Document Image Analysis with OCRopus

Faisal Shafait
German Research Center for Artificial Intelligence (DFKI GmbH)
D-67663 Kaiserslautern, Germany
faisal.shafait@dfki.de

Abstract— **Document image analysis is the field of converting paper documents into an editable electronic representation by performing optical character recognition (OCR). In recent years, there has been a tremendous amount of progress in the development of open source OCR systems. OCRopus is one of the leading open source document analysis system with a modular and pluggable architecture. This paper presents an overview of different steps involved in a document image analysis system and illustrates them with examples from OCRopus.**

I. Introduction

Paper documents like books, handwritten manuscripts, magazines, newspapers, etc. have traditionally been used as the main source for acquiring, disseminating, and preserving knowledge. The advent of personal computers has given birth to another class of documents called electronic documents. An electronic document is a representation of a document using data structures that can be understood by computers. Typical examples of electronic documents are PDF, Word, XML, E-mails, Web pages, etc.

Electronic documents offer several advantages over traditional paper documents like easier editing, retrieval, indexing, and sharing since document contents can be accessed electronically. Therefore, most documents are created today by electronic means [1]. An electronic document can be converted into a paper document by means of a printing device. Converting a paper document into electronic form, on the other hand, needs a way to transform the document into data structures that can be understood by computers. A scanning device can be used to obtain a digital image of a paper document. The transformation of a scanned document image into a structured electronic representation is a complex artificial intelligence task and is the focus of research in the field of document analysis and recognition.

A document image may contain different types of contents like text, graphics, half-tones, etc. The goal of optical character recognition (OCR) is to extract text from a document image. This is achieved in three steps. The first step locates text-lines in the image and identifies their reading order. This step is called *geometric layout analysis*. In the second step, text-lines identified by the layout analysis step are fed to a character recognition engine which converts them into an appropriate format (ASCII, UTF-8, ...). Finally, an appropriate language model is usually applied to correct for OCR errors.

Owing to the central role of optical character recognition in digitizing paper documents, several commercial and open source systems are available for OCR. The most notable commercial systems are ABBYY FineReader [2] and Nuance Omnipage [3]. An overview of the leading open source OCR systems is given in Section II. Performing different steps of OCR using the OCRopus open source OCR system will be shown in Section III followed by a summary in Section IV.

II. Open Source OCR Systems

Efforts in developing an open source OCR system started in late 90's. GOCR [4] was among the first open source character recognition engines. Other engines e.g. Clara OCR [5], and Ocrad [6] followed in the last decade. However, the performance and capabilities of these engines are very limited as compared to commercial OCR software. The launch of Google Print project (now called Google Book Search) stimulated a lot of interest in open source OCR systems. HP labs open-sourced their OCR engine called Tesseract in 2005. This followed by the development of a new high-performance OCR system at DFKI in 2007 based on funding from Google Inc. One year later, Cognitive Technologies released the kernal of its Cuneiform OCR system as open-source. These developments in the open source OCR systems over the last few years have made them competitive to commerical OCR systems and hence several research and commercial applications have started using them. A brief history of each of the three leading open source OCR systems is given in the following with a more detailed description of OCRopus OCR system.

A. Tesseract

Tesseract [7], [8] is an open source OCR engine that was developed at HP between 1985 and 1995. In 1995, it was one of the top three OCR engines at the OCR accuracy contest organized by University of Nevada in Las Vegas (UNLV) [9]. Unfortunately, the project was not developed further and in 2005, HP released the code to UNLV's Information Science Research Institute (ISRI), an academic center doing ongoing research into OCR, as open source for the benefit of the community. The code was then picked up by Google for further development and is still maintained and extended at Google.

B. Cuneiform/OpenOCR

Cuneiform [10], developed by Cognitive Technologies in the 90's was one of the first OCR packages to include adaptive character recognition. It remained a competitor

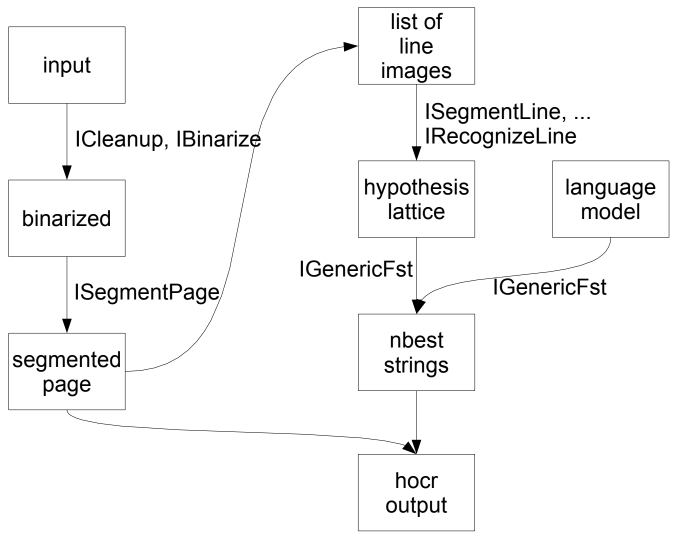


Fig. 1. Architecture of the OCRopus OCR system. Labels on the arcs between different processing blocks show the associated interfaces.

of ABBYY FineReader, but finally lost its market share. After several years with no development, its OCR engine was released as open source in 2008 under the name of OpenOCR [11].

C. OCRopus

OCRopus [12], [13] is a state-of-the-art document analysis and OCR system, featuring pluggable layout analysis, character recognition, statistical language modeling, and multilingual capabilities. The OCRopus engine is based on two research projects: a high-performance handwriting recognizer developed in the mid-90's and deployed by the US Census bureau, and novel high-performance layout analysis methods. Among leading open source OCR systems, OCRopus is unique in the sense that its design and development was done from scratch focusing on a modular and pluggable architecture. Besides, the interfaces have been designed to be able to handle any script and language. Since we will be seeing examples of different document processing steps using OCRopus later in this paper, more detail about its architecture and interfaces is given here.

An illustration of the architecture of OCRopus is shown in Figure 1. Note that the figure just depicts one possible processing pipeline. The exact pipeline can be specified by the user based on the application. A short description of the interfaces and data structures used in OCRopus is given below.

1) OCR Interfaces: Interfaces of different OCR system components are defined in `ocrinterfaces.h`. Class names of interfaces start with I (e.g. IBinarize, ICleanup, . . .). Interface classes themselves define a pure virtual function that should be overridden by the inheriting classes to provide the required functionality. An example interface class is shown below:

```
/// Cleanup for binary document images.
/// Should throw an error when applied
/// to grayscale.
struct ICleanupBinary : IComponent {
  /// Clean up a binary image.
  virtual void cleanup(bytearray &out,
                       bytearray &in)=0;
};
```

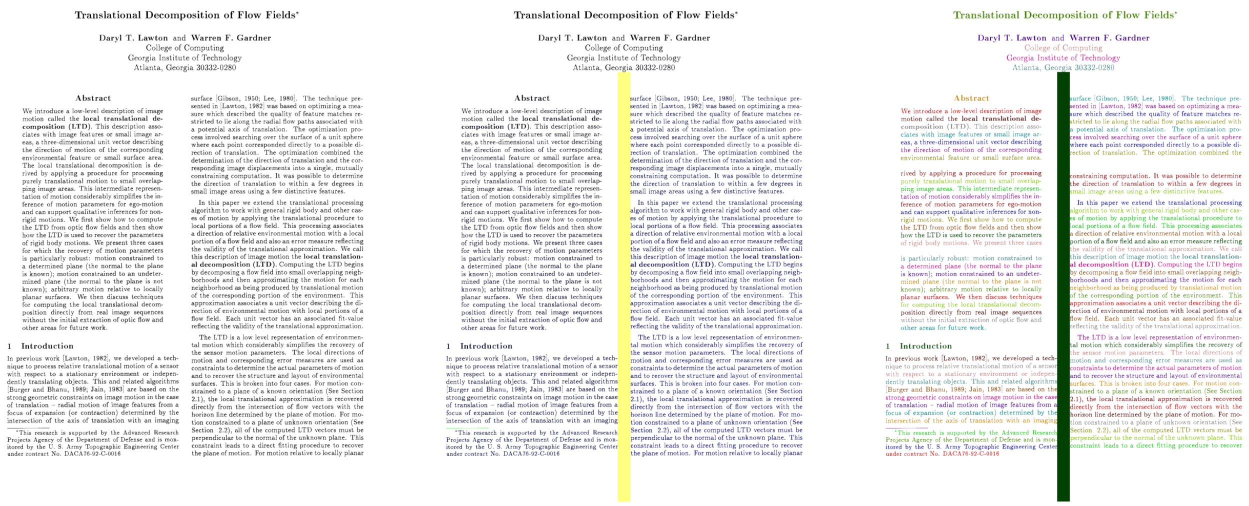
OCRopus uses the `iulib` image processing library [14] for storing image data and for generic image processing operations. Binary and gray scale images are represented as a `bytearray` object, whereas color images are represented as `intarray` objects. Each pixel in the color image is represented as an integer value, where the least significant byte (LSB) represents the intensity of the blue channel, second LSB represents the green channel, and the third LSB represents the red channel. A particular feature of the interfaces is their simplicity. Interfaces for all document image pre-processing steps take a `bytearray` as input and give a `bytearray` as output. Similarly, interfaces for segmenting a page image into zone or lines take a binary image (`bytearray`) as input, and produce a color coded segmentation image (`intarray`) as output.

2) Color Coded Segmentation Representation: The purpose of using color-coded segmentation representation [15] is not only to simplify the interface by avoiding to deal with complicated data structures representing boundaries of segmented regions, but also to give a pixel-accurate representation of segmentation. The segmentation follows a particular color coding convention[1] to represent different levels of segmentation results. For instance, for representing the physical layout of a text document, the red channel encodes the column number of the pixel, the green channel represents the paragraph number of the pixel counted within its column, and the blue channel represents the line number within its paragraph. Special R,G,B values are assigned to other page elements like half-tones, layout separators etc. An example image showing the color coded layout of a document is shown in Figure 2.

3) hOCR Output Format: The hOCR format [16] targets the major writing systems and languages of the world, and defines markup to represent different typographic and linguistic phenomena across a wide variety of languages and scripts. The format represents the output of OCR as an HTML document and therefore can represent these phenomena with already well-defined, widely understood markup. Additional tags are added to embed the OCR engine specific information into the HTML document.

One of the key advantages of the hOCR format over other OCR formats is that it can reuse the expertise that has gone into the development of textual representations for HTML. Generally speaking, all common style-, font-, script-, language-, and typesetting-specific phenomena (hyphenation, spacing, ruby, kashida, etc.) are to be represented using their

[1]http://code.google.com/p/ocropus/



Translational Decomposition of Flow Fields*

Daryl T. Lawton and Warren F. Gardner
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

Abstract

We introduce a low-level description of image motion called the **local translational decomposition (LTD)**. This description associates with image features or small image areas, a three-dimensional unit vector describing the direction of motion of the corresponding environmental feature or small surface area. The local translational decomposition is derived by applying a procedure for processing purely translational motion to small overlapping image areas. This intermediate representation of motion considerably simplifies the inference of motion parameters for ego-motion and can support qualitative inferences for nonrigid motions. We first show how to compute the LTD from optic flow fields and then show how the LTD is used to recover the parameters of rigid body motions. We present three cases for which the recovery of motion parameters is particularly robust: motion constrained to a determined plane (the normal to the plane is known); motion constrained to an undetermined plane (the normal to the plane is not known); arbitrary motion relative to locally planar surfaces. We then discuss techniques for computing the local translational decomposition directly from real image sequences without the initial extraction of optic flow and other areas for future work.

1 Introduction

In previous work [Lawton, 1982], we developed a technique to process relative translational motion of a sensor with respect to a stationary environment or independently translating objects. This and related algorithms [Burger and Bhanu, 1989; Jain, 1983] are based on the strong geometric constraints on image motion in the case of translation – radial motion of image features from a focus of expansion (or contraction) determined by the intersection of the axis of translation with an imaging surface [Gibson, 1950; Lee, 1980]. The technique presented in [Lawton, 1982] was based on optimizing a measure which described the quality of feature matches restricted to lie along the radial flow paths associated with a potential axis of translation. The optimization process involved searching over the surface of a unit sphere where each point corresponded directly to a possible direction of translation. The optimization combined the determination of the direction of translation and the corresponding image displacements into a single, mutually constraining computation. It was possible to determine the direction of translation to within a few degrees in small image areas using a few distinctive features.

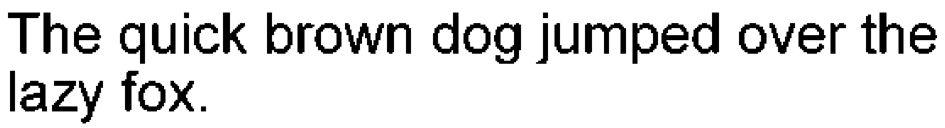
In this paper we extend the translational processing algorithm to work with general rigid body and other cases of motion by applying the translational procedure to local portions of a flow field. This processing associates a direction of relative environmental motion with a local portion of a flow field and also an error measure reflecting the validity of the translational approximation. We call this description of image motion the **local translational decomposition (LTD)**. Computing the LTD begins by decomposing a flow field into small overlapping neighborhoods and then approximating the motion for each neighborhood as being produced by translational motion of the corresponding portion of the environment. This approximation associates a unit vector describing the direction of environmental motion with local portions of a flow field. Each unit vector has an associated fit-value reflecting the validity of the translational approximation.

The LTD is a low level representation of environmental motion which considerably simplifies the recovery of the sensor motion parameters. The local directions of motion and corresponding error measures are used as constraints to determine the actual parameters of motion and to recover the structure and layout of environmental surfaces. This is broken into four cases. For motion constrained to a plane of a known orientation (See Section 2.1), the local translational approximation is recovered directly from the intersection of flow vectors with the horizon line determined by the plane of motion. For motion constrained to a plane of unknown orientation (See Section 2.2), all of the computed LTD vectors must be perpendicular to the normal of the unknown plane. This constraint leads to a direct fitting procedure to recover the plane of motion. For motion relative to locally planar

*This research is supported by the Advanced Research Projects Agency of the Department of Defense and is monitored by the U. S. Army Topographic Engineering Center under contract No. DACA76-92-C-0016

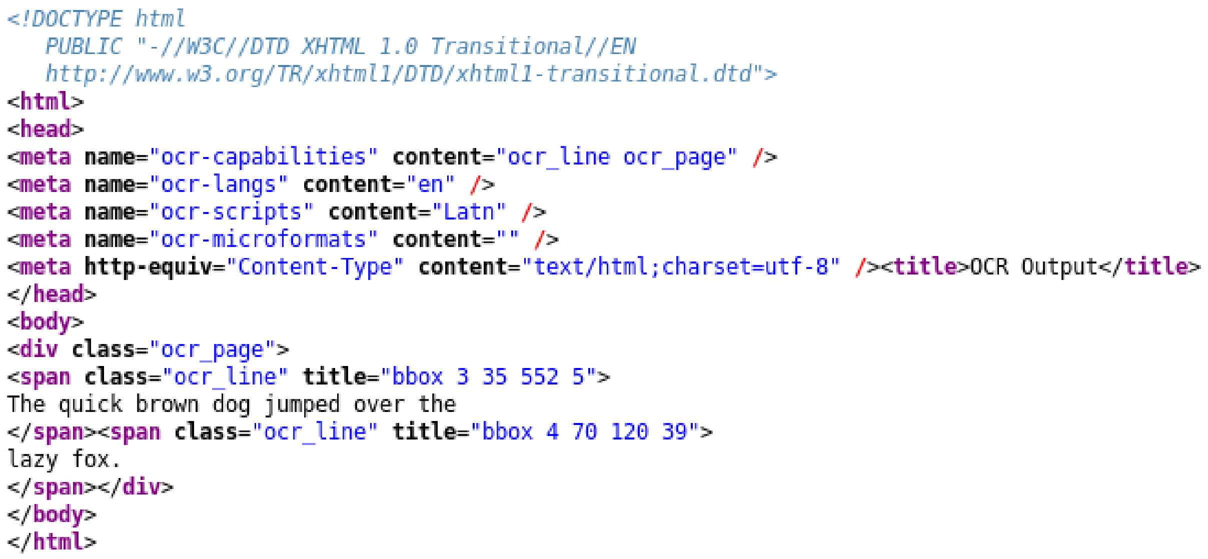
(a) Original Image (b) Layout Representation (c) Recolored Layout

Fig. 2. Color-coded representation of page layout. The color of each line represents its line and column number. Consecutive lines in the original representation only differ by a gray value of 1 and hence are hard to distinguish. The recolored image on the right is obtained by simple_recolor() method and is used for visualization of segmentation results.



The quick brown dog jumped over the
lazy fox.

(a) Input Image



```
<!DOCTYPE html
   PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN
   http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta name="ocr-capabilities" content="ocr_line ocr_page" />
<meta name="ocr-langs" content="en" />
<meta name="ocr-scripts" content="Latn" />
<meta name="ocr-microformats" content="" />
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" /><title>OCR Output</title>
</head>
<body>
<div class="ocr_page">
<span class="ocr_line" title="bbox 3 35 552 5">
The quick brown dog jumped over the
</span><span class="ocr_line" title="bbox 4 70 120 39">
lazy fox.
</span></div>
</body>
</html>
```

(b) OCRopus output in hOCR format

Fig. 3. hOCR output of a sample image. The contents of the document are embedded in a hierarchical structure. ocr_page tag encapsulates contents of each page, whereas ocr_line tag represents text of one line. Layout analysis information in terms of line bounding boxes is embedded within the title of the ocr_line tag.

HTML or Unicode representations. A sample image and the output of OCRopus for that image are shown in Figure 3.

III. Text Recognition Using OCRopus

A typical OCR system consists of three key components:

1) Geometric layout analysis identifies the location of text-lines in the scanned documents.
2) Text-line recognition classifies the characters in the text-lines into letters of a pre-defined alphabet.
3) Language modeling attempts to correct text-line recognition output by using language specific information.

Each of these steps is illustrated in the following using example code from OCRopus. A simple C++ toplevel file for performing OCR on an input image is shown in Figure 4. The source code also serves the purpose of demonstrating the clear separation of different processing steps in OCRopus, thereby making it easy to plug customized components or replace existing components.

A. Geometric Layout Analysis

Geometric layout analysis of a document image typically involves different processes. The exact order in which these processes are applied varies from one algorithm to another. Also, some algorithms might skip one or more of these processes or apply them in a hybrid way. However, most of the layout analysis systems use these processes in some form. Therefore, a brief outline of these processes is given here.

- **Binarization** is the process that converts a given input greyscale or color document image into a bi-level representation [17]. Since scanners produce a grayscale image by default, it is usually the first step in any document processing pipeline. Binarization algorithms in OCRopus implement the `IBinarize` interface. In Figure 4, after reading the image in Lines 16-17, binarization using the Otsu algorithm [18] is applied (Lines 19-21).
- **Noise removal** is a process that tries to detect and remove noise pixels in a document that are introduced by scanning or binarization process [19]. Noise removal methods for binary and grayscale images implement the `ICleanupBinary` and `ICleanupGray` interfaces respectively.
- **Skew correction** is a process that detects and corrects the deviation of a document's orientation angle from the horizontal direction [20]. Such a deviation naturally occurs due to paper positioning variations on the scanner. Skew correction methods in OCRopus also implement the `ICleanupBinary` and `ICleanupGray` interfaces.
- **Text/Image segmentation** is a process that classifies page regions into one of a set of predefined classes (e.g. text, image, graphics, . . .) [21], [22]. Methods for separating text and non-text regions in an image implement the `ITextImageClassification` interface.
- **Layout analysis** is a process that idenitifies text-lines in a document image while respecting the columnar structure of the document and extracts them in an appropriate reading order [23], [24]. In OCRopus, layout analysis algorithms implement the `ISegmentPage` interface. Our sample code in Figure 4 applies layout analysis directly after binarization (Lines 23-25).

Note that for each of the above mentioned steps, implementations of several state-of-the-art algorithms are available in OCRopus. Each method provides a constructor of the form `make_...()`. Hence replacing one component by another is simply achieved by calling the constructor of the new component. For instance, to use Sauvola [25], [26] binarization algorithm instead of Otsu, we just need to replace the `make_BinarizeByOtsu()` argument with `make_BinarizeBySauvola()` in Line 19 of Figure 4. Note that the interface class pointers are held in an `autodel` object, which is a smart pointer class that automatically deletes the objects its pointing to as soon as it goes out of scope.

B. Textline Recognition

Textline recognition is the process of classifying the characters in the text-line image obtained as a result of layout analysis. It proceeds into two steps. The first step is to segment the line image into isolated character images. In the second step, the isolated character images are fed to a trained classifier for recognition. The segmentation of a textline into characters is achieved by algorithms implementing `ISegmentLine` interface, whereas recognition of these components individually is performed by classes inheriting from `ICharacterClassifier` interface. However, this approach has certain limitations when applied to real world documents scanned under a wide variety of conditions. Due to scanning artifacts, characters might end up broken or merged with other characters. In such cases accurately spotting characters in a text-line is not possible. Secondly, when isolated characters are fed to the character classifier without the knowledge of their relative size w.r.t. other characters in the line and their vertical position w.r.t. the baseline; distinctions between many uppercase and lowercase letters are not possible (for instance "O" and "o", "S" and "s", "P" and "p" etc.).

To handle these cases a more sophisticated text-line recognition technique is required. OCRopus provides a generic interface for text line recognition called `IRecognizeLine`. This interface provides support for storing segmentation hypothesis of a text-line. These segmenatation hypotheses are represented as a weighted finite state transducer (FST) in which different segmentation possibilities are encoded as different paths in the transducer. Such a graph is termed as "segmentation graph". The segmentation graph is implemented as an instance of the `IGenericFst` interface. The classifier tries to recognize each of the segmentation hypothesis and associates a cost with it. The most likely recognition result is then obtained by the lowest cost path through the segmentation graph.

```cpp
#include "ocropus.h"
#include "glinerec.h"

namespace glinerec { IRecognizeLine *make_Linerec(); }

using namespace iulib;
using namespace ocropus;
using namespace glinerec;

int main(int argc, char **argv){
    try {
        if(argc!=3) {
            fprintf(stderr, "Usage: ... input.png output.txt\n");
            exit(1);
        }
        bytearray img_gray;
        read_image_gray(img_gray,stdio(argv[1],"r"));

        autodel<IBinarize> binarizer(make_BinarizeByOtsu());
        bytearray img_binary;
        binarizer->binarize(img_binary,img_gray);

        autodel<ISegmentPage> segmenter(make_SegmentPageByRAST());
        intarray img_layout;
        segmenter->segment(img_layout,img_binary);

        init_ocropus_components();
        init_glclass();
        init_glfmaps();
        init_linerec();
        autodel<IRecognizeLine> linerec(make_Linerec());
        linerec->load("/usr/local/share/ocropus/models/default.model");

        autodel<OcroFST> langmod(make_OcroFST());
        langmod->load("/usr/local/share/ocropus/models/default.fst");
        int beam_width = 100;

        RegionExtractor regions;
        regions.setPageLines(img_layout);
        for(int i=1;i<regions.length();i++) {
            bytearray line_image;
            regions.extract(line_image,img_gray,i,1);
            autodel<OcroFST> result(make_OcroFST());
            linerec->recognizeLine(*result,line_image);
            nustring str;
            double cost = beam_search(str,*result,*langmod,beam_width);
            if(cost>1e10) throw "beam search failed";
            iucstring output;
            output.clear();
            output.append(str);
            printf("%s\n",output.c_str());
        }

    }
    catch(const char *e) {
        fprintf(stderr,"Exception thrown: %s\n",e);
    }
}
```

Fig. 4. A simple toplevel C++ file for OCRopus taking a grayscale PNG image as input and sending the recognized text to stdout.

The sample source code in Figure 4 first initializes different components of line recognition in Lines 27-30. Then, an instance of a line recognition module is created in Line 31, and a trained model for line recognition is loaded in Line 32. The `RegionExtractor` class (Lines 38-39) takes the color-coded output of layout analysis and extracts images of individual text-lines (Line 41-42) in reading order. The `for` loop (Line 40) iterates over all text-lines in the image. The index of the loop starts from 1 since the index 0 represents page background. A single text-line image is then fed to the text-line recognition module in Lines 43-44. Note that the output of text-line recognition is an `OcroFST` object representing the segmentation graph as a finite state transducer. At this stage the lowest cost output of the classifier can be obtained by using the `bestpath()` method of the `OcroFST` output object.

Although well-trained text-line recognizers tend to be very accurate in recognizing text, there are some ambiguities that can not be resolved with high confidence without further linguistic analysis. One such example is the distinction between "l" (lower case letter L), "I" (upper case letter I), and "1" (digit). In some fonts like Sans Serif, upper case I and lower case L are represented with the same ligature making it impossible for the text-line recognizer to distinguish between the two. Therefore language modeling is used to post-process the output of text-line recognition.

C. Statistical Language Modeling

The purpose of statistical language modeling is to provide linguistic information to the OCR system for better recognition results. The language model might consist of a simple dictionary based lookup. Alternatively, it may provide statistical information about the frequency of occurrence of different characters/words (uni-gram language model). A even more sophisticated language model might provide information about the frequency of occurrence of a sequence of letters/words (n-gram language model). All of these language models can also be conveniently represented as a weighted finite state transducer. Hence language models in OCRopus implement the `IGenericFst` interface. This representation has the advantage that applying a language model to the output of text-line recognition result (also an FST) is a simple "compose" operation of two FSTs.

A language model is loaded and initialized in Lines 34-36 in the source code shown in Figure 4. This model is then used in the `beam_search()` method (Line 46) to find the most likely recognition output that also takes into account linguistic information. The beam search method finds the best path in the composition of two FSTs by constraining the search to a narrow region (beam) of the FST.

The output string `str` (Line 45) returned by the `beam_search()` method is the recognition output. It is an instance of class `nustring` which is a string class representing characters in UTF-32 encoding. Hence each index of the string represents one code point in Unicode. For sending the output to stdout, the string is converted to an `iucstring` object which is a UTF-8 encoded string (Lines 48-50). This UTF-8 encoded string can then be sent to stdout (Line 51) or stored in an output text file.

IV. Summary

This paper presented an overview of the state of the art in open source optical character recognition (OCR). The architecture of OCRopus open source OCR system was illustrated highlighting its interfaces and input/output formats. Sample code based on OCRopus was shown to demonstrate different steps of a typical OCR process.

References

[1] A. Holmes. Publishing trends and practices in the scientific community. *Canadian Journal of Communication*, 29:359–368, 2004.

[2] http://finereader.abbyy.com/.

[3] http://www.nuance.com/imaging/products/omnipage.asp.

[4] http://jocr.sourceforge.net/.

[5] http://freshmeat.net/projects/claraocr/.

[6] http://www.gnu.org/software/ocrad/.

[7] R. Smith. An overview of the Tesseract OCR engine. In *Proc. 9th Int. Conf. on Document Analysis and Recognition*, pages 629–633, Curitiba, Brazil, Sep. 2007.

[8] http://code.google.com/p/tesseract-ocr/.

[9] S. V. Rice, F. R. Jenkins, and T. A. Nartker. The fourth annual test of OCR accuracy. Technical report, Information Science Research Institute, University of Nevada, Las Vegas, 1995.

[10] http://www.cuneiform.ru/eng/.

[11] http://en.openocr.org/.

[12] T. M. Breuel. The OCRopus open source OCR system. In *Proc. SPIE Document Recognition and Retrieval XV*, pages 0F1–0F15, San Jose, CA, USA, Jan. 2008.

[13] http://code.google.com/p/ocropus/.

[14] http://code.google.com/p/iulib/.

[15] F. Shafait, D. Keysers, and T. M. Breuel. Pixel-accurate representation and evaluation of page segmentation in document images. In *18th Int. Conf. on Pattern Recognition*, pages 872–875, Hong Kong, China, Aug. 2006.

[16] T. M. Breuel. The hOCR microformat for OCR workflow and results. In *Proc. Int. Conf. on Document Analysis and Recognition*, pages 1063–1067, Curitiba, Brazil, Sep. 2007.

[17] M. Sezgin and B. Sankur. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, 13(1):146–165, 2004.

[18] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Trans. Systems, Man, and Cybernetics*, 9(1):62–66, 1979.

[19] F. Shafait, J. van Beusekom, D. Keysers, and T. M. Breuel. Document cleanup using page frame detection. *Int. Jour. on Document Analysis and Recognition*, 11(2):81–96, 2008.

[20] D. S. Bloomberg, G. E. Kopec, and L. Dasari. Measuring document image skew and orientation. In *Proc. SPIE Document Recognition II*, pages 302–316, San Jose, CA, USA, Feb. 1995.

[21] Y. Wang, I. Phillips, and R. Haralick. Document zone content classification and its performance evaluation. *Pattern Recognition*, 39(1):57–73, 2006.

[22] D. Keysers, F. Shafait, and T. M. Breuel. Document image zone classification - a simple high-performance approach. In *2nd Int. Conf. on Computer Vision Theory and Applications*, pages 44–51, Barcelona, Spain, Mar. 2007.

[23] T. M. Breuel. Two geometric algorithms for layout analysis. In *Proc. Document Analysis Systems*, volume 2423 of *Lecture Notes in Computer Science*, pages 188–199, Princeton, NY, USA, Aug. 2002.

[24] F. Shafait, D. Keysers, and T. M. Breuel. Performance evaluation and benchmarking of six page segmentation algorithms. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 30(6):941–954, 2008.

[25] J. Sauvola and M. Pietikainen. Adaptive document image binarization. *Pattern Recognition*, 33(2):225–236, 2000.

[26] F. Shafait, D. Keysers, and T.M. Breuel. Efficient implementation of local adaptive thresholding techniques using integral images. *Proc, of SPIE Electronic Imaging: Document Recognition and Retrieval*, 6815:81510–81510, 2008.