Package ‘rcaiman’

October 14, 2022

Type Package

Title CAnopy IMage ANalysis

Version 1.0.8

Date 2022-09-19

Description 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, such as those acquired with
15 mm lenses or with auxiliary fish-eye lenses attached to mobile devices.
Most of the functions also support restricted view photography.

License GPL-3

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

Encoding UTF-8

RoxygenNote 7.2.1

Depends filenamer, magrittr, colorspace, terra

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

Suggests autothresholdr, conicfit, EBImage, bbmle, imager

RdMacros Rdpack

NeedsCompilation no

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

Maintainer Gastón Mauro Díaz <`gastonmaurodiaz@gmail.com`>

Repository CRAN

Date/Publication 2022-09-19 22:56:05 UTC

R topics documented:

`apply_thr` *Apply threshold*

Description

Global or local thresholding of images.

Usage

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

Arguments

`r`	SpatRaster. A greyscale image.
`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 the `thr` argument, it is applied to every pixel of the object `r`. If, instead, a SpatRaster is provided as the `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()`, `ootb_obia()`, `regional_thresholding()`, `thr_image()`, `thr_isodata()`

Examples

```
r <- read_caim()
apply_thr(r$Blue, thr_isodata(r$Blue[]))
## 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, orientation = 0)
```

Arguments

`z`	SpatRaster built with `zenith_image`.
`orientation`	The azimuthal angle at which the top of the image is facing, in degrees. Generally, it corresponds to the angle at which the top of the camera was facing at the moment of acquisition.

Value

An object of class SpatRaster with azimuth angles in degrees. If the `orientation` argument is zero, North (0º) is pointing up as in maps, but East (90º) and West (270º) are flipped respecting to maps. To understand why is that, do the following: 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 one in the paper side that face down–It will be an awkward position, like if you were taking an upward-looking photo with a mobile device while looking at the screen–; finally, 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)
## Not run:
a <- azimuth_image(z, 45)
plot(a)

## End(Not run)
```

---

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 helps handle devices with a 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).

When working with non-circular hemispherical photography, this function will help to find the diameter that a circular image would have if the equipment would record the whole hemisphere.

The required data (radius-angle data) can be obtained following the instructions given in the user manual of Hemisfer software. The following is a slightly simpler alternative:

- Find a vertical wall and a leveled floor, both well-constructed. For instance, a parking lot.
- 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.
- Make a mark on the wall at chosen height over the wall-vertex nearest to the camera-vertex. 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.
- Before taking the photograph, do not forget to align the zenith coordinates with the 0º θ mark and check if the optical axis is leveled.

The line selection tool of ImageJ can be used to measure the distance in pixels between points on the image. Draw a line, and use the dropdown menu Analyze>Measure to obtain its length.

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

Value

Numeric vector of length one. Diameter adjusted to a whole number (see `zenith_image` for details about that constrain).

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)

calc_zenith_raster_coordinates(path_to_csv)
```

Arguments

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

Details

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 does 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. After digitizing the points on the image, use the dropdown menu Analyze>Measure to open the window Results. To obtain the CSV, use File>Save As...

Another method–only valid when enough of the circle perimeter is depicted in the image– 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 circle perimeter. 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. The latter property allows fitting a precise mathematical relation 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

These are the instructions to produce the CSV file required by this function. The following materials are required:

- this package and ImageJ
- camera and lens
- tripod
- standard yoga mat
- table about two times wider 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. 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. In order to take care of the line of pins at 90º relative to the optical axis, it may be better to use the naked eye to align 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.

Then, use the dropdown menu Analyze>Measure to open the window Results. To obtain the CSV, use File>Save As...

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

TIP: use `test_lens_coef` to test if coefficients 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. The latter should be double checked, particularly if the zenith push pin is not exactly on the zenith pixel. To that end, do the following on ImageJ: use the rectangular selection tool to create a small rectangle, open the Specify window by going to the dropdown menu Edit>Selection>Specify..., insert the zenith coordinates (obtained with `calc_zenith_raster_coord`) into the respective X and Y fields in order to align the upper-left corner of the rectangle with the zenith, mark it with the brush, use the straight selection tool to find the length within the zenith and the maximum zenith angle showed in the image.

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)
```

`chessboard` *Chessboard segmentation*

Description

Chessboard segmentation

Usage

```
chessboard(r, size)
```

Arguments

r	SpatRaster.
size	Numerica vector of length one. Size of the square segments.

Value

A single layer image of the class SpatRaster with integer values.

See Also

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

Examples

```
## Not run:
caim <- read_caim()
seg <- chessboard(caim, 100)
plot(caim$Blue)
plot(extract_feature(caim$Blue, seg))

## End(Not run)
```

`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()`, `ootb_sky_reconstruction()`

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)
```

---

colorfulness *Quantify the colorfulness of an image*

Description

Quantify the colorfulness of an sRGB image using a bidimensional space formed by the green/red and the blue/yellow axes of the CIE *L*a*b** space, symbolized with *a** and *b**, respectively. The proposed index is defined as the surface of the *a*b** plane covered by colors from the image relative to the surface that the whole sRGB cube covers in the same plane, expressed in percentage.

Usage

```
colorfulness(caim, m = NULL, plot = FALSE)
```

Arguments

`caim`	SpatRaster. The return of a call to `read_caim`.
`m`	SpatRaster. A mask. For hemispherical photographs, check `mask_hs`. Default (`NULL`) is the equivalent to enter `!is.na(caim$Red)`.
`plot`	Logical vector of length one. If is `TRUE`, a plot will be send to the graphic device, showing the data on the CIE *a*b** space.

Details

Pixels from the image covered by pixels from `m` with value `1` will be taking into account in the computations.

If `plot = TRUE` is used, a plot is sent to the active graphics device. It shows the color from the image plotted into a bidimensional space made by the axis *a** and *b** of the CIE *L*a*b* space*.

An early version of this function was used in Martin et al. (2020).

Value

A numeric vector of length one and, if the argument `plot` is `TRUE`, an object returned by `plot` is send to the graphic device.

References

Martin DA, Wurz A, Osen K, Grass I, Hölscher D, Rabemanantsoa T, Tscharntke T, Kreft H (2020). "Shade-Tree Rehabilitation in Vanilla Agroforests is Yield Neutral and May Translate into Landscape-Scale Canopy Cover Gains." *Ecosystems*, **24**(5), 1253–1267. doi:10.1007/s10021020-005865.

See Also

Other Tool Functions: `defuzzify()`, `extract_dn()`, `extract_feature()`, `extract_rl()`, `extract_sky_points()`, `masking()`, `read_bin()`, `read_caim()`, `write_bin()`, `write_caim()`

Examples

```
caim <- read_caim() %>% normalize()
plotRGB(caim*255)
colorfulness(caim)

path <- system.file("external/DSCN4500.JPG", package = "rcaiman")
caim <- read_caim(path, c(1280, 960) - 745, 745 * 2, 745 * 2) %>% normalize()
plotRGB(caim*255)
colorfulness(caim)
```

`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`.

Details

This method is also available in the HSP software package (Lang et al. 2013).

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 Tool Functions: `colorfulness()`, `extract_dn()`, `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 first 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. 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 = NULL,
  sky_blue = NULL,
  w_red = 0,
  thr = NULL,
  fuzziness = NULL,
  gamma = 2.2
)
```

Arguments

caim	SpatRaster. The return of a call to `read_caim`.
m	SpatRaster. A mask. For hemispherical photographs, check `mask_hs`. Default (`NULL`) is the equivalent to enter `!is.na(caim$Red)`. See the Details section in `local_fuzzy_thresholding` to understand how this argument can modify the output.
sky_blue	color. Is the `target_color` argument to be passed to `membership_to_color`. Default (`NULL`) is the equivalent to enter `sRGB(0.1, 0.4, 0.8)`–the HEX color code is #1A66CC, it can be entered into a search engine (such as Mozilla Firefox) to see a color swatch.
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 `thr_isodata`.
fuzziness	Numeric vector of length one. This number is a constant value that scales `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 argument `sky_blue` can be obtained from a photograph that clearly shows the sky. Then, it can be used to process all the others taken with the same equipment, configuration, and protocol.

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`; if not, different parameters should be tested.

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:
#circular hemispherical photo
path <- system.file("external/b4_2_5724.jpg", package = "rcaiman")
caim <- read_caim(path, c(1280, 960) - 745, 745 * 2, 745 * 2)
z <- zenith_image(1490, lens("Nikon_FCE9"))
a <- azimuth_image(z)
m <- !is.na(z)
blue <- caim$Blue %>% gbc()
plot(caim)

sky_blue_sample <- read_caim(path, c(1092,1243), 66, 48)
plot(sky_blue_sample)
sky_blue <- apply(sky_blue_sample[], 2, median) %>% normalize(.,0,255) %>%
  as.numeric() %>%
  matrix(., ncol = 3) %>%
  sRGB()
hex(sky_blue)
# Use hex() to obtain the HEX color code. To see a color swatch, enter the
# HEX code into a search engine (such as Mozilla Firefox). If the color is
# too pale (i.e., unsaturated), such as the one from the example (#6D90D0),
# it would be better to use the default. Alternatively, the values can be
# stretched, which often produces a more intense color. That is demonstrated
# below.
sky_blue_sample <- read_caim(path, c(1092,1243), 66, 48)
plot(sky_blue_sample)
sky_blue <- apply(sky_blue_sample[], 2, median) %>% normalize() %>%
  as.numeric() %>%
  matrix(., ncol = 3) %>%
  sRGB()
hex(sky_blue) #005AFF

caim <- normalize(caim)
ecaim <- enhance_caim(caim, m)
plot(ecaim)
```

```
plot(blue)

m2 <- !mask_sunlit_canopy(caim, m) & m
hist(ecaim[m2])
hist(blue[m])

plot(apply_thr(ecaim, thr_isodata(ecaim[m2])))
plot(apply_thr(blue, thr_isodata(blue[m])))

#hemispherical photo from a smartphone
path <- system.file("external/APC_0581.jpg", package = "rcaiman")
caim <- read_caim(path)
z <- zenith_image(2132/2, lens("Olloclip"))
a <- azimuth_image(z)
zenith_colrow <- c(1063, 771)/2
caim <- expand_noncircular(caim, z, zenith_colrow) %>% normalize()
m <- !is.na(caim$Red) & !is.na(z)
caim[!m] <- NA
blue <- caim$Blue %>% gbc()

ecaim <- enhance_caim(caim, m)
plot(ecaim)
plot(blue)

m2 <- !mask_sunlit_canopy(caim, m) & m
hist(ecaim[m2])
hist(blue[m])

plot(apply_thr(ecaim, thr_isodata(ecaim[m2])))
plot(apply_thr(blue, thr_isodata(blue[m])))

#restricted view canopy photo
path <- system.file("external/APC_0020.jpg", package = "rcaiman")
caim <- read_caim(path)
plot(caim)
blue <- gbc(caim$Blue)
plot(blue)

caim <- normalize(caim)
ecaim <- enhance_caim(caim)
plot(ecaim)

m <- !mask_sunlit_canopy(caim)
hist(ecaim[])
hist(ecaim[m])
hist(blue)
plot(apply_thr(ecaim, thr_isodata(ecaim[m])))
plot(apply_thr(blue, thr_isodata(blue[])))


## 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:
 #noncircular fisheye from a DSLR camera
 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, col = seq(0,1,1/255) %>% grey())
 plot(is.na(r$Red), add = TRUE, alpha = 0.3,legend = FALSE)

 #noncircular fisheye from a smartphone with an auxiliary lens
 path <- system.file("external/APC_0581.jpg", package = "rcaiman")
 caim <- read_caim(path)
```

```
z <- zenith_image(2132/2, lens("Olloclip"))
a <- azimuth_image(z)
zenith_colrow <- c(1063, 771)/2
caim <- expand_noncircular(caim, z, zenith_colrow)
plot(caim$Blue, col = seq(0,1,1/255) %>% grey())
m <- !is.na(caim$Red) & !is.na(z)
plot(m, add = TRUE, alpha = 0.3, legend = FALSE)

#restricted view canopy photo
path <- system.file("external/APC_0020.jpg", package = "rcaiman")
caim <- read_caim(path)
plot(caim)
caim <- normalize(caim)
diameter <- calc_diameter(lens(), sqrt(nrow(caim)^2 + ncol(caim)^2)/2, 90)
z <- zenith_image(diameter, lens())
caim <- expand_noncircular(caim, z, c(ncol(caim)/2, nrow(caim)/2))
m <- !is.na(caim$Red)
a <- azimuth_image(z)
caim[!m] <- 0
z <- normalize(z, 0, 90) * 20 # a diagonal FOV of 40 degrees
plot(caim$Blue, col = seq(0,1,1/255) %>% grey())
m <- !is.na(caim$Red) & !is.na(z)
plot(m, add = TRUE, alpha = 0.3, legend = FALSE)

## End(Not run)
```

`extract_dn` *Extract digital numbers*

Description

It is a wrapper function around `extract`.

Usage

```
extract_dn(r, img_points, use_window = TRUE, fun = NULL)
```

Arguments

`r`	SpatRaster.
`img_points`	The result of a call to `extract_sky_points`, or an object of the same class and structure.
`use_window`	Logical vector of length one. If `TRUE`, a 3×3 window will be used to extract the sky digital number from `r`.
`fun`	A `function` that takes a vector as input and returns a one-length numeric or logical vector as output (e.g. mean).

Value

An object of the class *data.frame*, which is the argument `img_points` with an added column per each layer from `r`. The layer names are used to name the new columns. If a function is provided as the `fun` argument, the result will be summarized per column using the provided function, and the *row* and *col* information will be omitted. Moreover, if `r` is an RGB image, a color will be returned instead of a *data.frame*. The latter feature is useful for obtaining the `sky_blue` argument for `enhance_caim`.

See Also

Other Tool Functions: `colorfulness()`, `defuzzify()`, `extract_feature()`, `extract_rl()`, `extract_sky_points()`, `masking()`, `read_bin()`, `read_caim()`, `write_bin()`, `write_caim()`

Examples

```
## Not run:
caim <- read_caim()
r <- gbc(caim$Blue)
bin <- apply_thr(r, thr_isodata(r[]))
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
a <- azimuth_image(z)
g <- sky_grid_segmentation(z, a, 10)
sky_points <- extract_sky_points(r, bin, g)
sky_points <- extract_dn(caim, sky_points)
head(sky_points)
sky_points <- extract_sky_points(r, bin, g)
sky_points

## End(Not run)

# ImageJ can be used to digitize points.
# See calc_zenith_raster_coord() for details.
path <- system.file("external/b4_2_5724.jpg", package = "rcaiman")
caim <- read_caim(path)
plot(caim)
path <- system.file("external/points_over_perimeter.csv",
                    package = "rcaiman")
img_points <- read.csv(path)
img_points <- img_points[,c(ncol(img_points), ncol(img_points)-1)]
colnames(img_points) <- c("row", "col")
head(img_points)
v <- cellFromRowCol(caim, img_points$row, img_points$col) %>%
  xyFromCell(caim, .) %>% vect()
plot(v, add = TRUE, col = 2)
extract_dn(caim, img_points, fun = median)
```

`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 return 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 Tool Functions: `colorfulness()`, `defuzzify()`, `extract_dn()`, `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 a canopy 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`. As an alternative, both ImageJ and HSP software package (Lang et al. 2013) can be used to manually digitize points. See `extract_dn` and `read_manual_input` for details.
`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

The search for near-zenith points starts in the region ranged between `0` and `z_thr`. If the number of near-zenith points is less than `no_of_points`, the region increases by steps of `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.

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 Tool Functions: `colorfulness()`, `defuzzify()`, `extract_dn()`, `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 a canopy 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 to be pure sky pixels.
g	SpatRaster built with `sky_grid_segmentation` or `chessboard`.
dist_to_plant, min_raster_dist	Numeric vector of length one or `NULL`.

Details

This function will automatically sample sky pixels from the sky regions 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 the first step, sky pixels from `r` are evaluated to find, for each cell of `g`, the pixel with maximum digital value (local maximum). The argument `dist_to_plant` allows users to establish a buffer zone for `bin`, meaning a size reduction of original sky regions.

The final step filters these local maximum values by calculating distances between points on the raster space. It discards new points that have a distance from existing points minor than `min_raster_dist`. Cell labels determine the order in which the points are evaluated.

To skip a given filtering step, use code `NULL` as argument input. For instance, to provide `min_raster_dist = NULL` will return points omitting raster distance calculation, which means a faster output in comparison with using `min_raster_dist = 1`.

Value

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

See Also

`fit_cie_sky_model`

Other Tool Functions: `colorfulness()`, `defuzzify()`, `extract_dn()`, `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)
bin <- ootb_obia(caim, z, a)
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 a canopy 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 to be pure sky pixels.
- `g` SpatRaster built with `sky_grid_segmentation` or `chessboard`.
- `max_angular_dist` Numeric vector of length one. Angle in degrees to control the maximum potential size of the solar corona.

Details

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

The selected cells are grouped based on adjacency, and new bigger segments are created from these groups. The degree of membership to the class *Sun* is calculated for every new segment by computing the number of cells that constitute the segment and its mean digital number (values taken from `r`). In other words, the largest and brightest 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.

Value

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

Examples

```
## Not run:
caim <- read_caim() %>% normalize()
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
a <- azimuth_image(z)
bin <- ootb_obia(caim, z, a)
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 a canopy 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 percentage of cells to sample. 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 uses 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 starts with a quantile probability of 0.99. After producing the binarized image, this function uses a search grid with cells of 5×5 degrees to count how many of these cells have 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 starts 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()`, `ootb_obia()`, `regional_thresholding()`, `thr_image()`, `thr_isodata()`

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 using the increase in the number of cells having no sky pixels (the so-called null cells) as 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 a canopy 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` or `chessboard`.
`slope`	Numeric vector of length one. See section Details in `thr_image`.

Details

The arguments `sky` and `slope` are passed to `thr_image`, which output 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()`, `ootb_obia()`, `regional_thresholding()`, `thr_image()`, `thr_isodata()`

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)

reproject_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).

Details

There is no interpolation, so NA values may be generated depending on both the radius argument and how much the lens projection differs from the polar one. As a rule of thumb, increase radius as long as it does not produce NA values on the regions to be analyzed.

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)
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.

Details

An early version of this function was used in Díaz et al. (2021).

References

Díaz GM, Negri PA, Lencinas JD (2021). "Toward making canopy hemispherical photography independent of illumination conditions: A deep-learning-based approach." *Agricultural and Forest Meteorology*, **296**, 108234. doi:10.1016/j.agrformet.2020.108234.

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)
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 best fit to data sampled from a canopy photograph.

Usage

```
fit_cie_sky_model(
  r,
  z,
  a,
  sky_points,
  zenith_dn,
  sun_coord,
  custom_sky_coef = NULL,
  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 a canopy 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 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, or an object with same structure and names. See also row_col_from_zenith_azimuth in case you want to provide values based on date and time of acquisition and the R package 'suncalc'.

custom_sky_coef — Numeric vector of length five. Custom starting coefficients of the sky model. By default, they are drawn from standard skies.

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 less root mean square error calculated by using sky_points as reference values. 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 is based on Lang et al. (2010). In theory, the best 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 handling any non-linearity since it is not a physical model with strict assumptions.

Ultimately, when 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.

Nevertheless, the recommended input for this function is data pre-processed with the HSP software package (Lang et al. 2013). Please, refer to write_sky_points for additional details about HSP.

The following code exemplifies how this package can be used to compare the manually-guided fitting provided by HSP against the automatic fitting provided by this package. The code assumes that the user is working within an RStudio project located in the HSP project folder.

```
r <- read_caim("manipulate/IMG_1013.pgm")
z <- zenith_image(ncol(r), lens())
a <- azimuth_image(z)
manual_input <- read_manual_input(".", "IMG_1013" )
sun_coord <- manual_input$sun_coord$row_col
sun_coord <- zenith_azimuth_from_row_col(z, sun_coord, lens())
sky_points <- manual_input$sky_points
```

```
rl <- extract_rl(r, z, a, sky_points)
model <- fit_cie_sky_model(r, z, a, rl$sky_points, rl$zenith_dn, sun_coord)
cie_sky <- model$relative_luminance * model$zenith_dn
plot(r/cie_sky)

r <- read_caim("manipulate/IMG_1013.pgm")
sky_coef <- read_opt_sky_coef(".", "IMG_1013")
cie_sky_manual <- cie_sky_model_raster(z, a, sun_coord$zenith_azimuth, sky_coef)
cie_sky_manual <- cie_sky_manual * manual_input$zenith_dn
plot(r/cie_sky_manual)
```

If you use this function in your research, please cite Lang et al. (2010) in addition to this package.

Value

The result includes the following: (1) the output produced by mle2, (2) the 5 coefficients, (3) observed and predicted values, the sun coordinates –zenith and azimuth angle in degrees–, (4) the relative luminance image calculated for every pixel using the estimated coefficients and corresponding sun coordinates, (4) the digital number at the zenith, and (5) the description of the standard sky from which the initial coefficients were drawn. See Li et al. (2016) to know more about these coefficients.

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.

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()`, `ootb_sky_reconstruction()`

Examples

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

bin <- ootb_obia(caim, z, a)
```

```
bin <- bin & mask_hs(z, 0, 80)

r <- gbc(caim$Blue*255)
g <- sky_grid_segmentation(z, a, 10)
sun_coord <- extract_sun_coord(r, z, a, bin, g)
sky_points <- extract_sky_points(r, bin, g)
rl <- extract_rl(r, z, a, sky_points)
model <- fit_cie_sky_model(r, z, a, rl$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)

#to provide custom starting coefficient
path <- system.file("external", package = "rcaiman")
skies <- utils::read.csv(file.path(path, "15_CIE_standard_skies.csv"))
custom_sky_coef <- skies[9, 1:5] %>% as.numeric()
fit_cie_sky_model(r, z, a, rl$sky_points,
                  rl$zenith_dn, sun_coord,
                  rmse = TRUE,
                  custom_sky_coef = custom_sky_coef)

#restricted view canopy photo
path <- system.file("external/APC_0020.jpg", package = "rcaiman")
caim <- read_caim(path)
plot(caim)
caim <- normalize(caim)
diameter <- calc_diameter(lens(), sqrt(nrow(caim)^2 + ncol(caim)^2)/2, 90)
z <- zenith_image(diameter, lens())
caim <- expand_noncircular(caim, z, c(ncol(caim)/2, nrow(caim)/2))
m <- !is.na(caim$Red)
a <- azimuth_image(z)
caim[!m] <- 0
z <- normalize(z, 0, 90) * 20 # a diagonal FOV of 40 degrees, a rough guess

bin <- ootb_obia(caim)

g <- sky_grid_segmentation(z, a, 5, sequential = TRUE)
col <- terra::unique(g) %>% nrow() %>% rainbow() %>% sample()
plot(g, col = col)
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$zenith_dn, sun_coord, twilight = FALSE)
sky_cie <- model$relative_luminance * model$zenith_dn
plot(sky_cie)
plot(r/sky_cie)
```

```
## End(Not run)
```

---

fit_coneshaped_model *Fit cone-shaped model*

Description

Statistical modeling 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 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) in addition to this package.

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(), ootb_sky_reconstruction()

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 a canopy 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 to be 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 considered as sky pixels erroneously. 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 as 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) in addition to this package.

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(), ootb_sky_reconstruction()

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 reconstructed sky*

Description

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

Usage

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

fix_predicted_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` (the data source).
`bin`	SpatRaster. The binarization of `r` used to select the sky DNs for building the `sky` argument.

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).

The proposed automatic edition consists of (1) flattening the values below the minimum value from the data source–defined by `r` and `bin`–and (2) forcing the values toward the horizon to become gradually the median value from the data source. The latter is achieved by calculating the weighted average of the median value and the predicted sky DNs, using the ratio of `z` to `90` 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()`, `ootb_sky_reconstruction()`

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, i.e., 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 as `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

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 canopy photographs.

Usage

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

Arguments

`sky_points`	An object of class *data.frame*. The result of a call to `extract_rl` or `extract_dn`, or a *data.frame* with same basic structure and names.
`g`	SpatRaster built with `sky_grid_segmentation` or `chessboard`.
`k`	Numeric vector of length one. Number of k-nearest neighbors.
`p`	Numeric vector of length one. Power for inverse-distance weighting.
`rmax`	Numeric vector of length one. Maximum radius where to search for knn.
`col_id`	Numeric vector of length one. ID of the column with the values to interpolate.

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

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

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(), ootb_sky_reconstruction()

Examples

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

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)

#modify g if the goal is to get the whole sky
g <- !is.na(z)
sky <- interpolate_sky_points(sky_points$sky_points, g)
plot(sky)
plot(r/sky)

#restricted view canopy photo
path <- system.file("external/APC_0020.jpg", package = "rcaiman")
caim <- read_caim(path)
plot(caim)
r <- gbc(caim$Blue)
caim <- normalize(caim)

bin <- ootb_obia(caim)
```

```
g <- chessboard(caim, 100)
plot(g)
sky_points <- extract_sky_points(r, bin, g)
sky_points <- extract_dn(r, sky_points)
head(sky_points)
sky <- interpolate_sky_points(sky_points, !is.na(r), col_id = 3)
plot(sky)
plot(r/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

In upward-looking leveled hemispherical photography, the zenith is the center of a circle whose perimeter is the horizon. This is true only if the lens field of view is 180º. The relative radius is the radius of concentric circles expressed as a fraction of the radius that belongs to the circle that has the horizon as perimeter. The equidistant model, also called polar, is the most widely used as a standard reference. Real lenses can approximate the projection models, but they always have some kind of distortion. In the equidistant model, the relation between zenith angle and relative radius is modeled with a straight line. Following Hemisfer software, this package uses a polynomial curve to model lens distortion. A third-order polynomial is sufficient in most cases (Frazer et al. 2001).

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)
- **Olloclip**: Auxiliary lens. Unpublished

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 coefficients 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.

Frazer GW, Fournier RA, Trofymow JA, Hall RJ (2001). "A comparison of digital and film fisheye photography for analysis of forest canopy structure and gap light transmission." *Agricultural and Forest Meteorology*, **109**(4), 249–263. doi:10.1016/s01681923(01)00274x.

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 first 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. For hemispherical photographs, check 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 thr_isodata.
fuzziness	Numeric vector of length one. This number is a constant value that scales 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) in addition to this package.

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. A mask. For hemispherical photographs, check `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 areas where `m` is equal to zero 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 Tool Functions: `colorfulness()`, `defuzzify()`, `extract_dn()`, `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, this function 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: `chessboard()`, `mask_sunlit_canopy()`, `polar_qtree()`, `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

It is a wrapper function around membership_to_color. It masks pixels that are likely sunlit canopy.

Usage

```
mask_sunlit_canopy(caim, m = NULL)
```

Arguments

caim	SpatRaster. The return of a call to read_caim.
m	SpatRaster. A mask. For hemispherical photographs, check mask_hs. Default (NULL) is the equivalent to enter !is.na(caim$Red). See the Details section in local_fuzzy_thresholding to understand how this argument can modify the output.

Value

An object of class SpatRaster with values 0 and 1.

See Also

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

Examples

```
## Not run:
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)

## End(Not run)
```

membership_to_color — *Compute the membership to a target color*

Description

This function was first 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) in addition to this package.

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()
plot(caim)
caim <- normalize(caim, 0, 255)
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
mem <- membership_to_color(caim, sRGB(0.25, 0.75, 0))
```

```
plot(mem)

## End(Not run)
```

---

normalize *Normalize data*

Description

Normalize numeric and raster data.

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. Default is equivalent to enter the minimum value from r.
mx	Numeric vector of length one. Maximum expected value. Default is equivalent to enter the maximum value from r.
force_range	Logical vector of length one. If it is TRUE, the range is forced to be between 0 and 1 by flattening values found below and above those limits.

Details

Normalize data laying between mn and mx to 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.

Value

An object from the same class as 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 the actual minimum and maximum from r, then the output will not cover the 0-to-1 range and 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 canopy photographs*

Description

Object-based image analysis targeting the canopy silhouette.

Usage

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

Arguments

`r`	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from a canopy 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 working binarization of `r` without gross errors.
`segmentation`	SpatRaster built with `polar_qtree` or `qtree`.
`gf_mn, gf_mx`	Numeric vector of length one. The minimum/maximum gap fraction that a segment should comply with to be considered as one containing foliage.

Details

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`.

This method produces 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*– the latter is defined by `bin` equal to one. To avoid spurious values, the quantile `0.9` is computed instead of the maximum. Pixels not belonging to the class *foliage* return as `NA`.

Default values of `z` and `a` allows the processing of restricted view photographs.

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

Value

SpatRaster.

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(), ootb_obia(), regional_thresholding(), thr_image(), thr_isodata()

Examples

```
## Not run:
caim <- read_caim()
r <- caim$Blue %>% gbc()
caim <- normalize(caim)
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
a <- azimuth_image(z)
m <- !is.na(z)
m2 <- !mask_sunlit_canopy(caim, m)
ecaim <- enhance_caim(caim, m)
bin <- apply_thr(ecaim, thr_isodata(ecaim[m2]))

seg <- polar_qtree(caim, z, a)
synth <- obia(r, z, a, bin, seg)
plot(synth)
foliage <- !is.na(synth)
hist(synth[foliage])
synth <- terra::cover(synth, bin)
plot(synth)
bin_obia <- apply_thr(synth, thr_isodata(synth[foliage]))
plot(bin - bin_obia)
plot(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, fix_cs_sky = FALSE)
```

Arguments

r	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from a canopy 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 to be pure sky pixels.
fix_cs_sky	Logical vector of length one. If it is TRUE, fix_reconstructed_sky is used to fix the cone-shaped sky.

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 the first step instead of a global one. Please refer to find_sky_pixels.
- It does not use asynchronous acquisition under the open sky. The cone-shaped model (fit_coneshaped_model) run without a filling source and 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, after fit_coneshaped_model.

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 a 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 happen with photographs of open forests for the same working principle.

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

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

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

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)
# Alternative 1
plot(apply_thr(ratio, thr_isodata(ratio[!is.na(z)])))

# Alternative 2
g <- sky_grid_segmentation(z, a, 10)
plot(defuzzify(ratio, g))

##Note: In this example, differences are small, but they can be notorious.

## End(Not run)
```

`ootb_obia` *Out-of-the-box object-based image analysis of canopy photographs*

Description

Out-of-the-box version of methods first presented in Díaz and Lencinas (2015).

Usage

```
ootb_obia(caim, z = NULL, a = NULL, m = NULL, sky_blue = NULL)
```

Arguments

`caim`	SpatRaster. The return of a call to `read_caim`.
`z`	SpatRaster built with `zenith_image`.
`a`	SpatRaster built with `azimuth_image`.
`m`	SpatRaster. Default (`NULL`) is the equivalent to enter `!is.na(z)` for hemispherical photography, or enter `!is.na(caim$Red)` for restricted view photography.
`sky_blue`	color. Is the `target_color` argument to be passed to `membership_to_color`. Default (`NULL`) is the equivalent to enter `sRGB(0.1, 0.4, 0.8)`–the HEX color code is #1A66CC, it can be entered into a search engine (such as Mozilla Firefox) to see a color swatch.

Details

This function is a hard-coded version of a pipeline that combines these main functions mask_sunlit_canopy, enhance_caim, polar_qtree/qtree, and obia. The code can be easily inspected by calling ootb_obia –no parenthesis. Advanced users can use that code as a template.

Pixels from the synthetic layer returned by obia that lay between 0 and 1 are assigned to the class *plant* only if they are:

- 0 after defuzzify with a sky grid segmentation of 10 degrees.
- 0 after apply_thr with a threshold computed with thr_isodata.
- Not exclusively surrounded by sky pixels.

Default values of z and a allows the processing of restricted view photographs.

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

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 values 0 and 1.

See Also

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

Examples

```
## Not run:
#circular hemispherical photo
path <- system.file("external/b4_2_5724.jpg", package = "rcaiman")
caim <- read_caim(path, c(1280, 960) - 745, 745 * 2, 745 * 2) %>%
  normalize()
z <- zenith_image(1490, lens("Nikon_FCE9"))
a <- azimuth_image(z)

bin <- ootb_obia(caim, z, a)
plot(bin)

## to compare
blue <- gbc(caim$Blue*255)
plot(apply_thr(blue, thr_isodata(blue[!is.na(z)])))
plot(blue, col = seq(0,1,1/255) %>% grey())

#hemispherical photo from a smartphone
path <- system.file("external/APC_0581.jpg", package = "rcaiman")
caim <- read_caim(path) %>% normalize()
z <- zenith_image(2132/2, lens("Olloclip"))
a <- azimuth_image(z)
```

```
zenith_colrow <- c(1063, 771)/2
caim <- expand_noncircular(caim, z, zenith_colrow) %>% normalize()
m <- !is.na(caim$Red) & !is.na(z)
caim[!m] <- 0

bin <- ootb_obia(caim, z, a)
plot(bin)

## to compare
blue <- gbc(caim$Blue*255)
plot(apply_thr(blue, thr_isodata(blue[m])))
plot(blue, col = seq(0,1,1/255) %>% grey())

#restricted view canopy photo
path <- system.file("external/APC_0020.jpg", package = "rcaiman")
caim <- read_caim(path) %>% normalize()

bin <- ootb_obia(caim)
plot(bin)

## to compare
blue <- gbc(caim$Blue*255)
plot(apply_thr(blue, thr_isodata(blue[])))
plot(blue, col = seq(0,1,1/255) %>% grey())

## 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, filling_source = NULL)
```

Arguments

r	SpatRaster. A normalized greyscale image. Typically, the blue channel extracted from a canopy 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 to be 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 considered as sky pixels erroneously. 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 as 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 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 Lang et al. (2010). 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 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.

Providing a filling source triggers an alternative pipeline in which the sky is fully reconstructed with interpolate_sky_points after a dense sampling (1×1 degree cells), which is supported by the fact that sky digital numbers will be available for almost every pixel, either from r gaps or from the filling source–an exception is a filling source with plenty of NA values, which should not be provided.

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.

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(), interpolate_sky_points()

Examples

```
## Not run:
#JPEG file
caim <- read_caim()
z <- zenith_image(ncol(caim), lens("Nikon_FCE9"))
a <- azimuth_image(z)
r <- gbc(caim$Blue)
bin <- ootb_obia(caim %>% normalize(), z, a)
bin <- bin & mask_hs(z, 0, 85)
sky <- ootb_sky_reconstruction(r, z, a, bin)
sky <- normalize(sky, 0, 1, TRUE)
plot(sky)
sky <- ootb_sky_reconstruction(r, z, a, bin, sky)

ratio <- r/sky
plot(ratio)
hist(ratio)
ratio <- normalize(ratio, 0, 1, TRUE)
g <- sky_grid_segmentation(z, a, 10)
plot(defuzzify(ratio, g))


#preprocessed with HSP
path <- system.file("external/DSCN6342.pgm", package = "rcaiman")
r <- read_caim(path) %>% normalize()
z <- zenith_image(ncol(r), lens())
a <- azimuth_image(z)
bin <- find_sky_pixels(r, z, a)
sky <- ootb_sky_reconstruction(r, z, a, bin)
bin <- apply_thr(r/sky, 0.5)
sky <- ootb_sky_reconstruction(r, z, a, bin, sky)
ratio <- r/sky
ratio[is.na(ratio)] <- 0
ratio <- normalize(ratio, 0, 1, force_range = TRUE)
plot(ratio)
g <- sky_grid_segmentation(z, a, 10)
plot(defuzzify(ratio, g))

## End(Not run)
```

`polar_qtree` *Quad-tree segmentation in the 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 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, 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.

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 controlling the size of the resulting segments. Ultimately, segments sizes will depend on both this parameter and the heterogeneity of `r`.

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.

Value

A single layer image of the class SpatRaster with integer values.

See Also

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

Examples

```
## Not run:
caim <- read_caim()
plot(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)
plot(extract_feature(caim$Blue, seg))

## End(Not run)
```

`qtree` *Quad-tree segmentation*

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 stops locally. This is the usual implementation of the quad-tree algorithm, so it produces squared segments of different sizes. This particular implementation allows up to five sizes.

Usage

```
qtree(r, scale_parameter = 0.2)
```

Arguments

`r` SpatRaster.

`scale_parameter`
Numeric vector of length one. Quad-tree is a top-down method. This parameter controls the stopping condition. Therefore, it allows controlling the size of the resulting segments. Ultimately, segments sizes will depend on both this parameter and the heterogeneity of `r`.

Details

The algorithm starts splitting the entire image into large squared segments following, depending on the aspect ratio, grids going from 4×4 to 1×4/4×1; then, splits each segment 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.

Value

A single layer image of the class SpatRaster with integer values.

See Also

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

Examples

```
## Not run:
caim <- read_caim()
plot(caim)
caim <- normalize(caim, 0, 255)
```

```
seg <- qtree(caim, scale_parameter = 0.5)
plot(caim$Blue)
plot(extract_feature(caim$Blue, seg))
plot(extract_feature(seg, seg, length))

## 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 certain pictures taken with auxiliary fisheye lens attached to smartphones.

Binarization

`apply_thr`, `defuzzify`, `find_sky_pixels_nonnull`, `find_sky_pixels`, `obia`, `ootb_mblt`, `ootb_obia`, `regional_thresholding`, and `thr_image`.

HSP

`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

`chessboard`, `mask_hs`, `mask_sunlit_canopy`, `polar_qtree`, `qtree`, `rings_segmentation`, `sectors_segmentation`, and `sky_grid_segmentation`.

Sky reconstruction

`extract_sun_coord`, `fit_cie_sky_model`, `fit_coneshaped_model`, `fit_trend_surface`, `fix_reconstructed_sky`, `interpolate_sky_points`, and `ootb_sky_reconstruction`.

Tools

`colorfulness`, `extract_feature`, `extract_dn`, `extract_rl`, `extract_sky_points`, `masking`, `read_bin`, `read_caim`, `write_bin`, and `write_caim`.

Batch Processing

Batch processing can be easily performed with standard R programming. Below is an example that can be used as a template.

```
require(rcaiman)

input_folder <- "c:/Users/janedoe/pics/"
output_folder <- "c:/Users/janedoe/bins/"
files <- dir(input_folder, full.names = TRUE)

for (i in 1:length(files)) {
 caim <- read_caim(file.path(files[i]))
 blue <- gbc(caim$Blue)
 bin <- apply_thr(blue, thr_isodata(blue[]))
 write_bin(bin, file.path(output_folder, basename(files[i])))
}
```

---

read_bin *Read binarized images*

Description

Wrapper functions for `rast`.

Usage

```
read_bin(path)
```

Arguments

`path`	Character vector of length one. Path to a binarized image.

Value

An object from class SpatRaster.

See Also

Other Tool Functions: `colorfulness()`, `defuzzify()`, `extract_dn()`, `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 = NULL, upper_left = NULL, width = NULL, height = NULL)
```

Arguments

path	Character vector of length one. Path to an image, including file extension. 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 these parameters, but any image editor can be used, such as 'GIMP' or 'Adobe Photoshop'.

TIP: For obtaining `upper_left`, `width`, and `height`, open the image on the Fiji distro of ImageJ, draw a rectangular selection, and go to Edit>Selection>Specify. The same workflow may work with other distros.

Value

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

See Also

Other Tool Functions: colorfulness(), defuzzify(), extract_dn(), 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)

# A pre-processed image
path <- system.file("external/DSCN6342.pgm", package = "rcaiman")
r <- read_caim(path)
plot(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. For instance, "C:/Users/johndoe/Documents/HSP/Projects/my_prj/".
img_name	Character vector of length one. For instance, "DSCN6342.pgm" or "DSCN6342". See details.

Details

Refer to the Details section of function `write_sky_points`.

Value

A list of numeric vectors named *weight*, *max_points*, *angle*, *point_radius*, *sun_coord*, *sky_points* and *zenith_dn*.

See Also

Other HSP Functions: `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. For instance, "C:/Users/johndoe/Documents/HSP/Projects/my_prj/".

`img_name`
: Character vector of length one. For instance, "DSCN6342.pgm" or "DSCN6342". See details.

Details

Refer to the Details section of function `write_sky_points`.

Value

Numeric vector of length five.

See Also

`cie_sky_model_raster`

Other HSP Functions: `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 a canopy 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–see section Details in `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 was recommended by Jonckheere et al. (2005).
- **Method isodata from this package**: Use `"thr_isodata"` to use `thr_isodata`.

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

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

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)
bin <- regional_thresholding(blue, rings, "thr_isodata")
plot(bin)

## End(Not run)
```

`rings_segmentation` *Rings segmentation*

Description

Segmenting an hemispherical view by slicing the zenith angle from zero 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: chessboard(), mask_hs(), mask_sunlit_canopy(), polar_qtree(), 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(z, za, lens_coef)
```

Arguments

z	SpatRaster built with zenith_image.
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: 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_from_zenith_azimuth(z, c(45, 270), lens())
```

sectors_segmentation *Sectors segmentation*

Description

Segmenting a hemispherical view by slicing the azimuth angle from zero 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: `chessboard()`, `mask_hs()`, `mask_sunlit_canopy()`, `polar_qtree()`, `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 cell 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: `chessboard()`, `mask_hs()`, `mask_sunlit_canopy()`, `polar_qtree()`, `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 if a lens projection function will work 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 a JPEG image, gamma back corrected.
`intercept, slope`	Numeric vector of length one. These are linear function coefficients–see section Details in `thr_image`.

Details

This function transforms background digital numbers into threshold values by means of the Equation 1 from 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 perforated surfaces 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, shifting 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 was a design decision aiming to harmonize the whole package, although it might sound counter intuitive.

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()`, `ootb_obia()`, `regional_thresholding()`, `thr_isodata()`

Examples

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

thr_isodata *Threshold calculated with the isodata method*

Description

Threshold calculated with the algorithm by Ridler and Calvard (1978), which was recommended by Jonckheere et al. (2005).

Usage

```
thr_isodata(x)
```

Arguments

x	Numeric vector or a single-column *matrix* or *data.frame* able to be coerced to numeric.

Details

The implementation is based on the IsoData method of Auto Threshold ImageJ plugin by Gabriel Landini, which is now available in the 'autothresholdr' package (`auto_thresh`). However, I found this implementarion more versatile since it is not restricted to an 8-bit input.

Value

Numeric vector of length one.

References

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

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

Examples

```
caim <- read_caim()
r <- gbc(caim$Blue)
thr <- thr_isodata(values(r))
bin <- apply_thr(r, thr)
plot(bin)
```

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

Other Tool Functions: colorfulness(), defuzzify(), extract_dn(), 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

Other Tool Functions: colorfulness(), defuzzify(), extract_dn(), 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(sky_points, path_to_HSP_project, img_name)
```

Arguments

- `sky_points` 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. For instance, "C:/Users/johndoe/Documents/HSP/Projects/my_prj/".
- `img_name` Character vector of length one. For instance, "DSCN6342.pgm" or "DSCN6342". 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`. 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 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 `read_caim`.

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

The following code exemplifies how this package can be used in conjunction with the HSP software. The code assumes that the user is working within an RStudio project located in the HSP project folder.

```
r <- read_caim("manipulate/IMG_1014.pgm")
plot(r)
z <- zenith_image(ncol(r), lens())
a <- azimuth_image(z)
g <- sky_grid_segmentation(z, a, 10)
mblt <- ootb_mblt(r, z, a)
bin <- find_sky_pixels_nonnull(r, mblt$sky_s, g)
bin <- mask_hs(z, 0, 85) & bin

sun_coord <- extract_sun_coord(r, z, a, bin, g)
write_sun_coord(sun_coord$row_col, ".", "IMG_1014")

sky_points <- extract_sky_points(r, bin, g)
write_sky_points(sky_points, ".", "IMG_1014")
```

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: 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(sun_coord, path_to_HSP_project, img_name)
```

Arguments

sun_coord	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 here this: sun_coord$row_col. See also row_col_from_zenith_azimuth in case you want to provide values based on date and time of acquisition and the R package 'suncalc'.
path_to_HSP_project	Character vector of length one. Path to the HSP project folder. For instance, "C:/Users/johndoe/Documents/HSP/Projects/my_prj/".
img_name	Character vector of length one. For instance, "DSCN6342.pgm" or "DSCN6342". See details.

Details

Refer to the Details section of function write_sky_points.

Value

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

See Also

Other HSP Functions: 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(z, row_col, lens_coef)
```

Arguments

z	SpatRaster built with `zenith_image`.
row_col	Numeric vector of length two. Row and col numbers.
lens_coef	Numeric vector. Polynomial coefficients of the lens projection function.

Value

Numeric vector of length two.

See Also

Other HSP Functions: `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 angle values.

Usage

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

Arguments

diameter	Numeric vector of length one. Diameter in pixels expressed as an even integer, so to simplify calculations by having the zenith point located between pixels. Snapping the zenith point between pixels does not affect accuracy because half-pixel is less than the uncertainty in localizing the circle within the picture.
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