libLDB: A Library for Extracting Ultrafast and Distinctive Binary Feature Description

Xin Yang[1], Chong Huang[2], Kwang-Ting (Tim) Cheng[1]

[1]Dept. of ECE, University of California, Santa Barbara, CA 93106, USA

[2]School of ICE, Beijing University of Posts and Telecommunications, Beijing, 100876, China

xinyang@umail.ucsb.edu, chonghuang@umail.ucsb.edu, timcheng@ece.ucsb.edu

ABSTRACT

This paper gives an overview of libLDB — a C++ library for extracting an ultrafast and distinctive binary feature LDB (Local Difference Binary) from an image patch. LDB directly computes a binary string using simple intensity and gradient difference tests on pairwise grid cells within the patch. Relying on integral images, the average intensity and gradients of each grid cell can be obtained by only 4~8 add/subtract operations, yielding an ultrafast runtime. A multiple gridding strategy is applied to capture the distinct patterns of the patch at different spatial granularities, leading to a high distinctiveness of LDB. LDB is very suitable for vision apps which require real-time performance, especially for apps running on mobile handheld devices, such as real-time mobile object recognition and tracking, markerless mobile augmented reality, mobile panorama stitching. This software is available under the GNU General Public License (GPL) v3.

Categories and Subject Descriptors

I.4.7 [Image Processing and Computer Vision]: Feature Measurement – *feature representation*; I.5 [Pattern Recognition]: General

General Terms

Algorithms, Design, Documentation

Keywords

Binary feature description, distinctiveness, efficiency, mobile handheld devices, mobile augmented reality, mobile object recognition and tracking

1. INTRODUCTION

Feature point descriptors are widely used in many computer vision tasks such as markerless augmented reality (AR), object recognition and tracking, image retrieval, and simultaneous localization and mapping (SLAM). Their broad applications have driven the development of a plethora of descriptors [1-8]. Notable results include SIFT [5] and SURF [6]. Several open source libraries, such as VLFeat [10], OpenSURF [11], have been released for extracting SIFT and SURF.

With the proliferation of mobile devices equipped with low-cost high-quality cameras, there is an increasing demand of running vision apps on mobile handheld devices. However, SIFT and SURF descriptors which require computation of gradients for every image pixel within an image patch are usually too computationally heavy for low-power mobile devices, especially for apps which demand real-time performance. The increasing demand of running vision apps efficiently on mobile devices stimulates the development of lightweight binary descriptors. BRISK [1], FREAK [2], BRIEF [13] and its variant rBRIEF (or ORB descriptor) [3] are some good examples. These descriptors aim primarily at fast runtime by directly generating bit strings by simple binary tests comparing pixel intensities in a smoothed image patch. In addition, they are also 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. Optimized implementation of these descriptors has been already included in OpenCV library [12]. 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 lots of false matches when matching between two images or against a large database. A large number of operations are usually required for post-verification methods (e.g. PROSAC [14]) to discover and validate matching consensus, increasing the runtime of the entire process.

In this paper, we introduce an open source package — libLDB, for extracting a new binary descriptor called Local Difference Binary (LDB) [7-9]. libLDB provides a distinctive replacement to existing binary features that has similar construction runtime yet generates much fewer false correspondences and in turn reduces the runtime for the entire matching and post-verification process. libLDB is capable for real-time image matching applications running on both standard PCs and low-power mobile handheld devices, e.g. feature-based object detection and recognition, patching tracking, augmented reality, panorama stitching.

The rest of the paper is organized as follows. Sec. 2 presents a framework of LDB extraction followed by detailed description of the principle components of the library. In sec. 3, we evaluate the performance of LDB for mobile object recognition and tracking. Sec. 4 presents potential applications and sec. 5 concludes the paper.

2. LIBRARY COMPONENTS

We briefly overview the key idea of LDB followed by detailed description of libLDB implementation, an explanation of how to use libLDB and a simple application demonstration.

Given an image patch, LDB divides the patch into $n \times n$ equal-sized grid cells and extract average intensity I_{avg} and gradients, d_x and d_y, from each grid cell. Then, LDB compares I_{avg}, d_x and d_y between pairwise grid cells respectively: if the comparison result is larger than 0, the corresponding bit is set to 1, otherwise it is set to 0. In this way, we can generate a binary bit string for an image patch. To enhance the robustness of LDB descriptors, we employ Gaussian pyramid and dominant orientation estimation to achieve scale invariance and rotation invariance. In addition, a multiple gridding strategy (i.e. partition a patch into different number of grid cells) is also applied to encode the structure of at different spatial granularities, enhancing distinctiveness of LDB.

libLDB implementation consists of 5 principle components: Gaussian pyramid construction, dominant orientation estimation, upright/rotated integral image construction, binary tests on pairwise grid cells, and bit selection and concatenation. Fig.1 illustrates the workflow of LDB feature extraction.

2.1 Gaussian Pyramid Construction

In order to achieve scale-invariance for LDB descriptors, we utilize Gaussian pyramid technique and compute LDB descriptors at the corresponding level of the pyramid for detected points. Specifically, given an image $I(x, y)$, we repeatedly smooth the image with Gaussian filters $G(x, y, \sigma_i)$ with increasing sigma values σ_i to construct a Gaussian pyramid Pyr_i $(1 \le i \le L)$ (see Eq. 1). In libLDB, the scale factor between consecutive pyramid levels is set to 1.2 and the number of scales is set to 3.

$$Pyr_i = I(x,y) * G(x,y,\sigma_i) = \frac{1}{2\pi\sigma_i} e^{-(x^2+y^2)/2\sigma_i^2}, \quad 1 \le i \le L \tag{1}$$

Once a Gaussian pyramid is constructed, we select a proper level of pyramid to compute LDB descriptors. In particular, for point detectors (e.g. SIFT detector) which provide salient scale of a detected point we crop an $N \times N$ patch centered around the point from the salient scale of the pyramid and then compute a LDB descriptor from the patch. For other detectors (e.g. ORB detector) which do not estimate the salient scales, for each detected point we crop patches at all levels of pyramid and compute LDB descriptors for them.

2.2 Dominant Orientation Estimation

Rotation-invariance is achieved by computing a dominant orientation for every image patch and rotating a patch to its dominant orientation before computing its LDB descriptor. Many dominate orientation estimation methods can be used and libLDB implements the intensity moments-based method for its efficiency and robustness. Specifically, the intensity moment of a patch is defined as:

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y) \tag{2}$$

where $I(x,y)$ image intensity of location (x, y). With the intensity moments of a patch, the intensity centroid of the patch is defined as:

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}\right) \tag{3}$$

In most scenarios intensity centroid of a patch is offset from its geometric center. According to this fact, we construct a vector $\overrightarrow{OC}$ from the geometric center O to the intensity centroid C. The orientation of the patch then simply is:

$$\theta = \tan^{-1}(m_{01}, m_{10}) = \tan^{-1}\left(\frac{\sum_{x,y} yI(x,y)}{\sum_{x,y} xI(x,y)}\right) \tag{4}$$

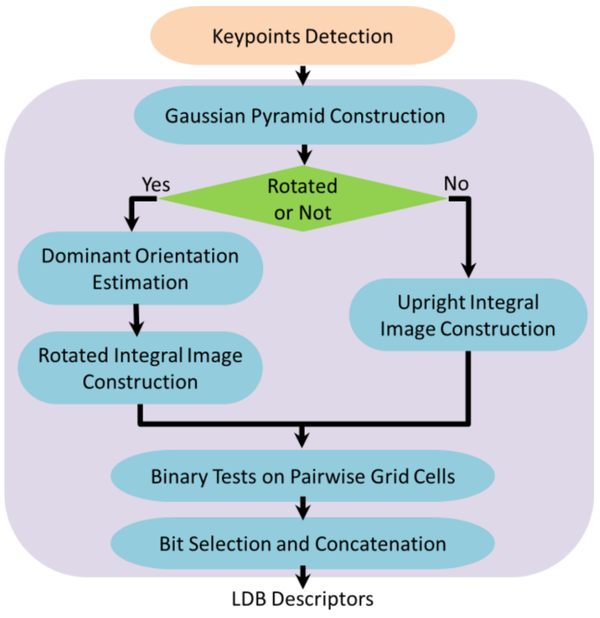

Keypoints Detection
Gaussian Pyramid Construction
Yes
Rotated or Not
No
Dominant Orientation Estimation
Upright Integral Image Construction
Rotated Integral Image Construction
Binary Tests on Pairwise Grid Cells
Bit Selection and Concatenation
LDB Descriptors

Fig.1: Workflow of LDB feature extraction

2.3 Integral Image Construction

libLDB leverages the integral image technique to efficiently compute average intensity I_{avg} and gradients d_x and d_y within a grid cell. Each value of an *upright integral image* at location (x, y) is calculated by summing up intensities above and to the left of (x, y). Accordingly, I_{avg}, d_x and d_y of a grid cell can be computed using 20 add/sub operations and two divide operations in total.

However, in order to achieve rotation invariance, image patches are rotated to their respective dominant orientations. In this case, the *upright integral image* of an entire image cannot be directly used because the patch axis is not parallel to the integral image axis. For these scenarios, libLDB provides an implementation for computing *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 the horizontal axis. Based on the *rotated integral image* of the patch, we can efficiently compute I_{avg}, d_x and d_y of any grid cell within the patch.

Two major computations for generating a *rotated integral image* lie in the calculation of rotated coordinates and interpolation for a steered patch. Intuitively, an efficient way to runtime of these two computations is to quantize the orientations and pre-storing the rotated coordinates in a lookup table. However, experimental results show that a fine orientation quantization usually leads to a large lookup table, and in turn, results in long runtime due to slow memory access. Therefore, libLDB computes rotated coordinates and uses nearest neighbor interpolation to generate a steered patch on-the-fly. Experimental result on Thinpad T420 shows that this implementation is faster than performing orientation quantization and using a lookup table.

2.4 Binary Tests on Pairwise Grid Cells

Once an upright integral image or rotated integral images are constructed, libLDB divides each image patch into $n \times n$ equal-sized grids, extract average intensity and gradients from each grid cell and perform binary test τ on a pair of grid cells (i and j) as:

$$\tau(Func(i), Func(j)) := \begin{cases} 1 & if\ (Func(i), Func(j)) > 0\ and\ i \neq j, \\ 0 & otherwise \end{cases} \quad (5)$$

where $Func(\cdot) = \{I_{avg}, d_x, d_y\}$.

The choice of the grid size *n* influences both robustness and distinctiveness of the LDB descriptor. Fine gridding enhances the distinctiveness of LDB while degrades its robustness since corresponding grids of two similar patches could be misaligned even for a small amount of patch shifting. Coarse gridding, on the other hand, results in a more stable but less distinctive descriptor. In libLDB implements a multiple gridding strategy, i.e. each image patch is partitioned in multiple ways (i.e. 2×2, 3×3, 4×4, and 5×5 in libLDB) from each of which a LDB binary string is derived. The strings resulting from all the partitions are then concatenated to form an initial LDB descriptor.

2.5 Bit Selection and Concatenation

Performing binary tests on pairwise grid cells at four gridding levels (i.e. 2×2, 3×3, 4×4 and 5×5) results in a string of 1386 bits. Many of those bits are actually correlated with each other and may not be distinctive enough. We select a subset of bits out the entire bit set by a modified AdaBoost method [8]. In libLDB, we provided the 256 selected bit indexes which were trained based on the Library dataset [15]. Finally, libLDB concatenates those selected bits to form a 32-byte binary description.

2.6 How to Use libLDB

libLDB provides two public functions for users to call:

1) **LDB::LDB(int _patchSize):** a constructor in which parameters of LDB are initialize. Users can determine the patch size for computing LDB descriptors. By default, this value is set to 48.

2) **void LDB::compute(const Mat& _image, vector& _keypoints, Mat& _descriptors, bool _flag)**: a function which performs LDB feature extraction for an input grayscale image.

   _image: an input grayscale image

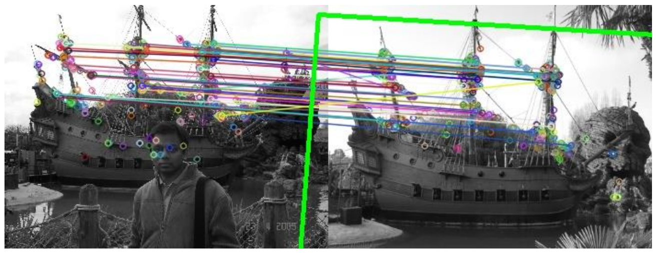
   _keypoints: input keypoints detected by a point detector

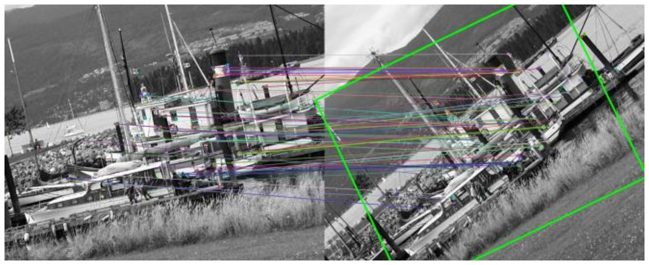
   _descriptors: output 32-byte LDB binary descriptors

   _flag: an input Boolean parameter determines whether LDB descriptors are steered or not. If it is set to 1, dominate orientation for every keypoint will be estimated and *rotated integral images* are calculated; otherwise, *upright integral image* of an entire image is calculated.

We provide a sample code to demonstrate how to use libLDB for LDB extraction and image matching based on LDB. In this demo, we detect keypoints using ORB point detector and compute LDB descriptors. Then, for each keypoint we search its nearest neighbor using a brute-force matching. The putative matches (e.g. the ratio between the shortest and second shortest distance is lower 0.8) are then used in RANSAC-based homography estimation. The demo is developed based on Red Hat Enterprise Linux Server release 5.5 (Tikanga), OpenCV 2.4.0 and GCC 4.7.1.

Users can use the command "**./ldb _image1 _image2 _flag"** to execute the demo. **_image1** and **_image2** indicate the path of two input images respectively and **_flag** is a Boolean parameter determines which LDB is steered or not. Fig. 2 illustrates the outputs of demo, i.e. matching results of LDB for three images pairs with perspective changes, rotation changes and scale changes respectively. Lines indicate corresponding feature points matched based on LDB and green boxes indicate the homograph changes between two images.


(a)


(b)

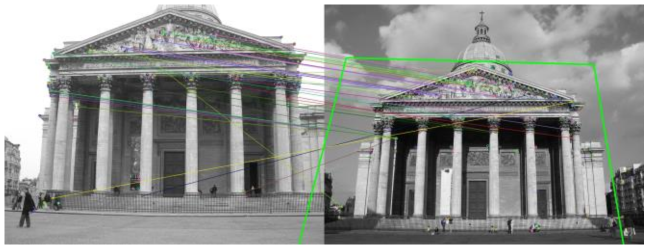

(c)

Fig.2: Matching results of LDB for image pairs with perspective changes (a), rotation changes (b) and scale changes (c), respectively.

3. PERFORMANCE

We evaluated the performance of LDB for mobile object recognition task by implementing a conventional pipeline. Specifically, the pipeline includes the following three steps: 1) ORB keypoint detection and LDB descriptor construction, 2) candidate images selection by matching each query LDB descriptor to all LDB descriptors in the database and returning top-ranked database image with most matches as candidate images, and 3) post verification to select the most relevant image with most number of consistent matches.

The evaluation is performed on a database contains 228 planar objects [16], 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 1140 query images.

In our experiment, with ORB, BRISK, FREAK and LDB all being binary descriptors, Locality Sensitive Hashing (LSH) is chosen

Desc.	Detection Rate (%)	Precision (%)	Constr. Time (ms)	Recog. Time (ms)
ORB-32	93.3	98.6	0.146	7.26
LDB-32	96.3	99.0	0.139	2.55
BRISK-64	97.7	99.4	0.034	7.64
FREAK-64	98.3	99.5	0.108	24.65
SURF-64	83.8	92.9	1.488	-

Table 1: Comparing performance of descriptors for recognizing 1140 manually captured images on Google Nexus 4 smartphone.

for approximate nearest neighbor (ANN) search instead of brute-forcing matching.

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) *Recognition Time*: the average time cost for searching the ANN of a query descriptor from the database;
All the time cost is recorded based on running the code on a single core of Google Nexus 4, operating at 1.5GHz and using Android 4.2 Jelly Bean operating system. The memory cost is read from the task manager of a PC.

Table 1 shows the comparison results. According to the results, LDB-32 achieves similar detection rate and precision as state-of-the-art binary descriptors. Considering the total time cost including runtimes for both descriptor constrcution and matching, LDB-32 is 2.8X faster than ORB, ~3X faster than BRISK and 9.5X faster than FREAK.

4. CONCLUSION

We developed an open source library for extracting ultrafast and highly distinctive binary feature LDB. Despite this project is still in an early development stage (without SSE or NEON acceleration on x86 and ARM processors respectively), it already shows fast runtime and good performance of recognition and tracking on mobile devices. Our plan is to further optimize the code and include hardware acceleration. In addition, we also plan to include Android source code of applying LDB to mobile object recognition and tracking, source code for salient bit selection based on AdaBoost learning.

5. REFERENCES

[1] Leutengger, S., Chli, M., and Siegwart, R.Y., BRISK: Binary Robust Invariant Scalable Keypoints. *In Proc. of International Conference on Computer Vision*, 2011, Barcelona, Spain.

[2] Alahi, A., Ortiz, R., and Vandergheynst, P. FREAK: Fast Retinal Keypoint, *In Proc. of Computer Vision and Pattern Recognition*, 2012.

[3] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G., ORB: An Efficient Alternative to SIFT or SURF. *In Proc. of International Conference on Computer Vision*, 2011, Barcelona, Spain.

[4] Rosten, Edward, and Tom Drummond. Machine learning for high-speed corner detection. *In Proc. of European Conference on Computer Vision*, 2006.

[5] Bay, H., Ess, A., Tuytelaars, T. and Gool, L.V., SURF: Speeded-Up Robust Features. *In Procs. of European Conference on Computer Vision*, 2006.

[6] Lowe, D. G., Distinctive Image Features from Scale-Invariant Keypoints. *In International Journal of Computer Vision*, 60, 2, pp. 91-110, 2004.

[7] Yang, X., and K.-T. Cheng. LDB: An ultra-fast feature for scalable Augmented Reality on mobile devices. *In Proc. of International Symposium on Mixed and Augmented Reality*, 2012.

[8] Yang, X., and K.-T. Cheng. Learning Optimized Local Difference Binaries for Scalable Augmented Reality on Mobile Devices. *IEEE Transaction on Visualization and Computer Graphics*, 2014.

[9] Yang, X., and K.-T. Cheng. Local Difference Binary for Ultra-fast and Distinctive Feature Description. *IEEE Transaction on Pattern Recognition and Machine Intelligence*, vol. 36, no. 1, 2014.

[10] VLFeat: http://www.vlfeat.org/

[11] OpenSURF: http://www.chrisevansdev.com/computer-vision-opensurf.html

[12] OpenCV: http://www.chrisevansdev.com/computer-vision-opensurf.html

[13] Calonder, M., Lepetit, V., Strecha, C., and Fua, P., Brief: Binary Robust Independent Elementary Features. *In Proc. of European Conference on Computer Vision*, 2010.

[14] Chum, O., and Matas, J., Matching with PROSAC — progressive sample consensus. *In Proc. of Computer Vision and Pattern Recognition*, vol 1, p: 220–226, 2005.

[15] Winder, S., Hua, G., and Brown, M., Picking the Best DAISY, *In Proc. of Computer Vision and Pattern Recognition*, 2009.

[16] Yang, X., Liu, Q., Liao, C.Y., and Cheng, K.-T. Large-scale EMM identification Based on Geometry-Constrained Visual Word Correspondence Voting. *In Proc. of ACM International Conference on Multimedia retrieval*, 2011.