Package 'image.textlinedetector'

November 10, 2023

Type Package

Title Segment Images in Text Lines and Words

Version 0.2.3

Maintainer Jan Wijffels <jwijffels@bnosac.be>

Description Find text lines in scanned images and segment the lines into words.
Includes implementations of the paper 'Novel A* Path Planning Algorithm for Line Segmentation of Handwritten Documents' by Surinta O. et al (2014) <doi:10.1109/ICFHR.2014.37> available at <https://github.com/smeucci/LineSegm>,
an implementation of 'A Statistical approach to line segmentation in handwritten documents' by Arivazhagan M. et al (2007) <doi:10.1117/12.704538>,
and a wrapper for an image segmentation technique to detect words in text lines as described in the paper 'Scale Space Technique for Word Segmentation in Handwritten Documents' by Manmatha R. and Srimal N. (1999) paper at <doi:10.1007/3-540-48236-9_3>, wrapper for code available at <https://github.com/arthurflor23/text-segmentation>.
Provides as well functionality to put cursive text in images upright using the approach defined in the paper 'A new normalization technique for cursive handwritten words' by Vinciarelli A. and Luettin J. (2001) <doi:10.1016/S0167-8655(01)00042-3>.

License MIT + file LICENSE

URL https://github.com/DIGI-VUB/image.textlinedetector

Encoding UTF-8

Imports Rcpp (>= 0.12.9), magick

Suggests opencv

LinkingTo Rcpp

SystemRequirements C++17 or C++11 and OpenCV 3 or newer: libopencv-dev (Debian, Ubuntu) or opencv-devel (Fedora)

RoxygenNote 7.1.2

NeedsCompilation yes

Author Jan Wijffels [aut, cre, cph] (R wrapper),
Vrije Universiteit Brussel - DIGI: Brussels Platform for Digital Humanities [cph] (R wrapper),
Jeroen Ooms [ctb, cph] (More details in LICENSE.note file),

Arthur Flôr [ctb, cph] (More details in LICENSE.note file),
Saverio Meucci [ctb, cph] (More details in LICENSE.note file),
Yeara Kozlov [ctb, cph] (More details in LICENSE.note file),
Tino Weinkauf [ctb, cph] (More details in LICENSE.note file),
Harald Scheidl [ctb, cph] (More details in LICENSE.note file)

Repository CRAN

Date/Publication 2023-11-09 23:20:02 UTC

R topics documented:

image_textlines_astar 2
image_textlines_crop 3
image_textlines_flor 4
image_wordsegmentation 5
lines.textlines 6
ocv_deslant 7

Index **9**

`image_textlines_astar` *Text Line Segmentation based on the A* Path Planning Algorithm*

Description

Text Line Segmentation based on the A* Path Planning Algorithm

Usage

```
image_textlines_astar(x, morph = FALSE, step = 2, mfactor = 5, trace = FALSE)
```

Arguments

`x`	an object of class magick-image
`morph`	logical indicating to apply a morphological 5x5 filter
`step`	step size of A-star
`mfactor`	multiplication factor in the cost heuristic of the A-star algorithm
`trace`	logical indicating to show the evolution of the line detection

Value

a list with elements

- n: the number of lines found
- overview: an opencv-image of the detected areas
- paths: a list of data.frame's with the x/y location of the baseline paths
- textlines: a list of opencv-image's, one for each rectangular text line area
- lines: a data.frame with the x/y positions of the detected lines

Examples

```
library(opencv)
library(magick)
library(image.textlinedetector)
path   <- system.file(package = "image.textlinedetector", "extdata", "example.png")
img    <- image_read(path)
img    <- image_resize(img, "x1000")
areas  <- image_textlines_astar(img, morph = TRUE, step = 2, mfactor = 5, trace = TRUE)
areas  <- lines(areas, img)
areas$n
areas$overview
areas$lines
areas$textlines[[2]]
areas$textlines[[4]]
combined <- lapply(areas$textlines, FUN=function(x) image_read(ocv_bitmap(x)))
combined <- do.call(c, combined)
combined
image_append(combined, stack = TRUE)



plt <- image_draw(img)
lapply(areas$paths, FUN=function(line){
  lines(x = line$x, y = line$y, col = "red")
})
dev.off()
plt
```

---

image_textlines_crop *Crop an image to extract only the region containing text*

Description

Applies a sequence of image operations to obtain a region which contains relevant texts by cropping white space on the borders of the image. This is done in the following steps: morphological opening, morphological closing, blurring, canny edge detection, convex hull contours of the edges, keep only contours above the mean contour area, find approximated contour lines of the convex hull contours of these, dilation and thresholding.

Usage

```
image_textlines_crop(x)
```

Arguments

x	an object of class magick-image

Value

an object of class magick-image

Examples

```
library(opencv)
library(magick)
library(image.textlinedetector)
path  <- system.file(package = "image.textlinedetector", "extdata", "example.png")
img   <- image_read(path)
image_info(img)
img   <- image_textlines_crop(img)
image_info(img)
```

---

image_textlines_flor *Text Line Segmentation based on valley finding in projection profiles*

Description

Text Line Segmentation based on valley finding in projection profiles

Usage

```
image_textlines_flor(
  x,
  light = TRUE,
  type = c("none", "niblack", "sauvola", "wolf")
)
```

Arguments

x	an object of class magick-image
light	logical indicating to remove light effects due to scanning
type	which type of binarisation to perform before doing line segmentation

Value

a list with elements

- n: the number of lines found
- overview: an opencv-image of the detected areas
- textlines: a list of opencv-image's, one for each text line area

Examples

```
library(opencv)
library(magick)
library(image.textlinedetector)
path   <- system.file(package = "image.textlinedetector", "extdata", "example.png")
img    <- image_read(path)
img    <- image_resize(img, "1000x")
areas  <- image_textlines_flor(img, light = TRUE, type = "sauvola")
areas  <- lines(areas, img)
areas$n
areas$overview
combined <- lapply(areas$textlines, FUN=function(x) image_read(ocv_bitmap(x)))
combined <- do.call(c, combined)
combined
image_append(combined, stack = TRUE)
```

---

image_wordsegmentation

Find Words by Connected Components Labelling

Description

Filter the image using the gaussian kernel and extract components which are connected which are to be considered as words.

Usage

```
image_wordsegmentation(x, kernelSize = 11L, sigma = 11L, theta = 7L)
```

Arguments

x	an object of class opencv-image containing black/white binary data (type CV_8U1)
kernelSize	size of the kernel
sigma	sigma of the kernel
theta	theta of the kernel

Value

a list with elements

- n: the number of lines found
- overview: an opencv-image of the detected areas
- words: a list of opencv-image's, one for each word area

Examples

```
library(opencv)
library(magick)
library(image.textlinedetector)
path  <- system.file(package = "image.textlinedetector", "extdata", "example.png")
img   <- image_read(path)
img   <- image_resize(img, "x1000")
areas <- image_textlines_flor(img, light = TRUE, type = "sauvola")
areas$overview
areas$textlines[[6]]
textwords <- image_wordsegmentation(areas$textlines[[6]])
textwords$n
textwords$overview
textwords$words[[2]]
textwords$words[[3]]
```

---

lines.textlines *Extract the polygons of the textlines*

Description

Extract the polygons of the textlines as a cropped rectangular image containing the image content of the line segmented polygon

Usage

```
## S3 method for class 'textlines'
lines(x, image, crop = TRUE, channels = c("bgr", "gray"), ...)
```

Arguments

x	an object of class textlines as returned by image_textlines_astar or image_textlines_flor
image	an object of class magick-image
crop	extract only the bounding box of the polygon of the text lines
channels	either 'bgr' or 'gray' to work on the colored data or on binary greyscale data
...	further arguments passed on

Value

the object x where element textlines is replaced with the extracted polygons of text lines

Examples

```
## See the examples in ?image_textlines_astar or ?image_textlines_flor
```

ocv_deslant *Deslant images by putting cursive text upright*

Description

This algorithm sets handwritten text in images upright by removing cursive writing style. One can use it as a preprocessing step for handwritten text recognition.

- `image_deslant` expects a magick-image and performs grayscaling before doing deslanting
- `ocv_deslant` expects a ocv-image and does not perform grayscaling before doing deslanting

Usage

```
ocv_deslant(image, bgcolor = 255, lower_bound = -1, upper_bound = 1)

image_deslant(image, bgcolor = 255, lower_bound = -1, upper_bound = 1)
```

Arguments

`image`	an object of class opencv-image (for `ocv_deslant`) with pixel values between 0 and 255 or a magick-image (for `image_deslant`)
`bgcolor`	integer value with the background color to use to fill the gaps of the sheared image that is returned. Defaults to white: 255
`lower_bound`	lower bound of shear values. Defaults to -1
`upper_bound`	upper bound of shear values. Defaults to 1

Value

an object of class opencv-image or magick-image with the deslanted image

Examples

```
library(magick)
library(opencv)
library(image.textlinedetector)
path <- system.file(package = "image.textlinedetector", "extdata", "cursive.png")
img  <- ocv_read(path)
img  <- ocv_grayscale(img)
img
up   <- ocv_deslant(img)
up

img  <- image_read(path)
img
image_deslant(img)
```

Index

image_deslant *(*ocv_deslant*)*, 7
image_textlines_astar, 2, *6*
image_textlines_crop, 3
image_textlines_flor, 4, *6*
image_wordsegmentation, 5

lines.textlines, 6

ocv_deslant, 7