Introduction

Even with all the existing alternatives, nowadays a lot of information is still printed on paper.

From historical documents to places that only recently started to use digital documents in their usual workflow, this information is conditioned by the limitations of paper: hard to index and search, risk of deterioration, ecological implications, etc.

Fortunately the state of the art Optical Character Recognition (OCR) engines, including Free Software solutions, have high success rates in converting printed text into a digital format. However, these command-line tools only convert graphics in text, usually not taking into consideration the layout of the documents analyzed. This means that a regular page with a mixture of text in columns and eventual graphics will be read as if it was only text.

OCRFeeder was created to solve this issue and offer an easy to use and complete application, for the GNOME desktop environment, that can convert document images into an editable format. It was developed as the Master's Thesis project of Joaquim Rocha. After this, it continued to be improved and it now developed on the GNOME Project's infrastructure.

Document Layout Analysis

OCRFeeder defines its own algorithm to perform document layout analysis, that is, to detect where the document's contents are.

To outline the contents of a page, OCRFeeder has its own algorithm which was named as Sliding Window Algorithm.

The Sliding Window Algorithm

This algorithm divides the document in small windows whose size is calculated by dividing the document's height by a predefined value which can be configured (default: 60). Then it goes from top to bottom, left to right and identifies, for each window, points that contrast with the background. After this, it gathers groups of contrasting windows. In the end, this information can
be used to define bounding boxes for the document's contents.

This is obviously a simple overview of the algorithm. It has some optimizations for saving processing time and uses heuristics when gathering the contrasting windows because their configuration doesn't usually represent perfectly rectangular forms.

Unlike some algorithms that try to match a document's structure with some previously known one (like a scientific paper) this method doesn't take into account any particular structure. The downside of this aspect is that it is not able to know what the contents semantically mean in the document, that is, it cannot tell if something is a title, footnote or page number. On the other hand, this is an intended characteristic so the recognition is versatile when attempting to find the contents of documents with different layouts.

**Improving the results**

Depending on the structure of the document, sometimes columns of text are identified as being a whole paragraph. To solve this issue, there is a post-detection method which goes through every content area and tries to divide it in columns. Besides this, the content areas normally have a considerably large margin between its borders and its contents. This case can be fixed by another post-detection method which adjusts the content areas to its contents.

Both these methods may be performed after an image's is added to the application and may be turned on or off from the preferences dialog.

**Manual edition of results**

From the graphical user interface it is possible to manually edit any result of the automatic layout analysis or to define ourselves the content areas from the beginning.

When the automatic recognition is finished, the page will show the content areas with boxes drawn on it. These boxes can be repositioned by dragging them or editing their bounds as well as removed from the page.

**Optical Character Recognition**

Having detected the contents of a document's page, OCRFeeder uses system-wide OCR engines to convert them into text. Any installed OCR engine that can be used from the command-line is able to be used in OCRFeeder.

The OCR engines can be configured from the OCR engines manager dialog. This dialog shows the engines currently in use and allows to add more, edit or delete them. Their configuration is as easy as specifying their name, path to the executable, expected image format and the command arguments. The command arguments are the ones that usually are typed from the command-line and there are two keywords, $IMAGE and $FILE that will be replaced by the
image to be recognized and a temporary file, respectively. Since this versatile configuration can be complicated for non-technical users, the most known OCR engines available for GNU/Linux are automatically detected and configured. Those are Tesseract, GOCR, Ocrad and Cuneiform.

Content classification

With the automatic recognition, after having the contents' text from the OCR engines, the content areas are classified as graphics or text using heuristics: the recognized text is compared with patterns usually found in areas that represent graphics content.

Manual edition of results

The text assigned to each content area may also be set or edited manually as well as their classification.

Improving the results

Usually, OCR engines return the text recognized line-by-line from the original image clip. This means that they will include newline characters and hyphenation which may not be what users want; this is especially true when talking about visually impaired users, who use a screen-reader to read the text. In order to correct these details, there is an optional post-recognition method that attempts to “clean” the recognized text using regular expressions.

Input formats

Unlike the OCR engines it uses, OCRFeeder may receive the most common image formats as it will convert those to the engines' expected formats. Apart from images, OCRFeeder is also able to import PDF documents.

Importing from a scanner device

To speed the workflow of having a scanned document imported into OCRFeeder, detect it and generate an editable version of it, there is support for importing an image from a scanner device.

The usage of this feature is rather simple as it obtains the image directly from the configured scanner device or, in case there is more than one device connected, shows a dialog with the list of devices for the user to choose from.
**Image deskewing**

Sometimes images, especially those that come from scanner devices, are slightly rotated and this can influence the recognition results. To solve this, there is a method for deskewing the image using the Hough Transform. This feature can be performed automatically after an image's addition or manually to any loaded image.

**Output formats**

Currently OCRFeeder exports the documents as OpenDocument Text, HTML or plain text (in this case, obviously, only having the contents classified as text).

**Saving and loading projects**

To allow for a continuous work on a document, OCRFeeder can save and load its own projects format. It saves and loads .ocrf files which are zipped archives containing XML files that describe the content areas, pages, etc. and the original images so they can be used in different machines.

**Accessibility**

Due to its capabilities, OCRFeeder is a very useful project for visually impaired users. The usual use case for these users is to use the application with its automatic recognition features and generate an document that can be later read with a screen-reader like Orca. Because of this, OCRFeeder tries to be as accessible as it can and many UI changes have been carried out to improve this.

Joaquim Rocha, Igalia, S.L.

OCRFeeder is released under the terms of the GPLv3 license.

**Resources**

OCRFeeder homepage: [http://live.gnome.org/OCRFeeder](http://live.gnome.org/OCRFeeder)

OCRFeeder source code repository: [http://git.gnome.org/browse/ocrfeeder/](http://git.gnome.org/browse/ocrfeeder/)

OCRFeeder bug tracking: [https://bugzilla.gnome.org/buglist.cgi?cmdtype=runnamed&namedcmd=OCRFeeder](https://bugzilla.gnome.org/buglist.cgi?cmdtype=runnamed&namedcmd=OCRFeeder)

GNOME Project: [http://www.gnome.org](http://www.gnome.org)