OULUN YLIOPISTO
UNIVERSITY of OULU

Department
of
Computer Science and Engineering

Timo Mätäsaho

Text search engine for digitized historical book

Master's Thesis
Computer Science and Engineering
April 2015

Mätäsaho T. (2015) Text search engine for digitized historical book. Department of Computer Science and Engineering, University of Oulu, Oulu, Finland. Master's thesis, 50 p.

ABSTRACT

There's need to digitalize numerous historical books and texts and make it possible to read them electronically. Also it is often wanted to preserve their original appearance, not just the text itself. For these operations there is a need for systems, which understand the books and text as they are and are able to distinguish the text information from other context. Traditional optical character recognition systems perform well when processing modern printed text, but they might face problems with old handwritten texts. These types of texts need to be analyzed with systems, which can analyse and segment the text areas well from other irrelevant information. That is why it is important, that the document image segmentation works well. This thesis focuses on manual rectification, automatic segmentation and text line search on document images in Orationes project. When the document images are segmented and text lines found, information from XML transcript is used to find characters and words from the segmented document images. Search engine was developed with with Python programmin language. Python was chosen to ensure high platform independency.

Keywords: optical document image analysis, optical character recognition, document image search engine, image enhancement, system, document image segmentation

Mätäsaho T. (2015) Tekstinhakujärjestelmä digitoidulle historialliselle kirjalle. Oulun yliopisto, Tietotekniikan osasto. Diplomityö, 50 s.

TIIVISTELMÄ

Lukuisia historiallisia kirjoja halutaan digitalisoida ja siirtää sähköisesti luettaviksi. Usein ne halutaan myös säilyttää alkuperäisessä ulkoasussaan. Tällaista operaatiota varten tarvitaan järjestelmiä, jotka osaavat ymmärtää kirjat ja tekstit sellaisinaan ja osaavat erottaa tekstin kirjan muusta kontekstista. Perinteiset optiset kirjaimentunnistusmenetelmät suorituvat hyvin painettujen tekstien analysoinnista, mutta ongelmia aiheuttavat käsinkirjoitetut vanhat tekstit. Tällaisten tekstien kohdalla dokumenttikuvat pitää pystyä ensin analysoimaan hyvin ja erottelemaan tekstialueet muusta tekstin kannalta irrelevantista informaatiosta. Siksi onkin tärkeää, että dokumenttikuvan segmentaatio onnistuu hyvin. Tässä työssä keskitytään Orationes projektin dokumenttikuvien manuaaliseen suoristamiseen, segmentaatioon ja tekstirivien löytämiseen. Lisäksi segmentaation jälkeen segmentoidusta dokumenttikuvasta yritetään löytää haluttuja kirjaimia ja sanoja, dokumenttikuvan XML transkriptista saadun informaation avulla. Hakumoottori toteutettiin Python ohjelmointikielellä, jotta saavutettiin alustariippumattomuus hakumoottorille.

Avainsanat: optinen dokumenttikuvan analyysi, optinen kirjainten tunnistus, dokumenttikuvan hakukone, kuvan parantaminen, järjestelmä, dokumenttikuvan segmentaatio

TABLE OF CONTENTS

FOREWORD

This thesis is done for Orationes project, funded by the Academy of Finland, in the Biosignal processing team research group in University of Oulu. I would like to thank professor Tapio Seppänen for the possibility to do this master's thesis in the university and also especially thank of all the feedback he gave me in the final stages of the thesis and writing process. Also I would like to thank the secondary supervisor and colleague Dr Eero Väyrynen for his many ideas and feedback during the actual development stage and also for many refreshing off topic conversations, which were both entertaining and instructive.

Many thanks goes to my current employer Aava Mobile Inc, which has given me lots of freedom to write this thesis and been flexible with my unexpected needs from time to time to stay off from work and focus on this thesis when needed. Also thanks to my friends who have helped me during this process.

Oulu, Finland April 23, 2015

Timo Mätäsaho

SYMBOLS AND ABBREVIATIONS

3D	3-dimensional
DSLR	Digital Single-Lens Reflex (camera)
EXIF	Exchangeable image file format
GIMP	GNU Image Manipulation Program
ICDAR	International Conference on Document Analysis and Recognition
JSON	JavaScript Object Notation
NAN	Not A Number
OCR	Optical Character Recognition
PHP	PHP Hypertext Preprocessor - server side script language
PMR	Poor Man Radon - simplified version of Radon transform developed for this thesis only
RGB	Red-Green-Blue color space
XML	Extensible Markup Language
C_{XMLpos}	Character location in XML transcript
$P_{X,Y}$	Location of wanted character in page coordinates
N_c	Number of characters on current line
X	x-coordinate on page
X_{1bb}	The leftmost coordinate of a bounding box
X_{2bb}	The rightmost coordinate of a bounding box
Y	y-coordinate on page
Y_{center}	The text line center y-coordinate got from text line search

1. INTRODUCTION

Most of the text humans read nowadays are on the internet and available for electronic reading devices. The texts on electornic form are easier to store, last longer and are less vulnerable to disasters, compared to traditional paper books and publications, especially if they are stored on multiple different locations or even in cloud storage. Also they are a lot easier to browse through with the developed search mechanisms, which one can use to directly search words on certain pages for example. These are some of the reasons why there's an urgent need to transform lots of old printed books into electronic books and publications with computer readable characters and make them easily available for all the scientists and readers all around the world. This can be done with the help of optical character recognition and similar systems.

However there are still lots of books and valuable information only in printed form and especially many important historical texts still only exist in their physical form as books. Should all the texts in the world to be converted to ascii formatted computer readable text, the information from old historical texts would be more accessible to historians and scientist. This would speed up the research projects for example on old languages.

The object of optical character recognition (OCR) is automatic reading of optically sensed document text materials to translate human-readable characters to machine-readable codes. Research in OCR is popular for its various application potentials in banks, post-offices and defense organizations. Other applications involve reading aid for the blind, library automation, language processing and multi-media design.

The history of OCR can be tracked to early 1900s when Russian scientis Tyurin made successful retina scanner to aid visually handicapped patients [1]. OCR took major steps in 1950s when first electronic devices were introduced for reading bank checks. Currently, PC-based systems are commercially available to read printed documents of single font with very high accuracy and documents of multiple fonts with reasonable accuracy [2]. However the documents written by hand cause more difficulties, because of their non-uniform nature.

As the majority of the OCR systems and techniques are focusing on OCR on printed Roman alphabets, some techniques have been developed to recognize other fonts for example Greek alphabets [3] and ancient Bangla scripts [2]. These old writings are usually hand written, so these methods offer great information of how to deal with handwritten texts in general.

The Orationes project includes an old handwritten book full of plays and acts telling a part of history of England. Being able to browse through these stories effectively with the help of computer is essential for historians and other researchers. It is not only necessary to get the character representation of the book, but also to be able to browse through the book as it is, so that the fonts and characters itself are easily accessible. The text itself is important, but also how it looks and how it is written.

In general in document image analysis, lots of different document image enhancement processes are required to make the original document image to be possible to browse with computer. These processes consist of basic digital image manipulation operations and their advanced variations.

The goal of this master's thesis is to develop a method to exploit text from a XML transcript file to a document image and, instead of regular OCR, align the text on the

image and use that alignment to determine the location of characters and words on the document image. This text alignment method could be great help in OCR systems, which tries to recognize characters from hand written texts.

The developed method follows roughly OCR systems and their processing steps, which are usually quite similar to each other independent of the application itself. The classification of these steps is generic and some methods use different steps and in different order.

The steps in this work are:

- Distortion correction
- Pre-processing XML information
- Image enhancement
- Bounding box detection
- Text line search and line processing
- Determining the hitboxes

Imaging the document images and parsing the XML file are part of the whole pipeline, but they are not represented in this work, because this work focuses only on the character location search engine and it is implementation.

This thesis presents a simple basin for a text and character location search and alignment engine. Engine uses techniques familiar from OCR environments, so it could be easier to embed it into a such system. The solution consists of basic image manipulation and signal processing techniques and their applications to the project specific requirements. The engine is developed in MATLAB and further ported to Python programming language to provide better OS independent execution.

This work does not try to improve already developed methods, which are mainly OCR systems, but instead first of all tries to offer necessary tool for historians in Orationes projects to easily browse through the Orationes document images and look for certain characters and words, and secondly to lead to the topic the future readers and researchers who are familiarizing themselves to the problem and point them the possible problem areas and which solutions works and which doesn't.

2. DIGITIZATION OF HAND-WRITTEN TEXT

2.1. General processing steps for document images

Before going deeper into the problem and its solution, it is good to go through and understand some basic processing steps, techniques and theoretical background of image processing and document image analysis.

The analysis of document images in general consists of several different steps that varies depending of which approach the system designer has selected. Mostly though there are couple general steps almost every system has to go through in their process pipeline.

The first step is to get the document images. After that the images usually has to be enhanced so that noise is removed, possible illumination defects are corrected and if the image is severely warped, it needs to be de-warped. After that the image and its textual areas and image areas are separated from each other with segmentation methods and then it's needed to find the locations of the text lines, so that the actual character recognition can be performed [2].

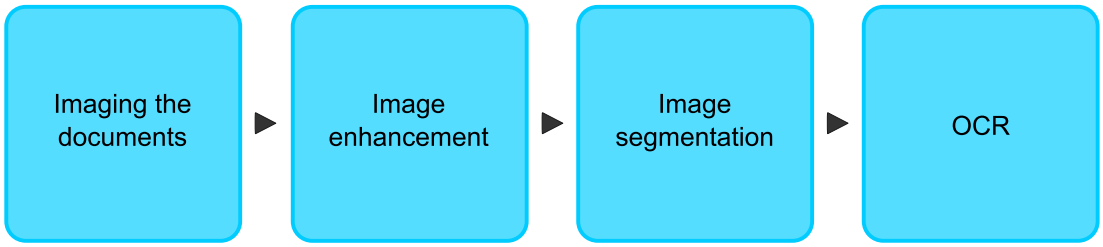


Figure 1. Generic OCR system.

2.1.1. Imaging the document images

There are various methods of capturing the document images, but mostly especially in the case of flat single page documents, the document images are usually attained by imaging them with a flat bed scanner. Sometimes the flat-bed scanners are also used to capture document images from books, though that method leaves some distortion to the images [2].

In some cases it is useful to take the images with a camera attached above the document that is to be captured. This method is gentle on old books and manuscripts as they are let to rest without any external stress that might affect them, if they were photographed with a flat-bed scanner [4]. These camera systems can be combined with some extra features such as the possibility to determine the exact 3D shape of the image with the help of stereo imaging techniques or by using structured light [4, 5].

It is very important that the images are taken in high enough resolution, because low resolution images obviously lose accuracy and information. Also different enhancing and modifying operations might be sometimes lossy methods and if the resolution in the images is not high enough, some essential information might be lost in the processing operations. Lossless RAW image format would make the image files large

in size and increase their processing time, because they contain more information. However RAW image is still preferable if the image processing needs to be done once only per image. They can be processed beforehand in one long processing run and save the pre-processed images, which contain lots of information, for further operations.

2.1.2. Image enhancement techniques

The image enhancement is not always a trivial process, because the information it aims to recover can be lost in other processing steps and different image distortions can result to a similar type of degradation in a document image [6]. There are lots of different methods available for image enhancement and the ones needed are dependent of what is wanted to achieve with the enhancement. Two almost always needed document image enhancement operations are document binarization and document skew and distortion corrections. Since historical document collections are most of the times of very low quality, an image enhancement stage is also essential [3].

Binarization

One of the most significant problems in Optical Character Recognition and Character Image Extraction is the conversion of non-ideal analog images into ideal binary images [7]. Binarization means transforming the image to a bi-color image, usually to black and white. It is usually the starting image enhancing step in most document image analysis systems and refers to the conversion of the grayscale image to a binary image [3]. Usually colored images are first transformed to gray scale images before thresholding.

(a) Original image (b) Gray scale image (c) Binarized image

Figure 2. Image binarization.

The most conventional approach is a global binarization with a global threshold, where one threshold value (single threshold) is selected for the entire image according to global information. Usually the global threshold is derived from the histogram of a gray scale image. Some global binarization methods use some additional information to decide the threshold, but they are closely related to adaptive binarization methods [8]. An old, but often cited and used method is Otsu's method, which tries to perform a global thresholding on the image by deciding the right threshold value from the gray levels on an image [9].

The binarization based on a global threshold is usually very easy to implement, but it is very vulnerable to different lighting conditions. If the lighting on the document is not uniform global binarization methods might not be able to tell the difference between text and the background. For example on a bending page the lighting causes gradual changes and at some point the background and text both might be above or below the threshold value causing both text and background to be converted either black or white. Also low resolution document image scans might add too much noise on to the image so that a global method might not be able to determine the global threshold correctly and thus the binarization may fail. Nowadays global binarization is not used much, unless it is accompanied with some other image enhancing techniques. Adaptive methods are more sophisticated and give better results with complex images [10].

Despite being somewhat old, still one of the most used and cited binarization methods today is Niblack's method, which is an adaptive binarization method. It varies the threshold over an image basing the threshold on a local mean and standard deviation calculated from a small neighborhood of each pixel [11]. Niblack's method is not very good with images where the background contains light texture, because the grey values of those unwanted details easily exceed threshold values [10].

Alongside with Otsu's and Niblack's methods one really often cited method is Sauvola's adaptive document image binarization. This method calculates individual threshold for each pixel from its context either directly or interpolating it from surrounding pixels. While being somewhat complex method, Sauvola's method performs very well when compared to the other most often used or cited methods [10].

The term binarization is usually broadened so that the term in a developed method actually contains lots of different enhancing and processing, like in the Orationes project. Adaptive Degraded binarization by Gatos, Pratikakis and Perantonis [12] includes several pre- and post-processing steps in their binarization process. First they filter the image and then use Sauvola's adaptive thresholding followed by interpolation and pixel contrast examination. It is not anymore just thresholding the image according to some gray scale value, but a whole process to achieve a black and white image with the interesting areas visible.

Skew and distortion correction

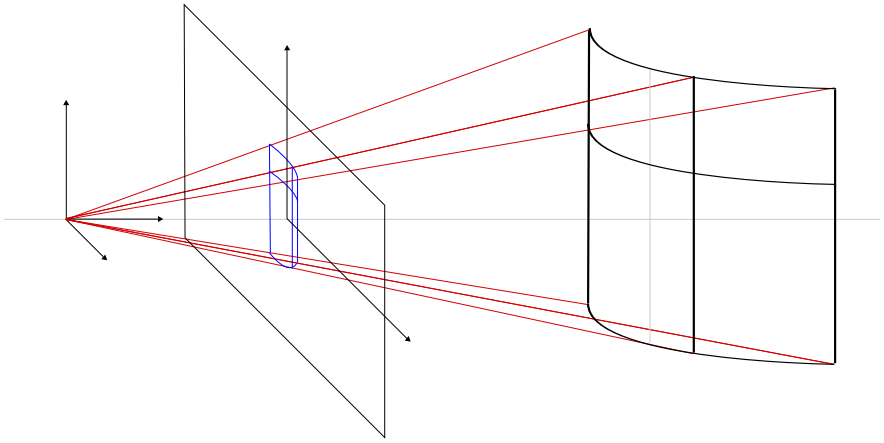
The use of different scanners often leads to skew in the document image. Skew means the document is not perfectly aligned with the scanner and therefore the text lines of the document image are not perfectly horizontal. Skew is usually corrected simply by first measuring the angle the text lines make with the horizontal direction and then rotating the image with the same angle to the opposite direction. Skew correction is necessary for success in any OCR system [2].

While skew is somewhat simple problem to solve, different distortions might be a tougher problem. Most often when scanning binded books, the binding causes heavy distortion on the pages as the pages tend to curve towards the binding and distort the images and text lines on the page.

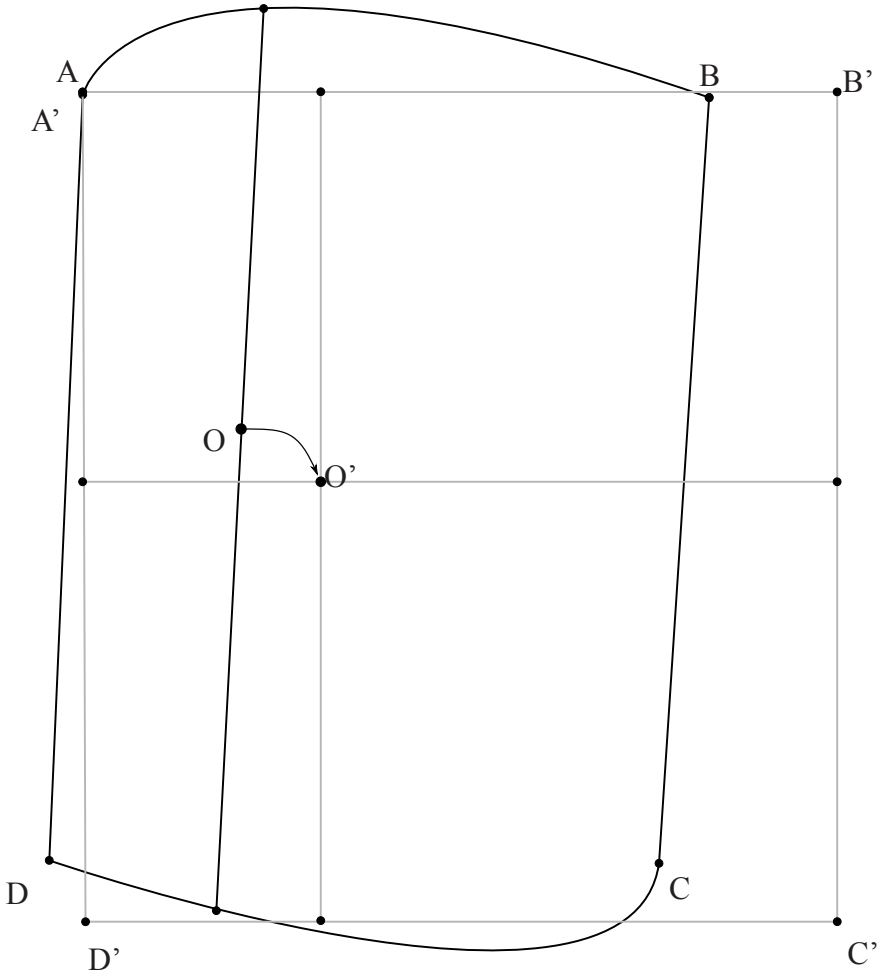
In ideal case it has been possible to push the document tight on to the scanner glass so that there wont be any distortion in the image. Older documents and especially

books can not be pushed heavily towards the scanner glass, because it would cause a high risk of breaking the book itself.

Several methods have been developed to correct the distortion.



(a) Generic rectifying geometry



(b) Deskewing on 2D spatial space

Figure 3. Distortion correction methods explained.

Some rectifying methods rely on creating or estimating the 3D model or shape of the real page and then flattening the 3D model [5, 13, 14]. The methods, which use

external information to create the 3D model, require that the pre-requisites for 3D modeling during the shooting are met. Instead the methods that computes the 3D shape from the images require that the photography geometry is well known, meaning that it is necessary to know the distance between the camera and the document, which lens was used, focal point of the lens used and all the other available information of the whole photography scene. However, despite being complex methods, these methods can be used to correct almost all types of distortion, because they reveal the real 3D shape of the document.

Always it is not necessary to create the 3D model of the page, but instead it might be sufficient enough to know some real 3-dimensional properties of the page and then use this information to calculate backwards the distortion to form a flat page [15, 16, 17, 18, 19, 20]. These methods often assume that the distortion is following some already known shape which can be then calculated backwards from the image, if the distortion geometry is known.

Some methods don't take the 3D shape of the page into account at all, but just corrects the distortion on spatial space with spatial transformations. This results in square pages, but it doesn't correct the distortion in the 3D depth, e.g. the foreshortening problem [21].

2.1.3. Text and graphics separation

Text and graphics separation namely aims to separate the text and graphical sections from each other. This is usually needed to increase the performance of the analysis, because after the separation the analysis can be focused on right sections of the page and the algorithm will not result in wrong positives on areas where text doesn't exist.

Text and graphics separation is especially important when handling document images that has both images and text. To increase the accuracy of recognizing text, the possible images should be excluded from the search, unless the images are analyzed as well. There are several methods of how to do the segmentation. One way is using different kinds of morphological operations [22]. On the other hand more sophisticated method is using globally matched wavelet filters, which are able to separate text and graphics areas from each other [23]. More detailed evaluation of the performance of different page segmentation methods can be found from the reports from ICDAR page segmentation competitions from 2001, 2003, 2005 and 2007 [24, 25, 26].

2.1.4. Line segmentation

Line segmentation is essential in all OCR and document image analysis systems. Without finding out the text lines it would be more difficult to find out where and what characters are in the document image that is been analyzed. Text line segmentation is usually done after binarization, because from binarized image the text lines are easier to extract by using Radon transform or Hough transform, which are common line segmentation methods [27].

Radon transform in general calculates the intensity sums over every projection over 180 degrees of an image. Radon transform that is taken alongside the vertical axis

of a page, outputs a one dimensional vector consisting of the intensity sum of each horizontal pixel line. From the vertical intensity projection it is possible to look for local maximums and minimas. If the image is correctly binarized the minimas or either maximas, depending on whether the text is black and background white or vice versa, of the transform represents the locations of the text lines [28]. Some variations of Radon transform methods tries to locate the top and bottom line from each text line and use this information to separate the text lines from each other and from the background [29].

On complex handwritten document images, where the text lines might not be perfectly aligned, Radon transform or Hough transform might not be good enough and therefore more sophisticated method for finding the lines is needed. One great method to find lines is fuzzy run length method by Shi and Govindaraju [30]. Though it is somewhat complex to implement, it offers great results in recognizing handwritten text lines which are not always uniformly formatted as the text lines in document images of machine printed text.

Many methods have been developed to detect the text lines from the handwritten document images. Many of these methods are reviewed in [31].

2.1.5. Zone and baseline detection

Zone detection is usually related closely to text line segmentation. Zone detection aims to separate different zones from the text line. Typically there are three zones on a text line: upper zone, middle zone and lower zone. Middle zone is the zone where the most of the character information resides. Upper and lower zones are zones of the text line where some of the letters are continued [2]. The boundary between middle and lower zone is usually called the baseline. Baseline is the imaginary line on which the characters are written. Zone and baseline detection is not always needed, but might be helpful with some algorithms [32].

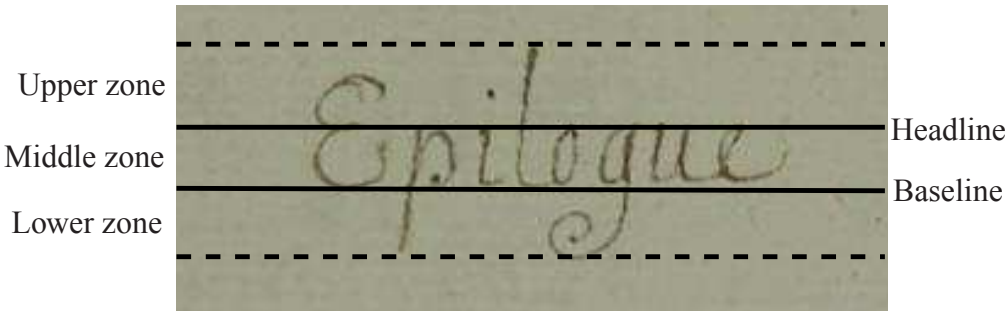


Figure 4. Zones and guidelines.

2.1.6. Word and character segmentation

Word and character segmentation is an essential step in OCR systems. Without segmenting the characters or the whole words, lots of calculation power should be needed to window through the whole image to find corresponding points for interesting areas.

Segmentation process aims to decompose an image into subimages of individual symbols or set of symbols in case of words. In the whole OCR system, segmentation step is one of the decision steps and makes a major contribution to the error rate of a system [33].

While the major question in OCR is 'how to define a character?', especially when there are numerous amount of different font types in machine printed text and infinite amount of different styles in handwriting and still humans are able to distinguish and recognize different characters and words from each others, to be able to analyze characters and words, it is essential that they are clearly separated from the uninteresting information in document image.

There are lots of methods developed for character and word segmentation. Because of the very different nature of handwritten and machine printed text, most methods are focusing only on segmenting either type of text. For example, Arabic Character Segmentation Algorithm shows great results on segmenting handwritten arabic characters. ACSA method is based on morphological features extraction. The characteristics of the arabic characters are pre-classified into a look-up list, which is then used to classify the characters and words according to the charasteristics found from them [34].

Another method developed by Louloudis et al. is based on the distances between adjacent overlapped components in a text line. This method is quite effective and also presents a method to locate thet text lines from a handwritten documents by using a Hough transform [35].

Machine printed text is usually lot easier to segment than handwritten text, because the handwritten characters, spaces and punctuation marks are uniform and there is dispersion in their characteristics hardly at all. Earliest methods can be traced back to 1960's. Usually in these methods the clear shapes and characteristics were exploited in automatic bank check reading. Also there were quite strict guidelines for the fonts used in the checks and the handwritten parts of the checks were either ignored or then the writer was guided to use certain types of letter to increase scanners' recognition rate [36].

3. DEVELOPED METHODS

The Orationes manuscript is processed with the help of existing XML transcripts. The idea is to use the transcripts to present some extra details about the information which is searched, like focus the search on some certain lines which are found from the image with regular image manipulation methods. The use of the XML files enables faster and more focused processing as it removes the need to analyze the whole page and every text line.

3.1. Distortion correction on pages

In this work the pages were not rectified automatically, but instead semi-automatically with some needed user input, to get to develop the actual algorithm in time. A procedure was developed to rectify the pages with GIMP image editor well enough for the purpose of this algorithm.

The method utilizes a GIMP plug-in ‘Curve Bend between paths‘ by GIMP registry user ‘Rob A‘ [1].

Proposed semi-automatic method goes through several processing steps which are explained more thoroughly here. In the manual page rectification there are several quite straightforward steps to go through when rectifying the pages.

1. Download and install GIMP and ’Curve Bend correction’ plugin.
2. Open the image that is needed to be rectified.
3. Select the path tool, which can be seen in figure 5
4. Follow some clearly visible path showing the distortion in the image. Mark it with 3-17 path dots from left to right. Create own paths for top and bottom paths as in figure 6. These paths can be seen in the path sheet, which is seen in figure 7.
5. Once the paths are set, select the curve bend correction tool as in figure 8.
6. Select the top curve and bottom curve where they are asked in the tool. There might be some random numbers after the curve names, but they are not important as long as the paths are named correctly. It needs to be ensured that the right curves are selected on each field. The tool user interface can be seen in figure 9.
   (a) Distortion compensation is usually left to 0. It can be changed if needed. If the rectification result is not good enough, it is usually better to fix the paths rather than the distortion compensation parameter.
   (b) Quality should be set to slow to ensure best possible distortion correction.
   (c) After bending option is usually good to set to ’Fit canvas to layer’.

[1] URL: http://registry.gimp.org/node/19214

Figure 5. GIMP path tool.

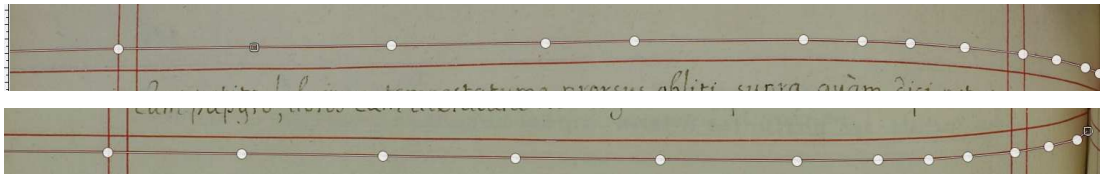


Figure 6. Top and bottom paths

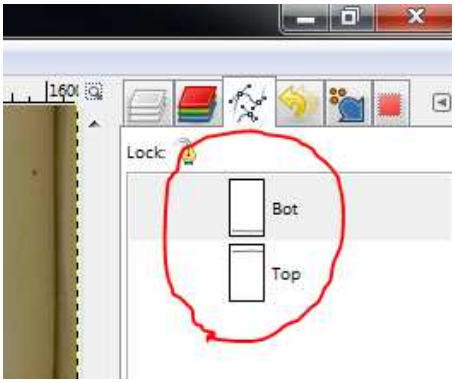


Figure 7. Both paths.

7. The distored page should be rectified now. However if there's still some distortion, undo the distortion and repeat the above steps either with new paths or different parameters in the correction tool. Also some images might need slight skewing or rotation after the bend correction.

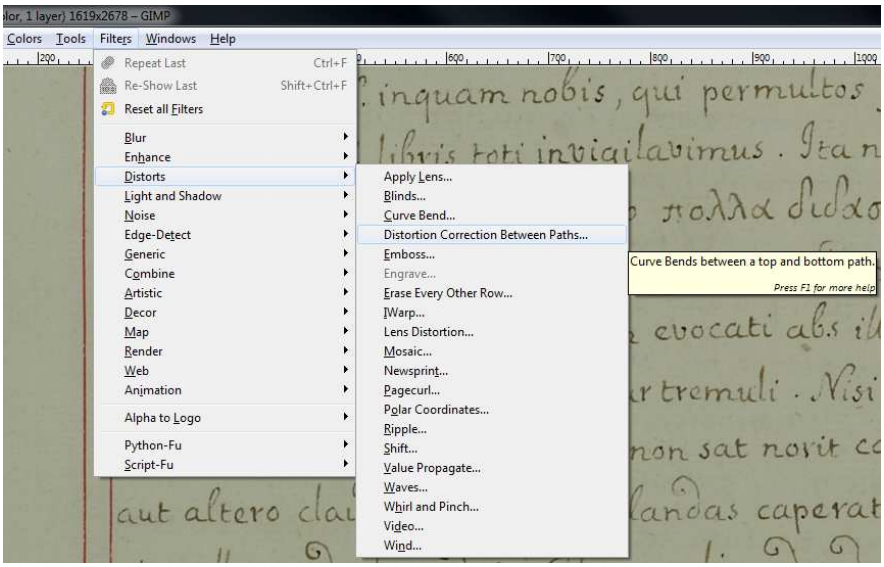


Figure 8. Bend correction between paths plugin.

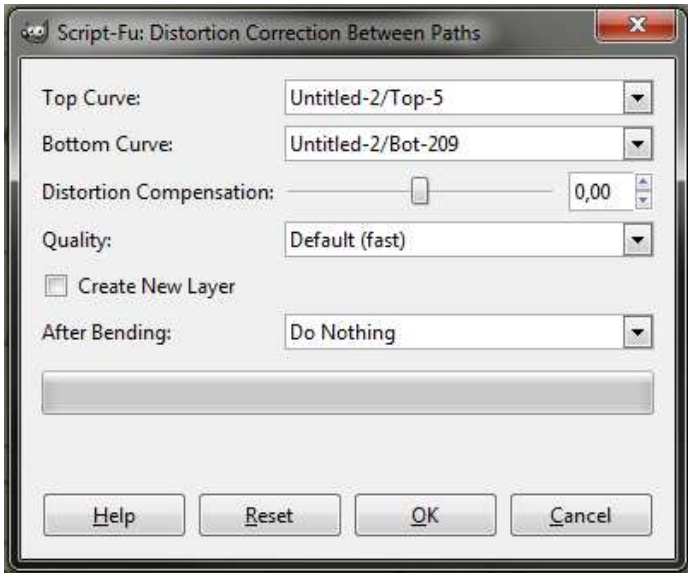


Figure 9. Path correction tool.

3.2. Pre-processing

The Orationes application and its Python engine works together with a PHP web user interface. Because of the existing XML transcripts of the Orationes text, the engine is given information from the XML, parsed in the PHP side of the whole application. The proposed algorithm needs to do some pre-processing of the XML file, because it relies on the XML transcript of the Orationes manuscript. Most of the pre-processing is done on the web UI side where the right parts of the XML is cleaned from the XML markup language into a plain understandable and readable text.

PHP side parses out all the XML markup language notations and modifies the text so that only the real text lines in correct order are given to the engine. This allows

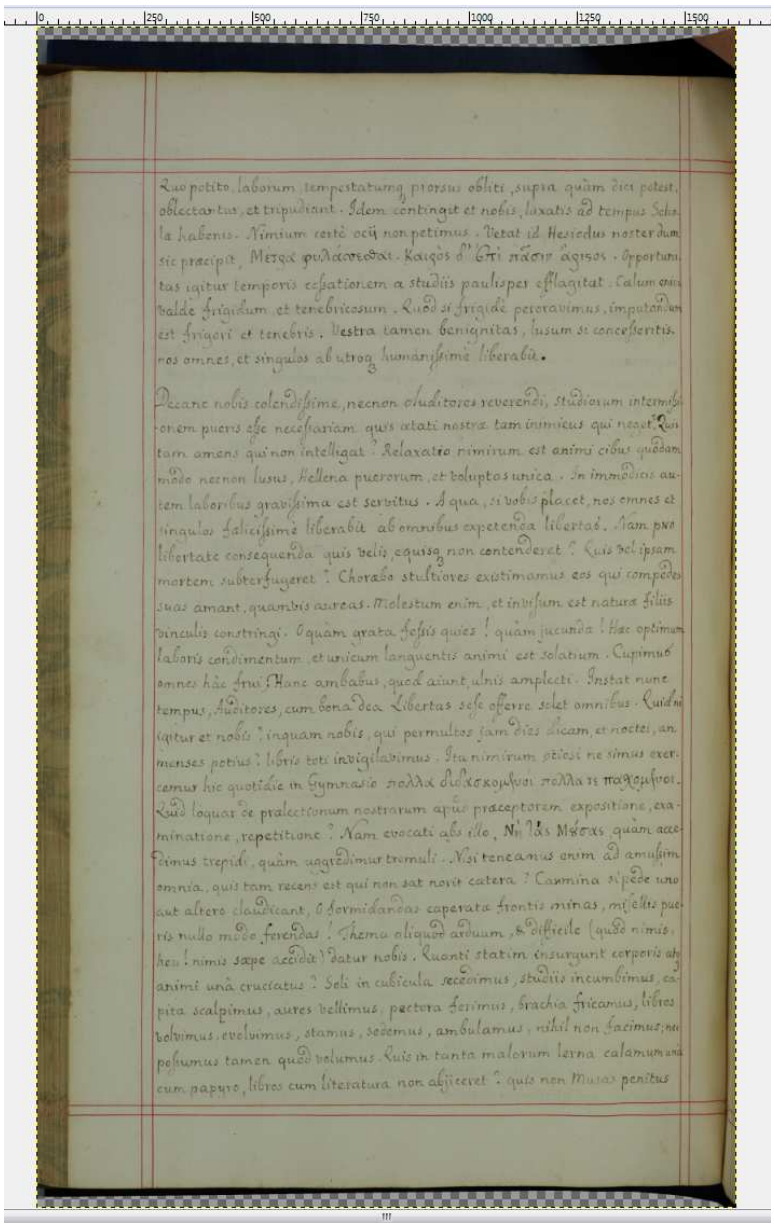


Figure 10. A rectified page in GIMP.

the engine to evaluate the validity of the information found from the images. These pre-processing steps can be done every time the algorithm is run or because of their repetitive and non-changing nature, they can be done once and be saved for later use. Only time the steps should be done again is, if the XML file is changed or the parser is changed.

The pre-parsed XML files are useful, if the XML files are not changed. The XML file as a string is useful only, if the XML file changes often, though due to the nature of the document and manuscript, changes are not very probable. The idea is that the XML pre-processing is done only once and the pre-processed XML files are then used over and over again so that the processing time is kept as fast as possible.

The PHP XML parser is not part of this thesis, so it is not explained any further.

3.2.1. Parsing the PHP execution

The program has the option of which kind of execution is to be used. The first option reads the parsed XML file directly from a file where the parsed information is stored or alternatively the parsed XML file can be passed to the python algorithm engine as a single long string. The Orationes Python engine is usually started from the web user interface, where it is given inside the PHP code the instructions of how the engine should be run. This part is kept simple and the PHP only tells the engine whether to use an already parsed XML file or a long string containing a freshly parsed XML information. Another alternative is to run the engine directly from the command line, but for the ease of use this is not recommended. However running from the command line uses exactly the same format as the underlying PHP code on the web page. Once the execution type is parsed the parser also checks which word or character is to be searched from the text and passes it to the engine.

The invocation command consists of telling the PHP code to run the Python engine with correct parameters, which are the name of the image that is to be analyzed, switch parameter telling whether the engine should read the raw text - achieved from an XML file - directly from a pre-parsed file or from a string that contains the information that is parsed on the fly, name of the parsed XML text file or either the whole parsed string, and finally the last parameter the word or characters that one wants to search from a certain page.

```
python osearch.py [i] [s] [fname] [w]

i         imagename
s         switch, -f for file, -s for string
fname     name of the file if -f or the string if -s
w         the word or characters which is to be searched

Example
python osearch.py "Lit_Ms_E_41 070r.jpg" -f "070" "y"
```

Figure 11. Example usage of python engine.

3.2.2. Handling the parsed XML file or given string

The pre-processing steps are quite trivial on the engine end, because most of the pre-processing happens on the PHP side. The engine just parses out the command given to it by PHP site and after parsing it starts its processing.

In this step it is needed to calculate the amount of real text lines on the page according to XML file, character count on each text line and also the character positions are calculated from the XML. If the searched item is a word, then instead of character position, the engine will search the positions of the first characters in the word.

After the needed text information from the XML file is passed to the engine, it has to be processed so that the needed information is easily accessible. There are two

functions from which either one is used for the task depending on whether the interesting information has to be gathered from a textfile or from a long string containing the parsed XML information.

Both functions perform the information gathering task similarly, but the difference between them is that the other one has to open a text file while the other one first has to format the given string so that it can be handled and the information gathered into correct elements.

The engine has to gather the information about the number of characters on each line, the position of the wanted characters or the position of the first characters of the wanted word, number of hits on each line and the length of the word that is searched.

Character count is simply a list containing the length of each line in the XML and also on the actual document image. Characters on every line are summed up from the parsed XML file. The index of each of these numbers defines the line which these characters belongs to.

Character position list consists of smaller lists telling the positions of characters found on each line. The index of the sublists defines the number of the line and the content of the sublists defines the position of the interesting characters or the position of the first characters of the interesting word on that text line. Empty sublist means there are no hits on that particular text line.

Even though the information about the hits on each line could be calculated from the character position information, the data telling how many hits are on each line are collected separately in a separate list to make the further processing and calculation faster. The *charlist* list consists of sublist where the index of each line is multiplicated into the list as many times as there are hits on that line.

The last item is also a collection of sublists where each sublist has as many elements as there are hits on that line and each of these elements states the length of the word that is searched from the page. In case of a single character the length is obviously 1.

The last two list of lists may seem to be repetitive information that could be calculated from other variables constructed inside the function. However, all that information is calculated and cached inside the function to reduce the processing time and repetitive function calls in the further stages. The amount of memory needed to store all that information is very negligible.

$$\begin{array}{cccc} \text{CharCount} & \text{CharPos} & \text{CharLines} & \text{WordLens} \\ \begin{bmatrix} 63 \\ 60 \\ 64 \\ 65 \\ \vdots \end{bmatrix} & \begin{bmatrix} [52] \\ [10, 47, 62] \\ [19, 62] \\ [51] \\ \vdots \end{bmatrix} & \begin{bmatrix} [3] \\ [4, 4, 4] \\ [6, 6] \\ [7] \\ \vdots \end{bmatrix} & \begin{bmatrix} [3] \\ [3, 3, 3] \\ [3, 3] \\ [3] \\ \vdots \end{bmatrix} \end{array}$$

Figure 12. Illustration of data calculated from parsed XML file.

3.3. Image enhancement

After the text has been processed from the text files, all the images in the project has to be enhanced to get better search results. In Orationes project all the document images

are color images. The best recognition results are usually achieved from gray images or even from binarized images with only black and white color in them. That's why the Orationes document images are first converted to gray images and enhanced to separate the characters from everything else shown in a page.

Enhancing process starts with transforming the images from RGB images to gray scale images. The transformation is basic gray scale transformation process where the luminance information is preserved. Because this is done by using the library functions, which doesn't change the precision to float, the precision has to be changed manually, because the further image enhancement process are done in floating point precision.

Gray level transformation is done by just taking the intensity levels of the RGB document image. The gray level images are filtered by using homomorphic filtering to remove slight intensity changes caused by small curvatures and bends on the page. After the image has been changed to gray scale, it has to be filtered and enhanced further by stretching the contrast.

After the filtering the contrast in the image is even more stretched, meaning that the value of each pixel in the image is either pushed up or down by a multiplier. This is emphasizes even more the fast intensity changes at the boundary of background and characters.

After this it is possible to perform traditional binarization on the image using a single threshold. The product is well binarized image with low noise and very few non-interesting areas visible. Though binarization is quite simple process, there are only few cases where the traditional binarization is used. The effects of this enhancement process can be seen in figure 13.

3.3.1. Filtering

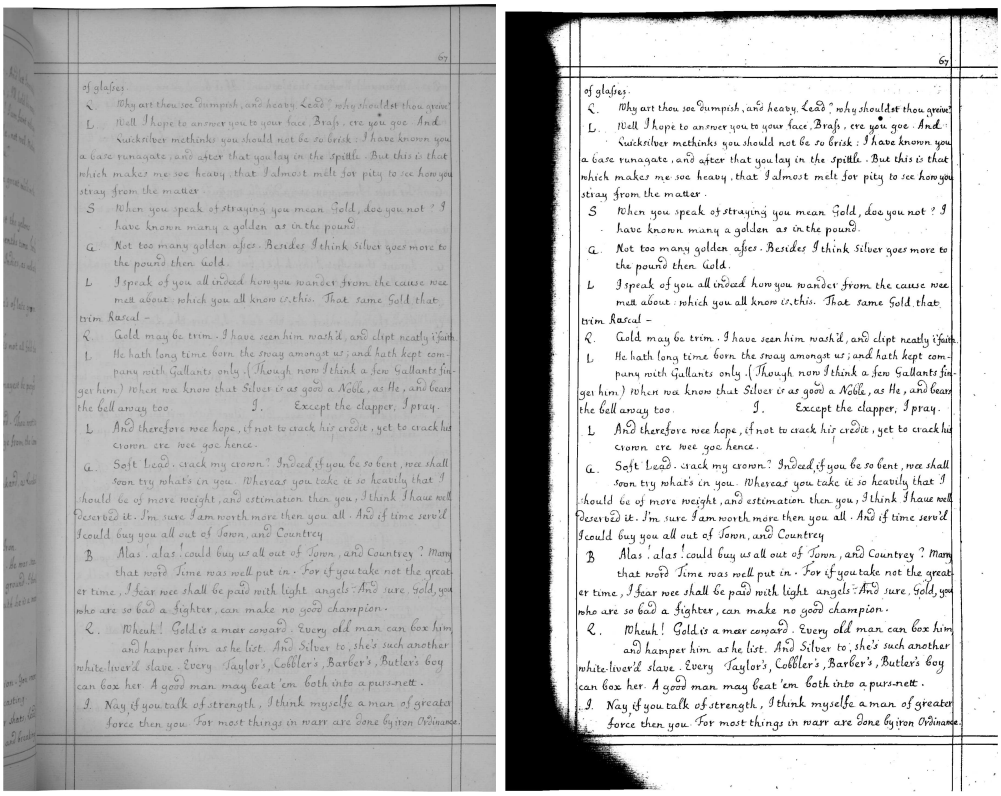
Filtering method used in the process is homomorphic filtering, which performs well, when filtering out low frequencies from slightly distorted and curved document images. It is useful especially because it simultaneously normalizes the brightness across the document image and increases its contrast.

Low intensity frequencies consist of the gradual intensity changes which happen on long spatial distance, i.e. lighting changes along slightly curved page, while instead high frequencies consists of sudden intensity changes on short distance for example between background and the characters.

Homomorphic filtering itself isn't a filter, but a filtering method. The actual filter used is a Butterworth high pass filter, which filters out the low frequencies, or in spatial space and terms slow intensity changes, from the image thus leaving only the high frequencies also known as the fast and sudden intensity changes in the spatial space.

Homomorphic filtering is used to bring up the textual content in the image and also to decrease noise and intensity changes caused by the curvature and bending on the pages.

In homomorphic filtering the whole image is Fourier transformed to get the frequency domain from the image. This frequency domain represents all the luminence changes and their speed in the image. The closer to the center of the Fourier transformed image the frequencies are, the slower intensity changes they represent in the



(a) Original (b) Enhanced

Figure 13. Graylevel image before and after enhancement process.

spatial image space. Because the intensity change in the border of background and text is very fast and sharp, it is possible to filter out the slow gradial intensity changes by filtering out majority of the slow frequencies. This leaves only the fast and sharp changes on the image.

According to the illumination-reflectance model of an image, the intensity of a pixel is the product of the illumination of the scene and the reflectance of the objects in the scene, $I(x, y) = L(x, y)R(x, y)$, where I is the image, L is the illumination and R is the reflectance component. Especially in document images taken of bent books, the illumination L is non-uniform. To enhance the image, the idea is to remove the illumination L and keep only the reflectance R.

Illumination changes very slowly over the images in Orationes project especially when compared to reflectance which changes very fast at the edges or letters and background. That's why the multiplicative components are first transformed to additive components by moving them to logarithmic domain as shown in equations 1 and 2.

$$ln(I(x, y)) = ln(L(x, y)R(x, y)) \tag{1}$$

$$ln(I(x, y)) = ln(L(x, y)) + ln(R(x, y)) \tag{2}$$

Because it is known that the non-uniform illumination component resides in the low-frequency parts of the Fourier transformed image, a high pass filter is used to filter the frequency domain image thus leaving only the high-frequency reflectance component.

After the high pass filtering the exponential function is applied to the image to invert the moving to the logarithmic domain.

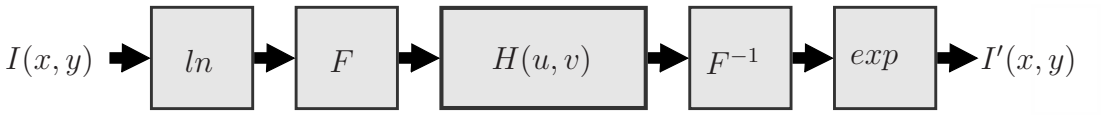


Figure 14. Image filtering pipeline.

First the image is moved to logarithmic domain. Because the pixel values are in floating point precision, 1 is added to the value of each pixel to get correct values when operating in logarithmic domain. This 1 is subtracted after applying the exponential function.

The image in logarithmic domain is then Fourier transformed to get the frequency domain of the image. This frequency domain image is filtered with a Butterworth high pass filter. Butterworth filter is used to get sharp and fast transition between the pass band and stop band and also there's not much rippel at all. Also it is fairly easy to implement with python vectors.

$$H = \sqrt{\frac{1}{1 + (\frac{D}{D_p})^{2N}}} \tag{3}$$

Constructing general Butterworth filter, shown in equation 3, by calculating the distance of each pixel one by one from the center of the image would take lots of time. Instead calculating the horizontal and vertical distances of each pixel can be done fast with python vectorization. Two image size distance grids are made. First one contains the values of the vertical distances of each pixel from the vertical middle line and the second grid contains respecticely the horizontal values. Then these two distance grids are put together and the distance of each pixel is obtained by calculating the distance with using the basic Pythagorean rule. This results to a image size distance grid which can be used when constructing the Butterworth filter.

Butterworth filter is the size of the image itself and the stop band is large compared to the image size, because only the very highest frequencies are the interesting frequencies which contains the information of the characters and the background boundaries.

When filtering in frequency domain, there are usually interference in the non-zero parts of the adjacent copies of a signal caused by the wraparound error, which happens because the discrete Fourier transform treats finite length signals, like the images, as infinite periodical signals, where the original image, the finite length signal, is considered to be one period of the infinite length signal. However, because the images in Orationes project are large, the wraparound error affects only the very borders of the image. Therefore the zero padding is not needed, because the interesting parts of the images resides in the middle of the image, far from the borders where the error might be seen. This fact makes it possible to skip zero padding without getting any significant error to the image and it speeds up the whole process.

The actual filtering happens when the frequency domain image is multiplied element by element with the filter. The result of the multiplication is the filtered image where the unwanted frequencies are zero or close to it. The filtered image is then transformed back to spatial domain by inverse Fourier transform. The two last steps are applying

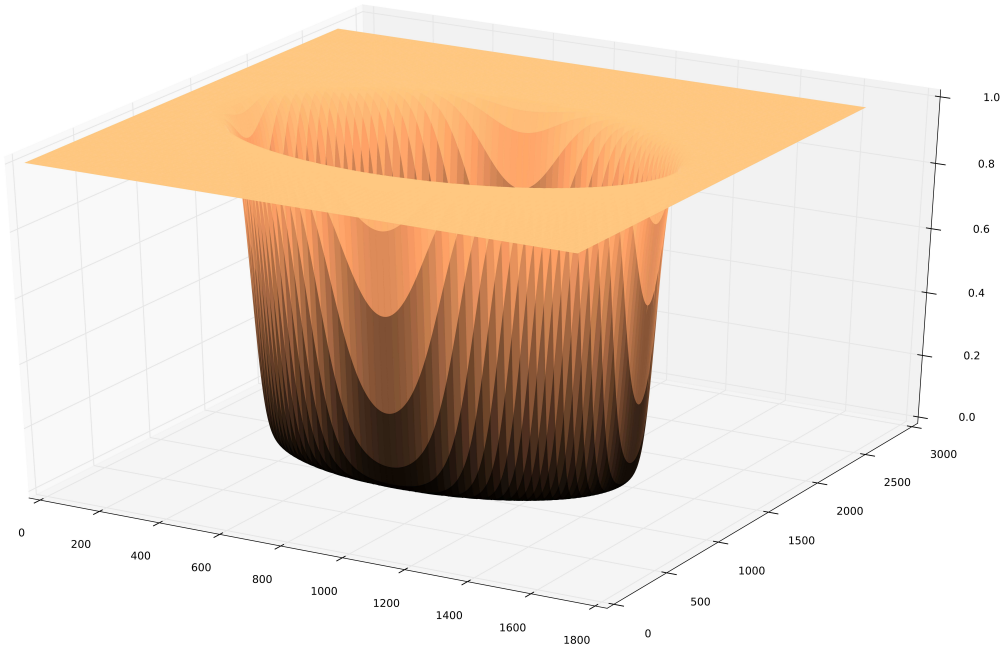


Figure 15. Butterworth High Pass filter.

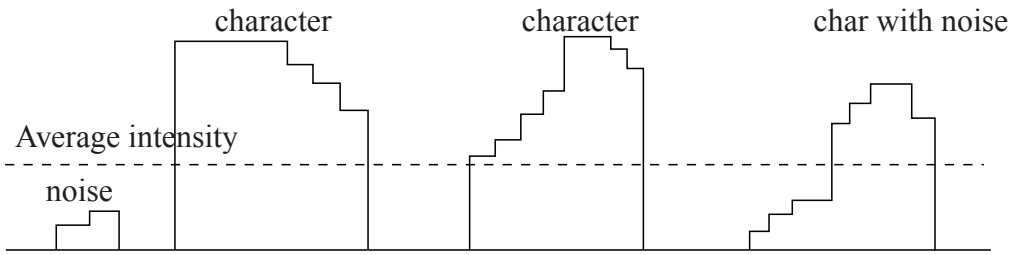
the exponential function to the new spatial image and then substracting 1 from every element, because 1 was added when moving the image to logarithmic domain. The final result is a image which has only the fast components, especially the text, visible.

3.3.2. Contrast stretching

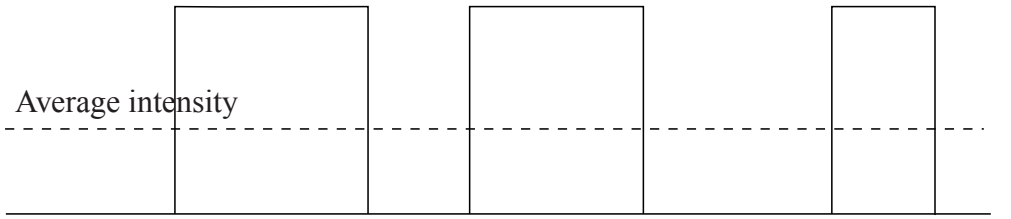
Even though the filtering increases the contrast over the whole image, the contrast is enhanced even further with a simple contrast stretching, where the values of each pixel is increased or decreased depending if it's already greater or smaller than the average value of the pixels. This ensures the best possible binarization result.

In contrast stretching, each pixel in the image is compared to the average gray-level of the image. If the gray-level of a pixel is under the average, it is pushed even further down by a certain coefficient. This effectively ensures that the area near the border of the background and characters becomes as clean as possible. This coarse contrast enhancement cuts off the possible remaining slow gradual intensity changes near the background and character border, thus sharpening the edges of the characters and making them more recognizable.

This function relies heavily on the homomorphic filtering and that it is able to separate the text pixels well enough from all the noise and non-uniform lighting. If the homomorphic filtering is not able to distinguish the characters well enough from the background and leaves too much noise and unwanted background pixels, they will become even more visible and create even more noise to the image in the contrast stretching. The aim for contrast stretching is to bring the shady boundaries of the characters up and sharpen the edge of the characters and background. This can be seen as a method to rise up the rounded edges of the characters.



(a) Pixel intensities before stretching



(b) Pixel intensities after stretching

Figure 16. Contrast Stretching explained.

Contrast stretching function performs contrast stretching for grayscale images. Pixel intensities are set to differ by a coefficient a times the average intensity from the original intensity values. The new intensity values are then sliced to stay between $[0, 255]$ to ensure that the difference between interesting pixels and unwanted pixels is easily distinguishable when the grayscale image needs to be binarized. Contrast stretching is shown in equation 4.

$$I_{stretched} = I_{old} + a * (I_{old} - I_{average}) I_{new} = \begin{cases} 0, & I_{stretched} < 0 \\ I_{stretched}, & 0 \leq I_{stretched} \leq 255 \\ 255, & I_{stretched} > 255 \end{cases} \quad (4)$$

The average intensity is usually calculated over the whole image, but there's also the option to calculate the average intensity from a certain part of the page. For this option, there is parameter h in the function and it 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.

3.4. Determining the bounding boxes over the textlines

In the case of Orationes manuscript where we can use the information gathered from the XML transcript, it is possible to use bounding boxes over each text line to increase the recognition rate. The search shouldn't be done over the whole page, because focusing the search on interesting parts of the page not only increases the recognition rate, but also speeds up the actual search.

The bounding boxes are determined so that every text line is first blurred and grown vertically to form a single blank thick line. Once all the words and characters on a line are grown together into a single patch, it is easy to determine its bounding box.

If the words or characters are too far away from each others, they might not grow together, but instead form two or more individual patches. To ensure that the search algorithm won't handle them as a separate lines, they have to be connected manually. The boxes are connected by comparing the height of the boxes. If the boxes are horizontally close enough to each other, they are considered to be on the same line and thus connected.

The growing of the patches is done in two stages and before each stage the image has to be cleaned from noise and debris, so the noise parts wont grow and create even bigger noise to the image. Cleaning is done by simply leaving out too small patches from the image. The average size of the smallest patches is approximated and this approximation is used to leave out too small patches. Also if a patch spans horizontally over too many pixels, it is left out as well, because it is very probably a noise patch from the margins of a page. These too high patches are very critical, because the height information is used to combine the boxes on a same line and if there are patches spanning multiple lines left on the image, it might cause the algorithm to combine all the boxes on two or more different lines.

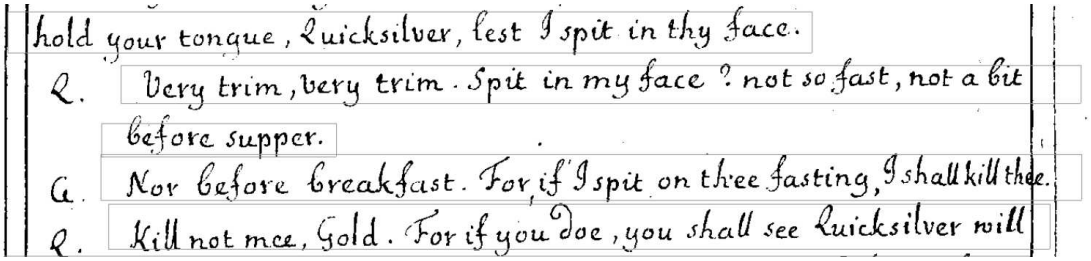


Figure 17. Bounding boxes bordering text lines.

3.4.1. Binarization and labeling

The *boundingBox* function tries to determine the bounding boxes for each text line from the processed gray scale image. The image passed to the function is a gray scale image and it is binarized before any bounding box calculations. It is known that after the contrast stretching the character pixels has approximately the value 0.95 or higher, so the binarization threshold is set to a static 0.95. A histogram based method could be used and in the function there is a mechanism to use dynamic binarization, but it is neglected, because the static binarization gives good enough results and consumes less time.

After the binarization the bi-color image is transformed to its complement image. This way the background and other unwanted parts become black and the interesting areas containing the text become white and the white parts can be labeled.

3.4.2. Cleaning and region growing

The function has to clean debris from the image every time it is labeled and there are two of these labeling and cleaning iterations. After each labeling it is very probable that there are some unwanted patches in the image. After the first labeling it is known that there is one enormous patch in the bi-color image. The patch is the area containing the margin lines and some noise connected to it. This area is not needed at all so it is removed, because all the interesting areas are inside the margins.

Once the largest single patch is removed, it's time to remove all the too small patches. It is known that the areas of character patches are usually larger than 50 pixels, so all the patches smaller or equal to 50 pixels will be removed. This number is also a static number, because calculating the character patches and their sizes dynamically would take too much time and also it is really hard to distinguish the smallest character patches from largest noise patches.

After the removal of small patches, the patches which are too high will be removed. This ensures that the possible unconnected margin patches are removed and that they wil not cause any connecting bounding boxes over two text lines, because every bounding box should contain only a single text line. When the image is cleared from debris and uninteresting parts, it is then first eroded with a small vertical line structure element.

$$SEe_{5,5} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad SEd_{70,70} = \begin{bmatrix} 0 & 0 & \dots & 0 & 0 \\ & \vdots & & \vdots & \\ 1 & 1 & \dots & 1 & 1 \\ & \vdots & & \vdots & \\ 0 & 0 & \dots & 0 & 0 \end{bmatrix}$$

Figure 18. Structure elements for erosion and dilation.

Erosion squeezes all the text lines slightly in vertical direction and it is needed, so that the text lines wouldn't grow together in the dilation phase. After erosion all the patches are dilated with a long horizontal line structure element. This grows all the characters in a text line together and creates a single patch covering the whole text line. After the dilation, last cleaning phase will be done and again all the too small pathces will be removed, because there still might be some noise and debris patches left, which have grown after the dilation. After the last cleaning there should be left only the patches covering the text lines. These single patches over the text lines are then used to determine the possible bounding box that is covering each individual text line.

3.4.3. Calculating the bounding boxes

The bounding boxes are calculated from the patches covering the text lines by taking the farthest pixels from each direction from every patch. These pixels are then used to determine the upper left corner and bottom right corner, thus the edges of a patch.

Figure 19. Bounding box finding steps.

On some cases two or more patches in a same text line are not grown together in the dilation phase so the bounding boxes over them have to be combined.

Two vertically close bounding boxes are assumed to be on a same text line if the difference between their highest pixels is small enough. Combining is done by getting the leftmost and rightmost x-coordinates of these patches and set to be the far end x-coordinates of the new bounding box. Similarly the highest and lowest y-coordinates are taken and set to be the farthest y-coordinates. These coordinates then determine the far end pixels for the new bounding box.

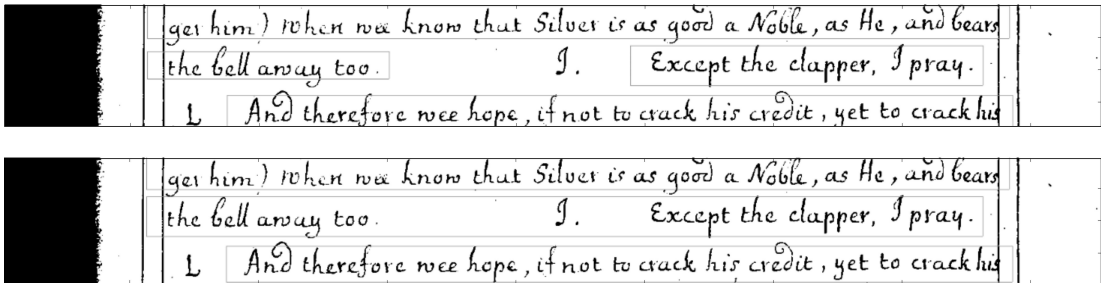


Figure 20. Combining the bounding boxes on same text line.

3.5. Line search

Line search is based on enhanced Radon transform alongside some morphological operations. The main idea is to create a radon transform, or simply put, an intensity projection, from the long edge of the inverted binarized page and pick the local maximas from it.

The intensity projection creates a wave signal showing the intensity density over each horizontal pixel line. For convenience, the image is inverted to show the text in white and background in black. Normally the text lines resides approximately at the minimas of the signal, but in the inverse signal, the text lines reside at the local maximas, which are easy to filter and detect with a peak detection function.

The signal includes lot of noise along the interesting information. The noise comes from random background noise, non-uniform character forms and variations in the intensity on the page in general. This noise and non-uniform character forms cause lots of false local maximas to the signal and also sometimes causes double tops at the correct maximas. These false and double maximas are then filtered out to ensure best possible peak detection results when deciding the locations of the text lines.

The number of text lines on each page is calculated from the XML file, but it doesn't tell their respective locations on the document image. The bounding boxes give some estimate of the locations, but sometimes the bounding boxes might not be accurate. Therefore the text lines are also searched by using a Radon transform method.

Because of the pre-conditions in Orationes project, the type and orientation of the pages and document images is known. In the engine the Radon transform is then modified to be so called 'Poor man Radon transform', which is a simplified version of the Radon transform made for this thesis only. In this PMR method the Radon transform is done only on one direction which is from left to right, to the direction of the text lines, whereas in general Radon transform the transformation is done on each direction in 360 degrees. The other angles are left out, because it is known that all the text lines on the images are always from left to right, because the images are not rotated in the Orationes project.

Before applying the PMR function to the image, the up and bottom part of the page are cleaned from unnecessary data. All the pixels until the lower top margin and from the upper bottom margin are set to be zero. Those pixels are not needed when detecting the text lines. Instead of removing them, they are set to be the background, so the image preserves its original dimensions and the text line locations are easier to locate on the original image.

Usually the text in books is printed with black or other colored ink on a white or light background. This means that the intensities on the lighter areas are higher than on text areas. Respectively on complement image, the intensity sums are higher on the horizontal lines where the text pixels are. In the PMR function the passed processed gray scale image is first binarized and then transformed to its complement image, so the intensity maximums will be on the actual text lines. Then the intensities of all the horizontal pixel lines are summed and then compared to each other. This creates an intensity sum vector with local maximums and minimums.

The intensity sum vector contains lot of noise and irregularities caused by different shape of the characters. Still it is known that the text lines cause some periodical elements on the vector and this periodic nature can be exploited in frequency domain,

meaning the vector is filtered after it has been Fourier transformed. Because the average distance between two text lines in the Orationes project and in the images with the given resolution is 55 pixels, all the frequencies smaller than 55 are cut off. This will leave only the intensity frequencies which map the text lines in the vector. If the average text line distance is not known or if the resolution changes, filtering should be done dynamically. The original intensity vector is seen as blue plot and the filtered intensity sum vector as green plot in the figure 21. It can be seen that the peaks in the filtered intensity sum vector aligns well with the text lines.

The peak detection function is used to find the peaks from the intensity sum vector. In the peak detection, it is assumed that a spike is considered a peak if it is 25 units away from a previous detected peak and also if its value difference is at least 3000 to its previous value. Once again static values are used, because the nature of the operating environment is known and it was necessary to save time in the development. Possible maximum points found from the margin area are removed. Only the area containing text will be considered allowed area for the maximum points.

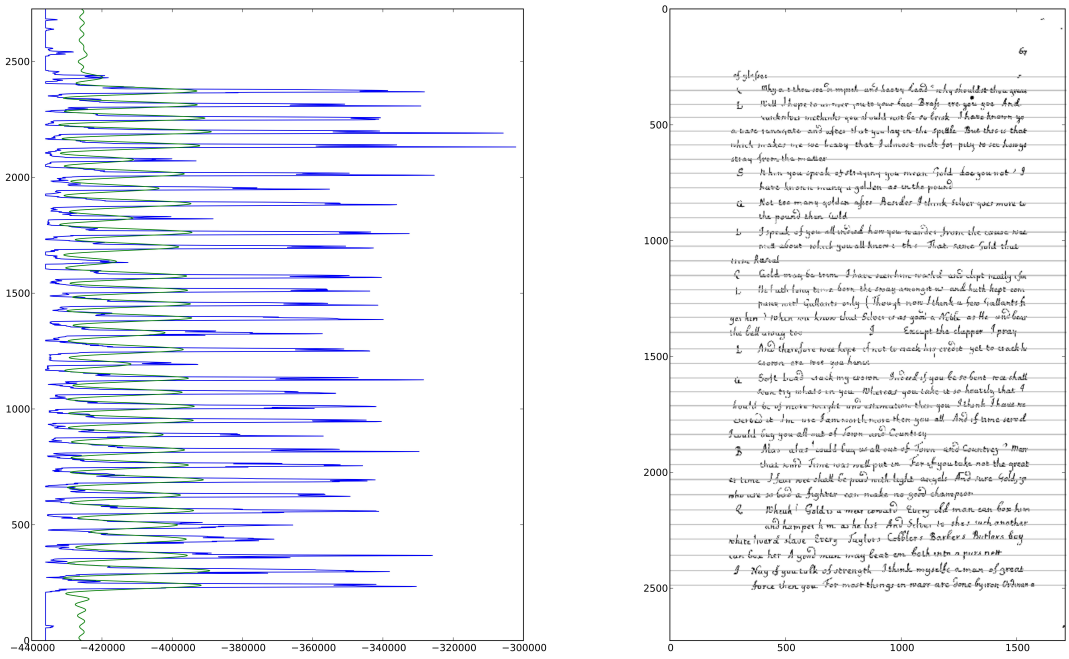


Figure 21. Radon transform correlation with text lines.

The *poormanradon* function returns a vector *imlines* which contains the y-coordinates of the found text lines. The indexes of this vector is assumed to align with the indexes of the vector containing the information of the text line information from the XML file. However, because the line finding phase may not always give right results, the results have to be checked and corrected if needed. This is done in the line processing phase.

3.6. Line processing

Once the text lines are found and located from the image, the results are processed and uninteresting lines are left out, so that only the text lines where the searched characters and words are, are left for further investigation. It is needed to check, if they match with the information in the XML transcript. The amount of real lines on a page is received from the XML files and it is then compared to the amount of lines found from the image through image manipulation. This is done in *processlines* and *padlines* functions.

Always the image manipulation methods might not work properly to find exactly the right or correct amount of lines from an image. This is because human perception is more complex and works in a different way than what's possible to implement on artificial platform. Therefore the line finding process is more like a sophisticated guess of the lines. Sometimes it goes wrong more or less and it is needed to correct the guess in a best possible way.

There are three possible results in the line search: There are not enough lines found from an image, there are exactly the right amount of lines found from an image and then there are too many lines found from an image. To simplify the line padding process it is assumed that when there are same amount of lines found from the image, they are the same correct lines which are recognized as text lines by human perception. While there are not enough lines found, it is assumed that the shortest lines are not recognized. However when there are too many lines found from the image, it is sometimes really hard to tell which lines are false positives.

Once the lines, their positions and amount is processed and decided, it is time to select only the lines of interest, because it is not necessary to process the whole page when finding the characters, assuming the line search and padding gives correct results.

The *processlines* 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. It takes in the character count from the XML parsing function and y-coordinates of found text lines from the *poormamnradon* function. It is really important to find the correlation between the text lines from XML and from the image so that the text alignment is correct.

The amount of lines found from the image and gotten from the XML are compared to each other. Clearly there are three possible cases: there are more lines found from the image than got from XML, there are equal amount of lines found from image than what is got from the XML and there are less lines found from the image than what is got from the XML. Each of these cases has to be dealt differently and that is done in the *padlines* function.

In first possible case there are less lines found from the image than what is got from the XML. At the moment, if this happens, there are not much to do, because there is no way yet to determine where the missing lines reside in the image. Therefore it is assumed that the missing lines are the shortest lines, because they are most probably missed in the peak finding algorithm during the line search phase and thus those short lines are completely neglected. This is done by going through a vector containing the information about text lines in XML file and their lengths. The same amount of the shortest lines are set to be neglected as there are non-found lines.

If there are equal amount of text lines found from the image than what is calculated from the XML file, it is assumed that the found lines are exactly the correct lines. This assumption is made, because the numerous tests of the line finding function showed that with the Orationes document images when there were same amount of lines found as is the number of lines in the XML, the found lines were the correct lines. If the engine is used on some other images, this case should be improved to include a method to check whether the found lines comply with the lines in the XML.

As the time ran out during the project, the third case, where there are too many lines found from image, is not implemented. However, the idea was to compare the amount of characters on each line to the length of the found lines on image, try to decide which line pairs would match and then leave out the remaining lines. Also if there are two lines too close to each other of the found lines, it is highly probable that one of the lines is a false result.

Because the vector *imlines* containing the positions of the found lines in the image is aligned with the vector containing the information about the text lines in XML, the indexes has to be corrected to point on right lines. A vector *rlines* is created so that, if a text line on a certain index is considered too short, the value that is the line y-coordinate is replaced with a NAN and the value in its index is set to be the value of the next text line.

$$\underset{\text{llines}}{\begin{bmatrix}1\\1\\1\\1\\0\\1\\1\\0\\1\end{bmatrix}} \quad \underset{\text{imlines}}{\begin{bmatrix}100\\200\\300\\400\\600\\700\\900\end{bmatrix}} \quad \underset{\text{rightlines}}{\begin{bmatrix}100\\200\\300\\400\\NAN\\600\\700\\NAN\\900\end{bmatrix}}$$

Figure 22. Correction of line locations and indexes.

Because of this, possible hits can not be found on these ignored lines causing some error in the final search results. The whole system still being more or less semi-automatic, it is left for the users responsibility to check those short lines, because they usually consist of only couple characters or a single word and doesn't consume too much time to manually go through them and check possible hits.

Finally in the end of the function the lines containing the hits found from the XML are filtered into a vector which is then returned to be the vector of the interesting lines. These are the lines that are used to search the hits from in the last phase of the execution.

3.7. Getting the hitboxes

The second to last step in the process is determining the possible locations of the characters or words which are searched. This is done again with the help of the XML

transcripts, since it is possible to calculate the position of characters and words in the XML files and then predict their respective positions on the image by using this location information together with the line location information calculated from the document image. This step is done on the interesting lines only and in the *findCorr* function.

Each bounding box on an interesting line is divided into small boxes and the size of a box depends on the size of the bounding box and the amount of characters on each line. It is assumed that the characters are equally divided along the line. While it is well known already at this stage that this assumption is causing inaccuracy, it is the best method what was achieved during the time allocated for developing this step.

Once the bounding boxes are divided into small sub-boxes, the sub-box which has same order number as the wanted character on that line, is chosen to be the approximate location of the character on that line. In case of words the first character of the wanted word is chosen to be the start location of the word and then as many adjacent boxes as are characters in the word are chosen to be the area of interest.

After all the locations of the hits are found, the final locations and sizes of the hit-boxes are saved for the results of the search and packed into a JSON string which is then sent back to the front-end web page.

In the *findCorr* function the found bounding boxes and lines are compared to each other. If the upper border of a bounding box is close enough to a text line they are considered to point to a same text line. Also if a found text line is too far away from the upper border, it is ignored as it is very probable that it points to some other text line.

The *findCorr* function loops through the interesting lines and tries to find the corresponding bounding boxes for them. During each iteration the minimum limit for the upper boundary of the bounding box is determined. The upper boundary is determined relatively to the current line that is processed. Because the *slines* vector contains the y-coordinates of the interesting lines, the minimum limit of the upper boundary is set to be either the y-coordinate of the previous line or if the line that is processed is the first element of the vector or 100 pixels away from the current line, the minimum limit of the border is set to be 70 pixels from the current line. This ensures that the minimum limit is not set to be too far away from the line and that the line is not assigned to a bounding box that belongs to an other text line.

Once the minimum limit for the bounding box is set, the vector containing the bounding box coordinates is sliced two times. First is sliced so that all the values that are greater than the minimum limit are left in the vector. Then it is sliced so that all the values that are smaller than the y-coordinate of the text line are left in the vector. This is happens in the *indices* function, which takes an lambda function as an argument. The structure of this method is somewhat complicated, but very efficient and it mimics very well MATLAB's *find()* function.

After the bounding boxes referring to the interesting text lines are found, the coordinates and measures of these boxes are taken and set to be the coordinates of the interesting bounding boxes and passed further as the correct boxes. In the current version of the code, the part where the bounding box coordinates are taken, fails sometimes causing the whole engine to crash and it needs to be fixed in the future versions.

```
cBB = HFun.indices(bbYs,lambda x: (x>minlim and x<slines[i]))

def indices(a, func):
    return [i for (i, val) in enumerate(a) if func(val)]
```

Figure 23. Coordinate slicer in Python mimicing MATLAB find() -function.

3.8. Returning the results to PHP front end

In the final processing phase of the engine, the hits are calculated and returned back to the front end PHP page. The equation 5 shows how the location of each character and the first character of each word is calculated.

The chosen lines and bounding boxes are divided into smaller boxes according to the number of characters on each line. Each bounding box is divided into equally wide small boxes. The position of a found character or the position of the first character of found word in the XML file marks the index and also the position of the hit in these boxes. If a single character is searched, the position of the hit on the text line on image is set to be on a small box which has the same index as the character. Instead if a word is searched, then the first character marks the box for the first character only. Then as many boxes as there are characters in the word are chosen adjacent to the first box and this area is marked as the hit for the whole word.

$$P_{X,Y} = \begin{cases} C_{XMLpos} * (X_{1bb} - X_{2bb})/N_c + X_{1bb} \\ Y_{center} \end{cases} \quad (5)$$

After the hits are calculated, the coordinates of the hits and bounding boxes are packed into a JSON strings. This single JSON string package is then send to the front end which draws the results on the original image and shows it to the user.

4. TESTING AND EXPERIMENTAL RESULTS

The previous chapters introduced and explained the document image analysis system developed for Orationes project. The system relies heavily on correctly parsed XML text files and rectified good quality document images. In this chapter the testing and results are shown and explained.

Initially there was an urgent need to get the pages from two chapters of the Orationes manuscript to get rectified: *The Conquest of Metals* and *Contention for Honour and Riches*. The 20 rectified images from The Conquest of Metals were used for testing, debugging and development. Because of the time limit and complexity and size of the whole engine, the testing was done alongside debugging and development. Unfortunately this left the engine in partially unstable with quite lots of phases where errors can happen. The testing was only functional testing and tested the whole process pipeline. No module testing was performed at all, therefore some pages might cause surprising problems and errors.

Because the engine first finds the words and characters from the parsed XML files and gets their locations in respect to the lines and characters in that file, it can always instantly tell, if the word or character exist in a certain page. However, knowing if it exists on a page, does not ensure that it is always found correctly from the document image. The XML file can contain parsing errors, thus placing the characters and words on wrong lines and the engine itself might not find the location correctly, because of possible segmentation errors.

The results consists of determining the precision of the engine and also goes through the processing time improvement results.

4.1. Processing time and improvements

The engine is a part of a web site application and it is not favorable for the web site, if the processing engine takes too much time to process the images. Therefore the engine processing time had to be improved.

In the early versions of the engine some solutions were very naive and programmed just to do what they were supposed to do. However, this resulted to long processing times and could take over 130 seconds to process a single page. This would have been intolerable even though the web site was not required to be responsive in real time.

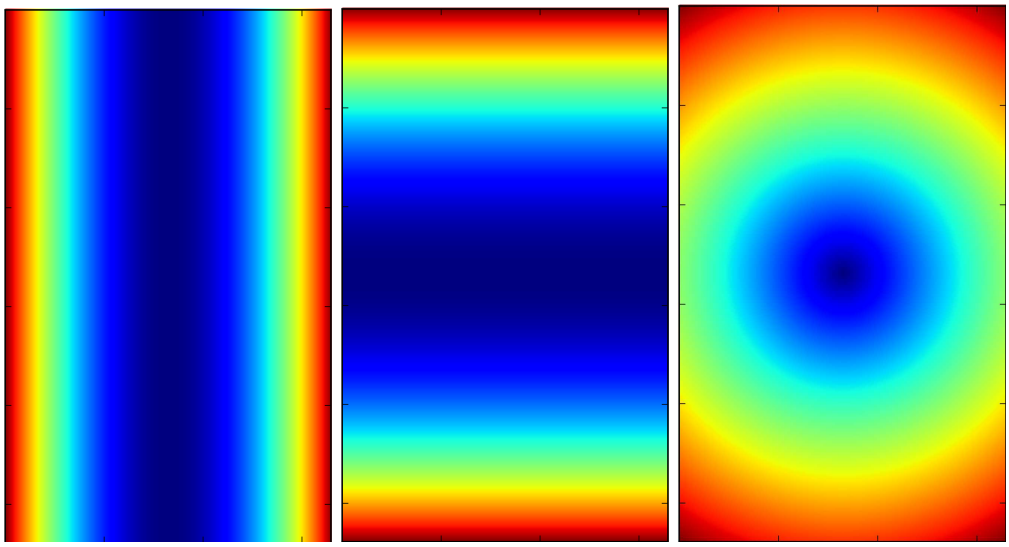
The processing times of each module and their functions were measured and it was noticed that the homomorphic filtering took long time to compute. That was mainly because it used naive *for*-loops to calculate the distance of each pixel from the center of the image and to create the filter for the whole page. This was corrected by using Python's vectorization methods.

The faster method calculated the distance of each vertical and horizontal line from the center line of the image separately and then combined these results into a single two dimensional vector representing each pixels' distances from the center.

Two masks were created. Each mask followed the same building principles. The vertical mask vector consisted of elements which showed the element's distance from the center element. The vector was then multiplied into a grid which width was the width of the page and height the length of the vector itself, thus the height of the

document image. The horizontal vector and grid was built respectively. These two grids were then combined, so that each element in the new grid contained the euclidian distance calculated from the respective values in the two previous vectors and resulted to a grid which values represented the distance to the center of the document image.

This simple solution removed the unnecessary for loops from the code that created the homomorphic filtering and increased drastically the processing time, dropping from 130 seconds to 15 seconds.



(a) Distances from the vertical center axis

(b) Distances from the horizontal center axis

(c) Distances from the image center

Figure 24. Dimension grids in homomorphics filter.

4.2. Precision

The interesting information is how well the system finds the correct locations for the hits. Because of the static segmentation process, each page is always processed similarly and that's why the characters are always placed on same locations.

As it has been explained in the previous chapters, the engine just segments the images, finds the text lines, their boundaries and then divides these bounding boxes into a smaller boxes which are assumed to contain just a single character. Then the assumed locations of the characters are determined from these boxes. The box with same index as the character in a set line is assumed to be the location of the character on the page. This means that the segmentation phase of the engine plays major role in determining the locations of the characters and words.

The results were obtained by running the engine on the rectified pages using some common letters and words, which show up quite often on those pages. The results were taken separately from pages which are segmented well, from pages where the segmentation fails in a way or another and then also combined results for the whole test set to get an approximate of how well the engine and this method can work on these types of pages.

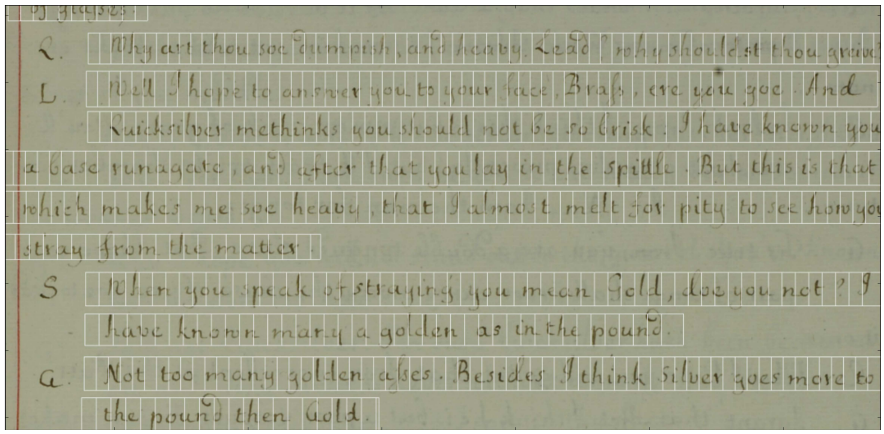


Figure 25. Segmented hit boxes.

Table 1. Precision for finding characters.

	Pure Hits	Near hits	Miss
Characters on well segmenting pages	40.0%	41.5%	18.5%
Characters on badly segmenting pages	12.2%	21.8%	66.0%

It was known that the engine cannot give precise results meaning that always the hits are not exactly right on top of the actual hit. That is why there are two types of hits shown for characters and words: hits and near hits. A hit is considered a pure hit, if the hit box touches the character that was searched. Near hits are hits that are very close to an actual hit, so that the user can spot the correct character or a word instantly by looking at the hit box. The hit is a near hit, if it is on the correct text line and only couple characters away from the original hit. If the hit box is three or more characters away or not on correct text line, it is considered to be a miss. Also if a hit box is shown on a correct character, but the hit box on that character is a hit box of another character, it was considered to be a miss as well, because it means that the hit locations are not reliable. The table 1 shows the precision of finding a single character.

For words, a hit is a hit if majority of the word stays in the hit box. Hit is a near hit, if the hit box hits only one character of the word. This type of definition is acceptable, because the user usually needs to see the approximate location of a word or character. Human eye can then spot the correct character very easily from the approximate location. The results for the precision, when finding words, are shown in table 2.

Table 2. Precision when finding words.

	Pure Hits	Near hits	Miss
Words on well segmenting pages	57,8%	38,9%	3,3%
Words on badly segmenting pages	35,4%	29,1%	35,4%

Table 3. Total precision of the engine.

	Pure Hits	Near hits	Miss
Total results for single characters	29,5%	29,6%	40,9%
Total results for words	47,1%	31,7%	21,1%

As was with the characters, naturally also the recognition rate of words diminish when the pages are not well segmented. However, the decrease in the recognition rate is not as drastic as with characters, because the density of the words on a page is smaller than with characters. This means that there are only several words on a page and larger percentage of them might be on properly segmented areas of a page, whereas there are usually tens of characters on a single page. That is why the character results shows better approximate of the total precision of the engine.

The results from the well segmented pages and badly segmented pages were combined to get an estimate of how well the engine performs currently on any given document image. This means that all the pure hits, near hits and misses from all pages were compared. This is shown in table 3. It can be argued, if it reasonable to combine the results for the whole engine as it is very obvious that the value of the results from badly segmented pages are close to zero in any case and has to be ignored.

From the results it can be seen that the engine performs quite well, when the segmentation succeeds. Even though the hits are not always exactly on top of the character that was searched, the combined result of hits and near hits shows that most of the wanted characters are easily spotted by the viewer. In case of words, the results are even better. This is mainly, because the hit box of a word is much larger and there are always more characters on a page than words, thus there are less words which has to be spotted by the human eye.

In practice the recognition rate would be even better for the user, because in the final product, the engine and PHP front end, would not let the through the badly segmented pages. On the other hand, this would be inconvenient for the user as there would not be any results for those pages and the user would have to go through them manually.

The errors are analysed and explained in the discussion section.

5. DISCUSSION

5.1. Problems and future development

During the implementation and testing phase, many problems were found. By developing these problematic parts, it could be possible to make the engine more robust, faster and more intelligent. In ideal case, the engine would be able to automatically rectify the pages, locate the interesting parts more accurately and use actual OCR to find the wanted characters and words from the pages.

Based on the developed method, testing and on previously done research it can be said that it is possible to create such processing engine, though this kind of method would probably be somewhat slower than the method shown in this thesis. However, creating such system would require lots of time and resources and unfortunately neither of those were available for this thesis work. This section focuses on the parts which needs to be developed in order to make the system better. Making a good system requires development in every stages of this engine.

As stated in previous chapter, the engine is heavily dependent of good XML parsing and page segmentation process. Because the XML parsing is not part of the engine, this chapter focus on pointing out the problems in the engine found during the development.

The engine consists of several steps, where small errors and problems can lead to a bigger error in the next step. This chapter focuses on the main problems found out from this method.

5.1.1. Challenges in taking the document images and page rectification

Taking the document images from the manuscript wasn't a part of this work, but the quality of the images affected greatly the project, so it is good to say few words about taking the document images. It is after all a very significant part of all kinds of document image analysis systems.

In the Orationes project it wasn't possible to take the document images by using a flat bed scanner, thus the document images had to be taken with a regular DSLR camera. When the images are taken with a camera, it is really important to note down the photography geometry and keep it fixed all the time during the imaging. This would ensure that all the document images are affected by the same conditions. From the imaging geometry it is important to know the basic information about the camera: focal length, resolution, distance to the document, original width and height of the document and the complete EXIF info from the camera. With these information it is possible to reconstruct the photographing situation and calculate possible distortions from the image and do automatic page rectification.

This information was not available during the Orationes project, so the automatic page rectification process for this thesis was first tried to do with estimates of these measurements. But because the measures and conditions seemed to differ between some pages, causing lots of problems, and the page rectification module didn't perform well and fast enough for the whole system, the pages were rectified manually with the method shown in section 3.1.

5.1.2. Image enhancement and image processing

Image enhancement and image processing was done mainly by using simple and basic image processing operations. It was thought that with these operations it could be possible to segment and process the image well enough to get good results. Also the image enhancement uses homomorphic filtering and simple contrast stretching to produce an image, which contains mostly the data from the text areas only.

All of these methods are not good enough as static processing methods, because the conditions between the images changes all the time. However, because of the size of the project and the time limits, too many parts were built to use static values. In this kind of cascading system, where output from another step is the input for the second, this static nature can lead to huge errors in the final output.

The image enhancement part performs quite well without any significant error possibilities. The homomorphic filtering part fixes majority of the illumination flaws. The filtered product is then further enhanced by the contrast stretching, which is just finetuning the contrast near the text. More problems occur in the actual image segmentation phase.

Binarization and labeling

First step in image segmentation is the image binarization. Because there was no time to implement any actual adaptive binarization method, the binarization is set to rely heavily on the well performing homomorphic filtering and contrast stretching. It uses a static threshold of 0.95 to binarize the image. Even though this static threshold performs well with the images in this project, it is first step where things might start going wrong. In the future, it would be better to change the binarization be adaptive.

After the binarization the regions are labeled. Before the morphological operations are applied to the regions, the largest region is removed, because it is assumed that the margin areas will always form the biggest region in this phase. Once again the engine may fail here. Firstly, after the binarization the margin area might be connected to the text lines in one or more places, thus the removal of the margin area might remove some characters or even words. Secondly, the margin area might not always be continuous, thus removing the largest region might not always remove all the margins. This was tried to compensate by later removing too high regions from the image, but there were some cases where all the vertical regions were not removed, because they were not thought high enough. When the regions are then grown, the remaining margin parts might cause several regions from different text lines to grow together. This will then force those text lines to be considered as a single text line by the engine. That's why, if taking this kind of segmentation route in the future, there should be more reliable way to remove the margins and every other non-text region, so that only the text area remains.

Cleaning and region growing

After the margin removal, some smaller areas are removed and then the text lines are grown together as well as possible. Once again the removal and growing steps use static values and structure elements. From the average dimensions of the text lines

and character areas, it was assumed that regions smaller than 50 pixels, very unlikely belong to any text line or represents a character on the page. The erosion process is not causing much problems, but sometimes the small erosion done to the text lines is not enough to separate two vertically adjacent text lines enough from each other and thus they might grow together in the dilation process. The dilation performed on the text lines work well, when the characters are close together on a text line. Pages, which consists of dialogue or lists or some other type of text, where words might not be close enough to each other, cause problems. In these pages the text areas on same text line are not always connected together and there are then two or more regions representing text areas in that text line. These regions are easily recognized to belong to the same text line, but the problem is that the characters on those text lines are not uniformly spread throughout the whole text line. However, the engine is built on this assumption, thus the non uniformly spread characters forces the hit boxes not to align correctly with the actual text.

Problems in the text line search

The text line search is done by using radon transform alongside the horizontal axis of a page. This radon transform is then further transformed to frequency domain and the frequency domain representation is even further filtered, or rather sliced, so that there wouldn't be frequencies smaller than the average text line height left in the Radon transform.

Even though this approach works quite well, it is once again using a static number to filter the radon transform's frequency representation. This step function filtering causes lots of rippel effect to the start and end parts of the radon transform signal. On pages where there are long text lines, the rippel effect doesn't affect the outcome very much, because the amplitude of the text lines surpasses the amplitude of the rippel waves, but when there are shorter text lines on a page, the maximas on the rippel effect may accidentally be considered to be actual text lines.

Another problem caused by this solution is, that it is based on the assumption that the text lines are periodical and uniformly distributed. In the Orationes manuscript there are pages where this assumption does not apply. The radon transform from these pages contains flat areas where there are not text lines, but when it is sliced, the sliced radon transform contains some maximas on these blank non-text areas. These maximas can then be incorrectly recognized as correct maximas and thus as text lines. These ghost text lines cause problems when aligning the text boxes and text lines. Figure 31 shows how the periodic assumption causes some extra and thus false text lines to be found from the image.

This problem is tried to be fixed in the text line post processing step, but it is not always trivial. Besides because the time ran out with the project, the post processing step is very naive and incomplete. It is not always able to handle all the situations, which might arise in the previous steps.

Problems in post processing

The *processlines* and *padlines* functions performs the post processing steps. *Processlines* function compares the number of rows, which presents the text lines, in XML

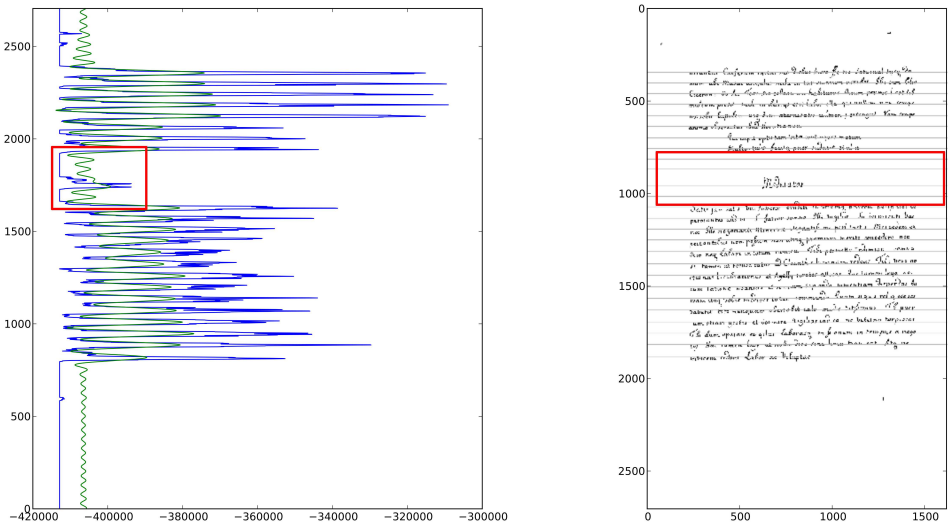


Figure 26. False text lines caused by periodical assumption.

to the number of text lines found from the document image. If the numbers are equal it is assumed, that all the text lines were found. However, if the numbers are not equal, it tries to determine which text lines are not found or which found text lines are extra text lines.

It was known before, that short text lines are more unlikely to be seen by the text line finding function, so if there are less text lines found from the image than what's in the XML, it is thought that the shortest text lines haven't been found. Unfortunately, if there are too many text lines found, the function does nothing, as there was not enough time to implement a method to process that case.

The problems in this function is that it is based on enlightened guesses and assumptions about the behavior of the engine and characteristics of the Orationes project document images. The first case assumes that, if the number of found text lines equals to the number of text lines in XML, the found text lines corresponds to the text lines in XML. Also when there are not enough text lines found from the image, the missed text lines might also be some other text lines than the shorter ones.

While the assumptions are true in most pages in the test and development set, it might not always be true. However there's no method to check, if these assumptions are true or not. When they fail, the results will be placed on wrong text lines and thus the results on such pages lose their value completely. Also when there are too many text lines found, the results fail too, because that case is not handled at all.

Problems and development in the hit box phase

In the hit box finding part, that's explained in section 3.7, the found text lines and bounding boxes are compared to each other and determined, which bounding box corresponds to which found text line. This mechanic tries to ensure that found text lines are real text lines and matches to the correct text lines, thus removing as many false positive text lines as possible.

The mechanic, that determines whether a bounding box belongs to a certain text line and vice versa, works that it simply compares the coordinate of a found text line to the

upper border of all the bounding boxes. If any of the upper borders are close enough to the found text line, they are assumed to belong to the same text line.

While this method never failed to find the correspondences between the text lines and boxes during testing and development, it was noticed, that some times it failed to create the $2\text{x}N$ correlation matrix, which collects the information of which found text line center coordinates corresponds to which bounding box. So there's a bug in the code in this part. When this fails, obviously the engine itself fails. Sometimes even, if this fails, the engine is still able to output some results, but the results might not be reliable.

The second thing that happens in this phase is, that each bounding box is divided into equal parts. Here it is assumed that the characters are uniformly divided on each text line and that each character takes approximately the same space, thus each bounding box is divided into equal size small boxes, which number is the same as the number of characters on that text line.

Even in the test set there were lots of pages where this uniform character distribution didn't happen on text lines. Usually they were text lines, which first character presented the speaker and then the rest of the text line was, what that speaker was about to say. Between the actual text line and speaker notation there are quite long empty space. On some pages these single characters are removed during the clean up phase. When this happens, the actual bounding box will contain less characters than what is got from the XML. This causes the bounding boxes to have too many smaller boxes inside them, because the character count still assumes that all the characters from the XML are still represented in the text line.

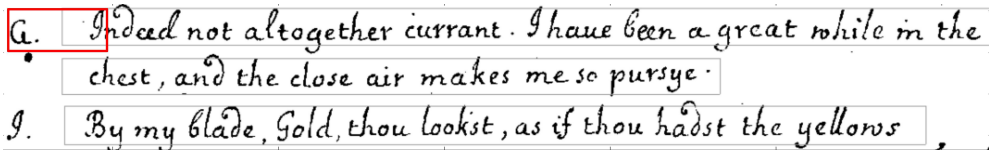
On the other hand, when the first characters are not cleaned during the clean up phase, the subdivision of the bounding box can not handle the bigger empty space between the first character and rest of the text line. This causes the text line to have correct amount of sub boxes, but they are larger than they should be and also they are not correctly aligned with the characters. This could be probably solved by having a notation in XML that would show where there are long empty spaces between words on a text line.

Sometimes the character count for a text line is mistakenly thought to be character count from some other line and the subdivision divides a bounding box to too small and too many sections. The error, caused by a incorrectly handled wide space and incorrect character count, can be seen in figure 27.

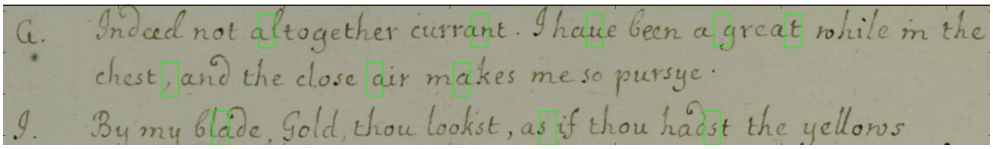
5.1.3. General thoughts of the whole engine

The last phase in the engine is sending the results via a JSON packet back to the PHP front end. This phase is quite trivial, because it just takes the calculated results and packs them into a regular JSON packet.

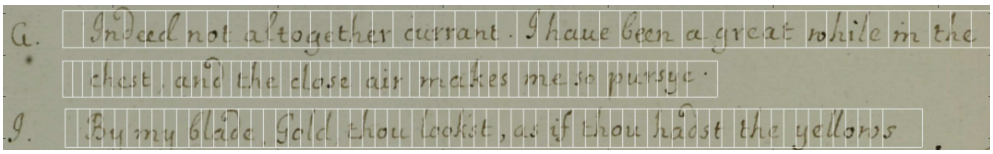
All in all, the whole engine is very much based of enlightened assumptions and knowledge of the conditions the engine is working on. However, there are many phases in the engine, where one or more of these assumptions don't apply to every page. In worst possible option, all these assumptions might cause a really erroneous cascade and even crash the whole engine. Therefore many dynamic and often more complicated solutions would be needed to adapt to varying situations and ensure that the engine



(a) Wide space between first character and the actual text line



(b) Incorrectly set bounding box causes offset on the subdivided hit boxes for character 'a'



(c) All the hit boxes on the text lines

Figure 27. Wide space and wrong character count causing problems in subdividing the bounding boxes.

would perform well in every phase. Also the engine is lacking many error handling procedures, which are essential to keep the engine running despite possible errors.

5.2. Discussion

The problems and possible future development steps were discussed in previous chapter. In this chapter it is discussed generally about the thesis itself and how viable this kind of project was as a master's thesis.

The original goal of this thesis was to create a platform independet system, which can automatically rectify given document images and then perform optical character recognition. While this seemed to be possible to achieve on paper, in practice the task was enormous and way too complicated and large for a master's thesis. Creating such system would need lots of time and more than just one person working on it.

At first the system was to be developed on MATLAB, but when came the time to get a working demo out, it was realized, that MATLAB wasn't the best possible environment to develope the system on, if it was to be run on different platforms. It was then decided that the whole system would be developed using Python, because it enabled great platform independecy. The port from MATLAB to Python wasn't very easy and trivial, because Python lacks lots of functions that are built in to MATLAB, so many of these features and functions had to be programmed into the engine. Python Scipy library was very helpful, as it has been thought to be a modern predecessor to MATLAB, but still it is really far from MATLAB and what can be done with it. Part of the development time was spent on creating equivalent functions to Python and this reduced the amount of time, that could be spent on actual development of the engine.

The current developed system, is not very useful yet. However, it works as a great pilot and shows, which parts need further development and points out how to possibly

fix the problems. Also one can see from it, which MATLAB like things can be done with Python. The current system is fast, but is not very precise. It gives mostly directional results, which helps the user to locate the actual interesting points from the document images.

Originally, as said before, the goal was to perform the rectification of the pages automatically by the engine. However, the researched methods, which were thought to be suitable and easy to implement for this project, ended up being very confusing and complicated. Besides the papers, which tried to describe the process, were somewhat misleading or rather not informative enough. Many months were spent on this research and eventually the automatic rectification was changed to manual rectification in order to continue to next step in the project.

The next step was to research ways to perform optical character recognition on the document images. At first it was thought that simple template matching could be enough to achieve some results. However, it was soon realized, that template matching performed poorly on handwritten text, because there are too much variation among individual characters. Also it was really hard to distinguis characters with similar charasteristics, like a and o or i and j for example, from each other by this method. Also it was very prone to errors and false positives. This method would have needed more research and work to get it work.

After seeing the inconsistency in the template matching, it was thought that, probably it could be possible to narrow down the search and areas on which to perform the template matching. This lead the development to the page segmentation and text line search. Still, despite segmenting the pages and thus eliminating many false positives from the non-text area, the template matching performed poorly. At this point time was really runnng out, so it was decided to take an approach to try and make some sort of text alignment engine, which would use the information from the XML file in combination with the page segmentation and found text areas. This approach performed better than the initial template matching approach, even though it is also very prone to errors.

Working on this kind of project was interesting and challenging. Still such project is way too big and complicated to be done by a single man only. Also the whole project was heavily disturbed by the urgent need to get something done. This caused lots of stress and interrupted the actual research process and eventually forced to use naive solutions. In future, this scale projects would probably be better to divide into smaller master's thesis works. For example, this project could have been divided into smaller individual works, which would have consisted of the automatic page rectification, document image enhancement and binarization, document image segmentation and optical character recognition on handwritten text. This way each part would have got the amount of research and work they would have deserved.

Despite the engine developed in this project is far from being an useful product, it shows that such system could be made, if given enough time and resources. The current engine can be used by scholars and historians to some extent to achieve information of the character and word charasteristics in the Orationes manuscript.

6. SUMMARY

This master's thesis was about development of an basic document image segmentation and text alignment engine for document images with handwritten text in Orationes project. The original goal was to develope an optical character recognition system, but eventually it was thought that a document image analysis and text alignment system would be sufficient. One goal was also to research, what kind of problems one would face when developing such system and give guidance of how to avoid and fix these problems.

The system was originally programmed with MATLAB, but later ported to Python programmin language to achieve platform independency. First part was the rectification of the distorted document images. Second was document image enhancement and noise reduction. Third part was document image segmentation and text line search. Fourth part was the text alignment in form of subdividing the segmented text areas by using the information about the text gathered from XML transcript. Final part was sending the achieved results to the PHP front end.

According to the research and implementation, it can be said that a system presented in this thesis could be feasible and more accurate results could be achieved. Python programming language is great for making platform independet MATLAB-like programs, but many MATLAB functions and features need to be implemented by the developer, because Python Scipy library doesn't have all similar features implemented yet. However, developing a fully automatic system with page rectification and better performance, would need more research, resources and time.

7. REFERENCES

[1] Li N. (1993) An implementation of ocr system based on skeleton matching. University of Kent, Computing Laboratory.

[2] Chaudhuri B. & Pal U. (1998) A complete printed Bangla OCR system. Pattern recognition 31, pp. 531–549.

[3] Ntzios K., Gatos B., Pratikakis I., Konidaris T. & Perantonis S.J. (2007) An old greek handwritten OCR system based on an efficient segmentation-free approach. International Journal of Document Analysis and Recognition (IJDAR) 9, pp. 179–192.

[4] Brown M.S. & Seales W.B. (2001) Document restoration using 3D shape: a general deskewing algorithm for arbitrarily warped documents. In: Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on, vol. 2, IEEE, vol. 2, pp. 367–374.

[5] Yamashita A., Kawarago A., Kaneko T. & Miura K.T. (2004) Shape reconstruction and image restoration for non-flat surfaces of documents with a stereo vision system. In: Pattern Recognition, 2004. ICPR 2004. Proceedings. 17th International Conference on, vol. 1, IEEE, vol. 1, pp. 482–485.

[6] Bolan S. (2012) Document image enhancement. Ph.D. thesis, National University Of Singapore, School of Computing.

[7] Hicks J. & Eby Jr J. (1979) Signal processing techniques in commercially available high-speed optical character reading equipment. In: Techincal Symposium East, International Society for Optics and Photonics, pp. 205–216.

[8] Abutaleb A. & Eloteifi A. (1988) Automatic Thresholding of Gray-Level Pictures Using 2-D Entropy. In: 31st Annual Technical Symposium, International Society for Optics and Photonics, pp. 29–35.

[9] Otsu N. (1975) A threshold selection method from gray-level histograms. Automatica 11, pp. 23–27.

[10] Sauvola J. & Pietikäinen M. (2000) Adaptive document image binarization. Pattern Recognition 33, pp. 225–236.

[11] Niblack W. (1985) An introduction to digital image processing. Strandberg Publishing Company, 115–116 p.

[12] Gatos B., Pratikakis I. & Perantonis S.J. (2006) Adaptive degraded document image binarization. Pattern recognition 39, pp. 317–327.

[13] Brown M.S. & Seales W.B. (2004) Image restoration of arbitrarily warped documents. Pattern Analysis and Machine Intelligence, IEEE Transactions on 26, pp. 1295–1306.

[14] Zhang Z., Tan C.L. & Fan L. (2004) Restoration of curved document images through 3D shape modeling. In: Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings. 2004 IEEE Computer Society Conference on, vol. 1, IEEE, vol. 1, pp. I–10.

[15] Cao H., Ding X. & Liu C. (2003) A cylindrical surface model to rectify the bound document image. In: Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on, IEEE, pp. 228–233.

[16] Cao H., Ding X. & Liu C. (2003) Rectifying the bound document image captured by the camera: A model based approach. In: Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on, IEEE, pp. 71–75.

[17] Liang J., DeMenthon D. & Doermann D. (2005) Flattening curved documents in images. In: Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, vol. 2, IEEE, vol. 2, pp. 338–345.

[18] Zhang L. & Tan C.L. (2005) Warped image restoration with applications to digital libraries. In: Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on, IEEE, pp. 192–196.

[19] Gumerov N., Zandifar A., Duraiswami R. & Davis L.S. (2004) Structure of applicable surfaces from single views. In: Computer Vision-ECCV 2004, Springer, pp. 482–496.

[20] Zhang Z., Liang X. & Ma Y. (2011) Unwrapping low-rank textures on generalized cylindrical surfaces. In: Computer Vision (ICCV), 2011 IEEE International Conference on, IEEE, pp. 1347–1354.

[21] Stamatopoulos N., Gatos B., Pratikakis I. & Perantonis S.J. (2008) A two-step dewarping of camera document images. In: Document Analysis Systems, 2008. DAS'08. The Eighth IAPR International Workshop on, IEEE, pp. 209–216.

[22] Das A., Chowdhuri S. & Chanda B. (2002) A complete system for document image segmentation. In: Proceedings of national Workshop on Computer Vision, Graphics and Image Processing (WVGIP2002), pp. 9–16.

[23] Kumar S., Gupta R., Khanna N., Chaudhury S. & Joshi S.D. (2007) Text extraction and document image segmentation using matched wavelets and mrf model. Image Processing, IEEE Transactions on 16, pp. 2117–2128.

[24] Gatos B., Mantzaris S. & Antonacopoulos A. (2001) First international newspaper segmentation contest. In: Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on, IEEE, pp. 1190–1194.

[25] Antonacopoulos A., Gatos B. & Bridson D. (2005) ICDAR2005 Page Segmentation Competition. In: Document Analysis and Recognition (ICDAR'2005). 8th International Conference, IEEE, pp. 75–79.

[26] Antonacopoulos A., Gatos B. & Bridson D. (2007) ICDAR2007 Page Segmentation Competition. In: Document Analysis and Recognition (ICDAR'2007). 9th International Conference, IEEE, pp. 1279–1283.

[27] Likforman-Sulem L., Hanimyan A. & Faure C. (1995) A Hough based algorithm for extracting text lines in handwritten documents. In: Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on, vol. 2, IEEE, vol. 2, pp. 774–777.

[28] KeshavarzBahaghighat M. & Mohammadi J. (2012) Novel Approach for Baseline Detection and Text Line Segmentation. International Journal of Computer Applications 51, pp. 9–16.

[29] Bazzi I., Schwartz R. & Makhoul J. (1999) An omnifont open-vocabulary OCR system for English and Arabic. Pattern Analysis and Machine Intelligence, IEEE Transactions on 21, pp. 495–504.

[30] Shi Z. & Govindaraju V. (2004) Line separation for complex document images using fuzzy runlength. In: Document Image Analysis for Libraries, 2004. Proceedings. First International Workshop on, IEEE, pp. 306–312.

[31] Likforman-Sulem L., Zahour A. & Taconet B. (2007) Text line segmentation of historical documents: a survey. International Journal of Document Analysis and Recognition (IJDAR) 9, pp. 123–138.

[32] Bahaghighat M.K. & Mohammadi J. (2012) Novel Approach for Baseline Detection and Text Line Segmentation. International Journal of Computer Applications 51.

[33] Casey R.G. & Lecolinet E. (1996) A survey of methods and strategies in character segmentation. Pattern Analysis and Machine Intelligence, IEEE Transactions on 18, pp. 690–706.

[34] Sari T., Souici L. & Sellami M. (2002) Off-line handwritten Arabic character segmentation algorithm: ACSA. In: Frontiers in Handwriting Recognition, 2002. Proceedings. Eighth International Workshop on, IEEE, pp. 452–457.

[35] Louloudis G., Gatos B., Pratikakis I. & Halatsis C. (2009) Text line and word segmentation of handwritten documents. Pattern Recognition 42, pp. 3169–3183.

[36] Farag R.F.H. (1979) Word-level recognition of cursive script. IEEE transactions on Computers 28, pp. 172–175.