

CHARACTER RECOGNITION BY MATCHING SEQUENCES OF PSEUDO-STROKE POSITIONS AND DIRECTIONS

HANHONG XUE AND VENU GOVINDARAJU

CEDAR, State University of New York at Buffalo, Buffalo, NY 14260, USA

E-mail: {hxue, govind}@cedar.buffalo.edu

Chain-coded contours are informative in off-line character recognition. As approximations to contours, sequences of pseudo-strokes consisting of both positional and directional information make up feature vectors for character images. In order to carry out fast pattern matching, a scheme of generating fixed-length feature vectors that combine information about outer contour and inner contours into a uniform data structure is proposed and tested on CEDAR databases.

1 Introduction

Character recognizers generally form the base of higher level recognizers. For example, a word recognizer built upon character features and dynamic programming yields high recognition accuracy¹. There are also highly accurate recognizers based on holistic methods such as the one in² which adopts an oscillation model, but holistic methods have their own difficulties in recognizing numbers. So character recognizers have always been of researchers' interest.

In handwriting recognition, chain-code is often used as a compact representation for off-line binary images. A chain-code string is formally defined as a sequence of components $C = c_1, c_2, \dots, c_l$. Each component c_i is a tuple (x_i, y_i, d_i) where x_i and y_i are its coordinates, and d_i its slope quantized into eight directions. Therefore, edges in a image can be encoded as chains of components. In handwriting recognition, edges are always circular and thus referred to as contours. Contours can be either inner or outer. It can be seen that a well-written English letter or Arabic digit has only one outer contour, except for 'i' and 'j', and at most two inner contours. So we are able to encode any given character as one chain of components that represent the primary shape of the character, together with additional information about inner contours and small outer contours such as their positions and sizes.

To be more general, a chain of components does not necessarily include all the pixels on the contour that it encodes, the position of a component in the chain is not necessarily represented in integers and the slope of the component is not necessarily quantized. Therefore, we can build chains of pseudo-strokes which abstract a group of neighboring pixels and these chains are better understood as polygon approximations to the contours they encode.

Based on the above, this paper first describes our preprocessing method including contour repair and slant correction, then it introduces a new method to encode character images in fixed-length feature vectors that contain information about both outer contours and inner contours, and finally it presents

our testing results on CEDAR databases.

2 Preprocessing

2.1 Repairing broken outer contours

Broken outer contours introduce irregularities, such as (1) an inner contour joined with an outer contour and (2) one outer contour broken into two, that should be taken care of in designing a stroke-based recognizer.

For type (1) irregularity, we first find those extrema in the contour such that their incoming slope is approximately the opposite to their outgoing slope. If any two of those extrema are close enough, they are considered to be originally connected but separated due to bad image quality. Suppose the outer contour is $C = c_1, c_2, \dots, c_l$ and the two extrema we found are c_i and c_j , $i < j$. If $j - i < l/2$, then $c_i, c_{i+1}, \dots, c_{j-1}$ is an inner contour since an inner contour is always smaller than the outer contour containing it; otherwise the sub-sequence is considered as an outer contour. The result is that we get one more inner contour and the number of outer contours remains unchanged.

For type (2) irregularity, we merge two outer contours if they are close enough and both of them are of considerable size. Suppose the two contours to be merged are c_1, c_2, \dots, c_{l_1} and d_1, d_2, \dots, d_{l_2} , and the two nearest components are c_i and d_j . We obtain one large outer contour as $c_1, c_2, \dots, c_i, d_j, d_{j+1}, \dots, d_{l_2}, d_1, d_2, \dots, d_{j-1}, c_{i+1}, c_{l_1}$.

In both cases, we just add a bridge to connect the nearest two components without considering the stroke width of the character because connectivity is much more important than stroke width during matching.

2.2 Correcting slant

A stroke based recognizer is also sensitive to the slant of the input image for the slant affects the direction of each pseudo-stroke in the image. Given a chain-code string $C = c_1, c_2, \dots, c_l$, its slant $k(C)$ is calculated as follows.

$$k(C) = \frac{-\sum_i f_x(d_i)f_y(d_i)}{\sum_{f_x(d_i)=0} |f_y(d_i)| + |\sum_i f_x(d_i)f_y(d_i)|}$$

where f_x and f_y are projection functions defined in Table 1. When $k(C)$ is positive, the contour is right slanted; when negative, it is left slanted. In other words, components with slope 1 and 5 are left slanted and those with slope 3 and 7 are right slanted. The numerator gives the difference between right slanted components and left slanted components, while the denominator yields the number of vertically oriented components with left slanted ones balancing right slanted ones.

The new sequence of components with corrected slant is obtained by applying the transform $\begin{cases} x'_i = x_i + k(C)y_i \\ y'_i = y_i \end{cases}$ to each component. Now the new

d_i	0	1	2	3	4	5	6	7
$f_x(d_i)$	-1	-1	0	+1	+1	+1	0	-1
$f_y(d_i)$	0	-1	-1	-1	0	+1	+1	+1

Table 1. Projection of slopes

sequence is not strictly a chain-code sequence compared to the old sequence, because all the coordinates and slopes are real numbers and will never be quantized from now on. That means we have omitted a contour smoothing procedure that is generally required after slant correction and we will show the smoothing is actually done in extracting pseudo-strokes from the contour.

3 Processing the largest outer contour

When an image is represented in chain-coded contours, its shape is best characterized by the outer contours. In most cases, people can easily recognize a character with its inner contours removed. After preprocessing, we choose the largest outer contour and encode it in pseudo-strokes. For other contours, being either inner or outer, we just encode their positions and sizes without considering their details.

3.1 Finding the starting point

Since contours are always circular, it is important to find an appropriate starting point before we translate them into stroke sequences. Otherwise, the comparison of two shifted stroke sequences will be much more expensive. Intuitively, a top-leftmost point could be the candidate. However, we find it not good enough for characters like a 'd' and a 't' because the position of the starting point varies too much. According to our experiments, a simple choice, the topmost point, gives a more stable starting point.

3.2 Extracting pseudo-strokes

We define a pseudo-stroke to be a segment of the outer contour and encode it as a tuple $s = (x, y, d)$, where x and y are the coordinates of the origin of the stroke and d is the direction of the stroke. The assumption behind such a definition is that every stroke has the same length.

In ³, the fixed stroke length is set to be 7 pixels long, thus the resulting feature vectors have various lengths and the similarity between them is measured by edit-distance, which incurs quadratic computational cost.

To get fixed-length feature vectors, we make the length of a stroke relative to that of the whole contour. Suppose N strokes make a feature vector and the length of the contour is L pixels, then the stroke length is L/N pixels. Here N depends on the complexity of those characters being recognized, i.e. how twisted they are. In this paper, $N = 32$ is used and proved to be sufficient.

As mentioned before, we did not smooth the outcome of slant correction. The first reason is that the position of points are actually represented by real number. Now we see the second reason. Pseudo-strokes are extracted every L/N pixels, so distortion resulting from slant correction is not so important if L/N is much greater than 1.

3.3 Matching two stroke sequences

To compare the similarity between two strokes that are represented in real numbers, we first define a penalty function $p(x)$ whose domain is $[0, +\infty]$ and range is the real interval $[0, 1]$: $p(x) = \begin{cases} x, & \text{if } x < \theta \\ 1, & \text{otherwise} \end{cases}$ where θ is the penalty threshold. Then the distance between two real numbers $x_1, x_2 \in [0, 1]$ is defined as $\delta(x_1, x_2) = p(|x_1 - x_2|)$. A reason for the use of the penalty function is that if the distance between two quantities exceeds some threshold, then these two are better considered as totally different, i.e. there is a phase transition at the point θ . θ is set to 0.25 in our implementation. We also tried several kinds of sigmoid functions to replace the penalty function defined above, but they turned out to be less satisfactory.

Furthermore, we need another function for circular real numbers as used in computing the difference between two angles in radians.

$$\delta_r(x_1, x_2) = \begin{cases} p(|x_1 - x_2|/\pi) & , |x_1 - x_2| < \pi \\ p(2 - |x_1 - x_2|/\pi) & , \text{otherwise} \end{cases}$$

The distance between two strokes $s_1 = (x_1, y_1, d_1)$ and $s_2 = (x_2, y_2, d_2)$ is defined as $\delta_s(s_1, s_2) = \delta(x_1, x_2)/2 + \delta(y_1, y_2)/2 + \delta_r(d_1, d_2)$. That is, the distance between two positions is the average of the penalized horizontal distance and the penalized vertical distance. Furthermore, the distance between two stroke sequences $v_1 = \{s_1, s_2, \dots, s_N\}$ and $v_2 = \{t_1, t_2, \dots, t_N\}$ is defined as $\delta_v(v_1, v_2) = \sum_{i=1}^N \delta_s(s_i, t_i)$. It is easy to see that the distance can be computed in time linear to the fixed-length of stroke sequences, i.e. $O(N)$.

4 Processing other contours

The largest outer contour is considered as the most characteristic part of a character image and it has been encoded in pseudo-strokes as described in the previous section. Inner contours and small outer contours like i-dots are less informative, so we are only interested in their presences, positions and sizes.

4.1 Extracting contour features

A contour which is not going to be encoded in a sequence of pseudo-strokes is simply encoded by a tuple $c = (x, y, r)$ where x and y give the centroid of the contour and $|r|$ is the average distance between a point on the contour and the centroid. r is positive for an outer contour and negative for an inner contour.

A special empty tuple $\epsilon = (0, 0, 0)$ marks the absence of a contour. The data structure here happens to be the same as the one we used for pseudo-strokes thus making a uniform representation of the feature vector.

For a given image, we consider only the M largest contours excluding the one that we have encoded in pseudo-strokes. In our implementation M is set to 2, because any clear character has at most 2 inner contours or at most 2 outer contours. We follow the same criterion in ordering these contours as in finding the starting point for an outer contour. The one whose centroid is the topmost is given the first place.

4.2 Matching contours

Matching two contours is similar to matching two strokes. Suppose two contours are represented as $c_1 = (x_1, y_1, r_1)$ and $c_2 = (x_2, y_2, r_2)$. Distance

between them is defined as $\delta_c(c_1, c_2) = \begin{cases} k\delta'_c(c_1, c_2) & , c_1 \neq \epsilon, c_2 \neq \epsilon \\ kp(|r_1|) & , c_1 \neq \epsilon, c_2 = \epsilon \\ kp(|r_2|) & , c_1 = \epsilon, c_2 \neq \epsilon \\ 0 & , c_1 = \epsilon, c_2 = \epsilon \end{cases}$ where

$\delta'_c(c_1, c_2) = \delta(x_1, x_2)/2 + \delta(y_1, y_2)/2 + \delta(r_1, r_2)$ and k is a constant factor to scale up the difference because a contour is considered more discriminative than a single stroke. The larger this scale-up factor, the more unlikely it is that two images that differ in their inner contours will give a good match. k is set to 8 in our implementation. Since the radius of an inner contour is negative and that of an outer contour is positive, the match between them usually yields a large distance.

We use dynamic programming to find out the best match between two contour sequences. The time complexity is the product of the lengths of the two sequences involved. However, since the number of contours in character images is very small, the time can be taken as constant.

5 Experimental results

We have tested our method on the CEDAR BR testing set, which contains 18,468 binary images of digits. A p portion of the whole image set is chosen randomly as the reference set and the rest $1 - p$ portion as the testing set. The nearest neighbor classification is used. Table 2 shows the recognition rate as well as the rates regarding individual classes, for different ps .

The best result shown in Table 2 is 98.82% when $p = 0.20$ (3,699 reference images). It is comparable to the 98.87% obtained by multiple resolution in ⁴ and better than the 96.08% obtained by a stroke based recognizer in ³. By increasing the size of the reference image set, better results could be obtained but recognition speed slows down correspondingly.

Table 2 also shows that about 70% classification errors occur in class 2,3,8 and 9, all of which share the same characteristic, an upwardly curved stroke

at the top. It suggests that a more stable starting point is needed.

The speed of the recognition process is about 75,000 matches per second on an Ultra Sparc 10 work station, which means 75 input images can be recognized per second if the reference set contains 1,000 images.

Classes		Total	0	1	2	3	4
p=.05	Size	17540	2722	2416	1944	1644	1592
	%Correct	97.81	98.93	99.63	96.91	97.14	97.74
p=.10	Size	16616	2579	2289	1842	1557	1508
	%Correct	98.39	99.38	99.83	97.45	97.88	98.41
p=.20	Size	14769	2292	2035	1637	1384	1340
	%Correct	98.82	99.74	99.85	98.17	98.19	98.73

Classes		5	6	7	8	9
p=.05	Size	1386	1635	1535	1380	1286
	%Correct	98.48	98.65	99.28	92.46	96.42
p=.10	Size	1313	1549	1454	1307	1218
	%Correct	98.63	99.23	99.59	94.19	97.37
p=.20	Size	1167	1377	1292	1162	1083
	%Correct	99.31	99.35	99.38	95.61	98.34

Table 2. Testing results

6 Conclusions

We have proposed the method to build fixed-length feature vectors based on pseudo-stroke extraction for off-line character recognition. Since stroke sequences are sensitive to the slant and the connectivity of contours, necessary preprocessing techniques are introduced to correct slant and repair broken contours. As shown by the testing results, the method is fast and accurate, thus is potentially a good base for building higher level recognizers.

References

1. G. Kim & V. Govindaraju, "A lexicon driven approach to handwritten word recognition for real-time applications", *IEEE Trans. on PAMI*, vol. 19, no. 4, pp. 366-379, 1997
2. G. Dzuba, A. Filatov, D. Gershuny & I. Kil, "Handwritten word recognition - the approach proved by practice", *IWFHR-VI*, pp. 99-111, 1998
3. S. Cha, Y. Shin & S. N. Srihari, "Approximate stroke sequence string matching algorithm for character recognition and analysis", *Proc. ICDAR '99*, pp.53-56, 1999
4. J. T. Favata, G. Srikantan & S. N. Srihari, "Handprinted character/digit recognition using a multiple feature/resolution philosophy", *IWFHR-IV*, pp. 57-66, Dec 1994