Asymmetric Feature Representation for Object Recognition in Client Server System

Yuji Yamauchi[1], Mitsuru Ambai[2], Ikuro Sato[2], Yuichi Yoshida[2], Hironobu Fujiyoshi[1], Takayoshi Yamashita[1]

1 Chubu University, Japan
2 Denso IT Laboratory, Inc., Japan

Abstract. This paper proposes asymmetric feature representation and efficient fitting feature spaces for object recognition in client server system. We focus on the fact that the server-side has more sufficient memory and computation power compared to the client-side. Although local descriptors must be compressed on the client-side due to the narrow bandwidth of the Internet, feature vector compression on the server-side is not always necessary. Therefore, we propose asymmetric feature representation for descriptor matching. Our method is characterized by the following three factors. The first is asymmetric feature representation between client- and server-side. Although the binary hashing function causes quantization errors due to the computation of the sgn function $(\cdot)$, which binarizes a real value into $\{1, -1\}$, such errors only occur on the client-side. As a result, performance degradation is suppressed while the volume of data traffic is reduced. The second is scale optimization to fit two different feature spaces. The third is fast implementation of distance computation based on real-vector decomposition. We can compute efficiently the squared Euclidean distance between the binary code and the real vector. Experimental results revealed that the proposed method helps reduce data traffic while maintaining the object retrieval performance of a client server system.

1 Introduction

Advances in object recognition technology and mobile device technology have enabled the realization of object recognition applications operating in partnership with client server systems. In such applications, a user captures the image of an object with a mobile device and the image or features computed from the image are then sent to a server. On the server, the image is recognized from its features and meta-information about the image is returned to the user. Many such systems operate using local descriptors [1–3], as typified by scale-invariant feature transform (SIFT) [4], which delivers excellent performance in object recognition.

In practice, extracting local descriptors on a client-side and then sending them to a server is problematic, since the data size of local descriptors is too large to transfer through the Internet. For example, the SIFT descriptors is a 128-dimensional vector that consumes 128 bytes when represented as a 1-byte

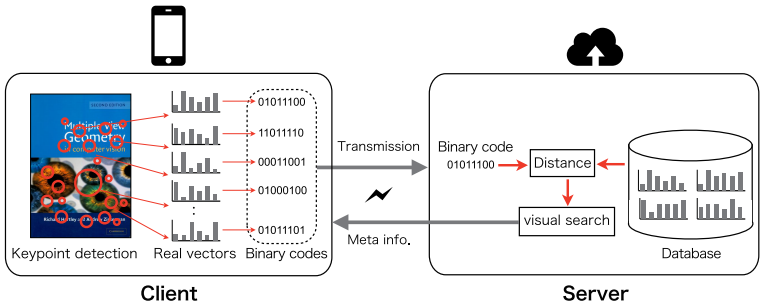

Fig. 1. Our system framework in client server system.

unsigned integer array. Since an image has anywhere from a few hundred to a few thousand SIFT descriptors, the total memory consumption reaches a few hundred KBytes per image. This has created a need to reduce the volume of data traffic sent from the client to the server in consideration of network load.

Chandrasekhar et al. suggested that the volume of data traffic can be reduced by having the client send compressed local descriptors instead of compressed images [5]. Memory-efficient descriptors have been proposed that represent a feature as a binary code, a sequence of binary values $\{-1, 1\}$[1], that can be compactly stored in main memory. BRIEF [7] and its extensions [8–10] generate a binary code by using L pixel pairs chosen from inside a nearby region around a keypoint, which produces an L bits binary sequence. One drawback to these methods is that they produce relatively longer binary codes with lengths ranging from 256 to 512 bits.

While these approaches directly compute a binary code by comparing pixel intensities around a keypoint, binary hashing [11–14, 6, 15] converts a local descriptor represented as a real vector into a much shorter binary code by using a hashing function. In general, binary representation degrades matching performance along with decreasing bit length. Generating a short yet informative feature description remains an open problem.

We focus on the fact that the server-side has more sufficient memory and computation power compared to the client-side. Fig. 1 shows the our framework of object recognition in client server system. Although local descriptors must be compressed on the client-side due to the narrow bandwidth of the Internet, feature vector compression on the server-side is not always necessary. Therefore, we propose asymmetric feature representation for descriptor matching. Our method consisted of three factors. The first is asymmetric feature representation between client- and server-side. The second is scale optimization to fit two different feature spaces. The third is fast implementation of distance computation based on real-vector decomposition.

[1] In this paper, as with [6], a binary code is expressed by $\{-1, 1\}$ instead of $\{0, 1\}$ in order to simplify the mathematical expressions.

1.1 Related works

Feature representation Since the SIFT was shown to be effective for keypoint matching, much research has focused on finding ways to reduce the keypoint matching computation time for use on mobile devices. Bay et al. proposed the Speeded-Up Robust Features (SURF) [1], which achieves high-speed computation by using integral images. Takacs et al. developed the Rotation-Invariant, Fast Feature (RIFF) descriptor [2] for achieving Mobile Augmented Reality (MAR) on mobile devices.

On the other hand, global descriptor[16, 17] besides local descriptor also have been proposed. Global descriptor such as VLAD [17], computational cost is large because it requires further computation after extracting local descriptor. Furthermore, the approach based on global descriptor, there is a disadvantage that the adoption of the strategy of improving performance by re-ranking using location information of the keypoints is difficult [18]. Therefore, we adopt a local descriptor at client for transmitting to server, as shown in Fig. 1.

These local descriptors require much memory, in order to represent as a high-dimensional real vector. Several methods in which local descriptors are extracted as binary code have been proposed for overcoming these problems [7–10]. These approaches generate binary code on the basis of the relation between the intensity of two pixels around each keypoint. One drawback to these methods is that they produce relatively longer binary codes with lengths ranging from 256 to 512 bits.

While these approaches directly compute a binary code by comparing pixel intensities around a keypoint, binary hashing [11–14, 6, 15] converts a local descriptor represented as a real vector into a much shorter binary code by using a hashing function. In this approach, a feature vector $\mathbf{x} \in \mathbb{R}^D$ is mapped into short binary code $\mathbf{b} \in \{-1, 1\}^L$ by using a binary hashing function $\mathbf{b} = \mathrm{sgn}(f(\mathbf{W}^{\mathrm{T}}\mathbf{x}))$, where D is a dimension, L is the bit length of the binary code, and $\mathbf{W} \in \mathbb{R}^{D \times L}$ is a weight matrix. Weight matrix $\mathbf{W}$ and function $f(\cdot)$[2] characterize each binary hashing. The simplest method proposed is random projection (RP) [11]. $f(\cdot)$ is an identity function, and the elements in $\mathbf{W}$ are sampled from a normal distribution in random projection. The Very Sparse Random Projections (VSRP) [12] was developed to speed up the calculation. It does this by limiting each element in $\mathbf{W}$ to $\{-1, 0, 1\}$. The very supervised sparse hashing (VSSH) [13] improves performance by introducing the concept of learning to VSRP. The VSSH has been proposed that learns each element of $\mathbf{W}$. Spectral hashing (SH) [14] uses principal directions of training data for $\mathbf{W}$ and applies a cosine-like function for $f(\cdot)$. The iterative quantization (ITQ) [6] defines $\mathbf{W} = \mathbf{W}_{PCA}\mathbf{R}$, where $\mathbf{W}_{PCA}$ is obtained by principal component analysis, and rotation matrix $\mathbf{R}$ is optimized to minimize the quantization error before and after binary code conversion. In general, binary representation degrades matching performance along with decreasing bit length L. Generating a short yet informative feature description remains an open problem.

[2] In many cases [11–13, 6, 19, 15], $f(\cdot)$ is an identity function.

Domain adaptation Domain adaptation is often used for fitting two or more feature spaces. Domain adaptation aims to learn classifier using a large number of samples in the source domain and a small number of samples or classifier in target domain. In order to achieve this, integrating feature spaces [20], adaptation of classifier [21][22], and fitting two feature spaces [23][24][25] have been proposed. The proposed method is related to domain adaptation for fitting two feature spaces.

Saenko et. al proposed a method for learning metric using Information-Theoretic Metric Learning (ITML)[26] to form a feature space in consideration of relationships between different domains [23]. High accuracy object recognition is achieved by using of k-Nearest Neighbor on the learned feature space. In addition, it is extended to non-linear mapping to apply the kernel method [25]. Geng et al. proposed Domain Adaptation Metric Learning (DAML) which learned a metric in reproducing kernel Hilbert space[24]. A common point of these methods is to learn a metric while considering the relations of a domains and the target categories.

The proposed method is also related to methods for fitting two feature spaces. However, a local descriptor of the same domain is represented by two feature spaces in the proposed method. The proposed method transforms the local descriptor $\mathbf{x} \in \mathbb{R}^D$ to short real vector $\mathbf{y} \in \mathbb{R}^L$, and it is transformed binary code $\mathbf{b} \in \{-1, 1\}^L$ by sgn $(\cdot)$ function. We obtain binary code and real vector from a same local descriptor. Since this is our specific problem, method to solve it has not been proposed.

1.2 Overview of our approach

In this paper, we focus on the fact that the server-side has more sufficient memory and computation power compared to the client-side. Although local descriptors must be compressed on the client-side due to the narrow bandwidth of the Internet, feature vector compression on the server-side is not always necessary. Therefore, we propose asymmetric feature representation for descriptor matching. Our method is characterized by the following three factors.

1. *Asymmetric feature representation*
   In our approach, local descriptors are computed on the client-side and converted into short binary codes, and the server stores local descriptors as real vectors. Although the binary hashing function causes quantization errors due to the computation of the sgn function $(\cdot)$, which binarizes a real value into $\{1, -1\}$, such errors only occur on the client-side. As a result, performance degradation is suppressed while the volume of data traffic is reduced.
2. *Defining distances between binary codes and real vectors*
   Since the feature space of the real vectors is different from that of the binary codes, they cannot be directly compared. We propose a simple method to scale one feature space to fit the other feature space that enables the computation of distances between such asymmetrically represented features.

3. *Fast implementation for computing distances*
   It has already been reported that by decomposing a real vector into a few scholar weight factors and a few binary basis vectors, the Euclidean distance between the binary code and the real vector can be computed extremely quickly [27]. We propose a decomposition method based on alternative optimization strategies that can approximate the real vector with fewer basis vectors than [27].

2 Asymmetric representation and distance

This section describes asymmetric feature representation and defining distances between binary codes and real vectors.

2.1 Euclidean distance between binary code and real vector

The binary hashing consists of two steps. First, an input vector $\mathbf{x} \in \mathbb{R}^{D}$[3] is converted into a short real vector $\mathbf{y} \in \mathbb{R}^{L}$ by

$$\mathbf{y} = f(\mathbf{W}^{\mathrm{T}}\mathbf{x}). \tag{1}$$

Second, a binary code $\mathbf{b} \in \{-1, 1\}^{L}$ is computed by

$$\mathbf{b} = \mathrm{sgn}(f(\mathbf{W}^{\mathrm{T}}\mathbf{x})). \tag{2}$$

In our framework, any conventional binary hashing method is available for use. The $f(\cdot)$ and the $\mathbf{W} \in \mathbb{R}^{\mathbf{D}\times\mathbf{L}}$ follow the definitions of conventional binary hashing methods.

As shown in [6], the $\mathrm{sgn}(\cdot)$ function used in the binary hashing can cause quantization errors that may degrade matching performance. Therefore, in our approach, while local descriptors computed on the client-side are converted into short binary codes $\mathbf{b}$, the server stores local descriptors as real vectors $\mathbf{y}$. As a result, since quantization errors only occur on the client-side, performance degradation is suppressed while the volume of data traffic is reduced. However, since each feature space is different, they cannot be directly compared.

2.2 Optimization of adjustment matrix

Since two feature spaces are different, it is necessary to fit the other feature space that enables the computation of distances between such asymmetrically represented features. An approach using the domain adaptation [23][24][25] introduced the metric learning for fitting feature spaces in different domains. These methods proposed introducing metric learning to form a feature space in consideration of relationships between different domains. However, a local descriptor of

[3] Before the local features are converted, they are mean-centered by using an average descriptor which is computed from training samples.

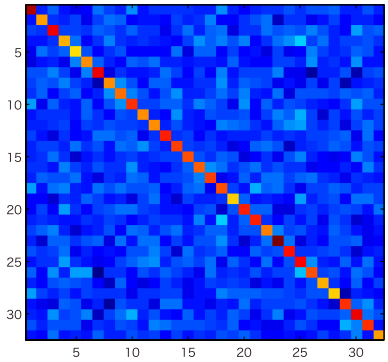

Fig. 2. Visualization of adjustment matrix ($L = 32$ bits). The red color means a high value.

Table 1. Euclidean norm and scale factor in each binary hashing method.

Binary hashing	Average Euclidean norm	Scale factor
Binry code	5.66	–
RP [11]	4.22	1.09
VSRP [12]	1.32	3.41
SH [14]	2.46	1.86
ITQ [6]	0.65	6.17

the same domain is represented by two feature spaces in the proposed method. We obtain binary code and real vector from a same local descriptor. Therefore, in order to minimize the error between real vectors and binary codes, we optimize the objective function as follows :

$$J(\mathbf{Q}) = ||\mathbf{B} - \mathbf{QY}||_F^2, \tag{3}$$

where $\mathbf{B} \in \{-1, 1\}^{L \times N}$ is a matrix of binary code corresponding to N keypoints obtained from training images. The matrix of real vectors $\mathbf{Y} \in \mathbb{R}^{L \times N}$ is similar. $\mathbf{Q} \in \mathbb{R}^{L \times L}$ is a adjustment matrix for fitting the two feature spaces. Two feature spaces can be fitted using optimized adjustment matrix.

Fig. 2 shows example of visualization of optimized adjustment matrix. The adjustment matrix is very similar to the diagonal matrix, and diagonal elements have high value. This means that the optimized adjustment matrix can be replaced as a scalar matrix. Furthermore, the size of the adjustment matrix $\mathbf{Q}$ is a square of the bit length L of the binary code. If the bit length is greater, there is a possibility that the matrix becomes over-fit to the training samples. Therefore, we fit two feature spaces by more simpler way.

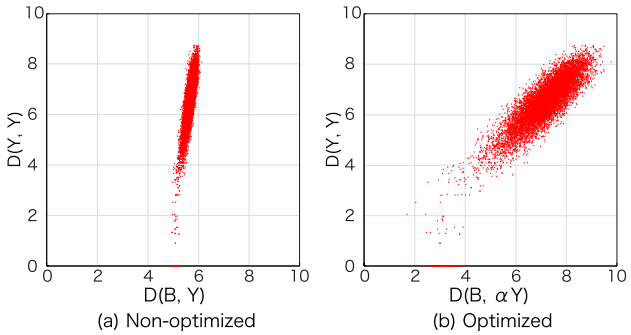

Fig. 3. Effect of optimizing scale factor. $D(\cdot, \cdot)$ is Euclidean distance.

2.3 Optimization of scale factor

We propose a simple method to scale one feature space to fit the other feature space and enable the computation of distances between such asymmetrically represented features. We introduce the optimization of scale factor α to absorb the scale difference between feature spaces. Optimization is done using a cost function:

$$J(\alpha) = ||\mathbf{B} - \alpha\mathbf{Y}||_F^2, \tag{4}$$

where $\mathbf{B} \in \{-1, 1\}^{L \times N}$ is a matrix of binary code $\mathbf{b}$ corresponding to N keypoints obtained from training images. The matrix of real vectors $\mathbf{Y} \in \mathbb{R}^{L \times N}$ is a matrix of real vector $\mathbf{y}$.

The Euclidean norms of binary code and real vectors when L is set to 32 are shown in Table 1 for several binary hashing methods. The Euclidean norm of a binary code $\mathbf{b}$ is a constant value $\sqrt{L}$ because the elements of the binary code only take two integer values $\{-1, 1\}$. In contrast, the Euclidean norm of a real vector depends on the binary hashing method. This difference may significantly degrade matching performance. Table 1 shows the optimized scale factors for each binary hashing method. The smaller the Euclidean norm of the real vector, the larger the scale factor becomes. The effect of the optimization of the scale factor is shown in Fig. 3. We used the ITQ binary hashing method. Without optimization (Fig. 3(a)), the distribution of the Euclidean distance between the binary code and real vectors was biased compared with the distribution of the Euclidean distance between real vectors. This is because the Euclidean norm of a real vector is very small compared to that of a binary code. With optimization (Fig. 3(b)), the Euclidean distances were about the same. For brevity purposes, $\mathbf{y}_\alpha = \alpha\mathbf{y}$ is used hereafter.

3 Fast computation of Euclidean distance by introducing decomposition method

This section describes fast computation of Euclidean distance by introducing decomposition method.

3.1 Real vector decomposition

In this section, we consider the efficient computation of squared Euclidean distance between the binary code $\mathbf{b}$ and the real vector $\mathbf{y}_\alpha$. This computation can be expanded as

$$\begin{aligned} d(\mathbf{b}, \mathbf{y}_\alpha) &= ||\mathbf{b} - \mathbf{y}_\alpha||_2^2 \\ &= \mathbf{b}^{\mathrm{T}}\mathbf{b} - 2\mathbf{b}^{\mathrm{T}}\mathbf{y}_\alpha + \mathbf{y}_\alpha^{\mathrm{T}}\mathbf{y}_\alpha. \end{aligned} \tag{5}$$

The first term of Eq. (5) is the dot product between binary codes in the client. This becomes a constant value because all of the elements in $\mathbf{b}$ take only two values $\{1, -1\}$. The third term is the dot product between real vectors stored in the server that can be calculated in advance. The problem here is computing the second term: it cannot be calculated in advance, so it requires a large number of floating-point computations.

To overcome this problem, Hare et al. [27] proposed decomposing the real vector $\mathbf{y}_\alpha$ into k weight factors and k binary basis vectors as

$$\mathbf{y}_\alpha \approx \mathbf{M}\mathbf{c}, \tag{6}$$

where $\mathbf{c} = (c_1, c_2, \cdots, c_k)^{\mathrm{T}} \in \mathbb{R}^k$ is the weight factor and $\mathbf{M} = \{\mathbf{m}_1, \mathbf{m}_2, \cdots, \mathbf{m}_k\} \in \{-1, 1\}^{L \times k}$ is the binary matrix composed of k binary basis vectors $\mathbf{m}_i \in \{-1, 1\}^k$.

Letting Eq. (6) into the second term of Eq. (5), we obtain

$$\begin{aligned} \mathbf{b}^{\mathrm{T}}\mathbf{y}_\alpha &\approx \mathbf{b}^{\mathrm{T}}\mathbf{M}\mathbf{c} \\ &= \sum_{i=1}^{k} c_i \mathbf{b}^{\mathrm{T}}\mathbf{m}_i. \end{aligned} \tag{7}$$

The computations $\mathbf{b}^{\mathrm{T}}\mathbf{m}_i$ that appeared in Eq. (7) are extremely fast because this is equivalent to computing the Hamming distance between $\mathbf{b}$ and $\mathbf{m}_i$, as

$$\mathbf{b}^{\mathrm{T}}\mathbf{m}_i = L - 2\mathrm{HammingDistance}(\mathbf{b}, \mathbf{m}_i) \tag{8}$$

Since the Hamming distance can be computed efficiently using a bitwise XOR followed by a bit-count, Eq. (7) can also be computed very fast.

Introducing the decomposition method provides one more advantage. The server only has to store $\mathbf{c}$, $\mathbf{M}$, and $\mathbf{y}_\alpha^{\mathrm{T}}\mathbf{y}_\alpha$ instead of $\mathbf{y}_\alpha$. This reduces memory usage in the server substantially.

3.2 Decomposition algorithms

The rest of this section discusses the decomposition algorithms used to obtain $\mathbf{M}$ and $\mathbf{c}$. Hare et al. [27] proposed a greedy algorithm that sequentially determines pairs of c_i and $\mathbf{m}_i$ one after another. In contrast to this, we propose a decomposition method based on an alternative optimization strategy that can

Algorithm 1 Decomposition.

for $i = 1 : I$ **do**
  Set $\mathbf{c}$ and $\mathbf{M}$ to random values.
  for $j = 1 : \infty$ **do**
    (1) Minimize $J(\mathbf{c}, \mathbf{M})$ by fixing $\mathbf{M}$ and updating $\mathbf{c}$.
      This optimization can be done by least squares method.
    (2) Minimize $J(\mathbf{c}, \mathbf{M})$ by fixing $\mathbf{c}$ and updating $\mathbf{M}$.
      This optimization can be done by exhaustive search.
    (3) Exit loop if converged.
  end for
end for
Select the best $\mathbf{c}$ and $\mathbf{M}$ that minimize $J(\mathbf{c}, \mathbf{M})$.

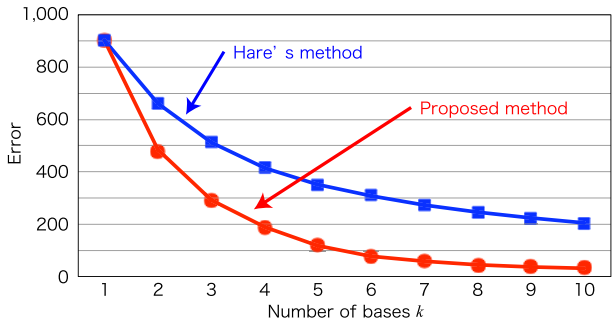

Fig. 4. Comparison of the proposed method with Hare's method [27].

approximate $\mathbf{y}_\alpha$ with fewer weights c_i and basis vectors $\mathbf{m}_i$ than Hare's greedy optimization.

In our approach, $\mathbf{M}$ and $\mathbf{c}$ are determined by minimizing the following cost function:

$$J(\mathbf{c}, \mathbf{M}) = ||\mathbf{y}_\alpha - \mathbf{M}\mathbf{c}||_2^2. \tag{9}$$

Our decomposition algorithm is shown in **Algorithm** 1. Since it is difficult to optimize $\mathbf{M}$ and $\mathbf{c}$ at the same time, we do so alternately. If the basis vector $\mathbf{M}$ is fixed, the weight factor $\mathbf{c}$ can be optimized by using the least squares method. In contrast, if $\mathbf{c}$ is fixed, the basis vector $\mathbf{M}$ can be optimized by exhaustive search. Thanks to the constraint that the binary basis vector $\mathbf{M}$ only takes two integer values $\{1, -1\}$, the i th row in the matrix $\mathbf{M}$ takes 2^k combinations. Therefore, all of the 2^k combinations can be exhaustively tested if k is small enough. We initialize $\mathbf{M}$ and $\mathbf{c}$ by random values and alternately update them until convergence. To avoid falling to a local minimum, several different initial values are tested.

In contrast to Hare's method, our method determines k pairs of c_i and $\mathbf{m}_i$ simultaneously. Therefore, the difference between the two methods appears in the approximate performance. It is obvious from Fig. 4 that our method can approximate $\mathbf{y}_\alpha$ using fewer basis vectors $\mathbf{m}_i$ than Hare's method.

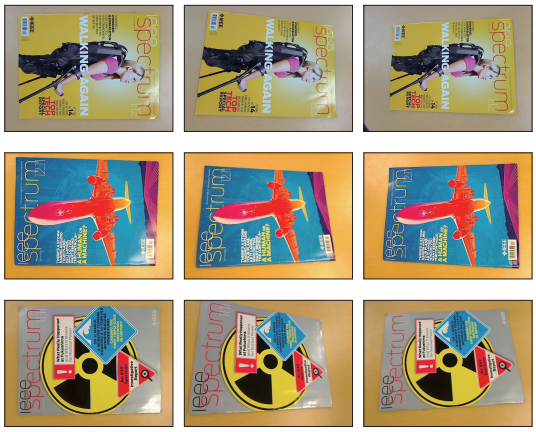

Fig. 5. Example images in dataset.

4 Experiments

We evaluated the performance of the asymmetric feature representation proposed in this paper by testing to find corresponding points between two images.

4.1 Datasets for evaluating keypoint matching

We evaluate the proposed method on two datasets of the IEEE Spectrum magazine dataset and Mikolajczyk's dataset [3].

IEEE Spectrum magazine dataset We prepared seven issues of the IEEE Spectrum magazine and captured them from six viewpoints. Example images from some various viewpoints are shown in Fig. 5. Let I_j^i denote an image in the database, with the number of where i is the magazine index ($1 \leq i \leq 7$), and j is the viewpoint index ($1 \leq j \leq 6$). Assuming that the magazines are planar, we prepared a homography matrix $\mathbf{H}_{1\to j}^i$ between I_1^i and I_j^i in advance, which gives ground truth correspondences between the image pairs. We used $I_j^1 \sim I_j^3$ for training to compute the weight matrix $\mathbf{W}$, and used $I_j^4 \sim I_j^7$ for testing.

Keypoint matching performance can be evaluated by using the image pairs I_1^i, I_j^i, and their homography matrix $\mathbf{H}_{1\to j}^i$. For each keypoint obtained from the I_1^i, the first and second nearest neighbors were searched from the keypoints extracted from the I_j^i. Let d_1 and d_2 denote the distances to the first and second nearest neighbors, respectively. If the ratio of the distances d_1/d_2 was less than a pre-defined threshold T, the query keypoint in I_1^i and the first nearest neighbor in I_j^i were regarded as corresponding points. If the first nearest neighbor was located within $\sqrt{(1+1)}$ pixels from the true location derived from $\mathbf{H}_{1\to j}^i$, such keypoint pair is regarded as inlier. We computed the average number of matches over test image pairs and the rate of correct matching.

Mikolajczyk's dataset We evaluate the proposed method using the standard dataset [3] for keypoint matching. Mikolajczyk's dataset is captured at eight

places from six viewpoints. We use images from four viewpoints for training to compute weight matrix $\mathbf{W}$, and used the remaining images for testing.

4.2 Comparing symmetric and asymmetric representation

We compared three kinds of feature representation:

- *Binary code vs. Binary code: BC-BC*
  This is conventional symmetric representation.
- *Binary code vs. Real vector: BC-RV without optimizing α*
  This is the asymmetric representation proposed in this paper. The scaling factor α is fixed to 1.
- *Binary code vs. Real vector: BC-RV with optimizing α*
  The scaling factor α was optimized by using training samples.

We used SIFT [4] as local descriptors, and we test four binary hashing functions as follow:

- Random Projection (RP) [11]
  $f(\cdot)$ was used as the identity function, and elements in $\mathbf{W}$ were sampled from a normal distribution.
- Very Sparse Random Projection (VSRP) [12]
  $f(\cdot)$ was used as the identity function, and elements in $\mathbf{W}$ were limited to $\{-1, 0, 1\}$, with probability $\{\frac{1}{2\sqrt{(D)}}\}, 1 - \{\frac{1}{\sqrt{(D)}}\}, \{\frac{1}{2\sqrt{(D)}}\}$.
- Spectral Hashing (SH) [14]
  Basis vectors obtained by principal component analysis were used as $\mathbf{W}$, and $f(\cdot)$ was used as an eigenfunction.
- Iterative Quantization (ITQ) [6]
  ITQ was used to define $\mathbf{W} = \mathbf{W}_{PCA}\mathbf{R}$, $\mathbf{W}_{PCA}$ was obtained by principal component analysis, and rotation matrix $\mathbf{R}$ was optimized to minimize the quantization error before and after binary code conversion.

The results are shown in Fig. 6. The asymmetric representation with optimizing α clearly outperformed the conventional symmetric representation when the shorter binary codes were used. This means that the proposed method can improve matching performance when short binary codes are used to reduce network traffic. The scaling factor α played an important role when ITQ and VSRP were used as the binary hashing function. In the case of ITQ and VSRP, the average Euclidean norm of binary codes in the client-side significantly differed from that of real vectors in the server-side, as shown in Table 1. This means that the scaling factor α absorbed such difference and contributed to improving the matching performance.

4.3 Effect of decomposition

We evaluated the effect of decomposition in terms of matching rate, computational time, and memory usage. In this experiment, random projections were used as binary hashing methods. Bit length L was set to 32.

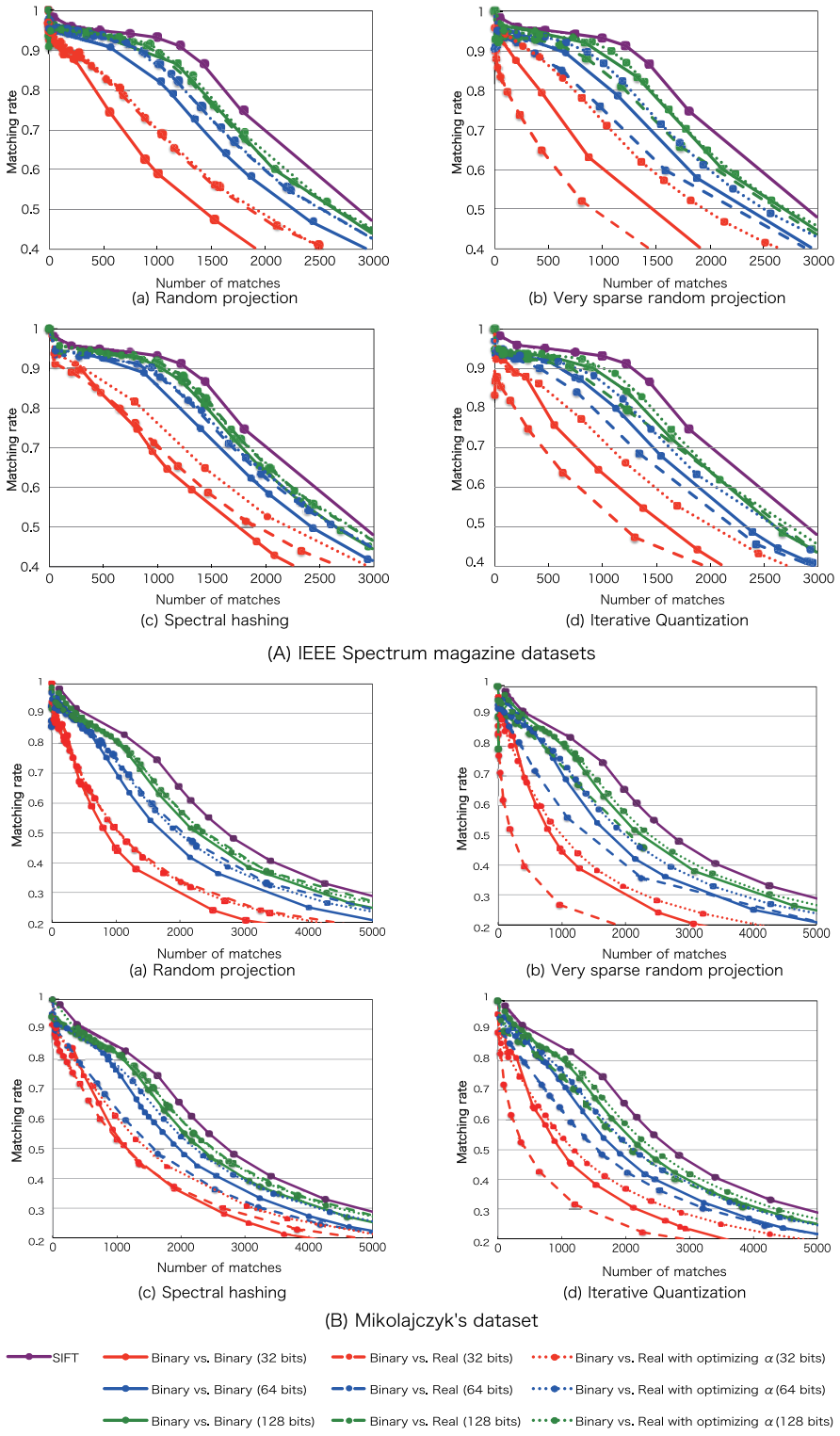

Fig. 6. Comparison with state-of-the-art methods.

Table 2. Computational time [ns].

Bits	BC - BC	BC - RV (No Decomposition)	BC - RV (Decomposition)
32	14.2	433.5	54.5
64	27.6	852.3	105.7
128	61.9	1683.2	177.5

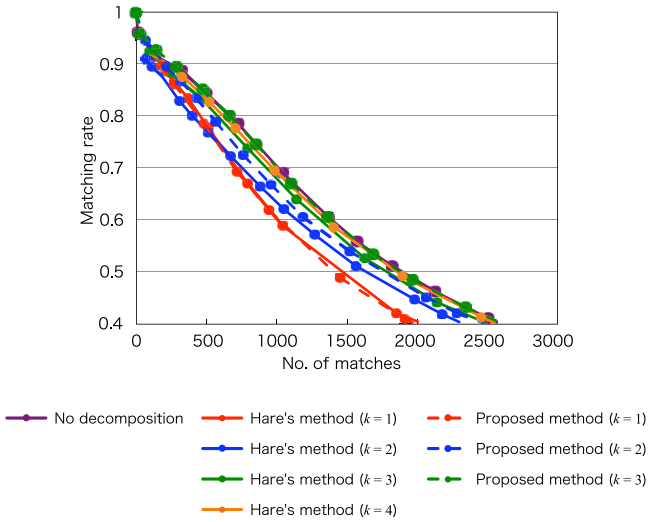

Fig. 7. Comparison with Hare's method [27].

Matching performance The results of a comparison between our decomposition algorithm and Hare's method [27] are shown in Fig. 7. When the number of basis vectors k was set to 1, there was little difference between our algorithm and Hare's method. However, when $k > 1$, the performance of the proposed method was higher. While Hare's method needed four basis vectors to sufficiently approximate the original real vector $\mathbf{y}_\alpha$, our algorithm only required three, which helped reduce the memory usage and computational time of matching.

Computational time We evaluated the computational time of keypoint matching. We used an Intel Xeon CPU 2.27-GHz processor. The number of basis vectors k was set to 3. As shown in Table 2, introducing the decomposition method drastically reduced the computational time compared to the case without decomposition: the computation time was eight times faster.

Memory usage We compared memory usage with and without decomposing $\mathbf{y}_\alpha$. The number of basis vectors k was set to 3. Table 3 shows the results. One

Table 3. Memory usage in server [MB]. (We assume that 1,000 keypoints are detected from an image.)

No. of keypoints	32 bits		64 bits		128 bits	
(No. of images)	Dec.	No dec.	Dec.	No dec.	Dec.	No dec.
0.1M (0.1K)	2.67	12.2	3.8	24.4	6.1	48.8
1 M (1 K)	26.7	122.1	38.1	244.1	61.0	488.3
10 M (10 K)	267.0	1,220.7	381.5	2,441.4	610.4	4,882.8
100 M (100 K)	2,670.3	12,207.0	3,814.7	24,414.1	6,103.5	48,828.1

example shows that if the server stores 100,000 images and the length of the binary code is set to 32 bits, memory usage is reduced to about 21%.

5 Conclusion

In this paper, we proposed asymmetric feature representation for matching local descriptors. Experimental results revealed that the proposed method helps reduce data traffic while maintaining the object retrieval performance of a client server system. Our method consisted of three factors. The first was asymmetric feature representation between client- and server-side. The second was scale optimization to fit two different feature spaces. The third was fast implementation of distance computation based on real-vector decomposition.

The range of application is not limited to object retrieval. We believe our method can be used not only for computer vision applications but also for similar applications such as speech recognition systems.

References

1. Bay, H., Ess, A., Tuytelaars, T., Van Gool, L.: Speeded-up robust features (surf). Computer Vision and Image Understanding **110** (2008) 346–359
2. Takacs, G., Chandrasekhar, V., Tsai, S., Chen, D., Grzeszczuk, R., Girod, B.: Unified real-time tracking and recognition with rotation-invariant fast features. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition. (2010)
3. Mikolajczyk, K., Schmid, C.: A performance evaluation of local descriptors. IEEE Transactions on Pattern Analysis & Machine Intelligence **27** (2005) 1615–1630
4. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision **60** (2004) 91–110
5. Chandrasekhar, V., Takacs, G., Reznik, Y., Grzeszczuk, R., Girod, B.: Compressed histogram of gradients: A low-bitrate descriptor. International Journal of Computer Vision (2011)
6. Gong, Y., Lazebnik, S.: Iterative Quantization : A Procrustean Approach to Learning Binary Codes. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition. (2011)

7. Calonder, M., Lepetit, V., Ozuysal, M., Trzcinski, T., Strecha, C., Fua, P.: BRIEF: Computing a Local Binary Descriptor Very Fast. IEEE Transactions on Pattern Analysis & Machine Intelligence **34** (2012) 1281–1298
8. Leutenegger, S., Chli, M., Siegwart, R.: BRISK: Binary Robust Invariant Scalable Keypoints. In: IEEE International Conference on Computer Vision. (2011)
9. Rublee, E., Rabaud, V., Konolige, K., Bradski, G.: ORB: An Efficient Alternative to SIFT or SURF. In: IEEE International Conference on Computer Vision. (2011)
10. Alahi, A., Ortiz, R., Vandergheynst, P.: FREAK: Fast Retina Keypoint. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition. (2012)
11. Achlioptas, D.: Database-friendly random projections: Johnson-Lindenstrauss with binary coins. Journal of Computer and System Sciences **66** (2003) 671–687
12. Li, P., Hastie, T.J., Church, K.W.: Very Sparse Random Projections. In: International Conference on Knowledge discovery and data mining. (2006)
13. Ambai, M., Yoshida, Y.: CARD: Compact And Real-time Descriptors. In: IEEE International Conference on Computer Vision. (2011) 97–104
14. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: Neural Information Processing Systems. (2008) 1753–1760
15. Heo, J.P., Lee, Y., He, J., Chang, S.F., Yoon, S.e.: Spherical hashing. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition. (2012)
16. Csurka, G., Dance, C.R., Fan, L., Willamowski, J., Bray, C.: Visual categorization with bags of keypoints. In: In Workshop on Statistical Learning in Computer Vision, in conjunction European Conference on Computer Vision. (2004)
17. Jégou, H., Douze, M., Schmid, C., Pérez, P.: Aggregating local descriptors into a compact image representation. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition. (2010) 3304–3311
18. Philbin, J., Chum, O., Isard, M., Sivic, J., Zisserman, A.: Object retrieval with large vocabularies and fast spatial matching. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition. (2007)
19. Weiss, Y., Fergus, R., Torralba, A.: Multidimensional spectral hashing. In: European Conference on Computer Vision. (2012)
20. Daume III, H.: Frustratingly easy domain adaptation. In: Annual Meeting of the Association of Computational Linguistics. (2007) 256–263
21. Duan, L., Tsang, I.W., Xu, D., Chua, T.S.: Domain adaptation from multiple sources via auxiliary classifiers. In: Annual International Conference on Machine Learning. (2009) 289–296
22. Yang, J., Yan, R., Hauptmann, A.G.: Cross-domain video concept detection using adaptive svms. In: International Conference on Multimedia. (2007) 188–197
23. Saenko, K., Kulis, B., Fritz, M., Darrell, T.: Adapting visual category models to new domains. In: European Conference on Computer Vision. (2010) 213–226
24. Geng, B., Tao, D., Xu, C.: Daml : Domain adaptation metric learning. IEEE Transactions on Image Processing **20** (2011) 2980–2989
25. Kulis, B., Saenko, K., Darrell, T.: What you saw is not what you get: Domain adaptation using asymmetric kernel transforms. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition. (2011) 1785–1792
26. Davis, J.V., Kulis, B., Jain, P., Sra, S., Dhillon, I.S.: Information-theoretic metric learning. In: International Conference on Machine Learning. (2007) 209–216
27. Hare, S., Saffari, A., Torr, P.H.S.: Efficient Online Structured Output Learning for Keypoint-Based Object Tracking. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition. (2012) 1894–1901