raster layer(s).
quantiles	the upper and lower quantiles used for color stretching.
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() && requireNamespace("EBImage")) {
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_bbox` *Compute Bounding Boxes from Contours*

Description

This function calculates the bounding boxes for a given list of contours.

Usage

```
object_bbox(contours)
```

Arguments

`contours`	A list of matrices, where each matrix contains two columns representing (x, y) coordinates of a contour.

Value

A list of bounding boxes, where each bounding box is represented as a list with `x_min`, `y_min`, `x_max`, and `y_max` values.

Examples

```
if(interactive()){
contours <- list(
  matrix(c(10, 20, 30, 40, 50, 60, 70, 80, 90, 100,
           110, 120, 130, 140, 150, 160, 170, 180, 190, 200),
         ncol = 2, byrow = FALSE)
)
bbox_list <- object_bbox(contours)
print(bbox_list)
}
```

`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

```
if (interactive() && requireNamespace("EBImage")) {
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,
  opening = 3,
  closing = FALSE,
  filter = FALSE,
  erode = FALSE,
  dilate = FALSE,
  threshold = "Otsu",
  extension = NULL,
  tolerance = NULL,
  object_size = "medium",
  edge = 20,
  remove_bg = FALSE,
  parallel = FALSE,
  workers = NULL,
  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.

`opening`, `closing`, `filter`, `erode`, `dilate`

Morphological operations (brush size)

- `dilate` puts the mask over every background pixel, and sets it to foreground if any of the pixels covered by the mask is from the foreground.
- `erode` puts the mask over every foreground pixel, and sets it to background if any of the pixels covered by the mask is from the background.
- `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects.

- `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects.
- `filter` performs median filtering in the binary image. 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.

`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).

`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.

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

Value

A `NULL` object.

Examples

```
if (interactive() && requireNamespace("EBImage")) {
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,
  workers = NULL,
  verbose = TRUE,
  viewer = pliman::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).

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.

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() && requireNamespace("EBImage")) {
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,
  opening = FALSE,
  closing = FALSE,
  filter = FALSE,
  erode = FALSE,
  dilate = 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).

`erode`, `dilate`, `opening`, `closing`, `filter`

Morphological operations (brush size)

- `dilate` puts the mask over every background pixel, and sets it to foreground if any of the pixels covered by the mask is from the foreground.
- `erode` puts the mask over every foreground pixel, and sets it to background if any of the pixels covered by the mask is from the background.
- `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects.
- `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects.
- `filter` performs median filtering in the binary image. 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.

Hierarchically, the operations are performed as opening > closing > filter. The value declared in each argument will define the brush size.

`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

```
if (interactive() && requireNamespace("EBImage")) {
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() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
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_scatter` *Plot object thumbnails at (x, y) coordinates derived from image features*

Description

Extracts connected objects from an image, computes their features, crops each object, converts the crop to a raster with alpha, and draws each thumbnail centered at its corresponding `(x, y)` feature location in a ggplot. Optionally overlays object IDs. Caching can be used to avoid recomputing object features on repeated calls.

Usage

```
object_scatter(
  img,
  x,
  y,
  scale = 0.1,
  xy_ratio = 1,
  xlab = x,
  ylab = y,
  erosion = 2,
  dilatation = FALSE,
  show_id = FALSE,
  color_id = "black",
  size_id = 3,
  cache = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

`img`	An image of class `EBImage::Image` (or compatible) from which objects will be segmented and measured.
`x`	Character scalar. Name of the feature column (returned by `get_measures(res)`) to use on the x-axis.

y	Character scalar. Name of the feature column to use on the y-axis.
scale	Numeric in (0, 1]. Relative thumbnail size as a fraction of the data range along each axis (larger values draw larger thumbnails).
xy_ratio	Numeric scalar. Factor applied to the vertical scaling (y-axis) of thumbnails. Use values other than 1 to stretch or compress thumbnails vertically.
xlab, ylab	Character scalars used as x- and y-axis labels. Defaults to x and y, respectively.
erosion, dilatation	Integer (non-negative). Size of the structuring element for morphological erosion/dilatation of the segmented objects.
show_id	Logical. If TRUE, overlays object IDs at their x, y locations.
color_id	Character. Color used for the ID labels when show_id = TRUE.
size_id	Numeric. Text size for the ID labels when show_id = TRUE.
cache	Logical. If TRUE (default), caches results of object extraction using a simple key based on image dimensions and parameters.
verbose	If TRUE (default), shows the progress of analysis.
...	Additional arguments forwarded to analyze_objects().

Value

A list with two elements:

- features — a data frame (or tibble) with object-level features returned by get_measures(res). Must contain columns named x and y.
- plot — a ggplot object. The thumbnail scatter plot.

Scaling behavior

Thumbnails are sized relative to the observed ranges in x and y. If the two axes differ substantially in range, perceived thumbnail aspect on the plotting device may vary. Use xy_ratio to adjust vertical scaling.

Examples

```
if(interactive()){
img <- image_pliman("potato_leaves.jpg")
plot(img)
res <- object_scatter(
 img = img,
 index = "B",
 x = "area",
 y = "solidity",
 watershed = FALSE,
 scale = 0.5
)
res$plot

# remove cached data
```

```
clear_pliman_cache()
}
```

`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,
  opening = 3,
  closing = FALSE,
  filter = FALSE,
  erode = FALSE,
  dilate = FALSE,
  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.

`opening`, `closing`, `filter`, `erode`, `dilate`

Morphological operations (brush size)

- `dilate` puts the mask over every background pixel, and sets it to foreground if any of the pixels covered by the mask is from the foreground.
- `erode` puts the mask over every foreground pixel, and sets it to background if any of the pixels covered by the mask is from the background.
- `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects.
- `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects.
- `filter` performs median filtering in the binary image. 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.

`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

```
if (interactive() && requireNamespace("EBImage")) {
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() && requireNamespace("EBImage")) {
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,
  pick_palettes = FALSE,
  background = NULL,
  foreground = NULL,
  index = "NB",
  color = "blue",
  plot = TRUE,
  ...
)
```

Arguments

img	An image object.
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.
foreground, background	A color palette for the foregrond and background, respectively (optional).
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

```
if (interactive() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
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,
  pattern = NULL,
  npal = 5,
  proportional = TRUE,
  colorspace = c("rgb", "hsb"),
  remove_bg = FALSE,
  index = "B",
  plot = TRUE,
  save_image = FALSE,
  prefix = "proc_",
  dir_original = NULL,
  dir_processed = NULL,
  return_pal = FALSE,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE
)
```

Arguments

`img`	An image object.
`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).
`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`.
`colorspace`	The color space to produce the clusters. Defaults to `rgb`. If `hsb`, the color space is first converted from RGB > HSB before k-means algorithm be applied.
`remove_bg`	Remove background from the color palette? Defaults to `FALSE`.
`index`	An image index used to remove the background, passed to `image_binary()`.
`plot`	Plot the generated palette? Defaults to `TRUE`.
`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".
return_pal	Return the color palette image? Defaults to FALSE.
parallel	If TRUE processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine.
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.
verbose	If TRUE (default) a summary is shown in the console.

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

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
img <- image_pliman("sev_leaf.jpg")
pal <- image_palette(img, npal = 5)
}
```

---

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

```
if (interactive() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
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.

`*.tif` format

The following `.tif` files are provided as sample data, representing a slice from a large orthomosaic with soybean plots in the vegetative stage. These files were kindly provided by Arthur Bernardeli.

- `ortho.tif`: An orthomosaic with soybean plots (5 rows and 3 columns).
- `dsm.tif`: A digital surface model (DSM) for the soybean plots.
- `dtm.tif`: A digital terrain model (DTM) for the area.
- `mask.tif`: A mask that represents the soybean plants.

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_indexes_ican_compute

List Computable Indexes Based on Available Bands

Description

This function reads index equations from a CSV file included in the `pliman` package, determines which bands are used in each index equation, and checks which indexes can be computed based on the provided available bands.

Usage

```
pliman_indexes_ican_compute(available)
```

Arguments

available	A character vector of available bands (e.g., c("R", "G")).

Value

A data frame of indexes that can be computed with the available bands.

Examples

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
available_bands <- c("R", "G")
computable_indexes <- pliman_indexes_ican_compute(available_bands)
print(computable_indexes)
}
```

---

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

```
if (interactive() && requireNamespace("EBImage")) {
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_bbox *Add Bounding Boxes to an Existing Plot*

Description

This function overlays bounding boxes onto an existing plot.

Usage

```
plot_bbox(bbox_list, col = "red")
```

Arguments

bbox_list	A list of bounding boxes, as returned by object_bbox().
col	The color for the bounding boxes. Defaults to "red".

Value

None (adds bounding boxes to an existing plot).

Examples

```
if(interactive()){
plot(NA,
    xlim = c(0, 200),
    ylim = c(0, 200),
    asp = 1)
contours <- list(
  matrix(c(10, 20, 30, 40, 50, 60, 70, 80, 90, 100,
           110, 120, 130, 140, 150, 160, 170, 180, 190, 200),
         ncol = 2, byrow = FALSE)
)
bbox_list <- object_bbox(contours)
plot_bbox(bbox_list)
}
```

`plot_id` *Generate plot IDs with different layouts*

Description

Based on a shapefile, number of columns and rows, generate plot IDs with different layouts.

Usage

```
plot_id(
  shapefile,
  nrow,
  ncol,
  layout = c("tblr", "tbrl", "btlr", "btrl", "lrtb", "lrbt", "rltb", "rlbt"),
  plot_prefix = "P",
  serpentine = FALSE
)
```

Arguments

`shapefile`	An object computed with `shapefile_build()`
`nrow`	The number of columns
`ncol`	The number of rows
`layout`	Character: one of • `'tblr'` for top/bottom left/right orientation • `'tbrl'` for top/bottom right/left orientation • `'btlr'` for bottom/top left/right orientation • `'btrl'` for bottom/top right/left orientation • `'lrtb'` for left/right top/bottom orientation • `'lrbt'` for left/right bottom/top orientation

- 'rltb' for right/left top/bottom orientation
- 'rlbt' for right/left bottom/top orientation

plot_prefix	The plot_id prefix. Defaults to 'P'.
serpentine	Create a serpentine-based layout? Defaults to FALSE.

Value

A vector of plot IDs with specified layout

---

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 = 1e+06,
  downsample = NULL,
  downsample_fun = NULL,
  color_regions = custom_palette(n = 100),
  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() using the argument return_mask = TRUE.
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() && requireNamespace("EBImage")) {
# Example usage:
library(pliman)
img <- image_pliman("sev_leaf.jpg")
plot_index(img, index = c("R", "G"))
}
```

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",
  r = 1,
  g = 2,
  b = 3,
  color = c("red", "yellow", "darkgreen"),
  viewer = c("mapview", "base"),
  max_pixels = 5e+05,
  downsample = NULL,
  downsample_fun = NULL,
  alpha = 0.7,
  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.
`r, g, b`	The layer for the Red, Green and Blue band, respectively. Defaults to `1`, `2`, and `3`.
`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() && requireNamespace("EBImage")) {
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 = 0.1,
                       buffer_y = 0.02,
                       nrow = 3,
                       ncol = 5,
                       plot = FALSE,
                       object_index = "DGCI")
plot_index_shp(res, attribute = "DGCI")
}
```

plot_line_segment *Plot Detected Line Segments*

Description

Plots the detected line segments from the output of `image_line_segment()`. Each segment is drawn as a red line on the existing plot.

Usage

```
plot_line_segment(x, col = "red", lwd = 1)
```

Arguments

`x`	A list returned by `image_line_segment()`, containing detected line segments.
`col`	The color of lines
`lwd`	The width of lines. Defaults to 1

Value

No return value. The function adds line segments to an existing plot.

Examples

```
library(pliman)
```

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

```
if (interactive() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
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() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
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)
}
```

---

sentinel_to_tif *Convert Sentinel data to GeoTIFF format*

Description

This function converts Sentinel satellite data files to GeoTIFF format.

Usage

```
sentinel_to_tif(layers = NULL, path = ".", destination, spat_res = 10)
```

Arguments

layers	(character) Vector of file paths to Sentinel data files. If NULL, the function searches for files in the specified path with names containing "B".
path	(character) Directory path where Sentinel data files are located. Default is the current directory.
destination	(character) File path for the output GeoTIFF file.
spat_res	(numeric) Spatial resolution of the output GeoTIFF file. Default is 10 meters.

Details

The function converts Sentinel satellite data files to GeoTIFF format using GDAL utilities. It builds a virtual raster file (VRT) from the input files and then translates it to GeoTIFF format. Compression is applied to the output GeoTIFF file using DEFLATE method.

`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.

shapefile_build *Build a shapefile from a mosaic raster*

Description

This function takes a mosaic raster to create a shapefile containing polygons for the specified regions. Users can drawn Areas of Interest (AOIs) that can be either a polygon with n sides, or a grid, defined by nrow, and ncol arguments.

Usage

```
shapefile_build(
  mosaic,
  basemap = NULL,
  controlpoints = NULL,
  r = 3,
  g = 2,
  b = 1,
  crop_to_shape_ext = TRUE,
  grid = TRUE,
  nrow = 1,
  ncol = 1,
  nsides = 200,
  plot_width = NULL,
  plot_height = NULL,
  layout = "lrtb",
  serpentine = TRUE,
  build_shapefile = TRUE,
  check_shapefile = FALSE,
  sf_to_polygon = FALSE,
```

```
  buffer_edge = 1,
  buffer_col = 0,
  buffer_row = 0,
  as_sf = TRUE,
  verbose = TRUE,
  max_pixels = 1e+06,
  downsample = NULL,
  quantiles = c(0, 1)
)
```

Arguments

mosaic	A SpatRaster object, typically imported using mosaic_input(). If not provided, a latitude/longitude basemap will be generated in the "EPSG:4326" coordinate reference system.
basemap	An optional mapview object.
controlpoints	An sf object created with mapedit::editMap(), containing the polygon that defines the region of interest to be analyzed.
r, g, b	The layer for the Red, Green and Blue band, respectively. Defaults to 1, 2, and 3.
crop_to_shape_ext	Crop the mosaic to the extension of shapefile? Defaults to TRUE. This allows for a faster index computation when the region of the built shapefile is much smaller than the entire mosaic extension.
grid	Logical, indicating whether to use a grid for segmentation (default: TRUE).
nrow	Number of rows for the grid (default: 1).
ncol	Number of columns for the grid (default: 1).
nsides	The number of sides if the geometry is generated with Draw Circle tool.
plot_width, plot_height	The width and height of the plot shape (in the mosaic unit). It is mutually exclusiv with buffer_col and buffer_row.
layout	Character: one of • 'tblr' for top/bottom left/right orientation • 'tbrl' for top/bottom right/left orientation • 'btlr' for bottom/top left/right orientation • 'btrl' for bottom/top right/left orientation • 'lrtb' for left/right top/bottom orientation • 'lrbt' for left/right bottom/top orientation • 'rltb' for right/left top/bottom orientation • 'rlbt' for right/left bottom/top orientation
serpentine	Create a serpentine-based layout? Defaults to FALSE.
build_shapefile	Logical, indicating whether to interactively draw ROIs if the shapefile is NULL (default: TRUE).

check_shapefile
: Logical, indicating whether to validate the shapefile with an interactive map view (default: TRUE). This enables live editing of the drawn shapefile by deleting or changing the drawn grids.

sf_to_polygon
: Convert sf geometry like POINTS and LINES to POLYGONS? Defaults to FALSE. Using TRUE allows using POINTS to extract values from a raster using exactextractr::exact_extract().

buffer_edge
: Width of the buffer around the shapefile (default: 5).

buffer_col, buffer_row
: Buffering factor for the columns and rows, respectively, of each individual plot'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 plot by 25% on each side.

as_sf
: Logical value indicating whether to convert the imported shapefile to an sf object (default is TRUE).

verbose
: Logical, indicating whether to display verbose output (default: TRUE).

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.

quantiles
: the upper and lower quantiles used for color stretching.

Details

Since multiple blocks can be created, the length of arguments grid, nrow, ncol, buffer_edge, buffer_col, and buffer_row can be either an scalar (the same argument applied to all the drawn blocks), or a vector with the same length as the number of drawn blocks. In the last, shapefiles in each block can be created with different dimensions.

Value

A list with the built shapefile. Each element is an sf object with the coordinates of the drawn polygons.

Examples

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))
shps <-
      shapefile_build(mosaic,
                      nrow = 6,
                      ncol = 3,
                      buffer_row = -0.05,
                      buffer_col = -0.25,
                      check_shapefile = FALSE,
                    build_shapefile = FALSE) ## Use TRUE to interactively build the plots
```

```
mosaic_plot(mosaic)
shapefile_plot(shps[[1]], add = TRUE)
}
```

`shapefile_edit` *Edit Features in a Shapefile*

Description

This function allows you to interactively edit features in a shapefile using the mapedit package.

Usage

```
shapefile_edit(
  shapefile,
  mosaic = NULL,
  basemap = NULL,
  r = 3,
  g = 2,
  b = 1,
  max_pixels = 3e+06
)
```

Arguments

`shapefile`	A shapefile (`sf` object) that can be created with `shapefile_input()`.
`mosaic`	Optionally, a mosaic (SpatRaster) to be displayed as a background.
`basemap`	An optional `mapview` object.
`r`	Red band index for RGB display (default is 3).
`g`	Green band index for RGB display (default is 2).
`b`	Blue band index for RGB display (default is 1).
`max_pixels`	Maximum number of pixels for down-sampling the mosaic (default is 3e6).

Value

A modified shapefile with user-edited features.

Examples

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
shp <- shapefile_input(system.file("ex/lux.shp", package="terra"))
edited <- shapefile_edit(shp)
}
```

shapefile_interpolate *Interpolate values at specific points based on coordinates and a target variable*

Description

This function interpolates values at specified points using x, y coordinates and a target variable from a shapefile. It supports "Kriging" and "Tps" interpolation methods.

Usage

```
shapefile_interpolate(
  shapefile,
  z,
  x = "x",
  y = "y",
  interpolation = c("Kriging", "Tps"),
  verbose = FALSE
)
```

Arguments

shapefile	An sf object containing the x, y, and target variable (z) columns. It is highly recommended to use `shapefile_measures()` to obtain this data.
z	A string specifying the name of the column in the shapefile that contains the target variable to be interpolated.
x	A string specifying the name of the column containing x-coordinates. Default is 'x'.
y	A string specifying the name of the column containing y-coordinates. Default is 'y'.
interpolation	A character vector specifying the interpolation method. Options are "Kriging" or "Tps".
verbose	Logical; if TRUE, progress messages will be displayed.

Value

A vector of interpolated values at the specified points.

shapefile_measures *Extract geometric measures from a shapefile object*

Description

shapefile_measures() calculates key geometric measures such as the number of points, area, perimeter, width, height, and centroid coordinates for a given shapefile (polygon) object.

Usage

```
shapefile_measures(shapefile, n = NULL)
```

Arguments

shapefile	An sf object representing the shapefile. It should contain polygonal geometries for which the measures will be calculated.
n	An integer specifying the number of polygons to process. If NULL, all polygons are considered.

Details

This function processes a single or multi-polygon sf object and computes geometric properties. It calculates distances between points, extracts the centroid coordinates, and computes the area and perimeter of the polygons. The width and height are derived from sequential distances between points.

Value

A modified sf object with added columns for:

- xcoord: The x-coordinate of the centroid.
- ycoord: The y-coordinate of the centroid.
- area: The area of the polygon (in square units).
- perimeter: The perimeter of the polygon (in linear units).
- width: The calculated width based on sequential distances between points. The result will only be accurate if the polygon is rectangular.
- height: The calculated height based on sequential distances between points. The result will only be accurate if the polygon is rectangular.

Examples

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)

path_shp <- paste0(image_pliman(), "/soy_shape.rds")
shp <- shapefile_input(path_shp)
shapefile_measures(shp)
```

```
}
```

`shapefile_operations` *Spatial Operations on Shapefiles*

Description

These functions perform various spatial operations on two shapefiles, including determining which geometries fall within, outside, touch, cross, overlap, or intersect another geometry. They also include functions for geometric operations such as intersection, difference, and union.

Usage

```
shapefile_within(shp1, shp2)

shapefile_outside(shp1, shp2)

shapefile_overlaps(shp1, shp2)

shapefile_touches(shp1, shp2)

shapefile_crosses(shp1, shp2)

shapefile_intersection(shp1, shp2)

shapefile_difference(shp1, shp2)

shapefile_union(shp1, shp2)
```

Arguments

`shp1`	An `sf` object representing the first shapefile.
`shp2`	An `sf` object representing the second shapefile.

Details

All functions ensure that the coordinate reference systems (CRS) of both shapefiles are the same before performing operations. If the CRSs are different, `shp2` will be transformed to match the CRS of `shp1`.

- `shapefile_within()`: Filters features in `shp1` that are fully within `shp2`.
- `shapefile_outside()`: Filters features in `shp1` that are outside or do not overlap `shp2`.
- `shapefile_overlaps()`: Filters features in `shp1` that overlap with `shp2`.
- `shapefile_touches()`: Filters features in `shp1` that touch the boundary of `shp2`.

- shapefile_crosses(): Filters features in shp1 that cross through shp2.
- shapefile_intersection(): Computes the geometric intersection of shp1 and shp2.
- shapefile_difference(): Computes the geometric difference of shp1 minus shp2.
- shapefile_union(): Computes the geometric union of shp1 and shp2.

Value

A filtered sf object or the result of the geometric operation.

Examples

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)

shp1 <- shapefile_input(paste0(image_pliman(), "/shp1.rds"))
shp2 <- shapefile_input(paste0(image_pliman(), "/shp2.rds"))
shapefile_view(shp1) + shapefile_view(shp1)

# Apply operations
shapefile_within(shp1, shp2)
shapefile_outside(shp1, shp2)
shapefile_overlaps(shp1, shp2)
shapefile_touches(shp1, shp2)
shapefile_crosses(shp1, shp2)
shapefile_intersection(shp1, shp2)
shapefile_difference(shp1, shp2)
shapefile_union(shp1, shp2)
}
```

---

shapefile_plot *A wrapper around terra::plot()*

Description

Plot the values of a SpatVector

Usage

```
shapefile_plot(shapefile, ...)
```

Arguments

shapefile	An SpatVector of sf object.
...	Further arguments passed on to terra::plot().

Value

A NULL object

Examples

```
if(interactive()){
library(pliman)
r <- shapefile_input(system.file("ex/lux.shp", package="terra"))
shapefile_plot(r)
}
```

---

shapefile_surface *Generate a spatial surface plot based on interpolated values*

Description

This function creates a surface plot from an interpolated spatial model, with options to customize plot appearance, grid resolution, and color palette.

Usage

```
shapefile_surface(
  model,
  curve = TRUE,
  nx = 300,
  ny = 300,
  xlab = "Longitude (UTM)",
  ylab = "Latitude (UTM)",
  col = custom_palette(c("darkred", "yellow", "forestgreen"), n = 100),
  ...
)
```

Arguments

model	An interpolated spatial object (e.g., from shapefile_interpolate()) containing the data for plotting.
curve	Logical; if TRUE, a contour plot is generated (type = "C"), otherwise an image plot (type = "I"). Default is TRUE.
nx	Integer; the number of grid cells in the x-direction. Default is 300.
ny	Integer; the number of grid cells in the y-direction. Default is 300.
xlab	Character; label for the x-axis. Default is "Longitude (UTM)".
ylab	Character; label for the y-axis. Default is "Latitude (UTM)".
col	A color palette function for the surface plot. Default is a custom palette from dark red to yellow to forest green.
...	Additional parameters to pass to fields::surface.

Value

A surface plot showing spatially interpolated data.

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

```
if (interactive() && requireNamespace("EBImage")) {
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. The function performs the conversion by applying a gamma correction to the input RGB values (raising them to the power of 2.2) and then transforming them using a specific transformation matrix. The result is clamped to the range 0-1 to ensure valid sRGB values.
- rgb_to_hsb() Transforms colors from RGB space (red/green/blue) to HSB space (hue/saturation/brightness). The HSB values are calculated as follows (see https://www.rapidtables.com/convert/color/rgb-to-hsv.html for more details).
  - Hue: The hue is determined based on the maximum value among R, G, and B, and it ranges from 0 to 360 degrees.
  - Saturation: Saturation is calculated as the difference between the maximum and minimum channel values, expressed as a percentage.
  - Brightness: Brightness is equal to the maximum channel value, expressed as a percentage.
- rgb_to_lab() Transforms colors from RGB space (red/green/blue) to CIE-LAB space, using the sRGB values. See grDevices::convertColor() for more details.

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>

References

See the detailed formulas here

Examples

```
if (interactive() && requireNamespace("EBImage")) {
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(), downsample = NULL, max_pixels = 1e+06)

distance(
  img,
  viewer = get_pliman_viewer(),
  downsample = NULL,
  max_pixels = 1e+06
)
```

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.
`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.
`max_pixels`	integer > 0. Maximum number of cells to use for the plot. If `max_pixels` < `npixels(img)`, regular sampling is used before plotting.

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
)
```

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

```
if (interactive() && requireNamespace("EBImage")) {
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_input(img, ...)

image_pliman(img, plot = FALSE)
```

Arguments

img	• For `image_import()`, a character vector of file names or URLs. • For `image_input()`, a character vector of file names or URLs or an array containing the pixel intensities of an image. • 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.
...	• For `image_import()` alternative arguments passed to the corresponding functions from the `jpeg`, `png`, and `tiff` packages. • For `image_input()` further arguments passed on to `EBImage::Image()`.
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

```
if (interactive() && requireNamespace("EBImage")) {
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_indexes *Utilities for image indexes*

Description

- `pliman_indexes()`: Get all the available indexes in pliman.
- `pliman_indexes_rgb()`: Get all the RGB-based available indexes in pliman.
- `pliman_indexes_me()`: Get all the multispectral available indexes in pliman.
- `pliman_indexes_hs()`: Get all the hyperspectral available indexes in pliman.
- `pliman_indexes_eq()`: Get the equations of the available indexes.

Usage

```
pliman_indexes()

pliman_indexes_eq()

pliman_indexes_rgb()

pliman_indexes_me()

pliman_indexes_hs()
```

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

```
if (interactive() && requireNamespace("EBImage")) {
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,
  opening = FALSE,
  closing = FALSE,
  filter = FALSE,
  fill_hull = FALSE,
  threshold = "Otsu",
  edge = 2,
  extension = NULL,
  tolerance = NULL,
  object_size = "medium",
  parallel = FALSE,
  workers = NULL,
  plot = TRUE,
  verbose = TRUE
)
```

```
object_contour(
  img,
  pattern = NULL,
  dir_original = NULL,
  center = FALSE,
  index = "NB",
  invert = FALSE,
  opening = FALSE,
  closing = FALSE,
  filter = FALSE,
  fill_hull = FALSE,
  smooth = 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,
  verbose = TRUE,
  ...
)

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

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`.

`opening`, `closing`, `filter`

Morphological operations (brush size)

- `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects.
- `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects.
- `filter` performs median filtering in the binary image. 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.

Hierarchically, the operations are performed as opening > closing > filter. The value declared in each argument will define the brush size.

`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`.

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

`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.

`smooth` whether the object contours should be smoothed with `poly_smooth()`. Defaults to `FALSE`. To smooth use a numeric value indicating the number of interactions used to smooth the contours.

`...`

- 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

```
if (interactive() && requireNamespace("EBImage")) {
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(),
  external_device = FALSE,
  size = 0.8,
  plot = TRUE,
  verbose = TRUE
)

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

pick_rgb(
  img,
  n = Inf,
  col = "red",
  viewer = get_pliman_viewer(),
  external_device = FALSE,
  size = 0.8,
  plot = TRUE,
  verbose = TRUE
)

pick_palette(
  img,
  n = Inf,
  r = 2,
  shape = "box",
  viewer = get_pliman_viewer(),
  external_device = FALSE,
  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.
external_device	Logical. If TRUE (default), opens an external graphics window when running inside RStudio to ensure accurate point selection using locator(). Ignored when not in RStudio or when using viewer = "mapview".
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 vieweris 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() && requireNamespace("EBImage")) {
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 (centroid) of a polygon given by the vertices in the vectors x and y using the following formulas:

$$C_x = \frac{1}{6A}\sum_{i=1}^{n}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

$$C_y = \frac{1}{6A}\sum_{i=1}^{n}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

where `C_x` and `C_y` are the coordinates of the center of mass, `A` is the area of the polygon computed by the Shoelace formula, `x` and `y` are the coordinates that form the corners of the polygon, and `n` is the number of coordinates.

- `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_centdist_mass()` Computes the Euclidean distance between every point on the perimeter and the center of mass 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_centdist_mass(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

```
if (interactive() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
iris2 <- iris |> rownames_to_column()
head(iris2)
iris2$rowname <- paste0("r", iris2$rowname)
iris2 |> column_to_rownames("rowname") |> head()
}
```

---

utils_shapefile *Import/export shapefiles.*

Description

- shapefile_input() creates or imports a shapefile and optionally converts it to an sf object. It can also cast POLYGON or MULTIPOLYGON geometries to MULTILINESTRING if required.
- shapefile_export() exports an object (sf or SpatVector) to a file.
- shapefile_view() is a simple wrapper around mapview() to plot a shapefile.

Usage

```
shapefile_input(
  shapefile,
  info = TRUE,
  as_sf = TRUE,
```

```
  multilinestring = FALSE,
  ...
)

shapefile_export(shapefile, filename, ...)

shapefile_view(
  shapefile,
  attribute = NULL,
  type = c("shape", "centroid"),
  color_regions = custom_palette(c("red", "yellow", "forestgreen")),
  ...
)
```

Arguments

shapefile	For `shapefile_input()`, character (filename), or an object that can be coerced to a SpatVector, such as an `sf` (simple features) object. See `terra::vect()` for more details. For `shapefile_export()`, a `SpatVector` or an `sf` object to be exported as a shapefile.
info	Logical value indicating whether to print information about the imported shapefile (default is `TRUE`).
as_sf	Logical value indicating whether to convert the imported shapefile to an `sf` object (default is `TRUE`).
multilinestring	Logical value indicating whether to cast polygon geometries to `MULTILINESTRING` geometries (default is `FALSE`).
...	Additional arguments to be passed to `terra::vect()` (`shapefile_input()`), `terra::writeVector()` (`shapefile_export()`) or `mapview::mapview()` (`shapefile_view()`).
filename	The path to the output shapefile.
attribute	The attribute to be shown in the color key. It must be a variable present in `shapefile`.
type	A character string specifying whether to visualize the shapefile as `"shape"` or as `"centroid"`. Partial matching is allowed. If set to `"centroid"`, the function will convert the shapefile's geometry to centroids before displaying. Defaults to `"shape"`.
color_regions	The color palette to represent `attribute`.

Value

- `shapefile_input()` returns an object of class `sf` (default) representing the imported shapefile.
- `shapefile_export()` returns a `NULL` object.
- `shapefile_view()` returns an object of class `mapview`.

Examples

```
if(interactive()){
library(pliman)
shp <- system.file("ex/lux.shp", package="terra")
shp_file <- shapefile_input(shp, as_sf = FALSE)
shapefile_view(shp_file)
}
```

`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,
  opening = 5,
  closing = FALSE,
  filter = FALSE,
  invert = FALSE,
  threshold = "Otsu",
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_crop(
  img,
  width = NULL,
  height = NULL,
  viewer = get_pliman_viewer(),
  downsample = NULL,
  max_pixels = 1e+06,
  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.

`opening`, `closing`, `filter`

Morphological operations (brush size)

- `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects.
- `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects.
- `filter` performs median filtering in the binary image. 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.

Hierarchically, the operations are performed as opening > closing > filter. The value declared in each argument will define the brush size.

`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).

`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.

`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.

`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.

`max_pixels` integer > 0. Maximum number of cells to use for the plot. If `max_pixels` < `npixels(img)`, regular sampling is used before plotting.

`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

```
if (interactive() && requireNamespace("EBImage")) {
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

```
if (interactive() && requireNamespace("EBImage")) {
get_wd_here()
set_wd_here()
open_wd_here()
}
```

---

uuid *Generate Version 7 UUIDs or Random UUIDs*

Description

This function generates one or more UUIDs (Universally Unique Identifiers). By default, it generates Version 7 UUIDs, which are time-ordered and suitable for use cases requiring efficient indexing and sorting by creation time. Alternatively, random Version 4 UUIDs can be generated by setting usetime = FALSE.

Usage

```
uuid(n = 1, uppercase = FALSE, usetime = FALSE)
```

Arguments

n	Integer. Number of UUIDs to generate. Default is 1.
uppercase	Logical. If TRUE, the generated UUIDs are returned in uppercase letters. Default is FALSE.
usetime	Logical. If TRUE, generates Version 7 UUIDs using the current timestamp. If FALSE, generates random Version 4 UUIDs. Default is FALSE.

Details

- **Version 7 UUIDs**: These are time-ordered UUIDs based on the current timestamp in milliseconds since the Unix epoch (1970-01-01 00:00:00 UTC). They are ideal for scenarios requiring chronological sorting or indexing.
- **Version 4 UUIDs**: These are randomly generated UUIDs that do not depend on time, ensuring uniqueness through random hexadecimal values.

Value

A character vector of UUIDs of length n.

Examples

```
library(pliman)
# Generate a single UUID
uuid()

# Generate 5 UUIDs in uppercase
uuid(n = 3, uppercase = TRUE)

# Generate two random UUIDs
uuid(n = 2, usetime = FALSE)
```

---

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

```
if (interactive() && requireNamespace("EBImage")) {
library(pliman)
img <- image_pliman("soybean_touch.jpg")
binary <- image_binary(img, "B")[[1]]
wts <- watershed2(binary)
range(wts)
}
```

Index

∗ **data**
    contours, 33
∗ **images**
    pliman_images, 163
%>% *(*pipe*)*, 161

analyze_objects, 5, *27*
analyze_objects(), *5*, *13*, *16*, *18*, *23*, *27*, *75*, *90*, *153*, *156*, *168*, *172, 173*, *190*, *200*, *204*, *217*
analyze_objects_iter *(*analyze_objects*)*, 5
analyze_objects_iter(), *5*, *13, 14*, *23*
analyze_objects_minimal, 18
analyze_objects_shp, 24, *150, 151*
analyze_objects_shp(), *62*, *170*
apply_fun_to_imgs, 28
as_image, 29

base::jitter(), *212*, *214*
base::lapply(), *213*
base::sapply(), *213*

calibrate, 30
calibrate(), *77*
ccc, 31
clear_pliman_cache, 32
cm_to_dpi *(*utils_dpi*)*, 192
cm_to_dpi(), *192*, *194*
cm_to_pixels *(*utils_dpi*)*, 192
cm_to_pixels(), *193, 194*
column_to_rownames *(*utils_rows_cols*)*, 217
contours, 33
conv_hull *(*utils_polygon*)*, 210
conv_hull_unified *(*utils_polygon*)*, 210
custom_palette, 34
custom_palette(), *76*, *139*, *169*

dist_transform, 34
distance *(*utils_dpi*)*, 192
distance(), *193*
dpi *(*utils_dpi*)*, 192
dpi(), *193, 194*
dpi_to_cm *(*utils_dpi*)*, 192
dpi_to_cm(), *192*, *194*
draw_circle *(*utils_shapes*)*, 220
draw_n_tagon *(*utils_shapes*)*, 220
draw_rectangle *(*utils_shapes*)*, 220
draw_square *(*utils_shapes*)*, 220
draw_trian_equi *(*utils_shapes*)*, 220
draw_trian_rect *(*utils_shapes*)*, 220

EBImage::bwlabel, *147*
EBImage::bwlabel(), *151*
EBImage::clahe(), *223*
EBImage::closing(), *223*
EBImage::dilate(), *223*
EBImage::erode(), *223*
EBImage::gblur(), *223*
EBImage::Image(), *29, 30*, *197*
EBImage::makeBrush(), *70*, *83, 84*, *229*
EBImage::medianFilter(), *223*
EBImage::opening(), *223*
EBImage::resize(), *223*
EBImage::watershed, *147*
EBImage::watershed(), *88*, *151*, *232*
efourier, 35
efourier(), *11*, *14*, *16*, *27*, *33*, *37–41*
efourier_coefs, 37
efourier_error, 37
efourier_error(), *16*
efourier_inv, 39
efourier_norm, 40
efourier_norm(), *16*, *37*, *40*
efourier_power, 41
efourier_power(), *16*
efourier_shape, 42
ellipse, 44
entropy, 45

exactextractr::exact_extract(), *117*, *119*

file_dir *(*utils_file*)*, 195
file_extension *(*utils_file*)*, 195
file_name *(*utils_file*)*, 195

get_biplot *(*utils_pca*)*, 205
get_biplot(), *190*, *206*
get_measures *(*utils_measures*)*, 199
get_measures(), *14*
get_pliman_viewer, 45
get_pliman_viewer(), *8*, *25*, *30*, *47*, *63*, *69*, *71*, *77*, *90*, *96*, *117*, *128*, *139*, *146*, *157*, *169*, *170*, *193*, *209*, *229*
get_uuid, 46
get_wd_here *(*utils_wd*)*, 230
get_wd_here(), *230*, *231*
getwd(), *198*
ggplot_color, 46
graphics::barplot(), *206*
graphics::lines(), *216*
graphics::plot.window(), *169*
graphics::points(), *216*
graphics::polygon(), *217*
graphics::text(), *201*
grDevices::colors(), *56*, *177*
grDevices::convertColor(), *191*

image_align, 47
image_align(), *47*, *177*
image_alpha, 48
image_augment, 49
image_augment(), *143*
image_autocrop *(*utils_transform*)*, 223
image_autocrop(), *228*
image_binary, 50
image_binary(), *13*, *73*, *156*, *158*, *160*, *203*, *229*
image_blur *(*utils_transform*)*, 223
image_canny_edge, 53
image_closing *(*utils_transform*)*, 223
image_combine, 54
image_combine(), *156*, *198*
image_contour_line, 55
image_contrast *(*utils_transform*)*, 223
image_create, 55
image_crop *(*utils_transform*)*, 223
image_crop(), *177*
image_dilate *(*utils_transform*)*, 223
image_dimension *(*utils_transform*)*, 223
image_erode *(*utils_transform*)*, 223
image_expand, 56
image_expand(), *72*
image_export *(*utils_image*)*, 197
image_filter *(*utils_transform*)*, 223
image_horizontal *(*utils_transform*)*, 223
image_hreflect *(*utils_transform*)*, 223
image_import *(*utils_image*)*, 197
image_index, 57
image_index(), *11*, *22*, *26*, *51*, *63*, *65*, *88*, *92*, *96*, *98*, *104*, *108*, *121*, *143*, *147*, *154*, *157*, *158*, *168*, *169*, *228*
image_input *(*utils_image*)*, 197
image_label, 60
image_line_segment, 61
image_line_segment(), *172*
image_opening *(*utils_transform*)*, 223
image_palette *(*palettes*)*, 159
image_pliman *(*utils_image*)*, 197
image_prepare, 62
image_prepare(), *25*, *27*, *96*
image_resize *(*utils_transform*)*, 223
image_rotate *(*utils_transform*)*, 223
image_segment, 63
image_segment_iter *(*image_segment*)*, 63
image_segment_kmeans, 67
image_segment_manual, 68
image_segment_mask, 69
image_shp, 71
image_shp(), *26*, *95*, *156*, *157*, *165*, *166*
image_skeleton *(*utils_transform*)*, 223
image_skeleton(), *141*
image_square, 72
image_square(), *143*, *146*
image_thinning *(*utils_transform*)*, 223
image_thinning_guo_hall, 73
image_to_mat, 74
image_trim *(*utils_transform*)*, 223
image_trim(), *228*, *229*
image_vertical *(*utils_transform*)*, 223
image_view, 75
image_vreflect *(*utils_transform*)*, 223

landmarks, 76
landmarks(), *35*, *78–80*
landmarks_add, 78
landmarks_angle, 79

landmarks_dist, 80
landmarks_regradi, 81
landmarks_regradi(), *35*, *78*
leading_zeros, 82
line_on_halfplot, 83
locator(), *209*

make_brush, 83
make_brush(), *69*
make_mask, 84
make_mask(), *69*
manipulate_files *(*utils_file*)*, 195
mapedit::editMap(), *76*, *139*, *182*
mapview::mapview(), *219*
max_list *(*utils_stats*)*, 222
mean_list *(*utils_stats*)*, 222
measure_disease, 85, *96*
measure_disease(), *91*, *94, 95*, *97*, *178*
measure_disease_byl, 91
measure_disease_byl(), *89*, *178*
measure_disease_iter *(*measure_disease*)*, 85
measure_disease_shp, 95
measure_injury, 97
min_list *(*utils_stats*)*, 222
mosaic_aggregate, 99
mosaic_aggregate(), *139*
mosaic_analyze, 100
mosaic_analyze(), *108*, *139*
mosaic_analyze_iter, 106
mosaic_chm, 108
mosaic_chm(), *110*
mosaic_chm_extract, 110
mosaic_chm_mask, 111
mosaic_chm_mask(), *103*
mosaic_classify, 112
mosaic_clip, 113
mosaic_crop, 114
mosaic_draw, 116
mosaic_epsg, 118
mosaic_export *(*mosaic_input*)*, 123
mosaic_extract, 119
mosaic_hist, 119
mosaic_index, 120
mosaic_index(), *102*
mosaic_index2, 122
mosaic_input, 123
mosaic_input(), *102*, *107*, *116*, *120*, *122*, *128*, *132–135*, *138*, *182*
mosaic_interpolate, 124
mosaic_lonlat2epsg, 125
mosaic_plot, 126
mosaic_plot_rgb, 127
mosaic_prepare, 127
mosaic_project, 129
mosaic_resample, 130
mosaic_rotate, 130
mosaic_segment, 131
mosaic_segment_pick, 133
mosaic_segment_pick(), *103*
mosaic_to_pliman, 134
mosaic_to_rgb, 135
mosaic_vectorize, 136
mosaic_view, 138
mosaic_view(), *102*, *107*, *114, 115*

npixels *(*utils_dpi*)*, 192
npixels(), *193*

object_bbox, 140
object_contour, *176*
object_contour *(*utils_objects*)*, 202
object_contour(), *35*, *44*, *78*, *81*, *173*
object_coord *(*utils_objects*)*, 202
object_coord(), *204*
object_edge, 141
object_edge(), *9*, *14*, *27*
object_export, 142
object_export_shp, 145
object_id *(*utils_objects*)*, 202
object_id(), *203*
object_isolate *(*utils_objects*)*, 202
object_label, 147
object_map, 149
object_mark, 150
object_rgb, 151
object_scatter, 152
object_scatter(), *32*
object_split, 154
object_split(), *91*, *142*
object_split_shp, 156
object_split_shp(), *27*, *145*
object_to_color, 157
open_wd *(*utils_wd*)*, 230
open_wd(), *230*
open_wd_here *(*utils_wd*)*, 230
open_wd_here(), *230, 231*
otsu, 159

palettes, 159
pca (utils_pca), 205
pca(), *190, 191*
pick_coords (utils_pick), 207
pick_count (utils_pick), 207
pick_palette (utils_pick), 207
pick_palette(), *8*, *90*, *158*
pick_rgb (utils_pick), 207
pipe, 161
pixel_index, 162
pixels_to_cm (utils_dpi), 192
pixels_to_cm(), *192*, *194*
pliman_images, 163
pliman_indexes (utils_indexes), 199
pliman_indexes(), *8*, *20*, *51*, *58*, *65*, *104*, *108*, *147*, *158*
pliman_indexes_eq (utils_indexes), 199
pliman_indexes_eq(), *11*, *27*
pliman_indexes_hs (utils_indexes), 199
pliman_indexes_ican_compute, 164
pliman_indexes_me (utils_indexes), 199
pliman_indexes_me(), *120*, *122*, *132*
pliman_indexes_rgb (utils_indexes), 199
pliman_indexes_rgb(), *120*, *122*, *132*
pliman_viewer, 165
plot.anal_obj (analyze_objects), 5
plot.anal_obj(), *5*
plot.anal_obj_ls (analyze_objects), 5
plot.anal_obj_ls_minimal (analyze_objects_minimal), 18
plot.anal_obj_minimal (analyze_objects_minimal), 18
plot.image_index (image_index), 57
plot.image_shp, 165
plot.pca (utils_pca), 205
plot_bbox, 166
plot_contour (utils_polygon_plot), 216
plot_ellipse (utils_polygon_plot), 216
plot_id, 167
plot_index, 168
plot_index(), *59*
plot_index_shp, 170
plot_line_segment, 172
plot_lw, 172
plot_lw(), *12*
plot_mass (utils_polygon_plot), 216
plot_measures (utils_measures), 199
plot_polygon (utils_polygon_plot), 216
png(), *179*
poly_align (utils_polygon), 210
poly_align(), *15*, *35*
poly_angles (utils_polygon), 210
poly_apex_base_angle, 173
poly_apex_base_angle(), *15*
poly_area (utils_polygon), 210
poly_caliper (utils_polygon), 210
poly_caliper(), *15*
poly_centdist (utils_polygon), 210
poly_centdist_mass (utils_polygon), 210
poly_center (utils_polygon), 210
poly_center(), *35*
poly_check (utils_polygon), 210
poly_circularity (utils_polygon), 210
poly_circularity_haralick (utils_polygon), 210
poly_circularity_norm (utils_polygon), 210
poly_close (utils_polygon), 210
poly_convexity (utils_polygon), 210
poly_distpts (utils_polygon), 210
poly_eccentricity (utils_polygon), 210
poly_elongation (utils_polygon), 210
poly_flip_x (utils_polygon), 210
poly_flip_y (utils_polygon), 210
poly_is_closed (utils_polygon), 210
poly_jitter (utils_polygon), 210
poly_limits (utils_polygon), 210
poly_lw (utils_polygon), 210
poly_mass (utils_polygon), 210
poly_measures (utils_polygon), 210
poly_pcv, 174
poly_perimeter (utils_polygon), 210
poly_rotate (utils_polygon), 210
poly_sample (utils_polygon), 210
poly_sample_prop (utils_polygon), 210
poly_slide (utils_polygon), 210
poly_smooth (utils_polygon), 210
poly_smooth(), *10*, *35*, *78*, *174*, *204*
poly_solidity (utils_polygon), 210
poly_spline (utils_polygon), 210
poly_unclose (utils_polygon), 210
poly_width_at, 175
prepare_to_shp, 177

random_color, 177
remove_rownames (utils_rows_cols), 217
rgb_to_hsb (utils_colorspace), 191

rgb_to_lab *(utils_colorspace)*, 191
rgb_to_srgb *(utils_colorspace)*, 191
round_cols *(utils_rows_cols)*, 217
rownames_to_column *(utils_rows_cols)*, 217

sad, 178
sad(), *179*
sd_list *(utils_stats)*, 222
sentinel_to_tif, 179
separate_col, 180
set_pliman_viewer, 181
set_pliman_viewer(), *25*, *30*, *47*, *63*, *69*, *71*, *77*, *90*, *96*, *117*, *128*, *139*, *146*, *157*, *169, 170*, *193*, *209*, *229*
set_wd_here *(utils_wd)*, 230
set_wd_here(), *230*
sf::sf, *113*
sf::st_simplify(), *103*
shapefile_build, 181
shapefile_build(), *125*, *167*
shapefile_crosses *(shapefile_operations)*, 187
shapefile_difference *(shapefile_operations)*, 187
shapefile_edit, 184
shapefile_export *(utils_shapefile)*, 218
shapefile_input *(utils_shapefile)*, 218
shapefile_input(), *115*, *125*, *184*
shapefile_interpolate, 185
shapefile_intersection *(shapefile_operations)*, 187
shapefile_measures, 186
shapefile_operations, 187
shapefile_outside *(shapefile_operations)*, 187
shapefile_overlaps *(shapefile_operations)*, 187
shapefile_plot, 188
shapefile_surface, 189
shapefile_touches *(shapefile_operations)*, 187
shapefile_union *(shapefile_operations)*, 187
shapefile_view *(utils_shapefile)*, 218
shapefile_within *(shapefile_operations)*, 187
stats::prcomp(), *205*
summary_index, 190

terra::datatype(), *124*
terra::hist(), *119*
terra::plot(), *126*, *139*, *188*
terra::plotRGB(), *127*
terra::project(), *129*
terra::rast(), *123, 124*
terra::resample(), *130*
terra::SpatRaster, *113*
terra::SpatVector, *113*
terra::vect(), *219*
terra::writeRaster(), *123, 124*
terra::writeVector(), *219*

utils_colorspace, 191
utils_dpi, 192
utils_file, 195
utils_image, 197
utils_indexes, 199
utils_measures, 199
utils_objects, 202
utils_pca, 205
utils_pick, 207
utils_polygon, 210
utils_polygon_plot, 216
utils_rows_cols, 217
utils_shapefile, 218
utils_shapes, 220
utils_stats, 222
utils_transform, 223
utils_wd, 230
uuid, 231

watershed2, 232