Memory-Based Recognition of Camera-Captured Characters

Masakazu Iwamura
Graduate School of Engineering
Osaka Prefecture University
masa@cs.osakafu-u.ac.jp

Tomohiko Tsuji
Graduate School of Engineering
Osaka Prefecture University
tsuji@m.cs.osakafu-u.ac.jp

Koichi Kise
Graduate School of Engineering
Osaka Prefecture University
kise@cs.osakafu-u.ac.jp

ABSTRACT

This paper addresses how to quickly recognize a character pattern using a lot of case examples *without learning*. Here *without learning* means just finding the most similar example from the case examples, and *pretend* as if the OCR understands the definition of the character. This strategy is expected to work well in most cases with a large dataset, however, also expected to take a lot of time for finding the most similar example. In this paper, we show that a lot of case examples can be processed in a short time. As a testbed, we handle recognition problem of camera-captured printed characters. Using a database storing 100 fonts, the proposed method achieved 97.0% of recognition rate for images captured from the right angle and 95.8% for those from 45 deg. with 4.56ms of processing time, that is about 220 characters per second including every process.

Categories and Subject Descriptors

I.7.5 [**DOCUMENT AND TEXT PROCESSING**]: Document Capture—*Optical character recognition (OCR)*; H.2.8 [**Database Applications**]: Image databases; H.3.3 [**Information Search and Retrieval**]: Search process

General Terms

Algorithms, Experimentation, Performance

1. INTRODUCTION

This paper is concerned with handling a lot of character images as case examples for character recognition. Suppose that there are a lot of, say trillions of, character images with class labels. In such a case, is it adequate to perform character recognition *without learning*? Here *learning* means condensation of information extracted from training data, like calculating decision boundaries of character classes. In the contrary, *without learning* means just finding the most similar example from the case examples, and *pretend* as if the OCR understands the definition of the character. The answer to the question can be both yes and no; the answer yes may come from expectation that most similar examples are covered in the case examples, and the answer no may come from concern about existence of a heavily deformed character image which may be not covered. Even if we have a lot of case examples and many people say yes to the question, taking a lot of time for finding the most similar example is meaningless. Therefore, a technique to do it quickly is required.

In this paper, we show that a lot of case examples can be processed in a short time. As a testbed, we handle recognition problem of camera-captured printed characters. However, camera-captured character images may suffer from perspective distortion and preparing all the distorted images is not realistic. Therefore, coping with the problem by combining case examples and invariants of affine distortion, which is an approximation of perspective distortion, should be well-balanced.

Following the line, we already proposed such a method enabling us to recognize camera-captured printed characters in a complex layout in real-time [3, 4] and demonstrated it in CBDAR2009. This is the first method which satisfies the following three requirements: (1) ready for real-time processing, (2) robust to perspective distortion, (3) free from layout constraints. The method is based on affine invariant matching of connected components (CCs) stored in the database and captured with a camera. The speed is around 200 to 250 camera-captured characters per second. However, there are two main drawbacks; one is that the recognition performance is not sufficient especially for perspective images; the other is that storing CCs of many fonts in the database reduces the recognition rate.

For the sake of resolving the problems above, in this paper we introduce an approximate nearest neighbor search method [5] originally designed for specific object recognition. In the experiments, we confirmed that our new method greatly improved the recognition performance with an evaluation using character images of 100 fonts. The proposed method achieved 97.0% of recognition rate for images captured from the right angle and 95.8% for those from 45 deg. with 4.56ms of processing time, that is about 220 characters per second including every process.

DAS '10, June 9-11, 2010, Boston, MA, USA

1.1 Problem Definitions

Let us confirm the problem definitions. First of all, we assume black characters are written on a white paper for simplicity. Since character images are captured by a camera, they can suffer from perspective distortion and be degraded by defocus and low resolution. We assume, however, CCs of characters are extractable. We also assume all characters in the image exist on a flat paper (co-planer).

1.2 Applications of the proposed method

Realization of camera-based character recognition has a tremendous amount of potential. A convincing application is a webcam-based interface. This can realize the *translation camera* [7] and *scene text explorer*. The former is a portable translation device integrated with a camera-based OCR, and tells us the translated word or text pointed out with a web camera. The latter is a browser to make a word or text in a scene *clickable* by pointing it with a webcam. By *clicking* a word or text, the user can obtain some related information such as a web page and movie tied with the word or text. Another convincing application is *keyword finder*. This enables us to find registered useful keywords in a scene and provide them to the user including a visually disabled person. The system is achievable by recognizing all words captured by an omni-directional camera and providing the place where the word found when it found.

2. OVERVIEW OF BASE METHOD

In this section, we present an overview of our previously proposed method in CBDAR2009 [4] as the base method. In order to realize real-time processing, the configuration of the method is simple; adaptive binarization and contour extraction are used for segmentation, and separated character handling module and fast affine invariant matching of CCs for recognition.

We present the basic idea of the fast affine invariant matching. Let us suppose that a situation shown in Fig. 1; a reference image and an input image suffering from affine distortion need to be matched. In the situation, simple comparison of two images like superimposing one image on the other is meaningless because the shape of the input image is deformed. However, if corresponding three points are known, they can be compared reasonably after normalization making the orange line segments in the figure have a predetermined length and cross at right angle. This shows that the problem to be solved is changed to how to determine corresponding three points. For the problem, well-known geometric hashing (GH) [6, 8] provides invariant matching on a trial-and-error basis using randomly chosen three points. The computational cost of GH is $O(P^4)$, where P is the number of feature points. As the feature points, all pixels on the external contour of a CC can be used to recognize character images. The base method reduced the computational cost to $O(P^2)$ by choosing two points uniquely as shown in Fig. 2, though the remaining one point is still chosen randomly.

An overview of the base system is shown in Fig. 3. The system consists of storage and retrieval modules of CCs and separated character handling module. The rest of the section is spent for presenting the details of them; the separated character handling module, the storage module and retrieval module are presented in Secs. 2.1, 2.2 and 2.3, respectively.

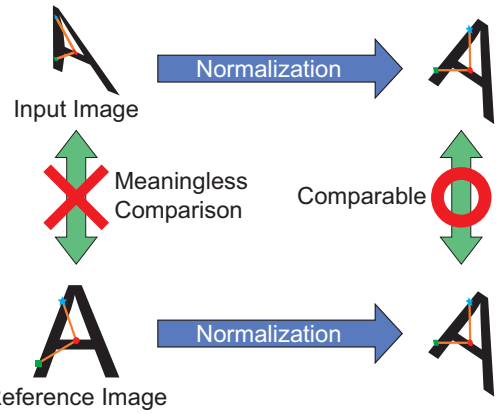


Figure 1: Basic idea of CC Matching of the base method. Since the input image is deformed, simple comparison of two images like superimposing one image on the other is meaningless. However, if corresponding three points are known, they are comparable after normalization making the orange line segments in the figure have a predetermined length and cross at right angle.

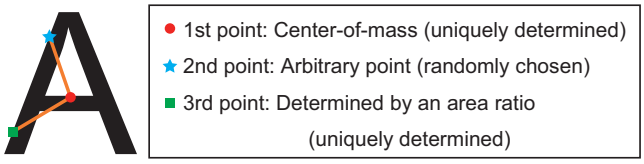


Figure 2: How to choose three points in the base method. Since two points are determined uniquely, computational cost is reduced from $O(P^4)$ to $O(P^2)$, where P is the number of feature points. As the feature points, all pixels on the external contour of a CC are used.

2.1 Handling Separated Characters

In this section, we present how to handle separated characters which consists of more than one CC such as "i" and "j." In order to do that, in the storage process, the number of CCs in a character is counted. If the number is greater or equals to two, each CC is processed separately in the storage and retrieval modules, and the character is registered into the separated character table shown in Fig. 4. The table stores the relative positions and sizes between the CCs of a character so that two CCs in the same relationship stored in the table will be recognized as a character.

In the case of Arial font, the bottom CC of "i (bar)" has the same shape of "I (capital ai)" and "l (lowercase el)," and they are indistinguishable. Thus in order to recognize "i" correctly, all CCs of the same shape such as "I" and "l" must be checked whether it is a part of "i" or not. In order to realize the process, all CCs in the same shape should be grouped and have the same CC ID. That is, during the storage process, each reference CC is checked one by one whether CCs in the same or quite similar shape are already stored or not.

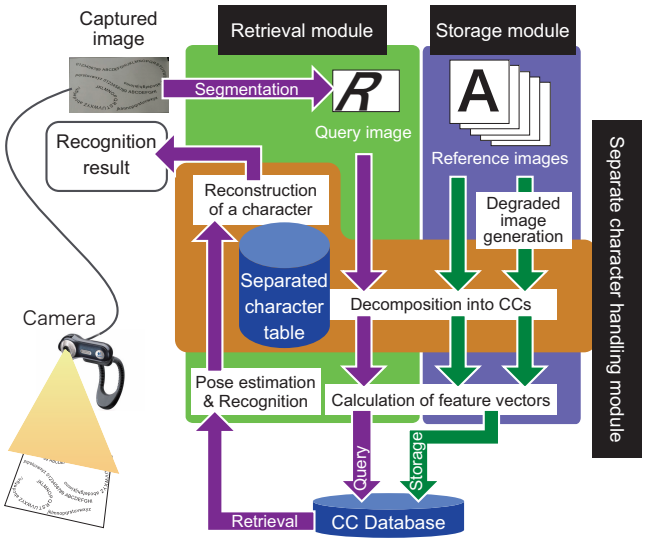


Figure 3: An overview of the base method proposed in CBDAR2009 [4].

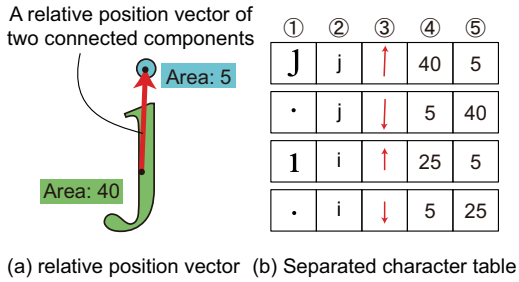


Figure 4: Separated character table. The elements of the table are, from left, ①Shape of the connected component (CC ID), ②original character, ③relational position of the paired CC, ④the area of the CC, ⑤the area of the paired CC.

In further detail, before storing a reference CC, it is recognized using the database in process of creation, and the same CC ID is assigned to CCs in the same or quite similar shape. Ideally the bottom CC of "i (bar)", "I" and "l" have the same CC ID. In the previous paper, grouping is performed manually because it did not get along with the generative learning mentioned below. However, in the current paper, we use automatic grouping to handle quite a bit of character images with a slight modification.

As mentioned above, the grouping method may group CCs of different characters into one group. In such a case, as shown in Fig. 5, we may associate more than one class label with a group. In the figure, the group 1 is associated with both class labels "0" and "c." If a CC is recognized as group 1, we cannot determine the CC is "0" or "c" in the current process and should be determined in the post processing. If a CC of "c" is recognized as either group 1 or group 2, it is regarded as correct recognition because both groups are associated with the class label "c."

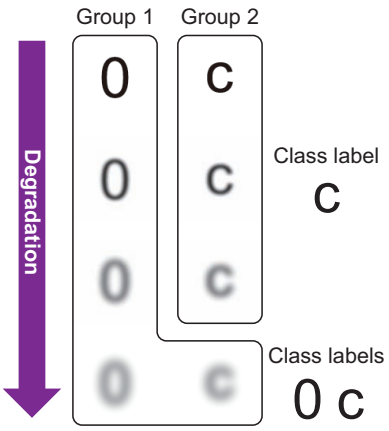


Figure 5: An example of grouping of connected components. In this case, a degraded "c" belongs to group 1 shared with "0." Thus, group 1 has two class labels "0" and "c." If a CC is recognized as group 1 in the following process, the CC is associated with both class labels "0" and "c."

2.2 Storage module

In the storage module, binary reference CC images are stored in the database. Each CC image undergone affine transformation is described with feature vectors whose elements are affine invariants.

2.2.1 *Degraded Image Generation*

In order to cope with degradation caused by defocus and low resolution, degraded character images are artificially generated by generative learning [1]. In this paper, nine degraded images (including the original reference image) are created from each reference image by applying three kinds of Gaussian blurring (including no degradation) and three kinds of degradation in resolution (including no degradation). The created images are binarized and treated as additional reference images.

2.2.2 *Feature Vector Calculation*

How to calculate feature vectors is presented in two steps with referring Fig. 6.

In the first step, a k-dimensional feature vector is created using an invariant coordinate system spanned by two bases. The figure in Fig. 6(a) is normalized into Fig. 6(b) with two bases. Then, $k(= l \times l)$ uniform subregions are defined and a histogram of black pixels shown in Fig. 6(c) is calculated. The histogram is normalized to satisfy that the sum of bins is 1, and finally a k-dimensional feature vector is obtained. In this paper, we used $l = 4$ according to preliminary experiment results.

In the second step, a $(3k)$-dimensional feature vector is obtained. In Fig. 6(a), two bases are determined from three points. Since three pairs of two bases can be calculated for three points by changing the point of intersect of two bases, we can calculate two more k-dimensional vectors in the same manner presented above and we have three vectors in total. Finally, by concatenating three k-dimensional feature vectors, a $(3k)$-dimensional feature vector is obtained.

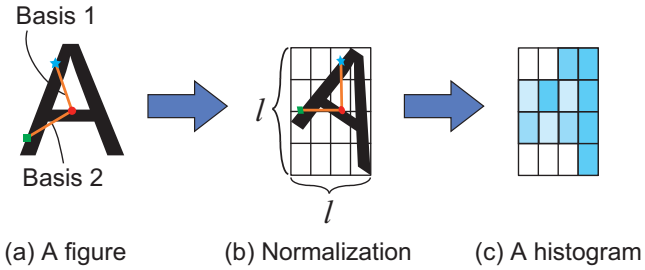


Figure 6: Calculation of a k-dimensional feature vector based on values of $k(= l \times l)$ uniform subregions. A $(3k)$-dimensional feature vector is calculated by concatenating three k-dimensional feature vectors.

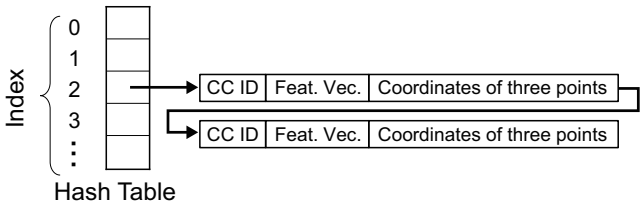


Figure 7: Configuration of the hash table for connected components.

A $(3k)$-dimensional feature vector is calculated for a set of three points. Since three points are determined uniquely from the 2nd point in Fig. 2, the number of the vectors calculated for a CC is the same as the number of pixels on the external contour of the CC. As long as the same three points are selected, the same vector is obtained regardless of affine distortion in theory.

2.2.3 *Storage into CC Database*

The base method uses the hash table shown in Fig. 7 as the database for CCs. Each entry consists of a CC ID, a feature vector and the coordinates of three points. The coordinates of three points are used for the pose estimation in the recognition process.

The index H_{index} of the hash table is calculated using a simple hash function given as

$$H_{\text{index}} = \left(\sum_{i=1}^{3k} D^{i-1} r_i \right) \bmod H_{\text{size}}, \qquad (1)$$

where H_{size} is the size of the hash table ($2^{19}-1$ was used) and r_i is the quantized value of the i-th element into D levels. $D = 2$ was used according to preliminary experiment results in this paper, though $D = 3$ was used in the previous paper. Entries are stored using the list structure if collisions occur. If the number of collisions in a hash bin is larger than c, all the entries are discarded. $c = 200$ was used in this paper.

2.3 Recognition module

2.3.1 *Image Acquisition*

An image to be recognized is captured by a digital camera or a web camera as a still image or a movie. A movie is decomposed into frame images. We call each obtained image *query image*.

2.3.2 *Segmentation*

CCs are extracted from the query image. The image is adaptively thresholded into the binary image and a contour extraction technique is applied.

2.3.3 *Feature Vector Calculation*

Feature vectors are calculated from a CC. The process to obtain the $(3k)$-dimensional feature vectors is almost the same as in Sec. 2.2.2. The only difference is that the number of feature vectors is reduced to S for speeding up. A smaller S decreases both recognition performance and processing time more. $S = 10$ was used in this paper.

2.3.4 *Recognition and Pose Estimation*

This section presents how to calculate the recognition results and poses for CCs in a query image. First of all, CC IDs and the coordinates of three feature points are obtained from the hash table shown in Fig. 7 using the feature vectors. The CC IDs are temporary recognition results and some of them are wrong. Thus we have to extract correct results. In the previous paper, a few steps of a voting procedure similar to [2] was used. That is, as shown in Fig. 8, pose estimation of the plane of the paper is performed, and then recognition and pose estimation of each CC are performed.

Firstly, from the correspondence between feature points in the query CC and a reference CC, the pose of the CC is calculated as an affine transformation matrix. Since some of them are wrong, they are filtered by weighted voting of CC IDs for each CC as shown in Fig. 8(a). The reason why the voting is weighted is a CC having longer external contour may have unfairly large number of votes. Letting N_i be the number of feature points (the length of the external contour) of i-th CC, the weight for the i-th CC is defined as $\frac{1}{\sqrt{N_i}}$. Let M be the number of the highest vote, and characters having larger number of votes than $0.9M$ are grouped in *estimation group* and characters having larger number of votes than $0.8M$ are grouped in *candidate group*. These groups are defined for each query CC in a query image.

Secondly, the pose of the paper is estimated. Since we assume all the characters exists on a flat paper, all CCs are expected to share the same parameters of shear and independent scaling. Thus, as similar to [2], a pair of plausible parameters are estimated by using density estimation in the 2D space as shown in Fig. 8(b). That is, affine transformation matrices of the estimation group are plotted in a 2D space and the densest point represented by a red star mark in Fig. 8(b) is selected. In order to increase reliability, only CCs satisfying $T_{\text{area}} \leq R/\beta^2 \leq 1/T_{\text{area}}$ are used for the estimation, where R is the area ratio of the CC of the query image and the corresponding CC of the reference image, and β is the scaling parameter calculated from the affine transformation matrix. $T_{\text{area}} = 0.7$ was used in the paper.

Thirdly, the recognition result of each CC is determined. As shown in Fig. 8(c), a pair of plausible rotation angle and recognition result of the CC are estimated by using density

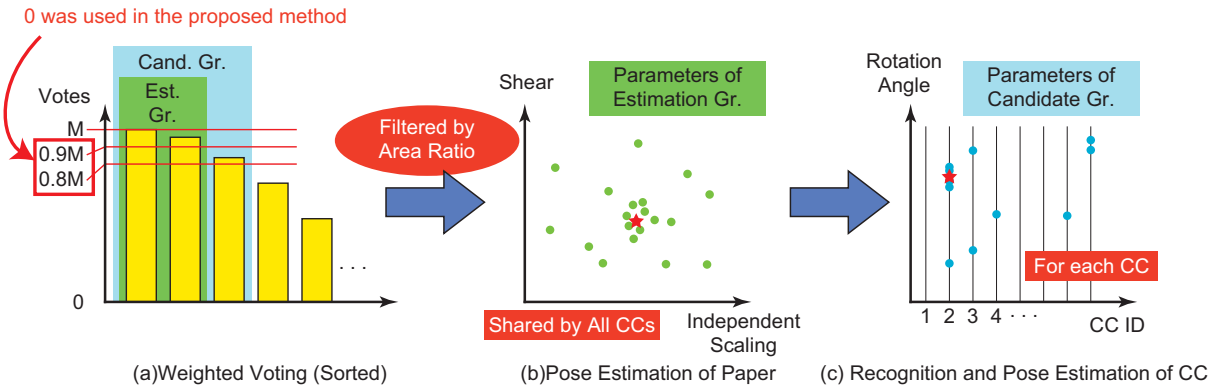


Figure 8: Recognition and pose estimation of a connected component in the base method. In the current paper, the thresholds for determining *estimation group* and *candidate group* are changed to 0, so that all the characters belong to both *estimation group* and *candidate group*.

estimation in the 2D space. Affine transformation matrices of the candidate group are used. The difference from Fig. 8(b) is that the density estimation is carried out in 1D space since the CC ID is a discrete value. Finally, the process in this section estimates recognition result (CC ID) and pose (shear, independent scaling and rotation) of the CC.

3. PROPOSED METHOD

In this section, we present the proposed method in the current paper, which improves recognition performance of the base method. For the sake of that, we introduce two strategies used in [5].

We begin with introduction of a *flipping a bit* strategy illustrated in Fig. 9. The strategy is for creating new query vectors in the recognition module. As presented in Sec. 2.2.3, each feature vector is binarized for calculating a hash index with Eq. (1). The strategy increases the number of hash indexes calculated from a vector. This can increase both recognition rate and processing time. In the preliminary experiment, we confirmed that this increased at most about 0.3% in recognition rate and constantly about 0.1ms in processing time. $e = 0.002$ and $b = 8$ were used for 48-dimensional vectors in this paper.

Secondly, we introduce distance calculation. As presented in Sec. 2.3.4, some of feature vectors retrieve wrong entries. In the previous method, relatively reliable ones are extracted from them by voting strategy illustrated in Fig. 8(a). In the current paper, the extraction is performed by distance calculation of feature vectors. That is, Euclidean distances between the original feature vector of the query CC and feature vectors obtained from the hash table are calculated. Then, feature vectors having smaller distances than a threshold are extracted. Since the extraction by the distance calculation is very effective, the weighted voting presented in Sec. 2.3.4 is not used anymore. In other words, the thresholds for determining *estimation group* and *candidate group* are changed to 0, so that all the characters belong to both *estimation group* and *candidate group*. 0.1 was used for the threshold in this paper. In the preliminary experiment, we confirmed that this increased from 2.86% to 3.14% in recognition rate and

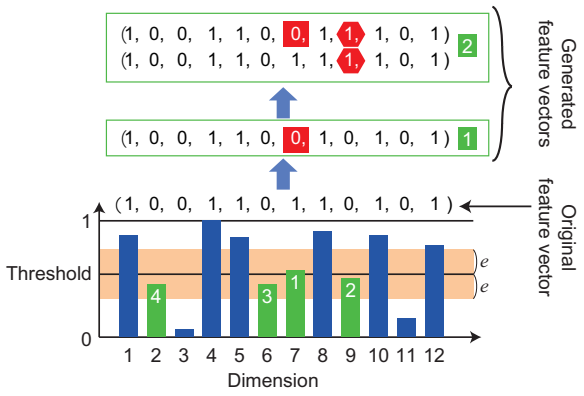


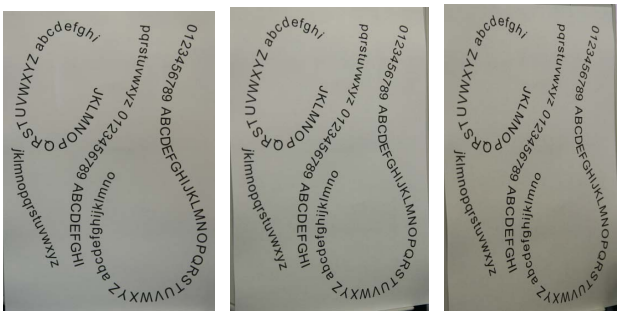
Figure 9: Binarization and query vector generation in [5] in the case of 12 dimensions and $b = 2$. When a real-valued vector is binarized, the differences between the threshold and the values of vector's elements are sorted in ascending order. First at most b elements whose differences are less than e are flipped to generate new query vectors. In the case, dimensions 7 and 9 are selected for flipping and three new vectors are created in addition to the original one.

at most 0.7ms in processing time. Note that the method in [5] is different a bit from the one we presented here. That is, the method uses only the feature vector which has the smallest Euclidean distance. However, in a preliminary experiment, the recognition performance of the strategy was lower than the one using a threshold.

4. EXPERIMENT

In order to evaluate the effectiveness of the proposed method, an experiment of recognizing camera-captured characters in various fonts was carried out. 100 fonts were selected from the ones installed to Microsoft Windows 7, excluding ones having thin strokes to avoid a CC being decomposed. 10 fonts shown in Fig. 10 were selected for testing from the 100 fonts. In the experiment, the number of fonts stored in the database was increased, and recognition rates and processing time were observed. When the number of fonts was one,

Figure 10: Fonts used as recognition targets.



(a) 0 deg. (b) 30 deg. (c) 45 deg.

Figure 11: Camera-captured images of a recognition target of Arial font captured from various angles and cropped manually.

only Arial font was stored in the database and used for testing. When the number increased to two, Arial and Century fonts were used for both storage and testing. Similarly, fonts stored in the database were used for testing up to 10 fonts. When the number of fonts was larger than 10, all the fonts shown in Fig. 10 were used for testing.

We employed 62 characters of numerals and alphabets: 10 figures, 26 lowercase alphabets and 26 capital alphabets. Since eight degraded images were created for each character image, in total 55800 images were stored in the database. Recognition targets were prepared to contain each character twice (124 characters in total) in the same layout for all fonts. They were printed on a A4 paper and captured with a digital camera. The camera-captured images of the recognition target of Arial font captured from 0, 30 and 45 deg. are shown in Fig. 11. The sizes of the images were 1549×2197, 1441×2201 and 1249×2213, respectively.

As presented in Sec. 2.1, CCs were automatically grouped during the storage process. For checking whether the database already contains CCs in the same or quite similar shape to the new CC, the proposed method without generating new query vectors was used. Table 1 shows the result of the automatic grouping when the number of fonts was one (i.e., Arial font only). For 62 characters of Arial font, 48 groups were created. The table shows only groups associated with more than one character class.

Table 1: The grouping result of 62 characters of Arial font. 48 groups were obtained by the automatic grouping presented in Sec. 2.1. The table shows that only groups associated with more than one character class.

0 O c o	6 8 9 S e	C c	E m
I i(bar) i(dot) j(dot) l	N Z z	S s	V v
W w	b q	d p	n u

The experiment was carried out on a server with AMD Opteron 2.8GHz. In order to reduce the computational cost, reference images and query images were normalized so that the largest sizes of the width and height of an image were 100 pixels and 50 pixels, respectively.

The results shown in Figs. 12(a) and (b) are recognition rates and average processing time per character including every process, respectively. In the figure, "CBDAR2009" represents the base system. The figures show that the proposed method achieved 97.0% of recognition rate for images captured from the right angle and 95.8% for those from 45 deg. with 4.56ms of processing time, that is about 220 characters per second including everything. The differences between the proposed method and the base method were 3.20% in recognition rate of 0 deg. and 3.63% in 45 deg. with increase of about 0.75ms in average processing time. They show that the effectiveness of the newly introduced strategies and that the total performance of the proposed method is very high.

Figs. 12(c), (d) and (e) show the number of groups, the number of entries stored in the database and memory amount, respectively. They show that all of them increased as the number of fonts increased. However, the gradients of them were not the same. The gradient of the number of groups was especially large when the number of fonts was less than 10. This means that CCs of newly added fonts tended to belong to existing groups rather than created a new group when the number of fonts was larger than 10. The gradient of the number of entries decreased as the number of fonts increased. The reason seems that the number of collisions in many hash bins exceeded the threshold and all the entries in the bins were discarded. In the contrary, the gradient of memory amount did not change so much. This means that even if the gradients of the number of groups and entries decreased, the information on the new CCs kept stored in the database. This may comes from an implementation matter related to the vector class of C++ STL. Finally, for 100 fonts, the memory amount exceeded 6.5GB and the number of entries was about 7.9 million. Nevertheless, average processing time did not increase so much. This also shows the effectiveness of the proposed method.

5. CONCLUSIONS

This paper addressed how to quickly recognize a character pattern using a lot of case examples. This strategy is expected to work well in most cases with a large dataset,

however, also expected to take a lot of time for finding the most similar example. In the experiment of camera-captured printed characters, we showed that a lot of case examples were processed in a short time. That is, using a database storing 100 fonts, the proposed method achieved 97.0% of recognition rate for images captured from the right angle and 95.8% for those from 45 deg. with 4.56ms of processing time, that is about 220 characters per second including everything.

In this paper, we did not take care about the memory amount. Thus future work includes reduction of memory amount.

6. ACKNOWLEDGMENTS

This research was supported by KAKENHI 21700202 and Research for Promoting Technological Seeds, JST, 2009.

7. REFERENCES

[1] H. Ishida, S. Yanadume, T. Takahashi, I. Ide, Y. Mekada, and H. Murase. Recognition of low-resolution characters by a generative learning method. In *Proc. CBDAR2005*, pages 45–51, 2005.

[2] M. Iwamura, R. Niwa, A. Horimatsu, K. Kise, S. Uchida, and S. Omachi. Layout-free dewarping of planar document images. In *Proc. DRR XVI*, 7247-36, Jan. 2009.

[3] M. Iwamura, T. Tsuji, A. Horimatsu, and K. Kise. Real-time camera-based recognition of characters and pictograms. In *Proc. 10th International Conference on Document Analysis and Recognition (ICDAR2009)*, pages 76–80, July 2009.

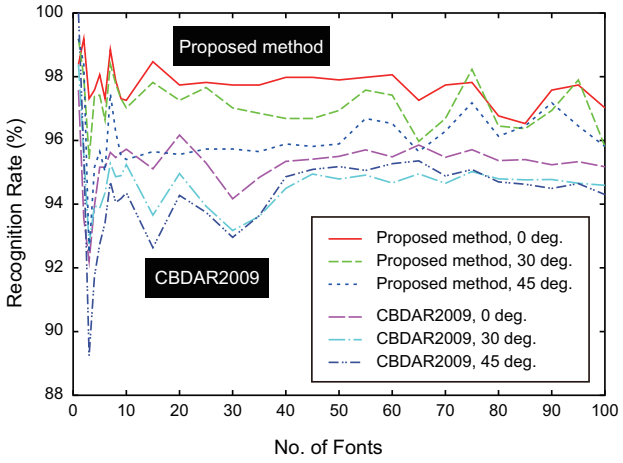
[4] M. Iwamura, T. Tsuji, A. Horimatsu, and K. Kise. Real-time recognition of camera-captured characters in complex layouts. In *Proc. Third International Workshop on Camera-Based Document Analysis and Recognition (CBDAR2009)*, pages 53–60, July 2009.

[5] K. Kise, K. Noguchi, and M. Iwamura. Simple representation and approximate search of feature vectors for large-scale object recognition. In *Proc. 18th British Machine Vision Conference (BMVC2007)*, volume 1, pages 182–191, Sept. 2007.

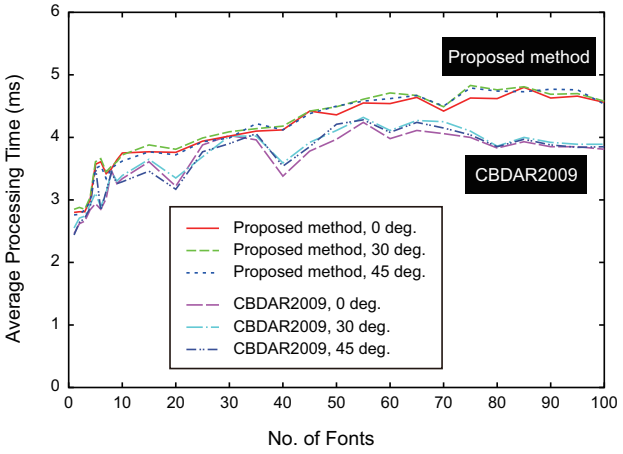
[6] Y. Lamdan and H. J. Wolfson. Geometric hashing: a general and efficient model-based recognition scheme. In *Proc. ICCV1988*, pages 238–249, 1988.

[7] Y. Watanabe, Y. Okada, Y.-B. Kim, and T. Takeda. Translation camera. In *Proc. ICPR1998*, pages 613–617, 1998.

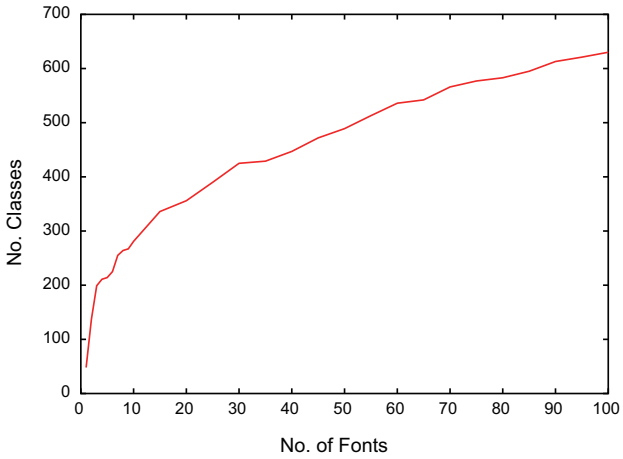
[8] H. J. Wolfson and I. Rigoutsos. Geometric hashing: an overview. *IEEE Computational Science and Engineering*, 4(4):10–21, 1997.



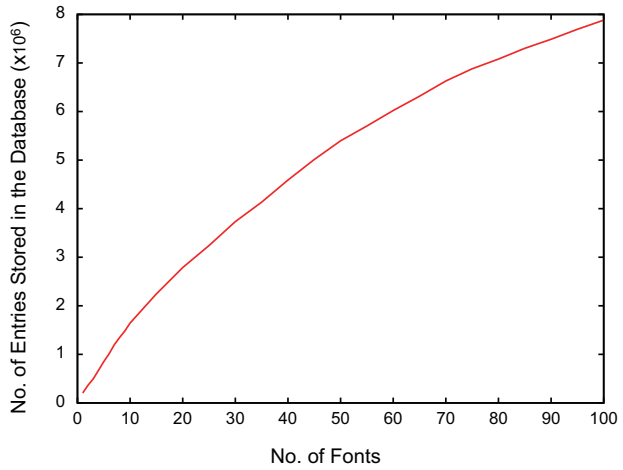
(a) Recognition rates.



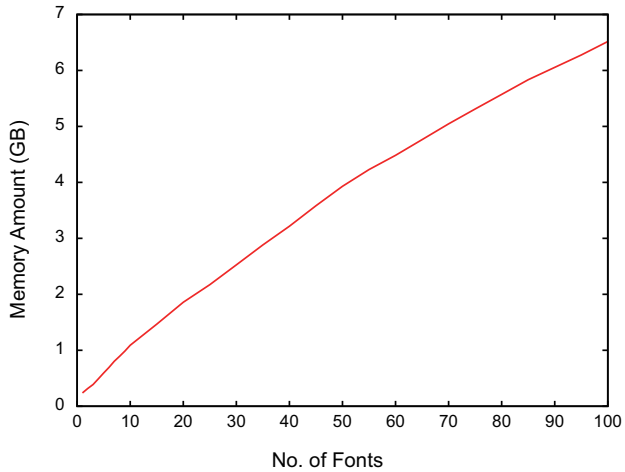
(b) Average processing time.



(c) Number of groups.



(d) Number of entries stored in the database.



(e) Memory amount.

Figure 12: Experimental results.