CARD: Compact And Real-time Descriptors

Mitsuru Ambai and Yuichi Yoshida
Denso IT Laboratory, Inc.
{manbai, yyoshida}@d-itlab.co.jp

Abstract

We propose Compact And Real-time Descriptors (CARD) which can be computed very rapidly and be expressed by short binary codes. An efficient algorithm based on lookup tables is presented for extracting histograms of oriented gradients, which results in approximately 16 times faster computation time per descriptor than that of SIFT. Our lookup-table-based approach can handle arbitrary layouts of bins, such as the grid binning of SIFT and the log-polar binning of GLOH, thus yielding sufficient discrimination power. In addition, we introduce learning-based sparse hashing to convert the extracted descriptors to short binary codes. This conversion is achieved very rapidly by multiplying a very sparse integer weight matrix by the descriptors and aggregating signs of their multiplications. The weight matrix is optimized in a training phase so as to make Hamming distances between encoded training pairs reflect visual dissimilarities between them. Experimental results demonstrate that CARD outperforms previous methods in terms of both computation time and memory usage.

1. Introduction

Visual correspondence between two images can be established by matching local descriptors. Scale-Invariant Feature Transform (SIFT)[13] is a representative method for extracting scale- and orientation-invariant descriptors that are used in many applications such as structure from motion[18], image retrieval[16], and generic object recognition[11]. However, prior works that have been done on extracting local descriptors still face drawbacks in terms of computation time and memory usage. These drawbacks prevent local descriptor-based applications from running on devices equipped with a low-end CPU, such as mobile phones and in-vehicle processors.

As is well known to the computer vision community, SIFT computation time greatly differs from real-time computation. Fig. 1 demonstrates SIFT computation time when 3,762 keypoints were extracted from a 640 × 480 size image. To clarify where the bottleneck is, each processing step is separately indicated in the Fig. 1. Seventy-five percent of the SIFT computation time is used for estimating orientations and extracting descriptors from each patch around the keypoints. Moreover, conventional descriptors consume a large amount of memory because of their high dimensionality. To reduce the memory usage, they are usually stored in a byte array (e.g., an unsigned char array in C language) instead of a single precision array. However, each descriptor still consumes many bits (e.g., 1024 bits are used for a 128-dimensional descriptor of SIFT).

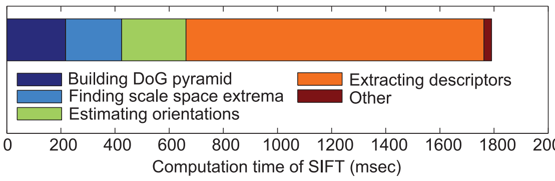


Figure 1. Estimating orientations and extracting descriptors from each patch account for 75% of the SIFT computation time. We used an Intel Core 2 Duo 2.66GHz processor in this experiment.

This paper addresses these issues. Specifically, we propose Compact And Real-time Descriptors (CARD) which can be computed very quickly and be represented by short binary codes.

1.1. Related works

Fast local descriptors: Improving SIFT computation time is a crucial issue to achieve real-time applications. Bay et al.[3] have proposed the Speeded-Up Robust Features (SURF), which provides scale- and orientation-invariant descriptors and processes several times faster than SIFT does. Takacs et al.[19] introduced the Rotation-Invariant, Fast Feature (RIFF) descriptor for fast feature tracking and video content recognition on mobile phones. Wagner et al.[21] implemented simplified SIFT on mobile phones for augmented reality applications. In more recent years, extremely fast descriptors have been proposed in [4, 20]. Although they have achieved significant speeding up compared to SIFT, the distinctiveness of their local descriptors was inferior to that of SIFT because of the simplicity of their descriptor designs.

Compact representation: Binary hashing[24, 22, 15, 23, 2, 12] is an interesting way to convert feature vectors to short binary codes. A descriptor $\mathbf{d} \in \mathbb{R}^D$ is mapped into short binary codes $\mathbf{b} \in \{0, 1\}^B$ by a binary hashing function $\mathbf{b} = (\operatorname{sgn}(f(\mathbf{W}^\top \mathbf{d})) + 1)/2$, where D is a dimension, B is a bit length of binary codes, and $\mathbf{W} \in \mathbb{R}^{D \times B}$ is a weight matrix. Since similarities of the binary codes can be measured by Hamming distances, computation time of finding near neighbors can be drastically reduced.

$\mathbf{W}$ and $f(\cdot)$ characterize each binary hashing. The simplest hashing function is random projections [15, 7], where $f(\cdot)$ is an identity function and elements in $\mathbf{W}$ are sampled from a normal distribution. Compared to the randomized approach, introducing machine learning technique leads to much shorter binary codes[24, 23]. Spectral hashing[24] uses principal directions of training data for $\mathbf{W}$ and applies a cosine-like function for $f(\cdot)$. In addition to the binary hashing, entropy-based compression[5] and dimensionality reduction[9] have also been used for compressing local descriptors.

While the above-mentioned methods lack discussion on conversion time for generating binary codes, sparse random projections have been used to address this issue[2, 12]. Achlioptas[2] has proposed using $\mathbf{W}$ which takes only three kinds of integer entries $\{-1, 0, 1\}$ with probabilities $\{\frac{1}{6}, \frac{2}{3}, \frac{1}{6}\}$. Li et al.[12] have recommended using the same integer entries $\{-1, 0, 1\}$ with different probabilities $\{\frac{1}{2\sqrt{D}}, 1 - \frac{1}{\sqrt{D}}, \frac{1}{2\sqrt{D}}\}$ to make the weight matrix $\mathbf{W}$ sparser, achieving significant speeding up of the conversion. In both cases, an identity function is used for $f(\cdot)$. As well as (dense) random projections, relatively long binary codes are required to preserve original distinctiveness.

1.2. Contribution

Fast descriptor extraction by lookup tables: We propose an efficient algorithm based on lookup tables for extracting descriptors from each patch around a keypoint, resulting in approximately 16 times faster computation time per descriptor than that of SIFT. While other methods simplified descriptor design to achieve fast computation in exchange for discrimination power, our lookup-table-based approach can extract histograms of oriented gradients from arbitrary layouts of bins, such as the grid binning of SIFT and the log-polar binning of GLOH[14], yielding sufficient discrimination power.

Learning-based sparse hashing: We introduce learning-based sparse hashing to convert the extracted descriptors to short binary codes. As well as the sparse random projections, we use a sparse integer matrix for fast conversion, but introduce machine learning technique to train $\mathbf{W}$ so as to preserve the dissimilarity of two descriptors even in Hamming space.

2. Fast descriptor extraction

2.1. Keypoint detection

Scale invariance is an important property of keypoint detection. Lowe[13] has regarded local extrema of a Difference of Gaussians (DoG) pyramid as scale-invariant keypoints. Dufournaud et al.[6] have proposed scale-invariant Harris corner detector that applies different sizes of Gaussian filters to an original image. The common drawback of DoG and scale-invariant Harris is that applying different sizes of Gaussian filters imposes heavy processing time.

Instead of applying time-consuming Gaussian filters, our method simply uses a multi-scale image pyramid to extract keypoints that are robust under scale change[19, 21]. The image pyramid is constructed in such a way that ratio of the size of the *n*-th level image to that of the (*n*-1)-th level image is $1/\sqrt{2}$ with the 0-th level being the original image. In practice, the *n*-th level is simply given by downsampling of (*n*-2)-th level so that the size gets half. In this way bilinear interpolation is avoided excepting the creation of the 1st level. Keypoints are then found from each of the pyramid levels by a corner detector[17].

2.2. Efficient descriptor extraction

It is a well-known fact that spatial binning followed by extracting histograms of oriented gradients gives better performance than other types of descriptors[14]. In this section, we show an efficient algorithm for extracting descriptors based on the orientation histograms. Our algorithm has two characteristics which contribute fast computation such that: (a) bilinear interpolation of pixel values is completely avoided and (b) computation of histograms of orientated gradients is simplified by using lookup tables.

2.2.1 Overview of conventional approach

Before explaining our approach, we first review conventional approaches for extracting descriptors based on histograms of oriented gradients such as SIFT and GLOH.

Conventions are as follows. An image coordinate of a keypoint is represented as $(p_x, p_y)^\top$. Magnitudes and orientations of gradients of a gray scale image $I(x, y)$ are defined as follows:

$$m(x, y) = \sqrt{I_x(x, y)^2 + I_y(x, y)^2} \tag{1}$$

$$\theta(x, y) = \operatorname{angle}(I_x(x, y), I_y(x, y)), \tag{2}$$

where I_x and I_y are x- and y-gradient images and $\operatorname{angle}(u, v)$ is a two-argument function that returns counterclockwise angle from the positive u-axis to a vector $(u, v)^\top$ in the range $[0, 2\pi)$.

A patch around a keypoint is resized and rotated according to an estimated scale and an estimated principal orientation of the patch. The reshaped patch is spatially divided

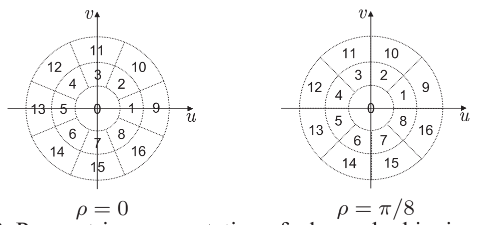


Figure 2. Parametric representation of a log-polar binning pattern.

into K bins. For example, SIFT and GLOH use a grid binning pattern and a log-polar binning pattern, respectively. L dimensional orientation histograms are extracted from every bin, which is denoted as follows:

$$\mathbf{h}_k = (h_{k,0}, h_{k,1}, \cdots, h_{k,(L-1)})^\top \in \mathbb{R}^L, \tag{3}$$

where k represents an index of a bin. A KL dimensional descriptor $\mathbf{d} \in \mathbb{R}^{KL}$ is obtained by concatenating them such that $\mathbf{d} = (\mathbf{h}_0^\top, \mathbf{h}_1^\top, \cdots, \mathbf{h}_{K-1}^\top)^\top$.

An angle quantization function given below is used for computing the orientation histograms:

$$Q_N(\theta) = \lfloor \frac{N\theta}{2\pi} + \frac{1}{2} \rfloor \bmod N, \tag{4}$$

which quantizes an angle in radian to an integer from 0 to $N-1$.

2.2.2 Descriptor extraction without interpolation

In the conventional approaches, a patch around a keypoint is reshaped before extracting the orientation histograms $\mathbf{h}_k$. This causes a large number of bilinear interpolations.

In contrast, instead of reshaping the patch, our idea is to rotate a binning pattern at the scale level where the keypoint is detected. This approach completely avoids time-consuming bilinear interpolation. Thus we introduce a parametric function $\psi(u, v, \rho)$ to represent rotated binning pattern, where ρ is the principal orientation represented in radian and $(u, v)^\top$ is a relative coordinate from the keypoint. The binning pattern lies on uv-plane, and is rotated counter clockwise by ρ. $\psi(u, v, \rho)$ returns an index of a bin in the range from 0 to $K-1$. An example of $\psi(u, v, \rho)$ is shown in Fig. 2. A bin to which a pixel $(x, y)^\top$ belongs can be determined as follows:

$$k = \psi(x - p_x, y - p_y, \rho). \tag{5}$$

An orientation of the pixel $(x, y)^\top$ is quantized to L levels to obtain $\mathbf{h}_k$. Since the binning pattern is rotated, the orientation of the pixel gradient must be shifted to cancel out ρ. This can be represented as follows:

$$l = Q_L(\theta(x, y) - \rho), \tag{6}$$

where l determines an element in $\mathbf{h}_k$ to which the edge power of the pixel $(x, y)^\top$ is voted.

Consequently, we obtain a simple algorithm to extract a descriptor $\mathbf{d}$ as below.

```
1. Initialize $\mathbf{h}_0, \cdots, \mathbf{h}_{K-1}$ by zero.
2. Do the following two steps for all the pixels $(x, y)^\top$ in
   a patch around a keypoint $(p_x, p_y)^\top$.
   (a) Calculate Eq. (5) and (6) to obtain $k$ and $l$.
   (b) $h_{k,l} \leftarrow h_{k,l} + m(x, y)G_{\sigma_1}(x - p_x, y - p_y)$
3. Concatenate all of the $\mathbf{h}_0, \cdots, \mathbf{h}_{K-1}$ to obtain $\mathbf{d}$.
4. Normalize $\mathbf{d}$.
```

G_σ is a normal distribution with variance σ.

Since [14] reported that a log-polar binning gave the best discrimination power, we assume this to be the case in this work. A parametric representation of a log-polar binning pattern is defined as follows:

$$\psi(u, v, \rho) = \begin{cases} 0 & , \quad 0 \le r < r_1 \\ 1 + Q_8(\mathrm{angle}(u, v) - \rho) & , \quad r_1 \le r < r_2, \\ 9 + Q_8(\mathrm{angle}(u, v) - \rho) & , \quad r_2 \le r \le r_3 \end{cases} \tag{7}$$

where $r = \sqrt{u^2 + v^2}$ and r_k is a threshold for binning in the radial direction. A patch is divided into 17 bins as shown in Fig. 2. Note however that arbitrary layouts of binning pattern can be used in our framework.

2.2.3 Efficient implementation using lookup tables

In practice, computation of Eq. (5) and (6) can be replaced with lookup tables if the principal orientation ρ is quantized. This leads to remarkable improvement in computation time. Two lookup tables, $\bar{\psi}$ and χ, defined later, are used in our algorithm.

Before explaining the lookup table based implementation, we briefly revisit how to estimate the principal orientation of a patch. The principal orientation is defined as dominant gradient orientation around a keyopoint. This is achieved by finding a maximum of a orientation histogram which has M bins. This procedure is summarized below.

```
1. Quantize the image orientation $\theta(x, y)$ to $M$ levels.
   $i_\theta(x, y) = Q_M(\theta(x, y))$
2. Initialize an orientation histogram
   $\hat{\mathbf{h}} = (\hat{h}_0, \cdots, \hat{h}_{M-1})^\top$ by zero.
3. Do the following two steps for all the pixels $(x, y)^\top$ in
   a patch around a keypoint.
   (a) $i = i_\theta(x, y)$
   (b) $\hat{h}_i \leftarrow \hat{h}_i + m(x, y)G_{\sigma_2}(x - p_x, y - p_y)$
4. Smooth the orientation histogram with a normal distri-
   bution with variance $\sigma_s$.
5. Find the maximum of the orientation histogram
   $i_\rho = \arg\max_i(\hat{h}_i)$.
```

First we consider to replace a runtime computation of (5) with a lookup table. Here we introduce an inverse function of $Q_N(\theta)$ as follows:

$$Q_N^{-1}(i) = (2\pi(i \bmod N))/N, \tag{8}$$

which maps a discrete value to a representative angle in radian at the quantization level. Letting $\rho = Q_M^{-1}(i_\rho)$ in Eq. (7), we obtain a spatial binning table $\bar{\psi}(u, v, i_\rho)$ as follows:

$$\bar{\psi}(u, v, i_\rho) = \psi(u, v, \rho) = \psi(u, v, Q_M^{-1}(i_\rho)). \tag{9}$$

Since u, v and i_ρ take only a finite number of integer values, $\bar{\psi}(u, v, i_\rho)$ can be precomputed. Fig. 3 visualizes $\bar{\psi}(u, v, i_\rho)$. As a result, Eq. (5) can be replaced as follows:

$$k = \bar{\psi}(x - p_x, y - p_y, i_\rho). \tag{10}$$

Second we discuss fast computation of Eq. (6). Since M-level quantized orientations $i_\theta(x, y)$ have already been obtained at the stage of principal orientation estimation, we consider to reuse $i_\theta(x, y)$ to obtain L-level quantized orientations. The computation of Eq. (6) can be approximated as below:

$$\begin{aligned} l = Q_L(\theta(x, y) - \rho) &\approx Q_L(Q_M^{-1}(i_\theta(x, y) - i_\rho)) \\ &= \chi(i_\theta(x, y), i_\rho). \end{aligned} \tag{11}$$

$i_\theta(x, y)$ is shifted to cancel out the angular offset given by i_ρ. The nested function $Q_L(Q_M^{-1}(\cdot))$ converts quantization level from M to L. This conversion is legitimate if $M \gg L$. Since $i_\theta(x, y)$ and i_ρ take only integer values from 0 to $M - 1$, the function χ can be precomputed and stored into an $M \times M$ array. By replacing Eq. (5) and (6) with Eq. (10) and (11), one achieves significant speeding up.

Needless to say, the normal distributions used in the above algorithms can also be precomputed. We found that the best choice of parameters $\{r_1, r_2, r_3, \sigma_1, \sigma_2, \sigma_s, M, L\}$ is $\{3, 10, 20, 15, 10, 3, 40, 8\}$, respectively. This brings $8 \times 17 = 136$ dimensional descriptors.

3. Learning-based sparse hashing

3.1. Cost function

In this section, we assume that the extracted descriptors are mean-centered by using an average descriptor which is computed from training samples. The extracted descriptor $\mathbf{d} \in \mathbb{R}^D$ is converted to a short binary code $\mathbf{b} \in \{0, 1\}^B$ by the binary hashing function

$$\mathbf{b} = (\mathrm{sgn}(\mathbf{W}^\top \mathbf{d}) + 1)/2, \tag{12}$$

where D is the dimension of the descriptor, B is the bit length, and $\mathbf{W} \in \mathbb{R}^{D \times B}$ is the weight matrix. The dissimilarity of the two binary codes can be measured by a normalized Hamming distance

$$D_\mathrm{h}(\mathbf{b}_u, \mathbf{b}_v) = \frac{1}{B} \parallel \mathbf{b}_u - \mathbf{b}_v \parallel_1 . \tag{13}$$

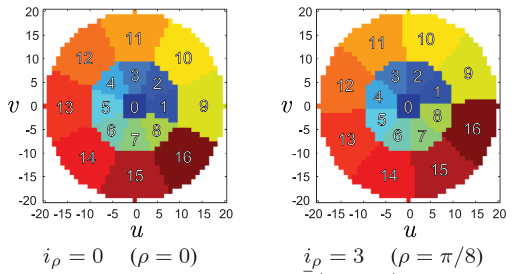


Figure 3. Spatial binning table $\bar{\psi}(u, v, i_\rho)$. $(M = 40)$

The aim is to train $\mathbf{W}$ so as to keep dissimilarities in original space and those in transformed space as close as possible. We minimize the sum of squared errors between the original dissimilarities and the normalized Hamming distances of training pairs. We introduce a cost function

$$C(\mathbf{W}) = \sum_{(u,v) \in P} (D_\theta(\mathbf{d}_u, \mathbf{d}_v) - D_\mathrm{h}(\mathbf{b}_u, \mathbf{b}_v))^2, \tag{14}$$

where P is a set of the training pairs and $D_\theta(\mathbf{d}_u, \mathbf{d}_v)$ is a normalized angle in the original feature space

$$D_\theta(\mathbf{d}_u, \mathbf{d}_v) = \arccos\left(\frac{< \mathbf{d}_u, \mathbf{d}_v >}{||\mathbf{d}_u|| \, ||\mathbf{d}_v||}\right)/\pi. \tag{15}$$

The binary hashing in Eq. (12) must be computed fast enough for real-time applications. The bottleneck of the conversion time is the computation of $\mathbf{W}^\top \mathbf{d}$, which requires $D \times B$ multiplications and $(D-1) \times B$ additions if a dense matrix is used. In contrast, our approach is motivated by the idea of sparse random projections[2, 12], which uses a sparse integer matrix for $\mathbf{W}$. In our approach, $\mathbf{W}$ is determined by minimizing $C(\mathbf{W})$ under the condition that the $\mathbf{W}$ takes only three kinds of integer entries $\{-1, 0, 1\}$ and the number of non-zero elements of W must maintain a predefined number S. This reduces the number of computation steps of $\mathbf{W}^\top \mathbf{d}$ to only S additions (or subtractions). A descriptor $\mathbf{d}$ is converted to a binary code sufficiently fast by setting S to a small value. Interestingly, we observed that ninety percent of the elements in $\mathbf{W}$ can be set to zero without loss in accuracy as is demonstrated in the next section. This leads to remarkable improvement in computation time.

3.2. Greedy algorithm for optimization

Since $C(\mathbf{W})$ is a non-linear and discontinuous function, gradient-based optimization methods such as the Levenberg-Marquardt and Gauss-Newton algorithms are not suitable. We therefore propose a greedy optimization algorithm to deal with this issue. In our optimization scheme, two randomly selected elements of $\mathbf{W}$ are replaced with optimal values at one time.

Since $\mathbf{W}$ must satisfy the two conditions $w_{ij} \in \{-1, 0, 1\}$ and $\sum |w_{ij}| = S$, the two elements can be replaced by checking several possible candidates. This leads to the following greedy-style algorithm:

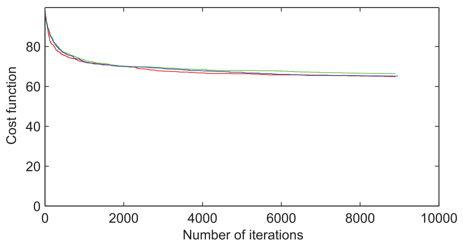


Figure 4. The cost value $C(\mathbf{W})$ is minimized by the algorithm proposed in Section 3.2. In this example, 25,000 training pairs were used. The bit length B was set to 64. 90% of the elements in $\mathbf{W}$ were set to zero.

Problem: Minimize $C(\mathbf{W})$ subject to $w_{ij} \in \{-1, 0, 1\}$ and $\sum |w_{ij}| = S$
Algorithm:

1. Initialize $\mathbf{W}$ by the following two steps.
   (a) Every w_{ij} is randomly chosen from $\{-1, 1\}$.
   (b) Randomly selected $(DB - S)$ elements in $\mathbf{W}$ set to zero.
2. Randomly pick two elements, w_{uv} and w_{pq}.
3. Update (w_{uv}, w_{pq}) for each of the following cases.
   (a) If both (w_{uv}, w_{pq}) are non-zero, check four kinds of pairs, i.e., $(1, -1)$, $(-1, 1)$, $(1, 1)$, and $(-1, -1)$, and take the one that best minimizes $C(\mathbf{W})$.
   (b) If both (w_{uv}, w_{pq}) are zeros, return to Step 2.
   (c) Otherwise, check four kinds of pairs, i.e., $(0, 1)$, $(0, -1)$, $(1, 0)$, and $(-1, 0)$, and take the one that best minimizes $C(\mathbf{W})$.
4. Return to Step 2 until the $C(\mathbf{W})$ converges.

Fig. 4 shows how the cost value $C(\mathbf{W})$ is decreased by this algorithm. Three different initial values are tested. As can be seen from the figure, the cost $C(\mathbf{W})$ is robustly decreased independent of different initial values.

4. Experimental results

In this section, we evaluate the performance of our descriptors both with and without binary coding. Here we define a *sparseness* as $1 - \frac{S}{DB}$, which means the ratio between the number of zero elements and the total number of elements of weight matrix $\mathbf{W}$.

4.1. Distinctiveness

Fig. 5 shows the probability density functions of angles between local descriptors. The left figure is the CARD result before encoding. The right figure is the SIFT result extracted from keypoints detected by DoG. The probability density of correct matching pairs and incorrect matching pairs are separately depicted in this figure. These matching pairs are extracted from Mikolajczyk's database[14, 1],

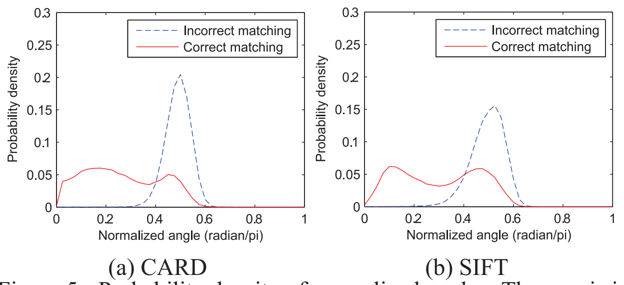


Figure 5. Probability density of normalized angles. The x-axis is the radian normalized by π.

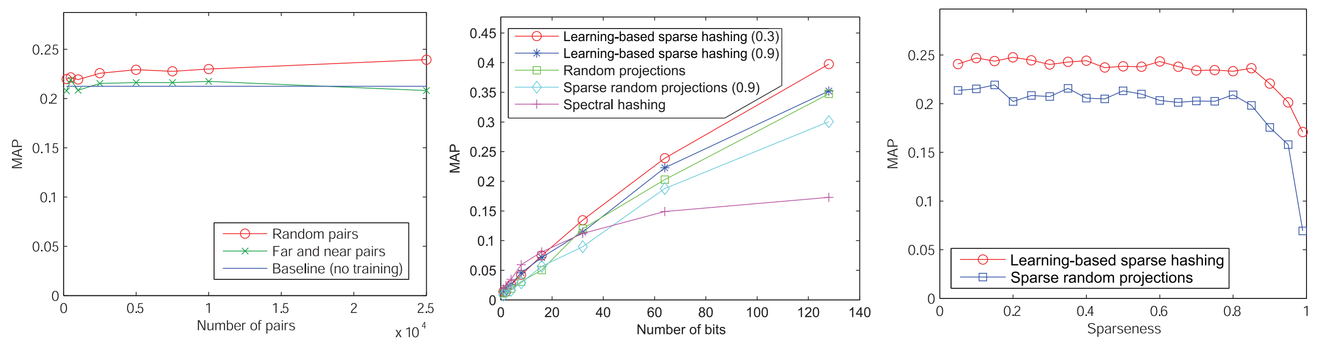
which includes true homography matrices of image pairs. The homography matrices were used to separate the correct and the incorrect matching pairs independently.

The probability densities obtained by CARD and SIFT were very similar. In both cases, the distributions obtained from incorrect matching pairs could be approximated by Gaussian distributions with small variances. In contrast, correct matching pairs provided broad density distributions, because Mikolajczyk's database includes severe image distortions such as affine transformations, which are not tractable by either of the descriptors.

Fig. 5 suggests that most of the incorrect matching pairs can be cut off by using an appropriate threshold value t. Since the mean and variance of normalized angles obtained from the incorrect matching pairs were 0.49 and 0.05 in the case of CARD, 99.7% of the incorrect matches can be cut off by setting $t = 0.49 - 3 \times 0.05 = 0.34$ according to 3σ rule. In the next section, the threshold t is used to define the mean average precision for evaluating binary hashing performance.

4.2. Approximating original distance metric

Near neighbors in the original feature space must also be close to each other in Hamming space. We evaluated the performances of different binary hashing methods by using the mean average precision (MAP), which approximates area under the precision-recall curve. The MAP was calculated as follows. We first collected training and testing samples, then chose one image from each category in Caltech101 dataset[10], and extracted 136 dimensional descriptors from them. 50,000 training samples and 10,000 testing samples were randomly chosen from all of the extracted descriptors. For every test sample, we defined *true near neighbors* from the training samples closely located to the test sample within the distance t determined in section 4.1. After all of the test and training samples were encoded to short binary codes, near neighbors of the test sample were searched from the training samples in Hamming space. The training samples were sorted in ascending order of Hamming distances from each test sample. Precisions at each location of the ranked list where the true near neighbor was



(a) MAP vs. number of training pairs (b) MAP vs. number of bits (c) MAP vs. sparseness

Figure 6. (a) MAP was improved by increasing the number of randomly chosen training pairs. (b) Different binary hashing functions are compared in this figure. Learning-based sparse hashing was able to improve the performance of sparse random projections. (c) Interestingly, MAPs provided by learning-based sparse hashing were not dropped until sparseness was set to greater than about 0.9.

correctly retrieved were averaged. The mean of the averaged precisions for all the test samples was defined as the MAP. A good binary hashing method provides a high MAP.

We observed that the more training pairs were used in the training phase, the higher was the MAP achieved as shown in Fig. 6(a). As a baseline, the MAP of sparse random projections is also shown in the figure. In this experiment, a sparseness of $\mathbf{W}$ was set to 0.3 and the bit length B was set to 64 for every case. Two different ways were tested to choose the training pairs, i.e., (a) *random pairs* which were randomly chosen from the training samples and (b) *far and near pairs*, which were preferentially selected from close and separated pairs in the feature space. As can be seen from Fig. 6(a), far and near pair selection resulted in no significant MAP improvement. Thus, we recommend using a simple random choice to collect training pairs.

Fig. 6(b) shows an experimental comparison of learning-based sparse hashing, spectral hashing[24], random projections[15, 7], and sparse random projections[2, 12]. Since Li et al.[12] recommended filling a weight matrix $\mathbf{W}$ by zeros with probability $1 - 1/\sqrt{D}$, the sparseness was set to $(1 - 1/\sqrt{136}) \approx 0.9$ in the case of the sparse random projections. For the learning-based sparse hashing, two kinds of sparseness, 0.3 and 0.9, were tested. As shown in Fig. 6(b), MAPs provided by learning-based sparse hashing were greater than or nearly equal to that of the other methods in spite of the significant sparseness. By comparing MAPs obtained by learning-based sparse hashing with sparse random projections at the same sparseness level of 0.9, we were able to prove that machine learning positively affected the performance of binary coding.

Interestingly, Fig. 6(c) demonstrates that MAPs of learning-based sparse hashing were not dropped until the sparseness was set to greater than about 0.9, as well as the sparse random projections. The MAPs were suddenly dropped when the sparseness exceeded 0.9 in both cases. In this experiment, the bit length was set to 64. It was proved that by setting the sparseness of $\mathbf{W}$ to 0.9, we could achieve significant improvement in processing time of generating binary codes without loss of the performance on approximating original distance metric by Hamming distance.

4.3. Performance in matching two views

We evaluated the capability of finding correspondences between two views based on Mikolajczyk's method[14, 1]. Mikolajczyk's et al. provided the evaluation framework of matching two views and compared many different descriptors. The database has eight different scene categories. Each category included six images capturing the same target under different conditions. Corresponding points between image pairs were determined by thresholding the distance ratio between the first and the second nearest neighbors. 1-precision vs. recall curves were used for evaluating the performances of different descriptors.

In our experiment, the first and second images in each category were chosen from Mikolajczyk's database for evaluating matching capability. 50,000 training samples were randomly chosen from descriptors extracted from all the remaining images in the database. 25,000 training pairs were randomly selected from the training samples to train weight matrices $\mathbf{W}$. The bit length B was set to 32, 64, and 128. The sparseness was set to 0.9.

Fig. 7 shows 1-precision vs. recall curves of four categories in the database: *bark*, *boat*, *wall*, and *graffiti*. The scenes of *bark* and *boat* were used for evaluating scale and orientation change. The scenes of *wall* and *graffiti* were used for evaluating viewpoint changes such as projective distortion. We compared GLOH, SIFT, PCA-SIFT[9], moment invariants[14], CARD without encoding and CARD with encoding to short binary codes. As binary hashing functions, we used learning-based sparse hashing and sparse random projections. The sparseness was set to 0.9. Since these two binary hashing functions incorporated randomness, we calculated 1-precision and recall 10

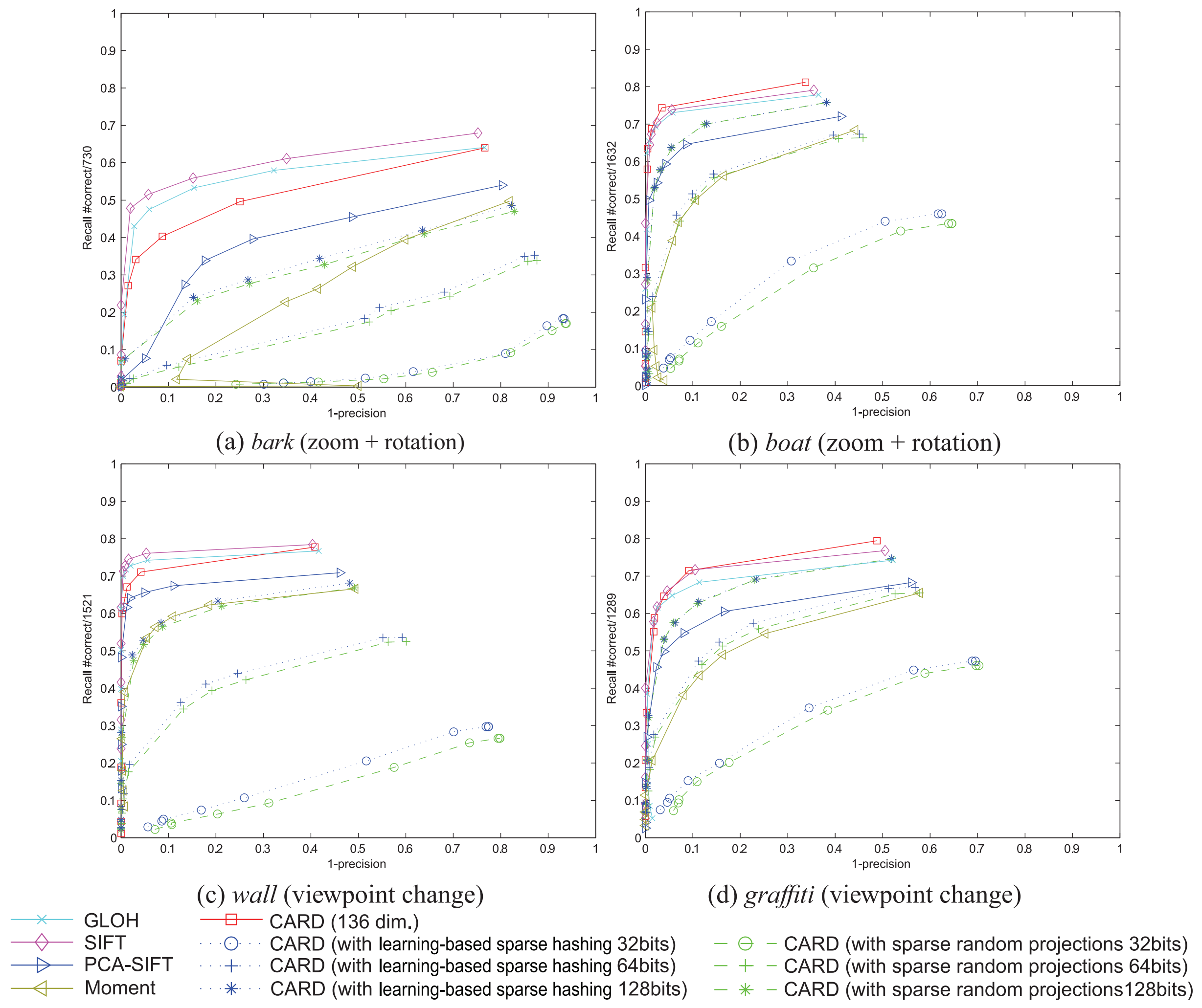


Figure 7. Different descriptors were compared according to the Mikolajczyk's method[14].

times and averaged them. The keypoint detection algorithm shown in Section 2 was commonly used for extracting all of the different descriptors. CARD before encoding provided comparable results of GLOH and SIFT in all of the four scenes. After converting to binary codes, we observed that the learning-based sparse hashing improved the matching performance compared to the sparse random projections at the same bit length.

Scale and orientation invariance of SIFT, SURF, and CARD are compared in Fig. 8. As a keypoint detector, DoG and Haar-like wavelet responses were used for SIFT and SURF, respectively. We chose one image as a reference from the *graffiti* category in Mikolajczyk's database and generated a set of transformed images. Corresponding points were obtained by finding nearest neighbor descriptors between the reference and the transformed image. The obtained matches were filtered out by using a homography matrix estimated by RANSAC. The ratio between the number of inliers and total matches was evaluated. CARD provided results that were as good as or better than SIFT and SURF results even when short binary codes were used.

4.4. Computation time

In this section, we evaluate the computation time of CARD. We used an Intel Core 2 Duo 2.66GHz processor in all of the experiments. Computation times per descriptor are shown in Fig. 9. In this experiment, we excluded the computation time of keypoint detection. The percentage of the computation time of CARD to that of SIFT implemented by Hess[8] is on the x-axis. We tested two binary hashing functions, i.e., (a) random projections that used a dense weight matrix and (b) learning-based sparse hashing at sparseness level of 0.9. The bit length was set to 128 in both cases. It was obvious that the use of the sparse integer matrix contributed to speeding up of the conversion. Thanks to the lookup table based algorithm for extracting

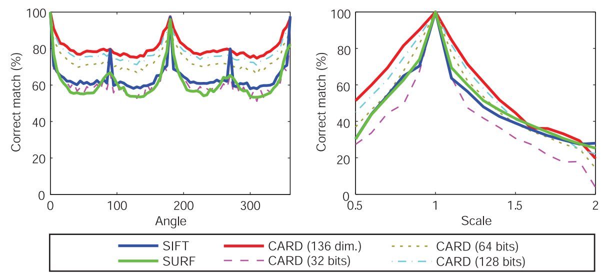


Figure 8. Scale and orientation invariance of SIFT, SURF, and CARD are compared in this figure.

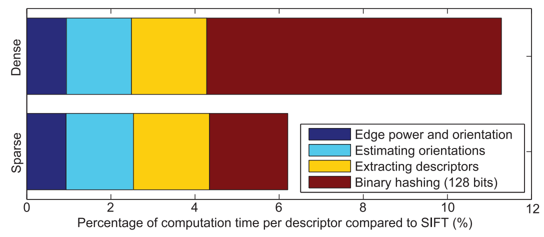


Figure 9. Percentage of computation time per descriptor compared to SIFT. 100% on the x-axis means the computation time of SIFT.

Table 1. Computation time of SIFT, SURF, and CARD (640×480 size image).

	SIFT	SURF	CARD (128 bits)
Computation time (msec)	1030.5	206.1	129.3
Number of keypoints	1553.8	1267.8	1658.0

descriptors and fast encoding by using the sparse integer matrix, the computation time of CARD per descriptor was approximately 16 times faster than that of SIFT.

Finally, we show a comprehensive comparison of computation time in Table.1. SIFT implemented by Hess[8] and SURF in OpenCV library were used for the comparison. In this experiment, 640×480 size images were used. 5 different images were processed to obtain average computation times and average number of keypoints. As can be seen from the table, CARD significantly outperformed both SIFT and SURF.

5. Conclusion

In this paper, we proposed Compact And Real-time Descriptors (CARD) that achieve fast computation and compact representation. Since our algorithm requires lower computational resource than conventional methods, CARD can be suitably used on devices equipped with a low-end CPU, such as mobile phones and in-vehicle processors. As subjects for future work, it will be interesting to try to develop an efficient optimization algorithm for learning-based sparse hashing to more accurately reconstruct an original distance metric by Hamming distance. Dealing with affine distortion will also be an important issue for designing good descriptors.

References

[1] Affine covariant regions datasets. http://www.robots.ox.ac.uk/ vgg/data/data-aff.html. 5, 6

[2] D. Achlioptas. Database-friendly random projections. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 274–281, 2001. 2, 4, 6

[3] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, 110:346–359, June 2008. 1

[4] M. Calonder, V. Lepetit, P. Fua, K. Konolige, J. Bowman, and P. Mihelich. Compact signatures for high-speed interest point description and matching. In *ICCV*, pages 357–364, 2009. 1

[5] V. Chandrasekhar, G. Takacs, D. Chen, S. Tsai, R. Grzeszczuk, and B. Girod. CHoG: Compressed histogram of gradients a low bit-rate feature descriptor. In *CVPR*, pages 2504–2511, 2009. 2

[6] Y. Dufournaud, C. Schmid, and R. Horaud. Matching images with different resolutions. In *CVPR*, pages 612–618, 2000. 2

[7] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42:1115–1145, November 1995. 2, 6

[8] R. Hess. An open-source SIFTLibrary. In *Proceedings of the international conference on Multimedia*, pages 1493–1496, 2010. 7, 8

[9] Y. Ke and R. Sukthankar. PCA-SIFT: A more distinctive representation for local image descriptors. In *CVPR*, pages 506–513, 2004. 2, 6

[10] R. F. L. Fei-Fei and P. Perona. Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. In *CVPR Workshop*, page 178, 2004. 5

[11] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, pages 2169–2178, 2006. 1

[12] P. Li, T. J. Hastie, and K. W. Church. Very sparse random projections. In *Proceedings of the international conference on Knowledge Discovery and Data mining*, pages 287–296, 2006. 2, 4, 6

[13] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60:91–110, November 2004. 1, 2

[14] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *PAMI*, 27:1615–1630, 2005. 2, 3, 5, 6, 7

[15] K. Min, L. Yang, J. Wright, L. Wu, X.-S. Hua, and Y. Ma. Compact projection: Simple and efficient near neighbor search with practical memory requirements. In *CVPR*, pages 3477–3484, 2010. 2, 6

[16] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *CVPR*, pages 2161–2168, 2006. 1

[17] J. Shi and C. Tomasi. Good features to track. In *CVPR*, pages 593–600, 1994. 2

[18] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3D. In *SIGGRAPH*, pages 835–846, 2006. 1

[19] G. Takacs, V. Chandrasekhar, S. Tsai, D. Chen, R. Grzeszczuk, and B. Girod. Unified real-time tracking and recognition with rotation-invariant fast features. In *CVPR*, pages 934–941, 2010. 1, 2

[20] S. Taylor, E. Rosten, and T. Drummond. Robust feature matching in 2.3μs. In *CVPR Workshop*, pages 15–22, 2009. 1

[21] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg. Pose tracking from natural features on mobile phones. In *ISMAR*, pages 125–134, 2008. 1, 2

[22] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for scalable image retrieval. In *CVPR*, pages 3424–3431, 2010. 2

[23] J. Wang, S. Kumar, and S.-F. Chang. Sequential projection learning for hashing with compact codes. In *Proceedings of the international conference on Machine Learning*, 2010. 2

[24] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2008. 2, 6