IPSJ Transactions on Computer Vision and Applications Vol.5 124–128 (July 2013)

[DOI: 10.2197/ipsjtcva.5.124]

Express Paper

Distance Computation Between Binary Code and Real Vector for Efficient Keypoint Matching

Yuji Yamauchi[1,a)] Mitsuru Ambai[2,b)] Ikuro Sato[2,c)] Yuichi Yoshida[2,d)]
Hironobu Fujiyoshi[1,e)]

Received: March 11, 2013, Accepted: April 24, 2013, Released: July 29, 2013

Abstract: Image recognition in client server system has a problem of data traffic. However, reducing data traffic gives rise to worsening of performance. Therefore, we represent binary codes as high dimensional local features in client side, and represent real vectors in server side. As a result, we can suppress the worsening of the performance, but it problems of an increase in the computational cost of the distance computation and a different scale of norm between feature vectors. Therefore, to solve the first problem, we optimize the scale factor so as to absorb the scale difference of Euclidean norm. For second problem, we compute efficiently the Euclidean distance by decomposing the real vector into weight factors and binary basis vectors. As a result, the proposed method achieves the keypoint matching with high-speed and high-precision even if the data traffic was reduced.

Keywords: asymmetric feature, client server system, binary hashing, real vector decomposition, keypoint matching

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 are returned to the user. Many such systems operate using local features [4], [13], [15], as typified by scale-invariant feature transform (SIFT) [12], which delivers an 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 feature is a 128-dimensional vector that consumes 128 bytes when represented as a 1-byte unsigned integer array. Since an image has anywhere from a few hundred to a few thousand SIFT features, 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 the 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 [6]. 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 the main memory. BRIEF [5] and its extensions [2], [10], [14] 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 [1], [3], [7], [9], [11], [16] 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 a 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. $\mathbf{W}$ and $f(\cdot)$[*2] characterize each binary hashing. In general, the binary representation the degrades the matching performance along with decreasing bit length L. Generating a short yet informative feature description remains an open problem.

1.1 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 fol-

1 Chubu University, Kasugai, Aichi 487–8501, Japan
2 Denso IT Laboratory, Inc., Shibuya, Tokyo 150–0002, Japan
a) yuu@vision.cs.chubu.ac.jp
b) manbai@d-itlab.co.jp
c) isato@d-itlab.co.jp
d) yyoshida@d-itlab.co.jp
e) hf@cs.chubu.ac.jp

*1 In this paper, as with Ref. [7], a binary code is expressed by $\{-1, 1\}$ instead of $\{0, 1\}$ in order to simplify the mathematical expressions.
*2 In many cases [1], [3], [7], [11], $f(\cdot)$ is an identity function.

lowing 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, 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 [8]. We propose a decomposition method based on alternative optimization strategies that can approximate the real vector with fewer basis vectors than Ref. [8].

2. Asymmetric Representation and Distance

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 $\mathbf{y} = f(\mathbf{W}^{\mathrm{T}}\mathbf{x})$. Second, a binary code $\mathbf{b} \in \{-1, 1\}^L$ is computed by $\mathbf{b} = \mathrm{sgn}(f(\mathbf{W}^{\mathrm{T}}\mathbf{x}))$. In our framework, any conventional binary hashing method is available for use. The $f(\cdot)$ and the $\mathbf{W}$ follow the definitions of conventional binary hashing methods.

As shown in Ref. [7], the sgn(·) function used in the binary hashing can cause quantization errors that may degrade the 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 the scale of each feature space is different, they cannot be directly compared. 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 the matching performance.

2.2 Optimization of Scale Factor

To solve this problem, 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. The optimization is done using a cost function:

$$J(\alpha) = \|\mathbf{B} - \alpha\mathbf{Y}\|_F^2, \tag{1}$$

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.

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. 1**. We used the ITQ binary hashing method. Without optimization (Fig. 1 (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. 1 (b)), the Euclidean distances were about the same. For brevity purposes, $\mathbf{y}_\alpha = \alpha\mathbf{y}$ is used hereafter.

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

Binary hashing	Average Euclidean norm	Scale factor
Binry code	5.66	–
RP [1]	4.22	1.09
VSRP [11]	1.32	3.41
SH [16]	2.46	1.86
ITQ [7]	0.65	6.17

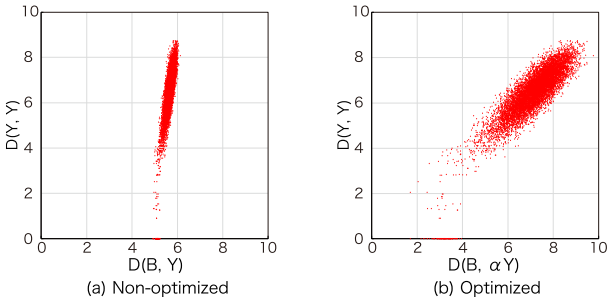

Fig. 1 Effect of optimizing scale factor. $D(\cdot, \cdot)$ is Euclidean distance.

3. Fast Computation of Euclidean Distance by Introducing a 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

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

The first term of Eq. (2) 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

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

number of floating-point computations.

To overcome this problem, Hare et al. [8] proposed decomposing the real vector $\mathbf{y}_\alpha$ into k weight factors and k binary basis vectors as

$$\mathbf{y}_\alpha \approx \mathbf{M}\mathbf{c}, \tag{3}$$

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. (3) into the second term of Eq. (2), we obtain

$$\mathbf{b}^{\mathrm{T}}\mathbf{y}_\alpha \approx \mathbf{b}^{\mathrm{T}}\mathbf{M}\mathbf{c} = \sum_{i=1}^{k} c_i \mathbf{b}^{\mathrm{T}}\mathbf{m}_i. \tag{4}$$

The computations $\mathbf{b}^{\mathrm{T}}\mathbf{m}_i$ that appeared in Eq. (4) are extremely fast because this is equivalent to computing the Hamming distance between $\mathbf{b}$ and $\mathbf{m}_i$, as

$$\mathbf{b}^{\mathrm{T}}\mathbf{m}_i = L - 2\mathrm{HammingDistance}(\mathbf{b}, \mathbf{m}_i) \tag{5}$$

Since the Hamming distance can be computed efficiently using a bitwise XOR followed by a bit-count, Eq. (4) 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. [8] 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 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:

$$J(\mathbf{c}, \mathbf{M}) = \|\mathbf{y}_\alpha - \mathbf{M}\mathbf{c}\|_2^2. \tag{6}$$

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 an 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. 2** that our method can approximate $\mathbf{y}_\alpha$ using fewer basis vectors $\mathbf{m}_i$ than Hare's method.

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 $J(\mathbf{c}, \mathbf{M})$.

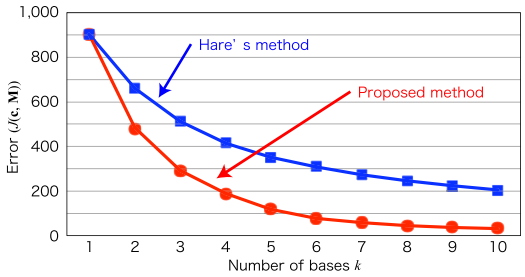

Fig. 2 Comparison of the proposed method with Hare's method [8].

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 Dataset for Evaluating Keypoint Matching

We prepared seven issues of the IEEE Spectrum magazine and captured them from six viewpoints. Let I_j^i denote an image in the database, where i is the magazine index ($1 \le i \le 7$), and j is the viewpoint index ($1 \le j \le 6$). 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 $\mathbf{W}$ if necessary, 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 a keypoint pair is regarded as inlier. We computed the average number of matches over test image pairs and the rate of a correct matching.

4.2 Comparing Symmetric and Asymmetric Representation

We compared three kinds of feature representation:

- *Binary code vs. Binary code: BC-BC*
  This is a conventional symmetric representation.
- *Binary code vs. Real vector: BC-RV without optimizing α*
  This is the asymmetric representation proposed in this paper.

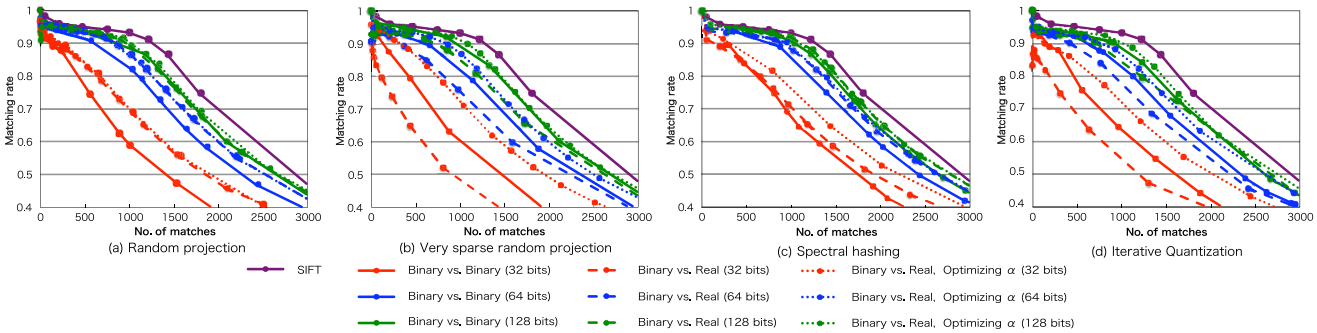

Fig. 3 Comparison with state-of-the-art methods.

Table 2 Computational time [ns].

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

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

No. of keypoints (No. of images)	32 bits		64 bits		128 bits	
	Dec.	No dec.	Dec.	No dec.	Dec.	No dec.
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

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.

Four binary hashing functions were tested: Random Projection (RP) [1], Very Sparse Random Projection (VSRP) [11], Spectral Hashing (SH) [16], and Iterative Quantization (ITQ) [7]. Each method uses different $\mathbf{W}$ and $f(\cdot)$. We used SIFT [12] as local descriptors.

The results are shown in **Fig. 3**. 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 the matching performance when short binary codes are used to reduce the 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 a difference and contributed to improving the matching performance.

4.3 Effect of Decomposition

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

4.3.1 Matching Performance

The results of a comparison between our decomposition algorithm and Hare's method [8] are shown in **Fig. 4**. 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 the computational time of matching.

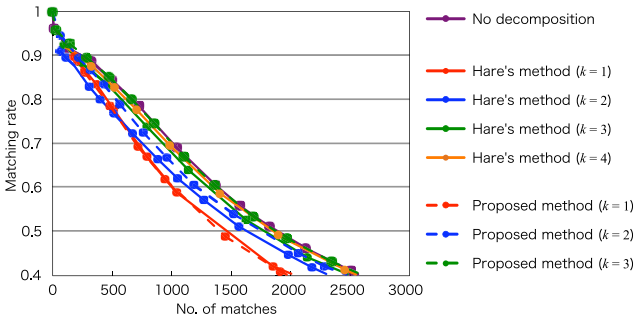

Fig. 4 Comparison with Hare's method [8].

4.3.2 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.

4.3.3 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 example shows that if the server stores 100,000 images and the length of the binary code is set to 32 bits, the 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 the asymmetric feature representation between client- and server-side. The second was the scale optimization to fit two different feature spaces. The third was the 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 ap-

plications but also for similar applications such as speech recognition systems.

References

[1] Achlioptas, D.: Database-friendly random projections: Johnson-Lindenstrauss with binary coins, *Journal of Computer and System Sciences*, Vol.66, No.4, pp.671–687 (2003).

[2] Alahi, A., Ortiz, R. and Vandergheynst, P.: FREAK: Fast Retina Keypoint, *CVPR* (2012).

[3] Ambai, M. and Yoshida, Y.: CARD: Compact And Real-time Descriptors, *ICCV*, pp.97–104 (2011).

[4] Bay, H., Ess, A., Tuytelaars, T. and Van Gool, L.: Speeded-Up Robust Features (SURF), *Computer Vision and Image Understanding*, Vol.110, No.3, pp.346–359 (2008).

[5] Calonder, M., Lepetit, V., Ozuysal, M., Trzcinski, T., Strecha, C. and Fua, P.: BRIEF: Computing a Local Binary Descriptor Very Fast, *PAMI*, Vol.34, No.7, pp.1281–1298 (2012).

[6] Chandrasekhar, V., Takacs, G., Reznik, Y., Grzeszczuk, R. and Girod, B.: Compressed Histogram of Gradients: A Low-Bitrate Descriptor, *IJCV* (2011).

[7] Gong, Y. and Lazebnik, S.: Iterative Quantization: A Procrustean Approach to Learning Binary Codes, *CVPR* (2011).

[8] Hare, S., Saffari, A. and Torr, P.H.S.: Efficient Online Structured Output Learning for Keypoint-Based Object Tracking, *CVPR*, pp.1894–1901 (2012).

[9] Heo, J.-P., Lee, Y., He, J., Chang, S.-F. and Yoon, S.-e.: Spherical Hashing, *CVPR* (2012).

[10] Leutenegger, S., Chli, M. and Siegwart, R.: BRISK: Binary Robust Invariant Scalable Keypoints, *ICCV* (2011).

[11] Li, P., Hastie, T.J. and Church, K.W.: Very Sparse Random Projections, *Int. Conf. on Knowledge Discovery and Data Mining* (2006).

[12] Lowe, D.G.: Distinctive image features from scale-invariant keypoints, *IJCV*, Vol.60, No.2, pp.91–110 (2004).

[13] Mikolajczyk, K. and Schmid, C.: A Performance Evaluation of Local Descriptors, *PAMI*, Vol.27, No.10, pp.1615–1630 (2005).

[14] Rublee, E., Rabaud, V., Konolige, K. and Bradski, G.: ORB: An Efficient Alternative to SIFT or SURF, *ICCV* (2011).

[15] Takacs, G., Chandrasekhar, V., Tsai, S., Chen, D., Grzeszczuk, R. and Girod, B.: Unified Real-time Tracking and Recognition with Rotation-invariant Fast Features, *CVPR* (2010).

[16] Weiss, Y., Torralba, A. and Fergus, R.: Spectral Hashing, *NIPS*, pp.1753–1760 (2008).

(Communicated by *Seiji Hotta*)