MVA'94 IAPR Workshop on Machine Vision Applications Dec. 13-15, 1994, Kawasaki

A PROCEDURE FOR SEGMENTING TOUCHING NUMBERS IN CADASTRAL MAPS

Gladys Monagan
Institut für Informationssysteme
Swiss Federal Institute of Technology (ETH)
ETH-Zentrum, CH-8092 Zurich, Switzerland

ABSTRACT

We present a procedure to split (or segment) touching numerals. We are neither assuming a dominant orientation for the text direction nor long character strings. The basic idea is to compute the convex hull of the outer contour, to choose a baseline from one of the sides of the convex hull and to project the pixels of the touching characters (either perpendicularly or at a slant). Lu's function is then used to try to identify the splitting position. Other steps are then undertaken to find the splitting position of the more difficult cases. We present some results taken from touching numerals in cadastral maps.

INTRODUCTION

Characters in a document touch each other for such reasons as poor quality printers or because the characters were hand written that way. In addition, characters in a scanned binary image can touch as a result of the binarization process. Yet, very few optical character recognition (OCR) programs can recognize touching characters and for this reason, these touching characters must be segmented. We devised this approach for splitting characters from a need to recognize parcel numbers, building numbers, coordinate net numbers, and street names in cadastral maps.

PROBLEM AT HAND

We start with a binary image (at this point in time we cannot obtain gray scaled images because we work cadastral maps that are on the average 700mm × 1100mm). After some simple image processing, the connected components are computed and these are roughly divided into a class of graphic components and a class of candidate symbols and characters. We group the connected components that could be characters and thus we obtain the bitmaps that could corresponds to the text entries. This is then the input to our character recognition program.

The following points are particular to our application of recognizing text in cadastral maps, that which made it difficult for us to take over a method for splitting characters from the literature.

- A text entry may have any orientation and different text entries can have different orientations (but within a text entry the text direction is consistent).
- Some text entries are short (2 or 3 characters) whereas others consist of longer numbers (like the coordinates of the net numbers) or of several words (like the street names).
- If the text cannot be recognized with a high certainty, it should not be recognized at all. We are dealing with cadastral maps, i.e. legal documents, thus the only types of errors allowed are *rejections*. Substitutions should never occur.
- The fonts are not proportional (the text is usually printed by hand using templates of the letters and numbers).
- The entries are sometimes in italics and recognizing whether they are italic or not is crucial. For example, if a number is written with a small font and in italics, it delineates a building, otherwise, if the number is in bold it delineates a parcel.

And so, there are several rules for the various types of text entries.

- There may be more than 2 characters touching and within a number (or word) there may be several occurrences of touching characters.

In our experiments we have not considered neither ligatures, broken characters, nor characters written in serif fonts.

OUR APPROACH

To solve the problem of touching characters, Kahan, Pavlidis and Baird (1987) list the three subproblems must be solved: recognition, segmentation, and reclassification.

Recognition: During the recognition process, connected components are spotted as touching characters. The OCR system we are using is contour-based using Fourier descriptors (Lorenz and Monagan, 1994). This OCR system returns the characters identified with a certainty value measuring the match between the contour and the character models. We decide that two or more characters are touching when the certainty value of a contour is below a certain threshold. Setting this threshold is not always trivial. For instance, if the threshold is too low, then $\mathfrak{F}$ is recognized as the numeral 8 and the touching fours go undetected.

Segmentation: The purpose of the segmentation is to identify which parts of the connected component belong to the contours of the single characters. This means determining a splitting position and possibly removing extra pixels which resulted from the split. Since the OCR system that we use is robust enough, after the split we do not need to remove the extra pixels. Following is the procedure that we use to segment.

1. Identify the outer contour of the connected component which was recognized to contain touching characters.
2. Compute the convex hull of this contour.
3. Find the baseline.
4. Form a parallelogram around the touching characters using the baseline and the vector making a slanted angle to the baseline (if there is such a vector). We know that the characters are written in italics when the parallelogram is slanted.
5. Project the pixels in the direction parallel to the sides of the parallelogram which make an angle to the baseline.
6. Find the splitting positions in this projection using Lu's peak-to-valley function.
7. If the split fails, because no splitting positions are found or because these are wrong, try determining the splitting positions with "brute force". We refer to the contours to the left and to the right of the splits as "resulting contours".
8. Change the parallelogram (as shown below) and repeat steps 5. to 7. In addition to the *else if* clauses below, we have flags in our program to prevent us from considering parallelograms that have already been tried out:

   if (resulting contours reclassified)
   then return splitting position
   else if (parallelogram is slanted)
   then swap baseline and side, do 5.–7.
   else if (parallelogram is slanted)
   then make it a rectangle, do 5.–7.
   else if (parallelogram is a rectangle)
   then swap baseline and side, do 5.–7.
   else if (resulting contours not too small)
   then split resulting contour(s)
   else
   fail.

Figure 1: This example shows how a slanted projection is not always successful and why we sometimes have to test with a parallelogram that is a rectangle (i.e. why we have to change the original parallelogram). (a) is an example of a 30 that was successfully split from the slanted parallelogram (as shown in (b)). In contrast, the 30 in (c), which was taken from the same map, needed a further iteration with a perpendicular projection. (d) shows the rectangle and the correct split.

Reclassification: The contours which result from the split(s) are input into the OCR

system again. It must be decided whether to accept these new contours (the old one), or neither. As before, we rely on the OCR program's certainty values. If the certainty values are not high enough, we reject the split(s) and return that splitting into meaningful characters was not possible.

SEGMENTATION DETAILS

Finding the Baseline: Our approach is somewhat similar to the work of Takizawa et al. (1993). Though used for the extraction of Chinese character strings in "unformed" documents, they too begin by finding the convex hull of the Chinese characters since the direction of the text is not known a priori.

We weight each vector of the convex hull with its length plus the lengths of the vectors in the convex hull that are parallel (or almost parallel) to this vector. The vector with the longest length and the highest weight is declared to be the baseline.

To determine whether the touching characters could have been written in italics we look for a "significant" vector of the convex hull whose angle to the baseline lies in a range of say 55^o to 75^o. We use the lengths of the angles parallel to a vector as a weight in determining which vector (if any) is significant, and in addition we want the ratio of the baseline to this vector to be large enough before we declare the vector to be significant. It is not always possible to tell which is the baseline and which is the vector slanted to the baseline. Thus we need to try sometimes both variations.

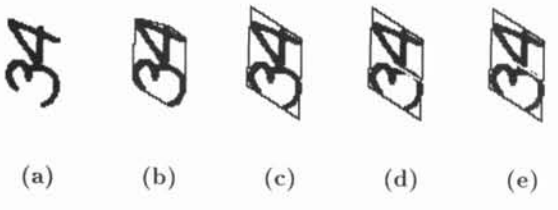
Determining the Parallelogram: The parallelogram around the touching characters is calculated by first fitting a rectangle, which is parallel to the baseline, from the convex hull. If a significant vector which makes a slanted angle to the baseline was found, second rectangle is formed, this time parallel to the slanted vector. Both rectangles are then intersected and the intersecting points determine the parallelogram (Jenny, 1993).

Splitting: To find the splitting position, we use the discrimination function given by Lu (1993). It is based on the vertical (and in our case angled) projection function $v(x)$. We repeat here her peak-to-valley function $pv(x) = \frac{v(lp)-2*v(x)+v(rp)}{v(x)+1}$ where x is the current position, lp is the peak location on the left side of x and rp is the peak location on the right side of x.

As described in the paper, when the minima in the projection are sharp, these correspond to maxima in $pv(x)$. However, the minima due to two characters are not always sharp. This function works well for "good" cases but it fails to find the dividing lines for "harder" cases like the number 584 shown on Figure 3. Lu writes that the break points should be considered as *potential* break points. We noticed that sometimes the break points are too close to each other, as in the numeral 443 shown on Figure 3 because the maxima are too close to each other along the x-axis. We correct this by averaging the two maxima into one.

Not satisfied with all the cases missed or with wrong splitting positions, we decided to use a "brute force approach" for the hard cases. We begin by splitting at positions roughly based on the number of characters that could be composing the touching contour. We consider one splitting position at a time. If the contours resulting from the split cannot be reclassified, then the splitting position is shifted to the right and to the left until either both resulting contours to the left and right of the split are recognized, or until the next splitting position is encountered.

If this fails, we assume that we took too few splits and we try the whole process again increasing the number of splits. However, we cannot guarantee that a split will be found. If the main contours of the bitmap are not identified, then we report that the split was not successful.



(a) (b) (c) (d) (e)

Figure 2: (a) is the original bitmap. (b) shows the convex hull and (c) the parallelogram. (d) shows a wrong split which would result if it was required that *only* one resulting contour to one side of the splitting position be reclassified. In (d) one can identify an A on its side, but the other contour, to the right of the splitting position, cannot be reclassified. (e) shows the correct split, with the characters to the left and to the right identified as 3 and 4.

EXPERIMENTS

We have tested the approach presented here on numbers of cadastral maps. Figure 3 shows some numbers that were segmented. We have drawn the parallelogram used for the projections and the split positions have been drawn as white lines. In all the cases in Figure 3, the results from the OCR program on the split contours were satisfactory except for 240 where the best answer returned by the OCR system is 2q0.

CONCLUSION

If a full map recognition system is to be built, a way of splitting touching characters will have to be implemented. The approach presented here is very specific for identifying numerals which have been written in any direction in a map, even when this direction is unknown. The results presented are satisfactory.

Acknowledgments: This work was partly supported by the AEW and the KWF, project 2540.1. AEW also provided us with the cadastral maps used in our tests. Parts of the program to split characters were implemented by T. Jenny. We are grateful to M. Monagan and O. Lorenz for their valuable input and to O. Lorenz for making his OCR program available.

REFERENCES

[1] T. Jenny. Trennung sich berührender Zeichen. Informatik-Semesterarbeit am Insitut für Informationssysteme der ETH Zürich, 1993.

[2] S. Kahan, T. Pavlidis, and H. S. Baird. On the Recognition of Printed Characters of Any Font and Size. *IEEE Trans. of Pat. Anal. and Mach. Int.*, PAMI-9(2):274–288, March 1987.

[3] O. Lorenz and G. Monagan. Retrieval of Line Drawings. In *Proc. of the Third Annual Symp. on Doc. Analysis and Inf, Retrieval*, pp. 461–468, Las Vegas, Nevada, Apr. 1103 1994.

[4] Yi Lu. On the Segmentation of Touching Characters. In *Proc. of the Second Int. Conf. on Doc. Analysis and Recognition*, pp. 440–443, Tsukuba, Japan, October 20-22 1993.

[5] K. Takizawa, D. Arita, M. Minoh, and K. Ikeda. Extraction of Character Strings from Unformed Docuemnt Images. In *Proc. of the Second Int. Conf. on Document Analysis and Recognition*, pp. 660–663, Tsukuba, Japan, October 20-22 1993.

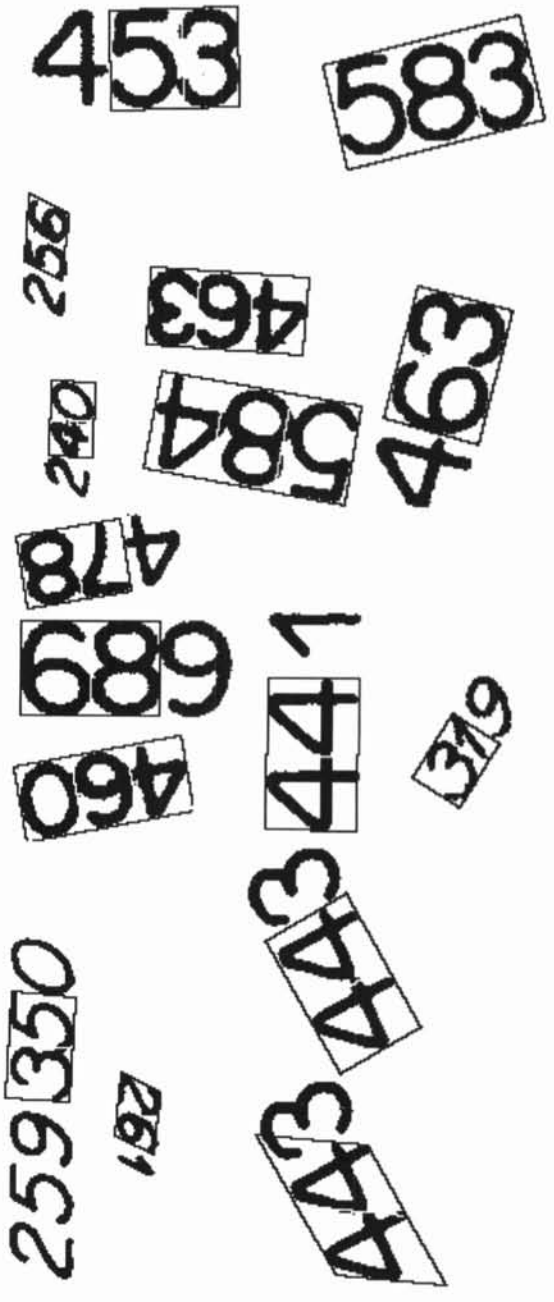


Figure 3: Results from splitting some touching characters. A threshold is used to determine whether a rectangle or slanted parallelogram is approximated around the touching 443. Note how the side of the 4 contributes towards the slanted parallelogram; nonetheless, 443 is split correctly both when the parallelogram is slanted and when it is not.