OrationesPython Documentation

Release 1.0.1

Timo Mätäsaho

December 13, 2013

CONTENTS

Contents:

CHAPTER

ONE

DOCUMENTATION FOR THE CODE

1.1 Orationes Search

`osearch.`**`init`**(*sysargv*)

The first function to be called when this script is run. Checks if the system arguments are correct and returns error when needed.

When calling from PHP use the $output and $return_var arguments to catch the correct output and possible return error code. If the $return_var is 1 then the program has finished without errors.

Refer to PHP exec command manual for further information of calling Python scripts and programs in general from PHP.

`osearch.`**`osearch`**(*img*, *switch*, *txtf*, *sw*)

The main program that only calls the processing methods from OratUtils and HFun classes.

Parameters

- **img** (*string*) – The name of the image.
- **txtf** (*string*) – The name of the cleaned XML file.
- **sw** (*string*) – The word or letter that is searched.

Returns error code or 1 and JSON string

The error codes are: 2 - Error in starting parameters 3 - Given image doesn't exist, it's path is faulty or its format is wrong or either the image file is corrupted. 4 - Given string or character to be searched cannot be found from the XML 5 - 6 - 7 - 8 - 9 - Unknown error while opening the image

There are still lots of places that are missing error handling!

The return option have to be chosen between returning the string as a return code or is it just printed out for the calling PHP program. Currently it is being printed.

1.2 Orationes getBoxes

`getBoxes.`**`getBoxesAndLines`**(*img*)

Optional directly callable program that can be used to extract the bounding box and line location information from an image.

Parameters **img** (*string*) – The name of the image.

Returns JSON string

Returns a JSON array containing the possible locations of the text lines and bounding boxes.

1.3 Utility functions

class `OratUtils.`**`OratUtils`**

This class contains only static utility methods that are called directly from the main program 'osearch.py'.

static **`boundingBox`** (*image*, *debug*)

This functions tries to determine the bounding boxes for each text line.

Parameters

- **image** (*ndarray*) – the processed image
- **debug** (*bool*) – debug switch

Returns ndarray – bboxes

$$bboxes_{nxm} = \begin{bmatrix} \text{patch label numbers} \\ \text{starting x-coordinates} \\ \text{starting y-coordinates} \\ \text{ending x-coordinates} \\ \text{ending y-coordinates} \end{bmatrix}$$

debug switch can be used to plot the results of the bounding box founding method and to see whether it is working correctly.

Process pipeline:

1.Calculate the histogram from the image

2.Binarize image with threshold 0.95

3.Label all the patched in on the binarized image

4.Calculate the sizes of the patches

5.Remove unnecessary patches

(a)Remove the largest patch. The largest patch is always the patch consisting of the borders and marginals.

(b)Remove patches which size is smaller or equal to 50 pixels

(c)Remove all the patches which are higher than 70 pixels. This removes the possible remaining marginal patches which weren't connected to the major marginal and border patch.

6.Perform morpholig operations to clean the image and bind the text lines together

(a)Perform erosion with a cross like structure element

$$SEe_{5,5} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

(b)Perform dilation with a long vertical line. (needs a 70x70 size structure element)

$$SEd_{70,70} = \begin{bmatrix} 0 & 0 & \dots & 0 & 0 \\ & \vdots & & \vdots & \\ 1 & 1 & \dots & 1 & 1 \\ & \vdots & & \vdots & \\ 0 & 0 & \dots & 0 & 0 \end{bmatrix}$$

7.Label the morphologically operated image

8.Remove patches which size is less or equal to 4000 pixels

9.Label the image again with new labels

10.Calculate the extreme dimensions of each patch. These values are used as the limiting bounding boxes.

11.Combine the boxes which are horizontally too close as they are thought to be separate boxes on the same textline.

12.Return the bounding boxes

static `contStretch` (*im*, *a*, *h*)

Performs contrast stretching for grayscale images. Pixel intensities are set to differ 'a' times the average intensity from the original intensity values. The new intensity values are sliced to stay between [0, 255].

$$I_{stretched} = I_{old} + a * (I_{old} - I_{average})$$

$$I_{new} = \begin{cases} 0, & I_{stretched} < 0 \\ I_{stretched}, & 0 \leq I_{stretched} \leq 255 \\ 255, & I_{stretched} > 255 \end{cases}$$

Parameters

- **im** (*ndarray*) – The image which contrast is to be stretched
- **a** (*int*) – multiplication coefficient
- **h** (*int*) – The height of image. Used as partial image average switch

Returns ndarray – contrast stretched image

Parameter *h* is a switch which could be used to determine if the average intensity is calculated over the whole image or from a small portion of it. Currently it is defaulted in the code to newer happen. Originally the idea was that if the image is very big, the intensity average would be taken from a small sample. To make the function more generic and also because of the nature of the images in Orationes project, it was decided that the average is always calculated over the whole image.

static `findCorr` (*bboxes*, *slines*, *charcount*, *imlines*, *debug*)

Used to find out which bounding box and which line are the same.

Parameters

- **bboxes** (*ndarray*) – Ndarray containing the coordinates of the bounding boxes
- **slines** (*ndarray*) – A vector containing the y-coordinates of the interesting lines
- **charcount** (*list*) – Contains the lengths of each line

- **imlines** (*ndarray*) – n*1 size ndarray containing the lines (or rather their y-position) got from the image by radontransform
- **debug** (*bool*) – Debug switch

Returns ndarray – m*n ndarray containing the starting and ending coordinates of hits

static `hfilter` (*image*, *diameter*, *height*, *length*, *n*)
This function performs homomorphic filtering on grayscale images.

Parameters

- **image** (*ndarray*) – 2-dimensional ndarray
- **diameter** (*int*) – filter diameter
- **height** (*int*) – Height of the image
- **length** (*int*) – Length of the image
- **n** (*int*) – Filter order

Returns ndarray – homomorphically filtered image

The image must in ndarray format. In osearch PIL images are converted to scipy images which are in ndarray format. Ndarray format allows easy and fast direct access to the pixel values and this function is written entirely only for the ndarrays.

static `packBoxesAndLines` (*bboxes*, *imlines*)

Parameters

- **bboxes** (*ndarray*) – Ndarray containing the coordinates of the bounding boxes
- **imlines** (*ndarray*) – n*1 size ndarray containing the lines (or rather their y-position) got from the image by radontransform

Returns jsondata – JSON packed string containing the found bounding boxes and lines

static `packCoordsToJson` (*slines*, *origimage*, *coords*, *charpos*, *wordlens*, *bboxes*, *debug*)
This function is used to pack the hit coordinates and bounding box coordinates into a JSON string which is returned to the calling PHP site.

Parameters

- **slines** (*ndarray*) – A vector containing the y-coordinates of the interesting lines
- **origimage** (*ndarray*) – Original image. Used when debugging
- **coords** (*ndarray*) – A ndarray containing the coordinates of the hits
- **charpos** (*list of lists*) – List of the character positions got from the XML
- **wordlens** (*list of lists*) – List of wordlengths
- **bboxes** (*ndarray*) – Ndarray containing the coordinates of the bounding boxes
- **debug** (*bool*) – Debug switch

Returns json-string – JSON packed string containing the hits and the bounding boxes

static `padlines` (*imlines*, *llines*, *charlines*)

Parameters

- **imlines** (*ndarray*) – n*1 size ndarray containing the lines (or rather their y-position) got from the image by radontransform
- **llines** (*ndarray*) – n*2 size ndarray containing the length information of the lines

- **charlines** (*list*) – list of lists telling the position(s) of searched character(s)/word(s) on each line

Returns ndarray – wantedlines

Long: Llines contains the information about the lines got from the XML and also it contains the information if some of the lines are remarkably shorter than other lines. That means that, if there are some lines that are not found from the image, it is assumed that those non-found lines are the shortest lines according to the XML and character count. Those lines are marked as 0 in the second column in llines.

Short: Llines[:,1] contains only 1s and 0s. 1 meaning a line with enough letters to be recognized by poormanradon (pmr) and 0 meaning a line which is probably undetected by pmr

Behavior: Number of lines found from the image using pmr is larger than the number of lines calculated from XML:

TODO! Currently this case is not handled!

Number of lines found from the image using pmr is smaller than the number of lines calculated from XML:

Pad the lines according to the information in *llines[:,1]*:

$$\begin{array}{ccccc}
\text{llines} & & & & \text{rlines}\\
\begin{bmatrix}1\\1\\1\\1\\0\\1\\1\\0\\1\end{bmatrix} &
\begin{array}{c}\searrow\\\searrow\\\searrow\\\searrow\\ \\\longrightarrow\\\longrightarrow\\ \\\nearrow\end{array} &
\begin{array}{c}\text{imlines}\\\begin{bmatrix}100\\200\\300\\400\\600\\700\\900\end{bmatrix}\\ \end{array} &
\begin{array}{c}\nearrow\\\nearrow\\\nearrow\\\nearrow\\ \\\longrightarrow\\\longrightarrow\\ \\\searrow\end{array} &
\begin{bmatrix}100\\200\\300\\400\\NAN\\600\\700\\NAN\\900\end{bmatrix}
\end{array}$$

Number of lines found from the image using pmr equals to the number of lines calculated from XML:

Pick unique lines from imlines and return them as lines the interesting lines.

static **`poormanradon`**(*image*, *iname*, *height*, *debug*)

Performs a naive radon-transform and peak detection on the binarized and contrast stretched image and tries to determine where the text lines are in the image.

Parameters

- **image** (*ndarray*) – Image
- **iname** (*string*) – Image name
- **height** (*int*) – Image height
- **debug** (*bool*) – Debug switch

Returns ndarray – Array containing the lines which are found using radon transform

Calculates the intensity sums over each vertical line. The sums are then inverted and peaks are detected from the inverted data. Data inversion wouldn't be necessary in the python code, but this convetion comes from the Matlab code that was ported to python.

Before the transform, the image is cleaned so that by using static values (very bad, should be dynamic, but so far there hasn't been time to do that) the marginals and everything outside them is erased and turned to white. Because the distance between the camera and the page differs in each image, the marginals aren't always on the same position. This combined with static values causes inaccuracy in the erasing process and might cause inaccuracy when detecting the peaks and the lines.

In the peak detection, it is assumed that a spike is considered a peak if it's 25 units away from a previous detected peak and also if its value difference is at least 1500 to its previous value.

upLim in the source means upper limit in the image coordinates, which increase when going down in the image. That's why *upLim* is small. Respectively the *downLim* means the bottom limit in the image coordinates and that's why it's bigger than the upper limit.

static `processlines` (*charcount*, *imlines*)

This functions compares the number of lines found from the image to the number of lines found from the XML file and creates a logical vector telling which lines are probably found and which are not.

Parameters

- **charcount** (*list*) – Contains the lengths of each line
- **imlines** (*ndarray*) – Contains the textlines which are found in 'poormanradon'

Returns ndarray – llines

Returns ndarray – imlines

llines is a n*2 vector, where the llines[:,1] is a logical vector containing the information of the probably found and non-found lines.

$$[1, 1, 1, 1, 0, 1, 0, 1, \ldots]^t$$

imlines is a ndarray containing the y-coordinates of the textlines found from the image with poormanradon. When padding some of the coordinates are removed (nofound < 0, not used), some NAN values are added in between some coordinates (nofound > 0) or it is returned unchanged (nofound == 0).

We calculate a number 'nofound'

$$\text{nofound} = \text{lines}_{XML} - \text{lines}_{image}$$

Naturally there are three cases.

nofound < 0: This means that there are more lines found from the image than there actually are. Currently nothing's done here to compensate this behavior.

nofound > 0: This means there aren't enough lines found from the image. Usually the non-found lines are assumed to be the very short lines. When padding the indices of the lines, the shortest lines are always set to be the non-found lines.

nofound == 0: It is assumed that all the textlines were found correctly and the imlines will be returned unchanged.

static `stringparser` (*tfile*, *c*)

Performs case sensitive search for text file tfile with string or character c (char on default). Argument c can be any regular expression

Parameters

- **tfile** (*string*) – The string containing the cleaned XML file as a string
- **c** (*string/char/regular expression*) – The letter or string that is searched from the tfile

Returns list – charcount

Returns list of lists – charpos

Returns list of lists – charlines

Returns list of lists – wordlens

•*Charcount* is a list containing the lengths of each line.

–`[63, 60, 4, 65, 66, 37, 66, ...]`

•*Charpos* is a list containing lists including the positions of the found characters or the first letters of the found words.

–`[[52], [10, 47, 62], [19, 62], [51], ...]`

•*Charlines* is a list of lists where the length of each sublist tells the number of hits on that line and the element values representing the line number from the XML file.

–`[[3], [4, 4, 4], [6, 6], [7], ...]`

•*Wordlens* is a list containing lists containing the lengths of the words on each line.

–`[[3], [3, 3, 3], [3, 3], [3], ...]`

static **`txtfparser`** (*tfile*, *c*)

Performs case sensitive search for text file tfile with string or character c (char on default). Argument c can be any regular expression

Parameters

- **tfile** (*string*) – The name of the cleaned XML file
- **c** (*string/char/regular expression*) – The letter or string that is searched from the tfile

Returns list – charcount

Returns list of lists – charpos

Returns list of lists – charlines

Returns list of lists – wordlens

•*Charcount* is a list containing the lengths of each line.

–`[63, 60, 4, 65, 66, 37, 66, ...]`

•*Charpos* is a list containing lists including the positions of the found characters or the first letters of the found words.

–`[[52], [10, 47, 62], [19, 62], [51], ...]`

•*Charlines* is a list of lists where the length of each sublist tells the number of hits on that line and the element values representing the line number from the XML file.

–`[[3], [4, 4, 4], [6, 6], [7], ...]`

•*Wordlens* is a list containing lists containing the lengths of the words on each line.

–`[[3], [3, 3, 3], [3, 3], [3], ...]`

1.4 Helper Functions

class `OratUtils.`**`HFun`**

Contains helper functions that are used in various places

static **`gray2uint8`**(*image*)

Converts grasycale images to uint8 type

Parameters **image** (*ndarray*) – The grayscale image to converted

Returns ndarray(uint8) - I

static **`im2float`**(*image*)

Changes uint8 type images to float64 images.

Parameters **image** (*ndarray(uint8)*) – The image to be converted

Returns ndarray(float64) – I

1.5 Peakdetection

This is the only part not written by me (the author of this program), thus all the glory and honors and what so ever goes to its right authors. Only one function is used. For more info about this module see Sixtenbe's github

NOTE that this file might not follow the Sphinx documentation guidelines and might not appear correctly!

`peakdet.`**`peakdetect`**(*y_axis*, *x_axis=None*, *lookahead=300*, *delta=0*)

Converted from/based on a MATLAB script at: http://billauer.co.il/peakdet.html

function for detecting local maximas and minmias in a signal. Discovers peaks by searching for values which are surrounded by lower or larger values for maximas and minimas respectively

keyword arguments: y_axis – A list containg the signal over which to find peaks x_axis – (optional) A x-axis whose values correspond to the y_axis list

and is used in the return to specify the postion of the peaks. If omitted an index of the y_axis is used. (default: None)

lookahead – (optional) distance to look ahead from a peak candidate to determine if it is the actual peak (default: 200) '(sample / period) / f' where '4 >= f >= 1.25' might be a good value

delta – (optional) this specifies a minimum difference between a peak and the following points, before a peak may be considered a peak. Useful to hinder the function from picking up false peaks towards to end of the signal. To work well delta should be set to delta >= RMSnoise * 5. (default: 0)

delta function causes a 20% decrease in speed, when omitted Correctly used it can double the speed of the function

return – two lists [max_peaks, min_peaks] containing the positive and negative peaks respectively. Each cell of the lists contains a tupple of: (position, peak_value) to get the average peak value do: np.mean(max_peaks, 0)[1] on the results to unpack one of the lists into x, y coordinates do: x, y = zip(*tab)

`peakdet.`**`peakdetect_fft`**(*y_axis*, *x_axis*, *pad_len=5*)

Performs a FFT calculation on the data and zero-pads the results to increase the time domain resolution after performing the inverse fft and send the data to the 'peakdetect' function for peak detection.

Omitting the x_axis is forbidden as it would make the resulting x_axis value silly if it was returned as the index 50.234 or similar.

Will find at least 1 less peak then the 'peakdetect_zero_crossing' function, but should result in a more precise value of the peak as resolution has been increased. Some peaks are lost in an attempt to minimize spectral leakage by calculating the fft between two zero crossings for n amount of signal periods.

The biggest time eater in this function is the ifft and thereafter it's the 'peakdetect' function which takes only half the time of the ifft. Speed improvementd could include to check if 2**n points could be used for fft and ifft or change the 'peakdetect' to the 'peakdetect_zero_crossing', which is maybe 10 times faster than 'peakdetct'. The pro of 'peakdetect' is that it resutls in one less lost peak. It should also be noted that the time used by the ifft function can change greatly depending on the input.

keyword arguments: y_axis – A list containg the signal over which to find peaks x_axis – A x-axis whose values correspond to the y_axis list and is used

in the return to specify the postion of the peaks.

pad_len – (optional) By how many times the time resolution should be increased by, e.g. 1 doubles the resolution. The amount is rounded up to the nearest 2 ** n amount (default: 5)

return – two lists [max_peaks, min_peaks] containing the positive and negative peaks respectively. Each cell of the lists contains a tupple of: (position, peak_value) to get the average peak value do: np.mean(max_peaks, 0)[1] on the results to unpack one of the lists into x, y coordinates do: x, y = zip(*tab)

`peakdet.`**`peakdetect_parabole`**(*y_axis*, *x_axis*, *points=9*)

Function for detecting local maximas and minmias in a signal. Discovers peaks by fitting the model function: y = k (x - tau) ** 2 + m to the peaks. The amount of points used in the fitting is set by the points argument.

Omitting the x_axis is forbidden as it would make the resulting x_axis value silly if it was returned as index 50.234 or similar.

will find the same amount of peaks as the 'peakdetect_zero_crossing' function, but might result in a more precise value of the peak.

keyword arguments: y_axis – A list containg the signal over which to find peaks x_axis – A x-axis whose values correspond to the y_axis list and is used

in the return to specify the postion of the peaks.

points – (optional) How many points around the peak should be used during curve fitting, must be odd (default: 9)

return – two lists [max_peaks, min_peaks] containing the positive and negative peaks respectively. Each cell of the lists contains a list of: (position, peak_value) to get the average peak value do: np.mean(max_peaks, 0)[1] on the results to unpack one of the lists into x, y coordinates do: x, y = zip(*max_peaks)

`peakdet.`**`peakdetect_sine`**(*y_axis*, *x_axis*, *points=9*, *lock_frequency=False*)

Function for detecting local maximas and minmias in a signal. Discovers peaks by fitting the model function: y = A * sin(2 * pi * f * x - tau) to the peaks. The amount of points used in the fitting is set by the points argument.

Omitting the x_axis is forbidden as it would make the resulting x_axis value silly if it was returned as index 50.234 or similar.

will find the same amount of peaks as the 'peakdetect_zero_crossing' function, but might result in a more precise value of the peak.

The function might have some problems if the sine wave has a non-negligible total angle i.e. a k*x component, as this messes with the internal offset calculation of the peaks, might be fixed by fitting a k * x + m function to the peaks for offset calculation.

keyword arguments: y_axis – A list containg the signal over which to find peaks x_axis – A x-axis whose values correspond to the y_axis list and is used

in the return to specify the postion of the peaks.

points – (optional) How many points around the peak should be used during curve fitting, must be odd (default: 9)

lock_frequency – (optional) Specifies if the frequency argument of the model function should be locked to the value calculated from the raw peaks or if optimization process may tinker with it. (default: False)

return – two lists [max_peaks, min_peaks] containing the positive and negative peaks respectively. Each cell of the lists contains a tupple of: (position, peak_value) to get the average peak value do: np.mean(max_peaks, 0)[1] on the results to unpack one of the lists into x, y coordinates do: x, y = zip(*tab)

`peakdet.`**`peakdetect_sine_locked`**(*y_axis*, *x_axis*, *points=9*)

Convinience function for calling the 'peakdetect_sine' function with the lock_frequency argument as True.

keyword arguments: y_axis – A list containg the signal over which to find peaks x_axis – A x-axis whose values correspond to the y_axis list and is used

in the return to specify the postion of the peaks.

points – (optional) How many points around the peak should be used during curve fitting, must be odd (default: 9)

return – see 'peakdetect_sine'

`peakdet.`**`peakdetect_zero_crossing`**(*y_axis*, *x_axis=None*, *window=11*)

Function for detecting local maximas and minmias in a signal. Discovers peaks by dividing the signal into bins and retrieving the maximum and minimum value of each the even and odd bins respectively. Division into bins is performed by smoothing the curve and finding the zero crossings.

Suitable for repeatable signals, where some noise is tolerated. Excecutes faster than 'peakdetect', although this function will break if the offset of the signal is too large. It should also be noted that the first and last peak will probably not be found, as this function only can find peaks between the first and last zero crossing.

keyword arguments: y_axis – A list containg the signal over which to find peaks x_axis – (optional) A x-axis whose values correspond to the y_axis list

and is used in the return to specify the postion of the peaks. If omitted an index of the y_axis is used. (default: None)

window – the dimension of the smoothing window; should be an odd integer (default: 11)

return – two lists [max_peaks, min_peaks] containing the positive and negative peaks respectively. Each cell of the lists contains a tupple of: (position, peak_value) to get the average peak value do: np.mean(max_peaks, 0)[1] on the results to unpack one of the lists into x, y coordinates do: x, y = zip(*tab)

`peakdet.`**`zero_crossings`**(*y_axis*, *window=11*)

Algorithm to find zero crossings. Smoothens the curve and finds the zero-crossings by looking for a sign change.

keyword arguments: y_axis – A list containg the signal over which to find zero-crossings window – the dimension of the smoothing window; should be an odd integer

(default: 11)

return – the index for each zero-crossing

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

INDEX