Package 'RootscanR'

October 30, 2025

Title Stitching and Analyzing Root Scans

Version 0.0.1

Author Sophie Kersting [aut, cre] (ORCID:
<https://orcid.org/0000-0002-1038-9246>),
Linda Knüver [aut],
Mareike Fischer [aut] (ORCID: <https://orcid.org/0000-0002-9429-0859>)

Maintainer Sophie Kersting <sophie.kersting@uni-greifswald.de>

Description Minirhizotrons are widely used to observe and explore roots and their growth. This package provides the means to stitch images and divide them into depth layers.
Please note that this R package was developed alongside the following manuscript:
Stitching root scans and extracting depth layer information -- a workflow and practical examples, S. Kersting, L. Knüver, and M. Fischer.
The manuscript is currently in preparation and should be citet as soon as it is available.
This project was supported by the project ArtIGROW, which is a part of the WIR!-Alliance ArtIFARM – Artificial Intelligence in Farming funded by the German Federal Ministry of Research, Technology and Space (No. 03WIR4805).

License GPL (>= 3)

Depends R (>= 3.5.0)

Imports stringr, grid, png, abind, parallel

Encoding UTF-8

NeedsCompilation no

RoxygenNote 7.3.3

Repository CRAN

Date/Publication 2025-10-30 20:00:02 UTC

Contents

`checkInput` *Check if the root scan directories contain everything necessary*

Description

`checkInput` - This function checks a set of root scan directories if they each contain two pictures, X.segmentation.png and X.skeleton.png, where X is the name of the directory.

Usage

```
checkInput(data_dir = NULL, data_dirs = NULL)
```

Arguments

`data_dir`	(Optional, default = NULL) String specifying the name (full path) of the directory (full path) containing all root scan directories of interest.
`data_dirs`	(Optional, default = NULL) Character vector specifying all of the individual root scan directories of interest (full paths). This is only used if `data_dir` is set to NULL.

Value

`checkInput` A list containing two TRUE-FALSE-vectors showing if the root scan directories each contain the two root pictures.

Examples

```
# Replace NULL with path to directory.
DATA_DIR <- NULL
# Apply the function directly...
cI1 <- checkInput(data_dir = DATA_DIR)
# ... or use it in combination with \code{getOverviewInput}:
cI2 <- checkInput(data_dirs = getOverviewInput(data_dir = DATA_DIR,
                                    naming_conv = "standard")$dir_name_full)
```

---

cm2px *Functions for converting pixels and centimeters*

Description

cm2px - Converts centimeters to pixels.

px2cm - Converts pixels to centimeters.

Usage

```
cm2px(cm, ppcm = NULL, ppi = NULL)

px2cm(px, ppcm = NULL, ppi = NULL)
```

Arguments

cm	Numeric value or numeric vector (centimeters) to be converted to pixels.
ppcm	Numeric value or numeric vector (default NULL) of the same length as cm or px specifying how many pixels there are per centimeter. If ppcm is not NULL, ppi is ignored. Examples: 300 ppi (printing) is 118 ppcm, 150 ppi is 59 ppcm, 72 ppi (screens) is 28 ppcm.
ppi	Numeric value or numeric vector (default NULL) of the same length as cm or px specifying how many pixels there are per inch. Set ppcm to NULL to use ppi. Examples: 300 ppi (printing) is 118 ppcm, 150 ppi is 59 ppcm, 72 ppi (screens) is 28 ppcm.
px	Numeric value or numeric vector (pixels) to be converted to centimeters.

Value

cm2px Numeric value or numeric vector (pixels).

px2cm Numeric value or numeric vector (centimeters).

Examples

```
cm2px(2, 4)
px2cm(2, 4)
```

combineStatsWithDL *Get an overview of the given root scans*

Description

combineStatsWithDL - This function goes through a set of image directories and combines all existing statistics_withDepthLayers.csv-files into one data frame, that is optionally saved as a csv.

Usage

```
combineStatsWithDL(data_dir, data_dirs = NULL, out_file = NA)
```

Arguments

data_dir	(Optional, default = NULL) String specifying the name (full path) of the directory containing all image directories of interest.
data_dirs	(Optional, default = NULL) Character vector specifying all of the individual root scan directories of interest. This is only used if data_dir is set to NULL.
out_file	(Optional, default = NA) Full path of the output-csv-file. If set to NULL, no output file is created. If its is set to NA and data_dir is not NULL, then the output file all_statistics_withDepthLayers.csv is created in data_dir.

Value

combineStatsWithDL A data frame containing the combined depth layer information of the images.

Examples

```
combineStatsWithDL(data_dir = NULL, out_file = NULL)
```

createDepthLayerMasks *Create masking layers for the depth layers*

Description

createDepthLayerMasks - Creates a masking layer, i.e., a true/false-matrix, indicating which pixels of an image belong to a specified depth layer.
We assume the image was taken inside a minirhizotron, a round plexiglas tube used to observe roots in the ground, and shows a 360 degree scan of a section of the tube. As the minirhizotron is installed at an angle (can be specified, by default 45 degrees) a depth layer in the ground is shown as a sinus curve in the scan.

dissectionLine - Returns the y-value of the dissection line of a tube that was positioned at an angle for a given position x. The dissection line is dependent on the minpoint, the angle, and the radius of the tube (have to be specified).

Usage

```
createDepthLayerMasks(
  ppcm = NULL,
  ppi = NULL,
  dims_px,
  depth_levels_cm = rbind(c(0, -10), c(-10, -20), c(-20, -30), c(-30, -40)),
  depth_highpoint_cm = 0,
  pos_highpoint_px = "center",
  angle = 45,
  top_side = "top",
  gap_cm = 0,
  gap_deg = 0
)

dissectionLine(x, minpoint, angle, radius)
```

Arguments

ppcm	Numeric value specifying how many pixels there are per centimeter (resolution of the image). Default NULL. If ppcm is not NULL or NA, ppi is ignored. Examples: 300 ppi (printing) is 118 ppcm, 150 ppi is 59 ppcm, 72 ppi (screens) is 28 ppcm.
ppi	Numeric value specifying how many pixels there are per inch (resolution of the image). Default NULL. Leave/set ppcm to NULL or NA to use ppi. Examples: 300 ppi (printing) is 118 ppcm, 150 ppi is 59 ppcm, 72 ppi (screens) is 28 ppcm.
dims_px	Integer vector of length two specifying the dimensions of the image/mask layer (in pixels).
depth_levels_cm	Numeric matrix with two columns and at least one row. Each row specifies a depth interval (in cm) that is of interest. For each interval a masking layer is generated. The first value of each row has to be larger than the second (decending). By default: Depth layers Depth layers 0-(-10)cm, -10-(-20)cm, -20-(-30)cm, and -30-(-40)cm.
depth_highpoint_cm	Numeric value specifying the depth at which the highest point of the image is (in cm). By default this is at depth 0. More precisely, this refers to the depth of the middle of the pixel in the first row of the top_side of the image.
pos_highpoint_px	Either one of "center" or "edge" (default), indicating that scanning starts at the bottom or top facing side of the minirhizotron, respectively, or it can also be an integer value (>=1 and <=width of the top_side of the image) specifying where (left<->right) in the top row of the image the highest point of the image lies, indicating the column where the minimum points of the depth-level lines lie.

	"edge" is equivalent to using 1 or width of the top_side and "center" to using half the width of the top_side. See also gap_cm if there is a non-zero gap.
angle	Numeric value >=0 and <=90 (default 45) specifying the installation angle of the minirhizotron in degrees (angle between the ground and the tube above the soil).
top_side	One of "left","top" (default),"right","bottom". Indicates where the upper side of the image is.
gap_cm	Numeric value (default 0) that specifies if there is a gap (in cm) between the two "matching" sides of the image, i.e., there is no full rotation of the scanner. If pos_highpoint_px is specified as "center" or "edge" (in characters), then it is assumed that the middle of the gap is right at the top or bottom facing side of the tube, i.e. it is not possible to start and end at the exact top/bottom. If pos_highpoint_px is specified as a numeric value, then it is considered as exact and the gap does not move it.
gap_deg	Numeric value (default 0) that specifies if there is a gap (in degrees, i.e. from 0 to 360) between the two "matching" sides of the image, i.e., there is no full rotation of the scanner. Works similar to gap_cm. Example: If the scanner scans only 355 degrees, then set the value to 5. gap_deg is only used if gap_cm is NA.
x	Numeric value indicating the position.
minpoint	Numeric vector of length 2 indicating the coordinates of the minimum point of the function.
radius	Positive numeric value indicating the radius of the tube.

Value

createDepthLayerMasks A list of 2-dimensional true/false-matrices with the specified dimensions (see dims). True indicates that the corresponding pixel belongs to the specified depth layer. The list contains nrow(depth_levels_cm)-many matrices, one for each specified depth layer.

dissectionLine

Examples

```
# Small example for creating a masking layer of a single depth interval:
createDepthLayerMasks(ppcm = 1, dims_px = c(10,10),
                      depth_levels_cm = matrix(c(-3,-5), ncol = 2))[[1]]
# Change some parameters to adapt to a different minimum point position and
# installation angle.
createDepthLayerMasks(ppcm = 1, dims_px = c(5,7),
                      depth_levels_cm = matrix(c(-1.5,-2.5), ncol = 2),
                      pos_highpoint_px = "center", gap_cm = 0)[[1]]
# And here is a cutout of that mask using a gap.
createDepthLayerMasks(ppcm = 1, dims_px = c(5,5),
                      depth_levels_cm = matrix(c(-1.5,-2.5), ncol = 2),
                      pos_highpoint_px = "center", gap_cm = 2)[[1]]
# Examples how different angles result in different dissection graphs:
test_x <- -12:12
plot(test_x, sapply(X=test_x, function(X){
```

```
                dissectionLine(x=X, minpoint = c(0,0), angle = 45,
                               radius = 3)}),
     ylim = c(0, 20))
plot(test_x, sapply(X=test_x, function(X){
                dissectionLine(x=X, minpoint = c(0,0), angle = 80,
                               radius = 3)}),
     ylim = c(0, 20))
plot(test_x, sapply(X=test_x, function(X){
                dissectionLine(x=X, minpoint = c(0,0), angle = 10,
                               radius = 3)}),
     ylim = c(0, 20))
```

`findAvgOverlap` *Find average image overlaps*

Description

`findAvgOverlap` - This function looks at several sets of images, which overlap in the same way, for example the root scans (with more than one depth window) of the same minirhizotron. It then computes the best overlap between each pair of consecutive images (see parameters for more details on the method) for each set of images. The best results are then found for each image transition across all image sets depending based on the correlation values.

Usage

```
findAvgOverlap(
  list_img_paths = NULL,
  list_imgs = NULL,
  overlap_px = NULL,
  max_shift_px = NULL,
  perc_better_per_px = 0.01,
  corr_formula = "1-rel_eucl_diff_colors",
  stitch_direction = "left_to_right"
)
```

Arguments

`list_img_paths`	(Optional, default = NULL) List containing several character vector specifying all of the individual image paths of interest. Each vector must contain the same number of image paths with the images all having the same dimensions. This is only used if `list_imgs` is set to NULL.
`list_imgs`	List of lists of images (e.g., provided by the RootDetector). Each sublist must contain the same number of images with the same dimensions. Each image can be a PNG, i.e., an array with 3 dimensions (3 layers each containing a 2-dim. numeric matrix with values between 0 and 1), or a 2-dim. matrix.

`overlap_px` Numeric vector (default NULL) specifying the (likely) widths of the overlaps between two consecutive images. The vector must have one element less than there are images and must not contain any negative values. If NULL, it is set to 1's.

`max_shift_px` Numeric vector (default NULL) specifying the maximal deviation in pixels from the `overlap_px`, i.e., all possible overlaps in that range from the likely overlap are compared and the best is chosen if it is better than the `overlap_px` (see also `perc_better_per_px`). If `overlap_px` is already exact, then set `max_shift_px` to zero(s). If at NULL, it is set to 10 percent of the image widths.

`perc_better_per_px`
Numeric value (percentage, default 0.01, i.e., 1 percent) specifying how much better the correlation of an overlap must be than the `overlap_px` per pixel difference, to be selected instead. See `findOverlap()` for more details. The corresponding threshold correlation values are only reported, but do not affect the computation of the average/best overlaps.

`corr_formula` Character value specifying the formula to be used (by default 1) for calculating how good an overlap of two images is, i.e., how similar the two overlapping images are. Available are the following formulas:

- "frac_matches_rgb_intensities": Fraction of matching intensities over all three color channels. Only suitable for images with few unique colors. Ranges from 0 (no matches) to 1 (full match).
- "frac_matches_colors": Fraction of matching colors in the images. Only suitable for images with few unique colors. Ranges from 0 (no matches) to 1 (full match).
- "weighted_matches_b_w": Counts the matches of black and white pixels and weighs them anti-proportional to the black and white pixel frequencies. Both black and white make up half of the correlation score. Only suitable for images with mostly pure black and white pixels. Ranges from 0 (no matches) to 1 (full match).
- "weighted_matches_colors": Counts the matches per unique color and weighs them anti-proportional to the frequencies of the colors. Each unique color makes up the same fraction of the correlation score. Only suitable for images with few unique colors. Ranges from 0 (no matches) to 1 (full match).
- "1-rel_sqrd_diff_rgb_intensities": One minus the relative squared difference of intensities across all color channels. Ranges from 0 (no matches) to 1 (full match).
- "1-rel_abs_diff_rgb_intensities": One minus the relative absolute difference of intensities across all color channels. Ranges from 0 (no matches) to 1 (full match).

- "1-rel_eucl_diff_colors" (default): One minus the relative Euclidean differences of colors. Ranges from 0 (no matches) to 1 (full match).

`stitch_direction`
: Character specifying in which order the images should be stitched. Available are: 'left_to_right' (default), 'right_to_left', 'top_to_bottom', and 'bottom_to_top'.

Value

`findAvgOverlap` List containing the best overlaps, i.e., a numeric vector containing the best (according to the parameters) widths of the overlaps between the consecutive images (this vector has one element less than there are images) as well as a list containing all possible overlap information per image set.

Examples

```
# Example of finding the best overlap of two sets of two matrices.
overlap <- findAvgOverlap(list_imgs = list(
                            list(matrix(c(1,0,0,0,
                                          0,1,0,0,
                                          0,0,1,1), ncol = 4, nrow = 3,
                                   byrow = TRUE),
                                 matrix(c(0,0,0,0,
                                          1,0,0,1,
                                          0,1,1,0), ncol = 4, nrow = 3,
                                   byrow = TRUE)),
                            list(matrix(c(0,0,0,0,
                                          0,0,0,0,
                                          0,0,1,1), ncol = 4, nrow = 3,
                                   byrow = TRUE),
                                 matrix(c(0,0,0,0,
                                          0,0,0,0,
                                          1,1,1,0), ncol = 4, nrow = 3,
                                   byrow = TRUE))),
                          overlap_px = 1, max_shift_px = 2,
                          corr_formula = "weighted_matches_b_w")
overlap$best_overlaps
```

`findAvgSurface` *Find the average surface level*

Description

`findAvgSurface` - This function looks at several images which should have the same surface level, for example the root scans (topmost depth window) of the same minirhizotron. Then, the best position of the surface level is determined by finding the best across all images.

Usage

```
findAvgSurface(
  img_paths = NULL,
  imgs = NULL,
  surface_col = "red",
  pos_highpoint_px = "center",
  radius_highpoint_px = 10,
  angle = 45,
  top_side = "left",
  ppcm = NULL,
  ppi = NULL
)
```

Arguments

img_paths	(Optional, default = NULL) Character vector specifying all of the individual image paths of interest. This is only used if imgs is set to NULL.
imgs	List of images (e.g., provided by the RootDetector). Each image can be a PNG, i.e., an array with 3 dimensions (3 layers each containing a 2-dim. numeric matrix with values between 0 and 1), or a 2-dim. matrix.
surface_col	Color of the area to be split of the mask (default "red"). Can be of any of the three kinds of R color specifications, i.e., either a color name (as listed by colors()), a hexadecimal string (see Details), or a positive integer (indicates using palette()).
pos_highpoint_px	Either one of "center" or "edge" (default), indicating that scanning starts at the bottom or top facing side of the minirhizotron, respectively, or it can also be an integer value (>=1 and <=width of the top_side of the image) specifying where (left<->right) in the top row of the image the highest point of the image lies, indicating the column where the minimum points of the depth-level lines lie. "edge" is equivalent to using 1 or width of the top_side and "center" to using half the width of the top_side.
radius_highpoint_px	The radius specifying how large the are should be to determine the split (default 10).
angle	Numeric value >=0 and <=90 (default 45) specifying the installation angle of the minirhizotron in degrees (angle between the ground and the tube above the soil).
top_side	One of "left","top" (default),"right","bottom". Indicates where the upper side of the image is.
ppcm	Numeric value specifying how many pixels there are per centimeter (resolution of the image). Default NULL. If ppcm is not NULL or NA, ppi is ignored. Examples: 300 ppi (printing) is 118 ppcm, 150 ppi is 59 ppcm, 72 ppi (screens) is 28 ppcm.
ppi	Numeric value specifying how many pixels there are per inch (resolution of the image). Default NULL. Leave/set ppcm to NULL or NA to use ppi.

Examples: 300 ppi (printing) is 118 ppcm, 150 ppi is 59 ppcm, 72 ppi (screens) is 28 ppcm.

Value

`findAvgSurface` A list with the two values 'pos_surf_px' and 'depth_highpoint_cm', and a matrix 'all_split_info' containing the information on all possible splits.

Examples

```
# Example of finding the best overlap of two sets of two matrices.
mat_R <- matrix(c(1,1,1,1,1,
                  0,1,1,1,1,
                  0,0,1,1,1), ncol = 5, nrow = 3, byrow = TRUE)
mat_G <- matrix(c(0,0,0,0,1,
                  0,1,0,0,1,
                  0,0,1,1,1), ncol = 5, nrow = 3, byrow = TRUE)
mat_G2 <- matrix(c(0,0,0,1,0,
                   0,0,0,0,0,
                   0,0,1,1,1), ncol = 5, nrow = 3, byrow = TRUE)
mat_B <- matrix(c(0,0,0,0,1,
                  0,0,0,0,1,
                  0,0,1,1,1), ncol = 5, nrow = 3, byrow = TRUE)
mat_B2 <- matrix(c(0,0,0,0,0,
                   0,1,0,0,1,
                   0,0,1,1,1), ncol = 5, nrow = 3, byrow = TRUE)
surface <- findAvgSurface(imgs = list(
                  array(c(mat_R,mat_G,mat_B), dim = c(dim(mat_R),3)),
                  array(c(mat_R,mat_G2,mat_B), dim = c(dim(mat_R),3)),
                  array(c(mat_R,mat_G,mat_B2), dim = c(dim(mat_R),3))),
                         radius_highpoint_px = 2, top_side = "top", ppcm = 1)
```

`findOverlap` *Find image overlaps*

Description

`findOverlap` - This function searches for the best overlap between each pair of consecutive images (see parameters for more details on the method).

`imageCorr` - This function computes the similarity/correlation of two images.

Usage

```
findOverlap(
  img_paths = NULL,
  imgs = NULL,
  overlap_px = NULL,
  max_shift_px = NULL,
  perc_better_per_px = 0.01,
```

```
  corr_formula = "1-rel_eucl_diff_colors",
  stitch_direction = "left_to_right",
  return_all_results = FALSE,
  show_messages = TRUE
)

imageCorr(
  img_paths = NULL,
  imgs = NULL,
  corr_formula = "1-rel_eucl_diff_colors"
)
```

Arguments

`img_paths`	(Optional, default = NULL) Character vector specifying all of the individual image paths of interest. This is only used if `imgs` is set to NULL. For `ImageCorr()` it must have length 2.
`imgs`	List of images (e.g., provided by the RootDetector). Each image can be a PNG, i.e., an array with 3 dimensions (3 layers each containing a 2-dim. numeric matrix with values between 0 and 1), or a 2-dim. matrix. For `ImageCorr()` it must have length 2.
`overlap_px`	Numeric vector (default NULL) specifying the (likely) widths of the overlaps between two consecutive images. The vector must have one element less than there are images and must not contain any negative values. If NULL, it is set to 1's.
`max_shift_px`	Numeric vector (default NULL) specifying the maximal deviation in pixels from the `overlap_px`, i.e., all possible overlaps in that range from the likely overlap are compared and the best is chosen if it is better than the `overlap_px` (see also `perc_better_per_px`). If `overlap_px` is already exact, then set `max_shift_px` to zero(s). If at NULL, it is set to 5 percent of the image widths.
`perc_better_per_px`	Numeric value (percentage, default 0.01, i.e., 1 percent) specifying how much better the correlation of an overlap must be than the `overlap_px` per pixel difference, to be selected instead. If set to 0, the overall best correlation determines the overlap. Example: If set to 0.01 = 1 percent, an overlap being 4 pixels away from the specified `overlap_px` must have a correlation better/higher than 1.01^4 times the correlation value of `overlap_px` to be chosen as the better overlap.
`corr_formula`	Character value specifying the formula to be used (by default 1) for calculating how good an overlap of two images is, i.e., how similar the two overlapping images are. Available are the following formulas: • "frac_matches_rgb_intensities": Fraction of matching intensities over all three color channels. Only suitable for images with few unique colors. Ranges from 0 (no matches) to 1 (full match). • "frac_matches_colors": Fraction of matching colors in the images. Only suitable for images with few unique colors. Ranges from 0 (no matches) to

1 (full match).

- "weighted_matches_b_w": Counts the matches of black and white pixels and weighs them anti-proportional to the black and white pixel frequencies. Both black and white make up half of the correlation score. Only suitable for images with mostly pure black and white pixels. Ranges from 0 (no matches) to 1 (full match).

- "weighted_matches_colors": Counts the matches per unique color and weighs them anti-proportional to the frequencies of the colors. Each unique color makes up the same fraction of the correlation score. Only suitable for images with few unique colors. Ranges from 0 (no matches) to 1 (full match).

- "1-rel_sqrd_diff_rgb_intensities": One minus the relative squared difference of intensities across all color channels. Ranges from 0 (no matches) to 1 (full match).

- "1-rel_abs_diff_rgb_intensities": One minus the relative absolute difference of intensities across all color channels. Ranges from 0 (no matches) to 1 (full match).

- "1-rel_eucl_diff_colors" (default): One minus the relative Euclidean differences of colors. Ranges from 0 (no matches) to 1 (full match).

`stitch_direction`
: Character specifying in which order the images should be stitched. Available are: 'left_to_right' (default), 'right_to_left', 'top_to_bottom', and 'bottom_to_top'.

`return_all_results`
: Specify if all checked overlaps with their respective correlation score should be returned (default FALSE).

`show_messages`
: Specify if messages should be depicted (default TRUE).

Value

`findOverlap` Numeric vector containing the best (according to the parameters) widths of the overlaps between the consecutive images. The vector has one element less than there are images. Its attribute "corr" holds the respective correlation value. If `return_all_results` is set to true, a list containing a 3-column-matrix for each element in the above mentioned vector, is returned, i.e, for each transition of two consecutive images we have the first column containing the overlaps in pixel, the second column holding the respective correlation values, and the third holding the threshold correlation values accoriding to `overlap_px` and `perc_better_per_px`.

`imageCorr` Numeric value (correlations score).

Examples

```
# Example of finding the best overlap of two matrices.
overlap <- findOverlap(imgs = list(matrix(c(1,0,0,0,
                                            0,1,0,0,
```

```
                                        0,0,1,1), ncol = 4, nrow = 3,
                                   byrow = TRUE),
                                matrix(c(0,0,0,0,
                                        1,0,0,1,
                                        0,1,1,0), ncol = 4, nrow = 3,
                                   byrow = TRUE)),
                         overlap_px = 1, max_shift_px = 2,
                         return_all_results = TRUE)
# Example of computing the similarity of two images/matrices.
imageCorr(imgs = list(matrix(c(1,0,0,1,
                               1,1,1,1,
                               0,1,1,1), ncol = 4, nrow = 3, byrow = TRUE),
                      matrix(c(1,0,0,0,
                               1,1,1,1,
                               0,1,1,1), ncol = 4, nrow = 3, byrow = TRUE)),
                      corr_formula = "weighted_matches_colors")
```

`findSurface` *Find the surface level and split images*

Description

`findSurface` - This function finds the best split of an image with a horizontal or vertical line into one color (often red, e.g., for the surface) and other colors (often black and white, e.g., for everything in the ground). This works for any images. For the application on minirhizotron root scans the function provides the additional functionality of using the installation angle of the minirhizotron and the resolution of the image to not only return the position of the line in pixels, but also the corresponding depth in cm of the highest point of the top side of the image, also called `depth_highpoint_cm` in other function of this package.

`splitImgHoriz` - This function finds the best split of an image with a horizontal line into one color on the top half (often red, e.g., for the surface) and other colors on the bottom half (often black and white, e.g., for everything in the ground). This works for any images.
Currently, the best split is determined as the one where there are the most equal mismatches on both sides, i.e., the same number of pixels in the top part of the image which have not the specified color as pixels in the bottom part of the image which have this specific color.

Usage

```
findSurface(
  img_path = NULL,
  img = NULL,
  surface_col = "red",
  pos_highpoint_px = "center",
  radius_highpoint_px = 10,
  angle = 45,
  top_side = "left",
  ppcm = NULL,
  ppi = NULL,
```

```
  return_all_results = FALSE
)

splitImgHoriz(img, surface_col = "red", return_all_results = FALSE)
```

Arguments

`img_path`	(Optional, default = NULL) Character vector specifying the image path of interest. This is only used if `img` is set to NULL.
`img`	Image (e.g., provided by the RootDetector) in the form of an array with 3 dimensions for the RGB color channels (3 layers each containing a 2-dim. numeric matrix with values between 0 and 1).
`surface_col`	Color of the area to be split of the mask (default "red"). Can be of any of the three kinds of R color specifications, i.e., either a color name (as listed by colors()), a hexadecimal string (see Details), or a positive integer (indicates using palette()).
`pos_highpoint_px`	Either one of "center" or "edge" (default), indicating that scanning starts at the bottom or top facing side of the minirhizotron, respectively, or it can also be an integer value (>=1 and <=width of the `top_side` of the image) specifying where (left<->right) in the top row of the image the highest point of the image lies, indicating the column where the minimum points of the depth-level lines lie. "edge" is equivalent to using 1 or width of the `top_side` and "center" to using half the width of the `top_side`.
`radius_highpoint_px`	The radius specifying how large the are should be to determine the split (default 10).
`angle`	Numeric value >=0 and <=90 (default 45) specifying the installation angle of the minirhizotron in degrees (angle between the ground and the tube above the soil).
`top_side`	One of "left","top" (default),"right","bottom". Indicates where the upper side of the image is.
`ppcm`	Numeric value specifying how many pixels there are per centimeter (resolution of the image). Default NULL. If `ppcm` is not NULL or NA, `ppi` is ignored. Examples: 300 ppi (printing) is 118 ppcm, 150 ppi is 59 ppcm, 72 ppi (screens) is 28 ppcm.
`ppi`	Numeric value specifying how many pixels there are per inch (resolution of the image). Default NULL. Leave/set `ppcm` to NULL or NA to use `ppi`. Examples: 300 ppi (printing) is 118 ppcm, 150 ppi is 59 ppcm, 72 ppi (screens) is 28 ppcm.
`return_all_results`	Specify if all checked overlaps with their respective correlation score should be returned (default FALSE).

Value

findSurface A list with following features: 'pos_surf_px' and 'depth_highpoint_cm'. If return_all_results is set to true, a list also comprises a matrix 'split_info' containing the information on all possible splits.

splitImgHoriz An integer value. The row where the upper "surface" part of the image starts (the row included). Called 'pos_surf_px' in the return value of the function findSurface(). If return_all_results is set to true, a list containing this best value 'best_split' as well as a matrix 'split_info' containing the information on all possible splits is returned instead.

Examples

```
# Example of finding the best row to horizontally split the image at the
# highpoint position.
mat_R <- matrix(c(1,1,1,1,1,
                  0,1,1,1,1,
                  0,0,1,1,1), ncol = 5, nrow = 3, byrow = TRUE)
mat_G <- matrix(c(0,0,0,0,1,
                  0,1,0,0,1,
                  0,0,1,1,1), ncol = 5, nrow = 3, byrow = TRUE)
mat_B <- matrix(c(0,0,0,0,1,
                  0,1,0,0,1,
                  0,0,1,1,1), ncol = 5, nrow = 3, byrow = TRUE)
findSurface(img = array(c(mat_R,mat_G,mat_B), dim = c(dim(mat_R),3)),
            radius_highpoint_px = 1, top_side = "top", ppcm = 1,
            return_all_results = TRUE)
# Example of finding the best row to horizontally split the image.
# Note that the top row and one pixel in the second row is red. All others
# are either black or white.
mat_R <- matrix(c(1,1,1,1,
                  0,1,0,1,
                  0,0,1,1), ncol = 4, nrow = 3, byrow = TRUE)
mat_G <- matrix(c(0,0,0,0,
                  0,1,0,0,
                  0,0,1,1), ncol = 4, nrow = 3, byrow = TRUE)
mat_B <- matrix(c(0,0,0,0,
                  0,1,0,0,
                  0,0,1,1), ncol = 4, nrow = 3, byrow = TRUE)
splitImgHoriz(img = array(c(mat_R,mat_G,mat_B), dim = c(dim(mat_R),3)),
              return_all_results = TRUE)
```

---

getDepthLayerInfo *Extract depth layer information from root scans*

Description

getDepthLayerInfo - Specify a set of root scan directories in the form of a data frame (e.g., by using getOverviewInput()). Then, the function will do the following:

- If there are several depth windows of the same root scan (project, tube, date and session have to be identical), then these will be stitched and saved in directory of the scan of depth window 1.

- Create depth layer masks corresponding to the specified `depth_levels_cm`.

- Computes a range of values from the scans (per depth layer: "pixels_root", "pixels_bg", "skel_pixels_total", "skel_pixels_low3", "skel_pixels_3-7", "skel_pixels_larg7","kimura_length6") and if (`save_csvs` = TRUE) saves them as a csv file in the directory of the scan of depth window 1.

- Returns a data frame containing values listed above of all scans.

Usage

```
getDepthLayerInfo(
  root_df = NULL,
  depth_levels_cm = rbind(c(0, -10), c(-10, -20), c(-20, -30), c(-30, -40)),
  overwrite = FALSE,
  save_csvs = TRUE,
  save_pngs_col = "grey"
)
```

Arguments

`root_df` Data frame (default = NULL) specifying the structure of the root scan data. It can be created using the function `getOverviewInput()`. The data frame must contain several mandatory features and can comprise other optional features, some of which have a functional importance and others have no influence but will be included in the output data frame). Any feature not listed here is treated as completely optional:

- ———— Mandatory ————

- `dir_name_full`: Full path to directory.

- `dir_name`: name of directory.

- `depth_window`: depth-level/window. 1 indicates the scan highest scan, 2 the next scan somewhat lower in ground, and so forth.

- ——— Optional, but functionally important ———

- `top_side`: One of "left","top","right","bottom". Indicates where the upper side of the image is. If not specified, "left" is used.

- tube_angle: Installation angle of the tube (angle between surface plane and tube above ground, in degrees, >0 and <90). If not specified, an angle of 45 degrees is assumed.

- depth_highpoint_cm: Depth in cm of the highest point in the image, e.g., 0 indicating that the top border of the scan scratched the surface of the soil or -5 indicating that it lies 5 cm deep in the soil. This is mostly important for all scans of depth_window 1 (or the lowest number if the depth windows are not 1,2,3,4,...) if all images are stitched together. If not specified, it is assumed that the top border of scan of depth_window 1 lies at depth 0.

- pos_highpoint_px: Either one of "center" or "edge", indicating that scanning starts at the bottom or top facing side of the minirhizotron, respectively, or it can also be an integer value (>=1 and <=width of the top_side of the image) specifying where (left<->right) in the top row of the image the highest point of the image lies, indicating the column where the minimum points of the depth-level lines lie.
  If not specified, "center" is assumed as scanning often starts at the bottom facing side of the minirzhizotron.
  See also gap_cm if there is a non-zero gap.

- ppcm Pixels per centimeter, the resolution of the image. If not specified (NULL or NA), ppi is used. Common resolutions are 300 ppi (or dpi, for print jobs) which is 118 px/cm, 150 ppi which is 59 px/cm, and 72 ppi (for screens) which is 28 px/cm.

- ppi Pixels per inch, the resolution of the image. If ppcm and ppi arenot specified, 300 ppi is used. Common resolutions are 300 ppi (or dpi, for print jobs) which is 118 px/cm, 150 ppi which is 59 px/cm, and 72 ppi (for screens) which is 28 px/cm.

- overlap_px Overlap in pixels of the current image and the image of the previous depth window for the stitching process, i.e., this is not important for the image of the first depth window. While stitching it will be checked if there is a better overlap in a small area around the specified value (see max_shift_px). If not specified it is set to 1's.

- max_shift_px Radius around overlap_px that is checked for better overlap matching. If not specified it is set to 0 (no alternatives are checked).

- gap_cm Numeric value that specifies if there is a gap (in cm) between the two "matching" sides of the image, i.e., there is no full rotation of the scanner. If pos_highpoint_px is specified as "center" or "edge" (in characters), then it is assumed that the middle of the gap is right at the top or bottom facing side of the tube, i.e. it is not possible to start and end at the exact top/bottom. If pos_highpoint_px is specified as a numeric value, then it is considered as exact and the gap does not move it.

If not specified and if gap_deg is NA as well, no gap (0) is assumed.

- gap_deg Numeric value that specifies if there is a gap (in degrees, i.e. from 0 to 360) between the two "matching" sides of the image, i.e., there is no full rotation of the scanner. Works similar to gap_cm. Only used if gap_cm is not specified.
- project: Project name. Needed to identify scans that should be stitched.
- tube: ID of the minirhizotron. Needed to identify scans that should be stitched.
- date: Date of the scanning. Needed to identify scans that should be stitched.
- session: ID of the scan session. Needed to identify scans that should be stitched.
- ———— Completely optional ————
- ID: ID of the scan (6 digits) or timecode.
- operator: ID of the person that scanned the root.
- file_extension: File format: png, tiff, jpg, or jpeg (upper or lowercase).

depth_levels_cm
: Numeric matrix with two columns and at least one row. Each row specifies a depth interval (in cm) that is of interest. For each interval a masking layer is generated. The first value of each row has to be larger than the second (decending).
By default: Depth layers 0-(-10)cm, -10-(-20)cm, -20-(-30)cm, and -30-(-40)cm.

overwrite
: If FALSE (default), root scan directories are skipped that have already been processed by a particular procedure, i.e., there already are stitched images before beginning the stitching procedure, there already are images with depth layers drawn in before saving thme or there already is a statistics_withDepthLayers.csv file when it comes to evaluating the image. If TRUE, all procedures are applied to the scan and corresponding files are overwritten.

save_csvs
: Specifies if csv files for the individual scans should be saved in the scan's directory (default TRUE).

save_pngs_col
: If not NULL, the root scans with depth layers drawn in are saved in the scan's directory. Specifies the color of the FALSE-regions of the mask (default "grey"). Can be a vector of the same length as the number of depth intervals to allow each layer to be drawn in with a different color.
Can be of any of the three kinds of R color specifications, i.e., either a color name (as listed by colors()), a hexadecimal string (see Details), or a positive integer (indicates using palette()).

Value

getDepthLayerInfo A data frame containing the depth layer information of all individual scans.

Examples

```
getDepthLayerInfo()
```

---

getDepthLayerInfo_par *Extract depth layer information from root scans (parallel)*

Description

getDepthLayerInfo_par - Specify a set of root scan directories in the form of a data frame (e.g., by using getOverviewInput()). Then, the function will do the following:

- If there are several depth windows of the same root scan (project, tube, date and session have to be identical), then these will be stitched and saved in directory of the scan of depth window 1.
- Create depth layer masks corresponding to the specified depth_levels_cm.
- Computes a range of values from the scans (per depth layer: "pixels_root", "pixels_bg", "skel_pixels_total", "skel_pixels_low3", "skel_pixels_3-7", "skel_pixels_larg7","kimura_length6") and if (save_csvs = TRUE) saves them as a csv file in the directory of the scan of depth window 1.
- Returns a data frame containing values listed above of all scans.

Usage

```
getDepthLayerInfo_par(
  root_df = NULL,
  depth_levels_cm = rbind(c(0, -10), c(-10, -20), c(-20, -30), c(-30, -40)),
  overwrite = FALSE,
  save_csvs = TRUE,
  save_pngs_col = "grey",
  core_number = 1L
)
```

Arguments

root_df	Data frame (default = NULL) specifying the structure of the root scan data. It can be created using the function getOverviewInput(). The data frame must contain several mandatory features and can comprise other optional features, some of which have a functional importance and others have no influence but

will be included in the output data frame). Any feature not listed here is treated as completely optional:

- ———————— Mandatory ————————
- `dir_name_full`: Full path to directory.
- `dir_name`: name of directory.
- `depth_window`: depth-level/window. 1 indicates the scan highest scan, 2 the next scan somewhat lower in ground, and so forth.
- ——— Optional, but functionally important ———
- `top_side`: One of "left","top","right","bottom". Indicates where the upper side of the image is. If not specified, "left" is used.
- `tube_angle`: Installation angle of the tube (angle between surface plane and tube above ground, in degrees, >0 and <90). If not specified, an angle of 45 degrees is assumed.
- `depth_highpoint_cm`: Depth in cm of the highest point in the image, e.g., 0 indicating that the top border of the scan scratched the surface of the soil or -5 indicating that it lies 5 cm deep in the soil. This is mostly important for all scans of `depth_window` 1 (or the lowest number if the depth windows are not 1,2,3,4,...) if all images are stitched together. If not specified, it is assumed that the top border of scan of `depth_window` 1 lies at depth 0.
- `pos_highpoint_px`: Either one of "center" or "edge", indicating that scanning starts at the bottom or top facing side of the minirhizotron, respectively, or it can also be an integer value (>=1 and <=width of the `top_side` of the image) specifying where (left<->right) in the top row of the image the highest point of the image lies, indicating the column where the minimum points of the depth-level lines lie.
  If not specified, "center" is assumed as scanning often starts at the bottom facing side of the minirzhizotron.
  See also `gap_cm` if there is a non-zero gap.
- `ppcm` Pixels per centimeter, the resolution of the image. If not specified (NULL or NA), `ppi` is used. Common resolutions are 300 ppi (or dpi, for print jobs) which is 118 px/cm, 150 ppi which is 59 px/cm, and 72 ppi (for screens) which is 28 px/cm.
- `ppi` Pixels per inch, the resolution of the image. If `ppcm` and `ppi` arenot specified, 300 ppi is used. Common resolutions are 300 ppi (or dpi, for print jobs) which is 118 px/cm, 150 ppi which is 59 px/cm, and 72 ppi (for

screens) which is 28 px/cm.

- `overlap_px` Overlap in pixels of the current image and the image of the previous depth window for the stitching process, i.e., this is not important for the image of the first depth window. While stitching it will be checked if there is a better overlap in a small area around the specified value (see `max_shift_px`). If not specified it is set to 1's.

- `max_shift_px` Radius around `overlap_px` that is checked for better overlap matching. If not specified it is set to 0 (no alternatives are checked).

- `gap_cm` Numeric value that specifies if there is a gap (in cm) between the two "matching" sides of the image, i.e., there is no full rotation of the scanner. If `pos_highpoint_px` is specified as "center" or "edge" (in characters), then it is assumed that the middle of the gap is right at the top or bottom facing side of the tube, i.e. it is not possible to start and end at the exact top/bottom. If `pos_highpoint_px` is specified as a numeric value, then it is considered as exact and the gap does not move it.
  If not specified and if `gap_deg` is NA as well, no gap (0) is assumed.

- `gap_deg` Numeric value that specifies if there is a gap (in degrees, i.e. from 0 to 360) between the two "matching" sides of the image, i.e., there is no full rotation of the scanner. Works similar to `gap_cm`. Only used if `gap_cm` is not specified.

- `project`: Project name. Needed to identify scans that should be stitched.

- `tube`: ID of the minirhizotron. Needed to identify scans that should be stitched.

- `date`: Date of the scanning. Needed to identify scans that should be stitched.

- `session`: ID of the scan session. Needed to identify scans that should be stitched.

- ———— Completely optional ————

- `ID`: ID of the scan (6 digits) or timecode.

- `operator`: ID of the person that scanned the root.

- `file_extension`: File format: png, tiff, jpg, or jpeg (upper or lowercase).

`depth_levels_cm`

Numeric matrix with two columns and at least one row. Each row specifies a depth interval (in cm) that is of interest. For each interval a masking layer is generated. The first value of each row has to be larger than the second (decend-

	ing). By default: Depth layers 0-(-10)cm, -10-(-20)cm, -20-(-30)cm, and -30-(-40)cm.
overwrite	If FALSE (default), root scan directories are skipped that have already been processed by a particular procedure, i.e., there already are stitched images before beginning the stitching procedure, there already are images with depth layers drawn in before saving thme or there already is a statistics_withDepthLayers.csv file when it comes to evaluating the image. If TRUE, all procedures are applied to the scan and corresponding files are overwritten.
save_csvs	Specifies if csv files for the individual scans should be saved in the scan's directory (default TRUE).
save_pngs_col	If not NULL, the root scans with depth layers drawn in are saved in the scan's directory. Specifies the color of the FALSE-regions of the mask (default "grey"). Can be a vector of the same length as the number of depth intervals to allow each layer to be drawn in with a different color. Can be of any of the three kinds of R color specifications, i.e., either a color name (as listed by colors()), a hexadecimal string (see Details), or a positive integer (indicates using palette()).
core_number	Integer value specifying the bumber of cores (default = 1) used in parallelized procedures.

Value

getDepthLayerInfo_par A data frame containing the depth layer information of all individual scans.

Examples

```
getDepthLayerInfo_par()
```

---

getNeighborCoords *Get coordinates of neighbors of specified distance*

Description

getNeighborCoords - Returns coordinates of all pixels in a two dimensional raster/image/... with a specified Euclidean distance from the center, i.e., a pixel circle, either a ring or the whole area with a specified radius.

Usage

```
getNeighborCoords(
  center,
  radius,
  dims_px = c(1000, 1000),
  type = "ring",
  tol = "strict",
  tol_val = 0.5
)
```

Arguments

center	Numeric vector of length two specifying the coordinates (in pixels) of the center point.
radius	Integer value specifying the radius (in pixels). A radius of 0 returns only the center point itself.
dims_px	Numeric vector of length two (default: 1,000 x 1,000) specifying the dimensions (in pixels). Neighbors outside are ignored.
type	Character, "ring" (default) or "area", specifying if the pixel coordinates of only the outer ring or of the whole circle area should be returned.
tol	Character, "strict" (default) or "loose", specifying how thick the ring/outer edge of the area should be, i.e., if pixels close but not exactly on the circle should be included. If set to "loose", all pixels that touch the circle are included. If set to "strict", only additional pixels whose centers have a distance of at most tol_val to the circle are included.
tol_val	Numeric value specifying the tolerance value (>=0,<1, default: 0.5), i.e., pixels are considered neighbors if the difference between their Euclidean distance to the center and the radius is less than or equal to the tolerance value. Only applies if tol is set to "strict". A radius of 1 with a tolerance of 0.5 returns all 8 surrounding pixels and with a tolerance of 0 only the 4 orthogonally neighboring pixels.

Value

getNeighborCoords Numeric matrix with two columns containing the neighbors' coordinates (in pixels).

Examples

```
# The neighbors with radius 1 of point (4,4) in an 8x8 grid.
# With tolerance 0.5:
test <- matrix(0, nrow = 8, ncol = 8)
test[getNeighborCoords(c(4,4), 1, c(8,8))] <- 1
test
# With tolerance 0.4:
test <- matrix(0, nrow = 8, ncol = 8)
test[getNeighborCoords(c(4,4), 1, c(8,8), tol_val = 0.4)] <- 1
test
```

---

getOverviewInput *Get an overview of the given root scans*

Description

getOverviewInput - This function filters a set of root scan directories by checking if they comply with the given naming convention and then returns overview data about these directories.

Usage

```
getOverviewInput(data_dir, data_dirs = NULL, naming_conv = "standard")
```

Arguments

data_dir	(Optional, default = NULL) String specifying the name (full path) of the directory containing all root scan directories of interest.
data_dirs	(Optional, default = NULL) Character vector specifying all of the individual root scan directories of interest. This is only used if data_dir is set to NULL.
naming_conv	A string specifying the naming convention, i.e., what information is provided within the names of the root scans. The file format can be png, tiff, jpg, or jpeg (upper or lowercase). Available are:

- "standard" (default): This is a commonly used naming convention and has the following structure:
  project_tube_depth_date_ID_session_operator
  Example:
  Testproject_T007_L004_12.12.2025_123394_016_Testoperator.jpg
  Explanation of the abbreviations:
  – project: Project name ('unlimited' letters or digits)
  – tube: ID of the minirhizotron ("T"+3 digits)
  – depth: ID of the depth-level/window ("L"+3 digits)
  – date: Date of the scanning (format day.month.year, 2 digits +"."+ 2 digits +"."+ 2 or 4 digits)
  – ID: ID of the scan (6 digits) or timecode 14:23:10 -> 142310
  – session: ID of the scan session (3 digits).
  – operator: ID of the person that scanned the root ('unlimited' letters or digits)

Value

getOverviewInput A data frame containing the information about the various root scan directories (see also getDepthLvlInfo() for further explanations).

Examples

```
getOverviewInput(data_dir = NULL, naming_conv = "standard")
```

---

kimuraLength *Computation of the Kimura length*

Description

kimuraLength - Computes the Kimura length according to the specified formula to estimate the root length shown in a skeletonized black & white image of a root.
Optionally, a masking layer can be specified that indicates which pixels of the image should be considered.

Usage

```
kimuraLength(
  skel_img,
  formula = 6,
  mask = NULL,
  strict_mask = TRUE,
  show_messages = TRUE
)
```

Arguments

skel_img	Skeleton image (provided by the RootDetector). Can be a PNG, i.e., an array with 3 dimensions (3 layers each containing a 2-dim. numeric matrix with values between 0 and 1), or a 2-dim. matrix.
formula	Integer value specifying the formula to be used (by default 6) based on Kimura et al., 1999 (see references). Available are the following formulas, where N_o denotes the number of orthogonal and N_d the number of diagonal connections (between white pixels), and $N_{whitepx}$ the number of white pixels in the image. Furthermore, there are three additional formulas to simply count white, or black, or red pixels.

- Formula 1: $1.1107 \cdot (N_d + N_o)$,
- Formula 2: $\sqrt{2} \cdot N_d + N_o$,
- Formula 3: $0.948 \cdot (\sqrt{2} \cdot N_d + N_o)$,
- Formula 4: $\sqrt{N_d^2 + (N_d + N_o)^2}$,
- Formula 6: $\sqrt{N_d^2 + (N_d + N_o/2)^2} + N_o/2$,
- Formula 7: $1.1107 \cdot N_{whitepx}$.
- Formula 97: $N_{whitepx}$.
- Formula 98: $N_{blackpx}$.
- Formula 99: N_{redpx}.

mask	2-dim. true/false-matrix with the same number of rows and columns as skel_img (optional, default = NULL, interpreted as a matrix consisting only of TRUEs, i.e., nothing is "removed" from the image).
strict_mask	Specifies how strictly the mask should be applied. Available are:

- TRUE (default): Connections between TRUE-pixels and neighboring FALSE-pixels are not counted. As a result, the root length is likely to be underestimated, especially if there are many TRUE-FALSE transitions in the mask.

- FALSE: Connections between TRUE-pixels and neighboring FALSE-pixels are counted. As a result, the root length is likely to be overestimated, especially if there are many TRUE-FALSE transitions in the mask.

`show_messages` Specify if messages about the counts of orthogonal and diagonal connection counts should be depicted (default TRUE).

Value

`kimuraLength` Numeric value (root length estimation).

References

Kimura, K., Kikuchi, S. & Yamasaki, Si. Accurate root length measurement by image analysis. Plant and Soil 216, 117–127 (1999). doi: 10.1023/A:1004778925316

Examples

```
# This is a simple image with 2 diagonal and 1 orthogonal connections.
# With Formula 6 (default):
kimuraLength(matrix(c(
  1, 0, 0, 0,
  0, 1, 0, 0,
  0, 0, 1, 1
), ncol = 4, nrow = 3, byrow = TRUE))
# With Formula 4:
kimuraLength(
  matrix(c(
    1, 0, 0, 0,
    0, 1, 0, 0,
    0, 0, 1, 1
  ), ncol = 4, nrow = 3, byrow = TRUE),
  formula = 4
)
# With Formula 6 and a mask which makes the function ignore the right side
# of the image. If stict_mask = TRUE, only 1 diagonal connection can be
# found. If set to FALSE, i.e., relaxed mask borders, then 2 diagonal
# connections are counted.
kimuraLength(
  skel_img = matrix(c(
    1, 0, 0, 0,
    0, 1, 0, 0,
    0, 0, 1, 1
  ), ncol = 4, nrow = 3, byrow = TRUE),
  mask = matrix(
    c(
      TRUE, TRUE, FALSE, FALSE,
      TRUE, TRUE, FALSE, FALSE,
      TRUE, TRUE, FALSE, FALSE
    ),
    ncol = 4, nrow = 3,
    byrow = TRUE
  ), strict_mask = FALSE
```

```
)
```

---

ppcm2ppi *Functions for converting resolutions: ppi and ppcm*

Description

ppcm2ppi - Converts a resolution given in pixels per cm to pixels per inch.

ppi2ppcm - Converts a resolution given in pixels per inch to pixels per cm.

Usage

```
ppcm2ppi(ppcm)

ppi2ppcm(ppi)
```

Arguments

ppcm	Numeric value or numeric vector of resolutions in ppcm to be converted to ppi.
ppi	Numeric value or numeric vector of resolutions in ppi to be converted to ppcm.

Value

ppcm2ppi Numeric value or numeric vector of resolutions in ppi.

ppi2ppcm Numeric value or numeric vector of resolutions in ppcm.

Examples

```
ppcm2ppi(2)
ppi2ppcm(2)
```

---

showImgMask *Depict images with masking layers*

Description

showImgMask - Depicts an image, e.g., a root scan, with an optional masking layer using a specified color.

applyImgMask - Changes the colouring of an image, e.g., a root scan, according to a masking layer using a specified color. If only a matrix is provided, it is treated like the r-layer (referring to the rgb color values).

Usage

```
showImgMask(root_img = NULL, mask = NULL, mask_col = "grey")

applyImgMask(root_img = NULL, mask = NULL, mask_col = "grey")
```

Arguments

root_img	Root image. Can be a PNG, i.e., an array with 3 dimensions (3 layers each containing a 2-dim. numeric matrix with values between 0 and 1), or a 2-dim. matrix. If the root image is NULL (default), then a completely black image is used.
mask	2-dim. true/false-matrix with the same number of rows and columns as skel_img (optional, default = NULL, interpreted as a matrix consisting only of TRUEs, i.e., nothing is "removed" from the image).
mask_col	Color of the FALSE-regions of the mask (default "grey"). Can be of any of the three kinds of R color specifications, i.e., either a color name (as listed by colors()), a hexadecimal string (see Details), or a positive integer (indicates using palette()).

Value

showImgMask No return value, called for side effects (plotting).

applyImgMask An array with 3 dimensions (3 layers each containing a 2-dim. numeric matrix with values between 0 and 1) like a PNG.

Examples

```
# Basic example:
showImgMask(matrix(c(1,0,0,0,
                     0,1,0,0,
                     0,0,1,1), ncol = 4, nrow = 3, byrow = TRUE),
            matrix(c(TRUE, FALSE,FALSE,TRUE,
                     TRUE, TRUE, TRUE, TRUE,
                     FALSE,TRUE, TRUE, FALSE), ncol = 4, nrow = 3,
                     byrow = TRUE))
# Example using the createDepthLayerMasks()-function:.
showImgMask(root_img = matrix(c(1,0,0,0,0,0,0,
                     0,1,0,0,0,0,0,
                     0,0,1,1,0,0,0), ncol = 7, nrow = 3, byrow = TRUE),
            mask = createDepthLayerMasks(ppcm = 1,
                         dims_px = c(3,7),
                         depth_levels_cm = rbind(c(-1,-2), c(-2,-3)))[[1]],
            mask_col ="brown")
# Basic example:
applyImgMask(matrix(c(1,0,0,0,
                     0,1,0,0,
                     0,0,1,1), ncol = 4, nrow = 3, byrow = TRUE),
            matrix(c(TRUE, FALSE,FALSE,TRUE,
                     TRUE, TRUE, TRUE, TRUE,
                     FALSE,TRUE, TRUE, FALSE), ncol = 4, nrow = 3,
```

```
                    byrow = TRUE))
```

`skelPxWidth` *Count skeleton root pixels with certain widths in the base root image*

Description

`skelPxWidth` - Counts the skeleton pixels that belong to root pieces of certain (circular) widths (diameter <3px, 3-7px, >7px).
For each white root pixel in the skeleton image, it checks how large (in terms of the categories <3px, 3-7px, >7px, which equal a radius of <1px, 1-3px, >=4px) the circle of white root pixels is that surrounds it in the base root image.
Optionally, a masking layer can be specified that indicates which pixels of the skeleton image should be checked (the radius of the white circle is still check in the complete base image).

Usage

```
skelPxWidth(base_img, skel_img, mask = NULL)
```

Arguments

`base_img`	Base image (provided by the RootDetector). Can be a PNG, i.e., an array with 3 dimensions (3 layers each containing a 2-dim. numeric matrix with values between 0 and 1), or a 2-dim. matrix.
`skel_img`	Skeleton image with the same number of rows and columns as `base_img` (provided by the RootDetector). Can be a PNG, i.e., an array with 3 dimensions (3 layers each containing a 2-dim. numeric matrix with values between 0 and 1), or a 2-dim. matrix.
`mask`	2-dim. true/false-matrix with the same number of rows and columns as `base_img` (optional, default = NULL, interpreted as a matrix consisting only of TRUEs, i.e., nothing is "removed" from the image).

Value

`skelPxWidth` Numeric vector of length 3 containing the counts.

Examples

```
# In this example there are 2 white root pixels in the skeleton image.
# The left one is completely surrounded by white pixels in the base image,
# it falls into Category 2 (3-7px). The bottom right one has a black
# neighboring pixel and thus falls in to Category 1 (<3px). Thus, the result
# is Categorie 1: 1 pixel, Cat. 2: 1 pixel, Cat. 3: 0 pixels.
skelPxWidth(base_img = matrix(c(1,1,1,0,
                      1,1,1,0,
                      1,1,1,1), ncol = 4, nrow = 3, byrow = TRUE),
            skel_img = matrix(c(0,0,0,0,
                      0,1,0,0,
```

```
                    0,0,0,1), ncol = 4, nrow = 3, byrow = TRUE))
# Similar example with a mask which makes the function "ignore" the right
# side of the skeleton image.
# The function still identifies the left white pixel as of Category 2.
skelPxWidth(matrix(c(1,1,1,0,
                     1,1,1,0,
                     1,1,1,1), ncol = 4, nrow = 3, byrow = TRUE),
            matrix(c(0,0,0,0,
                     0,1,0,0,
                     0,0,0,1), ncol = 4, nrow = 3, byrow = TRUE),
             matrix(c(TRUE,TRUE,FALSE,FALSE,
                      TRUE,TRUE,FALSE,FALSE,
                      TRUE,TRUE,FALSE,FALSE), ncol = 4, nrow = 3,
                      byrow = TRUE))
```

`stitchImgs` *Stitch images*

Description

`stitchImgs` - This function stitches all specified images (currently only PNGs supported) together in a line. It also searches for the best overlap between two consecutive images (see parameters and `findOverlap()`). If `overlap_px` is already known, this function calls `blendImgs()`.

`blendImgs` - This function stitches all specified images together in a line according to specified overlap widths.

Usage

```
stitchImgs(
  img_paths = NULL,
  imgs = NULL,
  out_file = NULL,
  overlap_px = NULL,
  max_shift_px = NULL,
  perc_better_per_px = 0.01,
  corr_formula = "1-rel_eucl_diff_colors",
  stitch_direction = "left_to_right",
  blending_mode = "under",
  show_messages = TRUE
)

blendImgs(
  img_paths = NULL,
  imgs = NULL,
  out_file = NULL,
  overlap_px,
  stitch_direction = "left_to_right",
  blending_mode = "under"
)
```

Arguments

`img_paths` (Optional, default = NULL) Character vector specifying all of the individual image paths of interest. This is only used if `imgs` is set to NULL. For `ImageCorr()` it must have length 2.

`imgs` List of images (e.g., provided by the RootDetector). Each image can be a PNG, i.e., an array with 3 dimensions (3 layers each containing a 2-dim. numeric matrix with values between 0 and 1), or a 2-dim. matrix. For `ImageCorr()` it must have length 2.

`out_file` Full path for how the stitched image should be saved, e.g. 'C:/path/stitched.png'. If no path is provided, the image is not saved (only returned).

`overlap_px` Numeric vector (default NULL) specifying the (likely) widths of the overlaps between two consecutive images. The vector must have one element less than there are images and must not contain any negative values.

`max_shift_px` Numeric vector (default NULL) specifying the maximal deviation in pixels from the `overlap_px`, i.e., all possible overlaps in that range from the likely overlap are compared and the best is chosen if it is better than the `overlap_px` (see also `perc_better_per_px`). If `overlap_px` is already exact, then set `max_shift_px` to zero(s). If at NULL, it is set to 10 percent of the image.

`perc_better_per_px`
Numeric value (percentage, default 0.01, i.e., 1 percent) specifying how much better the correlation of an overlap must be than the `overlap_px` per pixel difference, to be selected instead. If set to 0, the overall best correlation determines the overlap. Example: If set to 0.01 = 1 percent, an overlap being 4 pixels away from the specified `overlap_px` must have a correlation 4*1 percent better than `overlap_px` to be chosen as the better overlap.

`corr_formula` Character value specifying the formula to be used (by default 1) for calculating how good an overlap of two images is, i.e., how similar the two overlapping images are. Available are the following formulas:

- "frac_matches_rgb_intensities": Fraction of matching intensities over all three color channels. Only suitable for images with few unique colors. Ranges from 0 (no matches) to 1 (full match).
- "frac_matches_colors": Fraction of matching colors in the images. Only suitable for images with few unique colors. Ranges from 0 (no matches) to 1 (full match).
- "weighted_matches_b_w": Counts the matches of black and white pixels and weighs them anti-proportional to the black and white pixel frequencies. Both black and white make up half of the correlation score. Only suitable for images with mostly pure black and white pixels. Ranges from 0 (no matches) to 1 (full match).
- "weighted_matches_colors": Counts the matches per unique color and weighs them anti-proportional to the frequencies of the colors. Each unique color

makes up the same fraction of the correlation score. Only suitable for images with few unique colors. Ranges from 0 (no matches) to 1 (full match).

- "1-rel_sqrd_diff_rgb_intensities": One minus the relative squared difference of intensities across all color channels. Ranges from 0 (no matches) to 1 (full match).
- "1-rel_abs_diff_rgb_intensities": One minus the relative absolute difference of intensities across all color channels. Ranges from 0 (no matches) to 1 (full match).
- "1-rel_eucl_diff_colors" (default): One minus the relative Euclidean differences of colors. Ranges from 0 (no matches) to 1 (full match).

`stitch_direction`
: Character specifying in which order the images should be stitched. Available are: 'left_to_right' (default), 'right_to_left', 'top_to_bottom', and 'bottom_to_top'.

`blending_mode`
: Character value specifying how overlapping pixels are combined. Available are:

- "under" (default): The first image(s) dominate(s), and only non-overlapping parts of further images contribute.
- "over": The next image completely replaces the previous image in the overlap.
- "average": The RGB values of overlapping regions are averaged.
- "max": The maximal RGB values from both images are chosen.
- "min": The minimal RGB values from both images are chosen.

`show_messages`
: Specify if messages should be depicted (default TRUE).

Value

`stitchImgs` The stitched image in the form of a PNG, i.e., an array with 3 dimensions (3 layers each containing a 2-dim. numeric matrix with values between 0 and 1).

`blendImgs` The stitched image in the form of a PNG, i.e., an array with 3 dimensions (3 layers each containing a 2-dim. numeric matrix with values between 0 and 1).

Examples

```
# Small example of stitching two 3x4 matrices left to right together.
test <- stitchImgs(imgs = list(matrix(c(1,0,0,0,
                                        0,1,0,0,
                                        0,0,1,1), ncol = 4, nrow = 3,
                                      byrow = TRUE),
                               matrix(c(0,0,0,0,
                                        1,0,0,1,
```

```
                                      0,1,1,0), ncol = 4, nrow = 3,
                                    byrow = TRUE)),
           overlap_px = 1, max_shift_px = 2)
# The stitched image also contains the overlaps used for stitching.
attributes(test)
# Example of stitching the two matrices with a fixed (in this case not
# optimal) overlap.
blendImgs(imgs = list(matrix(c(1,0,0,0,
                               0,1,0,0,
                               0,0,1,1), ncol = 4, nrow = 3, byrow = TRUE),
                      matrix(c(0,0,0,0,
                               1,0,0,1,
                               0,1,1,0), ncol = 4, nrow = 3, byrow = TRUE)),
          overlap_px = 1, blending_mode = "over")[,,1]
```

Index