

Creation of a customised character recognition application

Frida Sandgren
frisand@stp.ling.uu.se

Master's thesis in Computational Linguistics
Språkteknologiprogrammet
(Language Engineering Programme)
Uppsala University · Department of Linguistics and Philology

November 26, 2004

Supervisor: Mats Dahllöf, Uppsala University

Abstract

This master's thesis describes the work in creating a customised optical character recognition (OCR) application; intended for use in digitisation of theses submitted to the Uppsala University in the 18th and 19th centuries. For this purpose, an open source software called *Gamera* has been used for recognition and classification of the characters in the documents. The software provides specific algorithms for analysis of heritage documents and is designed to be used as a tool for creating domain-specific (i.e. customised) recognition applications.

By using the Gamera classifier training interface, classifier data was created which reflects the characters in the particular theses. The data can then be used in automatic recognition of 'new' characters, by loading it into one of Gamera's classifiers. The output of Gamera are sets of classified glyphs (i.e. small images of characters), stored in an XML-based format.

However, as OCR typically involves translation of images of text into a machine-readable format, a complementary OCR-module was needed. For this purpose, an external Gamera module for *page segmentation* was modified and used.

In addition, a script for control of the OCR-process was created, which initiates the page segmentation on Gamera classified glyphs. The result is written to text files.

Finally, in a test for recognition accuracy, one of the theses was used for creation of training data and for test of data. The result from the test show an average accuracy rate of 82% and that there is a need for a better pre-processing module which removes more noise from the images, as well as recognises different character sizes in the images before they are run by the OCR-process.

Contents

1. Introduction	5
1.2 Background.....	5
1.2.1 Project background.....	5
1.2.2 The heritage theses.....	6
1.2.3 Gamera	6
1.2.4 The process of digitisation and optical character recognition.....	7
1.2.4.1 Scanning and pre-processing	7
1.2.4.2 Segmentation and classification	7
1.2.4.3 Page segmentation, translation and output	8
1.3 Purpose and outline.....	8
2. Creation of classifier training data	10
2.1 Using classifier data	10
2.2 Broken and touching characters	11
2.3 Optimisation of training data.....	12
3. Page segmentation.....	13
3.1 Identification of text in images.....	13
3.2 A script for recognition of words	13
3.3 Modifications on the page segmentation module.....	14
3.2.1 Translation into ASCII and Unicode	14
4. Creation of a script for the OCR –process and for a user interface.....	18
5. Recognition accuracy	20
5.1 Results.....	21
5.2 Other OCR-software	21
6. Discussion	23
6.1 Creation of representative training sets.....	23
6.2. Translation.....	23
6.3 Character sizes & isomorphic glyphs.....	24
6.4 Prerequisites	24
7. Future work	25
7.1 Pre-processing.....	25
7.2 Noise	25
8. Concluding remarks	26

Acknowledgements

I would like to express my sincerest gratitude towards the people at the Electronic Publishing Centre at the Uppsala University Library for all support during my work. Many thanks for giving me such useful advices and for sharing brilliant ideas and interesting discussions. Also, many thanks for guiding me through the imaging software.

1. Introduction

As the representational capabilities of computers have increased dramatically in recent years, more efforts are put into digitising sources of historical interest and disseminating them via the internet. Issues such as the importance of accessibility, re-use and preservation often serve as starting point in approaching the digitisation process.

Digitisation of text documents is often combined with the process of optical character recognition (OCR). Optical character recognition typically involves the process of translating digitised images of text (usually created by a scanner) into a machine-readable format (such as ASCII or Unicode). With regard to the document content, such a format has the advantage of being searchable, in contrast to image formats.

However, analysis and optical character recognition of ‘difficult’ heritage documents is not straightforward. New techniques are required for such material, as a contrast to recent, machine-written documents (for which there are numerous OCR software available today). Examples of issues that need to be dealt with in character recognition of heritage documents are:

- Degradation of paper, which often results in high occurrence of noise in the digitised images, or fragmented (broken) characters.
- Characters are not machine-written. If they are manually set, this will result in, for instance, varying space sizes between characters or (accidentally) touching characters.

This thesis describes the work in creating a customised OCR –application; intended for use in digitisation¹ of a collection of theses submitted to the University of Uppsala in the 18th and 19th century. For this purpose, an open source software called *Gamera*² has been used as a tool for recognition and classification of characters used in the theses. The software provides specific algorithms for analysis of heritage documents, though it is not a complete OCR-software. In order to be able to perform the OCR-process on images of the theses, a complementary module needed to be applied to Gamera and modified to suit the theses.

1.2 Background

1.2.1 Project background

The work, which can be seen as a pilot project, has been conducted at the Electronic Publishing Centre at the Uppsala University Library, where the theses are held. The Gamera system is being developed at the Johns Hopkins University in Baltimore, Maryland. Among other OCR-software available, Gamera was selected as the most appropriate. It could be used to build a customised

¹ The use of the terms ‘digitalisation’ and ‘digitisation’ vary. In this paper ‘digitisation’ is used, referring to the process of creating digital copies of the theses’ pages, using a scanner.

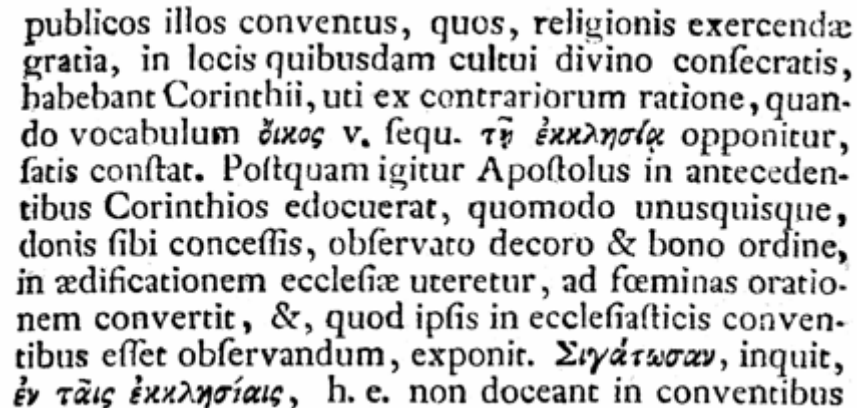
² For information about the Gamera software, see <http://dkc.mse.jhu.edu/gamera> ([1])

recognition application for the theses, as well as providing means for dealing with heritage documents.

1.2.2 The heritage theses

The collection of theses held by Uppsala University Library is extensive. Between 1700 and 1855 11,020 theses were submitted to the university [Örneholm, U., Östlund, K. 2000]. The collection consists of documents mainly written in the Latin language, which was commonly used in scientific literature at the time. Many of the theses are written by Johanne Ihre. Characteristics of these theses are, for instance, their size of 150x190mm (which is slightly smaller than A5) and the average number of pages is approximately 20 [Östlund, K. 2000]. The title pages often contain abbreviations such as D.D (Deo Duce) or A.D. (Auspice Deo) at the top. They also involve the title of work and often a statement that it has been approved by the university faculty [Östlund, K. 2000].

As for the text in the theses, some parts occur in several different languages such as Greek, Swedish or Hebrew. In creation of digital images of the document pages (using a scanner), the images will have a high occurrence of noise due to the degradation of the paper. Also, the images will contain many fragmented (broken) and (accidentally) touching characters, also due to the degradation or manual setting of characters.



publicos illos conventus, quos, religionis exercendæ gratia, in locis quibusdam cultui divino consecratis, habebant Corinthii, uti ex contrariorum ratione, quando vocabulum $\delta\acute{\iota}\kappa\omicron\varsigma$ v. sequ. τῆ ἐκκλησίᾳ opponitur, fatis constat. Postquam igitur Apostolus in antecedentibus Corinthios edocuerat, quomodo unusquisque, donis sibi concessis, observato decoro & bono ordine, in ædificationem ecclesiæ uteretur, ad fœminas orationem convertit, &, quod ipsis in ecclesiasticis conventibus esset observandum, exponit. Σιγάτωσαν, inquit, ἐν ταῖς ἐκκλησίαις, h. e. non doceant in conventibus

Figure 1. Image of a page of a thesis.

1.2.3 Gamera

The Gamera system is intended for creation of domain-specific character recognition applications³. It provides a classifier training interface, by which classifier data can be created. The training data will reflect the domain of characters in the documents and can be used in recognition and classification of ‘new’ characters, using the Gamera classifier API. The system is intended to be used as a tool for building recognition applications for a number of diverse materials, such as text or sheet music. (Gamera was also originally designed for recognition of sheet music, i.e. as an optical music recognition system). The core part of the Gamera system is the *segmentation* of images into its components or *glyphs* (i.e. small images of the symbols in the image) and classification of glyphs. For images of text, this implies that each character is

³ See <http://dkc.mse.jhu.edu/gamera> ([1])

recognised as a connected component (composed of connected pixels) in the image. Using the Gamera training interface, these glyphs can further be manually classified (by assigning names) and saved as training data. For storage of training data, Gamera uses an XML based format [3].

As for the theses, any training data will consist of sets of small (binary) images of characters, where each character is defined by, for instance, its given (id-) name and feature data. The features can be thought of as attributes which characterise glyphs differently from others; such as *area* or *compactness* [3]. Moreover, classified glyphs are also defined by co-ordinates, i.e. location in the image. The XML files of training data are also the *output* of Gamera.

The training data can be loaded into a Gamera K-nearest neighbour classifier and used in classification of 'new' characters. In example-based categorisation, symbols are identified by their similarity to one or more of the stored examples [Fujinaga, I. 1996]. In Gamera classification, the features of a given symbol, determines the class to which it belongs. Each symbol has a feature vector and distances between the feature vector of an unclassified symbol and previously classified symbol are calculated. The class represented by the closest neighbour is assigned to the unclassified symbol [Fujinaga, I. 1996].

1.2.4 The process of digitisation and optical character recognition

The process of digitisation and character recognition of the documents in this project would involve the following basic steps:

- Creation of digitised images of the document pages, using a scanner.
- Pre-processing of the images (e.g. sharpening or de-skewing).
- Segmentation of the images into its components (glyphs). Characters in the images need to be recognised as connected components, i.e. as connected pixels.
- Classification of the glyphs.
- Segmentation of the page structure. The text in the images needs to be reconstructed in terms of sections, lines and characters etc.
- Translation of the classified glyphs into machine-readable format. In this case it would involve translation into Unicode⁴ format. (As parts of the text in the theses may occur in various languages, the Unicode format is necessary).
- Output, i.e. writing the translated glyphs to resulting text files.

1.2.4.1 Scanning and pre-processing

Any scanning of pages of the theses was performed using commercial software for digital imaging (Agfa Fotolook 3.5 and Adobe Photoshop 7.0). The resolution set for images normally has an impact on the recognition accuracy of OCR software. In this project, images of the pages were digitised at 300 or 600 dpi (dots per inch) of resolution. In the case of Gamera, it is recommended that the images have a resolution of 600 dpi.

1.2.4.2 Segmentation and classification

As for the segmentation of the images into components (glyphs) and classification of glyphs, the Gamera training interface was used. Images were loaded and segmented into their components by the Gamera cc-analysis function (connected component analysis). Using Gamera, glyphs were

⁴ For more information, see www.unicode.org.

classified both manually and automatically, in two different workflows. In the first workflow, the glyphs were manually classified and stored as classifier training data. The second workflow involved using the data in automatic classification of ‘new’ characters.

1.2.4.3 Page segmentation, translation and output

The final parts of the character recognition process of the theses would involve segmentation of the page structure (i.e. identification of regions of text pertaining to the entire page) and translation of the classified glyphs into a machine-readable format; resulting in text files. In using Gamera, it is relevant to stress the difference between the process of classification of glyphs and the translation of these glyphs into a machine-readable representation. In the Gamera graphical user interface, there is currently no implemented function for default-translation; nor for the reconstruction of the text in the image. This is because such a function would be dependent on the appearance of the current document (characters used, layout etc.).

In March 2004, a script for recognition of words (translation of glyphs into ASCII format and grouping them into words) was shared on the Gamera mailing list. This script was used in the beginning of the project, for output of the classified glyphs.

Later on, in April 2004, an implementation of a *page segmentation algorithm*⁵ for the Gamera system was shared on the Gamera mailing lists and added as an external module in new versions of Gamera. This module for page segmentation then replaced the script for recognition of words in the project. The latter module is intended for recognition of the entire text of heritage (*roman* ⁶) documents, and includes a function for default-translation of classified glyphs. The module was not implemented as part of the Gamera interface, but left open for domain-specific usage.

The page segmentation module makes use of the specific attributes of Gamera classified glyphs (e.g. data about location in the image) in reconstruction of the text. Though, in order to reconstruct the text (and characters) used in the particular theses, the module needed to be complemented and modified.

1.3 Purpose and outline

The overall motive of this degree project has been the creation of a customised optical character recognition application, using the Gamera classifying system and the Gamera module for page segmentation. It aims at the creation of an application which can be used in optical character recognition of the theses submitted to the Uppsala University.

The main focus has been the modification and customisation of the module for page segmentation. Additionally, a script for control of the OCR-process needed to be created. This script imports the (modified) page segmentation module for use with the Gamera classifying system. It provides a user interface and writes the result of the page segmentation to text files. The page segmentation module and resulting script makes for the final part in the process of optical character recognition of the theses; it is the link between Gamera recognised and classified characters, and text files in machine-readable format. The work can further be divided into four main tasks:

⁵ Karl Macmillan, one of the developers of Gamera, is the author of the page segmentation module for Gamera.

⁶The module for text segmentation (*roman_text.py*) can be found in the gamera installed directory: `installed\Python23\Lib\site-packages\gamera\`.

1. The creation of a small set of classifier training data in order to enable the work with the page segmentation module. The procedure of creating training data, as well as problems and how they have been dealt with, are described in section 2.
2. The modification and customisation of the page segmentation module. This mainly involves creation of a function which enables for translation of the characters used in the theses into machine-readable format, as well as some modifications on functions concerning the analysis of lines and combination of text strings. The work with the page segmentation module is described in section 3.
3. The creation of a script for control of the OCR-process. Via the interface, any image of the theses can be imported, as well as any classifier training data. As the page segmentation module is run, the optical character recognition process on the image is initiated, and finally the result is written to a text file. The script is described in section 4.
4. A small test of recognition accuracy. This includes creation of a larger set of classifier training data, and test of data. (Section 5).

2. Creation of classifier training data

In any creation of classifier training data, images of the theses⁷ were loaded into the Gamera training interface and processed by the Gamera cc-analysis (connected components analysis). All recognised components (glyphs) of an image are shown separately as small images, together with the original image. A glyph can then be classified (named) by selecting its image (or by clicking in the original image) and typing its name into a textbox. As an example, in including one image of a page into a training set, all Latin characters were named with their corresponding Unicode name 'latin.small.letter.x', or 'latin.capital.letter.x'. In general, as many characters as possible were classified according to their Unicode names.

In the module for page segmentation and in translation of classified glyphs into machine-readable format, any Unicode name can be translated into its Unicode character via the Python built-in Unicode database⁸. However, some of the characters used in the theses are not in Unicode, i.e. they needed to be named in an alternative way. For instance, this involves all italic characters and some of the Latin ligatures used:

ct ss fi fl

Characters such as these were given names such as latin.small.ligature.ct or latin.small.ligature.long.ss. The approach to extend the Unicode character names with alternative names, was necessary in accounting for the characters used in the theses. However, this implies that the translation function will have to account *explicitly* for the set of characters that are not in Unicode (how this has been dealt with is explained in section 3). Some of the ligatures used in the theses, however, are in Unicode and can be named as any other character⁹. The advantage is that these characters can be shown in web browsers, but may be problematic if the text is to be searched for.

Training sets can also be merged. If they are created for each language used in the theses, they can be combined based on the languages used in the text in the image, before it is OCR-processed.

2.1 Using classifier data

The performance of the training data can be tested directly in the Gamera GUI, or in the Gamera Python console. It loads into one of Gamera's classifiers:

```
from gamera import knn
myclassifier = knn.kNNInteractive('', 'all', 1)
trainingData= r'C:\Python23\Lib\site-packages\Savings\testLDB.xml'
myclassifier.from_xml_filename(trainingData)
myImage = load_image(r'C:\Python23\Lib\site-packages\Images\image_new.tiff')
myGlyphs = myImage.cc_analysis()
# Classify glyphs in image
myResult = classifier.group_list_automatic(myGlyphs)
```

⁷ See appendix D for information about the theses used in this project.

⁸ The Gamera system is Python based. For more information about Python, see www.python.org.

⁹ See <http://www.unicode.org/charts/PDF/UFB00.pdf>

The result is a list of glyphs, i.e. small binary images. Each glyph will have a set of id-elements, with attributes such as <state> which defines how the glyph was classified (unclassified, automatic, manual etc.). The glyph will also have attributes that can be used to identify it differently from other glyphs in the entire image object: uly, ulx, nrows and ncols. These attributes will be used in reconstruction of the text (in the page segmentation module); using for instance `listofGlyphs[i].ul_x`. Below is an example of how a glyph is represented in the xml-file:

```
<glyphs>
  <glyph uly="148" ulx="514" nrows="14" ncols="11">
    <ids state="MANUAL">
      <id name="latin.small.letter.n" confidence="1.000000"/>
    </ids>
    <data>
      5 4 3 9 1 10 1 4 2 4 1 4 4 3 1 3 4 7 4 6 5 3 1 2 5 6 5 2 1 3 5 6 4 3
      1 4 3 4 1 2 3 4 1 0
    </data>
    <features scaling="1.0">
      <feature name="area">
        154.0
      </feature>
    </features>
  </glyph>
</glyphs>
```

2.2 Broken and touching characters

To be able to deal with broken or touching characters in the image, Gamera provides specific algorithms for the splitting or grouping of characters. In creation of training data, broken characters were grouped (by drawing a bounding box around the fragmented parts) and classified with the specified prefix_group.x.x.x. Touching characters present in the image, were split using the prefix split_splitx.

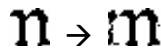


Figure 2: Grouping broken characters.

The grouping algorithm was also used for (for example) all 'i':s in the text; as such characters are always recognised as two separate components in the cc-analysis.

As regards the touching characters, some were such that they could not be split in any representative way. Most of these characters are a result from the standard used in setting; i.e. they are ligatures.



Figure 3: Splitting above character unit would require many split-operations recursively, as well as grouping operations.

Characters such as these were classified as units (and given alternative names, as they are not in Unicode).

2.3 Optimisation of training data

Gamera provides means for optimisation of training data. In optimisation, the relative importance of each of the features used for classification can be determined¹⁰. In this project, an attempt was made at optimising the created training data, using the Gamera GUI function for this. Though, much more effort needs to be put into exploring the ways of how to optimise the training data for the theses. This includes finding out which features should be used in optimisation of the particular training sets, by creating differently optimised data and comparing them.

¹⁰ For information about how Gamera classifier data can be optimised, see http://dkc.mse.jhu.edu/gamera/html/training_tutorial.html

3. Page segmentation

3.1 Identification of text in images

There are numerous algorithms or systems for automatic recognition of text in images. In applying a process of page segmentation; sections and sub-sections of text are recognised, as well as graphics or images within the image. Sections are further cut into lines, words, characters etc. Algorithms for page segmentation can be designed for general purposes, using for instance simple heuristics on text strings:

- Characters that have similar height constitute a line,
- Space is larger between words than between characters,
- Fonts do not usually change within a word etc.

More sophisticated optical character recognition software usually applies some sort of document page analysis before the OCR process is performed. This is so that the OCR system does not have to read non-text items; such as graphics or images. Some image text analysis methods uses top-down approaches, in which the page is cut into smaller and smaller blocks such as columns or paragraphs [Victor Wu et. al. 1997]. Other approaches involve bottom-up methods, in which the smallest components (pixels as connected components) are identified first and successively combined into larger components, resulting in that all blocks are found on the page. Currently, Gamera is being developed to include such algorithms.

3.2 A script for recognition of words

As mentioned earlier, a script¹¹ for recognition of words was used for output of the classified glyphs in the beginning of the project. The script combines Gamera classified glyphs into groups (i.e. words). In testing the script, a small set of classifier training data was created. A part of a page of one of the theses was used as input to training data, and also used in testing for output. All commas, full stops and noise were removed (manually) from the image, to facilitate the finding of lines and words in the text. This was done since full stops often are located exactly between two words in the theses. The image and resulting list of words can be seen below:

**tera aliis debita longo post se relinquat intervallo id
quod ex mensura obligationis abunde patefcit quæ
ceteris heic tanto fortior est quo majora et plura ad
funt motiva utpote utilitas publica privata Genc
ris humani & denique infantis quæ in relatione ad**

```
...  
ris (31, 189, 77, 221)  
ceteris (33, 101, 144, 134)  
rera (35, 22, 103, 46)  
quod (37, 54, 124, 101)  
unt (46, 151, 104, 177)
```

¹¹ The author of this script is Lukas Cenovsky. The code was shared on the Gamera mailing list in March 2004.

```

humani      (91, 187, 227, 220)
ahis        (123, 11, 191, 44)
motiva      (123, 144, 247, 176)
ex          (146, 65, 187, 87)
...

```

In the script, the words are sorted in leftmost order. This can be seen in the output; the word ‘ris’ is the leftmost word in the image, hence it is written first etc. The values within the parenthesis next to the words correspond to co-ordinates for the word boundaries; i.e. the value of:

1. The leftmost point of the word in the image
2. The topmost point of the word etc.
3. The rightmost point
4. The bottommost point

As can be seen from the list of words above, the ‘topmost’- values of the words which pertain to a certain line, are close. For example, for the first line in the image, the topmost values of these words are: 22, 11, 9, 10, 10, 9, 10, 9. The values differ within the interval of 0 – 15 (roughly). As for the second line of words in the image, the topmost -values are 54, 65, 54, 52, 64 etc. This information was used in an attempt to sort the words in a left-to-right and top-to-bottom order. If it had succeeded, the words would have been printed out in lines. This script also made use of a Python dictionary for translation of classified glyphs into ASCII format:

```

...
for i in list('ABCDEFGHIJKLMNOPQRSTUVWXYZ'):
    self._symbols['latin.capital.letter.%s' % i.lower()] = i
...

```

3.3 Modifications on the page segmentation module

The module for page segmentation was shared on the mailing list after the script for recognition of words. The work with this module was the second attempt to output the glyphs in a machine-readable format. The module uses an approach of the bottom-up method described earlier. In identifying the text, rectangular bounding boxes are drawn on each glyph and further for each line and section etc. This is done using the attributes (ul_x, ul_y, nrows, ncols) of the glyphs. Boundaries which intersect are merged, which results in larger and larger parts of the text being identified. Abnormally large glyphs (such as graphics) are also removed from the image.

In reconstruction of the text; the glyphs are sorted left-to-right and top-to-bottom, using the attributes ul_x and ul_y. In this project, the modifications on the page segmentation module concern mainly the function for translating the glyphs into a machine-readable format (Unicode and non-Unicode characters), but also on the finding of lines and combination of strings, i.e. of character-, word-, line-, and section-objects.

3.2.1 Translation into ASCII and Unicode

A new class (ocr) was created and added to the page segmentation module for the translation of the classified glyphs into a machine-readable format. The class initiates a Python dictionary, in the same way as in the script for recognition of words. The dictionary is used to translate the id-

names of glyphs that are not in Unicode. For instance, the dictionary will translate all Latin italic characters into ASCII format:

```
class ocr:
    def __init__(self):
        self._dictionary = {}
        for i in list('ABCDEFGHIJKLMNOPQRSTUVWXYZ'):
            self._dictionary['italic.latin.capital.letter.%s' |
                % i.lower()] = i
        for i in list('abcdefghijklmnopqrstuvwxyz'):
            self._dictionary['italic.latin.small.letter.%s' % i] = i
            self._dictionary['latin.small.ligature.si'] = 'si'
            self._dictionary['latin.small.ligature.ct'] = 'ct'
            self._dictionary['latin.small.ligature.long.ss'] = 'ss'
```

If the italic characters were to be represented in the resulting text file, they need to be encoded using markup. One way of dealing with this would be to have the ASCII characters in the dictionary directly enclosed by tags (e.g. <italic>).

```
self._dictionary['italic.latin.capital.letter.%s' |
    % i.lower()] = "<i> %s </i>" % i
```

This would also mean that each italic character in the resulting text would be enclosed by the tag; the text would be difficult to read by a human but it would not constitute a problem for web browsers. Currently, the italic characters are translated into ASCII characters only.

The original method `name_lookup_unicode()` (which is implemented as the default translation function in the page segmentation module), was used to translate classified characters that are in Unicode. The `id_name` of each classified glyph is looked up in the Python built-in Unicode database¹² and the corresponding Unicode character is returned. This function was modified to deal with a problem of naming hyphens in the text. As the id-name "hyphen-minus" becomes "hyphen_minus" in the xml-files, these glyphs needed to be translated explicitly.

```
def name_lookup_unicode(self, id_name):
    if (id_name == "hyphen"):
        return unicodedata.lookup(r"HYPHEN-MINUS")
    else:
        name = id_name.replace(".", " ")
        name = name.upper()
        try:
            return unicodedata.lookup(name)
        except KeyError:
            print "ERROR: Name not found:", id_name
            return ""
```

¹² For more information about python Unicode support, see <http://www.python.org/doc/current/lib/module-unicodedata.html>

With regard to the original function for translation of glyphs and combining them into strings (`make_string()`), the following changes or complements were done:

- The constant which is used to indicate spaces between words was changed to 1.4 instead of 2. The line `s = s + " "` puts a space in between words if a larger space between glyphs is found (i.e. larger than average space times the constant). In testing for the correct value for the constant, having it set to 1.3 would break words in the output; 1.5 would glue words together etc.

```
def make_string(self, lines)
...
if (glyphs[i].ul_x - glyphs[i - 1].lr_x) > (average_space * 1.4):
    s = s + " "
...
```

- The two ways of translating the characters needed to be accounted for; the dictionary and the `name_lookup_unicode` function. Each id-name of glyph is first matched against the dictionary and otherwise looked up in the Python built-in Unicode database.

```
# Below: if glyph [i] is in the dictionary; translate from there,
# otherwise use the name_lookup_unicode -function.

if (glyphs[i].get_main_id() in self._dictionary):
    s = s + self._dictionary[glyphs[i].get_main_id()]
else:
    s = s + self.name_lookup_unicode(glyphs[i].get_main_id())
...
# Below: break into lines
    s = s + '\r\n'
return s
```

Other modifications on the page segmentation module concern:

- The function `find_lines()`. The part where intersecting lines are merged was removed. This was done since lines will overlap in the images of the theses. It resulted in that no lines were identified in the image, hence no output.
- The function `ocr ()` was modified to include the grouping algorithm. Parameters are a classifier, image and segmented glyphs (to be classified). The final line initiates the page segmentation.

```
def ocr(self, image, classifier, glyphs):
#Classify and use the grouping algorithm
result = classifier.group_list_automatic(glyphs)
glyphs = [x for x in glyphs + result[0] \
if x.classification_state ]
#Initiate page segmentation
page = Page(image, glyphs)
```

```
page.segment ()  
return page
```

(An overview on the modified parts of the page segmentation module is included in appendix B.)

4. Creation of a script for the OCR –process and for a user interface

A script needed to be created for control of the OCR-process and for a user interface. The script uses the Gamera classifying system and the modified page segmentation module. Via the interface, a training set and image can be imported. When the ‘run OCR’ button is activated, the script imports the Gamera classifier modules, and runs the page segmentation module with the selected image and training set as input. The final parts of the script uses the `make_string()` function in the page segmentation. After the string (of language objects) has been created, it is written to a file. It is desirable to have the script inside the Gamera GUI, but it is currently separate. (See appendix C for an overview on the script.)

```
# Author Frida Sandgren
...
# On 'Run OCR'
def OnOpenOcr(self, event):
    init_gamera()
    from gamera import roman_2
    from gamera import knn
    import codecs

# Create an instance of Gamera Knn-Interactive Classifier
    classifier = knn.kNNInteractive('', 'all', 1)
    global LDB

# Load the Training set into the classifier
    classifier.from_xml_filename(LDB)
    global image
    Image = load_image(image)

# Segment the image into its Connected Components
    glyphs = Image.cc_analysis()

# Import the text segmentation module and create an instance of the ocr-class
    ocr = roman_2.ocr()

# Use the method ocr which initiates the text segmentation. Pass the image (to be
reconstructed), the classifier with training set and segmented glyphs (to be classified).
    page = ocr.ocr(Image, classifier, glyphs)

# Open a file to write
    try:
        f = codecs.open(r"G:\Documents and Settings\Frida Sandgren\Desktop\text.txt",
"w", "utf8")

        for section in page.sections:
# For each identified line in each identified section, make string and write to file
            str = ocr.make_string(section.lines)
            print "WRITE FILE"
            f.write(str)
            f.flush()
            f.close()

    except:
        print "NO OUT"
    self.Destroy()
```

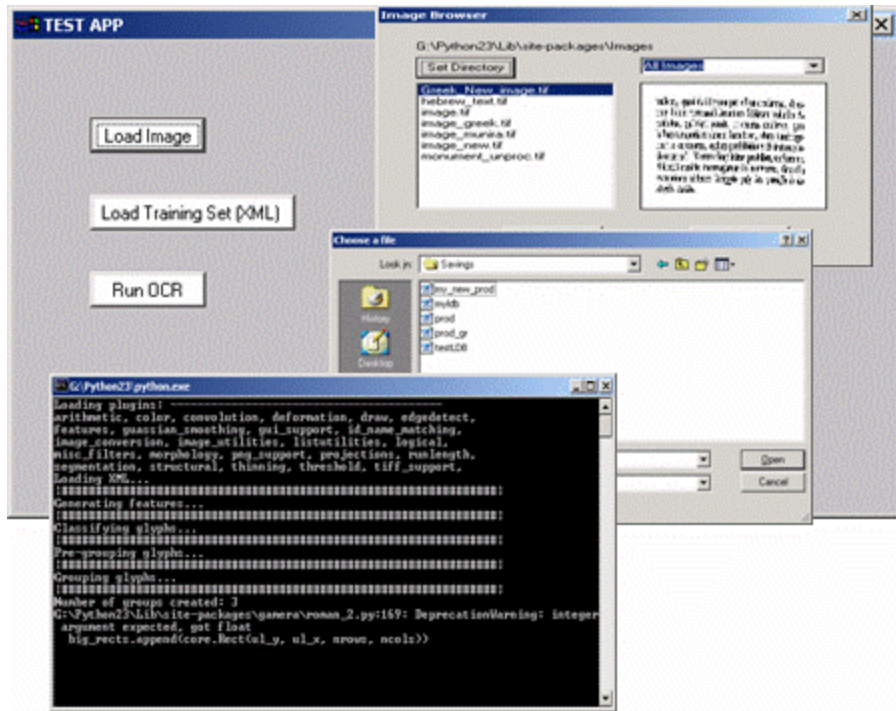


Figure 4: Interface for OCR-process. The image has been imported together with a selected training set. The window at the bottom shows the status of the script.

5. Recognition accuracy

In a small test for recognition accuracy, one thesis was selected both for training and for test. Five pages of the thesis were used in creation of a training set. The test was then performed on two different pages¹³. As the thesis consist of 17 pages all together, it was decided that five pages of training data would be sufficient. (The size of a Gamera training set is recommended to be approximately 50 pages. Section 6 further discusses the work and time needed to create representative training sets). The two pages used in the test contain Latin characters only.

Each image was imported together with the training set, via the script for OCR-process (described in the previous section). As the page segmentation is run via the script, each image is OCR-processed. The glyphs are translated into their corresponding Unicode characters and finally written to a text file. One of the images and the resulting text can be seen below. (The second image and its corresponding text are included in appendix A. This image was also used in a test with the OCR-software Omnipage Pro X (described below in section 5.2).

I. N. J.

§. I.

On alia de caussa illam hom.
intrinsecam agendi omitten-
dique facultatem certis legi-
bus circumscriptam Divina
voluit bonitas, quam ut actio-
nes suas homo, in vaga hac
circa moralia libertate, honeste
instituire possit, atque ad certum & naturæ suæ
convenientem finem dirigere, Est vero ea ple-
rumque mortalium indoles, ut, quemadmodum
nonnulla ad quævis bona erecta ingenia amore vir-
tutis & sola legis reverentia excitantur, sic magna
proh

On al,.u de caussa i.llam huul
..,ltr,.nfecam ageuhi. oml.tten-
d,.quo facultatem cert.. s legl,-.

¹³ Thesis [A]. Pages 3, 4, 8, 10 and 11 were used in creation of the training set. Page 7 and 9 were used in the test.


```

bus c , .rc ilmfer.-ptam El.v .-na
volui.t bonitas , q tlam ut act..ft.
nes fuas hnmo , l.n vagu hao
-. -
c, rca fl oral.a libfrtate, b on efto
i.nfti.tucre possit , arqxe ad certtim & naturæ fxa
oonftn,.entem fineui di-r..gere. Eft vero ea pls-
rumque mortali.um l.ndolea , ut, que.uiadmodum
no,inulla ad qxnvi.s bonn erects ingen,.a amore vi.r-
futis & fola legis rovcre. nti.a exci.tantur , fic msgnn
p r oh

```

5.1 Results

As can be seen from the resulting text (above), a lot of excessive hyphens and full stops are present in the text. This is due to the high occurrence of noise in the image. Many characters have also been classified incorrectly. This is due to the fact that it is difficult to account for ‘unknown’ characters for the theses. Characters may differ dramatically (and for *each* occurrence), in the way that they are broken or set.

 = **V** Two occurrences of the character ‘latin.small.letter.v’ in an image.

As such, even though the training set contains a class of many examples of a character, such a character can still be incorrectly identified. If there was a pattern on how the characters were broken, this would have been easier to solve. Worth noticing is that the thesis used in the test was selected because of its relatively good quality of paper, in comparison with other theses.

The table below shows the values for recognition accuracy.

Table 1.

	Image I	ImageII	Total
Number of correctly identified characters	302	893	1195
Total number of characters in image	370	1080	1450
Recall %	81.6	82.7	82.4
Total number of characters in text file	416	1228	1644
Precision¹⁴ %	72.59	72.71	72.68

5.2 Other OCR-software

In an OCR- test with Omnipage Pro X, the second image was used. This test showed that Omnipage is unable to deal with the Latin characters that are in presentation forms (e.g ligatures such as ‘st’) as well as all Latin long ‘s’-characters. These characters are in Unicode and could be represented in the text. Even though Omnipage produces a fairly good result on this particular

¹⁴ I.e. the number of symbols correctly recognised on a page normalized by the number of symbols in the OCR-generated text. (www.umiacs.umd.edu/lamp/pubs/TechReports/LAMP_030/LAMP_030.pdf)

image, the software would not be able to account for text written in more than one language, which is common for the theses. For images which contain text in both Latin and Greek, the Greek characters would be identified as Latin by Omnipage. The image and resulting text are included in appendix A. (It should be noted that a Gamera created training set may include any character – it is language independent).

6. Discussion

How characters are identified and translated is entirely dependent on how the training sets are created. The identification of characters has to be *created*, using Gamera as a tool. This proved to be difficult in terms of *representation*, as well as very time expensive.

6.1 Creation of representative training sets

To create a general training set for the entire collection of theses would not be possible, considering the varying use of characters, character sizes and paper quality. A larger-scaled OCR-project for the theses would require:

- Categorisation of theses that uses the same character fonts etc. into groups.
- Create training sets for each group.
- Each training set need to be optimised using the Gamera algorithm for optimisation.

Including one image of a page into a training set involves manual classification of all occurring characters in the image. In order to be able to account for the various (language-specific or presentational) characters occurring in the theses, the Unicode look-up function, which translates many of the characters into a machine-readable format, is without doubt the most appropriate solution for translation. If the character can be found in the Unicode code charts, it can be named directly and no changes need to be made to the page segmentation module.

However, characters that are not in Unicode need to be added to the dictionary in the page segmentation module, whenever they are found and included into the training set. The combination of Unicode characters is also not accounted for in the Python built-in Unicode database. This is desirable as, for instance, many of the Greek characters can be defined as combinations of Unicode characters.

6.2. Translation

There were also some difficulties in classification and translation of the Greek characters in some of the theses. This applies to the theses which use an extended alphabet for Greek (for instance, this is true for the thesis used in the test for recognition accuracy). The procedure were to find out for each character if it is in Unicode or not and if not, adding a new name to the dictionary and determine which ASCII character should represent the character. These Greek characters often carries ‘tonos’ or ‘perispomeni’, which means that the grouping algorithm must be applied for each such character in the training phase. This also affects the time of classification as the grouping algorithm slows down the process.

In some cases, the OCR process on some of the images with Greek characters, would take more than 10 minutes per image. Below are examples of Greek characters that need to be grouped. They are also difficult to translate into Unicode or ASCII:



Character ‘greek.capital.letter.omega.with.psili.and.perispomeni’



Character ‘latin.letter.small.capital.ou.combining.greek.perispomeni’



Character 'greek.theta.symbol.combining.latin.small.letter.v.with.hook'.

6.3 Character sizes & isomorphic glyphs

There are also some difficulties concerning the recognition of different character sizes used in the theses. Problems arise when training characters such as 'o' or 's', if more than one size is used. (In fact, there can be more than five different sizes of characters in the theses.) For instance, a small letter 'o' used in a large size can be equal to a capital 'o' in a small size (i.e. the same character was used to represent both in the setting). Thus, characters such as these need to be named with a common name 's' and not 'capital s' or 'small s'. There are no means for dealing with this in the page segmentation.

In creating the training set for the thesis used in the test, there were at least five different character sizes present in the text. The Unicode names only support two sizes (in the sense of semantics): small or capital. Another issue is that of isomorphic glyphs. For instance, the Greek character 'capital alpha' is identical to the Latin character 'capital a', but is a different character. Such isomorphic glyphs need to be trained under the same name.

6.4 Prerequisites

The development of the Gamera software is ongoing; which means that some functions have been modified during the time of this project. As an example, there has been a bug in the grouping algorithm, which 'contaminated' the created training sets. This resulted in that no 'i'-characters were classified and written to the text file. Also, there has been a bug in the Gamera GUI optimising function. This resulted in that the training data were loaded and the optimisation algorithm initiated, but when the optimisation was finished, the optimised data could not be saved. The time of running the optimisation algorithm for each training set is recommended to be for a day or two. At the time of writing though, these bugs have been solved.

7. Future work

Issues such as how to create representative training data and optimisation of training data need to be solved before proceeding with Gamera and the OCR-application. Future development of the application would concern, for instance, means for dealing with images in the text, isomorphic glyphs, and varying character sizes.

7.1 Pre-processing

There is a need for a pre-processing module that recognises and removes more noise from the images, before they are loaded into the classifier. In addition, different character sizes in the images also need to be recognised, before classification. Such a pre-processing module would identify regions of text (in contrast to graphics) and regions of text in different sizes etc. [A. Antonacopoulos et al. 2003]. The approach would be to create sub-images of the image of current interest:

1. An image of noise and graphics
2. An image containing text

The image of text can further be divided into:

1. An image containing normal /small text,
2. An image of medium text,
3. An image containing large text, etc.

These images would then be OCR-processed separately using different training sets (i.e. sets for sizes). The fact is that most pages in the theses contain characters of different sizes, graphics and larger (non-text) symbols. As regards OCR-software intended for recent, typewritten documents, there is usually a pre-processing module which performs these issues.

7.2 Noise

In some images, there can be more than 600 noise-components, of which the largest usually are classified as hyphens or full stops. This is true even though the image has been scanned in a way as to remove as much noise as possible. In the beginning of this project, a small test was conducted in order to see if results are improved if the training set contains examples of noise. (This implies that noise can get recognised as any other character.) In this case, all glyphs classified as noise were removed before processed by the text segmentation; by use of a regular expression:

```
import re
noise = re.compile('^noise.noise')
result= classifier.group_list_automatic(glyphs)
    glyphs = [x for x in glyphs + result[0] \
if x.classification_state \
    if (noise.match(x.get_main_id()) == None)
    page = Page(image, glyphs)
    page.segment()
return page
```

This solution proved to be somewhat problematic, as it seemed to affect the broken characters as well. A better solution would be to account for this in a pre-processing module (explained above). Gamera provides many devices for the processing of images.

8. Concluding remarks

In this project, the goal has been to enable for the OCR-process on the particular theses, using the Gamera software. Most important was to make for the classified characters to be written to text files, in order to see results. The Gamera software was chosen as it makes it possible to create a customised solution, as well as providing algorithms for dealing with heritage documents. There are many parts of Gamera that have not been used in this project, for instance, means for the creation of plug-ins, toolkits etc. With regard to the specific characters used in the theses (and to the different languages used in the texts), a customised solution was desirable. Gamera suits this purpose well. In the future, if issues such as how to deal with noise and optimisation of data are solved, Gamera could be used in a more large-scaled project for the theses.

References

[1] Gamera homepage: <http://dkc.mse.jhu.edu/gamera>

[2] Gamera documentation: <http://dkc.mse.jhu.edu/gamera/html/index.html>

[3] Gamera documentation (xml_format): http://dkc.mse.jhu.edu/gamera/html/xml_format.html

Fujinaga, I. 1996 *Exemplar-based learning in adaptive optical music recognition system*. Proceedings of the International Computer Music Conference. 55-6.

Victor Wu, R. Manmatha, Edward M. Riseman. *Finding Text in Images*. 1997. Multimedia Indexing and Retrieval Group, Computer Science Department, University of Massachusetts, Amherst. International Conference on Digital Libraries. Proceedings of the second ACM international conference on Digital Libraries. ISBN: 0-89791-868-1. ACM Press, New York, USA.

A. Antonacopoulos, B. Gatos, D.Karatzas: *ICDAR 2003 Page Segmentation Competition* Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICIDAR 2003): http://www.csc.liv.ac.uk/~prima/ICDAR2003/Papers/0125_800_antonacopoulos_a.pdf

Östlund, Krister. *Johan Ihre on the origins and history of the runes: Three Latin dissertations from the mid 18th century*. 2000. Studia Latina Upsaliensia, ISSN 0562-2859 ; 25. ISBN: 91-554-4642-6. (<http://publications.uu.se/abstract.xsql?dbid=422>).

Örneholm, Urban. Östlund, Krister. *Avhandlingsspråk vid Uppsala Universitet 1600 -1855*. Årsbok för idé- och lärdomshistoria. 2000. s. 180 – 182.

Appendix A

Below: Figure 1 shows the second image used in the test for recognition accuracy. This image was also OCR-processed by Omnipage Pro X. Figure 2 shows the resulting text, using the created training set. Figure 3 shows the resulting text from the test with Omnipage.

Figure 1.

10 DE PROPORZIONE INTER

riæ sunt agentium conditiones & status, actionum fines diversi, atque objectorum qualitates dispariles. Multiplices præterea agendi modi, temporum, locorum & aliarum genera conditionum, quæ dum ipsis actionibus superveniunt, diversam consensum in peccata quantitatem produunt, atque hinc eadem variis formis induunt. Etsi enim vel una circumstantia delinquendi voluntatem arguat, adeoque ad actionem malam sufficiat, non tamen dubium est, quin quo plures adgravantes circumstantiæ concurrant, eo vehementior fuit agentis ad peccandum nisus, & quo pronior nisus, eo etiam longius distat actio a lege, eo etiam plus turpitudinis & deformitatis in se continet. Quod cum ita sit, adparet fundamento suo non destitui, quod Moralium scriptores peccatum, etiam quoad formale consideratum, cum curvitate lineæ communiter comparent, & ut variâ curvæ lineæ a recta est deflexio, sic variam quoque peccati a lege, tanquam norma, distantiam esse doceant.

Si ad pœnas respicere velimus, illas inter multum quoque, pro diversitate peccatorum, deprehensuri sumus discriminis. Non enim quasi per saltum, nisi in extremo necessitatis gradu, ad ultimum supplicium jure pervenitur, sed prout majora vel minora sunt peccata, vindictam principum graviolem leviolemve subeunt. Nonnullis, qui extreme mali sunt, & quorum emendatio desperata, quod unum superest remedium, mors repræsentatur: Aliis, in quibus emendationis spes est, mul-

Figure 2.

In DE PROPORZIONE INTER
riæ sunt agentium conditiones & status, act.,.contim.

fines diverf. , atque objectortlm q.uslitstfs difpsu-
 lea- Multiplices præterea ageudi modi ,. .tcmo.
 rum . loenrum & alisrunl genera. conditiontln ,
 quæ d um i psils act ioiii.bue ft., pectel., u n.t , diverfcm
 oonfcnfus i.n peccsta quant,tstem pradunt , . atque
 hiae eadem variis formis induunt. Etfl en,m vel
 nna eireumftantia del.,nquendi vol,intatem arguat,
 udeoqne ad actioncm nlsiam f,,fficist , non tsmen
 dub,. unl cft , quin qtio plures adg.ravantee eire.unl.
 ftantiaæ &n.currant . eo vehement,or fuit agenns. ad
 pI o e nc ge a, . u. l s d ud m. . t l a nt i fauctl . . , o &a l qe gq
 ao , p er oo n. e , . t o j sr mn i fpu l su - s et ot l r e pt i , . st t
 mi -
 di.n,.s & deformitatis in fe. cuntlnet, Quod eum ita
 ft , adparet ftlildamento fuo non .defl.,ttii , tltiod
 Moral,.um feriptores peccatum , et..sm quoad for.-
 male conf.deratum, cum curvitlte l,neæ commrin,-
 fer comparent, & ut vsr,-s curvæ lineæ a rcffs eft
 deflcuto , fic variam quoque peecsti a li-ge . tsn-
 quam norma , diftantiam esse dmeant. .
 Si ad pænas rrfp.ieere velimm , illas inte.r mul,
 fum quoqxe , pro d,.verfltata peccato.rum , depre-
 henftlirt. fumus difcriminis. Non enim quafi per
 faltum , nif, in cttremo necetsitatis gradu , ad ulti-
 mum ftlpplicium l.ure . pervenitur , fed pr.otlt. ma,
 jura vel m,.aora funt pec.catl , vindictam pr,nc,piim.
 graviolem lev..orenlce fubeunt. Nonnullis , qul
 eatreme mali funt , & quorum emeodstio dcifpera-
 ta , quod unum fspereft rmedium , mors repræ-
 fentatrlr .. Aliis , m. quibrle cillendationis fpes eft ,
 R ul,

Figure 3.

ljt`PROPORTIONE INTER

riz ttht agentium conditiones & Ilatus, aaionum , ' fines
 diverfi., atque objeaoruni qualftates difpun ç Les. Multiplices
 prxterea~agendi',im'(?di, tempow rum, locoruni & aliarum
 genera conditionum qu dum iplis aaionibus fupePvenutit,
 diverfam , 4 confenlūs in petcata quantitatem
 produnt,. atque hinc eadem vari is formis,
 induunt. Etii enim vd una circumflantia delinquendi voluntatem
 arguar, adeoque ad. aaionem malam fuffiMt, non tamen dubium eft,
 gum quo plures adgravante6 circurn flantix, coricurrant, co
 vehementior fdit agenrisad .l peccandum ni fas, & qtjo
 pronior nifus, co tjam Iongius diffat aaio a lege, co etjani
 plus turpitu. dinis & deformitads in fe continet Quo d cum ka

it, adparet fundamento-, fluo non defâtm, quod Mq'raflum
fcriptores peccaturn, etam quoad for. male confderatum, cum
curvitate lin,ex. communi. ter comparent, & ut vana curv
line a reaa en defle*io, fic variam quöque
peccati a lege, tan quam norm a, diftariam efle
doceant.

Si ad pccilas refpjcere velimus, illas inter mule tum
quoque, pro diverfitatë peccatorum, depre' henfuri-
Aumus dikrim.inis. Non, enim quafi- per faltum, nW in
xtreernd neceffitatis gradu, ad ultimum fuppliciurn jute
pervenitur, fed protit- ma
joravelminorafunt,peceata5--vindiâztm~ principum
graviorem levioerenive flâb eunt. Non nullis, gul extreme
Mali flânt, & quorum emendatio defpera ta, quod unum
ftipereff remedium, mors repr fentatur: Ahis~ in quibus,
emendationis fpes-,eft,, muli,

Appendix B

The modified parts of the page segmentation module are shown below. The page segmentation module (`roman_text.py`) can be found in the Gamera installed directory.

Class ocr— a new class was added to initiate a dictionary. It is used to translate symbols that cannot be found in the Python Unicode database. The class includes the modified methods: `name_lookup_unicode()`, `write_string()` and `ocr()`. The last method (`ocr()`) creates an instance of the top class (`Page`). The `write_string()` -method is called from the script for OCR. It uses the dictionary and the method `name_lookup_unicode()`.

```
class ocr:
    def __init__(self):
        self._dictionary = {}
        for i in list('ABCDEFGHIJKLMNOPQRSTUVWXYZ'):
            self._dictionary['italic.latin.capital.letter.%s' % i.lower()] = i
        for i in list('abcdefghijklmnopqrstuvwxyz'):
            self._dictionary['italic.latin.small.letter.%s' % i] = i
        self._dictionary['latin.small.ligature.si'] = 'si'
        self._dictionary['latin.small.ligature.ct'] = 'ct'
        self._dictionary['latin.small.ligature.long.ss'] = 'ss'
```

Method `name_lookup_unicode()`:

The name (`id_name`) of glyph is looked up in the built-in Python Unicode database. The method returns the Unicode character. As "hyphen-minus" becomes "hyphen_minus" in the xml -file, those glyphs need to be translated explicitly.

```
def name_lookup_unicode(self, id_name):
    if (id_name == "hyphen"):
        return unicodedata.lookup(r"HYPHEN-MINUS")
    else:
        name = id_name.replace(".", " ")
        name = name.upper()
    try:
        return unicodedata.lookup(name) #correction
    except KeyError:
        print "ERROR: Name not found:", id_name
        return ""
```

Method `make_string()`: modified parts concern the constant for spacing and a way to use both translation methods.

```
def make_string(self, lines):
    s = ""
    for line in lines:
        glyphs = line.glyphs
        total_space = 0
        for i in range(len(glyphs) - 1):
            total_space += glyphs[i + 1].ul_x - glyphs[i].lr_x
        average_space = total_space / len(glyphs)
        for i in range(len(glyphs)):
```

```
if i > 0:
```

Below: `s = s+ " "` puts a space in between words if a larger space between glyphs is found. Having the constant set to 1.4 seems to be optimal; using 1.3 breaks words in the output; 1.5 'glues' words etc.

```
if (glyphs[i].ul_x - glyphs[i - 1].lr_x) > (average_space * 1.4):  
s = s + " "
```

Below: if the glyph is in the dictionary; translate from there, otherwise use the `name_lookup_unicode` function.

```
if (glyphs[i].get_main_id() in self._dictionary):  
    s = s + self._dictionary[glyphs[i].get_main_id()]  
else:  
    s = s + self.name_lookup_unicode(glyphs[i].get_main_id())  
#s = s + name_lookup_old(glyphs[i].get_main_id())  
#self._newline = s  
#self._output.append(self._newline)  
#self._newline = ""
```

Break into lines in writing to file

```
s = s + '\r\n'  
#return self._output  
return s
```

Method `ocr()`:

The regular expressions (import `re`) is for dealing with contaminated databases; because of a bug in the grouping (and splitting) algorithm. The `re` for noise is for a test in dealing with noise. The test was on two training sets; one with added examples of noise and one without. The `re` for noise then removed glyphs classified as noise, This, however, did not work so well; some fragments (belonging to broken characters) were classified as noise and removed. It did some good things though. It cleared the output, which was shown in less excessive stops or hyphens (which the noise otherwise would get classified as).

```
def ocr(self, image, classifier, glyphs):  
    import re  
    p = re.compile('^_group.[\S, .]+')  
    r = re.compile('^_split.split[\S, .]+')  
    #noise = re.compile('^noise.noise')  
    result = classifier.group_list_automatic(glyphs)  
    glyphs = [x for x in glyphs + result[0] \  
              if x.classification_state \  
              if (p.match(x.get_main_id()) == None) \  
              if (r.match(x.get_main_id()) == None) \  
              #if (noise.match(x.get_main_id()) == None)]  
    page = Page(image, glyphs)  
    page.segment()  
    return page
```

Appendix C

Below is the script for control of the OCR-process (described in section 4). It also provides a user interface.

''' Author: Frida Sandgren

This script uses a modified version of the Gamera module for page segmentation with the Gamera classifier system. Please see: <http://dkc.mse.jhu.edu/gamera/> for information about the Gamera software. The original module for page segmentation (roman_text.py) is included with the Gamera software. This script is intended to be inside the Gamera GUI, but is separate for now.'''

```
import os
import sys
from wxPython.wx import *
from gamera.core import *
from wxPython.lib.imagebrowser import *

global LDB
global image

ID_OPEN_IMAGE =100
ID_OPEN_FILE = 102
ID_OCR = 105

class MainWindow(wxFrame):
    def __init__(self,parent,id,title):
        wxFrame.__init__(self,parent,wxID_ANY, title,| style=wxDEFAULT_FRAME_STYLE|
wxNO_FULL_REPAINT_ON_RESIZE)

        panel = wxPanel(self, -1)
        panel.SetBackgroundColour("LIGHT GREY")
        button1 = wxButton(panel, ID_OPEN_IMAGE, "Load Image")
        button2 = wxButton(panel, ID_OPEN_FILE, "Load Training Set (XML)")
        button3 = wxButton(panel, ID_OCR, "Run OCR")
        button1.SetPosition(wxPoint(50, 50))
        button2.SetPosition(wxPoint(50, 100))
        button3.SetPosition(wxPoint(50, 150))
        EVT_BUTTON(self, ID_OPEN_IMAGE, self.OnOpenImage)
        EVT_BUTTON(self, ID_OPEN_FILE, self.OnOpenFile)
        EVT_BUTTON(self, ID_OCR, self.OnOpenOcr)
        button1.SetBackgroundColour("WHITE")
        button2.SetBackgroundColour("WHITE")
        button3.SetBackgroundColour("WHITE")
```

```

        button1.SetForegroundColour("BLACK")
        button2.SetForegroundColour("BLACK")
        button3.SetForegroundColour("BLACK")
        EVT_CLOSE(self, self.OnCloseWindow)

    def OnCloseWindow(self, event):
        print "OnCloseWindow"
        self.Destroy()
        sys.exit(1)
        app.ExitMainLoop()

# On Load Training Set
# LDB = loaded training set
    def OnOpenFile(self,event):
        wildcard = "All xml-files (*.xml)|*.xml|"
        frame1 = wxFrame(NULL, -1, "File Browser")
        dlg = wxFileDialog(frame1, "Choose a file", os.getcwd(), "", wildcard,| wxOPEN,
wxMULTIPLE)

        if dlg.ShowModal() == wxID_OK:
            global LDB
            LDB = dlg.GetPath()
        else:
            print "xml-file error"
            dlg.Destroy()

# On Load Image
# image = imported image
    def OnOpenImage(self,event):
        dir = os.getcwd()
        initial_dir = os.path.join(dir, r'G:\Python23\Lib\sitepackages\Images')
        win = ImageDialog(self, initial_dir)
        win.Centre()
        if win.ShowModal() == wxID_OK:
            global image
            image = win.GetFile()
        else:
            print "image error"
            win.Destroy()

# On 'Run OCR'
    def OnOpenOcr(self,event):
        init_gamera()
        from gamera import roman_2
        from gamera import knn
        import codecs

# Create an instance of Gamera Knn-Interactive Classifier
        classifier = knn.kNNInteractive('', 'all', 1)
        global LDB

```

```

# Load the Training set into the classifier
    classifier.from_xml_filename(LDB)
    global image
    Image = load_image(image)
# Segment the image into its Connected Components
    glyphs = Image.cc_analysis()
# Import the page segmentation module and create an instance of the ocr-class
    ocr = roman_2.ocr()
# Use the method ocr which initiates the text segmentation.
# Pass the image (to be reconstructed), the classifier with training set
# and segmented glyphs (to be classified).
    page = ocr.ocr(Image, classifier, glyphs)
# Open a file to write
    try:
        f = codecs.open(r"G:\Documents and Settings\Frida Sandgren\Desktop\text.txt",
"w", "utf8")
        for section in page.sections:
# For each identified line in each identified section, make string and write to file
            str = ocr.make_string(section.lines)
            print "WRITE FILE"
            f.write(str)
            f.flush()
            f.close()
    except:
        print "NO OUT"
    self.Destroy()

app = wxPySimpleApp()
frame = MainWindow(None, -1, "GAMERA OCR -TEST")
frame.Show(1)
app.MainLoop()

```

Appendix D

The following theses have been used in this project:

Dissertio Politica De Proportione inter Peccata et Poenas by Johanne Ihre. 1749 [1].

Dissertatio Theologica, Munera Ministerii Ecclesiastici by D.Mag Laurentio Benzelstierna. 1751.

Monumentorum Veterum Ecclesiae Sveogothicae by Ericus Benzelius

Collisione Conscientiae et Famae by Johanne Ihre. 1757.

Gradibus Officiorum By Johannis Ihre. 1749.

Meditationes Nonnullae Generaliores De Collisione Conscientiae et Fame, by Johanne Ihre.1757.