Automatic Cropping of Tagged Area

Project Report

Javier Turek
Supervisor: Ron Rubinstein

Geometric Image Processing Laboratory
Department of Computer Science
Technion, Israel

October 2005

Table of contents

Abstract

When photocopying or scanning a page, the machine works in full-page mode, though often, only a specific area of the page is wanted. A proposed solution suggests marking the desired area by two labels, thus allowing the machine to automatically identify it, and crop the page accordingly. In this project we present a convolution-based algorithm that detects these two labels, and crops the desired area. The entire process is fully automatic and requires no user intervention. In this paper we also describe a possible label design, which we used for testing the algorithm. We implemented the algorithm in C++, with a simple GUI, and have applied it successfully to several examples. We have found the algorithm to be highly accurate and stable.

1. Introduction

The photocopying and scanning processes don't enable the users to select what they want to copy from a page. An idea is introduced to automatically crop and photocopy a marked area. The user can select an axis-aligned rectangular area using two labels to mark the upper-leftmost vertex and the lower-rightmost vertex. Figure 1 shows an example of two labels in a page marking the wanted area. Figure 2 shows the desired result.

Figure 1: scanned image with labels.

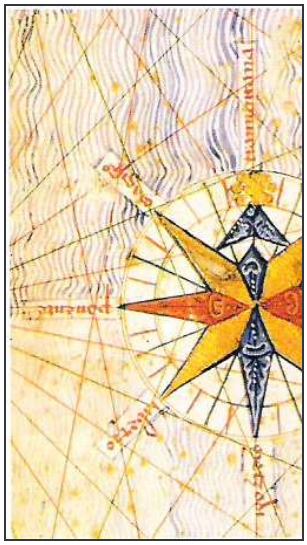

Figure 2: resulting cropped image.

In this project, we implement a shape matching algorithm based on a correlation method and an edge comparison method. The combination of these two methods, results in a fast and stable algorithm. Due to the fact that all the work is done on gray-scale images, color images can also be accepted as input with only a simple conversion.

2. Algorithm outline

The algorithm is divided into six main stages (see figure 3):

1. Image preprocessing: in order to work with a “normalized” image, the input image is preprocessed to produce a gray-scale, 100dpi image.
2. Correlation (or shape matching): once we have the “normalized” image, we calculate its correlation with the label’s image as kernel. The extreme correlation values give us an idea where the labels may be found.
3. Thresholding: the correlation values are thresholded to select candidates for the labels’ positions. Note that the best correlation values will not necessarily indicate the labels' positions (discussed later).
4. Cluster elimination: in general, near-by pixels will get similar values in the correlation stage; hence, the thresholded values will commonly appear in clusters. This step eliminates these clusters by leaving only the best candidate from each group.
5. Edge correlation: in this stage, the true positions of the two labels are identified from among the list of remaining candidates. To distinguish the true positions from other highly-correlated positions, we perform a shape-based comparison, which is done by correlating the edge maps of the label and image. The positions that achieve the best results from this correlation are selected as the label positions.
6. Cropping: once the label locations are known, the desired area is extracted from the source image, and returned at output.

3. Algorithm details

3.1 Preprocessing

In the first stage the input image is preprocessed to leave only the information needed for the next stage, producing to a “normalized” image. Due to the possibility of accepting color images, we begin with a color to gray-scale conversion (if needed). We then shrink the image to 100dpi resolution for faster calculations.

We selected the 100dpi resolution due to the label’s size. The lower this resolution, the faster the calculation will be. However, the possibility of using lower resolutions is limited by the degradation of the significant features in the labels during the down-sampling. For example, the “plus” sign seen in figure 4 could be easily degraded to mid-level grays in the low resolution image. Thus, a lower resolution may be possible with a different label than the one shown here.

Finally, intensity stretching is applied to enhance the image’s dynamic range. Assuming the intensity levels of the image are originally in the range [a,b], where $[a,b] \subset [0,1]$, we apply the transform $y=(x-a)/(b-a)$ so the intensity values fill the entire [0,1] range. The contrast enhancement provides us with a wider range of correlated values in the following stage. We end this stage with a gray-scale image of 100 dpi resolution and intensity stretched.

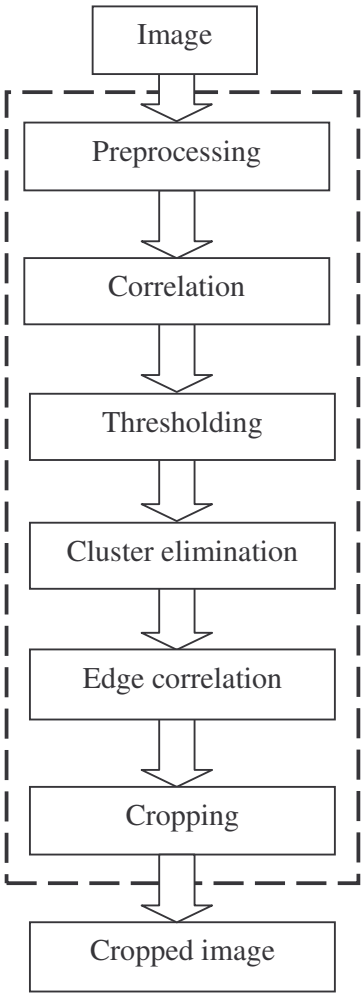

Figure 3: algorithm flow.

3.2 Correlation

In this part we correlate the image with the label's kernel, in order to get a range of significant values indicating the possible positions of the label. For this purpose, we apply the linear transform Y=2X-1 to the intensity values, thus taking them from the [0,1] range to the [-1,1] range. After applying the transform to the pixel values of both the kernel and scanned image, they are convolved.

If the correlation kernel has a total of K pixels, then the values in the resulting correlation matrix will oscillate from –K to K. The best correlated positions will have the most positive values, and will indicate the possible positions of the label. Furthermore, if the second label is defined as the inverse of its pair, then the worst correlated values will designate its position as well. This way both labels can be detected simultaneously, leading to a significant runtime saving, since the convolution is the most time-consuming part of the algorithm. Before continuing to the next stage, the correlation values are linearly scaled to the normalized range of [0,1], using the transform Z=(X+K)/(2K).

3.3 Thresholding

In this stage, we take the normalized correlation results and generate a list of candidate positions. Empirical results show that the true label position achieves

a correlation value in the range [0.7,1], while its inverse achieves a correlation value in the range [0,0.3]; these values take into account a maximum rotation tolerance of about 20 degrees. In this stage we select a group of candidates instead of a single value for each label, because due to quality aspects of the label and the scanner itself (for instance washed-out colors, noise, rotation etc), the best and worst values may not necessarily indicate the true labels' positions (see edge correlation stage for more information). Therefore we do not discard the rest of the candidates in these ranges; instead, we threshold the correlation values according to these ranges, obtaining two lists, one for each label.

3.4 Cluster elimination

We found that many of the thresholded candidate positions were neighbors, leading to groups of candidates in the same area. This phenomenon occurs because around every position that correlates very well to the pattern, several points near it will correlate well too, so it is enough to take one representative from each such area. In our case, we will prefer the best candidate as representative. Hence, we implemented a cluster-elimination step which leaves the best candidates in each area.

First, we sort the candidates by their correlation values. We order those in the range [0,0.3] in ascendant order, and those in the range [0.7,1] in descendant order. Then, for each label separately, we iterate through its ordered candidates, and for each one check if there exists another, better correlated candidate within a fixed radius about it. If so, the candidate is removed from the list. The process continues until all candidates in the list have been checked. The radius of the circular area has been selected as 30% the length of the label's shorter edge. Finally, we get a short list of candidates for each label.

3.5 Edge correlation

The motivation for this stage is that we cannot rely only on the highest correlation values to determine the exact positions of the labels. This is mainly because low-quality scanning may result in washed-out blacks, which appear as grays in the scanned image; in these cases, other positions with similar gray-scale distributions but with very different shapes may get better correlation values than the true label, if they contain darker grays than the label itself (see an example in figures 4 and 5).

Figure 4: bad quality label.

Figure 5: best correlated position.

For this reason, once the two final lists of candidates are established, we need a different method of comparing the candidates to the label. We therefore use edge correlation, which relies on more distinct characteristics of the labels.

First, the edge map of the label is calculated, to be used as a correlation kernel; we used the Sobel edge detector for this. However, the problem with edge correlation is that it is very unstable, and a slight rotation of the scanned label will lead to a low correlation between its edge map and the kernel edge map. To resolve this problem, we apply a low-pass filter to the kernel edge map before the correlation. The parameters of the low-pass filter were tuned through

testing: size=3x3, and σ=0.6. After the low-pass filter is applied, the blurred edge-map is thresholded, such that its pixels are mapped to binary black and white values; we map those above 0.2 to 1, and the remaining ones to 0. The entire process leads to a "fattening" of the edges in the kernel edge map, thus transforming it to a more stable correlation kernel. The choice of 0.2 as the threshold value has been determined through trial-and-error. Higher threshold values lead to thinner edges, which return us to the instability problems discussed above; lower threshold values make the edges thicker – hence stabilizing the process – however this also increases the chances of high correlation with false positions.

Once the correlation kernel is determined, we iterate all the remaining candidate positions, and for each one we extract from the normalized image what should (presumably) be the scanned label. The edge map of this segment is calculated and then correlated with the kernel. Once all candidate positions have been processed in this way, the candidate showing the best correlation is selected as the true location of the label – provided that its correlation value exceeds some minimum value X. Setting this minimum value X ensures that if no label exists in the input image (up to some maximal inclination), then the method will not return any result at all. Also, by altering the value of X we can control the amount of inclination that the method will accept for the label – higher values correspond to less tolerance to inclination. As explained in the experimental results section, we used X=100 pixels, which leads to an approximate inclination tolerance of 10 degrees.

3.6 Cropping

After the positions of both labels have been determined, we are finally prepared to crop the image. But first, if the source image originally had a resolution higher than 100dpi, we must translate the cropping margins, given in 100dpi accuracy, to their values in the original resolution. The translation is done by multiplying the 100dpi positions by the downsampling factor, and adding half the factor to the result. This calculation maps each position in the 100dpi image to the center of the rectangular area that corresponds to it in the original image. The translation adds an error of half the shrinking factor to the cropping process. Finally, the image is cropped.

4. Label design

In the previous sections, we mentioned some properties that the labels should assume. In this section we discuss these properties, and describe their effect on the algorithm performance.

1. Binary values: the label's image should contain only binary values (1 – white, 0 – black). The algorithm may fail without this property, because the label positions won't get significant correlation values in step 3.2 of the algorithm, due to the fact that multiplication between two numbers smaller than 1 –as absolute values–, results in an even smaller number.
2. One label as the inverse of the other: one of the labels should be the inverse of the other one. This property provides us with a significant runtime saving, as only one correlation is needed to find both labels.

With this property, the best and worst correlation values designate both the positive and negative labels simultaneously.

3. Unique and well defined shape: the label's shape is very important to achieve good results in the correlation and in the edge comparison stages. The lack of this property may result in the following problems:
    - In the correlation stage: a huge amount of well-correlated positions may be found (due to similarities in the image), thus leaving much work for the next stages.
    - In the edge correlation stage: a poor defined shape will have a small number of edges; therefore there may be other areas in the scanned image with similar edge maps, and the algorithm may fail to uniquely identify the real label.
4. Label size: the label's size is related to the runtime and accuracy of the algorithm. A big size can negatively affect the runtime; however a small size can cause recognition problems. We note however that if a bigger label is used, this can also allow downsizing the scanned image to less than 100dpi in the preprocessing stage, hence accelerating the algorithm. It is recommended not to use labels of less than 1cm x 1cm when downsizing to 100dpi.

In our experiments, we used the label shown in figure 6. Its edge mask is shown in figure 7. The blurred and thresholded edge map is shown in figure 8.

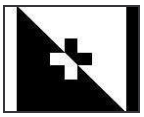

Figure 6: suggested label.

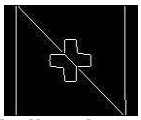

Figure 7: label's edges and its well defined shape.

Figure 8: final edge correlation kernel.

5. Software package

In the first stage of the algorithm's design, we used MATLAB for the testing. Once the algorithm was working, it was translated to C++ using MS-Visual C++. Our C++ implementation uses the LTI-LIB open-source image processing library (`http://ltilib.sourceforge.net`).

The project contains two executable files: a command line tool (`sac.exe`), and a Graphical User Interface (`sacGUI.exe`) which can be seen in figure 9. The command line tool executes the algorithm, and can be run without the GUI. It accepts BMP, JPEG and PNG image formats. The GUI application uses the command line tool in order to apply the algorithm; it requires Microsoft's .NET Framework 1.1 in order to run.

Command line usage:

```
sac.exe [options] -i <source_image>
```

Options:

-o <destination_image>	Indicates the output filename for the cropped image.
-v	Activates the verbose mode.
-graph	Displays the source and cropped images.
-dpi <resolution>	Indicates the source image resolution (default=100dpi). This argument must be used when the source image has a different resolution.

GUI usage:

Press the *Open* button and select the source image that contains the labels. After the image is shown in the left side of the window, press the *Run* button and wait until the algorithm finishes running. Finally, the cropped image will be shown at the right side of the window. Use *Exit* to close the program.

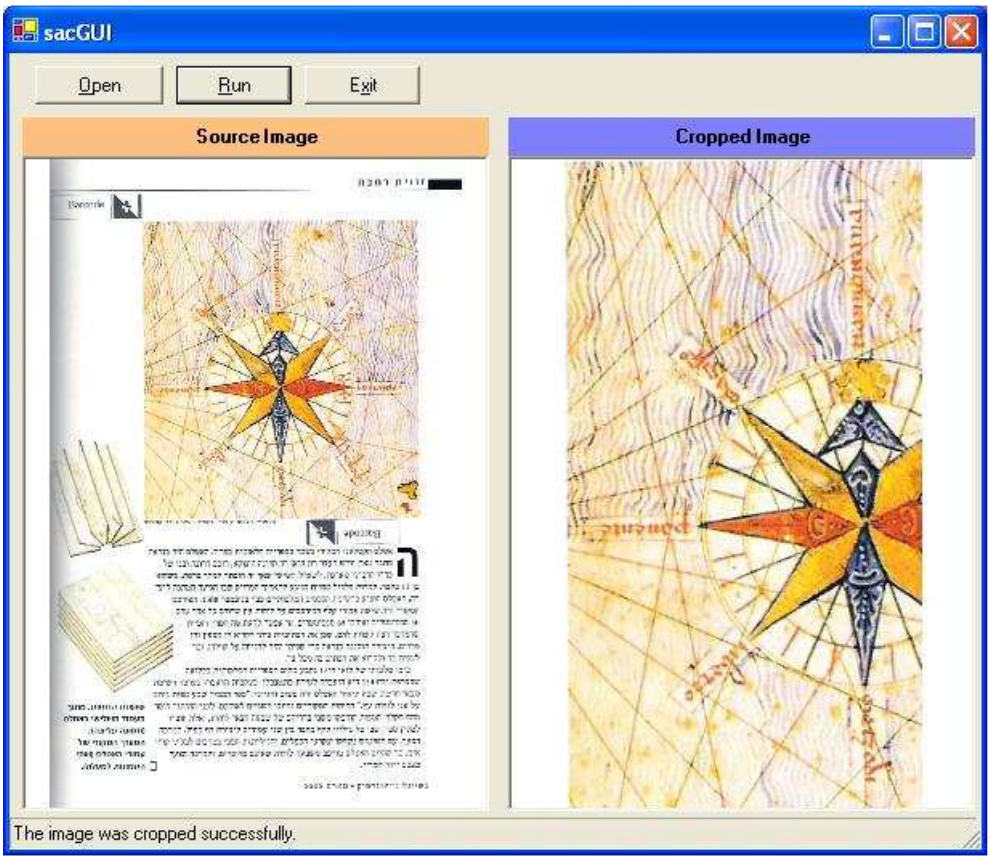

Figure 9: graphic user interface.

6. Experimental results

To determine the maximal inclination detected by the correlation stage, and to tune the edge correlation stage accordingly, a page with many inclined labels was used. As can be seen in figure 10(a), the correlation stage (3.2) is able to detect the label with an inclination of up to 20 degrees. The blue and red crosses indicate the candidate positions for the each of the labels after the thresholding stage (3.3). Figure 10(b) shows the identification results after the edge correlation stage, using a minimum acceptance value of X=100 pixels as explained in stage 3.5. As can be seen, this value accepts up to 10 degrees of inclination of the label. For each label that was detected, there is a single cross shown marking the label's final identified position.

We tested the algorithm on many images, with varying resolutions between 100 and 400 dpi. Some example results are shown in figures 11 and 12. As can be seen, the labels were found in the right places and as a result, the images were perfectly cropped.

Runtime and memory consumption results are shown in table 1. The tests were run on a Pentium M 1.6GHz with 512MB of RAM. In the table it can be seen that around 97% of the runtime is spent on correlating the image with the label mask (stage 3.2). The time required for the correlation stage depends on the size of the image after downsizing to 100dpi; the memory usage depends on the original image's resolution.

Figure 10: inclination acceptance after thresholding stage 3.3 (a) and after edge correlation stage 3.5 (b).

Figure 11: result of a text example.

Table 1: runtime and memory consumption.

Image	Source Resolution	Normalized Image size	Correlation time	Total runtime	Memory peak
Figure 9	100 dpi	645 x 1001	6.029 sec.	6.099 sec.	12MB.
Figure 10	100 dpi	645 x 996	6.019 sec.	6.079 sec.	12MB.
Figure 11	100 dpi	840 x 1000	7.941 sec.	8.032 sec.	14 MB.
Figure 11	300 dpi	840 x 1000	7.972 sec.	8.212 sec.	69 MB.
Figure 11	400 dpi	840 x 1000	7.982 sec.	8.342 sec.	114 MB.

Figure 12: result of a picture example.

7. Summary and conclusions

We have developed and implemented a simple algorithm, capable of identifying and cropping a desired area from a source image, designated by two labels. It features high stability while maintaining efficiency, may be applied to both color and gray-scale images, has a small number of parameters, and can be fine-tuned to tolerate varying amounts of noise and rotation. The algorithm presented is a combination of a correlation-based method and a shape-matching method, whose combination leads to a fast and robust process. We defined properties for the label's design, which provide better runtimes and precision; these properties still allow much flexibility in designing the label. Finally, we presented experimental results, demonstrating the algorithm's capabilities.

8. Future work

The correlation method uses 100dpi resolution images. This value is related to the label's size. We propose extending the algorithm, so it can accept new types of labels and for each one determine the lowest resolution to be used, without loosing stability. This is an important step to upgrading the current run-time performance of the algorithm.

A user may want to designate more than one cropping rectangle in a page; in this case there would be more labels on it. The current algorithm doesn't accept many labels on the same page. This algorithm extension, introduces a new problem of pair matching.

The current version of the algorithm assumes that the user wants to extract an axis-parallel rectangle from the image. This supposition can be relaxed to allow rotated cropping rectangles.

Appendix: Project file list

`main.cpp`	This is the main application source file. Contains the code that validates the command line parameters and the input file. It also reads the input file and writes to the output file.
`Mask.*`	These files define an abstract Mask class. All classes defining a label type should inherit from this class.
`MaskFactory.*`	These files define the factory class that generates instances of any label type.
`RegularMask.*`	These files implement the label suggested in the project.
`SearchAndCrop.*`	These files contain the algorithm implementation.
`tuple.h`	This file implements a tuple class, used to wrap a few classes into one class.