Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Graphical tools for ground truth generation in HTR tasks

Proyecto Final de Carrera

Ingeniería Informática

Autor: Jorge Martínez Vargas

Director: Moisés Pastor i Gadea

February 6, 2014

Abstract

This report will cover the development of several graphical tools for ground truth generation in HTR tasks, specifically for layout analysis, line segmentation, and transcription, as well as one ad hoc tool needed for point classification in an implemented line size normalization method. It will show the design process behind the tools, giving an overview of the internal structure through class diagrams. It will also explain the mentioned phases of the HTR with the aim of clarifying each tool context and utility. Finally, the report will close with a brief conclusions and considerations about the future of the tools.

Keywords: HTR, graphical tools, layout analysis, baseline detection, line size normalization, ground truth

Resum

Aquest informe cobrirà el desenvolupament de diverses ferramentes gràfiques utilitzades en la generació de ground truth en tasques de reconeixement de text manuscrit (HTR), especificament anàlisi de layout, segmentació en línies i transcripció, així com una ferramenta ad hoc requerida per a la classificació de punts necessària en un mètode de normalització de tamany de línia que vam implementar. Mostrarà el procès de disseny previ al desenvolupament de les ferramentes, donant una visió general de l'estructura interna a través de diagrames de classe. També explicarà les diferents fases del procès de HTR previament mencionades, amb l'intenció de clarificar el context i l'utilitat de les diferents ferramentes. Finalment, l'informe acabarà amb unes breus conclussions i algunes consideracions sobre el futur de les ferramentes.

Keywords: HTR, ferramentes gràfiques, anàlisi de layout, detecció de baselines, normalització de tamany de linia, ground truth

Resumen

Este informe cubrirá el desarrollo de diversas herramientas gráficas utilizadas en la generación de ground truth en tareas de reconocimiento de texto manuscrito (HTR). Específicamente análisis de layout, segmentación en lineas y transcripción, asi como una herramienta ad hoc requerida para la clasificación de puntos necesaria en un método de normalización de tamaño de línea que implementamos. Mostrará el proceso de diseño previo al desarrollo de las herramientas, dando una visión general de la estructura interna a través de diagramas de clase. También explicará las diferentes fases del proceso de HTR previamente mencionadas, con la intención de clarificar el contexto y la utilidad de las diferentes herramientas. Finalmente, el informe acabará con unas breves conclusiones y algunas consideraciones sobre el futuro de las herramientas.

Keywords: HTR, herramientas gráficas, análisis de layout, detección de baselines, normalización de tamaño de línea, ground truth

Contents

1 Introduction

Nowadays there are a big amount of ancient documents that have been preserved through time, and most of this documents contain useful information. Of course the usefulness of the texts is often limited by diverse factors, being availability one of the most important. That is the reason why the possibility of scanning them, making them available through the Internet to a much broader public, is almost a need, as it has made sharing documents much easier, allowing experts and research groups around the world to work on the same document without having to move the original nor the researchers.

Although having access to the documents is a big step forward, it does not always mean that you have access to the information contained within them. On the one hand the condition, the calligraphy and many other factors can make the documents almost impossible to read for non-experts. On the other hand, some archives have such a big amount of documents that searching for an specific topic or content is not possible in a reasonable amount of time. The obvious solution to both this problems is to have the texts transcribed, as this would allow both reading and automatic searching. As a side effect this would open the door for translation, making the information accessible to even more people. The problem is that the same difficulties that apply for accessing the information are there for translating the texts, making it a really time consuming task, and therefore really expensive.

All the mentioned in the previous paragraph leads us to the necessity of having good automatic transcription tools, able to go from the scanned document to a plain text transcription without human intervention, or at least minimizing the human effort as much as possible. In this context we will explain why, even when we aim at automating the process, supervised learning is needed, setting the focus of this report on explaining the tools we have developed to assist the supervision of data.

The use of graphical tools to assist in data manipulation processes is nothing new, and yet developing friendly and useful graphical user interfaces (GUIs), adapted to the user needs and expectations, is not an easy task, nor a completely solved one.

In our case, the context of the project is that of handwritten text recognition (HTR) tasks, ranging from page layout analysis to line size normalization, and even the final transcription process. As several tools were developed we will try to be general when possible and specific when needed.

This document will start by justifying the need of manually supervised data for HTR, and therefore the need for our tools. The next section will cover the tools design. It will first give some insight in how GUI programming works, and then focus on the ideas behind our design, starting with the general requisites and showing the decisions made to fit them. After this general ideas it will be shown through diagrams and class definitions the specifics of how the problem has been solved.

After explaining the design process, the next section will review the implemented tools, trying to make clear the specific use of each one and the relation between them. In order to do that we will describe the context of each tool, i.e. in which projects and processes are being used. This section will also explain the features of the tools and its use, trying to give the reader a good idea of why the developed software is quick to learn and work with, increasing its usefulness.

The document will close with some brief conclusions and future work.

2 Why supervised learning?

Most of the problems in HTR (also in transcription, and in general in pattern recognition) can be seen as classifying samples into a set of classes, or equivalently, assigning each sample a label. In order to do that we use statistical models such as hidden Markov models (HMMs). Those models learn from a set of data and then classify the new samples that you feed them, but of course for them to learn correctly you need to make sure that the learning data does not contain any errors, and there lies the problem.

When facing a new task it is usual to find that you do not have any labeled data, and therefore you cannot train any models. There are different solutions to that problem, such us using a heuristic or training a first model with similar data from a different problem, and using this to generate some initial data. The problem of this solutions is that, no matter how good is your heuristic or how similar the problem you used data from is, you will end up with some errors in your generated initial data. But, as we have already said, you need perfect data for training if you want a good model. Additionally, you will also need perfect data for testing your models and measuring the error rate, and the test data needs to be different from the training one.

This need for ground truth data (i. e. data we assume to have no errors) is what forces us to supervise the initial data we want to use, whether it is generated in some inexact way or labeled from scratch. Of course this initial data could be generated/supervised by just editing the plain text or XML files that contain the samples, but since the information is usually associated to an image it will be really hard to do for a human. Therefore, a graphical tool is needed to make this supervision quick and easy for the user.

3 Tools design

As we have already said, designing a graphical tool is not an easy task, but fortunately there are some general design principles that can serve as guidelines when doing so.

First off, we wanted our tool to be **flexible** and **easy to modify** or upgrade, specially being in a research environment where the necessities (and consequently the specifications) of a tool are continuously evolving. This was one of the reasons to go for object oriented programming (OOP), specifically in C++, and in this chapter you would find how we structured our code towards this objective.

Another early decision was to use **QT**[1] for the graphical interface. We choose this library because it is very complete and cross platform while being under GPL v3 license, which means it is open source. While programming GUIs it works with the user's graphical system, making the tool look familiar to him right from the start. It also has a lot of available tools, like its own integrated development environment (IDE).

The second important principle was **friendliness**, meaning that the tool needed to be easy to understand and use. If a user needed to spend too much time learning how to use the tool, or if it was too slow in performing the required tasks, the whole purpose of the tool would have been defeated. In order to achieve this goal we designed a GUI as intuitive as possible, trying to make it light by including only the necessary information. Also, when more general, multi-purpose tools were needed, we divided the functionalities into several interaction modes, each one for an specific task. Additionally, several keyboard shortcuts where added, meaning that once the user has been working with the tool for a bit and has learned the shortcuts his work will be much faster. Details about shortcuts and interaction with the tools will be explained in the sections that describe each tool.

[1]http://qt-project.org/

3.1. GUI programming

GUI programming is very different to traditional, non-interactive programming. We want to highlight some of the differences since they have had a considerable impact both in the design and the implementation of the tool.

In the first place the focus on GUI programming is on the user, in opposition to having the focus on the process. This means that interaction is very important, needing to be as intuitive and quick as possible, and also means that the system is event based. Instead of doing a set of operations and returning a result you want the system to react to the user input, mainly via mouse clicks, and to do so in a short amount of time.

We do this in QT by inheriting from their base classes and implementing the methods that catch the events, for instance *mousePressEvent*. This works only when the user interacts directly with an element, though. To handle communication between elements in the GUI (for instance if we want the display to react to a click on a button) the library uses a signal/slot model[2]. The idea behind this model is to have elements emit a signal when they want to communicate something, then connect this signal to a method (called slot) in another element. It is important to note that every element in the GUI, from the main window to each button or even each option in a menu is an entity, making the use of signals and slots a must. The use of this signals and slots requires an additional compilation step to translate the signal/slot definitions to standard C++, but if you use the provided tools (like qmake) it is transparent to both the programmer and the user.

The second important characteristic of GUI programming is that most of the times the user performs an action he is expecting to see the outcome of that action in a short amount of time. That is something that needs to be understood, and any heavy computation should be made, when possible, on batch programs, using the GUI application only for things that require interaction. A problem we had related to this was that the images we used were really big, having loading times of two or three seconds. This may seem unimportant for one image, but it becomes relevant when working with large sets. Since we could not solve this problem by modifying the software we decided to change the format to a compressed one, having to find an equilibrium between compression and image quality.

[2]http://qt-project.org/doc/qt-5.1/qtcore/signalsandslots.html

3.2. Tools internal structure

In this section we will present diagrams showing the classes implemented for each tool, along with the important relations between this classes. In order to do that we are going to divide the developed tools into two families: the first one is that of the single-purpose tools. This tools were the first ones to be developed and they all follow the same structure, inheriting from some base classes that we implemented (that in turn inherited from the QT classes) and adapting the specifics to the problem that the particular tool aimed to solve. The second one is the multi-purpose family, to which belongs only the *Ground truth generation tool*. For the implementation of this last tool some code was reutilized but the classes inherited from the QT base classes instead of ours. This had to be done this way because the first family base classes were designed for single-purpose applications, making impossible the use in the new tool. We also used this re-work as an opportunity to improve the GUI, as it will be explained later.

Note that the class diagrams are simplified, showing only the class name and the list of methods as methodName() : returnType, without their arguments. For derived classes only implemented methods are present in the diagram. Class diagrams should be read as follows: Green classes represent QT classes. The special character before the method name indicates its type, # means protected, - means private and + public. Also, bold methods are abstract methods that should be implemented by the child classes. Therefore, any class with such method is a virtual class. Slots and signals are shown along the methods with [SL] and [SI] tags respectively before the name of the slot/signal, and have no return type in the diagram (in the implementation the return type is always void).

As you can see in the diagram in figure 3.1 only some of the methods need to be reimplemented, reducing the amount of code needed for each tool and making sure that the common code remains the same through modifications. For instance, the GUI elements (dock and menu) are the same for the three tools. That also ensures that if someone has learned how to use one of the tools, the interface of the others will be familiar, reducing the learning time.

Additionally, some of the reimplemented methods make a call to the base class method and then add the new things they need (v.g. paintGL, the mouse event handlers or connectSignalsToSlots). This is another way of avoiding replicated code while adding new functionalities.

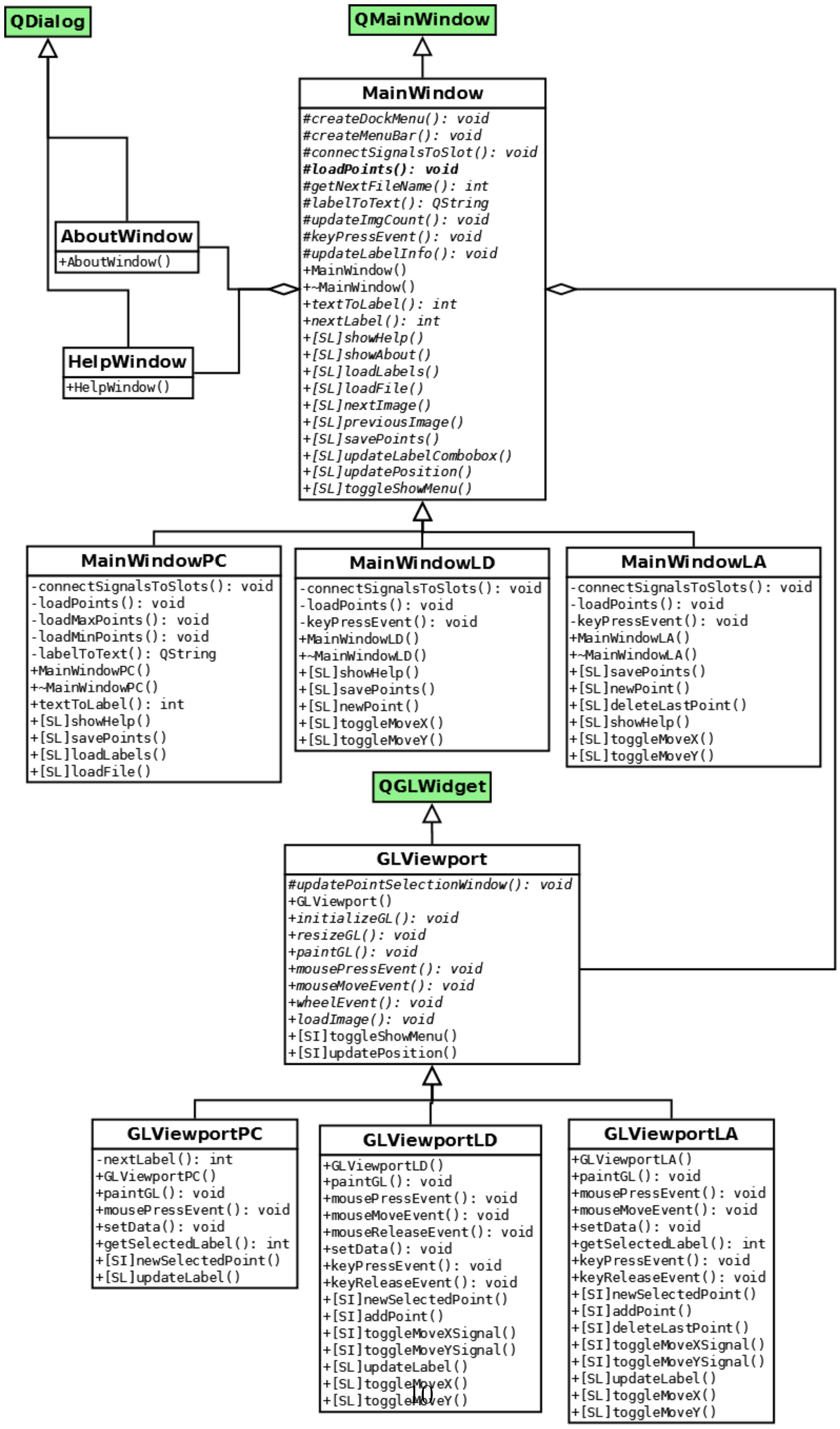


Figure 3.1: Class diagram for the first family of tools.

As for the second family (*Ground truth generation tool*, or *GT_Tool_PAGE*) you can see in figure 3.3 that its organization in classes is similar to the first family. The implementation of these classes is quite different though, having to adapt to the new requirements, and thus giving support to multiple modes and adding features. The specifics about working modes and features will be discussed in section 4.4.

Another requirement that implied big changes was the decision to use the XML PAGE format[3] for the tool, since it is being used as a common ground truth format for all the partners in the **tranScriptorium**[4] project. To solve this problem a new class was implemented to store all the PAGE information, and also to handle the reading and writing of that data. A DOM tree parser was used for the implementation, specifically the one included in the QT library (QDomDocument). In figure 3.4 you can see the diagram for the XML PAGE Document class and its associated structs.

The XML PAGE format project and the **tranScriptorium** will be further explained in sections 4.4.2 and 4.4.3 respectively. For now, we should note that all the developed tools were used in the **tranScriptorium** project, except for the point classification one that was part of a UPV project. The details of this work will be explained in section 4.3.

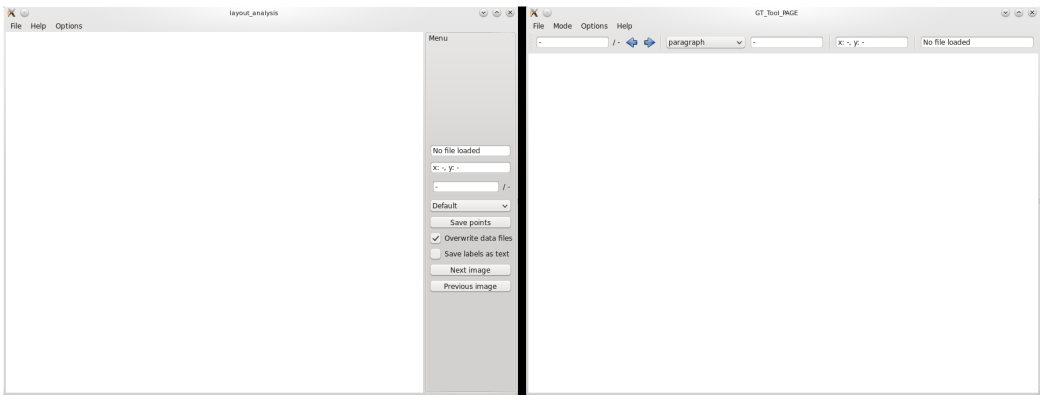


Figure 3.2: Comparison between first and second family GUI.

In the design phase of the new tool the GUI was also revised, as we have already mentioned. We decided to change the dock menu for a toolbar, giving

[3]http://schema.primaresearch.org/PAGE/gts/pagecontent/2013-07-15/pagecontent.xsd

[4]http://transcriptorium.eu/

the tool a lighter appearance and increasing the space the central OpenGL window has. In figure 3.2 you can see a comparison between the first and second family interfaces, and how the second is more compact.

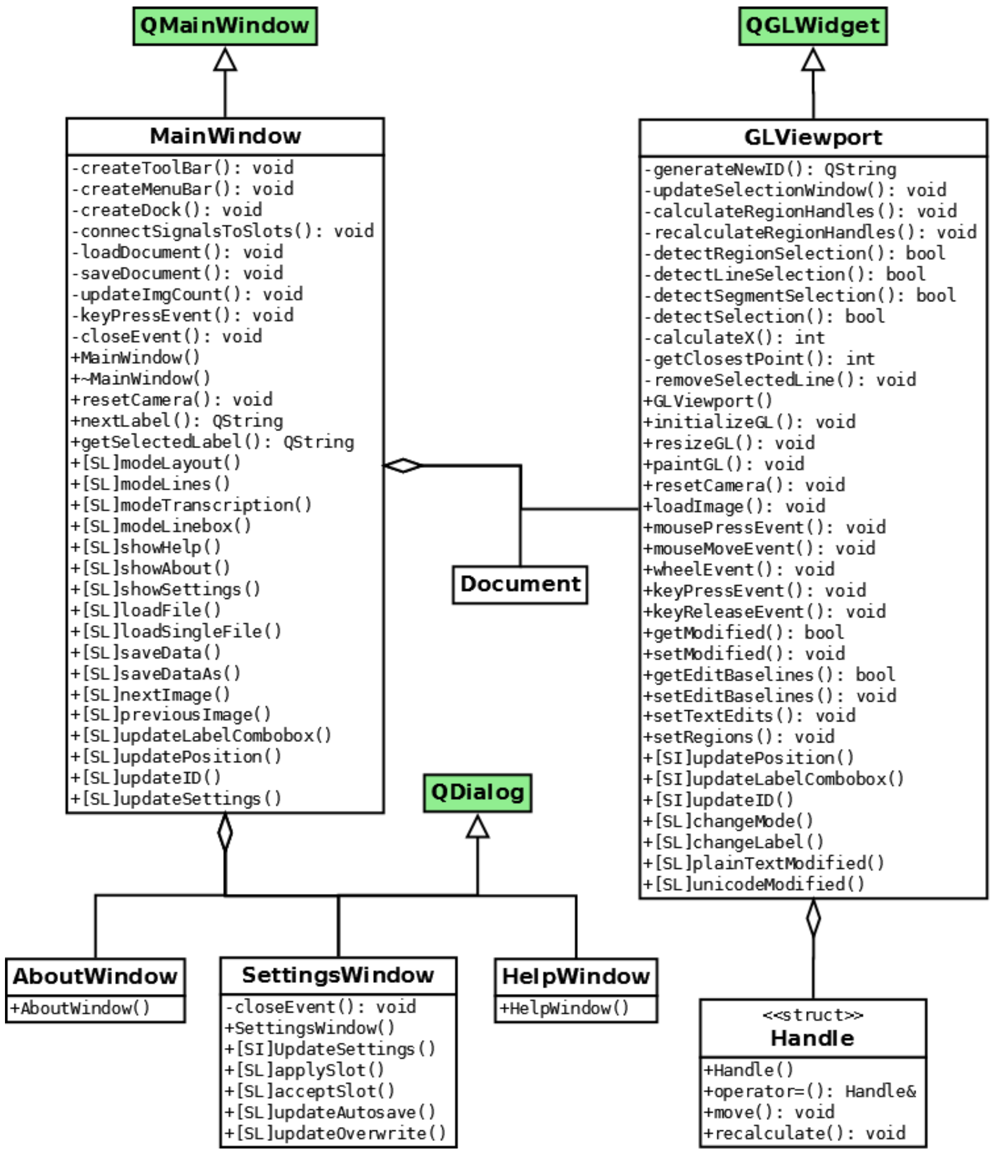


Figure 3.3: Class diagram for the second family (GT_Tool_PAGE).

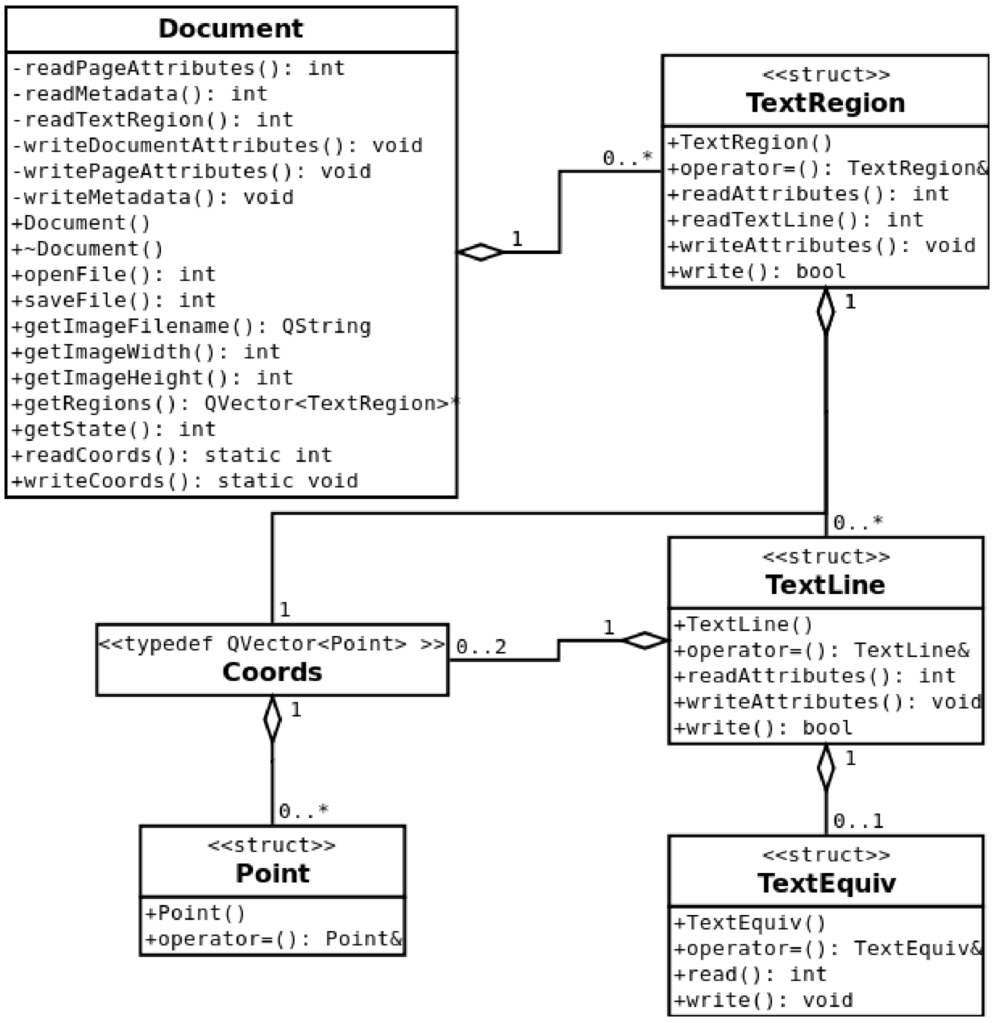


Figure 3.4: Detail of the Document class and its associated structs.

4 Implemented tools

As we have shown in the previous section all the tools have a similar design, thus having a similar layout. They all present a central OpenGL window where information is displayed and most of the interaction is performed, a menu bar with some options and actions (most of them with associated keyboard shortcuts) and either a dock menu in the first family of tools or a toolbar in the latest one. The main difference from the user point of view is the central window, since is the one that determines the task. We will review all the tools attending the order of the HTR process. That means that we will go from layout analysis to point classification for line size normalization, leaving for last the GT_Tool_PAGE, since that one is multi-purpose.

Despite the tools having some common functions and behaviours we will explain what each tool can do in a different section for the sake of clarity. By doing it this way we may repeat some information that all the tools share, but at the same time it will be possible to read the information about one specific tool and fully understand how it works without having to jump through sections.

4.1. Page layout correction tool

This tool, along with the baseline and ground truth ones, were all developed for the **tranScriptorium** European project.

When trying to recognize handwritten text, it is usual that text appears in paragraphs that are part of pages. Since we want the text segmented in lines to recognize it, first we need to work at a page level to identify and extract these lines, and then apply the HTR, line level, techniques to them. Thus, this section and the following one will describe tools that work with whole pages.

The first thing we need to do when trying to extract lines from a page is to see which parts of the image are text paragraphs and which are images or just white space, i.e. perform the layout analysis.

As this tool was developed to assist in the supervision of page layout information, in the following subsections we will explain what layout analysis is, justifying the need for our tool, and then we will show the tool itself.

4.1.1. Layout analysis

Layout analysis, also known as document structure analysis, is the task of finding the physical structure of a document page, as well as the logical structure. The physical structure consists of elements such as pages, columns, words, tables or images (and a long et cetera). It identifies what is on the page image and where, without giving information about what is the meaning of each element. On the other hand, the logical structure consists of elements such as titles, authors, abstracts, sections, etc. That is, it tries to roughly identify the content of each region of the page image.

As you can already imagine, those two kinds of structure analysis are entwined, since a paragraph can be, for instance, an abstract or part of a section, a text line can be both a title or contain the information about an author, and so on. Because of that the developed tool allows to correct or mark regions and to assign them a label that can contain both types of information.

4.1.2. Motivation

If we look at the literature on the subject of document structure analysis we can see that there are several different approaches. For instance, [Tsujimoto90] proposes the use of trees in both the physical an logical structure, trying to map the physical tree to a logical one. In [Conway93], on the other hand, the use of deterministic grammars is proposed, and in [Tateisi94] they try using stochastic grammars. And yet another approach is the use of rule-based recognition systems, like in [Lin97] and [Kim00], although those two use OCR to obtain information about the content of each part of the page and will be hardly applicable to handwritten documents.

Since it is not the subject of this project we will not go into the details of the different algorithms and approaches. It should suffice to say that none of them obtains perfect results, thus being a problem that is not completely solved by automatic means. Additionally, most of the consulted works on the topic work with printed text, which is usually better structured than handwritten text, and therefore we should expect worse results if we were to apply those methods to our documents.

If you are interested in the matter, in [Mao03] you can read a survey on numerous document structure analysis algorithms, some of which we have already mentioned. There you can see, yet again, that none of them can reach a hundred percent accurate solutions, even when attacking easy problems as printed, modern text. It is safe to assume that the problem of analyzing the layout of antique, manuscript texts is a more difficult one, and therefore we can safely state that an accurate automatic solution for the problem does not exist.

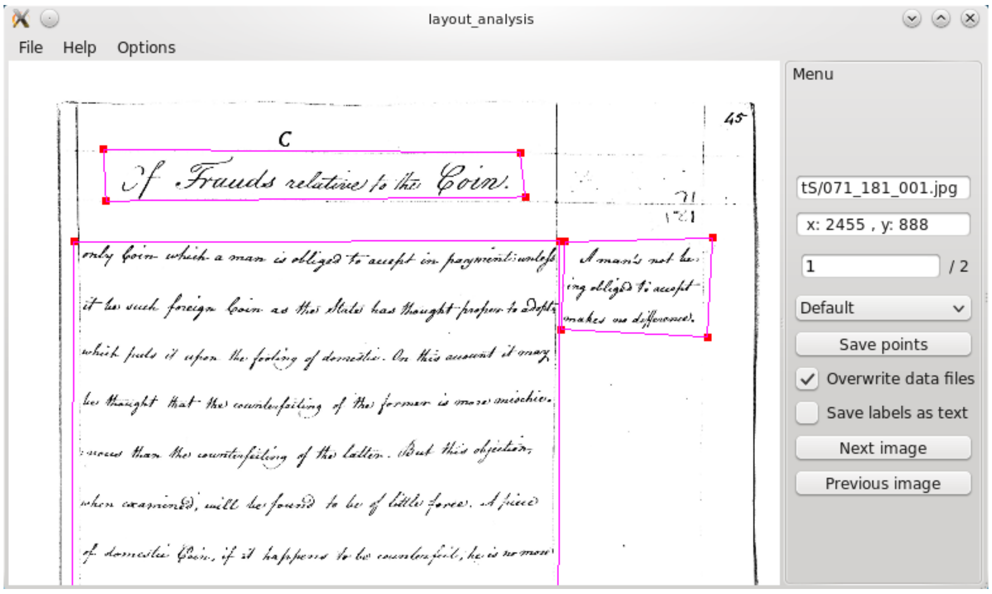
This fully justifies the need for our tool, as layout analysis is one of the first tasks that needs to be done when processing a document page, thus having a great impact on the whole process, and only through manual supervision we can reach the needed accuracy for our ground truth data.

4.1.3. Tool description

In figure 4.1 you can see how the tool looks like with an image loaded and some layout information displayed over it. We will now explain how to work with the tool, highlighting the key features.

First off, through the file menu it is possible to load a list of images, like the one that is already loaded in the image, and also a labels file. The loading of those lists could also be done when launching the tool from the command line, adding the -l list_filename and -e labels_filename options to the call.

Navigation through the list of files is possible by either using the next/previous image buttons or the right/left keyboard arrows. Labels files are used to customize the set of labels that will be used when working with the tool, allowing to easily switch the focus from physical to logical structure, or to add new elements if need be. You can also, of course, save the information currently displayed. This option should be rarely used though, since the tool has an autosave feature that allows you to go from one file to the next/previous in



tS/071_181_001.jpg

x: 2455 , y: 888

1 / 2

Default

[✓] Overwrite data files

[ ] Save labels as text

Of Frauds relative to the Coin.

only Coin which a man is obliged to accept in payment, unless it be such foreign Coin as the State has thought proper to adopt, which puts it upon the footing of domestic. On this account it may be thought that the counterfeiting of the former is more mischievous than the counterfeiting of the latter. But this objection, when examined, will be found to be of little force. A piece of domestic Coin, if it happens to be counterfeit, he is no more

A man's not being obliged to accept makes no difference.

Figure 4.1: Screenshot of the layout correction tool.

the list without having to worry about manually saving the changes every time.

The autosave feature was implemented because most of the time large sets of files were to be corrected, and it made the work faster since you always want to save the corrections when you finish editing a file. Of course, if you are using it for some other purpose or just do not like the option, you can also deactivate it. Another available option related to that is to tick or untick the overwrite checkbox. If the option is not selected the tool will ask for a new filename every time it needs to save, whether it is automatically or when the save command is selected from the file menu.

Another small save-related feature was the possibility to save the labels as text or as a code (a number). This was made this way because two different formats were used simultaneously, the textual one making visual scanning of the file easier, and the coded being more compact.

As for the main features of the tool, that is the interaction with the data, the tool has the basic zooming and panning options that any image display usually has. Two quick camera reset functions were implemented too, allowing you to go to a whole page view or to fit the page in width with a single click of the mouse wheel. We think these are really useful because manually centering the image after zooming in or moving to a specific region can take a few seconds, while using the shortcuts will take you there instantly.

The layout information is displayed on top of the page image, and the tool allows you to select the polygon defining points, being able to see their coordinates and assigned label. You can also move the points if a region is not correctly bounded. For this matter, a couple of assistance features were implemented, allowing the user to lock movement in one axis while adjusting the position of a point on the other axis, or to move a whole line of the quad instead of a single point.

Regarding labels we figured that, specially for small sets of labels, there was a quicker way to change to the right one than selecting in from the drop down list. Consequently the tool allows you to change the label to the next one in the list by clicking a point while holding the shift button, rotating back to the first one if it reaches the end of the list.

Finally, the tool would not have been complete without the possibility to add new regions in case you wanted to start from scratch, or the automatic process has missed some. By simply holding the ctrl key and clicking the place of the image where you want to add a point, new regions can be created.

We want to highlight that in a normal use of the tool, working with a large set of files, a user is able to correct any mistakes without having to move the mouse out of the display window, using the described key modifiers for label changing and then going to the next file with the keyboard arrows, auto-saving the changes. Since this was a common use scenario, we noticed that the dock menu on the right side was mainly used for setting options or when learning how to use the tool, but then rarely used by an experienced user in a usual work session. Since it was taking a considerable amount of space without adding any actual value, we decided to add an option to hide the dock menu, increasing the image workspace without losing functionality.

4.2. Baseline correction tool

As we have already said, we need to segment the text in lines in order to apply HTR techniques. Hence, it is only logical that the next step in the page preprocessing is to detect the lines inside of the regions we found in the previous step. Since there are several ways of doing this, we will explain why it is a difficult problem and give a general overview of the possible solutions, focusing then in the method used at PRHLT group. After this the need for the baselines and the correction tool should be obvious, so we will then describe the tool.

4.2.1. Line segmentation

Text line segmentation is the task of finding the lines in a document. It is usually part of the preprocessing when trying to do word spotting or text recognition, and even some methods of document structure analysis need it, specially in printed, modern text. Being an important part of the preprocessing, many techniques have been developed. Despite that, the problem is far from solved in the case of ancient, handwritten text, probably because of the difficulty of the task.

In the first place we have to consider that the format and structure of the page may differ greatly between documents, since different epochs and locations had different format rules, thus making the solutions dependant on the corpus. Additionally, we have problems that make noise really hard to remove, like faint typing or bleed through (in pages with text in both sides, text from the other side that shows in the side you are trying to recognize). This kind of problems sometimes makes impossible to filter the noise beforehand, thus making it necessary for the line extraction algorithms to handle them, removing the noise at a line level.

Another common problem that does not exist in printed text is having irregular space between lines, or even having overlapping lines (i.e. the descenders of a line going over the ascenders of the next one). Also, handwritten text is often poorly aligned, making it harder to recognize. All in all, it is a difficult and complex problem, and different methods solve better some parts of it, while struggling with some others. A survey on different methods and their application can be read in [Likforman-Sulem07] in case you are interested, since we will focus on the one used in our research group.

In [Bosch12a] and [Bosch12b] you can read about the method we use in the PRHLT group. To put it simple, the main idea is to use machine-learning, heuristic-free techniques, for the line detection. Without getting into the formal definition, the method aims at dividing the document in four types of regions, i.e. giving each detected region a label. Text is represented by two of this regions, normal text lines, which comprise the main body of the handwritten text, and inter lines, formed by the space between two normal text lines, including some ascenders and descenders. The rest of the regions can be labeled as blank lines, representing empty space at the beginning and the end of the page, and as non-text lines. This last label is used for everything that does not belong in any of the other region types. The final result of the process is usually saved as the baselines of each line, which are basically the lines that go under the main body.

4.2.2. Motivation

Despite the promising results shown by some of the methods, reaching quite low error rates, and whether they try to search directly for the baselines, for the space between lines or for some other indicator to separate the document in lines, all the line extraction techniques have one thing in common: the results are not perfect. As it happens with almost everything in pattern recognition tasks, there are always some lines that are harder to detect, or some parts of the page where the algorithm works not so well.

Since HTR can only work in lines that are properly extracted, line detection is a critical step in the preprocessing of a document, and even if the error rate is as low as a 5% it deeply affects the final results. Thus, because of the importance of having good lines, we need to have a step in the process to supervise and correct the results of the automatic line extraction. This is where our tool is needed, providing the team with a way of quickly reviewing the automatic detection, moving, adding or deleting lines when an error is detected.

4.2.3. Tool description

You can see in figure 4.2 that, as we have already said, the tool interface is identical to that of the layout analysis tool. You can also appreciate that the information displayed over the image in the central window is different, corresponding to lines instead of text regions. We will now explain the tool in detail, even if repeating some information that is the same in the previous tool, highlighting the key features.

First off, the file menu is quite the same than in the layout tool. Being so, you can load a list of images and a labels file. Of course, and also as in the previous tool, you can also load those lists when launching the tool by

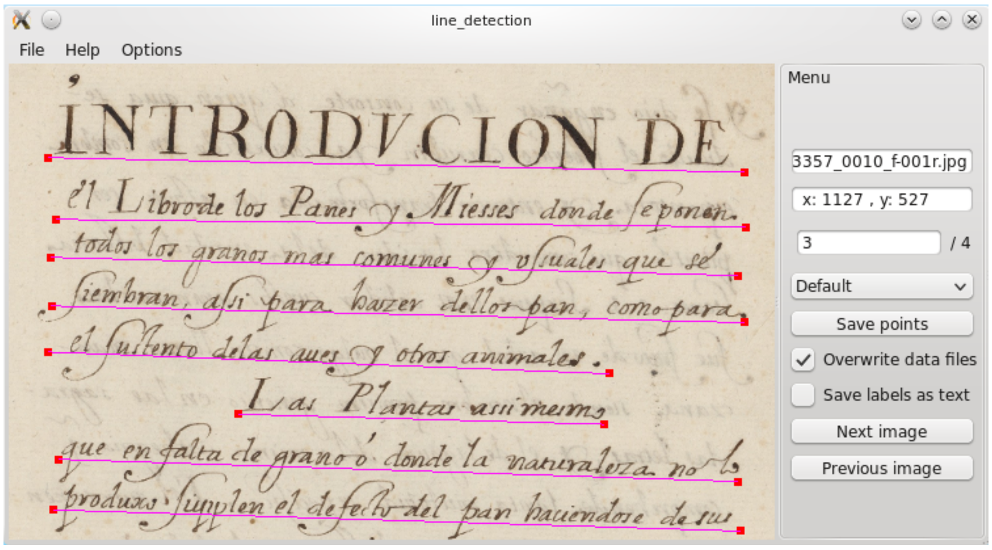


Figure 4.2: Screenshot of the baseline correction tool.

adding -l list_filename and -e labels_filename to the call.

Labels files allow you to customize the set of labels available for the lines, though you could also opt for not using any labels if you just want to work with the line position information. Through the same menu you can also save, although as said before this option should be scarcely used due to the autosave feature, since that allows you to go from file to file within the list without worrying about manually saving every time.

As one would expect the same saving options that the layout tool has remain available for this one, being able to deactivate the autosave feature and to choose not to overwrite the existing files, giving a new file name each time you want to save. The choice of saving the labels as text or as code (number) is also available.

As for the central window of the tool, this is where the interaction with the data happens, and therefore where the main differences with the other tools lie. The basic functionalities of zooming and panning the image remain the same, but the information displayed on top of the image corresponds to baselines instead of regions. As you can see in the detail view of figure 4.3 the baselines can be defined as straight, two points lines, or as polylines if you want a closer fit.

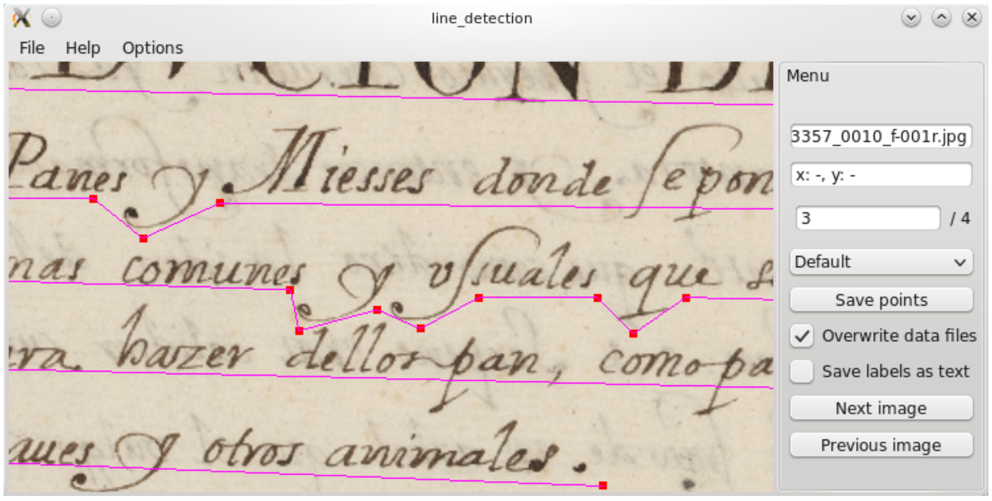


Figure 4.3: Screenshot of the baseline correction tool working with polylines.

Whether you are working with single lines or polylines, you can select single points, being able to see their coordinates and the line assigned label. You can also move points, as well as add new points to an existing lines, or even create a new line. All of this operations are done with a single click, using keyboard modifiers to indicate which one you want to perform. For instance, just clicking is used for moving a point, since that is the most common operation, while ctrl+click will add a point to the line that is the closest to where you clicked.

As in the previous tool, some assistance features were implemented in order to make it easier to add and correct lines. One of the assistance features available is, as in the layout tool, locking one of the axis of movement when changing the position of a point. Another one is the possibility to move a whole line, in case the length and inclination is correct but it is misplaced. Last but not least you can also add whole lines with a single click. Lines added through this method lack in precision, since the tool does not know the boundaries of the text and just draws two points, one at each border of the image, to define the line. However, this feature allows for really quick annotation of baselines when generating new data, giving a rough but useful approximation.

Finally, as in the layout tool, users can choose to hide the dock menu, allowing them to have more space for the interaction and visualization of the documents without losing functionalities. Of course the navigation through the image list using the arrow keys remains the same, rendering the use of the dock by experienced users unnecessary most of the time, and thus making hiding it a great feature.

4.3. Points classification for size normalization

Going forward in the top-down process of transcribing a page, after we have extracted the lines finally comes the moment of actually recognising the text in those lines. However, there is a lot of information on the lines, like remaining noise or writer-dependant features, that make the HTR process harder. In order to make it easier for the models to learn, some preprocessing of the lines is needed. This preprocessing comprises several steps, which will be explained in the following subsections, focusing in the size normalization since it is the step where the developed tool was used.

Whereas the rest of the tools were created on the more general context of the **tranScriptorium** project, this tool was created to solve a very specific problem: relevant points classification for line size normalization. Since it was where my work with the PRHLT group started and a fair amount of time was invested in this part of the project, we will broaden the scope of the explanation a bit, introducing the problem and the solution in detail first. This will serve a double purpose: on one hand, it will give an accurate idea of the tool context, and in the other hand will show some of the additional work made during the project, beside the graphical tools.

On the note of this additional work, the proposed size normalization technique was implemented. Furthermore, since the improvement of the size normalization was a stand-alone project, experimental results were obtained to assess the quality of the new implementation. All the work related to the implementation and experimentation will be presented in the following subsections, along with the experimentation results, allowing you to see how our work influenced the quality of the HTR process results.

4.3.1. Problem introduction

We wanted to implement the semi-supervised text size normalization technique published by Gorbe et al. [Gorbe08]. In the following subsections we will explain our motivations for doing that and follow with a brief overview of the line preprocessing process. After this overview we will focus on the specific part of the preprocessing that we were changing and explain the technique and our needs in order to reproduce it. Knowing those needs the purpose of the tool will be evident. We will then show the developed tool, since we think it will be better understood in its context. To finish this section we will give some experimental results and some conclusions for this part of the project.

4.3.2. Motivation

It is well known that the writing style does not help HTR systems to transcribe handwritten texts. In fact, it makes the morphological modeling more difficult, thus making the overall transcription process harder. Writing style is characterized mainly by the slant, slope, and character shape and size. Usual HTR line preprocessing leads to minimize the impact of this writing style characteristics by applying several transformations to the image in order to correct slant and slope, and then normalizing it.

In addition to the writing style problem, due to the way features are extracted from the image [Toselli03], ascenders and descenders force to include white zones (non informative) in the feature vector representation, providing useless information. Text size normalization tries to minimize these zones, reducing the empty background, while keeping ascenders and descenders. The central body size is normalized too, trying to make the system as invariant to the character size as possible.

Along this work we have implemented the semi-supervised text size normalization technique published by Gorbe et al. [Gorbe08]. The aim of this technique is to segment the text line into ascenders, descenders and central body zones by adjusting four polylines to the text profiles, segmenting the text line into these three zones and then using this segmentation to do the size normalization. (see Fig. 4.4).

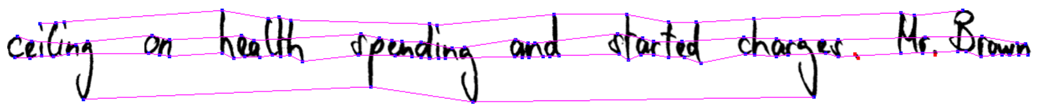


Figure 4.4: Three regions separated by polylines.

The main reason for reproducing the proposed technique was that Joan Puigcerver measured a more than five points improvement (from 45.5 to 40.0) in the word error rate (WER) by using the group's transcription software, changing only the size normalization module for the one used in the other preprocessing.

4.3.3. Handwritten Text Preprocessing

As we have said, in this part of the project we were trying to improve the size normalization phase of the handwritten text preprocessing. Nevertheless, for the sake of completeness, before explaining our specific work we are going to give a brief explanation on the whole preprocessing process.

As you may know each person has its own writing style, making the recognition process harder. Since writer dependant variations do not carry useful information when we are trying to recognize the words, the preprocessing tries to eliminate as much as possible of this variability, as well as all the noise that the image may have. In order to do that several operations are applied: noise filtering, slope removal, slant removal and finally size normalization. It is well known that a good preprocess can greatly improve the text recognition process.

It is important to note that since there is no standard solution to each step of the preprocess, neither is it the focus of our project, we will only explain what each phase does without discussing particular techniques.

Noise Filtering

The digitalization of a handwritten text, even if the original is in a good state, is not a clean process. This process often adds some parasites to the signal that have nothing to do with the information on it. Moreover, when working with historical documents it is frequent that paper degradation and the state of conservation of the document add inherent noise.

Consequently, after digitalizing an image we often see salt and pepper noise appear, as well as darkened areas. Having noise is an obvious problem, as it obfuscates the text making it harder to recognize. It is only logical that the first step in preprocessing the image needs to eliminate that noise, producing a cleaner and clearer text.
In Figure 4.5 you can see an example of text before and after noise filtering, and how the second one is much clearer.

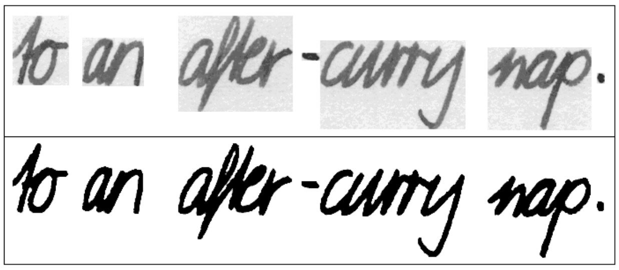


Figure 4.5: Image before and after applying noise filtering.

Note that, as we said previously, sometimes the noise is filtered at a page level. However, in this specific work we had to perform it at a line level.

Slope Removal

After filtering the noise we need to correct the angle the text forms with the horizontal axis of the image, i.e. correct the slope. Therefore, the product of this phase will be an aligned text, which is needed for slant removal and size normalization to work correctly.

Once again, a certain degree of slope removal can be performed at a page level, depending on the corpus, but in our case we had to make it at a line level. Anyways, even if corrected at a page level, slope can vary between text segments in the same line, so this step is present almost always when working with lines.

As we have said, since each word or text segment can have a different slope, even if they are in the same line, the line needs to be divided into segments separated by blanks. The slope removal algorithm is then applied to each segment. It is important to note that this division is only to perform slope removal and does not try to separate the text into words.
In Figure 4.6 you can see an example of text before and after slope removal.

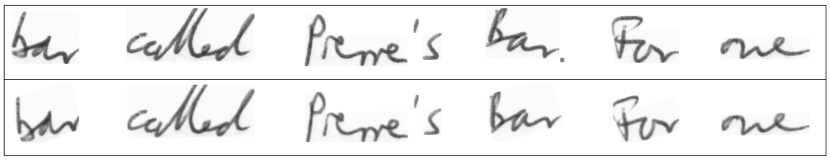

bar called Pierre's bar. For one
bar called Pierre's bar For one

Figure 4.6: Image before and after applying slope removal.

Slant Removal

The next necessary step in the text preprocessing is the slant removal. Slant is the angle between the vertical axis and the direction of the vertical text strokes. In addition to making the text less dependant on writing style, slant removal is very important because it deeply affects the local extrema detection and the size normalization process. If we normalize a non vertical stroke, specially with ascenders and descenders, it will suffer heavy deformations.
In Figure 4.7 you can see an example of text before and after slant removal.

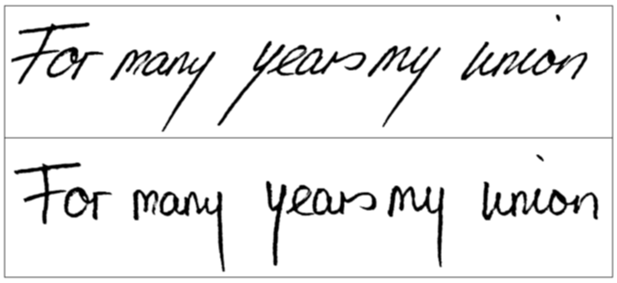


Figure 4.7: Image before and after applying slant removal.

Size Normalization

The last operation in the preprocessing is the size normalization. This step tries to make the text recognition process invariant to character size, as well as to reduce the space used by ascenders and descenders. While the previous size normalization tried to segment the text using two parallel lines for each line of text, the new polylines based approach gives a more precise fit to the text. You can see an example phrase for both methods in figure 4.8. Since our work is focused on this particular phase of the process, it will be further discussed with examples and detailed explanations in subsections 4.3.3 and 4.3.3.

Relevant Points Classification

For the segmentation and size normalization to work properly, the polylines need to be adjusted to the text contour. As we have explained, the method we were trying to reproduce is based in local extrema classification, and since the local extrema will determine the polylines used for the size

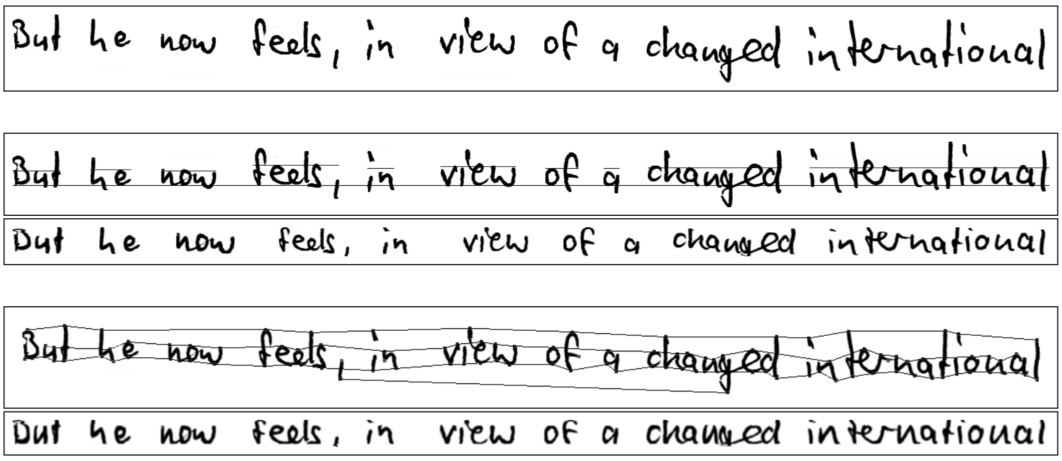

But he now feels, in view of a changed international
But he now feels, in view of a changed international
But he now feels, in view of a changed international
But he now feels, in view of a changed international
But he now feels, in view of a changed international

Figure 4.8: From top to bottom: 1. Noise cleaned image. 2. Current text (asc., body and des.) segmentation. 3. Character height normalized. 4. New text segmentation method. 5. Character height normalized.

normalization it is critical that the automatic classifier works correctly.

In order to obtain the points we automatically find local maxima and minima of the upper and lower contour respectively. This points are then classified in five classes: ascenders, descenders, upper and lower central body and others in case they do not belong to any of the groups. After point classification we have the four polylines segmenting the text into three zones: ascender, central body and descender.

Therefore, the first problem to solve is the point classification. Following the work described in [Gorbe08], we have used an Artificial Neural Network (ANN) for the local extrema classification. The input is a downsampled window around the point to classify. We take a 500x250 window around the point in the original image, and then apply a fisheye lens that expands the closer context and shrinks the farther. Then we downsample it to a 50x30 image that we use as input for the ANN (i.e. the input layer size is 1500).

The ANN topology consists of an input layer of 1500 neurons, as we have already said, two hidden layers of 64 and 16 neurons, and an output layer of size 3. We will talk more about the output in a moment. The activation function used for both the hidden layers and the output layer is the symmetric sigmoid.

The three output neurons correspond to the classification of the point as ascendant/descendant, upper/lower central body and others. An important consideration is that if the higher value of the output layer is lower than a given threshold value we consider the ANN is not sure enough about the classification, labeling the point as others.

The library used to train and classify with the ANNs was FANN[1], a free and open source library developed in C. The use of this library has limited the available choice of training algorithms and activation functions, but at the same time has allowed us to work with ANNs without spending time developing our own implementation.

We have trained the networks using the iRPROP- algorithm as described in [Igel00]. This training algorithm automatically adjusts the learning speed each iteration, both removing the difficult problem of choosing an adequate learning rate and accelerating convergence.

As a first approximation two ANNs were trained to classify the points, one for the upper contour and another one for the lower. Due to the lack of labeled points for training, a bootstrapping technique was adopted.

A set of 800 random text line images were extracted from the IAMDB corpus and divided into chunks of 200. We will give more information about the corpus later. The first subset was labeled using a heuristic classifier and corrected by a human using the developed graphical application, which we will explain later, then two ANNs were trained with this data. For the next iterations a new set of points was labeled using the ANNs trained in the previous step, and then manually corrected using the graphical tool. Two new ANNs were trained using all the available labeled data.

Some statistics on the number of needed corrections and the progression between steps of the bootstrapping process can be seen in table 4.1.

	corrections	totalPoints	% correct
Heuristic	2968	20082	14.8
1st ANN	1867	19921	9.4
2nd ANN	1872	20000	9.4
3rd ANN	1770	20097	8.8

Table 4.1: Classification error progress during the bootstrapping process

[1]http://leenissen.dk/fann/wp/

After doing some tests with the final ANNs we observed that the upper contour point classification was working much better than the lower. We then decided to make some more tests, leading to a recount of the amount of points in each class. The result of that recount can be seen in table 4.2.

	Number of points	% of the total
Ascendants	10631	13.27
Upper central body	26945	33.64
Lower central body	36982	46.17
Descendants	3758	4.69
Others	1784	2.23

Table 4.2: Number of points of each class in the training set

Since the problem of classifying upper or lower contour points was the same but mirrored, and considering that having a small amount of descendants was difficulting the learning of the ANN, we decided to try and use a single ANN for both problems. In order to do that we slightly modified the input we were going to feed to the network. For the upper contour points it remained the same, but for the lower contour the input image (after the window was extracted and the fisheye lens was applied) was turned upside down. You can see an example of this in figure 4.9

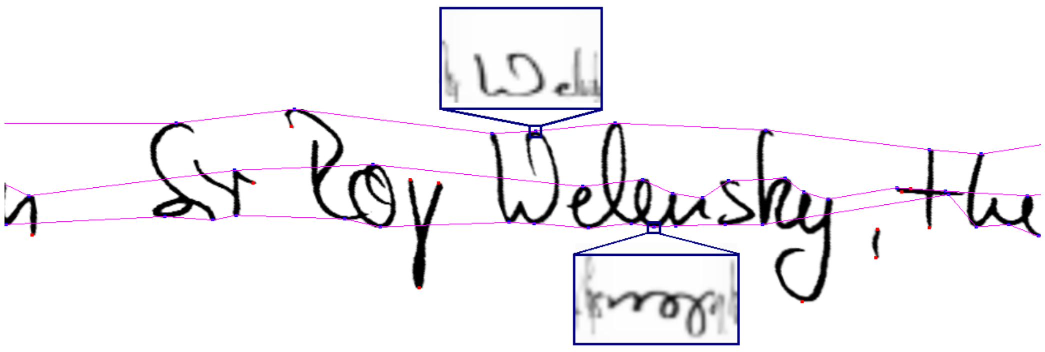


Figure 4.9: Fisheye lens (up) and fisheye lens + 180^o rotation (down).

With this new ANN we obtained better classification results, but even so we think that the point classification is probably the weakest part of the process, and has room for improvement. In this regard some work has been done using Extremely Randomized Trees (ERTs) as classifiers, but without clear results to date.

Size Normalization

Once we have obtained and classified the local extrema, the size normalization is not too complicated. Each set of points is used to define a polyline, thus dividing the image in three regions: ascenders, descenders and central body. The central body zone is then normalized to a given height without modifying the width. After some tests we decided to calculate the new central body height using the mode with context as described in [Romero06]. By doing this we attempt to minimize the changes in the image aspect ratio, consequently avoiding the text deformation. The ascenders and descenders zones are normalized too, scaling them to a height of 0.3 and 0.15 times the central body size, respectively.

It is worth noting a couple of details about the normalizing process. First, we needed the polylines to go from side to side of the image. That did not happen for most images, so we decided to project the first and the last point of every set of points into the image border.

The other important issue was the possibility of two polylines from different sets crossing each other, or getting too close. Since that should not happen and it is probably caused by the misclassification of a point, when it happens we ignore the point out of place and recalculate the polylines. By doing this we discard incorrectly classified points and avoid unnecessary deformations of the image.

Experimentation

Experiments have been conducted in order to determine if the studied method is an improvement. The HTK tool (Hidden Markov Model Toolkit) has been used for the experimentation. This tool was originally developed to be used in Speech Recognition, but nowadays it is used in Offline Handwritten Text Recognition too. The text recognition model makes use of HMMs for the morphologic level, Stochastic Finite State Automata for the lexical level and n-grams as language model.

Corpora

For the experimentation we decided to use the same corpus used in [Gorbe08], since that was the work we were using as a reference. That corpus consists of continuous sentences of English handwriting. The sentences have been handwritten from the Lancaster-Oslo/Bergen (LOB) text corpus [Johansson86] and compiled by the research group on Computer Vision and Artificial Intelligence (FKI) of the Institute of Computer Science and Applied Mathematics (IAM) from Berna. It is a well known corpus, and you can see some statistics about the partition used in table 4.3.

	Train	Validation	Test	Total
Lines	6161	920	2781	9862
Running words	53884	8718	25473	88073
Vocabulary size	7764	2425	5312	11368
Writers	283	162	57	500

Table 4.3: Statistics about the IAMDB corpus partition used for experimentation

Note that for this experiments the lines were already properly extracted, thus working only in the preprocessing and transcription of lines, without doing anything at a page level.

Language Model

A word bigram language model was trained with three different text corpora: the LOB corpus (excluding those sentences that contain lines from the test set of the IAM database), the Brown corpus, and the Wellington corpus. An open dictionary of 20000 words was used, ignoring case sensitiveness.

Results

In table 4.4 are presented several results on the same corpus, including the one with the new preprocessing. In row two and three you can see the best result to the date this project was made, with 45.5 WER, and in bold the result that served as motivation for our work, with more than a five points difference. The other rows show the result obtained by [Espana10], and the last row shows our result.

	Validation	Test
ELiRF [Espana10]	32.80	38.80
PRHLT + Prep. PRHLT	38.74	45.54
PRHLT + Prep. ELiRF	32.99	**40.08**
PRHLT + New prep.	31.78	**39.65**

Table 4.4: WER comparison

As you can see we have managed to win back the five point difference, which was our objective, and even improve a little bit. You can see nonetheless that better results already existed when this report was written, indicating that there's still room to improve and work to be done.

Conclusions

We have reproduced the technique proposed by Gorbe et al. and obtained the more than five point difference we were expecting, achieving our objective for this work. We have also observed that with the new size normalization we do not need to correct the slope because working with polylines corrects it implicitly. While correcting slope was a solved problem for large bodies of text, it was based on projections. That made short words (like 'it' or 'as') troublesome, because their projection looks almost the same no matter how you rotate them. As we have said, with the new size normalization explicit slope correction is not needed, so this should no longer be a problem.

4.3.4. Tool description

In figure 4.10 you can see how the tool looks like with an image loaded and the associated points information displayed over it. The first big difference with the other tools you may appreciate is that only a single line is loaded at a time, instead of full pages. You can also see that, in addition to the points and its label information, the tool automatically draws the polylines that result of the current classification. We implemented it this way because it makes a lot easier to identify incorrectly classified points. We will now explain how to work with the tool, highlighting the key features.

First off, as in the other tools, through the file menu it is possible to load a list of images, like the one that is already loaded in the screenshot. You can

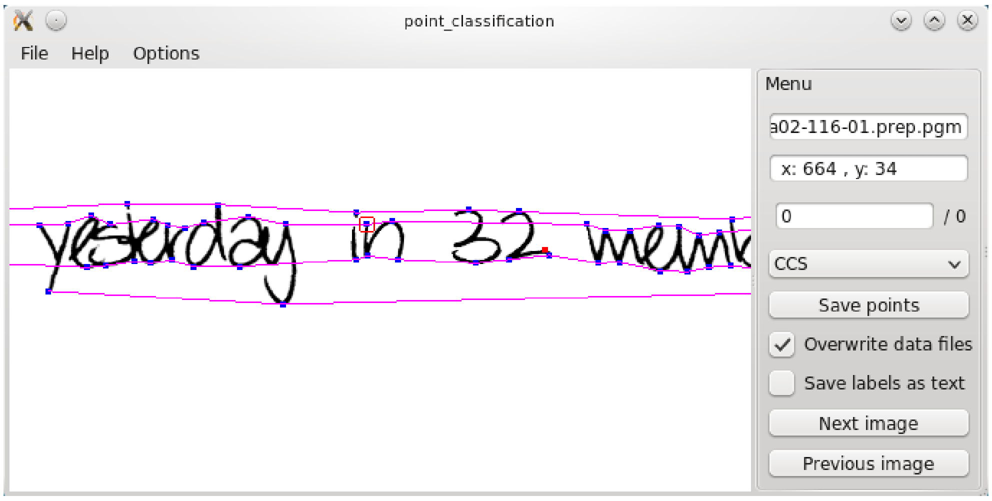


Figure 4.10: Screenshot of the point classification correction tool.

also load the list when launching the tool from the command line by adding -l list_filename to the call. Label classes were fixed for this tool, since it was designed for a very specific task, instead of being loaded from a file. The available labels corresponded to the point classification previously described. As explained in the other tools, you can also save the information currently displayed, but the tool is designed to save changes automatically when going through the list, without having to manually save anything.

The autosave feature was much necessary because, as explained in section 4.3.3, chunks of 200 lines were corrected each time, and you always wanted to correct the few points that had a wrong label and then save and continue with the next line. However unlikely, if the tool was to be used for some other purpose, the autosave option can be deactivated. Another available option related to that is to tick or untick the overwrite checkbox. Although you usually want to overwrite to save time, because otherwise you will need to introduce a new name every time you go to a different image, it is possible to disable the overwriting.

Another small save-related feature was the possibility to save the labels as text or as a code (a number). Usually the code representation was used, specially because the ANNs needed this format in order to work, but labels could be saved as text for easy visual inspection of the files, if need be.

As for the main window features, this tool presents a simpler interaction mode, due to the fact that it was designed for the sole purpose of point classification correction, or what is the same, checking and changing the labels assigned to points. First of all, the tool has the basic zooming and panning options that any image display usually has. A camera reset on mouse wheel click is available too, although for correcting labels you work mostly without zoom, and since the image of a line is much bigger in width than in height, when you zoom in you probably want to move the image from left to right as you read. This makes the camera reset feature less critical than when working at a page level, when you may zoom to a specific paragraph to see some details and then want to quickly go back to full page, but it is still a good addition to have.

As we said, in figure 4.10 you can see the tool with a line loaded. The display shows the line image and the associated points information on top, along with the polylines the current point classification would generate. By selecting a point you can see its label and coordinates. Being able to see the point coordinates was really important when trying to detect problems in the new size normalization process, since we needed that information to be able to obtain, for instance, the fisheye window that served as input for the ANNs.

In the same way as the previously explained tools, this tool allows the user to cicle through the labels by holding shift and clicking on a point. Since there are only five different labels, and also because the expected polylines update in real time when changing labels, it is really quick and easy to spot and correct any misclassified point. Of course you can also use the drop down list if you are new to the tool, want to refresh what labels are available, or in which order. However, for regular use the quick way should be used.

Finally, as in the other tools, you can choose to hide the dock menu to have more display space. This is specially useful in this tool since, as we have said, the line images are usually way wider than higher, and once you have learned how to use the tool the dock is not needed most of the time.

4.4. Ground truth generation tool

As we previously stated, the ground truth generation tool is a multi-purpose tool. At first it was designed as the combination of the layout and the baseline tools, but afterwards a transcription mode was added, allowing to visualize and edit the transcription associated to lines and paragraphs. As we have already said, it was developed for the **tranScriptorium** project, about which we will give more information in section 4.4.3

Since we have already justified the need of tools for layout and line supervision, in this section we will focus on explaining why a multi-purpose one was needed and what differences the tool had with the previous ones. We will also talk about the format used for storing the data, since it was an important factor when designing the tool, and give some detailed information about the project for which the software was developed. After that we will describe the tool in detail, even if some things are similar to the other described tools and may seem redundant. As explained before, we do it this way because we want each section to be complete, avoiding having to jump to the other sections.

4.4.1. Motivation

Being in a research environment means continuously evolving requirements for the developed software, and sometimes those new requirements call for a new tool instead of a patch over an old one. This was the case when, after the PRHLT group started working with the layout and baseline tools, we found that sometimes you needed to correct lines taking in consideration the region in which they were contained. Since we could not display that information in the existing tools as they were, and no easy workaround was found, we were inclined towards developing a new tool.

At the same time, a common format to share ground truth information between the partners of the **tranScriptorium** project was agreed, the XML PAGE. We will explain the format in detail in the next subsection, but the fact is that the introduction of this new format would have required a rework of a big part of the tools. Therefore, considering this and the aforementioned need of displaying layout and line information simultaneously, it was decided that a new, more general tool, needed to be developed.

Note that this tool was created to substitute both the layout and the baseline correction ones, unifying the generation and revision of page level ground truth information into a single tool, instead of having to do it as a two-step process.

4.4.2. The XML PAGE format

As we explained previously, the XML PAGE format[2] was agreed amongst the partners in the **tranScriptorium** to be used as a common format to share the generated ground truth information. This has the obvious advantage of being able to share and utilize information without having to do format conversion, but it also has some disadvantages. In this section we will explain briefly the format, and what implications its use had for us.

First off, the format is well defined through an XML schema, which is available online on the link provided as footnote. As you can see there, the format gives support to a lot of the information that you can detect within a document, ranging from regions (paragraphs, but also image regions, glyphs and the like) to bounding boxes of words, including of course the bounding boxes of lines and the associated baselines. You can also add transcriptions to the different elements, from single words to whole regions. Additionaly, each one of those elements can have associated information such as language or script. Finally, you can also save information at a page level, for instance the reading order or the relations between elements.

As you can see, it gives a general enough framework to hold most of the information that can be generated as ground truth for layout analysis, line extraction or even transcription. However, having the possibility to have the same information at several levels (v.g. transcription of a region and the transcriptions of the lines inside that region) can cause some problems with coherence and has to be handled carefully, specially when combining ground truth information from different sources. In PRHLT group we worked mostly at a line level.

Regarding the tool, following a strict format means that you know exactly what you can expect of a file and what your implementation has to support, but in our case also meant that some elements our implementation needed (v.g. the baselines) were not present in the definition. That was not in the current version but in the 2010 one, which was the only one that existed when we started working with the format. We solved the problem by using a custom format that was PAGE with some modifications, thus having to implement format conversion tools, and our needs were added in the next version of the format.

[2]http://schema.primaresearch.org/PAGE/gts/pagecontent/2013-07-15/pagecontent.xsd

Although a little out of the scope of this report, it is worth noting that due to my work with XML PAGE to develop the tool I was the person in charge of all the format-related problems during the time I worked with the PRHLT group, having to implement several side-tools for format conversion and insertion of transcriptions into an XML with the detected baselines of a document, as well of acting as a consultant in PAGE related decisions.

4.4.3. The tranScriptorium european project

We have mentioned the **tranScriptorium**[3] project several times, and since this tool was developed exclusively for such project, we think this will be the right place to give some details about the tool context, thus giving some information about **tranScriptorium**. Quoting the official project website:

> ”**tranScriptorium** is a STREP of the Seventh Framework Programme in the ICT for Learning and Access to Cultural Resources challenge. **tranScriptorium** is planned to last from 1 January 2013 to 31 December 2015.
>
> **tranScriptorium** aims to develop innovative, efficient and cost-effective solutions for the indexing, search and full transcription of historical handwritten document images, using modern, holistic Handwritten Text Recognition (HTR) technology.”

The partners of the project are:

- Universitat Politècnica de València - UPV (Spain)
- University of Innsbruck - UIBK (Austria)
- National Center for Scientific Research "Demokritos" - NCSR (Greece)
- University College of London - UCL (UK)
- Institute for Dutch Lexicology - INL (Netherlands)
- University of London Computer Centre - ULCC (UK)

[3]http://transcriptorium.eu/

With the UPV having the management role, as well as working in different parts of the project. A detailed list of the specific tasks each partner develops could be found, when published, under the deliverables tag of the website.

Aside from the management tasks in which I did not take part, during my time working in the project we focused mainly in layout analysis and line extraction, and lately on transcription of the detected lines, making the tool a fundamental part in several steps of the workflow.

4.4.4. Tool description

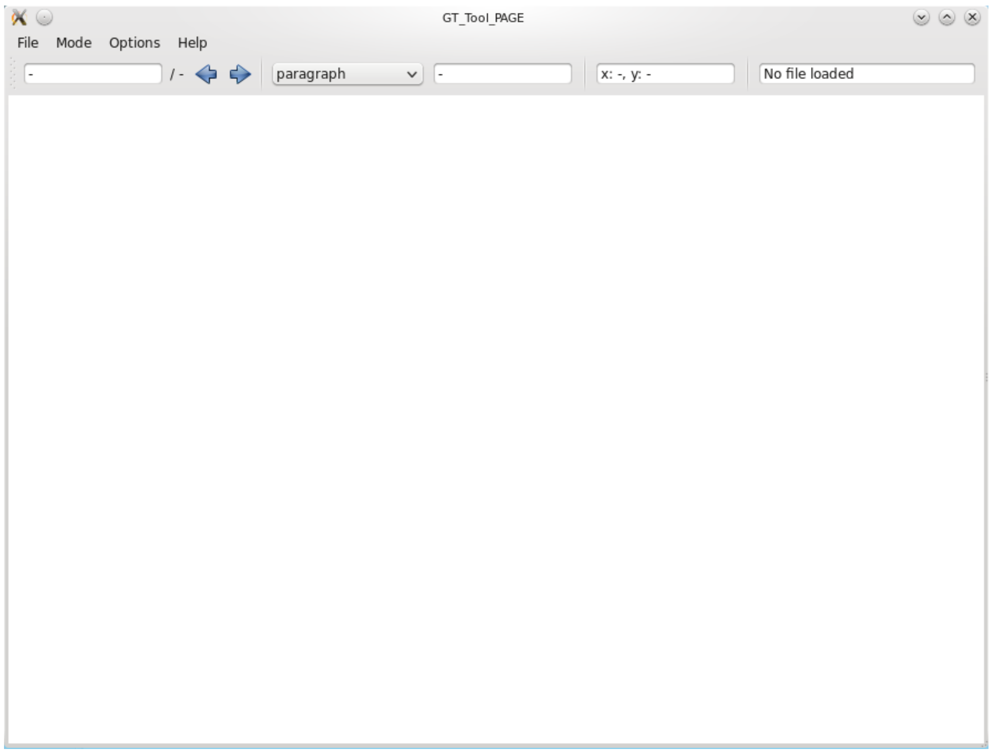


Figure 4.11: Screenshot of the GT_Tool without any image loaded.

In figure 4.11 you can see a screenshot of the tool. Since this is a multi-purpose tool we will explain first the menus and general features, and then have a subsection for each work mode. Although some of the modes are similar to previously described tools, we will explain each one in detail so this section is self-contained.

First off, we have a menu similar to the one in the single-purpose tools, only that adding a mode option to select in which mode we are going to work. The change between modes can also be performed through keyboard shortcuts. As one could expect, the file menu allows you to load either a single file or a list of files, and also to save (or save as) the file in which you are working. You can also, as with the previous tools, launch the tool with a single file or a list loaded, by adding either the filename or -l list_filename respectively. There are no labels files in this tool, since the list of available labels is defined in the XML PAGE schema, thus being fixed.

In addition to supporting the new format and displaying the relations between lines and layout, we used the opportunity to redesign the user interface a bit, improving the usability and the look of the tool. The dock menu has been changed for a toolbar that displays the same information and has the same functionality, but in a more compact way. Additionally, the configuration options that were present in the old dock (autosave and overwrite) have been moved to the options menu. Through this menu you can access a settings window where the different configuration options can be configured. Also, the settings window has been designed in a way that allows for easy extension, being able to add and manage new options easily.

The general features are similar to the ones in the previous tools, being able to navigate the list of files using the keyboard arrows, autosaving on the change of file if you want and asking for new file names if configured that way. As for the central window where the image and the associated information are displayed, it maintains the usual zooming and panning utilities, along with the quick camera resets to fit the image in width or height. The rest of the interaction options are dependant on the active mode, and will be detailed in the following subsections.

Note that, as explained in section 3.2, not only graphical improvements were made. The tool internal design was also changed to fit the new requirements, and it was built in a way that allowed for creating other modes if need be. This is the case of the transcription utility, that was not required nor planned in the first version of the tool, yet it was added later, when the need arose in the project, without any trouble.

Despite the changes, we have tried to maintain the main principles of the tool, those being ease to use and learn, and the expert user being able to work with mouse and keyboard shortcuts without having to leave the main window. A detailed list with the specifics of those shortcuts can be found in the tool user manual that accompanies the report as an annex.

Region mode

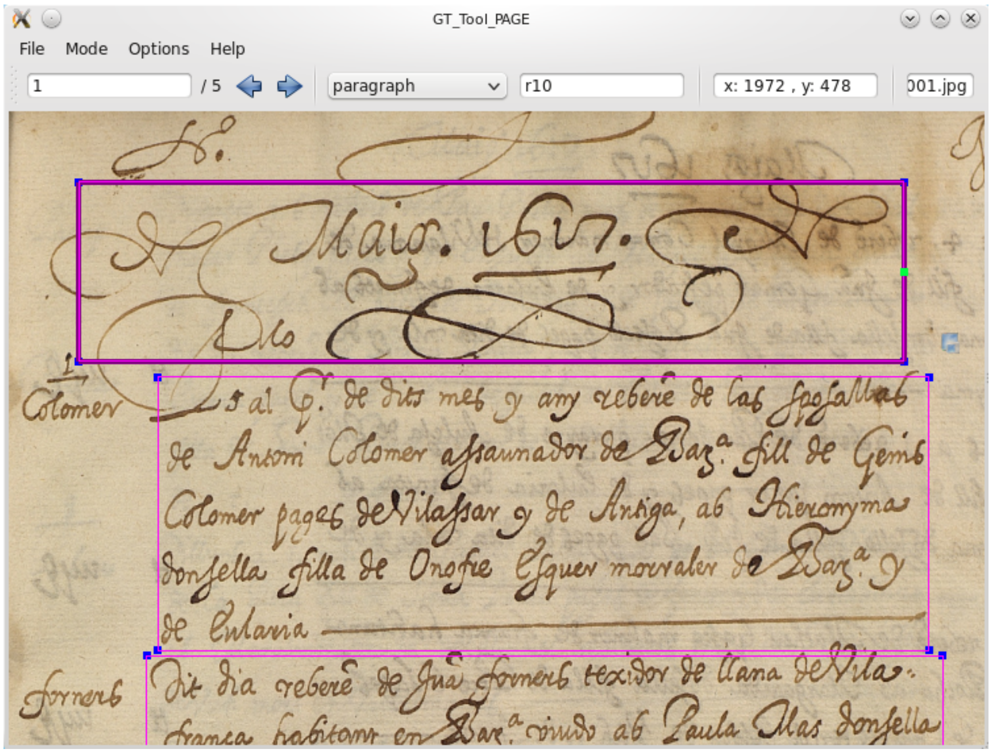


Figure 4.12: Screenshot of the GT_Tool in region mode.

In figure 4.12 you can see a screenshot of the tool with a file loaded and set in region mode. The aim of the region mode is to produce or supervise layout analysis ground truth information. Since the problem of document structure analysis was explained in section 4.1.1, and the motivation for having this mode remains the same, we will focus on explaining the functionality.

As seen in the figure in this mode you have regions defined usually by four points, although there can be regions defined by any number of points. In any case, you can select and drag points by using the mouse, thus redefining the regions, and also add points to create new regions. On this note, while the tool supports loading files with regions defined by an arbitrary number of points, you can only create new regions with four points. Of course you can also change the label of the selected region using shortcuts instead of the drop-down list, and delete regions you do not want.

In addition to the described functionalities, that are pretty much the same as the layout analysis tool had, some improvements were made. The first one was changing the "locking in one axis" system, that felt unintuitive and awkward to most users. This system was mostly used to try and align points after moving them, so as instead handles were implemented. Handles are fictitious points that appear at the middle of each line when mousing over, as you can see in figure 4.12 (the green point is the handle for that line). By selecting and moving the handle the user is able to move the whole line parallel to where it was, thus avoiding the need to manually lock the other axis before moving, and also the need to realign the points afterwards.

Another addition that was key in greatly reducing the required time for adding missing regions or generating layout information from zero was the possibility of adding a rectangular region. By clicking on one point while holding the required keys, you start a rectangle. Then you can move the mouse, previsualizing the rectangle you would generate, and click again to add it to the page.

Line bounding box mode

In figure 4.13 you can see the same file from the previous section, but with the tool set in line bounding box mode. This mode does not correspond to any of the previous tools, and that is because we usually do not work with bounding boxes but with baselines. However, the PRHLT group needed to correct bounding boxes provided by another partner of the tranScriptorium project, so this mode was added.

It is worth noting that the bounding box of a line and the baseline can be associated to the same text line, that is saying, more or less, that the baseline is inside of the bounding box. That is why, as you can see, the baselines are also shown in this mode, and the baseline associated to the selected box is highlighted too. Also, because of the mentioned relation, most of the operations that affect a whole baseline will also affect the associated bounding box and vice versa. Consequently, some of the possible operations and shortcuts available for baselines were added to the bounding box mode for the sake of coherence. All in all, the two modes are quite similar, being the only difference the kind of elements you can add, i.e. if you can add points

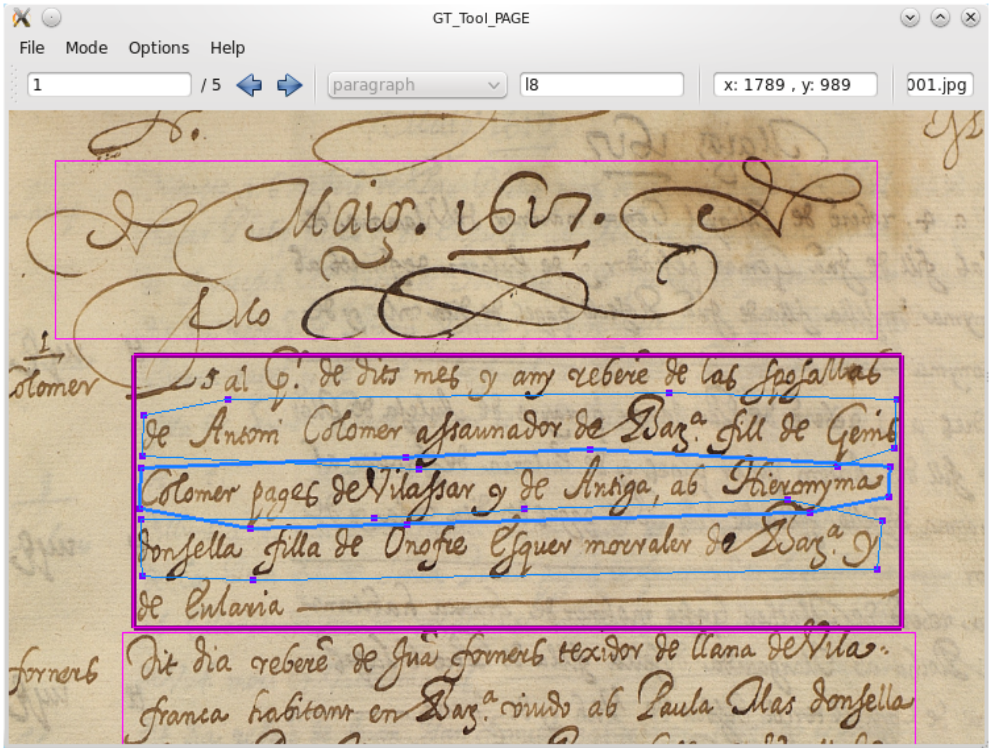


Figure 4.13: Screenshot of the GT_Tool in line bounding box mode.

to bounding boxes or to baselines.

Specifically, in this mode the user can select and move points to redefine bounding boxes, as well as delete existing points or add new points, either to a existing box or to define a new one. You can also delete whole bounding boxes, along with the associated baseline information, if any. Lastly, you can select the next line by pressing the spacebar, and move the selected line up and down using the keyboard arrows. This two features were used mostly in baseline mode and do not have much use in this mode, but as we said we decided to include them just in case.

Baseline mode

In figure 4.14 you can see the same file set in baseline mode. As you can appreciate this mode is, as already stated, almost the same as the line bounding box mode. This mode corresponds to the baseline tool, with the addition of the bounding boxes, but some utility options where added too.

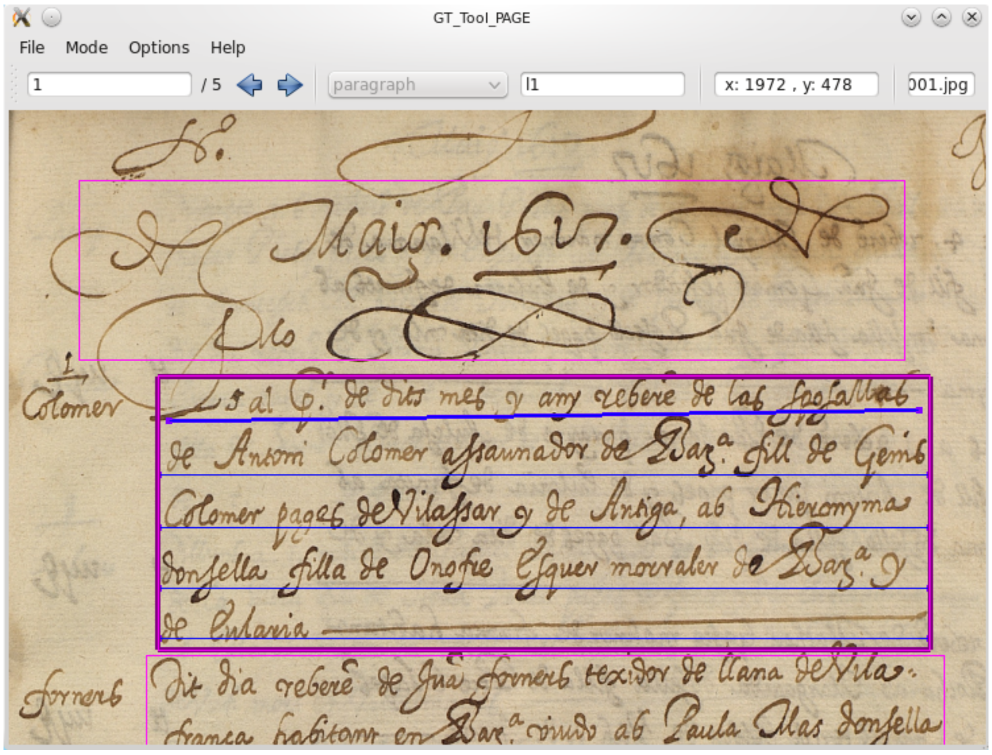
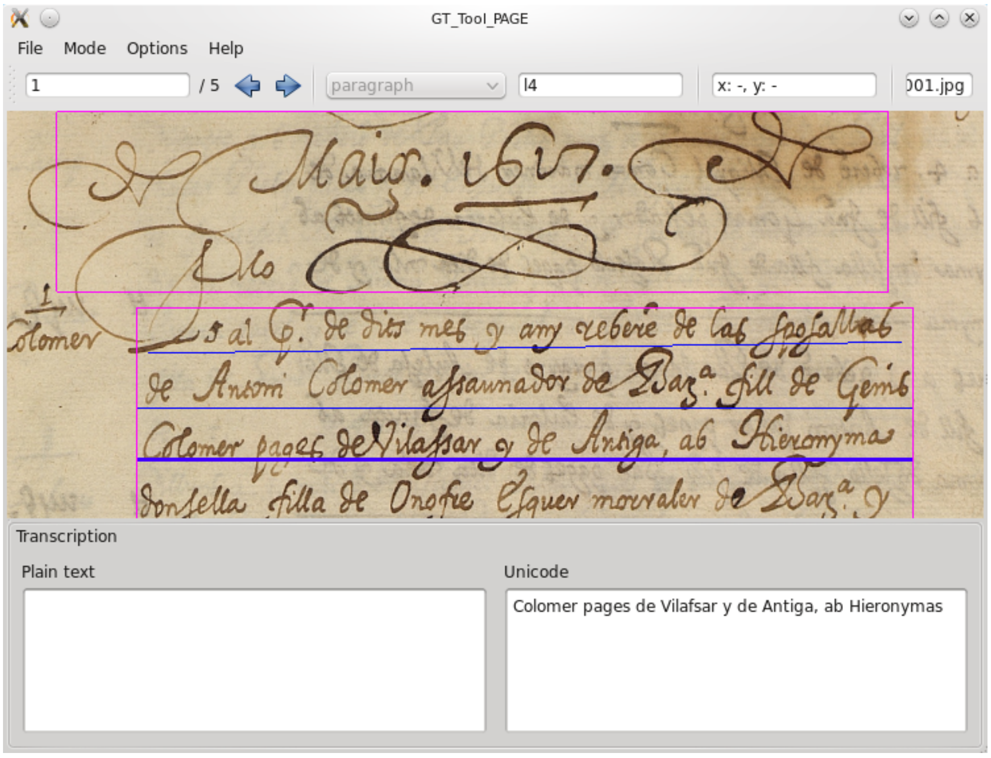


Figure 4.14: Screenshot of the GT_Tool in baseline mode.

In this mode you can do the same operations that in the previous one, only adding or deleting points from baselines instead of bounding boxes. Additionally, you can move a whole line freely using a keyboard shortcut and dragging with the mouse, allowing to freely reposition it without losing the relative position of the points. Also, the possibility of adding whole lines that was present on the baseline tool was implemented, since it allows for a rough but quick approximation when working with pages that have no ground truth information. However, this approximation was improved by making the two points that form the line adjust to the region in which is being added, instead of to the whole page. This is one of the advantages of having a multi-purpose tool and being able to use the information about the relations between elements.

Although it was mentioned in the bounding box mode description we want to highlight here the two features we said that were mostly used in baseline mode, that is, being able to reposition lines and move from one to another using only the keyboard. This options allowed to quickly improve the correction time in instances where you only want to check if the lines are in place and adjust them to fit the text line closely, which is not an uncommon problem when doing automatic line detection.

Transcription mode



Colomer pages de Vilafsar y de Antiga, ab Hieronymas

Figure 4.15: Screenshot of the GT_Tool in transcription mode.

In figure 4.15 you can see the same file with the tool set in transcription mode. As you can see this mode allows the user to see all the information about the page, that is, all the regions and lines. However, you can only interact with the different elements to see and edit the transcription.

This mode was the last to be implemented, after the first version of the tool was in use for a while. The project had the need to transcribe pages, as well as to check the results of automatic transcription, and since it makes sense associating each transcription to a line instead of doing whole pages in a text editor, it was decided to add the feature to the tool.

There are two main differences between this mode and the others. First off, the one already mentioned, the possibility to interact with elements of both layout and line level, although in a more limited fashion than in the corresponding modes. The second difference introduced was the creation of the transcription dock, which you can see in the screenshot, on the lower part of the screen. After some thought and some tests, we decided that this was the better place to situate it, since it allows for a whole width view of the text, and can be set close enough to the line you are transcribing so you do not have to shift your view much during work.

We would like to highlight a couple of things about transcription mode. In the first place you can see that the transcription dock is divided in two text fields. This was done that way because the XML PAGE has two transcriptions associated to each element, one as plain text and another one as unicode, and this way we gave support to both options. The second important thing, also to support more features of the XML PAGE, is that even when we usually wanted to do transcription at a line level, you could also transcribe whole regions if you want to do so.

5 Conclusions

We have developed several tools, covering each step of the ground truth generation process for page level analysis. We have also, with the multi-purpose tool, contributed an integrated tool for the **tranScriptorium** project, allowing to visualize and correct the data in all the phases of the ground truth generation in a single tool. The tool has been used extensively and is a key element in the current page processing workflow of the project, proving that it has great utility and is well fitted to its purpose.

The design of the tool was also put to the test when transcription mode had to be added after the tool was finished and in use. This has shown that the modular design we chose is flexible and allows for further growth without much difficulty or time investment, which we think is a key feature in modern software development.

Additionally, as explained in section 4.3, we have implemented a new tool for the size normalization step of the line level preprocessing, and also performed the required experimentation to prove that is an actual improvement over the method in use up to this moment.

As more of a personal conclusions, I would say that the problem of developing well integrated, easy to use GUIs is far from solved in general, and needs to be approached with an specific analysis depending on the problem and the target users. However, as explained in chapter 3, I think that there are several principles and guidelines that hold true for most cases. In any case, this project has allowed me to learn how to analyze the user needs and try to develop graphical tools to solve those needs. At the same time I have been able to work in collaboration with other people and experience what is like to be integrated in a research group.

As for the future of the tools, being in a research environment it is nearly impossible to have a finished product, and even in other environments, whenever software is used in real applications it is usual that it gets updated and improved along the way. In our case, several additions and improvements have already been made to the tools during the time of the project. However, it is likely that some modifications will be needed in the future, for instance if the PAGE format gets updated, as it happened during 2013, leaving the old 2010 version outdated. It is also possible that more features of the format that are not supported currently are needed, and this will require to modify the multi-purpose tool.

As we have explained the tool is designed in a way that should make easy the addition of new modes or the modification of the existing ones, as well as the modification of the data structures as long as the interface between elements remains similar. Since new versions of the format should build on the previous ones, this should allow for an easy adaptation of the tool in the future. Therefore, we think that if the PRHLT group wants to keep using the tool it should not be too difficult for someone to maintain and update it when need be.

6 Acknowledgments

This project has been partially supported by EU 7th FPtranScriptorium project (Ref: 600707) and UPV Primeros Proyectos (PAID-06-11).

Thanks to Moisés Pastor i Gadea and Joan Andreu Sánchez Peiró for trusting me to work with them, and to the whole group for making me feel part of the team.

To my family and friends, for supporting me no matter what I decided to do, and specially to Pepe, Juaneu and Sergi for listening to my blabbering when I was trying to figure out something difficult. Also, Pepe deserves a special mention for helping me learn GUI design with QT at the start, when I was utterly clueless, probably saving me quite some time.

Finally, to my grandfather Rafael, who always kept his curiosity and his illusion for learning new things, inspiring me and helping shape who I am today.

Bibliography

[Bosch12a] Bosch, Vicente, Alejandro Héctor Toselli, and Enrique Vidal. "Statistical Text Line Analysis in Handwritten Documents." Frontiers in Handwriting Recognition (ICFHR), 2012 International Conference on. IEEE, 2012.

[Bosch12b] Bosch, Vicente, Alejandro Héctor Toselli, and Enrique Vidal. "Natural language inspired approach for handwritten text line detection in legacy documents." Proceedings of the 6th Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities. Association for Computational Linguistics, 2012.

[Conway93] Conway, Alan. "Page grammars and page parsing. a syntactic approach to document layout recognition." Document Analysis and Recognition, 1993., Proceedings of the Second International Conference on. IEEE, 1993.

[Espana10] S.Espana-Boquera, M.J.Castro-Bleda, J.Gorbe-Moya, F.Zamora-Martínez. "Improving Offline Handwritten Text Recognition with Hybrid HMM/ANN Models". IEEE Transactions on Pattern Analysis and Machine Intelligence. vol. 99. 2010.

[Gorbe08] J.Gorbe-Moya, S.España, F.Zamora and M.J.Castro, "Handwritten Text Normalization by using Local Extrema Classification", "PRIS'08", pp. 164-172, 2008.

[Igel00] C. Igel, M. Husken, "Improving the Rprop learning algorithm." In Proceedings of the second international ICSC symposium on neural computation, pp. 115-121, 2000.

[Johansson86] S.Johansson, E.Atwell, R.Garside, and G.Leech, "The Tagged LOB Corpus: User's Manual", Norwegian Computing Centre for the Humanities, Bergen, Norway, 1986.

[Kim00] Kim, Jongwoo, Daniel X. Le, and George R. Thoma. "Automated labeling in document images." Photonics West 2001-Electronic Imaging. International Society for Optics and Photonics, 2000.

[Likforman-Sulem07] Likforman-Sulem, Laurence, Abderrazak Zahour, and Bruno Taconet. "Text line segmentation of historical documents: a survey." International Journal of Document Analysis and Recognition (IJDAR) 9.2-4 (2007): 123-138.

[Lin97] Lin, Chun Chen, Yosihiro Niwa, and Seinosuke Narita. "Logical structure analysis of book document images using contents information." Document Analysis and Recognition, 1997., Proceedings of the Fourth International Conference on. Vol. 2. IEEE, 1997.

[Mao03] Mao, Song, Azriel Rosenfeld, and Tapas Kanungo. "Document structure analysis algorithms: a literature survey." Electronic Imaging 2003. International Society for Optics and Photonics, 2003.

[Romero06] V.Romero, M.Pastor, A.H.Toselli and E.Vidal, “Criteria for handwritten off-line text size normalization”, Procc. of The Sixth IASTED international Conference on Visualization, Imaging, and Image Processing (VIIP 06)”,2006.

[Tateisi94] Tateisi, Yuka, and Nohuyasu Itoh. "Using stochastic syntactic analysis for extracting a logical structure from a document image." Pattern Recognition, 1994. Vol. 2-Conference B: Computer Vision & Image Processing., Proceedings of the 12th IAPR International. Conference on. Vol. 2. IEEE, 1994.

[Toselli03] A.Toselli, A.Juan, D.Keysers, J.González, I.Salvador, H.Ney, E.Vidal and F.Casacuberta “Integrated Handwriting Recognition and Interpretation using Finite-State Models.” International Journal of Pattern Recognition and Artificial Intelligence, 2003.

[Tsujimoto90] Tsujimoto, Suichi, and Haruo Asada. "Understanding multi-articled documents." Pattern Recognition, 1990. Proceedings., 10th International Conference on. Vol. 1. IEEE, 1990.

Appendix: Ground truth supervision tool user manual

Use Manual
Ground truth supervision tool

Jorge Martínez Vargas
jormarv5@fiv.upv.es

October 7, 2013

1 Introduction

The tool has been designed to allow the user to quickly supervise and correct both region and line detection information, aswell as reading and editing the associated transcriptions. This manual assumes you have already compiled the tool. Nevertheless, you can find the compilation instructions in the annex at the end of the document.
The following sections explain in detail how to work with the tool. The file format used for the region and line information is XML PAGE [1]. Note that while the tool does fully support the XML PAGE format, meaning that if you open and save a file all elements will be preserved, a lot of the features of the XML PAGE format are not accessible through the tool.

2 GUI description

This section will cover what each element of the user interface is for. It will aswell describe all the menu entries.
In figure 1 you can see the user interface. The following parts could be identified:

1. **Central window:** The central window is an openGL display where you can see the region/lines information. The interaction with this window will be explained later.

2. **Menu bar:** The menu bar contains four sub-menus, left to right:

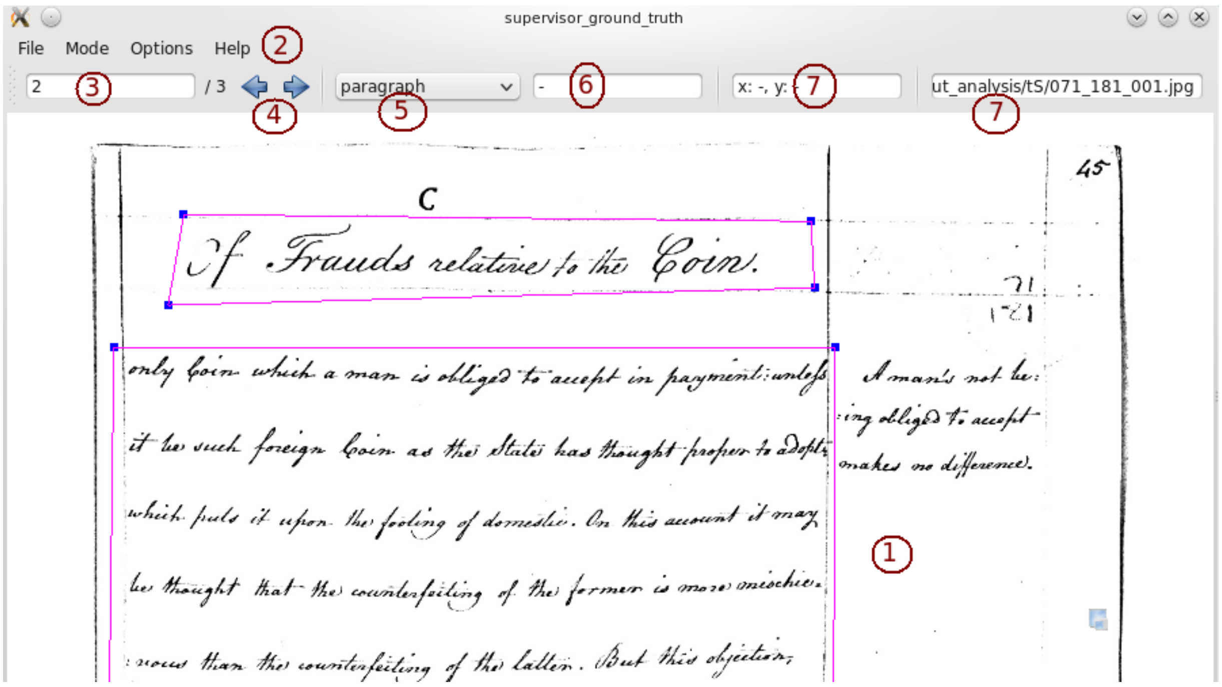
   2.1 **File:** Open either a single file or a list of files. Save/save as the changes made to the loaded data.

   2.2 **Mode:** Set the tool to region/line supervision mode, or transcription mode.

   2.3 **Options:** Opens the Settings window. We will talk about the available settings later.

   2.4 **Help:** Display the program help (refers to this manual) and about.

[1] http://schema.primaresearch.org/PAGE/gts/pagecontent/2013-07-15_draft/pagecontent.xsd



C

Of Frauds relative to the Coin.

45

only Coin which a man is obliged to accept in payment: unless
it be such foreign Coin as the State has thought proper to adopt;
which puts it upon the footing of domestic. On this account it may
be thought that the counterfeiting of the former is more mischie-
-vous than the counterfeiting of the latter. But this objection;

A man's not be-
-ing obliged to accept
makes no difference.

Figure 1: GUI general view.

3. **File counter:** Shows the current image number and the total amount of images in the loaded list.
   You can change the number and press Enter to go to a different image.

4. **Navigation arrows:** Pressing the left/right arrow you can go to the previous/next image in the list.

5. **Label selection:** Shows the selected region label. You can also click and select another one to change the label.

6. **Selected item ID:** Shows the selected region/line ID.

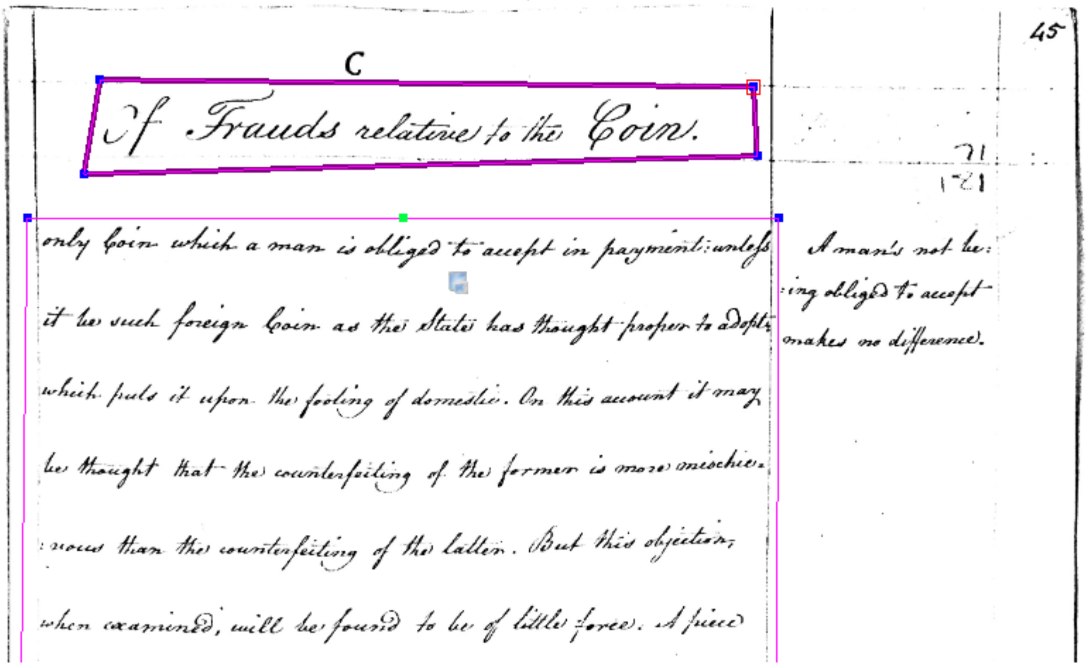
7. **Position:** Displays the selected item position.

8. **Image name:** Shows the loaded image name, as it appears in the XML file.

In addition to the graphical menu interaction, you can also open a file/list file when launching the tool, using the following commands:
GT_Tool file.xml opens the tool with file.xml loaded.
GT_Tool -l listfile opens the tool with the listfile loaded and the first xml file in the list opened.

2.1 Central window: Region mode

At figure 2 you can see the central window in region mode. For every region of text it shows the four points that define it in blue, and the lines between them in purple. The selected


C

Of Frauds relative to the Coin.

only Coin which a man is obliged to accept in payment: unless it be such foreign Coin as the State has thought proper to adopt, which puts it upon the footing of domestic. On this account it may be thought that the counterfeiting of the former is more mischie-:vous than the counterfeiting of the latter. But this objection, when examined, will be found to be of little force. A piece

A man's not be-:ing obliged to accept makes no difference.

45

71

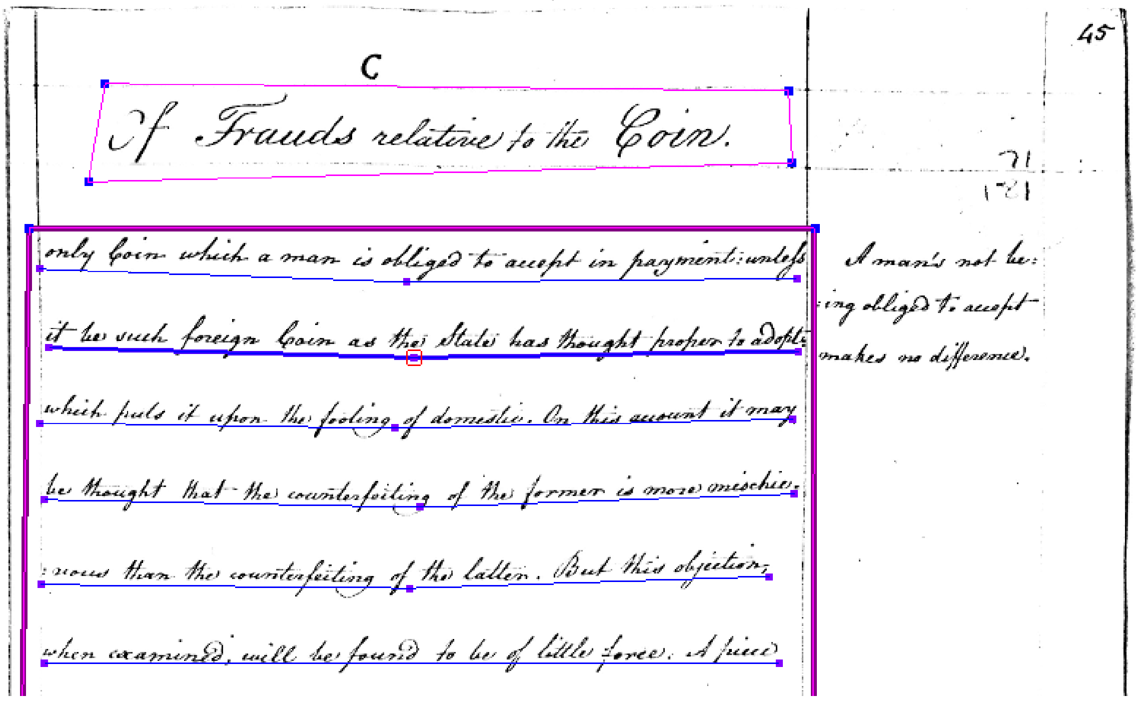
Figure 2: Central window in region mode.

point is indicated by a red square around it, and the selected region is highlighted in a darker tone that the rest of the regions.
In the figure you can also see a green point in the middle of one of the lines. That green point is a handle, and every line has one, but they're shown only when hovering over with the mouse.
It is important to note that while you can open files that contain regions defined by more than four points, you can only define new regions with four points each.
The following operations are possible while in this mode:

- Move the camera focus: right click and drag with the mouse.
- Zoom in/out: mouse wheel.
- Reset the camera to fit image width: mouse wheel click.
- Reset the camera to fit image height: Ctrl+mouse wheel click.
- Select a region: left click the region (one of the lines that forms it).
- Select a point: left click the point.
- Move a point: left click the point and drag with the mouse.
- Move a line in parallel: left click the line handle and drag with the mouse.
- Change the selected region label: Shift+left click.

- Delete the selected region: Del.
- Add a point: Ctrl+left click. When you have added four new points, they will form a region.
- Add a rectangular region: Ctrl+Shift+left click. Hold Ctrl+Shift and drag the mouse to see the preview of the region, then left click again to add the new region.

2.2 Central window: Baseline mode



45

C

Of Frauds relative to the Coin.

only Coin which a man is obliged to accept in payment: unless
it be such foreign Coin as the State has thought proper to adopt:
which puts it upon the footing of domestic. On this account it may
be thought that the counterfeiting of the former is more mischie:
:vous than the counterfeiting of the latter. But this objection,
when examined, will be found to be of little force. A piece

A man's not be:
:ing obliged to accept
makes no difference.

Figure 3: Central window in baseline mode.

At figure 3 you can see the central window in baseline mode. For every block of text detected in layout mode it shows the lines in purple. The selected layout when the mode changed is highlighted, and only the lines inside it are displayed. The baselines are shown in blue, and the selected line is highlighted in dark blue. Note that the baselines are polylines, defined by a set of points.
In addition to the baselines the bounding boxes of each line can be present too. Those will appear in a different blue, and have the same interaction possibilities as the baselines (except for the addition of new elements).
The following operations are possible while in this mode:

- Move the camera focus: right click and drag with the mouse.
- Zoom in/out: mouse wheel.

- Reset the camera to fit image width: mouse wheel click.
- Reset the camera to fit image height: Ctrl+mouse wheel click.
- Select a point: left click the point.
- Select a line: left click the line (or alternatively a point of the line).
- Move a point: left click the point and drag it with the mouse.
- Move a whole line: Shift + left click a point of the line and drag it with the mouse.
- Delete the selected point: Del.
- Delete the selected line: Shift + Del. Note that this will delete both the baseline and the bounding box if they are associated to the same TextLine element.
- Add a point: Ctrl+left click. The point will be added to the closest line.
- Add a new line: Ctrl+Shift+left click. A single point, belonging to a new line, will be added.
- Add a new horizontal line: Ctrl+Shift+right click. Two points will be added, creating a line from side to side of the region.
- Move the selected line up/down: Up/Down arrow key.
- Select the next line: Spacebar.

2.3 Central window: Line bounding box mode

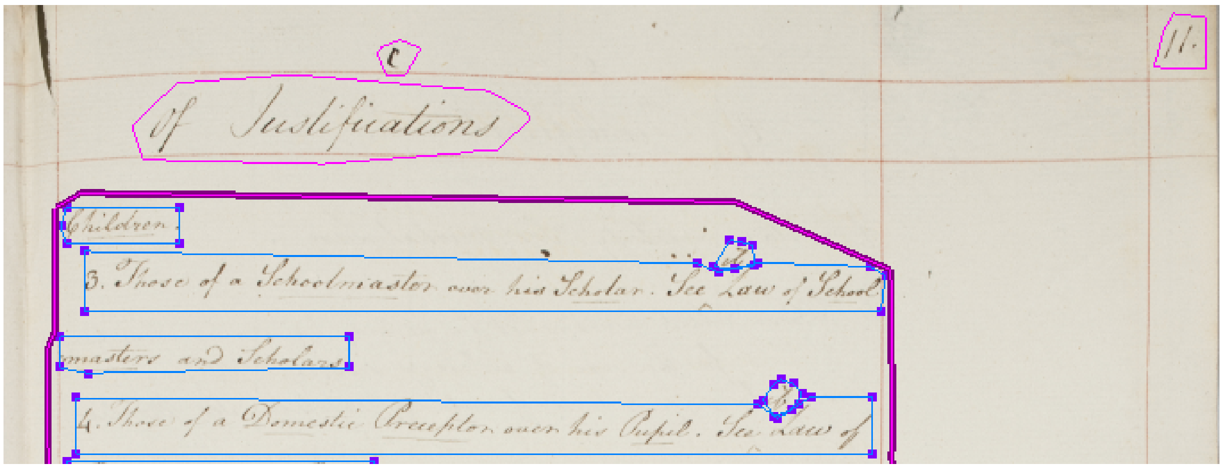

11.
Of Justifications
Children
3. Those of a Schoolmaster over his Scholar. See Law of School
masters and Scholars
4. Those of a Domestic Preceptor over his Pupil. See Law of

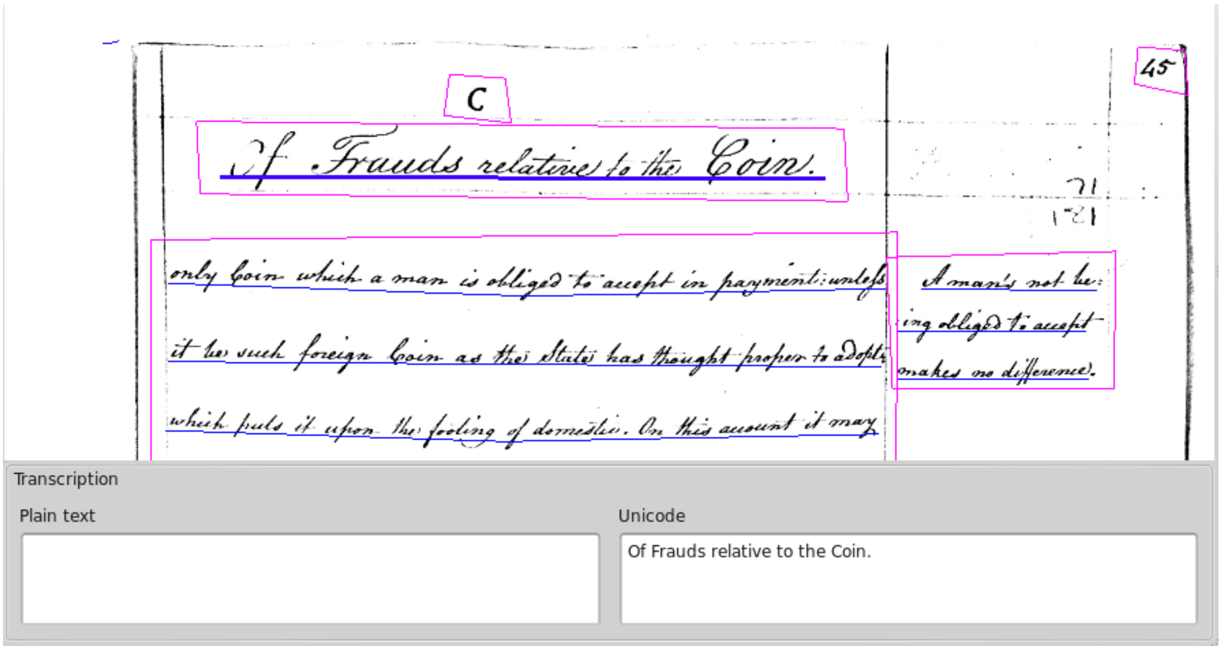
Figure 4: Central window in line bounding box mode.

At figure 4 you can see the central window in line bounding box mode. As it has been already said, this mode has the same options that baseline mode except for the addition of new elements. In this mode you can add new bounding boxes using the following operations:

- Add a point: Ctrl+left click. The point will be added to the closest box.
- Add a new box: Ctrl+Shift+left click. A single point, belonging to a new line, will be added.

We recommend adding the points of each box in order.

2.4 Central window: Transcription mode


C

45

Of Frauds relative to the Coin.

only Coin which a man is obliged to accept in payment: unless
it be such foreign Coin as the State has thought proper to adopt
which puts it upon the footing of domestic. On this account it may

A man's not be-
ing obliged to accept
makes no difference.

Transcription

Plain text

Unicode

Of Frauds relative to the Coin.

Figure 5: Central window in transcription mode.

At figure 5 you can see the central window in transcription mode. The first important thing to note is that a new dock item appears in the lower part of the screen, containing two text boxes. This is where the transcription information will appear.
We still have the central window too. For every block of text detected in region mode it shows the lines in purple. The selected region (if any) is highlighted in a similar way as it is in region and lines modes. All the lines detected in the image are shown in blue, and when selected a line is highlighted the same way it was in lines mode. Note that the lines are polylines, defined by a set of points, but we do not show the points because they give no relevant information in this mode.
The following operations are possible while in this mode:

- Move the camera focus: right click and drag with the mouse.
- Zoom in/out: mouse wheel.
- Reset the camera to fit image width: mouse wheel click.

- Reset the camera to fit image height: Ctrl+mouse wheel click.
- Select a region: left click the region (one of the lines that forms it).
- Select a line: left click the line.
- Edit the transcription of a selected item: Write in the corresponding text box.

3 Keyboard shortcuts

Some of the operations that can be done through menus can also be called with a keyboard shorcut. Here is the list of available shortcuts:

- **F1:** Switch to layout mode.
- **F2:** Switch to baseline mode.
- **F3:** Switch to transcription mode.
- **F4:** Switch to line bounding box mode.
- **Ctrl+S:** Save current files.
- **Ctrl+Shift+S:** Save current files as. A saving window will pop to ask for a new file name.
- **Left arrow:** Go to previous image.
- **Right arrow:** Go to next image.

4 Settings

This last section will describe the options you can configure through the Settings window, as well as list their default values.

- **Autosave** before loading a new image/closing the program. *Default value:* Disabled. If autosave is disabled, the progam will ask before discarding changes.
- **Overwrite XML file** when saving. *Default value:* Enabled.

A How to compile the tool

First of all, in order to compile the tool, aswell as for the tool to work, you will need QT4.x[2] installed in your computer.
Once you have installed the QT library, you need to open a terminal and go to the folder where you have the source for the tool, then execute the script MakeGT_Tool_PAGE (you may need to do chmod +x MakeGT_Tool_PAGE first). If the compiling script was not included with the code you need to follow this steps:

1. Write *qmake -project*. A qt project file will be generated (.pro extension).
2. Open the .pro file that was just created, add the line *QT+=opengl xml* and save the file.
3. Write *qmake* on the console. A makefile will be generated.
4. Write *make* on the console. A binary with the same name as the folder should be generated, that's the tool.

[2]Any of the QT4 versions should do, http://qt-project.org/downloads and scroll down to the required version. The tool was developed and tested using qt 4.7