Package ‘rcaiman’

December 23, 2021

Type Package

Title An R Package for CAnopy IMage ANalysis

Version 0.0.1

Date 2022-01-03

Description Its main strength is to classify hemispherical photographs of the plant canopy with algorithms specially developed for such a task and well documented in Díaz and Lencinas (2015) ¡doi:10.1109/lgrs.2015.2425931¿ and Díaz and Lencinas (2018) ¡doi:10.1139/cjfr-2018-0006¿. It supports non-circular hemispherical photography.

License GPL-3

BugReports https://github.com/GastonMauroDiaz/rcaiman/issues

Encoding UTF-8

RoxygenNote 7.1.2

Depends raster

Imports methods, testthat, rgdal, pracma, stats, magrittr, utils, Rdpack, spatial, rgeos, sp, colorspace

Suggests autothresholdr, conicfit

RdMacros Rdpack

NeedsCompilation no

Author Gastón Mauro Díaz [aut, cre] (¡https://orcid.org/0000-0002-0362-8616¿)

Maintainer Gastón Mauro Díaz <gastonmaurodiaz@gmail.com>

R topics documented:

`apply_thr` *Apply threshold*

Description

Global or local thresholding of images.

Usage

```
apply_thr(r, thr)
```

Arguments

`r`	RasterLayer
`thr`	Numeric vector of length one or RasterLayer. Threshold.

Details

It is a wrapper function around the operator `>` from the 'raster' package. If a single threshold value is provided as `thr` argument, it is applied to every pixel of the raster object `r`. If instead a RasterLayer is provided, then a particular threshold is applied to each particular pixel.

Value

RasterLayer with values `0` and `1`.

See Also

regional_thresholding.

Other Tools functions: extract_feature(), gbc(), masking(), normalize(), read_bin(), read_caim(), regional_thresholding(), write_bin(), write_caim()

Examples

```
r <- read_caim()
apply_thr(r$Blue, 120)
## Not run:
# This function is useful in combination with the ‘autothresholdr’
package. For examples:
require(autothresholdr)
thr <- auto_thresh(r$Blue[], "IsoData")[1]
bin <- apply_thr(r$Blue, thr)
plot(bin)

## End(Not run)
```

---

azimuth_image *Azimuth image*

Description

Build a single layer image with azimuth angles as pixel values.

Usage

```
azimuth_image(z)
```

Arguments

z	RasterLayer built with zenith_image.

Value

RasterLayer.

See Also

Other Lens functions: calc_diameter(), calc_zenith_raster_coordinates(), calibrate_lens(), expand_noncircular(), lens(), reproject_to_equidistant(), test_lens_coef(), zenith_image()

Examples

```
z <- zenith_image(1490, lens("Nikon_FCE9"))
azimuth_image(z)
plot(z)
```

calc_diameter *Calculate diameter*

Description

Calculate the diameter in pixels of a 180º fisheye image.

Usage

```
calc_diameter(lens_coef, radius_px, angle)
```

Arguments

lens_coef	Numeric vector. Polynomial coefficients of the lens projection function.
radius_px	Numeric vector. Distance in pixels from the zenith.
angle	Numeric vector. Zenith angle in degrees.

Details

This function is useful to handle devices with field of view different than 180 degrees. Given a lens projection function and data points consisting of radii (pixels) and their correspondent zenith angle (θ), it returns the radius of the horizon (i.e., the radius for θ equal to 90 degrees).

It is particularly useful when working with non-circular hemispherical photography. It will help to find the diameter that a circular image would have if the equipment would depict the whole hemisphere.

The required data (radius-angle data) can be obtained following the instructions given in the user manual of Hemisfer software. They suggests using a corner to set up markers on the walls from 0º to 90º θ. A fast way of obtaining a photograph showing several targets with known θ is to find a wall, draw a triangle of $5 \times 4 \times 3$ meters on the floor, with the 4-meter side over the wall. Locate the camera over the vertice that is 3 meters away from the wall. Place it at a given height above the floor, 1.3 meters for instance. Point the camera to the wall. Make a mark on the wall at 1.3 meters over the vertice that is in front of the camera. Next, make four more marks with one meter of distance between them and on a horizontal line. This will create marks for 0º, 18º, 34º, 45º, and 54º θ. Don't forget to align the zenith coordinates with the 0º θ mark and check if the optical axis is leveled.

For obtaining the lens projection of a new lens, refer to `calibrate_lens`.

See Also

Other Lens functions: `azimuth_image()`, `calc_zenith_raster_coordinates()`, `calibrate_lens()`, `expand_noncircular()`, `lens()`, `reproject_to_equidistant()`, `test_lens_coef()`, `zenith_image()`

Examples

```
# Nikon D50 and Fisheye Nikkor 10.5 mm lens
calc_diameter(lens("Nikkor_10.5_mm"), 1202, 53)
```

`calc_zenith_raster_coordinates`
Calculate zenith raster coordinates

Description

Calculate zenith raster coordinates from points digitized with the open-source software package 'ImageJ'. The zenith is the point on the image that represents the zenith when upward-looking photographs are taken with the optical axis parallel to the vertical line.

Usage

```
calc_zenith_raster_coordinates(path_to_csv)
```

Arguments

`path_to_csv`	Character vector of length one. Path to a CSV file created with the point selection tool of 'ImageJ' software.

Details

It is important to note the difference between the raster coordinates and the Cartesian coordinates. In the latter, the vertical axis value decreases when you go down, but the opposite is true for the raster coordinates, which works like a spreadsheet.

The technique described under the headline 'Optical center characterization' of the user manual of the software Can-Eye can be used to acquire the data for determining the zenith coordinates. This technique was used by Pekin and Macfarlane (2009), among others. Briefly, it consists in drilling a small hole in the cap of the fisheye lens (it must be away from the center of the cap), and taking about ten photographs without removing the cap. The cap must be rotated about 30º before taking each photograph.

The point selection tool of 'ImageJ' software should be used to manually digitize the white dots and create a CSV file to feed this function.

Another method –only valid for circular hemispherical photographs– is taking a very bright picture (for example, a picture of a room with walls painted in light colors) with the lens completely free (do not use any mount). Then, digitize points over the perimeter of the circle. This was the method used for producing the example (see below). It is worth noting that the perimeter of the circle depicted in a circular hemispherical photograph is not necessarily the horizon.

References

Pekin B, Macfarlane C (2009). "Measurement of crown cover and leaf area index using digital cover photography and its application to remote sensing." *Remote Sensing*, **1**(4), 1298–1320. doi: 10.3390/rs1041298.

See Also

Other Lens functions: `azimuth_image()`, `calc_diameter()`, `calibrate_lens()`, `expand_noncircular()`, `lens()`, `reproject_to_equidistant()`, `test_lens_coef()`, `zenith_image()`

Examples

```
path <- system.file("external/points_over_perimeter.csv",
                    package = "rcaiman")
calc_zenith_raster_coordinates(path)
```

`calibrate_lens` *Calibrate lens*

Description

Calibrate a fisheye lens. This type of lens has wide field of view and a consistent azimuthal distortion, so a precise mathematical relation can be fit between the distance to the zenith on the image space and the zenith angle on the hemispherical space

Usage

```
calibrate_lens(path_to_csv, degree = 3)
```

Arguments

`path_to_csv`	Character vector of length one. Path to a CSV file created with the point selection tool of 'ImageJ' software.
`degree`	Numeric vector of length one. Polynomial model degree.

Details

If you cannot find the coefficient of your lens on the literature, you may want to try the solution offered here. It requires, in addition to this package and the open-source ImageJ software package, the following materials:

- camera and lens
- tripod
- standard yoga mat
- table larger than the yoga mat
- twenty two push pins of different colors
- scissors
- One print of this sheet (A1 size, almost like a poster).

Cut the sheet by the dashed line. Place the yoga mat extended on top of the table. Place the sheet on top of the yoga mat. Align the dashed line with the yoga mat border closest to you, and place push pins on each cross. If you are gentle, the yoga mat will allows you to do that without damaging the table. Of course, if you can access to a workshop you may have other tools to do this setup, such as a board and nails.

Place the camera on the tripod, align its optical axis with the table while looking for getting an image showing the overlapping of the three pairs of push pins as instructed in the print. Take a photograph and check if it looks more or less like this one.

Transfer the photograph to the computer, open it with ImageJ, and use the point selection tool to digitize the push pins, starting from the zenith pushpin and not skipping any showed push pin. This method was inspired by the calibration board from Clark and Follin (1988).

Value

An object of class list with named elements. 'lens_coef' stands for lens coefficients, 'max_theta' for maximum zenith angle in degrees, and 'max_theta_px' for distance in pixels between the zenith and the maximum zenith angle in pixels units.

References

Clark JA, Follin GM (1988). "A simple equal area calibration for fisheye photography." *Agricultural and Forest Meteorology*, **44**(1), 19–25. doi: 10.1016/01681923(88)900305, `https://doi.org/10.1016/0168-1923(88)90030-5`.

See Also

Other Lens functions: `azimuth_image()`, `calc_diameter()`, `calc_zenith_raster_coordinates()`, `expand_noncircular()`, `lens()`, `reproject_to_equidistant()`, `test_lens_coef()`, `zenith_image()`

Examples

```
path <- system.file("external/Results_calibration.csv", package = "rcaiman")
calibration <- calibrate_lens(path)
calibration$lens_coef
calibration$max_theta
calibration$max_thera_px
```

`enhance_caim` *Enhance canopy image*

Description

This function is presented in Díaz and Lencinas (2015). It uses the color perceptual attributes to enhance the contrast between the sky and plants through fuzzy classification. Color has three different perceptual attributes: hue, lightness, and chroma. The algorithm was developed following this premise: the color of the sky is different from the color of plants. It performs the next classification rules, here expressed in natural language: clear sky is blue and clouds decrease its chroma; if clouds are highly dense, then the sky is achromatic, and, in such cases, it can be light or dark; everything that does not match this description is not sky. These linguistic rules were translated to math language by means of fuzzy logic.

Usage

```
enhance_caim(caim, m, w_red = 0.5, sky_blue = NULL)
```

Arguments

`caim`	RasterBrick. The return of a call to `read_caim`.
`m`	RasterLayer. A mask. Usually, the result of a call to `mask_hs`.
`w_red`	Numeric vector of length one. Weight of the red channel. A single layer image is calculated as a weighted average of the blue and red channels. This layer is used as lightness information. The weight of the blue is the complement of `w_red`.
`sky_blue`	color. Is the `target_color` argument to be passed to `membership_to_color`. By default is pure Blue.

Details

This is a pixel-wise methodology that evaluates the possibility for a pixel to be member of the class Gap. High score could mean either high membership to `sky_blue` or, in the case of achromatic pixels, a high membership to `thr`. The algorithm internally uses `membership_to_color` and `local_fuzzy_thresholding`. The argument `sky_blue` is the `target_color` of the former function, which output is the argument `mem` of the latter function.

References

Díaz GM, Lencinas JD (2015). "Enhanced Gap Fraction Extraction From Hemispherical Photography." *IEEE Geoscience and Remote Sensing Letters*, **12**(8), 1785–1789. doi: 10.1109/lgrs.2015.2425931.

See Also

Other Fuzzy logic functions: `local_fuzzy_thresholding()`, `membership_to_color()`

Examples

```
## Not run:
caim <- read_caim()
caim <- normalize(caim, 0, 255)
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
m <- !is.na(z)
ecaim <- enhance_caim(caim, m)
plot(ecaim)

## End(Not run)
```

`expand_noncircular` *Expand non-circular*

Description

Expand a non-circular hemispherical photograph.

Usage

```
expand_noncircular(caim, z, zenith_colrow)
```

Arguments

`caim`	RasterBrick. The return of a call to `read_caim`.
`z`	`RasterLayer` built with `zenith_image`.
`zenith_colrow`	Numeric vector of length two. Raster coordinates of the zenith. See `calc_zenith_raster_coordinates`.

See Also

Other Lens functions: `azimuth_image()`, `calc_diameter()`, `calc_zenith_raster_coordinates()`, `calibrate_lens()`, `lens()`, `reproject_to_equidistant()`, `test_lens_coef()`, `zenith_image()`

Examples

```
## Not run:
   my_file <- path.expand("~/DSC_2881.JPG")
   download.file("https://osf.io/x8urg/download", my_file,
               method = "auto", mode = "wb"
   )

   r <- read_caim(my_file)
   diameter <- calc_diameter(lens("Nikkor_10.5_mm"), 1202, 53)
   zenith_colrow <- c(1503, 998)
   z <- zenith_image(diameter, lens("Nikkor_10.5_mm"))
   r <- expand_noncircular(r, z, zenith_colrow)
   plot(r)

## End(Not run)
```

`extract_feature` *Extract feature*

Description

Extract features from raster images.

Usage

```
extract_feature(r, segmentation, fun = mean, return_raster = TRUE)
```

Arguments

`r`	RasterLayer. Single layer raster.
`segmentation`	RasterLayer. The segmentation of `r`.
`fun`	`function` that takes a vector as input and returns a one-length numeric or logical vector as output (e.g. mean).
`return_raster`	Logical vector of length one.

Details

Given a single layer raster, a segmentation, and a function, `extract_features` will returns a numeric vector or a RasterLayer depending on whether the parameter `return_raster` is `TRUE` or `FALSE`. For the first case, each pixel of each segment will adopt the respective extracted feature value. For the second case, the return will be the extracted feature as a vector of length equal to the total number of segments. Each extracted feature value will be obtained by processing all pixels that belong to a segment with the provided function.

See Also

Other Tools functions: `apply_thr()`, `gbc()`, `masking()`, `normalize()`, `read_bin()`, `read_caim()`, `regional_thresholding()`, `write_bin()`, `write_caim()`

Examples

```
## Not run:
r <- read_caim()
z <- zenith_image(ncol(r),lens("Nikon_FCE9"))
a <- azimuth_image(z)
g <- sky_grid_segmentation(z, a, 10)
extract_feature(r$Blue, g, return_raster = FALSE)
plot(extract_feature(r$Blue, g, return_raster = FALSE))

## End(Not run)
```

`find_sky_dns` *Find sky DNs*

Description

Find sky digital numbers automatically

Usage

```
find_sky_dns(r, z, a, no_of_samples = 30)
```

Arguments

`r`	RasterLayer. A normalized greyscale image. Typically, the blue channel extracted from an hemispherical photograph. Please see `read_caim` and `normalize`.
`z`	RasterLayer. The result of a call to `zenith_image`.
`a`	RasterLayer. The result of a call to `azimuth_image`.
`no_of_samples`	Numeric vector of length one. Minimum number of samples required.

Details

This function assumes that (1) there is at least one pure sky pixel at the level of cells of 30×30 degrees, and (2) sky pixels have a digital number (DN) greater than canopy pixels have.

For each cell, it compute a quantile value and use it as a threshold to select the pure sky pixels of the cell, which produce binarized image as a result in a regional binarization fashion (`regional_thresholding`). This process start with a quantile probability of 0.99. After producing the binarized image, this function use a search grid with cells of 5×5 degrees to count how many cells on the binarired image have at least one sky pixel. If the count does not reach argument `no_of_samples`, it goes back to the binarization step but decreasing the probability by 0.01 points.

See Also

Other MBLT functions: `fit_cone_shaped_model()`, `fit_trend_surface()`, `ootb_mblt()`, `thr_image()`

Examples

```
## Not run:
path <- system.file("external/4_D_2_DSCN4502.JPG", package = "rcaiman")
caim <- read_caim(path, c(1280, 960) - 745, 745 * 2, 745 * 2)
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
a <- azimuth_image(z)
blue <- gbc(caim$Blue)
bin <- find_sky_dns(blue, z, a)
plot(bin)

## End(Not run)
```

`fit_cone_shaped_model` *Fit cone shaped model*

Description

Generate the digital numbers of the whole sky through statistical modelling.

Usage

```
fit_cone_shaped_model(
  r,
  z,
  a,
  bin,
  prob = 0.95,
  filling_source = NULL,
  use_azimuth_angle = TRUE,
  parallel = TRUE,
  free_cores = 0
)
```

Arguments

`r`	RasterLayer. A normalized greyscale image. Typically, the blue channel extracted from an hemispherical photograph. Please see `read_caim` and `normalize`.
`z`	RasterLayer. The result of a call to `zenith_image`.
`a`	RasterLayer. The result of a call to `azimuth_image`.
`bin`	RasterLayer. A working binarized image. This should be a preliminary binarization of `r`. If the function returns `NA`, the quality of this input should be revised.
`prob`	Logical vector of length one. Probability for `quantile` calculation. See reference Díaz and Lencinas (2018).
`filling_source`	RasterLayer. Default is `NULL`. Above-canopy photograph. This image should contain pixels with sky DN values and `NA` in all the other pixels. A photograph taken immediately after or before taking `r` under the open sky with the same equipment and configuration is a very good option. The ideal option is one taken at the same time and place but above the canopy. The orientation relative to the North must be the same than for `r`.

`use_azimuth_angle`
: Logical vector of length one. If `TRUE`, Equation 4 from Díaz and Lencinas (2018) is used: $sDN = a + b \cdot \theta + c \cdot \theta^2 + d \cdot sin(\phi) + e \cdot cos(\phi)$, where sDN is sky digital number, a, b, c, d and e are coefficients, θ is zenith angle, and ϕ is azimuth angle. If `FALSE`, a simplified version based on Wagner (2001) is used: $sDN = a + b \cdot \theta + c \cdot \theta^2$.

`parallel`
: Logical vector of length one. Allows parallel processing.

`free_cores`
: Numeric vector of length one. This number is subtracted to the number of cores detected by `detectCores`.

Details

An explanation of this function can be found on Díaz and Lencinas (2018), under the heading *Estimation of the sky DN as a previous step for our method.*

Value

A list with two objects, one of class RasterLayer and the other of class `lm` (see `lm`).

References

Díaz GM, Lencinas JD (2018). "Model-based local thresholding for canopy hemispherical photography." *Canadian Journal of Forest Research*, **48**(10), 1204–1216. doi: 10.1139/cjfr-20180006.

Wagner S (2001). "Relative radiance measurements and zenith angle dependent segmentation in hemispherical photography." *Agricultural and Forest Meteorology*, **107**(2), 103–115. doi: 10.1016/s01681923(00)00232x, `https://doi.org/10.1016/s0168-1923(00)00232-x`.

See Also

Other MBLT functions: `find_sky_dns()`, `fit_trend_surface()`, `ootb_mblt()`, `thr_image()`

Examples

```
## Not run:
path <- system.file("external/4_D_2_DSCN4502.JPG", package = "rcaiman")
r <- read_caim(path, c(1280, 960) - 745, 745 * 2, 745 * 2)
z <- zenith_image(ncol(r), lens("Nikon_FCE9"))
a <- azimuth_image(z)
blue <- gbc(r$Blue)
bin <- find_sky_dns(blue, z, a)
sky <- fit_cone_shaped_model(blue, z, a, bin, parallel = FALSE)
plot(sky$image)
persp(sky$image, theta = 90, phi = 0) #a flipped rounded cone!

## End(Not run)
```

fit_trend_surface *Fit a trend surface to sky digital numbers*

Description

Fit a trend surface using spatial::surf.ls as workhorse function.

Usage

```
fit_trend_surface(
  r,
  bin,
  m = NULL,
  filling_source = NULL,
  prob = 0.95,
  fact = 5,
  np = 6
)
```

Arguments

`r`	RasterLayer. A normalized greyscale image. Typically, the blue channel extracted from an hemispherical photograph. Please see `read_caim` and `normalize`.
`bin`	RasterLayer. A working binarized image. This should be a preliminary binarization of `r`. If the function returns `NA`, the quality of this input should be revised.
`m`	RasterLayer. A mask. Usually, the result of a call to `mask_hs`.
`filling_source`	RasterLayer. Default is `NULL`. Above-canopy photograph. This image should contain pixels with sky DN values and `NA` in all the other pixels. A photograph taken immediately after or before taking `r` under the open sky with the same equipment and configuration is a very good option. The ideal option is one taken at the same time and place but above the canopy. The orientation relative to the North must be the same than for `r`.
`prob`	Logical vector of length one. Probability for `quantile` calculation. See reference Díaz and Lencinas (2018).
`fact`	postive integer. Aggregation factor expressed as number of cells in each direction (horizontally and vertically). Or two integers (horizontal and vertical aggregation factor) or three integers (when also aggregating over layers). See Details
`np`	degree of polynomial surface

Details

This function is meant to be used after `fit_cone_shaped_model`.

A short explanation of this function can be found on Díaz and Lencinas (2018), under the heading *Estimation of the sky DN as a previous step for our method*, after the explanation of the `fit_cone_shaped_model`.

The argument fact is passed to aggregate. That argument allows to control the scale at which the fitting is performed. In general, a coarse scale lead to best generalization. The function used for aggregation is quantile, to which the argument prob is passed. Essentially, the aggregation step works as the one from fit_cone_shaped_model, but it is made on the raster space rather than on the hemispherical space.

Value

A list with an object of class RasterLayer and of class trls (see surf.ls).

References

Díaz GM, Lencinas JD (2018). "Model-based local thresholding for canopy hemispherical photography." *Canadian Journal of Forest Research*, **48**(10), 1204–1216. doi: 10.1139/cjfr20180006.

See Also

Other MBLT functions: find_sky_dns(), fit_cone_shaped_model(), ootb_mblt(), thr_image()

Examples

```
## Not run:
path <- system.file("external/4_D_2_DSCN4502.JPG", package = "rcaiman")
r <- read_caim(path, c(1280, 960) - 745, 745 * 2, 745 * 2)
z <- zenith_image(ncol(r), lens("Nikon_FCE9"))
a <- azimuth_image(z)
blue <- gbc(r$Blue)
bin <- find_sky_dns(blue, z, a)
sky <- fit_cone_shaped_model(blue, z, a, bin, parallel = FALSE)
m <- mask_hs(z, 0, 80)
sky <- fit_trend_surface(blue, bin, m, filling_source = sky$image)
plot(sky$image)

## End(Not run)
```

---

gbc *Gamma back correction*

Description

Gamma back correction of JPEG images.

Usage

```
gbc(DN_from_JPEG, gamma = 2.2)
```

Arguments

DN_from_JPEG	Numeric vector or object from the Raster class. Digital numbers from a JPEG file.
gamma	Numeric vector of length one. Gamma value. Please see Díaz and Lencinas (2018) for details.

Value

Normalized Raster.

References

Díaz GM, Lencinas JD (2018). "Model-based local thresholding for canopy hemispherical photography." *Canadian Journal of Forest Research*, **48**(10), 1204–1216. doi: 10.1139/cjfr20180006.

See Also

`normalize`

Other Tools functions: `apply_thr()`, `extract_feature()`, `masking()`, `normalize()`, `read_bin()`, `read_caim()`, `regional_thresholding()`, `write_bin()`, `write_caim()`

Examples

```
r <- read_caim()
gbc(r)
```

`lens` *Lens database*

Description

Database of lens projection functions and field of views.

Usage

```
lens(type = "equidistant", max_fov = FALSE)
```

Arguments

`type`	Character vector of length one. The name of the lens, see details.
`max_fov`	Logical. Use `TRUE` to return the maximum field of view in degrees.

Details

Eventually, this will be a large database, but only the following lenses are available at the moment:

- **equidistant**: standard equidistant projection (Schneider et al. 2009).
- **Nikon_FCE9**: Nikon FC-E9 auxiliary lens (Díaz and Lencinas 2018)
- **Nikkor_10.5_mm**: AF DX Fisheye-Nikkor 10.5mm f/2.8G ED (Pekin and Macfarlane 2009)

References

Díaz GM, Lencinas JD (2018). "Model-based local thresholding for canopy hemispherical photography." *Canadian Journal of Forest Research*, **48**(10), 1204–1216. doi: 10.1139/cjfr-20180006.

Pekin B, Macfarlane C (2009). "Measurement of crown cover and leaf area index using digital cover photography and its application to remote sensing." *Remote Sensing*, **1**(4), 1298–1320. doi: 10.3390/rs1041298.

Schneider D, Schwalbe E, Maas H (2009). "Validation of geometric models for fisheye lenses." *ISPRS Journal of Photogrammetry and Remote Sensing*, **64**(3), 259–266. doi: 10.1016/j.isprsjprs.2009.01.001.

See Also

Other Lens functions: `azimuth_image()`, `calc_diameter()`, `calc_zenith_raster_coordinates()`, `calibrate_lens()`, `expand_noncircular()`, `reproject_to_equidistant()`, `test_lens_coef()`, `zenith_image()`

Examples

```
lens("equidistant")
```

---

local_fuzzy_thresholding

local fuzzy thresholding

Description

This function is presented in Díaz and Lencinas (2015). It uses a threshold value as the location parameter of a logistic membership function whose scale parameter depends on a variable, here named `mem`. This dependence can be explained as follows: if the variable is equal to `1`, then the membership function is same as a threshold function because the scale parameter is `0`; lowering the variable increases the scale parameter, thus blurring the threshold because it decreases the steepness of the curve. Since the variable is defined pixel by pixel, this should be considered as a **local** fuzzy thresholding method.

Usage

```
local_fuzzy_thresholding(lightness, m, mem, thr = NULL, fuzziness = NULL)
```

Arguments

`lightness`	RasterLayer. A normalized greyscale image, the lightness value. Values should range between zero and one –please see `normalize`.
`m`	RasterLayer. A mask. Usually, the result of a call to `mask_hs`.
`mem`	RasterLayer. It is the scale parameter of the logistic membership function. Typically it is obtained with `membership_to_color`.
`thr`	Numeric vector of length one. Location parameter of the logistic membership function. Use `NULL` (default) to order to estimate it automatically with the function `auto_thresh`, method `"IsoData"`.

`fuzziness`	Numeric vector of length one. This number is a constant that scale `mem`. Use `NULL` (default) to order to estimate it automatically as the midpoint between the maximum and minimum values of `lightness`.

Details

Argument `m` can be used to affect the estimation of `thr` and `fuzziness`.

References

Díaz GM, Lencinas JD (2015). "Enhanced Gap Fraction Extraction From Hemispherical Photography." *IEEE Geoscience and Remote Sensing Letters*, **12**(8), 1785–1789. doi: 10.1109/lgrs.2015.2425931.

See Also

Other Fuzzy logic functions: `enhance_caim()`, `membership_to_color()`

Examples

```
## Not run:
caim <- read_caim()
caim <- normalize(caim, 0, 255)
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
target_color <- colorspace::sRGB(matrix(c(0.529, 0.808, 0.921), ncol = 3))
mem <- membership_to_color(caim, target_color)
m <- !is.na(z)
mem_thr <- local_fuzzy_thresholding(mean(caim), m,  mem$membership_to_grey)
plot(mem_thr)

## End(Not run)
```

`masking` *Image masking*

Description

Image masking

Usage

```
masking(r, m, RGB = c(1, 0, 0))

## S4 method for signature 'RasterLayer'
masking(r, m, RGB = c(1, 0, 0))

## S4 method for signature 'RasterStackBrick'
masking(r, m, RGB = c(1, 0, 0))
```

Arguments

`r`	Raster. The image. Values should be normalized, see `normalize`.
`m`	RasterLayer. The mask, see `mask_hs`.
`RGB`	Numeric vector of length three. RGB color code. Red is the default color.

Value

RasterStack

See Also

mask_hs

Other Tools functions: apply_thr(), extract_feature(), gbc(), normalize(), read_bin(), read_caim(), regional_thresholding(), write_bin(), write_caim()

Examples

```
 r <- read_caim()
 z <- zenith_image(ncol(r), lens())
 a <- azimuth_image(z)
 m <- mask_hs(z, 20, 70) & mask_hs(a, 90, 180)

 masked_caim <-  masking(normalize(r, 0, 255), m)
 plotRGB(masked_caim * 255)
## Not run:
 masked_bin <- masking(apply_thr(r$Blue, 125), m)
 plotRGB(masked_bin * 255)

## End(Not run)
```

mask_hs *Mask hemisphere*

Description

Given a zenith or azimuth image and angle restrictions, it produces a mask.

Usage

```
mask_hs(r, from, to)
```

Arguments

r	RasterLayer. The result of a call to zenith_image or azimuth_image.
from, to	angle in degrees, inclusive limits.

See Also

masking

Other Segmentation functions: rings_segmentation(), sectors_segmentation(), sky_grid_segmentation()

Examples

```
## Not run:
z <- zenith_image(1000, lens())
a <- azimuth_image(z)
m1 <- mask_hs(z, 20, 70)
plot(m1)
m2 <- mask_hs(a, 330,360)
plot(m2)
plot(m1 & m2)
plot(m1 | m2)

# if you want 15 degress at each side of 0
m1 <- mask_hs(a, 0, 15)
m2 <- mask_hs(a, 345, 360)
plot(m1 | m2)

# better use this
plot(!is.na(z))
# instead of this
plot(mask_hs(z, 0, 90))

## End(Not run)
```

`membership_to_color` *Compute membership to a color*

Description

This function is presented in Díaz and Lencinas (2015). It Computes the degree of membership to a color with two Gaussian membership functions and the dimensions A and B from the *CIE L*a*b** color space. The lightness dimension is not considered in the calculations.

Usage

```
membership_to_color(caim, target_color, sigma = NULL)
```

Arguments

`caim`	RasterBrick. The return of a call to `read_caim`.
`target_color`	color.
`sigma`	Numeric vector of length one. Use `NULL` (default) to estimate it automatically as the euclidean distance between `target_color` and grey in the *CIE L*a*b** color space.

Value

It returns an object from the class RasterBrick or RasterStack –this will depend on the input. First layer is the membership to the target color. Second layer is the membership to grey. Both memberships are calculated with same `sigma`.

References

Díaz GM, Lencinas JD (2015). "Enhanced Gap Fraction Extraction From Hemispherical Photography." *IEEE Geoscience and Remote Sensing Letters*, **12**(8), 1785–1789. doi: 10.1109/lgrs.2015.2425931.

See Also

Other Fuzzy logic functions: `enhance_caim()`, `local_fuzzy_thresholding()`

Examples

```
caim <- read_caim()
caim <- normalize(caim, 0, 255)
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
target_color <- colorspace::sRGB(matrix(c(0.529, 0.808, 0.921), ncol = 3))
mem <- membership_to_color(caim, target_color)
plot(mem)
```

`normalize` *Normalize data*

Description

Normalize data laying between `mn` and `mx` in the range `0` to `1`. Data greater than `mx` get values greater than `1` in a proportional fashion. Conversely, data less than `mn` get values less than `0`.

Usage

```
normalize(r, mn = NULL, mx = NULL)
```

Arguments

`r`	`Raster` or numeric vector.
`mn`	Numeric vector of length one. Minimum expected value.
`mx`	Numeric vector of length one. Maximum expected value.

See Also

`gbc`

Other Tools functions: `apply_thr()`, `extract_feature()`, `gbc()`, `masking()`, `read_bin()`, `read_caim()`, `regional_thresholding()`, `write_bin()`, `write_caim()`

Examples

```
normalize(read_caim(), 0, 255)
```

`ootb_mblt`	*Out-of-the-box model-based local thresholding*

Description

Out-of-the-box version of the model-based local thresholding (MBLT) algorithm.

Usage

```
ootb_mblt(r, z, a)
```

Arguments

`r`	RasterLayer. A normalized greyscale image. Typically, the blue channel extracted from an hemispherical photograph. Please see `read_caim` and `normalize`.
`z`	RasterLayer. The result of a call to `zenith_image`.
`a`	RasterLayer. The result of a call to `azimuth_image`.

Details

This function is a hard-coded version of a MBLT pipeline that starts with a working binarized image and ends with a refined binarized image. The pipeline combines `find_sky_dns`, `fit_cone_shaped_model`, `fit_trend_surface`, and `thr_image`. The code can be easily inspected by calling `ootb_mblt` –no parenthesis. Advanced users can use that code as a template.

The MBLT algorithm was first presented in Díaz and Lencinas (2018). The version presented here differs from that in the following main aspects:

- *intercept* is set to 0, *slope* to 1, and w to 0.5
- This version implements a regional threholding approach as first step instead of a global one. Please refer to `find_sky_dns`. The minimum number of samples (sky DNs) required is equals to the 30 percent of the population, considering that it is made of 5×5 sky grid cells.
- It does not use asynchronous acquisition under the open sky. So, the cone shaped model (`fit_cone_shaped_model`) run without a filling source, but the result of it is used as filling source for trend surface fitting (`fit_trend_surface`).

This function searches for black objects against a light background. When regular canopy hemispherical images are provided as input, the algorithm will find dark canopy elements against a bright sky almost everywhere in the picture, and the result will fit user expectations. However, if an hemispherical photograph taken under the open sky is provided, this algorithm would be still searching black objects against a light background, so the darker portions of the sky will be taken as objects, i.e., canopy. As a consequence, this will not fit users expectations, since they require the classes 'Gap' and 'No-gap'. This kind of error could be find in photographs of open forests for the same reason.

Value

Object of class list with the binarized image (named 'bin') and the reconstructed skies (named 'sky_cs' and 'sky_s').

References

Díaz GM, Lencinas JD (2018). "Model-based local thresholding for canopy hemispherical photography." *Canadian Journal of Forest Research*, **48**(10), 1204–1216. doi: 10.1139/cjfr20180006.

See Also

Other MBLT functions: `find_sky_dns()`, `fit_cone_shaped_model()`, `fit_trend_surface()`, `thr_image()`

Examples

```
## Not run:
path <- system.file("external/4_D_2_DSCN4502.JPG", package = "rcaiman")
r <- read_caim(path, c(1280, 960) - 745, 745 * 2, 745 * 2)
z <- zenith_image(ncol(r), lens("Nikon_FCE9"))
a <- azimuth_image(z)
blue <- gbc(r$Blue)
bin <- ootb_mblt(blue, z, a)
plot(bin$bin)

## End(Not run)
```

`read_bin` *Read binarized images*

Description

Wrapper functions for `raster`.

Usage

```
read_bin(path)
```

Arguments

`path`	One-length character vector. Path to read or a binarized image.

See Also

`write_bin`

Other Tools functions: `apply_thr()`, `extract_feature()`, `gbc()`, `masking()`, `normalize()`, `read_caim()`, `regional_thresholding()`, `write_bin()`, `write_caim()`

Examples

```
## Not run:
z <- zenith_image(1000, lens())
m <- !is.na(z)
write_bin(m, "mask")
m_from_disk <- read_bin("mask.tif")
plot(m - m_from_disk)

## End(Not run)
```

`read_caim` *Read a canopy image from a file*

Description

Wrapper function for `raster`.

Usage

```
read_caim(path_to_file, upper_left = NULL, width = NULL, height = NULL)

## S4 method for signature 'character'
read_caim(path_to_file, upper_left = NULL, width = NULL, height = NULL)

## S4 method for signature 'missing'
read_caim(path_to_file)
```

Arguments

`path_to_file`	Character vector of length one. Path to a JPEG or TIFF file. The function will return a data example (see details) if no arguments are provided.
`upper_left`	An integer vector of length two (see details).
`width, height`	An integer vector of length one (see details).

Details

Run `read_caim()` to obtain an example of a hemispherical photo taken in non-diffuse light conditions in a *Nothofagus pumilio* forest from Argentina with a FC-E9 auxiliary lens attached to a Nikon Coolpix 5700.

Since this function aims to read born-digital color photographs, RGB-JPEG and RGB-TIFF are expected as input. To read a region of the file use `upper_left`, `width`, and `height`. The `upper_left` parameter indicates the pixels coordinates of the upper left corner of the region of interest (ROI). These coordinates should be in the raster coordinates system, which works like a spreadsheet, i.e, when you go down through the vertical axis, the *row* number increases (**IMPORTANT: column and row must be provided instead of row and column**). The `width`, and `height` parameters indicate the size of the boxy ROI. I recommend using 'ImageJ' to obtain this parameters, but any image editor can be used, such as 'GIMP' and 'Adobe Photoshop'.

Value

`RasterBrick`.

Functions

- `read_caim,character-method`: Provide the path to a file. If The file is stored in the working directory, just provide the file name. File extension should be included in the file name.
- `read_caim,missing-method`: It returns an example (see details).

See Also

write_caim

Other Tools functions: apply_thr(), extract_feature(), gbc(), masking(), normalize(), read_bin(), regional_thresholding(), write_bin(), write_caim()

Examples

```
# This is the example image
r <- read_caim()
plotRGB(r)

# This is also the example
path <- system.file("external/b4_2_5724.jpg", package = "rcaiman")
# the zenith raster coordinates can be easily transformed to the "upper_left"
# argument by subtracting from it the radius expressed in pixels.
zenith_colrow <- c(1280, 960)
diameter_px <- 1490
r <- read_caim(path,
               upper_left = zenith_colrow - diameter_px/2,
               width = diameter_px,
               height = diameter_px)
```

---

regional_thresholding *Regional thresholding*

Description

Regional thresholding of greyscale images

Usage

```
regional_thresholding(
  r,
  segmentation,
  method,
  intercept = NULL,
  slope = NULL,
  prob = NULL
)
```

Arguments

r	RasterLayer. Normalized greyscale image. See normalize and gbc
segmentation	RasterLayer. The result of segmenting r. Probably, rings_segmentation will be the most used for fisheye images.
method	Character vector of length one. See details for current options.
intercept	Numeric vector of length one. These are linear function coefficients. Please, see the Details section of thr_image.
slope	Numeric vector of length one. These are linear function coefficients. Please, see the Details section of thr_image.
prob	Logical vector of length one. Probability for quantile calculation. See reference Díaz and Lencinas (2018).

Details

Methods currently implemented are:

- **Diaz2018**: method presented in Díaz and Lencinas (2018) applied regionally. If this method is selected, the arguments `intercept`, `slope`, and `prob` should be provided. It works segmentwise extracting the digital numbers (dns) per segment and passing them to `quantile(dns,prob)`, which aggregated result (x) is in turn passed to `thr_image(x,intercept,slope)`. Finally, this threshold image is applied to obtain a binarized image.
- **Methods from autothresholdr package**: this function can call methods from `auto_thresh`. Use `"IsoData"` to use the algorithm by Ridler and Calvard (1978), which is the one recommended by Jonckheere et al. (2005).

Value

RasterLayer.

References

Díaz GM, Lencinas JD (2018). "Model-based local thresholding for canopy hemispherical photography." *Canadian Journal of Forest Research*, **48**(10), 1204–1216. doi: 10.1139/cjfr-20180006.

Jonckheere I, Nackaerts K, Muys B, Coppin P (2005). "Assessment of automatic gap fraction estimation of forests from digital hemispherical photography." *Agricultural and Forest Meteorology*, **132**(1-2), 96–114. doi: 10.1016/j.agrformet.2005.06.003.

Ridler TW, Calvard S (1978). "Picture thresholding using an iterative selection method." *IEEE Transactions on Systems, Man, and Cybernetics*, **8**(8), 630–632. doi: 10.1109/tsmc.1978.4310039.

See Also

`thr_image`

Other Tools functions: `apply_thr()`, `extract_feature()`, `gbc()`, `masking()`, `normalize()`, `read_bin()`, `read_caim()`, `write_bin()`, `write_caim()`

Examples

```
r <- read_caim()
blue <- gbc(r$Blue)
z <- zenith_image(ncol(r), lens("Nikon_FCE9"))
rings <- rings_segmentation(z, 10)
bin <- regional_thresholding(blue, rings, "Diaz2018", -8, 0.5, 0.9)
```

reproject_to_equidistant

Reproject to equidistant

Description

Reproject to equidistant

Usage

```
reproject_to_equidistant(r, z, a, radius = 745)

## S4 method for signature 'RasterLayer'
reproject_to_equidistant(r, z, a, radius = 745)

## S4 method for signature 'RasterStackBrick'
reproject_to_equidistant(r, z, a, radius = 745)
```

Arguments

`r`	RasterLayer. A normalized greyscale image. Typically, the blue channel extracted from an hemispherical photograph. Please see `read_caim` and `normalize`.
`z`	RasterLayer. The result of a call to `zenith_image`.
`a`	RasterLayer. The result of a call to `azimuth_image`.
`radius`	Numeric integer of length one. Radius of the reprojected hemispherical image.

See Also

Other Lens functions: `azimuth_image()`, `calc_diameter()`, `calc_zenith_raster_coordinates()`, `calibrate_lens()`, `expand_noncircular()`, `lens()`, `test_lens_coef()`, `zenith_image()`

Examples

```
## Not run:
caim <- read_caim()
caim <- normalize(caim, 0, 255)
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
a <- azimuth_image(z)
bin <- apply_thr(caim$Blue, 0.5)
bin_equi <- reproject_to_equidistant(bin, z, a, radius = 400)
bin_equi <- apply_thr(bin_equi, 0.5)
plot(bin_equi)
write_bin(bin_equi, "bin") #ready for CIMES, GLA, CAN-EYE, etc.

## End(Not run)
```

`rings_segmentation` *Rings segmentation*

Description

Segmenting an hemispherical view by slicing the zenith angle from `0` to `90º` in equals intervals.

Usage

```
rings_segmentation(z, angle_width, return_angle = FALSE)
```

Arguments

`z`	`RasterLayer` built with `zenith_image`.
`angle_width`	Numeric vector of length one. Angle in degrees able to divide the angle range into a whole number of segments.
`return_angle`	Logical vector of length one. If it is `FALSE`, all the pixels that belong to a segment are labeled with an ID number. Otherwise, the angle mean of the segment is assigned to the pixels.

Value

RasterLayer with segments shaped like concentric rings.

See Also

Other Segmentation functions: `mask_hs()`, `sectors_segmentation()`, `sky_grid_segmentation()`

Examples

```
z <- zenith_image(1490, lens())
rings <- rings_segmentation(z, 15)
plot(rings == 1)
```

`sectors_segmentation` *Sectors segmentation*

Description

Segmenting an hemispherical view by slicing the azimuth angle from `0` to `360º` in equals intervals.

Usage

```
sectors_segmentation(a, angle_width, return_angle = FALSE)
```

Arguments

`a`	`RasterLayer` built with `azimuth_image`.
`angle_width`	Numeric vector of length one. Angle in degrees able to divide the angle range into a whole number of segments.
`return_angle`	Logical vector of length one. If it is `FALSE`, all the pixels that belong to a segment are labeled with an ID number. Otherwise, the angle mean of the segment is assigned to the pixels.

Value

RasterLayer with segments shaped like pizza slices.

See Also

Other Segmentation functions: `mask_hs()`, `rings_segmentation()`, `sky_grid_segmentation()`

Examples

```
z <- zenith_image(1490, lens())
a <- azimuth_image(z)
sectors <- sectors_segmentation(a, 15)
plot(sectors == 1)
```

`sky_grid_segmentation` *Sky grid segmentation*

Description

Segmenting the hemisphere view into segments of equal angular resolution for both zenith and azimuth angles.

Usage

```
sky_grid_segmentation(z, a, angle_width, sequential = FALSE)
```

Arguments

`z`	`RasterLayer` built with `zenith_image`.
`a`	`RasterLayer` built with `azimuth_image`.
`angle_width`	Numeric vector of length one. It should be `30,15,10,7.5,6,5,3.75,3,2.5,1.875,1` or `0.5` degrees. This constrain is rooted in the requirement of a value able to divide both the `0` to `360` and `0` to `90` ranges into a whole number of segments.
`sequential`	Logical vector of length one. If it is `TRUE`, the segments are labeled with sequential numbers. By default (`FALSE`), labeling numbers are not sequential (see Details).

Details

Intersecting rings with sectors makes a grid in which each segment is a portion of the hemisphere. Each pixel of the grid is labeled with an ID that codify both ring and sector IDs. For example, a grid with a regular interval of one degree has segment from `1001` to `360090`. This numbers are calculated with: `sectorID * 1000 + ringsID`, where `sectorID` is the ID number of the sector and `ringsID` is the ID number of the ring.

Value

RasterLayer with segments shaped like windshields, although some of them will look elongated in height. The pattern is two opposite and converging straight sides and two opposite and parallel curvy sides.

See Also

Other Segmentation functions: `mask_hs()`, `rings_segmentation()`, `sectors_segmentation()`

Examples

```
z <- zenith_image(1490, lens())
a <- azimuth_image(z)
g <- sky_grid_segmentation(z, a, 15)
plot(g == 36009)
## Not run:
g <- sky_grid_segmentation(z, a, 15, sequential = TRUE)
plot(g, col = sample(rainbow(length(raster::unique(g)))))

## End(Not run)
```

`test_lens_coef` *Test lens projection functions*

Description

Test that lens projection function works between the 0-to-1 range.

Usage

```
test_lens_coef(lens_coef)
```

Arguments

`lens_coef`	Numeric vector. Polynomial coefficients of the lens projection function.

See Also

Other Lens functions: `azimuth_image()`, `calc_diameter()`, `calc_zenith_raster_coordinates()`, `calibrate_lens()`, `expand_noncircular()`, `lens()`, `reproject_to_equidistant()`, `zenith_image()`

Examples

```
test_lens_coef(lens("Nikon_FCE9"))
test_lens_coef(2 / pi)
test_lens_coef(c(1.06065, -0.49054, 0.14044))
```

thr_image *Threshold image*

Description

Transform background digital number into threshold values.

Usage

```
thr_image(dn, intercept, slope)
```

Arguments

`dn`	Numeric vector or RasterLayer. Digital number of the background. These values should be normalized and, if they are extracted from JPEG image, gamma back corrected.
`intercept, slope`	Numeric vector of length one. These are linear function coefficients. Please, see the Details section of `thr_image`.

Details

This function transforms background digital number into threshold values by means of the Equation 1 presented in Díaz and Lencinas (2018), which is a linear function with the slope modified by a weighting parameter. This simple function was found by studying canopy models, also known as targets, which are planes with holes made of a rigid and dark material. These models were backlighted with homogeneous lighting, photographed with a Nikon Coolpix 5700 set to acquire in JPEG format, and those images were gamma back corrected with a default gamma value equal to 2.2 (see `gbc`). Results clearly shown that the optimal threshold value was linearly related with the background digital number. Therefore, that shifts the aim from finding the optimal threshold to obtaining the background DN as if the canopy was not there. Functions `fit_cone_shaped_model` and `fit_trend_surface` address that topic.

It is worth noting that Equation 1 was developed with 8-bit images, so calibration of new coefficient should be done in the 0 to 255 domain since that is what `thr_image` expect, although the input `dn` should be normalized. The latter –that might sound counter intuitive– was a design decision aiming to harmonize the whole package.

To apply the weighting parameter (w) from Equation 1, just provide the argument `slope` as `slope_value * w`.

Type `thr_image` –no parenthesis– in the console to inspect the code, which is very simple to follow.

References

Díaz GM, Lencinas JD (2018). "Model-based local thresholding for canopy hemispherical photography." *Canadian Journal of Forest Research*, **48**(10), 1204–1216. doi: 10.1139/cjfr20180006.

See Also

normalize, gbc, apply_thr and regional_thresholding.

Other MBLT functions: find_sky_dns(), fit_cone_shaped_model(), fit_trend_surface(), ootb_mblt()

Examples

```
thr_image(gbc(125), -8, 1)
```

---

write_bin *Write binarized images*

Description

Wrapper functions for writeRaster.

Usage

```
write_bin(bin, path)
```

Arguments

bin	RasterLayer.
path	Character vector of length one. Path for writing the image.

See Also

read_bin

Other Tools functions: apply_thr(), extract_feature(), gbc(), masking(), normalize(), read_bin(), read_caim(), regional_thresholding(), write_caim()

Examples

```
## Not run:
z <- zenith_image(1000, lens())
m <- !is.na(z)
write_bin(m, "mask")
m_from_disk <- read_bin("mask.tif")
plot(m - m_from_disk)

## End(Not run)
```

write_caim — *Write canopy image*

Description

Wrapper function for writeRaster.

Usage

```
write_caim(caim, path, bit_depth)
```

Arguments

caim	Raster.
path	Character vector of length one. Path for writing the image.
bit_depth	Numeric vector of length one.

See Also

write_bin

Other Tools functions: apply_thr(), extract_feature(), gbc(), masking(), normalize(), read_bin(), read_caim(), regional_thresholding(), write_bin()

Examples

```
## Not run:
require(magrittr)
caim <- read_caim() %>% normalize(., 0, 255)
write_caim(caim * 2^8, "test_8bit", 8)
write_caim(caim * 2^16, "test_16bit", 16)

## End(Not run)
```

zenith_image — *Zenith image*

Description

Built a single layer image with zenith angles values.

Usage

```
zenith_image(diameter, lens_coef)
```

Arguments

diameter	Numeric vector of length one. Diameter in pixels.
lens_coef	Numeric vector. Polynomial coefficients of the lens projection function.

Value

RasterLayer.

See Also

Other Lens functions: azimuth_image(), calc_diameter(), calc_zenith_raster_coordinates(), calibrate_lens(), expand_noncircular(), lens(), reproject_to_equidistant(), test_lens_coef()

Examples

```
z <- zenith_image(1490, lens("Nikon_FCE9"))
plot(z)
```

Index