Memory Reduction Method for Specific Object Recognition Based on Lossy Encoding of Feature Vectors with a Bloomier Filter

Katsufumi INOUE† and Koichi KISE†

† Graduate School of Engineering, Osaka Prefecture University
1–1 Gakuencho, Naka, Sakai, Osaka, 599–8531 Japan
E-mail: †inoue@m.cs.osakafu-u.ac.jp, †kise@cs.osakafu-u.ac.jp

Abstract Specific object recognition method with the nearest neighbor search of local features needs an immense of memory storage to store them for distance calculation since the number of local features is very large. A way to solve this problem is to skip the distance calculation. In this paper, we propose a memory reduction method with a Bloomier filter which is far memory efficient than hash tables. From the experiments with 55 3D objects and 5,000 planar objects, the proposed method was successful to reduce the required memory for specific object recognition compared with the method with a hash table.

Key words Bloomier filter, Bloom filter, Memory reduction, Specific object recognition

1. Introduction

Object recognition is one of attractive tasks in computer vision or pattern recognition. Many researchers deal with this theme and their accomplishments help to develop a new service. For example a camera based information retrieval service such as "Google Goggles" [1] is a famous one. The tasks of object recognition are divided into two categories: generic [2] and specific [3], [4]. The former is to recognize *classes* of objects. On the other hand the latter is to identify object *instances*. In this paper, we focus on the latter task, especially the methods which utilize local features such as PCA-SIFT [5].

Local features are often employed for object recognition because they have highly discriminative power. Matching between local features extracted from a query image and images for the database is one of simple methods for object recognition. The nearest neighbor search is often utilized for matching. Although even this simple method enables us to achieve high recognition accuracy, it poses the problem that an immense of memory storage is required to store all of local features for distance calculation since they are multidimensional vectors and hundreds to thousands of them are extracted from a single image. Some methods to solve this problem have already been proposed. One of them is based on vector quantized feature vectors called "*visual words*" [6], [7]. This method allows us to reduce the amount of memory by reducing the number of feature vectors stored in the database with the performance of k-means clustering. However, only a large number of "*visual words*" enables us to achieve high accuracy of recognition [7], [8]. Therefore, the achievement of memory reduction with this method is very difficult while keeping the recognition accuracy as high as possible.

A possible approach to solve this problem is to recognize the objects without distance calculation. From this viewpoint, a hash-based method has been proposed [9]. In this method, the similarity of feature vectors is identified not by distance calculation of them but by checking whether they have same hash values. The contribution of this method is to reduce the memory usage dramatically because only the object IDs are stored in the hash table and there is no need to store feature vectors for distance calculation. However this method still has a problem about the amount of memory: although a very large size of hash table is necessary to improve the discriminative power of feature vectors, most bins of hash table are empty. Accordingly, there is still a room of improving the space efficiency of this method.

Our approach to solve this problem is as follows. Our approach has two key ideas, 1) we utilize a small size of hash table to reduce the amount of memory, 2) multiple hash functions are employed for identifying the similarity of feature vectors. From these ideas, we address the problem of memory usage and control the discriminative power of feature vectors. From this viewpoint, we propose a memory reduction method with a Bloomier filter [10]. Bloomier filter is one of probabilistic data structures and is more space efficient than hash tables. Similar to hash tables, the Bloomier filter allows us to check the similarity of feature vectors without distance calculation of them. The difference between hash tables and

the Bloomier filter is the accuracy of identifying the similarity of feature vectors. Although the accuracy of identification with the Bloomier filter is lower than that of hash tables, we utilize the Bloomier filter to reduce the memory usage. In this paper, we compared the proposed method with the conventional method with a hash table by using planar and 3D specific objects.

2. Conventional Method Based on a Hash Table

In this section, we explain a conventional method with a hash table [9]. This method is a base of the proposed method.

2.1 Database Construction

First, we explain the database construction process of the conventional method. The conventional method employs feature vectors calculated by PCA-SIFT [5]. The number of dimensions of feature vectors is 36. Let $\boldsymbol{p}$ be an original feature vector $\boldsymbol{p} = (p_1, p_2, \ldots, p_{36})$ extracted from a "model image" which is the image for database construction. First, in order to index $\boldsymbol{p}$ in the hash table, we convert it into its binary representation as a bit vector $\boldsymbol{u} = (u_1, u_2, \ldots, u_d)$ using first d dimensions of $\boldsymbol{p}$ as follows:

$$u_j = \begin{cases} 1 & \text{if } p_j \geqq 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Next the following hash function

$$H_{\text{index}} = \left(\sum_{j=1}^{d} u_j 2^{(j-1)} \right) \bmod H_{\text{size}} \tag{2}$$

where H_{size} is the size of the hash table is applied in order to obtain the hash value of $\boldsymbol{p}$. Then only ID of the object from which $\boldsymbol{p}$ is extracted is stored in the bin of hash table indicated by the hash value. If multiple vectors have the same hash value, we utilize the chaining algorithm to store them in the hash table. After this process is applied to all feature vectors extracted from model images, if the length of the chain exceeds the threshold c, we delete all entries with the same hash value. This is the strategy called "stopword elimination".

2.2 Object Recognition

Next, we introduce the object recognition process of the conventional method. The most important step of this process is to retrieve the feature vectors having the same hash value which is obtained from a query vector $\boldsymbol{q}$. Recall that a feature vector is converted into a bit vector. Unfortunately, since variation of a query vector may change its bit vector, the hash value calculated from $\boldsymbol{p}$ close to $\boldsymbol{q}$ might not be obtained. To solve this problem, in [9], a query feature vector having the values close to threshold of 0 for binarization of the j-th dimension can be converted into bit vectors different from its original one. This conventional method simply employs the threshold of error range e (e.g. 200 in our experiments) as a parameter for generating different bit vectors. For the dimension j such that $|q_j| \leqq e$, the other bit value $u'_j = 1 - u_j$ is also utilized for recognition process. Note that unlimited application of this strategy increases the expanded bit vectors exponentially. Therefore, in order to avoid this, the conventional method utilizes the limit b of the number of dimensions for the application. If the number of dimensions that satisfies the threshold e exceeds the limit b, those with larger indices are adopted up to the limit.

Next, the conventional method simply retrieves the object IDs from the hash table with the set of bit vectors calculated by the above strategy. Then we simply vote the objects. Finally, the object having the maximum number of votes is regarded as the recognition result.

2.3 Problem of the Conventional Method

In the conventional method, a large size of hash table is necessary to achieve high accuracy. In [9], since the hash size $H_{\text{size}} = 2^d$ is utilized, the parameter d controls the size of hash table. Although a larger d allows us to achieve high accuracy of recognition, we need more memory space for the hash table. Moreover, it causes the problem that most bins of hash table are empty. In the preliminary experiment with a hash table whose size is $H_{\text{size}} = 2^d$ and 10 million feature vectors calculated by PCA-SIFT, more than 65% bins of the hash table with $d = 24$ and more than 96% of them with $d = 28$ were empty. From this preliminary experiment, we confirmed that there was still a room of improving the space efficiency of the conventional method.

To solve this problem, we utilize a small size of hash table to reduce the memory usage and multiple hash functions to control the discriminative power of feature vectors. From this viewpoint, we propose a memory reduction method by using the Bloomier filter.

3. Bloom and Bloomier Filters

In this section, we explain the Bloomier filter [10] and the Bloom filter [11] which is a basic technology of the Bloomier filter.

3.1 Bloom Filter

The Bloom filter is a space efficient probabilistic data structure. This is a bit array of m bits. In the following, we call m as "*Table Size*". The Bloom filter is used to memorize whether an element is a member of dataset and has a risk of false positives which are a type of error that an elements is erroneously recognized as a member of dataset.

The storage process of the Bloom filter is as follows. First, a bit array of m bits which are all set to 0 is prepared as shown in Fig. 1. Next, an element is converted into k hash values by

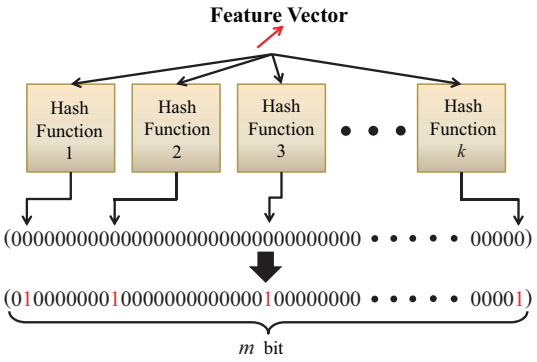


Figure 1 Overview of the storage process of the Bloom filter.

using k different hash functions. These hash values are integers ranging from 1 to m and each hash value indicates one of the m array positions. Then the bits at all positions indicated by hash values are set to 1. In the retrieval process, feed the query element to the k hash functions to determine the k array positions. The query element is regarded as a member of the dataset if and only if all bits at these positions are 1. In this process, there is the case that all bits happen to be 1 though the query element is not actually a member of the dataset. This is because all bits have accidentally been set to 1 during the storage of some other elements in the dataset. In this case, the false positive occurs.

If we can construct hash functions which convert an element into an integer indicating one of m array positions, the Bloom filter can store elements of all data types. In this research, we utilize feature vectors as elements for the Bloom filter and the dataset for the Bloom filter means the object database.

3.2 Bloomier Filter

The Bloomier filter is an associative array which can associate a value with a feature vector and is composed of multiple Bloom filters. Because of the effect of false positive in the Bloom filter, the Bloomier filter also has a problem that we cannot necessarily obtain the correct associated value. Let us explain how to associate a value with a feature vector by using the following example. Suppose the values associated by the Bloomier filter are simply "0" and "1". In this case, we prepare a pair of Bloom filters $B^{(0)}$ and $B^{(1)}$. Then the Bloom filter $B^{(0)}$ store the feature vectors whose associated value is "0", and the Bloom filter $B^{(1)}$ store those whose associated value is "1". In the estimating process, we check both Bloom filters in order to decide the value associated with a query feature vector. If the query feature vector is contained in $B^{(0)}$ and not in $B^{(1)}$, we consider the probability that the associated value is 0 is high and vice versa.

In this research, we utilize the Bloomier filter in order to associate object IDs. Suppose an object ID is represented by v bits, in other words, object ID is converted into *bit ID*, which is a binary representation of an object ID, we prepare the Bloomier filter consisting of $2v$ Bloom filters. In other words, we distinguish 2^v objects with the above Bloomier filter.

4. Proposed Method

We propose a specific object recognition method using the Bloomier filter. Like the conventional method mentioned above, the approach of this method for reduction of the memory usage is to skip the distance calculation for matching feature vectors. Additionally, this method also stores not the information of feature vectors for distance calculation but only the existence of feature vectors in the Bloomier filter. In the following, we explain the concrete process of the proposed method.

4.1 Database Construction

First, let us show how to construct the database with the Bloomier filter. Like the conventional method, the proposed method utilizes feature vectors calculated by PCA-SIFT.

Let $B_1^{(0)}, B_2^{(0)}, \ldots, B_v^{(0)}$ be the Bloom filters whose associated value is 0, and $B_1^{(1)}, B_2^{(1)}, \ldots, B_v^{(1)}$ be the Bloom filters whose associated value is 1. The *Table Size* of each Bloom filter is calculated as:

$$m_\kappa^\tau = a \times n_\kappa^\tau \quad \text{[bit]} \tag{3}$$

where $m_\kappa^\tau, n_\kappa^\tau (\kappa \in \{1,2,\ldots,n\}, \tau \in \{0,1\})$ is the *Table Size* of Bloom filter and the total number of feature vectors, whose κ-th bit of *bit ID* is τ, respectively. a means how many bits are employed for storing one feature vector in the Bloom filter.

By using these setting, first, in order to link the feature vectors and object IDs, the object ID is converted into "*bit ID*". For example, suppose that the number of objects stored in the database is 4. In this case, the object ID 2 is represented as "10". Then, since the 1st bit of *bit ID* is '0' and 2nd bit is '1', the feature vectors extracted from the object whose ID is 2 are stored in the Bloom filter $B_1^{(0)}$ and $B_2^{(1)}$. Similar to the conventional method, we store not the feature vectors but the bit vectors which are calculated by Eq.(1). In the proposed method, 8 hash functions introduced by [12] are utilized for the storage process of the Bloom filter. Additionally, we employ the strategy like "stopword elimination".

4.2 Object Recognition

Next, we explain the recognition process with the Bloomier filter. This process is similar to that of the conventional method. The overview of the process is as follows. First, the *bit ID* is obtained by passing the query feature vector through the Bloomier filter and is converted into the object ID. Then, we simply vote the object. By iterating this process, finally, the object having the maximum number of votes is recognized as the result. In the following, we explain the concrete process of the proposed method.

First, in order to determine whether the $i(i = 1,2,\ldots,n)$-th

Figure 2 Examples of 55 3D objects.

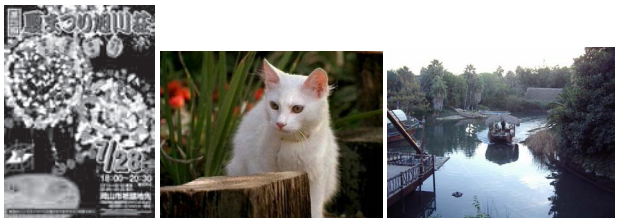


Figure 3 Examples of 5,000 planar objects.

bit of *bit ID* is 0 or 1, we decide the Bloom filters which contain a feature vector $\boldsymbol{p}$ close to a query vector $\boldsymbol{q}$. Recall that a feature vector is converted into a bit vector. Therefore the proposed method also has the same problem as the conventional method. To solve this problem, it also utilizes the strategy of expanded bit vectors. Then, it checks the Bloom filters with the set of bit vectors calculated by the above strategy. If only $B_i^{(0)}(B_i^{(1)})$ contains a query bit vector, the i-th *bit ID* is 0 (1). In the case that a query bit vector is not contained in both $B_i^{(0)}$ and $B_i^{(1)}$, the proposed method discards the query vector. On the other hand, there is also the case that a query bit vector is contained in both $B_i^{(0)}$ and $B_i^{(1)}$. To handle this case, the proposed method creates both *bit ID*s whose i-th bit is 0 and 1. However, unlimited application of this strategy increases the number of *bit ID*s exponentially. To solve this problem, the proposed method employ the limit t of the number of bits for the application. After this process, the *bit ID*s are converted into their object IDs. Then, we vote the objects. Finally, the objects having the maximum number of votes becomes the result of recognition.

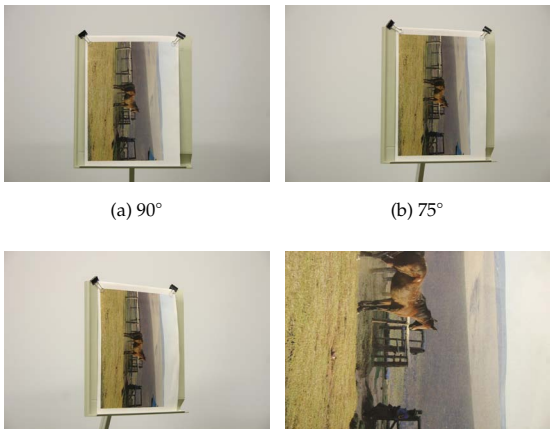
5. Experiments

We have evaluated the proposed method with the following two dataset: 55 3D objects and 5,000 planar objects.

5.1 Experimental Settings

First we explain the dataset of 55 3D objects. We prepared the dataset by ourselves by taking images of 55 objects. Figure 2 shows some examples. The images were captured by rotating each object on a turn table in increments of 5° from frontal view and the above diagonal 15° and 30° by using the web camera. The resolution of these images is 640×480. In these images, we employed the images in increments of $10°(0°, 10°, \ldots, 350°)$ as model images and the rest images were utilized as query images. In total 1.2 million feature vectors were extracted from all model images.

Next, we introduce the dataset of 5,000 planar objects. As model images, we have prepared in total 5,000 images collected from the web site such as Flickr. Some examples are shown in Fig. 3. In the experiment of 5,000 planar objects, we recognize 5,000 planar objects by using 5,000 images. This means the number of model image per object is 1. The average



(a) 90° (b) 75°

(c) 60° (d) Part

Figure 4 Example of query images.

number of feature vectors extracted from an image was about 2,000. The query images were prepared as follows. First, we chose 500 images from the model images randomly. Then these were converted into images by taking their pictures in four different ways as shown in Fig. 4. About 600 feature vectors were obtained on an average from a query image.

The proposed method was compared to the conventional method proposed in [9]. Parameters for each method were set as follows. For both methods, we had three parameters, i.e., the limit b of the number of dimensions for expansion of the original bit vector, the number d of dimensions for generating bit vectors, and the threshold c which is the number of feature vectors converted into same bit vector in the proposed method and is the length of hash chains for deleting. The tested ranges were as follows. $b = 0, 1, \ldots, 10, c = 1, 2, \ldots, 10, d = 24, 28$. We employed a computer with AMD Opteron8378 2.4GHz CPU and 128GB RAM. In the following results, the processing time indicates the average time required for recognition of a single query image excluding the time for extracting feature vectors.

5.2 Experimental Results for 55 3D Objects

First we evaluated the accuracy and the processing time. We tested the proposed method by combining the parameter $a(a = 8, 16, 24, 32)$ and $t(t = 1, 2, \ldots, 5)$ in addition to the parameters mentioned above. Figure 5 and 6 show the experimental results with $d = 24$ and $d = 28$, respectively. As you can see the results, we confirmed that the proposed method needs the longer processing time compared with the conventional

Table 1 Recognition accuracy, processing time and memory usage for 55 3D objects.

Method	c	d	Other parameters	Recognition accuracy[%]	Memory usage[MB]	Processing time[ms]
Proposed method	7	24	$b=2, e=200, t=1$	99.21	72	0.50
	10	28	$b=3, e=200, t=3$	99.45	72	0.50
Conventional method	3	24	$b=2, e=200$	99.11	231	0.09
	2	28	$b=3, e=200$	99.31	2199	0.12

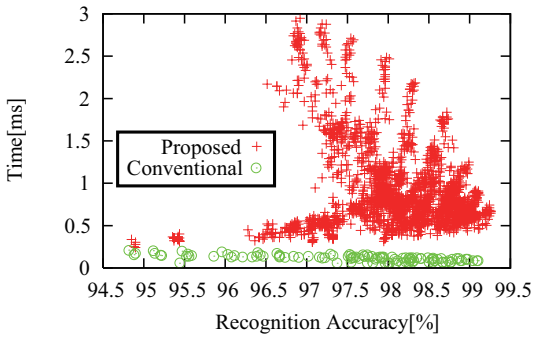


Figure 5 Recognition accuracy and processing time for 55 3D objects ($d = 24$).

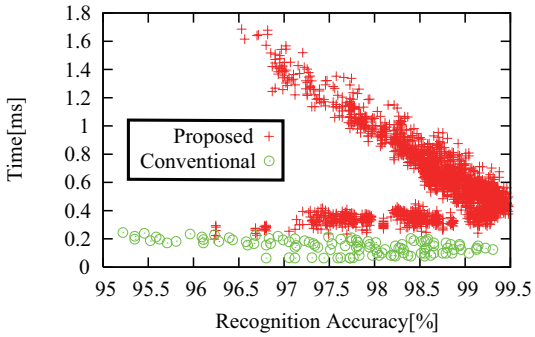


Figure 6 Recognition accuracy and precessing time for 55 3D objects ($d = 28$).

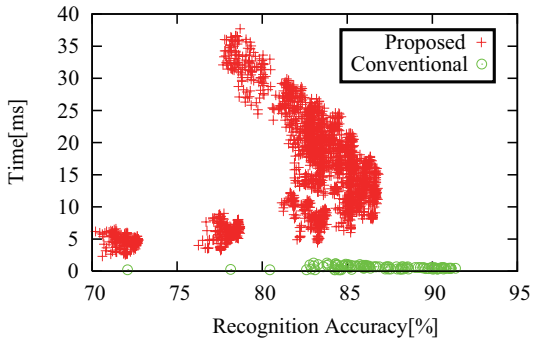


Figure 7 Recognition accuracy and processing time for 5,000 planar objects ($d = 24$).

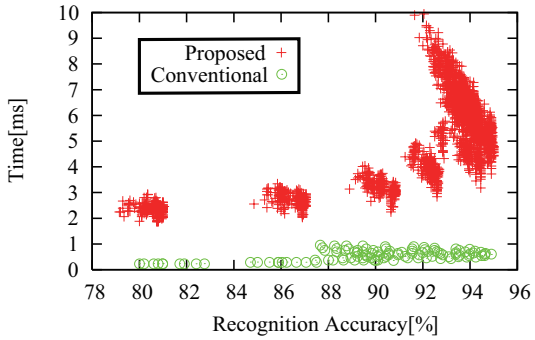


Figure 8 Recognition accuracy and processing time for 5,000 planar objects ($d = 28$).

method. This is because the number of process to determine the object IDs with the proposed method is greater than that with the conventional method. In the conventional method, object IDs are decided by accessing the hash table only once. On the other hand, we need to check 2ν Bloom filters in order to determine the object IDs. Therefore we have considered parallel processing is necessary to speed up the decision of the object IDs.

Next we checked the relationship among the accuracy, processing time and memory usage by changing the parameter c and d. Table 1 shows the experimental results where the recognition accuracy is the best performance for each d with both methods. Note that we utilize $a = 8$ with the proposed method in this investigation. From the experimental results, we have confirmed that the proposed method allows us to recognize the objects with smaller memory usage compared with the conventional method while keeping the high recognition rate.

5.3 Experimental Results for 5,000 Planar Objects

We also evaluated the accuracy and the processing time with 5,000 planar objects. In this experiment, we employed the same tested range of the parameters as that with the previous experiments. Figure 7 and 8 show the experimental results with $d = 24$ and $d = 28$, respectively. From Fig. 7, we confirmed that the proposed method was inferior to the conventional method in the accuracy. This is because bit vectors with $d = 24$ caused many false positives that deteriorated the accuracy of the proposed method. Therefore we confirmed that the proposed method needs to employ a larger d in order to recognize more objects while keeping the high recognition rate. On the other hand, as you can see Fig. 8, we have obtained the results that are similar to that of the previous experiments. From these results, we consider that we need to investigate the value of d that is appropriate for the size of database in the future work.

Next we investigated the relation among the accuracy, processing time and memory usage. In this experiment, we also

Table 2 Recognition accuracy, processing time and memory usage for 5,000 planar objects.

Method	c	d	Other parameters	Recognition accuracy[%]	Memory usage[MB]	Processing time[ms]
Proposed method	10	24	$b=4, e=200, t=1$	86.50	353	10.94
	2	28	$b=6, e=200, t=3$	94.80	352	5.75
Conventional method	6	24	$b=3, e=200$	91.35	479	0.45
	1	28	$b=6, e=200$	94.90	2418	0.59

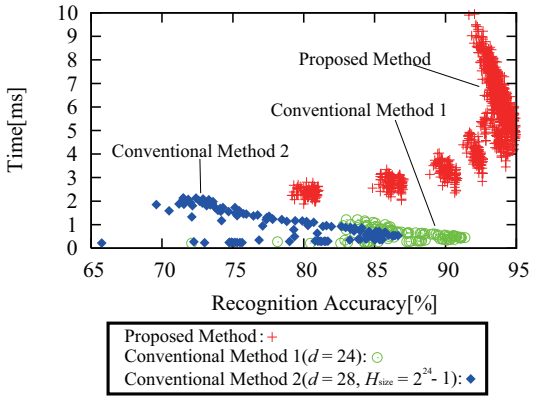


Figure 9 Experimental result with 3 methods.

utilized same tested range of parameters as that with the previous experiments. Table 2 shows the results where the recognition accuracy is the best performance for each d with both methods. From the experimental results, although the performance of the proposed method with $d = 24$ was worse than that of the conventional method, the proposed method with $d = 28$ enabled us to keep the similar recognition rate as well as to obtain a higher space efficiency compared with the conventional method.

Additionally, we researched the relation between the recognition accuracy and processing time where the memory usage of the conventional method was almost equal to that of the proposed method with $d = 28$ by changing the parameter d or hash size. The range of tested parameters for the proposed method were $d = 28$ and $a = 8, 16$. On the other hand, the tested parameters for the conventional method were $d = 24, H_{\text{size}} = 2^{24}$ and $d = 28, H_{\text{size}} = 2^{24} - 1$. We show the experimental results in Fig. 9. From the experimental results, although the longer processing time was necessary for the proposed method compared with the conventional method, the proposed method allowed us to improve the recognition accuracy where the same memory usage. Thus, we confirmed that the proposed method enables us to store the feature vectors effectively compared with the conventional method.

6. Conclusion

In this paper, we have proposed a memory reduction method for specific object recognition by using the Bloomier filter. From the experimental results for 55 3D objects, the proposed method allowed us the recognition rate over 99% with about 1/3 of the memory usage for the conventional method. In the experimental results for 5,000 planar objects, we achieved the recognition rate over 94% with about 3/4 of the memory usage for the conventional method. From these results, it can be said that the proposed method is successful to reduce the required memory for specific object recognition.

Future work is to evaluate the proposed method with more objects, to speed up the processing time and to logically analyze the relationship among the recognition accuracy, the processing time and the required memory.

Acknowledgment This work was supported in part by the Grant-in-Aid for Scientific Research (B) (22300062) and (22 · 8970) from Japan Society for the Promotion of Science (JSPS).

References

[1] http://www.google.com/mobile/goggles/.
[2] L. Yang, R. Jin, R. Sukthankar and F. Jurie: "Unifying Discriminative Visual Codebook Generation with Classifier Training for Object Category Recognition", Proc. of CVPR2008 (2008).
[3] M. Özuysal, M. Calonder, V. Lepetit and P. Fua: "Fast Keypoint Recognition using Random Ferns", IEEE TPAMI Vol. 32, pp. 448–461 (2009).
[4] J. Sivic and A. Zisserman: "Video Google: A Text Retrieval Approach to Object Matching in Videos", Proc. of ICCV2003, pp. 1470–1477 (2003).
[5] Y. Ke and R. Sukthankar: "PCA-SIFT:A more distinctive representation for local image descriptors", Proc. of CVPR2004, Vol. 2, pp. 506–513 (2004).
[6] H. Jégou, M. Douze and C. Schmid: "Packing Bag-of-features", Proc. of ICCV2009, pp. 2357–2364 (2009).
[7] D. Nistér and H. Stewénius: "Scalable Recognition with a Vocabulary Tree", Proc. CVPR2006, Vol. 2, pp. 2161–2168 (2006).
[8] K. Kise, K. Noguchi and M. Iwamura: "Robust and Efficient Recognition of low-Quality Images by Cascaded Recognizers with Massive Local Features", Proc. of WS-LAVD2009, pp. 2125–2132 (2009).
[9] K. Kise, K. Noguchi and M. Iwamura: "Simple representation and approximate search of feature vectors for large-scale object recognition", Proc. 18th British Machine Vision Conference (BMVC2007), **1**, pp. 182–191 (2007).
[10] B. Chazelle, J. Kilian, R. Rubinfeld and A. Tal: "The Bloomier Filter: An Efficient Data Structure for Static Support Lookup Table", Proc. 15th Annual ACM-SIAM SODA, pp. 30–39 (2004).
[11] B. H. Bloom: "Space/Time Trade-offs in Hash Coding with Allowable Errors", Commun. ACM, Vol. 13, No. 7, pp. 422–426 (1970).
[12] General Purpose Hash Function Algorithms: http://www.partow.net/programming/hashfunctions/index.html#RSHashFunction.