Package 'rcaiman'

July 14, 2022

Type Package

Title An R Package for CAnopy IMage ANalysis

Version 1.0.1.9000

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

LazyData true

RoxygenNote 7.2.0

Depends filenamer, magrittr, colorspace, terra

Imports methods, testthat, pracma, stats, utils, Rdpack, spatial, lidR, sf

Suggests autothresholdr, conicfit, EBImage, bbmle

RdMacros Rdpack

R topics documented:

`apply_thr` *Apply threshold*

Description

Global or local thresholding of images.

Usage

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

Arguments

`r`	SpatRaster. Image to binarize.
`thr`	Numeric vector of length one or SpatRaster. Threshold.

Details

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

Value

An object of class SpatRaster with values `0` and `1`.

See Also

Other Binarization Functions: `find_sky_pixels_nonnull()`, `find_sky_pixels()`, `obia()`, `ootb_mblt()`, `regional_thresholding()`, `thr_image()`

Examples

```
r <- read_caim()
apply_thr(r$Blue, 120)
## Not run:
# This function is useful in combination with the 'autothresholdr'
# package. For example:
require(autothresholdr)
thr <- auto_thresh(r$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	SpatRaster built with zenith_image.

Value

An object of class SpatRaster with azimuth angles in degrees. North (0º) is pointing up as in maps, but East (90º) and West (270º) are flipped respecting to maps. To understand why is that, take two flash-card size pieces of paper. Put one on a table in front of you and draw on it a compass rose. Take the other and hold it with your arms extended over your head, and, following the directions of the compass rose in front of you, draw another compass rose in the paper side that face down. Then, put it down and compare both compass roses.

See Also

Other Lens Functions: calc_diameter(), calc_zenith_raster_coord(), calibrate_lens(), expand_noncircular(), fisheye_to_equidistant(), fisheye_to_pano(), lens(), test_lens_coef(), zenith_image()

Examples

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

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. It 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 vertex 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 vertex that is in front of the camera. Next, make four more marks with one meter of spacing and following 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`.

Value

Numeric vector of length one. The diameter is expressed in whole numbers following the standard practice.

See Also

Other Lens Functions: `azimuth_image()`, `calc_zenith_raster_coord()`, `calibrate_lens()`, `expand_noncircular()`, `fisheye_to_equidistant()`, `fisheye_to_pano()`, `lens()`, `test_lens_coef()`, `zenith_image()`

Examples

```
# Nikon D50 and Fisheye Nikkor 10.5 mm lens
calc_diameter(lens("Nikkor_10.5_mm"), 1202, 54)
```

`calc_zenith_raster_coord` *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_coord(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

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 method implemented here do not support multiple holes**.

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 file (see Examples). It is worth noting that the perimeter of the circle depicted in a circular hemispherical photograph is not necessarily the horizon.

Value

Numeric vector of length two. Raster coordinates of the zenith, assuming a lens facing up with its optical axis parallel to the vertical line. It is important to note the difference between the raster coordinates and the Cartesian coordinates. In the latter, the vertical axis value decreases downward, but the opposite is true for the raster coordinates, which works like a spreadsheet.

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()`, `fisheye_to_equidistant()`, `fisheye_to_pano()`, `lens()`, `test_lens_coef()`, `zenith_image()`

Examples

```
## Not run:
path <- system.file("external/points_over_perimeter.csv",
                    package = "rcaiman")
calc_zenith_raster_coord(path)

## End(Not run)
```

`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 of width about two times larger than the yoga mat width
- 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, other materials could be used to obtain the same result, such as cardboard, foam, nails, etc.

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. To take care of the line of pins at 90º relative to the optical axis it is better to use the naked eye to alight the front of the lens with the pins.

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 push pin and not skipping any showed push pin.

This method was inspired by the calibration board from Clark and Follin (1988).

TIP: use `test_lens_coef` to test if coefficient are OK. If not, try moving the last points a little bit. Putting the last one a few pixels farther from the zenith is usually enough.

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.

See Also

Other Lens Functions: `azimuth_image()`, `calc_diameter()`, `calc_zenith_raster_coord()`, `expand_noncircular()`, `fisheye_to_equidistant()`, `fisheye_to_pano()`, `lens()`, `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$max_thera_px
test_lens_coef(calibration$lens_coef)
```

---

cie_sky_model_raster *CIE sky model raster*

Description

CIE sky model raster

Usage

```
cie_sky_model_raster(z, a, sun_coord, sky_coef)
```

Arguments

z	SpatRaster built with `zenith_image`.
a	SpatRaster built with `azimuth_image`.
sun_coord	Numeric vector of length two. Zenith and azimuth angles in degrees, corresponding to the location of the solar disk center.
sky_coef	Numeric vector of length five. Parameters of the sky model.

See Also

Other Sky Reconstruction Functions: `fit_cie_sky_model()`, `fit_coneshaped_model()`, `fit_trend_surface()`, `fix_reconstructed_sky()`, `interpolate_sky_points()`

Examples

```
## Not run:
z <- zenith_image(1400, lens())
a <- azimuth_image(z)
path <- system.file("external", package = "rcaiman")
skies <- read.csv(file.path(path, "15_CIE_standard_skies.csv"))
# parameters are from http://dx.doi.org/10.1016/j.energy.2016.02.054
sky_coef <- skies[4,1:5]
sun_coord <- c(45, 0)
plot(cie_sky_model_raster(z, a, sun_coord, sky_coef))

## End(Not run)
```

`defuzzify` *Defuzzify fuzzy classification*

Description

This function translates degree of membership into Boolean logic using a regional approach. The result will ensure that the fuzzy and Boolean version will agree at the chosen level of aggregation (controlled by the argument `segmentation`). This method makes perfect sense to translate a subpixel classification of gap fraction –or a linear ratio (Lang et al. 2013)– into a binary product.

Usage

```
defuzzify(mem, segmentation)
```

Arguments

`mem`	An object of the class SpatRaster. Degree of membership.
`segmentation`	An object of the class SpatRaster, such as the result of a call to `sky_grid_segmentation`.

Value

An object of the class SpatRaster containing binary information.

References

Lang M, Kodar A, Arumäe T (2013). "Restoration of above canopy reference hemispherical image from below canopy measurements for plant area index estimation in forests/ Metsa võrastiku läbipaistvuse mõõtmine digitaalsete poolsfäärikaamerate abil." *Forestry Studies*, **59**(1), 13–27. doi:10.2478/fsmu20130008.

See Also

Other Tools Functions: `extract_feature()`, `extract_rl()`, `extract_sky_points()`, `masking()`, `read_bin()`, `read_caim()`, `write_bin()`, `write_caim()`

Examples

```
## Not run:
path <- system.file("external/DSCN4500.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)
r <- gbc(caim$Blue)
r[is.na(z)] <- 0 # because FOV > 180
bin <- ootb_mblt(r, z, a)
plot(bin$bin)
ratio <- r / bin$sky_s
ratio <- normalize(ratio, 0, 1, TRUE)
plot(ratio)
g <- sky_grid_segmentation(z, a, 10)
bin2 <- defuzzify(ratio, g)
plot(bin2)
plot(bin$bin - bin2)

## End(Not run)
```

`enhance_caim` *Enhance canopy image*

Description

This function was proposed in Díaz and Lencinas (2015). It uses the color perceptual attributes (hue, lightness, and chroma) to enhance the contrast between the sky and plants through fuzzy classification. 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,
  sky_blue,
  w_red = 0,
  thr = 0.5,
  fuzziness = 10,
  gamma = NULL
)
```

Arguments

`caim`	SpatRaster. The return of a call to `read_caim`.
`m`	SpatRaster. A mask. Usually, built with `mask_hs`.
`sky_blue`	color. Is the `target_color` argument to be passed to `membership_to_color`.

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.
thr	Numeric vector of length one. Location parameter of the logistic membership function. Use NULL 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 to estimate it automatically as the midpoint between the maximum and minimum values of lightness.
gamma	Numeric vector of length one. This is for applying a gamma back correction to the lightness information (see Details and argument w_red).

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 values above 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.

The gamma argument, along with gbc, is used to back-correct the values passed to local_fuzzy_thresholding.

If you use this function in your research, please cite Díaz and Lencinas (2015) in addition to this package.

Value

An object of class SpatRaster –with same pixel dimensions than caim– that should show more contrast between the sky and plant pixels than any of the individual bands from caim.

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 Pre-processing Functions: gbc(), local_fuzzy_thresholding(), membership_to_color(), normalize()

Examples

```
## Not run:
caim <- read_caim()
caim <- normalize(caim, 0, 255)
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
sky_blue_sample <- crop(caim, ext(610,643,760,806))
sky_blue <- apply(sky_blue_sample[], 2, median) %>%
  as.numeric() %>%
  matrix(., ncol = 3) %>%
  sRGB()
ecaim <- enhance_caim(caim, !is.na(z), sky_blue, gamma = 2.2)
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	SpatRaster. The return of a call to read_caim.
z	SpatRaster built with zenith_image.
zenith_colrow	Numeric vector of length two. Raster coordinates of the zenith. See calc_zenith_raster_coord.

Value

An object of class SpatRaster that is the result of adding margins (NA pixels) to caim. The zenith point depicted in the picture should be in the center of the image or very close to it.

See Also

Other Lens Functions: azimuth_image(), calc_diameter(), calc_zenith_raster_coord(), calibrate_lens(), fisheye_to_equidistant(), fisheye_to_pano(), lens(), test_lens_coef(), zenith_image()

Examples

```
## Not run:
   my_file <- file.path(tempdir(), "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)
   plot(is.na(r$Red), add = TRUE, alpha = 0.5)

## End(Not run)
```

`extract_feature` *Extract feature*

Description

Extract features from raster images.

Usage

```
extract_feature(
  r,
  segmentation,
  fun = mean,
  return_raster = TRUE,
  ignore_label_0 = TRUE
)
```

Arguments

`r`	SpatRaster. Single layer raster.
`segmentation`	SpatRaster. The segmentation of `r`.
`fun`	A `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, see details.
`ignore_label_0`	Logical vector of length one. If this is `TRUE`, then the segment labeled with `0` will be ignored.

Details

Given a single layer raster, a segmentation, and a function, `extract_features` will returns a numeric vector or a SpatRaster 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.

Value

If `return_raster` is set to `TRUE`, then an object of class SpatRaster with the same pixel dimensions than `r` will be returned. Otherwise, the return is a numeric vector of length equal to the number of segments found in `segmentation`.

See Also

Other Tools Functions: `defuzzify()`, `extract_rl()`, `extract_sky_points()`, `masking()`, `read_bin()`, `read_caim()`, `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)
print(extract_feature(r$Blue, g, return_raster = FALSE))
plot(extract_feature(r$Blue, g, return_raster = TRUE))

## End(Not run)
```

`extract_rl` *Extract relative luminance*

Description

Extract the luminance relative to the zenith digital number.

Usage

```
extract_rl(
  r,
  z,
  a,
  sky_points,
  no_of_points = 20,
  z_thr = 2,
  use_window = TRUE
)
```

Arguments

`r`	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from an hemispherical photograph. Please see `read_caim` and `normalize`.
`z`	SpatRaster built with `zenith_image`.
`a`	SpatRaster built with `azimuth_image`.
`sky_points`	An object of class *data.frame*. The result of a call to `extract_sky_points`.
`no_of_points`	Numeric vector on length one. The number of near-zenith points required for the estimation of the zenith DN.
`z_thr`	Numeric vector on length one. The starting maximum zenith angle used to search for near zenith points.
`use_window`	Logical vector of length one. If `TRUE`, a 3×3 window will be used to extract the sky digital number from `r`.

Details

To estimate the zenith digital number (zenith DN), the search for near-zenith points starts in the region region ranged between `0` and `z_thr`. If the number of near zenith points is less than `no_of_points`, the region increase by `2` degrees of zenith angle till the required number of points is reached

Value

A list of three objects, *zenith_dn* and *max_zenith_angle* from the class numeric, and *sky_points* from the class *data.frame*; *zenith_dn* is the estimated zenith digital number, *max_zenith_angle* is the maximum zenith angle reached in the search for near-zenith sky points, and *sky_points* is the input argument `sky_points` with the additional columns: *a*, *z*, *dn*, and *rl*, which stand for azimuth and zenith angle in degrees, digital number, and relative luminance, respectively. If `NULL` is provided as `no_of_points`, then *zenith_dn* is forced to one and *dn*, and *rl* are equals.

See Also

Other Tools Functions: `defuzzify()`, `extract_feature()`, `extract_sky_points()`, `masking()`, `read_bin()`, `read_caim()`, `write_bin()`, `write_caim()`

Examples

```
## Not run:
path <- system.file("external/DSCN4500.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)
r <- gbc(caim$Blue)
g <- sky_grid_segmentation(z, a, 10)
bin <- find_sky_pixels(r, z, a)
sky_points <- extract_sky_points(r, bin, g)
rl <- extract_rl(r, z, a, sky_points, 1)

## End(Not run)
```

`extract_sky_points` *Extract sky points*

Description

Extract sky points for model fitting.

Usage

```
extract_sky_points(r, bin, g, dist_to_plant = 3, min_raster_dist = 3)
```

Arguments

`r`	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from an hemispherical photograph. Please see `read_caim` and `normalize`.
`bin`	SpatRaster. This should be a preliminary binarization of `r` useful for masking pixels that are very likely pure sky pixels.
`g`	SpatRaster built with `sky_grid_segmentation`.
`dist_to_plant, min_raster_dist`	Numeric vector of length one or `NULL`.

Details

This function will automatically sample sky pixels from the sky region delimited by `bin`. The density and distribution of the sampling points is controlled by the arguments `g`, `dist_to_plant`, and `min_raster_dist`.

As first step, digital numbers from `r` covered by pixels values equal to one on the `bin` layer– are evaluated to extract its maximum value per cell of `g`. The argument `dist_to_plant` allows users to establish a buffer zone for `bin`.

The final step filters these local maximum using the `min_raster_dist` argument as a minimum distance threshold between points that is applied in the raster space.

Using code `NULL` as argument skip the filtering step in question.

Value

An object of the class *data.frame* with two columns named *col* and *row*.

See Also

Other Tools Functions: `defuzzify()`, `extract_feature()`, `extract_rl()`, `masking()`, `read_bin()`, `read_caim()`, `write_bin()`, `write_caim()`

Examples

```
## Not run:
caim <- read_caim() %>% normalize()
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
a <- azimuth_image(z)
sky_blue_sample <- crop(caim, ext(610,643,760,806))
sky_blue <- apply(sky_blue_sample[], 2, median) %>%
  as.numeric() %>%
  matrix(., ncol = 3) %>%
  sRGB()
ecaim <- enhance_caim(caim, !is.na(z), sky_blue, gamma = 2.2)
bin <- apply_thr(ecaim, 0.75)
g <- sky_grid_segmentation(z, a, 10)
r <- gbc(caim$Blue*255)
sky_points <- extract_sky_points(r, bin, g)
cells <- cellFromRowCol(z, sky_points$row, sky_points$col)
hist(r[cells][,1])
xy <- xyFromCell(z, cells)
plot(r)
plot(vect(xy), add = TRUE, col = 2)

## End(Not run)
```

`extract_sun_coord` *Extract sun coordinates*

Description

Extract the sun coordinates for CIE sky model fitting.

Usage

```
extract_sun_coord(r, z, a, bin, g, max_angular_dist = 30)
```

Arguments

r	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from an hemispherical photograph. Please see read_caim and normalize.
z	SpatRaster built with zenith_image.
a	SpatRaster built with azimuth_image.
bin	SpatRaster. This should be a preliminary binarization of r useful for masking pixels that are very likely pure sky pixels.
g	SpatRaster built with sky_grid_segmentation.
max_angular_dist	Numeric vector of length one. Angle in degree to establish the maximum size of the sun corona.

Details

This function uses an object-based image analyze theoretical framework. The segmentation are given by g and bin. For every cell of g, the maximum value is calculated from the pixel on r that are equal to one on bin. Then, the 95th percentile of these maximum values is computed and it is used to filter out cells below that threshold; i.e, only the ones with at least one extremely bright sky pixel is keep.

After grouping the selected segments based on adjacency, the degree of membership to the class *Sun* is calculated for every segment by using linear membership functions for the digital number from r and number of cells that constitute the segment. In other words, the brighteners and lagers segments are the ones that score higher. The one with the highest score is selected as the *sun seed*.

The angular distance from the sun seed to every other segments are computed, and only the segments not farther than max_angular_dist are classified as part of the sun corona. A multi-part segment is created by merging the sun-corona segments and, finally, the center of its bounding box is considered as the sun location.

The bin argument should be the same than for extract_sky_points.

Value

Object of class list with two elements named *row_col* and *zenith_azimuth*, both are numeric vectors of length two. The former is raster coordinates of the solar disk (row and column), and the other is angular coordinates (zenith and azimuth angles in degrees).

See Also

fit_cie_sky_model

Other HSP Functions: read_manual_input(), read_opt_sky_coef(), row_col_from_zenith_azimuth(), write_sky_points(), write_sun_coord(), zenith_azimuth_from_row_col()

Examples

```
## Not run:
caim <- read_caim() %>% normalize()
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
a <- azimuth_image(z)
```

```
sky_blue_sample <- crop(caim, ext(610,643,760,806))
sky_blue <- apply(sky_blue_sample[], 2, median) %>%
  as.numeric() %>%
  matrix(., ncol = 3) %>%
  sRGB()
ecaim <- enhance_caim(caim, !is.na(z), sky_blue, gamma = 2.2)
bin <- apply_thr(ecaim, 0.75)
g <- sky_grid_segmentation(z, a, 10)
r <- gbc(caim$Blue*255)
sun_coord <- extract_sun_coord(r, z, a, bin, g, max_angular_dist = 30)
xy <- cellFromRowCol(z, sun_coord$row_col[1], sun_coord$row_col[2]) %>%
  xyFromCell(z, .)
plot(r)
plot(vect(xy), add = TRUE, col = 2)

## End(Not run)
```

`find_sky_pixels` *Find sky pixels*

Description

Find sky pixels automatically.

Usage

```
find_sky_pixels(r, z, a, sample_size_pct = 30)
```

Arguments

`r`	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from an hemispherical photograph. Please see `read_caim` and `normalize`.
`z`	SpatRaster built with `zenith_image`.
`a`	SpatRaster built with `azimuth_image`.
`sample_size_pct`	Numeric vector of length one. Minimum sample size percentage required. The population is comprised of 1296 cells of 5×5 degrees.

Details

This function assumes that:

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

For each 30×30 cell, this method computes a quantile value and use it as a threshold to select the pure sky pixels from the given cell. As a result, a binarized image is produced 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 in how many of these cells are at least one sky pixel (pixels equal to one in the binarized image). If the percentage of cells with sky pixels does not reach argument `sample_size_pct`, it goes back to the binarization step but decreasing the probability by 0.01 points.

If probability reach 0.9 and the `sample_size_pct` criterion were not yet satisfied, the `sample_size_pct` is decreased one percent and the process start all over again.

Value

An object of class SpatRaster with values `0` and `1`. This layer masks pixels that are very likely pure sky pixels.

See Also

Other Binarization Functions: `apply_thr()`, `find_sky_pixels_nonnull()`, `obia()`, `ootb_mblt()`, `regional_thresholding()`, `thr_image()`

Examples

```
## Not run:
path <- system.file("external/DSCN4500.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)
r <- gbc(caim$Blue)
bin <- find_sky_pixels(r, z, a)
plot(bin)

## End(Not run)
```

---

find_sky_pixels_nonnull

Find sky pixels following the non-null criteria

Description

Find sky pixels by using the increase in the number of cells having no sky pixels (the so-called null cells) as an stopping criteria.

Usage

```
find_sky_pixels_nonnull(r, sky, g, slope = 0.5)
```

Arguments

`r`	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from an hemispherical photograph. Please see `read_caim` and `normalize`.
`sky`	An object of class SpatRaster produced with `fit_coneshaped_model`, `fit_trend_surface`, `fit_cie_sky_model`, or `ootb_sky_reconstruction`.
`g`	SpatRaster built with `sky_grid_segmentation`.
`slope`	Numeric vector of length one. Please, see the Details section of `thr_image`.

Details

The arguments `sky` and `slope` is passed to `thr_image`, which result is in turn passed to `apply_thr` along with `r`. As a result, `r` is binarized and used along with g, to compute the number of null cells. The process is repeated but increasing `slope` in steps of 0.05 as long as the number of null cells remains constant.

Value

An object of class SpatRaster with values 0 and 1.

See Also

Other Binarization Functions: apply_thr(), find_sky_pixels(), obia(), ootb_mblt(), regional_thresholding() thr_image()

Examples

```
## Not run:
path <- system.file("external/DSCN4500.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)
r <- gbc(caim$Blue)
bin <- find_sky_pixels(r, z, a)
g <- sky_grid_segmentation(z, a, 10)
sky_points <- extract_sky_points(r, bin, g)
sky_points <- extract_rl(r, z, a, sky_points, NULL)
model <- fit_coneshaped_model(sky_points$sky_points)
sky_cs <- model$fun(z, a)
g[mask_hs(z, 0, 10) | mask_hs(z, 70, 90)] <- NA
bin <- find_sky_pixels_nonnull(r, sky_cs, g)
plot(bin)

## End(Not run)
```

---

fisheye_to_equidistant

Fisheye to equidistant

Description

Fisheye to equidistant projection (also known as polar projection).

Usage

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

Arguments

r	SpatRaster.
z	SpatRaster built with zenith_image.
a	SpatRaster built with azimuth_image.
radius	Numeric integer of length one. Radius of the reprojected hemispherical image (i.e., the output).

See Also

Other Lens Functions: azimuth_image(), calc_diameter(), calc_zenith_raster_coord(), calibrate_lens(), expand_noncircular(), fisheye_to_pano(), 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 <- fisheye_to_equidistant(bin, z, a, radius = 400)
bin_equi <- apply_thr(bin_equi, 0.5)
plot(bin_equi)
# use write_bin(bin, "path\file_name") to have a file ready
# for calculating LAI with CIMES, GLA, CAN-EYE, etc.

## End(Not run)
```

---

fisheye_to_pano *Fisheye to panoramic*

Description

Fisheye to panoramic (cylindrical projection)

Usage

```
fisheye_to_pano(r, z, a, fun = mean, angle_width = 1)
```

Arguments

r	SpatRaster.
z	SpatRaster built with zenith_image.
a	SpatRaster built with azimuth_image.
fun	A function that takes a vector as input and returns a one-length numeric or logical vector as output (e.g. mean).
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.

See Also

Other Lens Functions: azimuth_image(), calc_diameter(), calc_zenith_raster_coord(), calibrate_lens(), expand_noncircular(), fisheye_to_equidistant(), lens(), test_lens_coef(), zenith_image()

Examples

```
## Not run:
caim <- read_caim()
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
a <- azimuth_image(z)
pano <- fisheye_to_pano(caim, z, a, radius = 400)
```

```
plotRGB(pano)

## End(Not run)
```

`fit_cie_sky_model` *Fit CIE sky model*

Description

Use maximum likelihood to estimate the coefficients of the CIE sky model that fit best to data sampled from a real scene.

Usage

```
fit_cie_sky_model(
  r,
  z,
  a,
  sky_points,
  zenith_dn,
  sun_coord,
  std_sky_no = NULL,
  general_sky_type = NULL,
  twilight = TRUE,
  rmse = FALSE,
  method = "BFGS"
)
```

Arguments

`r`	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from an hemispherical photograph. Please see `read_caim` and `normalize`.
`z`	SpatRaster built with `zenith_image`.
`a`	SpatRaster built with `azimuth_image`.
`sky_points`	The *data.frame* returned by `extract_rl`, or a *data.frame* with the same structure and names.
`zenith_dn`	Numeric vector of length 1. Zenith digital number, see `extract_rl` for how to obtain it.
`sun_coord`	An object of class *list*. The result of a call to `extract_sun_coord`.
`std_sky_no`	Numeric vector. Standard sky number from Table 1 from Li et al. (2016).
`general_sky_type`	Character vector of length one. It could be any of these: "Overcast", "Clear", or "Partly cloudy". See Table 1 from Li et al. (2016) for additional details.
`twilight`	Logical vector of length one. If it is `TRUE` and the initial standard parameters belong to the "Clear" general sky type, sun zenith angles from 90 to 96 degrees will be tested (civic twilight). This is necessary since `extract_sun_coord` would mistakenly recognize the center of what can be seen of the solar corona as the solar disk.

rmse	Logical vector of length one. If it is TRUE, the criteria for selecting the best sky model is to choose the one with lest root mean square error calculated from the sample (sky marks). Otherwise, the criteria is to evaluate the whole hemisphere by calculating the product between the square ratio of r to the sky model and the fraction of pixels from this new layer that are above one or below zero, and selecting the sky model that produce the least value.
method	Optimization method to use. See optim.

Details

This function assumes an hemispherical image as input. It is based on Lang et al. (2010). In theory, the better result would be obtained with data showing a linear relation between digital numbers and the amount of light reaching the sensor. However, because the CIE sky model is indeed the adjoin of two mathematical model, it is capable of cope with any non-linearity since it is not a physical model with strict assumptions.

Ultimately, if the goal is to calculate the ratio of canopy to sky digital numbers, if the latter is accurately constructed, any non-linearity will be canceled. Please, see interpolate_sky_points for further considerations.

Value

The result include the output produced by mle2, the 5 model coefficients, observed and predicted values, the sun coordinates (zenith and azimuth angle in degrees), the relative luminance calculated for every pixel using the estimated coefficients and corresponding sun coordinate, the digital number at the zenith, and the description of the standard sky from which the initial coefficients were drawn. See Li et al. (2016) to known more about these coefficients.

References

Lang M, Kuusk A, Mõttus M, Rautiainen M, Nilson T (2010). "Canopy gap fraction estimation from digital hemispherical images using sky radiance models and a linear conversion method." *Agricultural and Forest Meteorology*, **150**(1), 20–29. doi:10.1016/j.agrformet.2009.08.001.

Li DH, Lou S, Lam JC, Wu RH (2016). "Determining solar irradiance on inclined planes from classified CIE (International Commission on Illumination) standard skies." *Energy*, **101**, 462–470. doi:10.1016/j.energy.2016.02.054.

See Also

Other Sky Reconstruction Functions: cie_sky_model_raster(), fit_coneshaped_model(), fit_trend_surface(), fix_reconstructed_sky(), interpolate_sky_points()

Examples

```
## Not run:
caim <- read_caim() %>% normalize()
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
a <- azimuth_image(z)
sky_blue_sample <- crop(caim, ext(610,643,760,806))
sky_blue <- apply(sky_blue_sample[], 2, median) %>%
  as.numeric() %>%
  matrix(., ncol = 3) %>%
  sRGB()
ecaim <- enhance_caim(caim, !is.na(z), sky_blue, gamma = 2.2)
```

```
bin <- apply_thr(ecaim, 0.75)
g <- sky_grid_segmentation(z, a, 10)
r <- gbc(caim$Blue*255)
sun_coord <- extract_sun_coord(r, z, a, bin, g)
sky_points <- extract_sky_points(r, bin, g)
rl <- extract_rl(r, z, a, sky_points)
model <- fit_cie_sky_model(r, z, a, rl$sky_points,
                           rl$zenith_dn, sun_coord,
                           rmse = TRUE,
                           general_sky_type = "Partly cloudy")
sky_cie <- model$relative_luminance * model$zenith_dn
sky_cie <- normalize(sky_cie, 0, 1, TRUE)
plot(sky_cie)
plot(r/sky_cie)

## End(Not run)
```

`fit_coneshaped_model` *Fit cone-shaped model*

Description

Statistical modelling to predict the digital numbers from spherical coordinates.

Usage

```
fit_coneshaped_model(sky_points, use_azimuth_angle = TRUE)
```

Arguments

`sky_points`	The *data.frame* returned by `extract_rl`, or a *data.frame* with the same structure and names.
`use_azimuth_angle`	Logical vector of length one. If `TRUE`, the 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`, the next simplified version based on Wagner (2001) is used: $sDN = a + b \cdot \theta + c \cdot \theta^2$.

Details

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

If you use this function in your research, please cite Díaz and Lencinas (2018).

Value

A list of two objects, one of class `function` and the other of class `lm` (see `lm`). If the fitting fails, it returns `NULL`. The function requires two arguments –zenith and azimuth in degrees– to return relative luminance.

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.

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.

See Also

thr_image

Other Sky Reconstruction Functions: cie_sky_model_raster(), fit_cie_sky_model(), fit_trend_surface(), fix_reconstructed_sky(), interpolate_sky_points()

Examples

```
## Not run:
path <- system.file("external/DSCN4500.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)
r <- gbc(caim$Blue)
g <- sky_grid_segmentation(z, a, 10)
bin <- find_sky_pixels(r, z, a)
sky_points <- extract_sky_points(r, bin, g)
sky_points <- extract_rl(r, z, a, sky_points, NULL)
model <- fit_coneshaped_model(sky_points$sky_points)
sky_cs <- model$fun(z, a)
persp(sky_cs, 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 surf.ls as workhorse function.

Usage

```
fit_trend_surface(r, z, a, bin, filling_source = NULL, np = 6)
```

Arguments

r	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from an hemispherical photograph. Please see read_caim and normalize.
z	SpatRaster built with zenith_image.
a	SpatRaster built with azimuth_image.
bin	SpatRaster. This should be a preliminary binarization of r useful for masking pixels that are very likely pure sky pixels.

filling_source SpatRaster. An actual or reconstructed above-canopy image to complement the sky pixels detected through the gaps of r. If an incomplete above-canopy image is available, non-sky pixels should be turned NA or they will be erroneously considered as sky 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 but not recommended under fleeting clouds. The orientation relative to the North must be the same than for r. If it is set to NULL (default), only sky pixels from r will be used as input.

np degree of polynomial surface

Details

This function is meant to be used after fit_coneshaped_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_coneshaped_model.

If you use this function in your research, please cite Díaz and Lencinas (2018).

Value

A list with an object of class SpatRaster 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

thr_image

Other Sky Reconstruction Functions: cie_sky_model_raster(), fit_cie_sky_model(), fit_coneshaped_model(), fix_reconstructed_sky(), interpolate_sky_points()

Examples

```
## Not run:
path <- system.file("external/DSCN4500.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)
r <- gbc(caim$Blue)
g <- sky_grid_segmentation(z, a, 10)
bin <- find_sky_pixels(r, z, a)
sky_points <- extract_sky_points(r, bin, g)
sky_points <- extract_rl(r, z, a, sky_points, NULL)
model <- fit_coneshaped_model(sky_points$sky_points)
sky_cs <- model$fun(z, a)
m <- mask_hs(z, 0, 80)
sky <- fit_trend_surface(r, z, a, bin, filling_source = sky_cs)
plot(sky$image)

## End(Not run)
```

`fix_reconstructed_sky` *Fix predicted sky*

Description

Automatically edit a raster image of sky digital numbers (DNs) predicted with functions such as `fit_coneshaped_model` and `fit_trend_surface`.

Usage

```
fix_reconstructed_sky(sky, z, r, bin)
```

Arguments

`sky`	SpatRaster. Sky DNs predicted with functions such as `fit_coneshaped_model` and `fit_trend_surface`.
`z`	SpatRaster built with `zenith_image`.
`r`	SpatRaster. The source of the sky DNs used to build `sky`.
`bin`	SpatRaster. The binarization of `r` used to select the sky DNs for building `sky`.

Details

The predicted sky DNs are usually erroneous near the horizon because either they are a misleading extrapolation or are based on corrupted data (non-pure sky DNs).

This automatic edition consists of (1) flattening values below the minimum input data (2) and forcing the values toward the horizon to gradually become the median input data. The latter is achieved by calculating the weighted average of the median value and the predicted sky DNs –the ratio of `z` to `90` is used to determine the weights.

Value

An object of class SpatRaster. The argument `sky` with dimensions unchanged but values edited.

See Also

Other Sky Reconstruction Functions: `cie_sky_model_raster()`, `fit_cie_sky_model()`, `fit_coneshaped_model()`, `fit_trend_surface()`, `interpolate_sky_points()`

Examples

```
## Not run:
path <- system.file("external/DSCN4500.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)
r <- gbc(caim$Blue)
g <- sky_grid_segmentation(z, a, 10)
bin <- find_sky_pixels(r, z, a)
sky_points <- extract_sky_points(r, bin, g)
sky_points <- extract_rl(r, z, a, sky_points, NULL)
model <- fit_coneshaped_model(sky_points$sky_points)
```

```
sky_cs <- model$fun(z, a)
sky_cs <- fix_reconstructed_sky(sky_cs, z, r, bin)
persp(sky_cs, theta = 90, phi = 0)

## 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 SpatRaster class. Digital numbers from a JPEG file (0 to 255, the standard 8-bit encoded).
gamma	Numeric vector of length one. Gamma value. Please see Díaz and Lencinas (2018) for details.

Value

The same class than DN_from_JPEG, with dimension unchanged but values rescaled between 0 and 1 in a non linear fashion.

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 Pre-processing Functions: enhance_caim(), local_fuzzy_thresholding(), membership_to_color(), normalize()

Examples

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

`interpolate_sky_points` *Interpolate sky points*

Description

Interpolate values from hemispherical photographs.

Usage

```
interpolate_sky_points(sky_points, g, k = 3, p = 2, rmax = 200)
```

Arguments

`sky_points`	The *data.frame* returned by `extract_rl`, or a *data.frame* with the same structure and names.
`g`	SpatRaster built with `sky_grid_segmentation`.
`k`	integer. Number of k-nearest neighbours. Default 10.
`p`	numeric. Power for inverse-distance weighting. Default 2.
`rmax`	numeric. Maximum radius where to search for knn. Default 50.

Details

This function use `knnidw` as workhorse function, so arguments `k`, `p`, and `rmax` are passed to it.

This method is based on Lang et al. (2010). In theory, interpolation requires a linear relation between DNs and the amount of light reaching the sensor. To that end, photographs should be taken in RAW format to avoid gamma correction (Lang et al. 2010). As a compromise solution, `gbc` can be used.

The vignetting effect also hindered linear relation between DNs and the amount of light reaching the sensor. Please refer to Lang et al. (2010) for more details about the vignetting effect.

The use of `k = 1` solve the linear dilemma from the theoretical point of view since no averaging is taking place in the calculations. However, probably is best to use `k` greater than 1.

Default parameters are the used by Lang et al. (2010). The argument `rmax` should account for between 15 to 20 degrees, but is expressed in pixels units. So, image resolution and lens projections should be taken into account to properly set this argument.

The argument `g` should be the same used to obtain `sky_points`. The result will be limited to the cells with at least one pixel covered by the convex hull of the sky points.

Value

An object of class SpatRaster.

References

Lang M, Kuusk A, Mõttus M, Rautiainen M, Nilson T (2010). "Canopy gap fraction estimation from digital hemispherical images using sky radiance models and a linear conversion method." *Agricultural and Forest Meteorology*, **150**(1), 20–29. doi:10.1016/j.agrformet.2009.08.001.

See Also

Other Sky Reconstruction Functions: cie_sky_model_raster(), fit_cie_sky_model(), fit_coneshaped_model(), fit_trend_surface(), fix_reconstructed_sky()

Examples

```
## Not run:
caim <- read_caim() %>% normalize()
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
a <- azimuth_image(z)
sky_blue_sample <- crop(caim, ext(610,643,760,806))
sky_blue <- apply(sky_blue_sample[], 2, median) %>%
  as.numeric() %>%
  matrix(., ncol = 3) %>%
  sRGB()
ecaim <- enhance_caim(caim, !is.na(z), sky_blue, gamma = 2.2)
bin <- apply_thr(ecaim, 0.75)

g <- sky_grid_segmentation(z, a, 10)
r <- gbc(caim$Blue*255)
sky_points <- extract_sky_points(r, bin, g)
sky_points <- extract_rl(r, z, a, sky_points, NULL)
sky <- interpolate_sky_points(sky_points$sky_points, g)
plot(sky)

## End(Not run)
```

---

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.
max_fov	Logical vector of length one. 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)

Value

If `max_fov` is set to `TRUE`, it returns a numeric vector of length one, which is the lens maximum field of view in degrees. Otherwise, it returns a numeric vector with the coefficient of the lens function.

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.

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_coord()`, `calibrate_lens()`, `expand_noncircular()`, `fisheye_to_equidistant()`, `fisheye_to_pano()`, `test_lens_coef()`, `zenith_image()`

Examples

```
lens("Nikon_FCE9")
lens("Nikon_FCE9", max_fov = TRUE)
```

---

local_fuzzy_thresholding

local fuzzy thresholding

Description

This function was 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`	SpatRaster. A normalized greyscale image (see `normalize`).
`m`	SpatRaster. A mask. Usually, built with `mask_hs`.
`mem`	SpatRaster. 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 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 to estimate it automatically as the midpoint between the maximum and minimum values of lightness.

Details

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

If you use this function in your research, please cite Díaz and Lencinas (2015).

Value

An object of class SpatRaster with same pixel dimensions than caim. Depending on mem, changes could be subtle. However, they should be in the direction of showing more contrast between the sky and plant pixels than any of the individual bands from caim.

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 Pre-processing Functions: enhance_caim(), gbc(), membership_to_color(), normalize()

Examples

```
## Not run:
caim <- read_caim()
caim <- normalize(caim, 0, 255)
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
target_color <- 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))
```

Arguments

r	SpatRaster. The image. Values should be normalized, see normalize. Only methods for images with one or three layers have been implemented.
m	SpatRaster. The mask, see mask_hs.
RGB	Numeric vector of length three. RGB color code. Red is the default color.

Value

An object of class SpatRaster that essentially is r with the areas delimited by m painted in a solid color. If r is a single layer image, then the layer is triplicated to allow the use of color.

See Also

mask_hs

Other Tools Functions: defuzzify(), extract_feature(), extract_rl(), extract_sky_points(), read_bin(), read_caim(), write_bin(), write_caim()

Examples

```
## Not run:
 r <- read_caim()
 z <- zenith_image(ncol(r), lens())
 a <- azimuth_image(z)
 m <- mask_hs(z, 20, 70) & mask_hs(a, 90, 180)
 m <- as.logical(m)

 masked_caim <-  masking(normalize(r, 0, 255), m)
 plotRGB(masked_caim * 255)

 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	SpatRaster built with zenith_image or azimuth_image.
from, to	angle in degrees, inclusive limits.

Value

An object of class SpatRaster with values 0 and 1.

See Also

masking

Other Segmentation Functions: mask_sunlit_canopy(), polar_qtree(), 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)
```

---

mask_sunlit_canopy *Mask sunlit canopy*

Description

Mask sunlit canopy

Usage

```
mask_sunlit_canopy(caim, m)
```

Arguments

caim	SpatRaster. The return of a call to read_caim.
m	SpatRaster. A mask. Usually, built with mask_hs.

Value

An object of class SpatRaster with values 0 and 1. It masks pixels that are very likely sunlit canopy.

See Also

Other Segmentation Functions: mask_hs(), polar_qtree(), rings_segmentation(), sectors_segmentation(), sky_grid_segmentation()

Examples

```
caim <- read_caim() %>% normalize()
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
m <- !is.na(z)
sunlit_canopy <- mask_sunlit_canopy(caim, m)
plot(sunlit_canopy)
```

---

membership_to_color *Compute the membership to a target color*

Description

This function was 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. To be clear, the lightness dimension is not considered in the calculations.

Usage

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

Arguments

caim	SpatRaster. 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.

Details

If you use this function in your research, please cite Díaz and Lencinas (2015).

Value

It returns an object from the class SpatRaster. 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 Pre-processing Functions: enhance_caim(), gbc(), local_fuzzy_thresholding(), normalize()

Examples

```
## Not run:
caim <- read_caim()
caim <- normalize(caim, 0, 255)
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
target_color <- sRGB(matrix(c(0.25, 0.75, 0), ncol = 3))
mem <- membership_to_color(caim, target_color)
plot(mem)

## End(Not run)
```

`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`.This function can be used for linear stretching of the histogram.

Usage

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

Arguments

`r`	`SpatRaster` or numeric vector.
`mn`	Numeric vector of length one. Minimum expected value. Defaults is equivalent to the minimum value from `r`.
`mx`	Numeric vector of length one. Maximum expected value. Defaults is equivalent to the maximum value from `r`.
`force_range`	Logical vector of length one. If it is `TRUE`, the range is forced to be between zero and one by flattening below and above those limits.

Value

An object from the same class than `r` with values from `r` linearly rescaled to make `mn` equal to zero and `mx` equal to one. Therefore, if `mn` and `mx` do not match with the actual minimum and maximum from `r`, the output will not cover the 0-to-1 range, and it may be outside that range if `force_range` is set to `FALSE`.

See Also

Other Pre-processing Functions: `enhance_caim()`, `gbc()`, `local_fuzzy_thresholding()`, `membership_to_color()`

Examples

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

obia *Object-based image analysis of hemispherical photographs*

Description

This method was first presented in Díaz and Lencinas (2015). This version is simpler since it relies on a better working binarized image. The version from 2015 uses an automatic selection of samples followed by a knn classification of segments containing foliage. This version uses de gap fraction extracted from `bin` to classify *foliage* by defining upper and lower limits through the arguments `gf_mx` and `gf_mn`.

Usage

```
obia(r, z, a, bin, segmentation, gf_mn = 0.2, gf_mx = 0.95)
```

Arguments

`r`	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from an hemispherical photograph. Please see `read_caim` and `normalize`.
`z`	SpatRaster built with `zenith_image`.
`a`	SpatRaster built with `azimuth_image`.
`bin`	SpatRaster. This should be a preliminary binarization of `r` useful for masking pixels that are very likely pure sky pixels.
`segmentation`	SpatRaster built with `polar_qtree`.
`gf_mn, gf_mx`	Numeric vector of length one. The minimum/maximum gap fraction that a segment should have to be considered as one containing foliage.

Details

This method produce a synthetic layer by computing the ratio of `r` to the maximum value of `r` at the segment level. This process is carried out only on the pixels covered by the classes *foliage* and *sky* –`bin` equal to one. To avoid spurious values, the quantile `0.9` is computed instead of the maximum. Pixels from the synthetic layer comprised between `0` an `1` are binarized following two criteria in such a way that to be turned plant on `bin` (i.e, to be assigned as `0`) it have to be `0` under both criteria. Those criteria are `defuzzify` with a sky grid segmentation of `10` degrees and `apply_thr` with a threshold equal to `0.5`. In addition, it cannot be exclusively surrounded by sky pixels.

Value

SpatRaster. An improved version of `bin`.

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 Binarization Functions: `apply_thr()`, `find_sky_pixels_nonnull()`, `find_sky_pixels()`, `ootb_mblt()`, `regional_thresholding()`, `thr_image()`

Examples

```
## Not run:
caim <- read_caim()
caim <- normalize(caim, 0, 255)
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
a <- azimuth_image(z)
sky_blue_sample <- crop(caim, ext(610,643,760,806))
sky_blue <- apply(sky_blue_sample[], 2, median) %>%
  as.numeric() %>%
  matrix(., ncol = 3) %>%
  sRGB()
ecaim <- enhance_caim(caim, !is.na(z), sky_blue, gamma = 2.2)
bin <- apply_thr(ecaim, 0.5)
seg <- polar_qtree(caim, z, a)
bin_obia <- obia(gbc(caim$Blue), z, a, bin, seg)
plot(bin - bin_obia)

## End(Not run)
```

`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, bin = NULL)
```

Arguments

`r`	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from an hemispherical photograph. Please see `read_caim` and `normalize`.
`z`	SpatRaster built with `zenith_image`.
`a`	SpatRaster built with `azimuth_image`.
`bin`	SpatRaster. This should be a preliminary binarization of `r` useful for masking pixels that are very likely pure sky pixels.

Details

This function is a hard-coded version of a MBLT pipeline that starts producing a working binarized image and ends with a refined binarized image. The pipeline combines these main functions `find_sky_pixels` –if `bin` is `NULL`–, `fit_coneshaped_model`, `find_sky_pixels_nonnull`, and `fit_trend_surface`. 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 thresholding approach as first step instead of a global one. Please refer to find_sky_pixels.
- It does not use asynchronous acquisition under the open sky. So, the cone-shaped model (fit_coneshaped_model) run without a filling source, but the result of it is used as filling source for trend surface fitting (fit_trend_surface).
- find_sky_pixels_nonnull is used to update the first working binarized image.

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, therefore, 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 are looking for the classes *Gap* and *No-gap*, no matter if one of those are not in the picture itself. This kind of error could be find in photographs of open forests for the same working principle.

If you use this function in your research, please cite Díaz and Lencinas (2018).

Value

Object from class list containing 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

thr_image

Other Binarization Functions: apply_thr(), find_sky_pixels_nonnull(), find_sky_pixels(), obia(), regional_thresholding(), thr_image()

Examples

```
## Not run:
path <- system.file("external/DSCN4500.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)
r <- gbc(caim$Blue)
r[is.na(z)] <- 0 #because FOV > 180
bin <- ootb_mblt(r, z, a)
plot(bin$bin)

## End(Not run)
```

`ootb_sky_reconstruction` *Out-of-the-box sky reconstruction*

Description

Build an above canopy image from a single below canopy image.

Usage

```
ootb_sky_reconstruction(r, z, a, bin = NULL, filling_source = NULL)
```

Arguments

`r`	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from an hemispherical photograph. Please see `read_caim` and `normalize`.
`z`	SpatRaster built with `zenith_image`.
`a`	SpatRaster built with `azimuth_image`.
`bin`	SpatRaster. This should be a preliminary binarization of `r` useful for masking pixels that are very likely pure sky pixels.
`filling_source`	SpatRaster. An actual or reconstructed above-canopy image to complement the sky pixels detected through the gaps of `r`. If an incomplete above-canopy image is available, non-sky pixels should be turned `NA` or they will be erroneously considered as sky 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 but not recommended under fleeting clouds. The orientation relative to the North must be the same than for `r`. If it is set to `NULL` (default), only sky pixels from `r` will be used as input.

Details

This function is a hard-coded version of a pipeline that uses these main functions: `ootb_mblt`, `fit_cie_sky_model`, and `interpolate_sky_points`. The code can be easily inspected by calling `ootb_sky_reconstruction` –no parenthesis. Advanced users could use that code as a template.

This pipeline is based on these two studies: Lang et al. (2010) and Díaz and Lencinas (2018).

Details about how the original method by Díaz and Lencinas (2018) was adapted can be read here `ootb_mblt`.

The main differences between the original method by Lang et al. (2010) and the one implemented here are:

- it is fully automatic,
- the residuals of the CIE sky model ($residuals = model - data$) are interpolated instead of the sky digital numbers (the data), and
- the final sky reconstruction is obtained by subtracting the interpolated residuals to the CIE sky model instead of by calculating a weighted average parameterized by the user.

The recommended input for this function is data pre-processed with the HSP software package (Lang et al. 2013). Please, refer to `link{write_sky_points}` for additional details about HSP and refer to `fit_cie_sky_model` and `interpolate_sky_points` to know why the HSP pre-processing is convenient.

A binarized image can be provided trough the `bin` argument. In that case, `ootb_mblt` will not be used.

Providing a `filling source` trigger an alternative pipeline in which the sky is fully reconstructed with `interpolate_sky_points` after a dense sampling (1×1 degree cells) supported by the fact that sky digital numbers wil be available for every pixel, either from `r` gaps or from the filling source –an exception is a filling source with plenty of `NA` values.

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.

Lang M, Kodar A, Arumäe T (2013). "Restoration of above canopy reference hemispherical image from below canopy measurements for plant area index estimation in forests/ Metsa võrastiku läbipaistvuse mõõtmine digitaalsete poolsfäärikaamerate abil." *Forestry Studies*, **59**(1), 13–27. doi:10.2478/fsmu20130008.

Lang M, Kuusk A, Mõttus M, Rautiainen M, Nilson T (2010). "Canopy gap fraction estimation from digital hemispherical images using sky radiance models and a linear conversion method." *Agricultural and Forest Meteorology*, **150**(1), 20–29. doi:10.1016/j.agrformet.2009.08.001.

Examples

```
## Not run:
caim <- read_caim()
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
a <- azimuth_image(z)
r <- gbc(caim$Blue)
sky <- ootb_sky_reconstruction(r, z, a)
plot(sky)
ratio <- r / sky
plot(ratio)
hist(ratio)

## End(Not run)
```

---

polar_qtree *Quad-tree segmentation in a polar space*

Description

The quad-tree segmentation algorithm is a top-down process that makes recursive divisions in four equal parts until a condition is satisfied and then stops locally. The usual implementation of the quad-tree algorithm is based on the raster structure and this is why the result are squares of different sizes. This method implements the quad-tree segmentation in a polar space, so the segments are shaped like windshields (please see `sky_grid_segmentation`).

Usage

```
polar_qtree(r, z, a, scale_parameter = 0.2)
```

Arguments

r	SpatRaster.
z	SpatRaster built with zenith_image.
a	SpatRaster built with azimuth_image.
scale_parameter	Numeric vector of length one. Quad-tree is a top-down method. This parameter controls the stopping condition. Therefore, it allows the user to controls the size of the resulting segments. Ultimately, segments sizes will depend on the heterogeneity of r and this parameter.

Details

The algorithm splits segments of 30 degrees resolution into four sub-segments and calculates the standard deviation of the pixels from r delimited by each of those segments. The splitting process stops locally if the sum of the standard deviation of the sub-segments minus the standard deviation of the parent segment (named *delta*) is less or equal than the scale_parameter. If r has more than one layer, *delta* is calculated separately and *delta* mean is used to evaluate the stopping condition.

See Also

Other Segmentation Functions: mask_hs(), mask_sunlit_canopy(), rings_segmentation(), sectors_segmentation(), sky_grid_segmentation()

Examples

```
## Not run:
caim <- read_caim()
caim <- normalize(caim, 0, 255)
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
a <- azimuth_image(z)
seg <- polar_qtree(caim, z, a)
plot(seg)
tmp <- extract_feature(caim$Blue, seg)
plot(tmp)

## End(Not run)
```

---

rcaiman *rcaiman: An R package for CAnopy IMage ANalysis*

Description

Solutions for binarizing canopy images, particularly hemispherical photographs, including non-circular ones, such as the ones from Smartphone-based hemispherical photography.

Binarization

apply_thr, defuzzify, find_sky_pixels_nonnull, find_sky_pixels, obia, ootb_mblt, ootb_sky_reconstructio
regional_thresholding, and thr_image.

HSP

extract_sun_coord, read_manual_input, read_opt_sky_coef, row_col_from_zenith_azimuth, write_sky_points, write_sun_coord, and zenith_azimuth_from_row_col.

Lens

azimuth_image, calc_diameter, calc_zenith_raster_coord, calibrate_lens, expand_noncircular, fisheye_to_equidistant, fisheye_to_pano, lens, test_lens_coef, and zenith_image.

Pre-processing

enhance_caim, gbc, local_fuzzy_thresholding, membership_to_color, and normalize.

Segmentation

mask_hs, mask_sunlit_canopy, polar_qtree, rings_segmentation, sectors_segmentation, and sky_grid_segmentation.

Sky reconstruction

fit_cie_sky_model, fit_coneshaped_model, fit_trend_surface, fix_reconstructed_sky, and interpolate_sky_points.

Tools

extract_feature, extract_rl, extract_sky_points, masking, read_bin, read_caim, write_bin, and write_caim.

---

read_bin *Read binarized images*

Description

Wrapper functions for rast.

Usage

```
read_bin(path)
```

Arguments

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

Value

An object from class SpatRaster.

See Also

write_bin

Other Tools Functions: defuzzify(), extract_feature(), extract_rl(), extract_sky_points(), masking(), read_caim(), write_bin(), write_caim()

Examples

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

## End(Not run)
```

---

read_caim *Read a canopy image from a file*

Description

Wrapper function for rast.

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 if no arguments are provided.
upper_left	An integer vector of length two.
width, height	An integer vector of length one.

Details

Run read_caim() to obtain an example of a hemispherical photo taken in non-diffuse light conditions in a *Nothofagus pumilio* forest 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. Use upper_left, width, and height to read a region of the file. 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 as in objects from the class data.frame and others alike**). 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

An object from class SpatRaster with its layers named Red, Green, and Blue.

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: `defuzzify()`, `extract_feature()`, `extract_rl()`, `extract_sky_points()`, `masking()`, `read_bin()`, `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)
plotRGB(r)
```

---

read_manual_input *Read manual input*

Description

Read manual input stored in an HSP project.

Usage

```
read_manual_input(path_to_HSP_project, img_name)
```

Arguments

path_to_HSP_project	Character vector of length one. Path to the HSP project folder.
img_name	Character vector of length one. See details.

Details

Refer to the Details section of this function: write_sky_points.

Value

A list of numeric vectors named as *weight*, *max_points*, *angle*, *point_radius*, *sun_mark*, *sky_marks* and *zenith_dn*.

See Also

Other HSP Functions: extract_sun_coord(), read_opt_sky_coef(), row_col_from_zenith_azimuth(), write_sky_points(), write_sun_coord(), zenith_azimuth_from_row_col()

---

read_opt_sky_coef *Read optimized sky coefficients*

Description

Read optimized CIE sky coefficients stored in an HSP project.

Usage

```
read_opt_sky_coef(path_to_HSP_project, img_name)
```

Arguments

path_to_HSP_project	Character vector of length one. Path to the HSP project folder.
img_name	Character vector of length one. See details.

Details

Refer to the Details section of this function: write_sky_points.

Value

Numeric vector of length five.

See Also

cie_sky_model_raster

Other HSP Functions: extract_sun_coord(), read_manual_input(), row_col_from_zenith_azimuth(), write_sky_points(), write_sun_coord(), zenith_azimuth_from_row_col()

`regional_thresholding` *Regional thresholding*

Description

Regional thresholding of greyscale images

Usage

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

Arguments

`r`	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from an hemispherical photograph. Please see `read_caim` and `normalize`.
`segmentation`	SpatRaster. 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, slope`	Numeric vector of length one. These are linear function coefficients. Please, see the Details section of `thr_image`.
`prob`	Numeric vector of length one. Probability for `quantile` calculation.

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 segment-wise 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

An object of class SpatRaster with values `0` and `1`.

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.

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 Binarization Functions: `apply_thr()`, `find_sky_pixels_nonnull()`, `find_sky_pixels()`, `obia()`, `ootb_mblt()`, `thr_image()`

Examples

```
## Not run:
path <- system.file("external/DSCN4500.JPG", package = "rcaiman")
r <- read_caim(path, c(1280, 960) - 745, 745 * 2, 745 * 2)
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, 1)
plot(bin)

## 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`	SpatRaster 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

An object from the class SpatRaster with segments shaped like concentric rings.

See Also

Other Segmentation Functions: `mask_hs()`, `mask_sunlit_canopy()`, `polar_qtree()`, `sectors_segmentation()`, `sky_grid_segmentation()`

Examples

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

`row_col_from_zenith_azimuth`

Row and col numbers from zenith and azimuth angles

Description

Row and col numbers from zenith and azimuth angles

Usage

```
row_col_from_zenith_azimuth(r, za, lens_coef)
```

Arguments

`r`	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from an hemispherical photograph. Please see `read_caim` and `normalize`.
`za`	Numeric vector of length two. Zenith and azimuth angles in degrees.
`lens_coef`	Numeric vector. Polynomial coefficients of the lens projection function.

Value

Numeric vector of length two.

See Also

Other HSP Functions: `extract_sun_coord()`, `read_manual_input()`, `read_opt_sky_coef()`, `write_sky_points()`, `write_sun_coord()`, `zenith_azimuth_from_row_col()`

Examples

```
z <- zenith_image(1000, lens())
row_col <- row_col_from_zenith_azimuth(z, c(45, 270), lens())
```

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	SpatRaster 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

An object from the class SpatRaster with segments shaped like pizza slices.

See Also

Other Segmentation Functions: mask_hs(), mask_sunlit_canopy(), polar_qtree(), 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	SpatRaster built with zenith_image.
a	SpatRaster 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

An object from the class SpatRaster with segments shaped like windshields, though 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(), mask_sunlit_canopy(), polar_qtree(), rings_segmentation(), sectors_segmentation()

Examples

```
z <- zenith_image(1490, lens())
a <- azimuth_image(z)
g <- sky_grid_segmentation(z, a, 15)
plot(g == 24005)
## Not run:
g <- sky_grid_segmentation(z, a, 15, sequential = TRUE)
col <- terra::unique(g) %>% nrow() %>% rainbow() %>% sample()
plot(g, col = col)

## 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.

Value

Returns `invisible(TRUE)` and print "Test passed" if all tests pass, otherwise throws an error.

See Also

Other Lens Functions: `azimuth_image()`, `calc_diameter()`, `calc_zenith_raster_coord()`, `calibrate_lens()`, `expand_noncircular()`, `fisheye_to_equidistant()`, `fisheye_to_pano()`, `lens()`, `zenith_image()`

Examples

```
test_lens_coef(lens("Nikon_FCE9"))
test_lens_coef(2 / pi)
```

---

thr_image *Threshold image*

Description

Transform background digital number into threshold values.

Usage

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

Arguments

dn	Numeric vector or SpatRaster. 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_coneshaped_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 $slope \times w$.

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

Value

An object of the same class and dimensions than `dn`.

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 Binarization Functions: `apply_thr()`, `find_sky_pixels_nonnull()`, `find_sky_pixels()`, `obia()`, `ootb_mblt()`, `regional_thresholding()`

Examples

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

---

write_bin *Write binarized images*

Description

Wrapper functions for `writeRaster`.

Usage

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

Arguments

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

Value

No return value. Called for side effects.

See Also

read_bin

Other Tools Functions: defuzzify(), extract_feature(), extract_rl(), extract_sky_points(), masking(), read_bin(), read_caim(), write_caim()

Examples

```
## Not run:
z <- zenith_image(1000, lens())
m <- !is.na(z)
my_file <- file.path(tempdir(), "mask")
write_bin(m, my_file)
my_file <- as.filename(my_file) %>%
  insert(., ext = "tif", replace = TRUE) %>%
  as.character()
m_from_disk <- read_bin(my_file)
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	SpatRaster.
path	Character vector of length one. Path for writing the image.
bit_depth	Numeric vector of length one.

Value

No return value. Called for side effects.

See Also

write_bin

Other Tools Functions: defuzzify(), extract_feature(), extract_rl(), extract_sky_points(), masking(), read_bin(), read_caim(), write_bin()

Examples

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

## End(Not run)
```

`write_sky_points` *Write sky points*

Description

Create a special file to interface with the HSP software.

Usage

```
write_sky_points(x, path_to_HSP_project, img_name)
```

Arguments

`x`	An object of the class *data.frame*. The result of a calling to `extract_sky_points`.
`path_to_HSP_project`	Character vector of length one. Path to the HSP project folder.
`img_name`	Character vector of length one. See details.

Details

This function is part of a workflow that connects this package with the HSP software package (Lang et al. 2013).

This function was designed to be called after `extract_sky_points`, and is part of a workflow that connects the MBLT algorithm with the HSP software package. In such a workflow, the `r` argument provided to `extract_sky_points` should be an image pre-processed by the HSP software. Those images are stored as PGM files by HSP in the subfolder "manipulate" of the project folder (which will be in turn a subfolder of the "project**s**" folder). Those PGM files can be read with `rast`. For instance: `r <-rast("C:/Users/johndoe/Documents/HSP/Projects/my_prj/manipulate/img01.pgm")`. Then, they will be ready for use as input after running `normalize(r)`.

The `img_name` argument of `write_sky_points()` should be the name of the file associated to the aforementioned `r` argument.

Value

None. A file will be written in the HSP project folder.

References

Lang M, Kodar A, Arumäe T (2013). "Restoration of above canopy reference hemispherical image from below canopy measurements for plant area index estimation in forests/ Metsa võrastiku läbipaistvuse mõõtmine digitaalsete poolsfäärikaamerate abil." *Forestry Studies*, **59**(1), 13–27. doi:10.2478/fsmu20130008.

See Also

Other HSP Functions: extract_sun_coord(), read_manual_input(), read_opt_sky_coef(), row_col_from_zenith_azimuth(), write_sun_coord(), zenith_azimuth_from_row_col()

---

write_sun_coord *Write sun coordinates*

Description

Create a special file to interface with the HSP software.

Usage

```
write_sun_coord(x, path_to_HSP_project, img_name)
```

Arguments

x	Numeric vector of length two. Raster coordinates of the solar disk that can be obtained by calling to extract_sun_coord. **TIP**: if the output of extrac_sun_coord() is sun_coord, then you should provide to write_sun_coord() this: sun_coord$row_col. See also row_col_from_zenith_azimuth.
path_to_HSP_project	Character vector of length one. Path to the HSP project folder.
img_name	Character vector of length one. See details.

Details

Refer to the Details section of this function: write_sky_points.

Value

None. A file will be written in the HSP project folder.

See Also

Other HSP Functions: extract_sun_coord(), read_manual_input(), read_opt_sky_coef(), row_col_from_zenith_azimuth(), write_sky_points(), zenith_azimuth_from_row_col()

zenith_azimuth_from_row_col *Zenith and azimuth angles from row and col numbers*

Description

Zenith and azimuth angles from row and col numbers

Usage

```
zenith_azimuth_from_row_col(r, row_col, lens_coef)
```

Arguments

r	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from an hemispherical photograph. Please see read_caim and normalize.
row_col	Numeric vector of length two. Row and col numbers.
lens_coef	Numeric vector. Polynomial coefficients of the lens projection function.

See Also

Other HSP Functions: extract_sun_coord(), read_manual_input(), read_opt_sky_coef(), row_col_from_zenith_azimuth(), write_sky_points(), write_sun_coord()

Examples

```
z <- zenith_image(1000, lens_coef = lens())
zenith_azimuth_from_row_col(z, c(501, 750), lens())
```

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

An object of class SpatRaster of zenith angles in degrees, showing a complete hemispherical view, with the zenith on the center.

See Also

Other Lens Functions: azimuth_image(), calc_diameter(), calc_zenith_raster_coord(), calibrate_lens(), expand_noncircular(), fisheye_to_equidistant(), fisheye_to_pano(), lens(), test_lens_coef()

Examples

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

Index