Local Difference Binary for Ultrafast and Distinctive Feature Description

Xin Yang, *Member*, *IEEE*, and
Kwang-Ting (Tim) Cheng, *Fellow*, *IEEE*

Abstract—The efficiency and quality of a feature descriptor are critical to the user experience of many computer vision applications. However, the existing descriptors are either too computationally expensive to achieve real-time performance, or not sufficiently distinctive to identify correct matches from a large database with various transformations. In this paper, we propose a highly efficient and distinctive binary descriptor, called local difference binary (LDB). LDB directly computes a binary string for an image patch using simple intensity and gradient difference tests on pairwise grid cells within the patch. A multiple-gridding strategy and a salient bit-selection method are applied to capture the distinct patterns of the patch at different spatial granularities. Experimental results demonstrate that compared to the existing state-of-the-art binary descriptors, primarily designed for speed, LDB has similar construction efficiency, while achieving a greater accuracy and faster speed for mobile object recognition and tracking tasks.

Index Terms—Binary feature descriptor, mobile devices, object recognition, tracking, augmented reality

1 Introduction

Feature point descriptors are widely used in many computer vision tasks such as marker-less augmented reality (AR) [10], [11], simultaneous localization and mapping (SLAM) [16], and image retrieval [19], [21]. Their broad applications have driven the development of a plethora of descriptors [1], [2], [3], [4], [5], [6], [7], [8]. However, as the application requirements are increasingly demanding, for example, handling larger databases and/or running real time on handheld devices, the demand for more advanced descriptors is stronger than ever.

An ideal descriptor should achieve two competing goals: high-quality description and low computational complexity. High-quality descriptors capture the most representative information in an image, such that different image content can be distinguished (i.e., high distinctiveness) and the same content subject to various image distortions can be recognized (i.e., high robustness). High-speed descriptors enable the entire application task to run in real time at a sufficiently high frame rate.

Many research efforts have been made to achieve either strict quality requirements or low computational speed. The SIFT descriptor [2], proposed over a decade ago, has been widely adopted as one of the highest quality options. However, it imposes a heavy computation burden. This drawback has drawn extensive efforts [1], [3] for optimizing its speed without compromising its quality too much. Among the enhancements, SURF [1] is arguably the most noticeable. But recent experiments [22] have shown that the SURF descriptor is still too computationally heavy; thus only a limited number of points can be handled for real-time applications such as AR, especially for handheld devices such as smartphones and tablets. On the other end of the spectrum aiming primarily at fast runtime, lightweight binary descriptors such as BRISK [6], FREAK [7], BRIEF [4], and its variant rBRIEF (or ORB descriptor) [5] have become increasingly popular as they are very efficient to store and to match (simply computing the Hamming distance between descriptors via XOR and bit-count operations). These runtime advantages make them more suitable for real-time applications and handheld devices. However, these binary descriptors utilize overly simplified information, i.e., raw intensities of a subset of pixels within an image patch for binary tests, and thus have low discriminative ability. Lack of distinctiveness incurs an enormous number of false matches when matching against a large database. Expensive postverification methods (e.g., RANSAC [23] or PROSAC [13]) are usually required to discover and validate matching consensus, increasing the runtime of the entire process.

In this paper, we introduce a new binary descriptor, named local difference binary (LDB), which achieves similar computational speed and robustness as the state-of-the-art binary descriptors [4], [5], [6], [7], yet offering much higher distinctiveness compared to them. The high quality of LDB is achieved through three schemes. First, LDB utilizes both average intensity I_{avg} and first-order gradients, d_x and d_y, of grid cells within an image patch. Specifically, the internal patterns of the image patch are captured through a set of binary tests, each of which compares the I_{avg}, d_x and d_y of a pair of grid cells (see Figs. 1a and 1b). The average intensity and gradients provide a more complete description than other binary descriptors. Second, LDB employs a multiple-gridding strategy to encode the structure at different spatial granularities (see Fig. 1c). Coarse-level grids can cancel out high-frequency noise, while fine-level grids can capture detailed local patterns, thus enhancing distinctiveness. Third, LDB leverages a modified AdaBoost method to select a set of salient bits. The modified AdaBoost targets the fundamental goal of ideal binary descriptors: minimizing distances between matches while maximizing them between mismatches, optimizing the performance of LDB for a given descriptor length. Computing LDB is extremely fast. Relying on integral images, the average intensity and first-order gradients of each grid cell can be obtained by only 4-8 add/subtract operations.

Our experimental results demonstrate that the construction speed of LDB is much faster than that of SURF and is comparable to those of the state-of-the-art binary descriptors, including ORB, BRISK, and FREAK, while the robustness and distinctiveness of LDB is higher than these descriptors.

The remainder of this paper is organized as follows: Section 2 reviews the related work. Section 3 presents details of the proposed descriptor. In Sections 4 and 5 we compare performance of LDB with the state-of-the-art descriptors on public benchmarks and evaluate its speed, robustness, and discriminative power for mobile applications. Section 6 concludes the paper.

2 Related Work

SIFT is currently among the best quality descriptors in the literature. It relies on local gradient histograms and represents an image patch using a 128D real-value vector. Despite its high descriptive power and robustness to a variety of image transformations, the intensive computations for obtaining gradients and the high dimensionality of SIFT make it prohibitively slow to compute and match, especially on low-power devices. PCA-SIFT [3] reduced the descriptor from 128D to 36D to reduce the matching cost, whereas the increased time for descriptor formation almost annihilates the increased speed of matching. To date, SURF descriptor is considered as the most popular replacement for SIFT. It greatly accelerates the gradient computations using integral images as [14], while almost preserving the quality of SIFT.

- *The authors are with the Electrical and Computer Engineering Department, University of California, Santa Barbara, Harold Frank Hall, Rm 4109, Santa Barbara, CA 93106-9560.*
  E-mail: xinyang@umail.ucsb.edu, timcheng@ece.ucsb.edu.

Manuscript received 9 Sept. 2012; revised 7 May 2013; accepted 21 July 2013; published online 13 Aug. 2013.
Recommended for acceptance by T. Tuytelaars.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number TPAMI-2012-09-0713.
Digital Object Identifier no. 10.1109/TPAMI.2013.150.

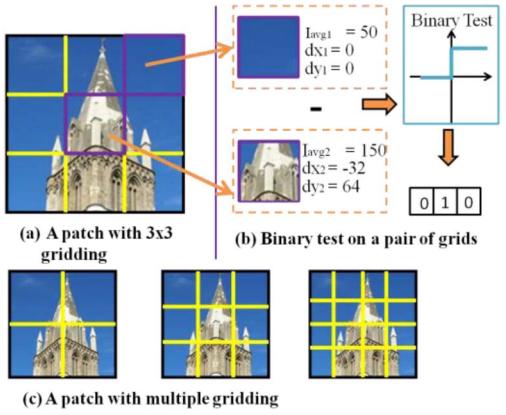

Fig. 1. Illustration of LDB extraction. (a) An image patch is divided into 3×3 equal-sized grids. (b) Compute the intensity summation (I), gradient in x and y directions (d_x and d_y) of each patch, and compare I, d_x, and d_y between every unique pair of grids. (c) Three-level gridding (with 2×2, 3×3, and 4×4 grids) is applied to capture information at different granularities.

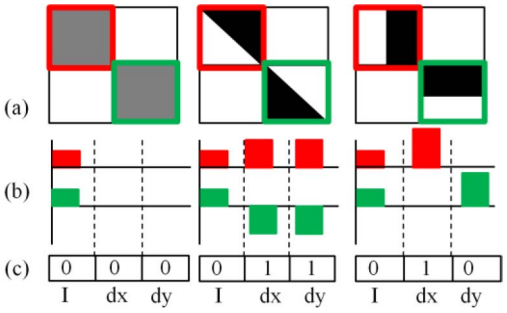

Fig. 2. Illustration of binary tests on the pair of diagonal grids for three image patches. (a) Three image patches with different pixel intensity values and distributions. (b) The average intensity value I and gradients in the x and y directions, d_x and d_y, for each pair of diagonal grids. For each of these three cases, red and green grids have the same I, while having different d_x and d_y. Thus, individual grids can still be distinguished from each other. (c) Three-bit binary test results for the three patches.

However, recent results [22] reveal that SURF is still too costly to construct and to match, thus it remains unsuitable for real-time mobile applications.

The increasing demand of handling a larger database or running on mobile devices stimulates the development of light-weight binary descriptors that are efficient to construct, to match, and to store. Notably, the BRIEF descriptor [4] directly generates bit strings by simple binary tests comparing pixel intensities in a smoothed image patch. But BRIEF is very sensitive to image scale and rotation changes, restricting its application to general tasks. To address these limitations, Rublee et al. [5] proposed ORB which incorporates image pyramids and an orientation operator into BRIEF to achieve scale and rotation invariance. In addition, rather than randomly selecting pixel pairs as in BRIEF, ORB selects highly-variant and uncorrelated pixel pairs based on an ad-hoc scheme. BRISK [6] is another binary descriptor which also targets achieving better quality than BRIEF. BRISK leverages similar approaches as ORB to cope with scale and rotation invariance. Unlike ORB, BRISK leverages a circular sampling pattern based on which it computes intensity comparisons between short-distant point pairs to form a binary descriptor string. Alahi et al. [7] further enhanced BRISK by leveraging a sampling strategy which resembles the retinal ganglion cells distribution. However, all the existing binary descriptors utilize only raw intensities of a subset of pixels within an image patch and ad-hoc schemes for bit selection, thus are not distinctive enough to effectively localize matched patches in large databases. Postprocessing for removing false matches is usually required to ensure sufficient matching accuracy, increasing the total time cost for the entire process.

3 LDB: Local Difference Binary

Our feature design was inspired by Schechtman and Irani's *self-similarity descriptor* [8], in which an image patch was divided into grids and cross correlations between the center grid and the others are computed. Since most of the photometric changes (e.g., lighting/contrast changes, blurring, image noises, etc.) can be removed by computing the difference between two subregions, the *self-similarity descriptor* is resilient to photometric changes, yet captures intrinsic structures within the image.

However, computing cross correlations between grids demands sum-of-square-differences operations for each pixel, which is computationally expensive and cannot be sped up by integral images. Moreover, the *self-similarity descriptor* is a real-value feature vector and thus is not efficient to match and store. Here, we create a bit vector based on binary tests between every pair of grid cells and avoid cross-correlation operations.

More specifically, we divide each image patch P into $n \times n$ equal-sized grids, extract representative information from each grid cell, and perform a binary test τ on each pair of grid cells (i and j) as

$$\tau(Func(i), Func(j)) := \begin{cases} 1, & \text{if } (Func(i) - Func(j)) > 0 \text{ and } i \neq j, \\ 0, & \text{otherwise,} \end{cases} \tag{1}$$

where $Func$ is the function for extracting information from a grid cell.

The information used for the binary tests determines the quality as well as the efficiency of a descriptor. The most basic yet fast-to-compute information is the average intensity, which represents the direct component of a grid cell and can be computed extremely fast via the integral image technique. However, the average intensity is too coarse to describe the intensity changes inside a grid cell. For example, Fig. 2a shows three patches, each with 2×2 grid cells. Consider the up-left grid cell in each of the three patches (denoted by red rectangles). Although the pixel intensity values and distributions in these three grid cells are very different, their means of intensities are exactly the same (see Fig. 2b). A binary test τ between diagonal grid cells yields identical results for these three distinct patches.

To improve its quality, we incorporate the first-order gradients, in addition to using the average intensity. It is known that gradients are more resilient to photometric changes than average intensities and can also capture intensity changes inside a grid such as the magnitude and direction of an edge. Considering the patches in Fig. 2, the first-order gradients in the x and y directions are able to distinguish them (see Fig. 2b). In addition, gradients can be computed using box filters [14] which can be easily sped up by integral images. We define $Func(i)$ for our LDB as a tuple $Func(i) \in \{I_{intensity}(i), d_x(i), d_y(i)\}$, where

$$\begin{aligned} I_{\text{intensity}}(i) &:= \frac{1}{m} \sum_{k=1 \sim m} \text{Intensity}(k), \\ d_x(i) &:= Gradient_x(i), \\ d_y(i) &:= Gradient_y(i), \end{aligned} \tag{2}$$

and m is the total number of pixels in grid cell i. Since we use equal-sized grids, m is the same in all cases and can be omitted in the computation. $Gradient_x(i)$ and $Gradient_y(i)$ are the regional gradients of grid cell i in the x and y directions, respectively.

Based on (2), the average intensity and the x and y gradients are computed for each grid cell; thus comparing the respective values

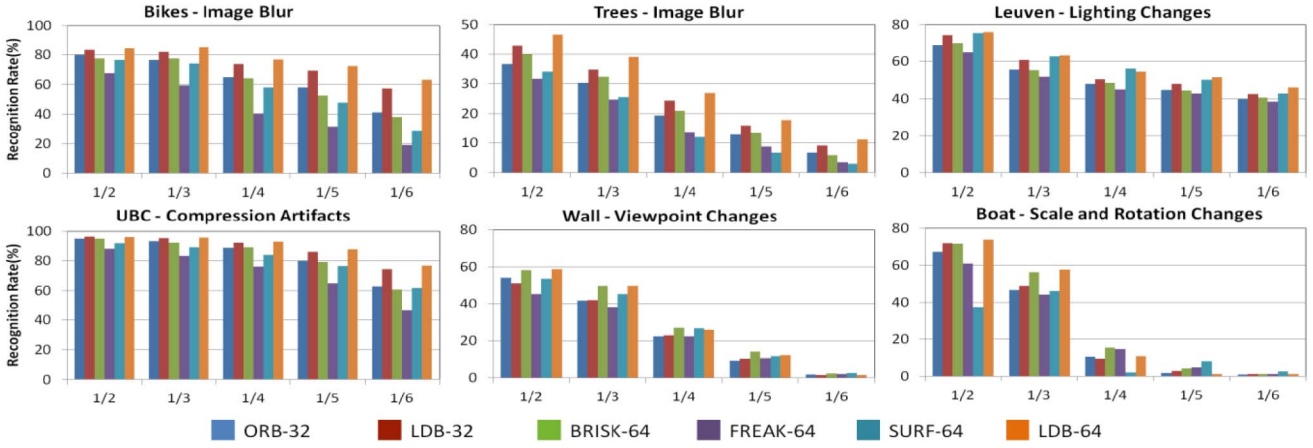

Fig. 3. Recognition rate obtained by LDB, ORB, BRISK, FREAK, and SURF for the six image sequences of VggAffine data set.

for each pair of grid cells results in 3 bits using test τ defined in (1) (see Fig. 3c).

Performing binary tests on pairwise grid cells out of $n \times n$ grids results in a bit string of $3n^2(n^2-1)/2$. As n increases, the distinctiveness of a descriptor increases, while the complexity for matching and storage increases as well. To form a compact LDB descriptor, we choose a subset of n_d bits out of $3n^2(n^2-1)/2$ bits to form a unique LDB descriptor.

In Sections 3.1 and 3.2, we discuss the gridding and bit-selection strategy, respectively. In Section 3.3, we introduce steered LDB to cope with rotation and show how to compute it efficiently.

3.1 Gridding Choices

The choice of the grid size influences both robustness and distinctiveness of the LDB descriptor. If the gridding is fine, i.e., having a large number of small grids per patch, corresponding grids of two similar patches could be misaligned even for a small amount of patch shifting. Hence, the descriptor's stability would be lower. Whereas, the descriptor extracted from smaller grids captures more detailed local patterns of a patch and thus is more distinctive than that extracted from larger grids. Dividing a patch into a small number of large grids, on the other hand, results in a more stable but less distinctive descriptor.

With this observation, we employ a strategy incorporating multiple-gridding choices to achieve both high robustness and high distinctiveness. Each image patch is partitioned in multiple ways, for example, 2×2, 3×3, 4×4, and 5×5 from each of which a LDB binary string is derived. We then concatenate the strings resulting from all the partitions to form an initial LDB descriptor.

3.2 AdaBoost-Based Bit Selection

The above mentioned gridding strategy may produce a long bit string. For example, concatenating binary strings resulting from 2×2, 3×3, 4×4, and 5×5 partitions to a patch leads to a binary string of 1,386 bits. Further increasing the number of ways for partitioning would result in even more bits. Despite being distinctive, long bit strings reduce the efficiency for matching and storage. Moreover, bits in a long string may have strong correlation, yielding redundancy. To address this, we employ a bit-selection strategy which aims at achieving two fundamental goals of an ideal binary description: 1) maximizing the distance between mismatches, while minimizing it between matches, and 2) having low correlations between dimensions.

The Adaboost learning, which is used in Viola and Jones's face detection system [17], has demonstrated high effectiveness for selecting a small set of face features which can best separate face and nonface images. With a similar objective for the face features and binary bits, differentiating faces from nonfaces versus differentiating matches from mismatches, it should be intuitive that AdaBoost would be effective for achieving the first goal. In each round of face-feature selection, AdaBoost reweights the training examples to increase the emphasis on those incorrectly classified by the previous weak classifiers. This scheme implicitly reduces correlations among selected features as the feature selected in each round tends to classify the data in a different way from the ones selected in previous rounds.

However, the AdaBoost procedure in [17] is not directly applicable to our bit selection due to two reasons:

1. It assigns a weight to each selected feature which reflects its impact on the final performance. However, including the weights would result in floating-point descriptors, instead of binary descriptors, which would drastically increase the complexity for matching. To produce binary descriptors, we force all weights to be identical for all selected features.
2. It aims at a cascaded rejection-based pipeline: The face features in the latter pipeline stages are selected with the objective of rejecting the remaining negative examples that have not been removed in the previous stages. To support this goal, their learning algorithm was designed to select a single feature in each round which best separates the remaining negative examples from the positive ones. However, our objective is to select a set of bits to form an optimal binary descriptor. Thus in each round, the selected bit combined with the previously selected ones should jointly produce small distances between matches and large distances between mismatches. Therefore, we modify AdaBoost's feature selection criterion: Instead of selecting a bit which gives a minimum single-round classification error, we choose the one which gives a minimum accumulated error, as the accumulated error reflects the total distance based on the entire set of selected bits.

Pseudocode 1 details the procedure of our salient bit selection. It is worth noting that as the number of selected bits increases, the tasks for AdaBoost becomes more difficult and may produce a classification error rate greater than 0.5. In these cases, the following selected bits are not better than random selection. To handle this problem, we prepare several training sets and switch to a new training set and reset the weights for all training data once the error rate is greater than 0.5. We also tried three other unsupervised methods for salient bit selection: random selection, entropy-based selection [20], and variance-correlation-based selection [5]. Experimental results show that the modified AdaBoost method produces a more robust and distinctive description than those three methods.

Pseudo Code 1: AdaBoost-based Bit Selection.

Input: training data $T = \{(X_i, Y_i) \mid X_i = (p_a, p_b)$ is a pair of patches, $Y_i = 1$ if X_i is a match; otherwise is $Y_i = -1, 1 \leq i \leq |T|\}$

Output: K salient bits

Step1. Compute N-bit strings for all patches in T.

Step2. Assign equal weight $D_i = 1/|T|$ to all training data T_i.

Step3. For $\mathrm{t} = 1$ to K

- For each bit, classify all patch pairs X_i according to the following: X_i who has identical bit values for its two patches is classified as 1; else is labeled as -1.
- Find a bit b_t which gives minimum accumulated classification errors $\varepsilon_{acc}(t)$.
  $b_t = \arg\min \varepsilon_{acc}(t)$,
  where $\varepsilon_{acc}(t) = \varepsilon_{acc}(t-1) + \varepsilon_j, \varepsilon_{acc}(-1) = 0$,
  $\varepsilon_j = \sum_{i=1}^{|T|} D_i[Y_i \neq h_j(X_i)]$
- If $\varepsilon_j < 0.5$, update weights of training data; otherwise, switch to a new training set and reset weights to $1/|T|$ for all training data.

We use three publicly available data sets (Liberty, Yosemite, and Notre Dame [18]) as a source of training data for the AdaBoost learning. Each data set contains over 400 K patches, sampled densely from multiple images of 3D scenes. The data sets also include ground truth data indicating the match and mismatch information. The major transformations in these data sets include scaling, illumination changes, and viewpoint changes. These transformations commonly exist in real-world scenarios; thus training descriptors using the data sets can help optimize the performance for practical applications. In the training phase, we prepared three training sets and for each training set we randomly chose 5 K pairs of matching patches as positive data and 20 K pairs of nonmatching patches as negative data. More negatives are selected since in practice there are significantly more mismatches than matches for most matching tasks.

3.3 Steered LDB

We aim at mobile real-time applications, such as mobile object tracking and augmented reality, in which rotation changes are quite common. Thus a description that is invariant to rotation changes is very important for a good user experience. To make LDB invariant to in-plane rotation, we estimate a dominant orientation for a patch and align the patch to this orientation before computing its descriptor. Several orientation estimation approaches have been proposed in the past. Among them, we follow the intensity moments-based method [5] for its good performance and efficiency.

One problem of computing LDB descriptors for rotated patches is that the integral image of an entire image cannot be directly used for speeding up the computations because the patch axis is not parallel to the integral image axis. To address this problem, we compute a rotated integral image for each rotated patch. The rotated integral image of a patch is calculated by summing up pixels along the dominant orientation, instead of horizontal axis. Based on the rotated integral image of the patch, we can efficiently compute the intensity summation and gradients of any grid cell within the patch.

4 Comparison with State-of-the-Art

In this section, we compare the robustness, distinctiveness, and construction efficiency of LDB with ORB, BRISK, FREAK, and SURF on public benchmarks.

4.1 Invariance to Transformations

We first test the robustness of LDB with respect to common image transformations, including image blur, illumination changes, compression artifacts, and viewpoint, scale-and-rotation changes.

We performed the evaluation using six image sequences (i.e., Bikes, Trees, Leuven, UBC, Wall, and Boat) from the widely used data set introduced by Mikolajczyk and Schmid [12]. Each image sequence contains six images, sorted in order of an increasing degree of distortions with respect to the first image. In particular, the Bikes and Trees sequences are sorted in increasing image blur, the Leuven and UBC sequences are in order of increasing illumination changes and compression artifacts, respectively, and the Wall and Boat sequences are in order of view point and scale-and-rotation changes respectively. For each sequence, the task is to match the first image to the remaining five, yielding five image pairs per sequence which are denoted as pair 1/2 to pair 1/6.

For each test image, we compute 1,000 keypoints using an oFAST [5] detector (scales $= 3$ and scale factor $= 1.2$) and binary descriptors. Then for each keypoint in the first image, we find its nearest neighbor (NN) in the second one via brute-force matching. To obtain the ground truth of correct matches, for each keypoint in the first image, we infer its corresponding point in the second image using the known homography between them. We use the same evaluation metric as used in BRIEF [4], the *Recognition Rate* (i.e., the number of correct matches divided by the total number of matches), to measure the robustness of a descriptor. We use the OpenCV2.4.4 implementation for the oFAST detector, and the ORB, BRISK, FREAK, and SURF descriptors. The LDB descriptor is constructed by partitioning each patch in five ways, i.e., 2×2, 3×3, 4×4, and 5×5, and then utilizing the n_d bits selected based on the modified AdaBoost algorithm to form a final description. Two versions of LDB descriptors are generated and tested in this work, i.e., LDB-32 and LDB-64 which stand for 32-byte and 64-byte LDB descriptions, respectively.

Fig. 3 illustrates the recognition rate achieved by LDB-32/-64, ORB-32, BRISK-64, FREAK-64, and SURF-64. In general, LDB-64 exhibits higher robustness than other descriptors for all image sequences except for the Wall and Boat sequences in which it achieves similar results as BRISK. In principle, BRISK is inherently more robust to viewpoint, scaling and rotation changes than LDB, since BRISK selects short-distant pixel pairs and uses a circular sampling strategy with greater emphasis on pixels closer to the patch center. In comparison, LDB utilizes grid partitioning. However, as our training set includes samples with viewpoint and scaling changes, the learning process is able to select bits which are robust with respect to such geometric deformations for the LDB descriptors, yielding a comparable performance to BRISK.

Surprisingly, SURF is not better than (or even much worse than) LDB and other binary descriptors in several cases. Similar observations were already made and reported in BRISK [6] and FREAK [7]. One potential reason is that the performance of a descriptor is highly related to the detector, i.e., some descriptors are more stable and descriptive for blobs than corners. As a result, the SURF descriptor which is originally designed for use with a blob detector (i.e., approximate Hessian detector) exhibits poor performance when combined with oFAST, which is a corner detector. It is worth noting that for the Leuven sequence which has increasing lighting changes, both LDB-64 and SURF outperform other binary descriptors for all image pairs. We believe that the superior robustness of LDB-64 and SURF is due to the use of gradient information, which is more robust with respect to illumination changes than raw image intensities which are used in other binary descriptors.

The desired binary description should also be highly distinctive so as to differentiate correct matches from false matches. To examine the distinctiveness of binary descriptors we plot the

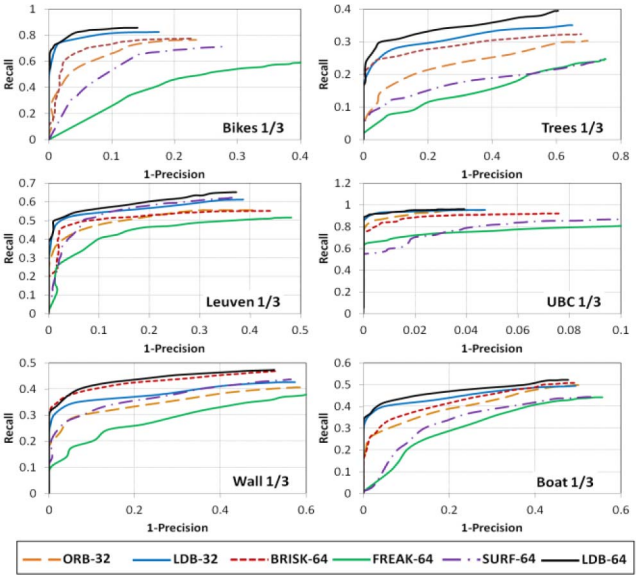

Fig. 4. Recall versus 1-Precision curves obtained by LDB, ORB, BRISK, FREAK, and SURF for image pairs 1/3 of VggAffine data set.

Recall versus *1-Precision* curves based on a threshold-based similarity matching as in [6] and [7]. Given a certain threshold, the recall is defined as the number of correctly returned matches divided by the total number of correct matches. The precision is the number of correctly returned matches over the total number of returned matches. A distinctive description should provide a high precision at any given recall. Fig. 4 shows the *Recall* versus *1-Precision* curves for image pairs 1/3 of all six image sequences. Results show that LDB-64 outperforms other descriptors for all sequences except for the Wall sequence in which it achieves almost the same results as BRISK.

4.2 Construction Efficiency

Table 1 presents the results of average time cost for constructing a single descriptor. The time cost was recorded on a Lenovo T420 laptop with an Intel core i5-2410M processor running at 2.3 GHz and a Google Nexus 4 smartphone with a Qualcomm Snapdragon S4 Pro processor running at 1.5 GHz, respectively.

The results show that constructing an LDB-32 descriptor takes 0.048 ms on the laptop and 0.22 ms on the phone, which is faster than ORB with the same descriptor length. Increasing the descriptor length of LDB to 64 bytes (i.e., LDB-64) will slightly increase the construction time, but it is still close to those for constructing ORB and FREAK. All binary descriptors listed in Table 1 have obvious advantages in construction efficiency comparing to SURF. In particular, constructing an LDB-64 descriptor is $\sim 13\times$ faster than SURF on the laptop and $\sim 8\times$ faster on the mobile platform.

5 Applications on Mobile Devices

In this section, we evaluate the efficiency and effectiveness of LDB for two mobile applications: mobile object recognition and real-time mobile object tracking, which will be described below in Sections 5.1 and 5.2, respectively.

5.1 Mobile Object Recognition

We implement a conventional object recognition pipeline on a mobile handheld platform: We first detect oFAST keypoints and construct descriptors for a captured image frame. Then we match descriptors to our database and return K top-ranked ($K = 10$ in our experiment) database images with most matching keypoints as potential recognized images. Finally, we perform PROSAC for each candidate image. The image with the most and a sufficient number of matches after PROSAC validation is reported as the final recognized image.

TABLE 1
Time Cost for Constructing a Single Descriptor on a Lenovo T420 Laptop and a Google Nexus 4 Smartphone

Descriptors	Time on PC (ms)	Time on Mobile (ms)
ORB-32	0.056	0.27
LDB-32	0.048	0.22
BRISK-64	0.026	0.05
FREAK-64	0.039	0.21
SURF-64	0.667	1.93
LDB-64	0.054	0.24

Time cost is the per-descriptor average time of computing 1,000 descriptors on each image (size: $1{,}000 \times 700$ *pixels) of the Bike sequence.*

Our database contains 228 planar objects [19], including 109 document images and 119 natural images. For each database image, we manually captured five pictures as the query images: One was taken with minor geometric changes and the other four were taken with up-scaling, down-scaling, rotation and shifting changes, respectively. As a result, there are a total of 1,140 query images.

5.1.1 Experimental Setup

We extract 1,000 features on average from each query image and database image, yielding 228 K descriptors in the database. Since it is infeasible to perform nearest neighbors search via brute-forcing matching for a large database, we leverage an indexing structure for efficient approximate nearest neighbors (ANN) search.

With ORB, BRISK, FREAK, and LDB all being binary descriptors, locality sensitive hashing (LSH) [15] is chosen for ANN search in our experiment. The key of LSH is a hash function, which maps similar descriptors into the same bucket of a hash table and different descriptors into different buckets. To find the NN of a query descriptor, we first retrieve its matching bucket, and then check all the descriptors within the matched bucket using a brute-force search. For binary features, the hash function can simply be a subset of bits from the original bit string; descriptors with a common sub-bit-string are cast to the same table bucket. The size of the subset, i.e., the hash key size, determines the upper bound of the Hamming distance among descriptors within the same buckets. We use multitable and multiprobe LSH to improve the detection rate of NN by looking at multiple hash tables and neighboring buckets in which a query descriptor falls within each table. In our experiment, we set the key size as 18, the number of hash tables as 5, and the number of probes as 19. For SURF descriptors, we use the kd-tree indexing structure for fast ANN search. For the sake of a fair comparison between SURF and other binary descriptors, we set the number of kd-trees as five, equal to the number of hash tables.

We employ five metrics for performance evaluation:

1. *Detection Rate*: the number of correctly recognized objects over the total number of objects;
2. *Precision*: the number of correctly recognized objects over the total number of recognized objects;
3. *Construction Time*: the average time cost for constructing a descriptor;
4. *Matching Time*: the average time cost for searching the ANN of a query descriptor from the database; and
5. *Memory Usage*: the total amount of memory cost for running a program.

All the time cost is recorded based on running the code on a single core of Google Nexus 4, operating at 1.5 GHz, and using an

TABLE 2
Comparing Performance of Descriptors for Recognizing 1,140 Manually Captured Images on a Google Nexus 4 Smartphone

Descriptor	Detection Rate (%)	Precision (%)	Construction Time (ms)	Matching Time (ms)	Memory Usage (MB) (recorded on PC)
ORB-32	93.3	98.6	0.146	7.26	118
LDB-32	96.3	99.0	0.139	2.55	118
BRISK-64	97.7	99.4	0.034	7.64	168
FREAK-64	98.3	99.5	0.108	24.65	168
SURF-64	83.8	92.9	1.488	-	466
LDB-64	98.6	99.5	0.143	4.76	168

LDB-64 achieves the highest detection rate and precision. Considering the total time cost including runtimes for both description and matching, LDB-64 is 1.6× faster than BRISK and 5.1× faster than FREAK.

Android 4.2 Jelly Bean operating system. The memory cost is read from the task manager of a PC.

5.1.2 Results

Table 2 summarizes the performance of the six descriptors. First, we compare ORB-32 and LDB-32 which are descriptors of the same length. Results show that LDB-32 achieves a greater detection rate (96.3 percent) and a higher precision (99 percent) than ORB-32. Regarding the runtime, the former is as fast as the latter to construct and 2.8× faster to match than the latter. As both LDB-32 and ORB-32 utilize 32 bytes to represent a single descriptor, the memory usages for these two descriptors are the same, which is 118 MB.

Second, we compare LDB-64 with BRISK-64, FREAK-64, and SURF-64. Results show that the detection rate and the precision of LDB-64 are greater than those of BRISK-64 and SURF-64 and almost the same as those of FREAK-64. It is worth mentioning that it is not completely fair to compare SURF-64 with other binary descriptors as they employ different indexing structures, i.e., kd-trees for SURF versus LSH for binary descriptors. Regarding the runtime of descriptor construction, LDB-64 takes 0.143 ms for deriving a descriptor, which is $10.4\times$ faster than SURF-64, $4\times$ slower than BRISK (0.034 ms), and $1.3\times$ slower than FREAK (0.108 ms). Although LDB-64 is slower than BRISK and FREAK for descriptor construction, it is faster for ANN matching. Specifically, LDB-64 achieves $1.6\times$ and $5.2\times$ speedups comparing to BRISK and FREAK, respectively. Considering the total time cost including runtimes for both description and matching, LDB-64 is $1.6\times$ faster than BRISK and $5.1\times$ faster than FREAK.

The speedup in ANN matching is most likely due to the better distinctiveness of LDB. Since nondistinctive descriptors may cluster together in the feature space, resulting in a large number of descriptors residing in some quantized subregions (i.e., buckets of a hash table), and consequently yielding a large number of checks during the matching process. On the contrary, distinctive descriptors distribute more uniformly in the feature space, yielding fewer and more relevant checks. To further validate this point, we took a close look at the distribution of bucket sizes in the hash tables, as shown in Fig. 5. We observe that the bucket size distribution of LDB-64s hash tables is more even than those of other binary descriptors.

In terms of the memory usage, even though SURF and binary descriptors (LDB-64, BRISK-64, and FREAK-64) use equal amount of bytes for representing a feature, it takes ~2× more memory space (466 MB) than binary descriptors (168 MB). We believe this is because kd-trees of SURF need to store a lot of intermediate tree nodes in addition to the feature descriptors, resulting in a large storage overhead. Due to the high memory cost of SURF, it fails to run successfully on a smartphone which has limited memory resource allocated for user applications.

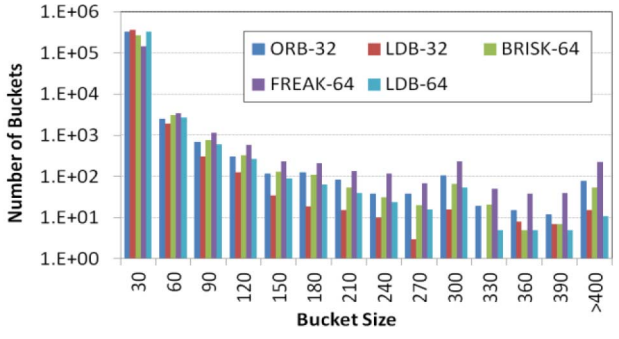

Fig. 5. LSH bucket size distribution of the 228 image data set. LDB-32/-64 have fewer large-sized buckets than FREAK, BRISK, and ORB and thus are faster in ANN matching.

5.2 Real-Time Mobile Object Tracking

Tracking on mobile handheld devices involves matching the live frames to descriptors of a target object. We compute 200 descriptors on each incoming frame and for each descriptor we search its NN in the target object using brute-force matching. The 100 top-ranked matches (i.e., matches with the shortest distances) are then validated by homography estimation based on PROSAC.

We evaluated the tracking performance of descriptors using the "Phone" data set (http://www.samhare.net/research/keypoints) which contains over 750 frames. We set the first frame as the target object. Table 3 summarizes the tracking performance of six descriptors. The results show that both LDB-32 and LDB-64 achieve a greater detection rate (i.e., the number of frames that the target object has been successfully detected divided by the total number of frames) than other descriptors. Regarding the runtime of matching 200 descriptors, ORB-32 and LDB-32 take approximate 27 ms on Google Nexus 4, which is 2× faster than BRISK-64, FREAK-64, and LDB-64. In comparison with binary descriptors, matching SURF descriptors is much slower, which takes ~1.4 s on the phone. In terms of the time cost for verification based on PROSAC, LDB-64, and BRISK-64 takes the least amount of time

TABLE 3
Performance of Object Tracking on a Google Nexus 4 Smartphone

Desc.	Detection Rate (%)	Runtime (ms)			Inlier Ratio(%)
		Match	Verify	Total	
ORB-32	95.2	27	20	119	50.0
LDB-32	98.9	27	11	112	55.1
BRISK-64	97.6	54	10	135	56.2
FREAK-64	95.2	54	43	152	39.4
SURF-64	98.4	1398	30	1761	43.0
LDB-64	100	54	10	142	56.9

LDB-32 gives the best result by achieving both a high detection rate and the fastest tracking speed.

(i.e., 10 ms). The faster speed for verification is due to the higher inlier ratio (i.e., the number of matches consistent with the estimated homography divided by the total number of matches). As shown in the last column of Table 3, LDB-64 and BRISK-64 achieve a greater inlier ratio than other descriptors, yielding fewer iterations for PROSAC to converge and a shorter runtime for the verification process. Finally, we compare the total time cost for tracking, including runtimes for feature extraction, matching and verification. As shown in the fifth column of Table 3, LDB-32 achieves the fastest tracking speed.

6 Conclusion

In this paper, we introduce a new binary descriptor, named LDB, which achieves greater robustness and discriminative ability than the state-of-the-art binary descriptors, while maintaining high runtime efficiency for descriptor construction. LDB employs a scalable, multiple-gridding strategy and computes brightness and gradient differences between pairwise grid cells to form a binary descriptor string. A modified AdaBoost method is leveraged to optimize its performance for a given descriptor length. Its superior performance was demonstrated based on an extensive evaluation using established benchmarks and a couple of mobile applications. The unique characteristics of LDB make it particularly suitable for computation-intensive tasks with hard real-time constraints running on a mobile platform with limited computing resources.

References

[1] H. Bay, A. Ess, T. Tuytelaars, and L.V. Gool, "SURF: Speeded-Up Robust Features," *Proc. European Conf. Computer Vision (ECCV),* 2006.

[2] D.G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *Int'l J. Computer Vision,* vol. 60, no. 2, pp. 91-110, 2004.

[3] Y. Ke and R. Sukthankar, "PCA-SIFT: A More Distinctive Representation for Local Image Descriptors," *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR),* 2004.

[4] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary Robust Independent Elementary Features," *Proc. European Conf. Computer Vision (ECCV),* 2010.

[5] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An Efficient Alternative to SIFT or SURF," *Proc. IEEE Int'l Conf. Computer Vision (ICCV),* 2011.

[6] S. Leutengger, M. Chli, and R.Y. Siegwart, "BRISK: Binary Robust Invariant Scalable Keypoints," *Proc. IEEE Int'l Conf. Computer Vision (ICCV),* 2011.

[7] A. Alahi, R. Ortiz, and P. Vandergheynst, "FREAK: Fast Retinal Keypoint," *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR),* 2012.

[8] E. Schechtman and M. Irani, "Matching Local Self-Similarities across Images and Videos," *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR),* 2007.

[9] E. Rosten, R. Porter, and T. Drummond, "Faster and Better: A Machine Learning Approach to Corner Detection," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 32, no. 1, pp. 105-119, Jan. 2010.

[10] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg, "Pose Tracking from Natural Features on Mobile Phones," *Proc. Int'l Symp. Mixed and Augmented Reality (ISMAR),* 2008.

[11] D. Wagner, D. Schmalstieg, and H. Bischof, "Multiple Target Detection and Tracking with Guaranteed Framerates on Mobile Phones," *Proc. Int'l Symp. Mixed and Augmented Reality (ISMAR),* 2009.

[12] K. Mikolajczyk and C. Schmid, "A Performance Evaluation of Local Descriptors," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 27, no. 10, pp. 1615-1630, Oct. 2005.

[13] O. Chum and J. Matas, "Matching with PROSAC—Progressive Sample Consensus," *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR '05),* vol. 1, pp. 220-226, 2005.

[14] P. Simard, L. Bottou, P. Haffner, and Y. LeCun, "Boxlets: A Fast Convolution Algorithm for Signal Processing and Neural Networks," *Proc. Neural Information Processing Systems (NIPS),* 1998.

[15] A. Gionis, P. Indik, and R. Motwani, "Similarity Search in High Dimensions via Hashing," *Proc. Int'l Conf. Very Large Databases (VLDB),* 2004.

[16] A.J. Davison, N.D. Molton, I. Reid, and O. Stasse, "MonoSLAM: Real-Time Single Camera SLAM," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 29, no. 6, pp. 1052-1067, June 2007.

[17] P. Viola and M. Jones, "Robust Real-time Object Detection," *Int'l J. Computer Vision,* vol. 57, no. 2, pp. 137-154, May 2004.

[18] S. Winder, G. Hua, and M. Brown, "Picking the Best DAISY," *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR),* 2009.

[19] X. Yang, Q. Liu, C.Y. Liao, and K.T. Cheng, "Large-Scale EMM Identification Based on Geometry-Constrained Visual Word Correspondence Voting," *Proc. ACM Int'l Conf. Multimedia Retrieval (ICMR),* 2011.

[20] X. Yang and K.T. Cheng, "LDB: An Ultra-Fast Feature for Scalable Augmented Reality on Mobile Devices," *Proc. Int'l Symp. Mixed and Augmented Reality (ISMAR),* 2012.

[21] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object Retrieval with Large Vocabularies and Fast Spatial Matching," *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR),* 2007.

[22] M. Calonder, V. Lepetit, K. Konolige, J. Bowman, P. Mihelich, and P. Fua, "Compact Signatures for High-Speed Interest Point Description and Matching," *Proc. IEEE Int'l Conf. Computer Vision (ICCV),* 2009.

[23] M.A. Fischler and R.C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Comm. ACM,* vol. 24, pp. 381-395, 1981.