DESIGNING A FLEXIBLE DOCUMENT IMAGE ANALYSIS SYSTEM: THE MODULES

Andrei Tigora [1*]

ABSTRACT

The project described here represents a flexible and easily extensible OCR system, designed to be configurable at runtime. The system itself is composed of standalone binary files than interact with one another using XML files. The system is composed of a back-end that is represented by binaries, external libraries and a couple of third party binaries, and a GUI front-end that allows the user to correct imperfections in the results.

KEYWORDS: *OCR, Digitization, Document Image Analysis, Document Export, Retroconversion*

1. INTRODUCTION

A modular system as the one proposed in this paper is easy to configure, adding to the processing flow those components that offer optimal results for the task. Also, if there is the need to expand the system's functionality, new components can be developed independently of the existing ones, and without requiring recompiling of the entire system, whose source code might not be available for some reason or another.

This paper is the result of the 2 year's work done for the master thesis of the author [30] and the continuation of the work presented in [32]. Other papers published during this period, which are connected to the presented subject and can add relevant information, which was omitted here for the sake of consistency, are [26] and [27]. They go into the details of preprocessing and contain detailed related work.

2. THE SYSTEM'S MODULES

Grayscale conversion modules

"iterative_recoloring" binary

Conversion from color to grayscale is a lossy process as it reduces a three-dimensional domain to a single dimension. The loss is an acceptable compromise that ensures compatibility with software that cannot handle the extra complexity associated with a

[1]* corresponding author, Engineer, Politehnica University of Bucharest, 060042 Bucharest, Romania, andrei.tigora@cti.pub.ro

three channel input. The main challenge is therefore creating an image that retains the core features of the original one, without introducing visual abnormalities.

The iterative recoloring begins by computing the luminance of the image using the classic formula for transforming RGB images. However, as a result of this transformation, the difference between neighboring pixels has dropped significantly as compared to what it was when the pixels were colored. Therefore, the algorithm attempts to recover as much of that initial difference as possible from the original image [3][28][29][31].

At each iteration, pixels influence the associated values of their neighbors, in order to achieve a difference that is as close to the one in the original color image. When all pixels have been evaluated, their influences on same value pixels are added up and averaged, and the pixels have their values changed accordingly. After a preset number of iterations, or when the image no longer changes significantly, the processing stops, producing a grayscale image.

Binarization modules

"otsu" binary

The application achieves image binarization using a global threshold value computed through Otsu's method [20]. Binarization refers to transforming a grayscale image to a black and white image.

For the grayscale image (usually represented at 8bpp) histogram of pixel values is created. Next, for each value in the histogram, the histogram is split into two sets of values and the inter-class variance is computed. The value in the histogram that generated the highest variance is chosen as threshold.

The final step consists simply of cataloguing the pixels as either foreground or background based on how they relate to the threshold.

Other binary pixel modules

"deskew" binary

This component detects and corrects the skew of a binary image. The algorithm attempts to rotate the image for a given set of angles and chooses that value for which the distribution of the projection of pixel on the vertical axes is most like that of correctly scanned text. This method is called “Projection Profiling”.

Layout modules

"layout" binary

This component is the most complex one of the entire project. Given that the relations between the components is fairly complex, a UML diagram is presented below, summarizing the interactions between the most important classes of the binary and their operations.

The functioning of this component will be exemplified on the image presented in Figure 2, with following figures outlining the structures that are being constructed as result of the processing. The image contains a paper fragment, structured into two columns, and using two distinct fonts, one for the author's own words, the other for citations.

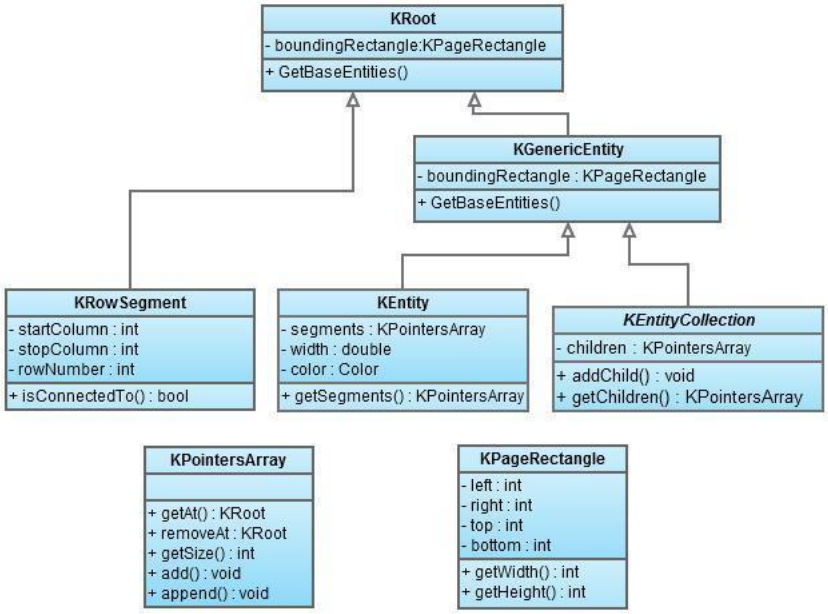
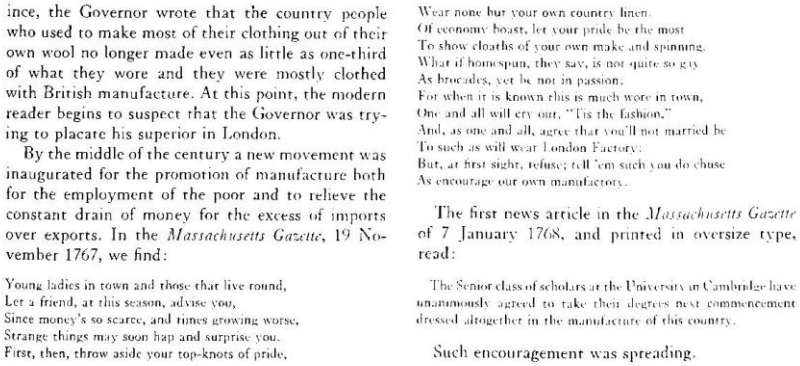


Figure 1. UML class diagram


ince, the Governor wrote that the country people who used to make most of their clothing out of their own wool no longer made even as little as one-third of what they wore and they were mostly clothed with British manufacture. At this point, the modern reader begins to suspect that the Governor was trying to placate his superior in London.

By the middle of the century a new movement was inaugurated for the promotion of manufacture both for the employment of the poor and to relieve the constant drain of money for the excess of imports over exports. In the *Massachusetts Gazette*, 19 November 1767, we find:

Young ladies in town and those that live round,
Let a friend, at this season, advise you,
Since money's so scarce, and times growing worse,
Strange things may soon hap and surprise you.
First, then, throw aside your top-knots of pride,
Wear none but your own country linen.
Of economy boast, let your pride be the most
To show cloaths of your own make and spinning.
What if homespun, they say, is not quite so gay
As brocades, yet be not in passion,
For when it is known this is much wore in town,
One and all will cry out, "'Tis the fashion."
And, as one and all, agree that you'll not married be
To such as will wear London Factory:
But, at first sight, refuse; tell 'em such you do chuse
As encourage our own manufactory.

The first news article in the *Massachusetts Gazette* of 7 January 1768, and printed in oversize type, read:

The Senior class of scholars at the University in Cambridge have unanimously agreed to take their degrees next commencement dressed altogether in the manufacture of this country.

Such encouragement was spreading.

Figure 2. Fragment of a binarized image

Entity Detection

First of all, the entities of the image are detected by grouping together foreground pixels (in our case, black) that are adjacent to other similar pixels. Such a grouping, will eventually be named "entity".

Initially, maximum length horizontal rows of black pixels are identified in the image. Such a maximum length row is called a segment. As the number of segments is pretty large, they have to be stored efficiently, using a structure that employs three parameters: row index of the segment, the index of the starting column and the one of the last column.

Previously identified segments are then grouped based on their adjacency. To limit the copy operations as much as possible, each segment is encapsulated in a node structure, which is the base component of a linked list. Apart from storing a reference to the actual segment, this node also stores a pointer to the head of the list (the head references itself), a pointer to the next node in the list and the number of elements in the list.

Determining segment adjacency is done by checking all nodes in the list in the order of the rows. This does not require additional reordering of the segments, as they are already ordered due to the way they were formed, and the node construction can closely follow segment creation.

For each node encapsulating a segment on row k, one has to determine those nodes encapsulating segments on row k-1 that overlap with the current segment on the axis. If two segments have the same node set as the first element of the list, then the two entities have to be merged to form a bigger entity. Merging the two entities is a relatively simple operation, with the smaller list being concatenated to the larger one; this involves changing the last element of the large list to point to the first element of the smaller one, modifying the first node reference for all elements in the smaller list and changing the length of the current list. An example of such an entity is displayed in Figure 3.

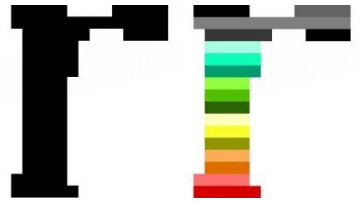


Figure 3. Letter "r" and its maximal horizontal pixel rows

By the time the last element has been analyzed, all segments will have been grouped in entities as shown in Figure 4. Each identified entity has been enclosed by a red axis aligned rectangle. The process appears to have been successful, yet, on closer inspection, it becomes clear that this is not the case.

Figure 4. Letter entities before filtering

As seen in Figure 4, there are several problems. Most characters have indeed been correctly identified, but character fragmentation resulting from the binarization lead to the

identification of some incorrect entities. In the areas where the character stroke is thinnest, page deterioration will eventually cause it to fade altogether. When binarization is applied, these areas look more like background than foreground, therefore they end up being wrongly classified, which in turn causes character segmentation.

As can be observed, the top part of the letter "n" is very "thin" and for one instance it is even fragmented resulting in the creation of two entities. The letters "h" and "u" face a similar problem, due to their similar shape. The "y" character is also fragmented, with the first entity looking like a "v" and the other like a small dash slightly below it.

Another fragmentation type occurs for the symbols "a" and "8"; for these 2, the bounding boxes of the fragments are partially or completely overlapping.

A more inconvenient situation is represented by the merging of the "fi" symbols into a single entity on the first line. This is caused by the fact that too many pixels, the pixels between the two symbols to be more precise, were classified as foreground. Besides this "main" merging issues, there is also a problem with the fact that the dot of "i" which is an entity itself, is overlapping the greater entity formed out of "f" and "i".

A last observation is related to the dots of "i" characters and punctuation marks. Although they are correctly identified, they may cause problems in the future. They are a lot smaller than any other neighboring symbol, and are weirdly placed, either completely above the characters or mostly beneath their level.

Given all the problems detailed above, the necessity of applying some correction filters becomes obvious [5].

First, an inclusion filter is applied, which checks if an entity is found completely within the bounding rectangle of another entity. If this is the case, the two entities are merged. After applying this filter, the fragment looks as in Figure 5. The problem with "a" and "8" has been fixed, and the "fi" entity and the dot of "i" have also been merged into a single entity.

Figure 5. Detailed view of the entities after inclusion filter

Next, a concatenation filter checks if an entity can be connected with another one that is closely placed on the vertical direction. If one of the entities has its horizontal entities bound by the margins of the other and they are close enough, with the distance being beneath a specified threshold, then the two entities become one. The results are presented in Figure 6. This filter solves the problem of the "y" character, uniting the two into a single entity. The distance between the two was significantly greater than the one between the entities that form "n", "h" or "u", but concatenating on the horizontal direction is a much more sensitive problem, because at this point, it cannot be said with any degree of

certainty that the two entities are indeed part of the same symbol. Concatenating horizontal entities can be done, but this might require character features knowledge, which is more appropriate for the OCR phase.

With the entities identified and the corrections applied, they have to be grouped in lines. Grouping entities into lines of text requires iterating through all entities and assigning them to one of the lines. Lines are dynamically created, when an entity cannot be assigned to any other previously created line. Two entities belong to the same line if the y axis projections of their bounding boxes intersect and the horizontal distance between the two is beneath a selected threshold. The threshold is chosen in such a way that lines belonging to adjacent text columns will not be considered as being a single line.

Figure 6. Detailed view of the entities after concatenation filter

Line Detection

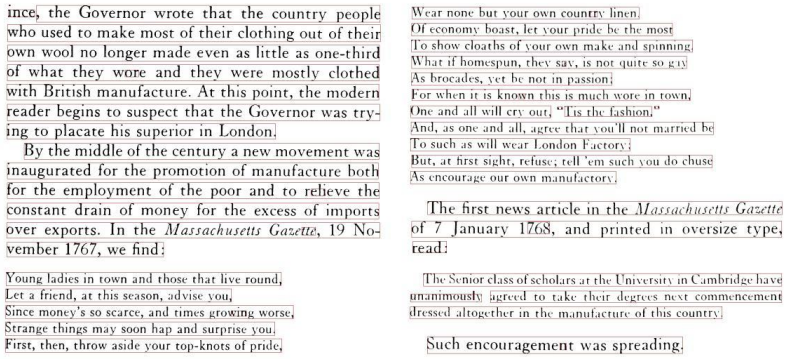


Figure 7. Detected text lines in paper fragment

The previously analyzed sample text has been organized in lines, as seen in Figure 7. Most of the lines are correctly identified, but there are also some exceptions. The first characters of the text are grouped into their own line, most likely because the algorithm failed to eliminate the comma, which in turn changed the limits of the bounding box, so the line could not be evaluated as one. The second problem is related to lines nested within lines, as is the case of the line of the second column that is composed of the characters "768" or the lines made of only the dots of "i".

Paragraph Detection

For the last layout analysis step, the lines are grouped into paragraphs. Determining which lines belong to which paragraph is more than a question of proximity, it is also matter of the features of the characters that compose the lines.

The features that are tracked are:

- character size
- character boldness / character italics
- line spacing

For a well-defined set of entities, character size detection consists of determining the point of minimum and maximum in the entity height histogram. Maxima correspond to lower case letters, upper case letters and potential noise and punctuation marks. If two or more maxima exist, the largest one corresponds to lower case letters, and the others are considered to correspond either to uppercase or punctuation marks if they fit within a predefined interval. If a single maximum exists, the letters are considered to be upper case, and lower case letters are assigned value 0. For better results, the histogram itself is preprocessed, by applying a triangle filter whose width is 10% of the histogram width. By doing so, maxima within a group are greatly outlined.

Determining character boldness is established on a per character basis. The algorithm determines for each pixel that composes the entity the shortest of the vertical and horizontal segments to which the current pixel belongs. Once all pixels have been analyzed, a histogram of the width of the segments is created. The most frequent value represents the stroke width. If, however, there are two lengths of close values (which usually happens for differences of a few pixels) an average of these values is computed and this is used to character boldness.

To determine if a character is italic or not the algorithm evaluates the width of the bounding box of the entity both in its initial form and following a rotation of the entity. The pixels are rotated with 16 degrees (trigonometrically), as italic characters are usually rotated 16 degrees clockwise and the widths for both cases are calculated. The reasoning behind this is that rotation will produce a symbol with a width smaller that the initial letter, whereas rotating a regular character will increase the width.

To determine the distance between row entities, only vertically adjacent entities are evaluated. The algorithm determines if characters are on consecutive lines by comparing the distance between them with an average of the character heights and checking if their horizontal coordinates overlap. When the processing is done, a numeric value is generated, representing the average distance in pixels between rows for the analyzed text area.

Having all these features determined, the lines are analyzed pairwise. First, the heights of the lower case letters of the two are computed; if at least one of the lines has 0 sized lower case letters, then one of the lines contains only uppercase letters, so the algorithm jumps directly to uppercase analysis; otherwise, a ratio of the two values is computed. A similar ratio is computed for the uppercase letter sizes and then the two values are averaged.

If there is a significant difference between the two ratios, there is no need for extra computation, as the two rows will be considered too dissimilar to belong to the same paragraph. Otherwise, the algorithm proceeds with comparing character boldness. The largest valued is kept as reference for both rows and if the difference between the two is below a specified threshold (1 pixel to be more precise), the processing stops. Otherwise, character italics are analyzed and if the difference is not too great the lines are classified as belonging to the same paragraph.

The algorithm successfully separated paragraphs containing italics from those without italics, also in part due to the significant distance between the paragraphs (see Figure 8). Also, the paragraphs containing citations were flawlessly identified. For the other type of paragraphs, there is an error in the first column, splitting the paragraph into two at line 8. The cause of the problem is that the algorithm is blind to line alignment; instead it only relies on line distance and character similarity.

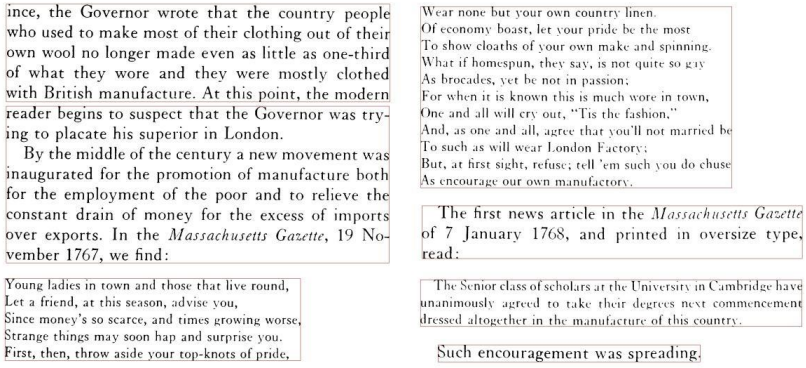


Figure 8. Detected paragraphs in paper fragment

Character Recognition modules

"tesseract_wrapper" binary

This binary handles the character recognition for the system. The binary processes the output of the layout analysis component, extracts the lines of text from the image as indicated by the XML file, and creates image files which it sends as input to the actual Tesseract binary. The output is a text file, containing the OCR matching, which is extracted and added to the output XML file as the contents for the corresponding text line.

Tesseract OCR Overview

Tesseract is an open office OCR engine released under the Apache 2 license. The application was initially developed as a Hawlett-Packard proprietary software, being recognized in 1995 as one of the top 3 best applications of this type. After 1995, though, development on the application stopped, till it was released as open source a decade later. Starting from 2006, its development is supported by Google [23].

Up until version 2.0, Tesseract only accepted single text column TIFF images. Because these initial versions did not offer the possibility of taking into account text placement, the results were usually poor. Starting from version 3.0, Tesseract offers support for formatting the resulting text, as well as other information related to localization and layout conformant to the hOCR standard.

Another improvement to the software was the addition of language support. These are organized as individual files that can be added to the application whenever they are needed.

Tesseract Architecture

Image processing is done the standard way, sequentially. The first step consists of detecting the connected component, and retaining the bounding elements of the components. Based on these, an inclusion hierarchy is created, which, by analyzing the number of levels, permits a uniform handling of the cases of black text on white background and white text on black background.

Following this step, the bounding elements are grouped through inclusion into blobs. These are organized into text lines, which are then split into words, all based on character spacing. If the spacing is uniform, the splitting is done right away. For more complex fonts with complex spacing, word splitting relies on using fuzzy spaces [23].

Character recognition is next, which is done in two steps. First, an attempt is made to identify each individual word. A correctly identified word is used for training an adaptive classifier, thus permitting the identification of future words. Because following the initial traversal it might be that useful information were discovered that were not available when the process began, a new identification is ran, for those words for which the confidence level is not high enough [23].

Symbol Matching

Tesseract employs a best-first strategy over the segmentation graph for identifying characters and words, a search that grows exponentially with the size of the blob. Although this approach works reasonably well for words written with the Latin script (which are relatively few in number) and for whom the search ends as soon as a valid entry is found in the dictionary, for texts using Chinese ideograms, the computational resources are exhausted relatively fast.

To process such a script, it is necessary to limit the segmentation points and establish a halt condition that is easier to reach. Reducing the segmentation level is done by considering the symbols as having fixed width, at most some of them being half the width of a standard symbol.

Another strong constraint is character consistency within a phrase. Introducing more recognizable symbols extends the capacity of correctly identifying and cataloguing, but introduces a certain degree of ambiguity. Although one may say which character set is dominant for a particular text, one cannot eliminate the possibility of having symbols belonging to a different alphabet appear in the text (e.g. Greek letters for formulas). As a result, when analyzing the current word, Tesseract will assume it is using a certain

character set only if more than half of the previous symbols belong to that set. Even in this conditions, other character sets will not be ignored, as the shape recognition score is taken into consideration [23].

The features that are tracked during classification are components of the approximation of the polygonal contour of the character. During the learning phase, a four dimensional feature vector (position, direction, length) is created out of each element of the polygonal approximation; these are then clustered to create prototype vectors. In the recognition phase, the elements of the polygon are split into equally sized fragments, which removes the length component from the vectors. These smaller fragments are compared to the elements that compose the prototype; using small components makes the identification process more reliable to discontinuities related to character representation.

Classifying the symbols is a two-step process; in the first step, the set of possible characters is reduced to a list of about 1 to 10 characters, using a method similar to Locality Sensitive Hashing. The second step computes the distance of the inspected symbol to each of the characters in the list, so that the one with the best score may be chosen.

The good results of the classification process are due to its structure of selection by voting. Instead of using a unique bulky classifier, multiple small classifiers are used instead. To avoid feature explosion in prototypes, Tesseract limits their number to 256, which is more than enough, including for representing Chinese symbols or various syllabic systems.

Document generating modules

"pdf_converter" binary

The binary consists of two parts: one that converts the hierarchy analysis XML file (with possible text contents) into METS/ALTO files, and one that converts these files into the actual PDF using the third party binary mets2pdf.

For starters, the METS file is created; as the aim is to obtain a pdf file, some of the information contained in the XML are either not filled in or contain default values. These data would usually be used for managing a collection of documents and are not relevant to the PDF file. Moreover, in the context of the current document analysis system, the METS file is no more than an intermediary product, which loses its relevance once the final PDF file is generated.

Generating the metsHdr element and its associated subtree is independent of the file (or files if more files are currently dealt with) that are being analyzed. This section concerns the METS file itself and will not be part of the final PDF. For example, the end file will make no use of the creation time of the intermediary file, but this is still filled in nonetheless.

Usually, the elements of the dmdSec section are more relevant to the final PDF, as they are used to create navigation labels within the file. However, as the hierarchy analysis tool is fairly primitive and is having difficulties correctly identifying title, subtitles, chapters and so on, only two such elements are generated, that have as id "MODSMD_PRINT"

and "MODSMD_ELEC" respectively, that allow the introduction of a title by hand by the user; otherwise, the default value is Unknown.

The amdSec elements describe images that compose the document, so the information associated to these is more sensitive, and should not be left to be default. Apart from the actual features of the images, this section also contains information regarding the position within the document.

The next section is called fileSec, and it is composed of two fileGrp sections, under which files of a particular type are grouped. The first one contains the elements describing images, and the second one contains ALTO type files.

The last elements necessary to the METS file are two structMap files. The first one of the two, identified by the "LABEL" attribute "Physical Structure", contains, for this particular application, the elements that associate image elements with ALTO files with the corresponding text. The next structMap element, whose attribute "LABEL" is this time "Logical Structure", describes the hierarchical structure of the document; more exactly, the identified text is logically organized into articles, chapters, subchapters, titles etc. However, given that the current analysis is quite lacking, this hierarchy is missing altogether. Instead, all paragraphs are catalogued as regular text and are associated the identifier corresponding to the appropriate ALTO file.

Oher modules

"controller" binary

This binary, as its name suggests, controls and coordinates the system based on the XML configuration file. The application parses the XML and extracts from the DOM tree the tasks corresponding to each individual component. After the subtree is selected, it is written to a distinct file and given for processing to the appropriate binary. Once the processing is done, the XML result file is analyzed, in order to determine if the processing may carry in or not.

Graphic components

Although editing XML files "by hand" using nothing more than a file editor is not a complicated task, problems may arise as the user must be knowledgeable enough to create valid files. Even for experienced users writing such files may seem like a daunting task, not because of the difficulty, but due to the fact that it is tedious.

As a result, to speed up the interaction with the binaries, several graphic components were created that allow the user to manipulate XML files without having any knowledge of the structure of these file and also to run the individual binaries.

Graphic controller

The graphic controller provides the same functionality as the regular controller, but in a more "user-friendly" manner.

On startup, the controller parses all xsd files in the configuration folder and accordingly groups each binary. As mentioned earlier, each processing task may be performed by multiple binaries, so the user may be able to choose from one of many alternative.

The users begins by choosing a start processing step and an end processing step. Then, for all the processing steps within, they have to choose an appropriate binary and if necessary specify appropriate parameters. The parameters are presented in a user friendly manner as toggles, drop down lists or spinner inputs, as indicated by the type of the components described in the xsd file.

Once the users are satisfied with their configurations, they may launch the processing. This notifies the controller to create the actual XML input files, which are then passed on to the binaries. Just like the regular controller, the graphic controller will check the result XML files and present a report to the users, indicating whether the processing was successful or an error occurred.

Correction interface

Regardless of how good a processing component is, errors are bound to appear. And while deciding on a correct binarization has some degree subjectivity attached to it, deciding whether two entities are part of the same line is a clear cut procedure with well-defined metrics. The correction interface allows the user to fix some of the aberrations that appear as result of the layout analysis or even the OCR processing.

3. CONCLUSION

The presented modules should be regarded as a minimal set that a Document Image Analysis System may need in order to fully complete the entire execution flow. Converting a set of digitally-acquired images into a meaningful output format with proper tagging and ranking is now possible with minimal user interaction and proper configuration of the aforementioned modules.

4. REFERENCES

[1] *How to extend ocropus in c++*. http:// code.google.com/ p/ ocropus/ wiki/ CxxProgramming.

[2] The METAe engine. http:// meta-e.aib.uni-linz.ac.at/ metaeengine/ engine.html.

[3] C. A. Boiangiu, A. I. Dvornic. "*Methods of Bitonal Image Conversion for Modern and Classic Documents*". WSEAS Transactions on Computers, Issue 7, Volume 7, pp. 1081 – 1090, July 2008.

[4] B. S. Almeida, R. D. Lins, and G. D. F. Pereira e Silva. Thanatos: *Automatically retrieving information from death certificates in Brazil*. Proceedings of the 2011 Workshop on Historical Document Imaging and Processing, pages 146-153, 2011.

[5] A. Boiangiu, A. C. Spataru, A. I. Dvornic, and C. C. Cananau. *Normalized text font resemblance method aimed at document image page clustering*. WSEAS Transactions on Computers, July 2008.

[6] F. S. C. da Silva C. Antonio Peanho, H. Stagni. *Semantic information extraction from images of complex documents*. Applied Intelligence, 37(4), December 2012.

[7] S.H. Kim C. B. Jeong. *A document image preprocessing system for keyword spotting*. Proceedings of the 7th international Conference on Digital Libraries: international collaboration and cross-fertilization, December 2004.

[8] The Association for Automatic Identification and Capture Technologies. *Optical character recognition (OCR)*. http:// www.aimglobal.org/ technologies/ othertechnologies/ ocr.pdf.

[9] TROY Group. Micr basics handbook. http:// www.troygroup.com/ support/ documents/50-70300-001_CMICRBasicsHandbook_000.pdf, Accessed: October 2004.

[10] P. W. Handel. *Statistical machine*, June 1933.

[11] R. Holley. *How good can it get? Analyzing and improving OCR accuracy in large scale historic newspaper digitization programs*. D-Lib Magazine, 15(3):1-13, 2009.

[12] Phoenix Software International. *Optical character recognition - what you need to know*. http://www.phoenixsoftware.com/pdf/ocrdataentry.pdf, Accessed: January 2009.

[13] T. Ishihara, T. Itoko, D. Sato, A. Tzadok, and H. Takagi. *Transforming Japanese archives into accessible digital books categories and subject descriptors*. Proceedings of the 12th ACM/IEEE-CS joint conference on Digital Libraries, pages 91-100, 2012.

[14] C. Downton J. He. *User-assisted archive document image analysis for digital library construction*. Proceedings of Seventh International Conference on Document Analysis and Recognition, August 2003.

[15] E. Klijn. *The current state-of-art in newspaper digitization*. D-Lib Magazine, January-February 2008.

[16] L. Likforman-Sulem, P. Vaillant, and A. B. de la Jacopière. *Automatic name extraction from degraded document images*. Pattern Analysis and Applications, 9(2-3), October 2006.

[17] R. D. Lins, G. D. F. Pereira e Silva, and A. D. A. Formiga. *Enhancing a platform to process historical documents*. Proceedings of the 2011 Workshop on Historical Document Imaging and Processing, 0:169-176.

[18] X. Lu, S. Kataria, W. J. Brouwer, J. Z. Wang, P. Mitra, and C. Lee Giles. *Automated analysis of images in documents for intelligent document search*. International Journal on Document Analysis and Recognition, 12(2), June 2009.

[19] E. Matthaiou and E. Kavallieratou. *An information extraction system from patient histor-ical documents*. Proceedings of the 27th Annual ACM Symposium on Applied Computing, page 787, 2012.

[20] N. Otsu. *A threshold selection method from gray-level histograms*. IEEE Transactions on Systems, Man and Cybernetics, 1979.

[21] R. Sanderson, B. Albritton, R. Schwemmer, and H. Van De Sompel. Sharedcanvas: *A collaborative model for medieval manuscript layout dissemination*. Proceedings of the 11th Annual International ACM/IEEE Joint Conference on Digital Libraries, pages 175-184, 2011.

[22] United States Postal Service. *How a letter travels*. http:// about.usps.com/ publications/ pub100/ pub100_078.htm.

[23] R. Smith. *An overview of the Tesseract OCR engine*. Ninth International Conference on Document Analysis and Recognition, 2:629-633, 2007.

[24] H. Toselli, E. Vidal, and A. Juan. *Interactive layout analysis and transcription systems for historic handwritten documents categories and subject descriptors*. Proceedings of the 10th ACM Symposium on Document Engineering, pages 219-222, 2010.

[25] P. Tranouez, S. Nicolas, V. Dovgalecs, A. Burnett, L. Heutte, Y. Liang, and R. Guest. Docexplore: *Overcoming cultural and physical barriers to access ancient documents*. Pro-ceedings of the 2012 ACM symposium on Document engineering, pages 205-208, 2012.

[26] Tigora, M. Zaharescu, "*A Document Image Analysis System for Educational Purposes*", Journal of Information Systems & Operations Management, 05/2013, 7(1), pp. 116-175.

[27] Tigora, "*An Overview of Document Image Analysis System*", Journal of Information Systems & Operations Management, 12/2013; 7(2), pp. 378-390.

[28] A. Boiangiu, A. V. Stefanescu, "*Target Validation and Image Color Calibration*", International Journal of Circuits, Systems and Signal Processing, Volume 8, 2014, pp. 195-202.

[29] A. Boiangiu, A. I. Dvornic. "*Bitonal Image Creation for Automatic Content Conversion*". Proceedings of the 9th WSEAS International Conference on Automation and Information, WSEAS Press, pp. 454 - 459, Bucharest, Romania, June 24-26, 2008.

[30] Tigora, "*Document Image Analysis System*", Master Thesis, Unpublished Work, Bucharest, Romania, 2013.

[31] A. Boiangiu, I. Bucur, A. Tigora - „*The Image Binarization Problem Revisited: Perspectives and Approaches*", The Proceedings of Journal ISOM Vol. 6 No. 2 / December 2012, pp. 419-427.

[32] Andrei Tigora - "*Designing A Flexible Document Image Analysis System – Part 1: The Architecture*", The Proceedings of Journal ISOM, Vol. 10 No. 1 / May 2016 (Journal of Information Systems, Operations Management), pp 235-245.