LIP-VIREO in Action

Version 1.40 stable release

Author: Wan-Lei Zhao

July 18, 2022

Preface

Image local features (also known as keypoint features) have been widely explored in the last decade due to its unique advantages over image global features. Comparing with global features, image local feature characterizes image at a more fine-grained level. They have been successfully applied in wide range of applications and systems, such as wide baseline matching, object retrieval and recognition, and near-duplicate image/video detection. Due to the success of the keypoint detectors (e.g., Harris and DoG) and descriptors (e.g., SIFT and SURF), these tasks are no longer as challenging as they were thought many years ago. Tasks such as visual object detection and recognition become also tractable. Although there are already many contexts the keypoint feature can be fit in, it seems still too early to say we have witnessed the best that keypoint features could bring us.

The development of this toolkit is a reflection of state-of-the-art research on keypoint features. Since as indicated in many research works, it is hard to find any detector or descriptor performs the best all the time, allowing user to choose among as many as possible options is expected. Such that user may explore and try them with different combinations under different contexts. So it would not be surprising some new properties will be discovered from long existing features some day. Bear this in mind, I try to cover popular detectors and descriptors as much as possible in the toolkit. However, I might not be able to catch up with the recent progress in the literature due to the limited spare time, limited ability and most of all, fast evolving of this area. To this point, beg your tolerance and patience!

The development of this toolkit is out of the personal interests. I am not regretful that hundreds of weekends were spent on this interesting topic. Instead, it is my pleasure that finally it turns out to be helpful for others as it grows reasonably big. Wish it could be more stable and powerful as time goes on. However, for the limited time I can spend on it, it is not promised to fix the known bugs timely and work out a new release regularly. However any bug reports or suggestions are appreciated all the time.

In this manual, the basic theories and the implementation of each detector and descriptor are reviewed briefly. We try to outline the brief idea behind each detector and descriptor. This helps user to understand how they work and what the potential drawbacks they might have. User is also provided with the basic hints based on which one is able to choose the suitable combination (between detector and descriptor) for his specific task. In order to avoid possible confusion between our implementation and the original implementations, the differences that our implementation from original paper are highlighted. For the user who wants to repeat the implementation, referring to the

original papers or implementations is encouraged.

In addition, a brief evaluation on the performances of the detectors and descriptors are presented in the end of corresponding chapters. The evaluations are conducted on popular datasets, which makes it easier to objectively judge the performances of different detectors and descriptors. However, the evaluation might only reflect the behaviors of the detectors and descriptors of certain aspects, it is therefore not surprising that the performance rank changes when they are tested under other situations. The third part of this manual is to guide user to work with **LIP-VIREO** smoothly. It includes the instructions on how to detect keypoints, how to extract the descriptors and how to visualize the detected features.

Currently, **LIP-VIREO** supports keypoint feature extraction of various types. The detectors it covers are: *Harris-Laplacian* [1], *Hessian* [1], *Hessian-Laplacian* [1], *DoG* [2], *LoG* [6] , *Fast Hessian* [10] and *Dense sampling*. For each detector, one can choose to either output keypoints or display them in the image or both. The following keypoint descriptors have been integrated in LIP-VIREO. They are *SIFT* [2], *Flip invriant SIFT*, which is newly proposed by us, *SURF* [10], *AoD*, SPIN image [5], RIFT [5] and Enhanced RIFT, FIND [18] and Steerable Filters which is also known as '*Local Jet*'. After several rounds of re-factoring, the framework and output format of **LIP-VIREO** become stabilized. For the sake of compatibility, new version is kept to behave as much as it used to be. However, major update could happen for good reason. For each new version, the release tries to cover all popular platforms such as Windows, Linux, MacOS and Unix.

LIP-VIREO is fully implemented and maintained by me. In its early version, the implementation of DoG detector is inspired from the PCA-SIFT source code shared by Dr. Ke Yan.

Contents

Chapter 1

Detectors

LIP-VIREO integrates seven most effective detectors in the literature: Harris-Laplacian, Hessian-Laplacian, Hessian, Fast Hessian (SURF detector), Difference of Gaussian (DoG), Laplacian of Gaussian (LoG) and dense detector. In this chapter, fundamental theories and techniques about detectors are first reviewed. Then a brief introduction on our implementations are given respectively. Although we try to stick to the descriptions in the original papers, possible changes have been made due to several reasons. In this case, we will point out the differences between our implementation and the one from the original authors. Finally, a brief evaluation is conducted on the dataset provided by INRIA-LEAR [3]. It evaluates the performances of different detectors under different image transformations. This evaluation acts as a reference for users. However, we don't verdict that the performance rank will be kept the same across various circumstances. The exploration on different combinations between detector and descriptors in different applications is left to the user. The commands about how to choose among different detectors can be found in Section 3.1.

1.1 Harris and Harris-Laplacian Detectors

For approaches which are based on the saliency functions, different functions are adopted to measure the saliency of the region around a pixel. Harris [1] detector is mainly based on the second moment matrix which is defined in Eqn. 1.1 for a point X.

$$\mu(X, \sigma_I, \sigma_D) = \sigma_D^2 g(\sigma_I) * \begin{bmatrix} L_x^2(X, \sigma_D) & L_x L_y(X, \sigma_D) \\ L_x L_y(X, \sigma_D) & L_y^2(X, \sigma_D) \end{bmatrix}, \tag{1.1}$$

where σ_I is the integration scale, σ_D is the differentiation scale and L_g is the derivative computed in the g (x or y) direction. This matrix mainly describes the gradient distribution in a local neighborhood of point X. The local derivatives are computed with Gaussian kernels of the size determined by the local scale σ_D (differentiation scale). The derivatives are then averaged in the neighborhood of the points by smoothing with a Gaussian window of size σ_I (integration scale). The eigenvalues of this matrix represent two principal signal changes in the neighborhood of a point. Based on this function, Harris detector favors pixels which hold large curvature values in both principal directions.

The selection function is therefore defined as Eqn. 1.2.

$$Harris(X, \sigma_I, \sigma_D) = |\mu(X, \sigma_I, \sigma_D)| - \alpha * trace^2(\mu(X, \sigma_I, \sigma_D)), \tag{1.2}$$

where α is a constant. Consequently, local interest point is identified at where one pixel attains local maxima with respect to *cornerness*. The local maxima can be identified after non-maximal suppression.

In order to achieve scale invariance, an appropriate scale must be chosen for each detected local interest point. This process involves in seeking local extrema in the scale space. In the scale-adapted second moment matrix (Eqn. 1.1), parameter σ_I determines the scale of local region centering on point X. Different σ_I result in different local maxima of function 1.2. However, not all the local maximas generated by different σ_I are meaningful. According to [6], only local maximas which achieve local extrema in scale space shows stable and corresponds to local canonical structures. Furthermore, σ_D must be determined. To simplify the problem, σ_D is related to σ_I by a constant ratio, e.g., $\sigma_D = 0.8 \cdot \sigma_I$. As a result, the problem of seeking proper parameters σ_I and σ_D has been reduced to searching for local extrema in the scale space.

However, as indicated by Lindeberg [6], Eqn. 1.2 rarely attains maxima in the scale space. Instead, if the region saliency is measured by Laplacian-of-Gaussian function (as shown in Eqn. 1.3) in the scale space, local extrema in the scale space can be more precisely defined.

$$LoG(X, \sigma_I) = \sigma_I(L_{xx}(x, \sigma_I) + L_{yy}(x, \sigma_I)), \tag{1.3}$$

where L_{gg} denotes the second order derivative in direction g. As a result, for Harris-Laplacian detector, instead of measuring saliency of each pixel by Eqn. 1.2 solely, Eqn. 1.3 also applied on each pixel. This process has been repeated for multiple scales (by increasing σ_I constantly). The final keypoints are localized in X-Y space where Eqn. 1.2 attains local maxima and Eqn. 1.3 attains local extrema simultaneously.

Comments The search in scale space can be viewed as a simulation of viewing objects from different distances (scales). The local extrema in the scale space corresponds to the scale (distance) from which the object has been captured as a uniform and distinctive local structure. Choosing Eqn. 1.3 as the saliency function in scale space is more or less empirical [6].

1.2 Hessian and Hessian-of-Laplacian Detectors

Different from Harris detector, given Hessian matrix for point X:

$$H(X, \sigma) = \begin{bmatrix} L_{xx}(X, \sigma) & L_{xy}(X, \sigma) \\ L_{xy}(X, \sigma) & L_{yy}(X, \sigma) \end{bmatrix}, \tag{1.4}$$

where σ is the Gaussian smoothing parameter. Eqn. 1.6 defines the saliency function solely based on the determinant of Hessian matrix as following:

$$\mathcal{H}(P, \sigma_D) = \begin{bmatrix} L_{xx}(P, \sigma_D) & L_{xy}(P, \sigma_D) \\ L_{yx}(P, \sigma_D) & L_{yy}(P, \sigma_D) \end{bmatrix} \tag{1.5}$$

Similar to Harris-Laplacian, selecting proper σ_D is required. This involves the construction of scale space with Eqn. 1.5. Hessian points are therefore defined at which Eqn. 1.5 attains local extrema in spatial and scale space.

Actually, to select proper σ_D in the scale space, we have another choice available, namely, the Laplacian-of-Gaussian function. If the detected points are required to attain local extrema in Eqn. 1.3 in scale space instead, we obtain another detector: Hessian-of-Laplacian.

$$Hessian(X, \sigma) = det(H(X, \sigma)) \times \sigma^4 \tag{1.6}$$

Comments Notice that Eqn. 1.3 is shared between Harris-Laplacian and Hessian-Laplacian. While Eqn. 1.5 is shared between Hessian and Hessian-Laplacian detectors. As a result, it is not surprising that keypoints obtained from different detectors can still be matched.

1.3 Laplacian of Gaussian

The saliency measurement for `LoG` is mainly based on Eqn. 1.3. The points which attain local maxima on Eqn. 1.3 in their spatial and scale spaces simultaneously are selected. Comparing to Harris-Laplacian and Hessian-Laplacian, the saliency function in X-Y space has been replaced with Laplacian function. As a result, it might share the same group points with Harris-Laplacian and Hessian-Laplacian.

1.4 Difference of Gaussian

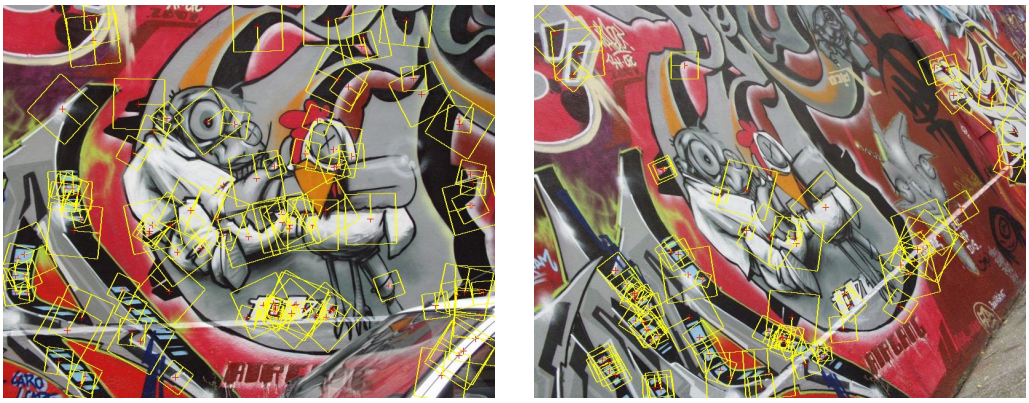
The computation cost of Eqn. 1.3 is expected to be relatively high since it involves estimation of the second derivatives in x and y directions respectively, a more efficient way is to approximate *LoG* via Difference of Gaussian (DoG) [2] which only requires convolving images in a constant manner (as indicated in Eqn. 1.7).

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma), \tag{1.7}$$

where σ is the Gaussian smoothing parameter and k is an integer multiplier. David G. Lowe [2] demonstrates the effectiveness of this approach. Although *DoG* is computationally efficient, localization precision in both spatial and scale space has been sacrificed. To alleviate this issue, *Taylor* expansion on Eqn. 1.7 is adopted to approximate the exact local extrema. This scheme turns out to be quite successful. In [1], K. Mikolajczyk and C. Schmid also show that *DoG* detector outperforms *LoG* based detectors in terms of speed efficiency. As indicated later, the repeatability score of DoG detector is also pretty high.

1.5 Fast Hessian

Fast Hessian is the detector of SURF feature [10]. The basic idea is to calculate Eqn. 1.6 in an efficient way with the help of integral images. In order to allow fast calculation,



(a) Graffiti (the 1st) (b) Graffiti (the 4th)

Figure 1.1: Fast Hessian points on Graffiti sequence [7].

Eqn. 1.6 has been approximated by Eqn. 1.8.

$$Det(H_{approx}) = D_{xx}D_{yy} - (0.9D_{xy})^2, \tag{1.8}$$

where D_{xx}, D_{yy} and D_{xy} all can be calculated efficiently using box filters. Follow the way described in this note [11], we further normalize Eqn. 1.8 by the size of box filter. Notice that, this operation will not convert non-extrema into a local extrema. It only changes the saliency score of a keypoint. Following the original implementation, the detection is performed in four scales and three octaves. Due to the considerable loss in the approximation, keypoint localization in either X-Y or scale space cannot be precise. Similar to DoG detector, *Taylor* expansion on Eqn. 1.8 is adopted to approximate the exact location of the extrema. Based on our observation, our implementation performs fairly well. In the original SURF detector, it has been coupled with SURF descriptor for speed concern. However, it is not necessary if time cost is not the first priority. In our implementation, Fast Hessian with SIFT descriptor demonstrates fairly stable performances in various contexts.

1.6 Dense Sampling

Dense sampling has shown considerably better performances over well designed detectors in object detection tasks. In version 1.06 and later, we incorporate implementation of dense sampling. It samples pixels in multiple scales and multiple octaves. In addition, the dominant orientation for each sampled patch will be detected. This information is considered when we calculate the descriptor for the sampled patch. The default sampling rate in our implementation is one point for every 5 pixels for both x and y directions. However, user is allowed to specify it in the configuration file by set 'step=n', where n is an integer larger than zero. Once it has been set, sampling rate becomes one point for every n pixels.

Different from most of the implementations in the literature, in **LIP-VIREO** the densely extracted features have been rotated to its dominant orientation.

1.7 Affine Estimation

In order to adapt to the local structures, detectors like Harris-Laplacian, Hessian-Laplacian and Laplacian-of-Gaussian provide affine estimation for detected keypoints. Particularly, for Harris-Laplacian, the estimation is based on Eqn. 1.2 and while for Hessian-Laplacian and Laplacian-of-Gaussian, the affine region is estimated by Eqn. 1.5. Note that, we do not fully adopt iteration procedure described in [1] for the affine adaptation. Current implementation is equivalent to only one round of iteration presented in [1].

To enable the affine estimation in **LIP-VIREO**, user is suggested to set "affine=yes" item in the configuration file. Based on this setting, **LIP-VIREO** outputs the three parameters for the affine region in the ".key" file. These three parameters are a, b and c which determine the affine matrix (shown in Eqn. 1.9). The matlab code which can be downloaded in **LIP-VIREO** homepage helps user to draw the affine region. Accordingly, based on this setting the descriptor is calculated in the affine region centering around the keypoint. According to our observation, keypoint features which are generated with such affine adaptation show few performance difference comparing with the one without the adaptation.

$$A = \begin{bmatrix} a & b & 0 \\ b & c & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.9}$$

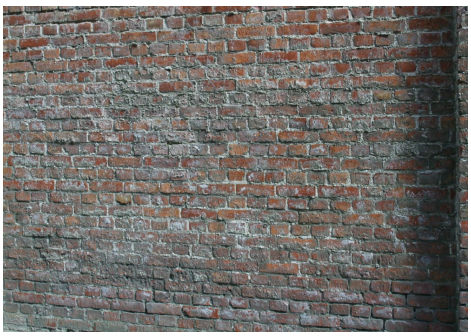
1.8 'non' detector

The integration of 'non' detector is an effort in responding to the requirements from couple of users, who are wishing to detect keypoint by themselves. The keypoints are supplied to **LIP-VIREO** for extracting the descriptors or displaying keypoints in the images via **LIP-VIREO**. The 'non' detector in itself does nothing more than loading keypoints from the user specified location (specified by '-kpdir' option). Please be noted that the input keypoint file must be suffixed with ".keys" and subject to the format shown in Appendix III.

1.9 Affliated Properties for the keypoint

Characteristic Scale For all the scale invariant detectors, each keypoint is assigned with a scale, which basically regularizes a canonical region centering around the keypoint. The descriptors extracted within this region will achieve scale-invariance. In the keypoint visualization, user is provided with the option to display this scale (circle or ellipse) around the keypoint. Additionally, this information can be quite helpful. It can be capitalized for fast geometric verification as demonstrated in [16, 17].

Sign of *LoG* For all the scale invariant detectors we discussed so far, two functions are adopted to achieve scale invariance, they are Laplacian-of-Gaussian (Eqn. 1.3) and Hessian Eqn. 1.5. For LoG, as discussed in [10], the sign of Eqn. 1.3 implies different



(a) Wall (the 1st)

(b) Wall (the 4th)

(c) Boat (the 1st)

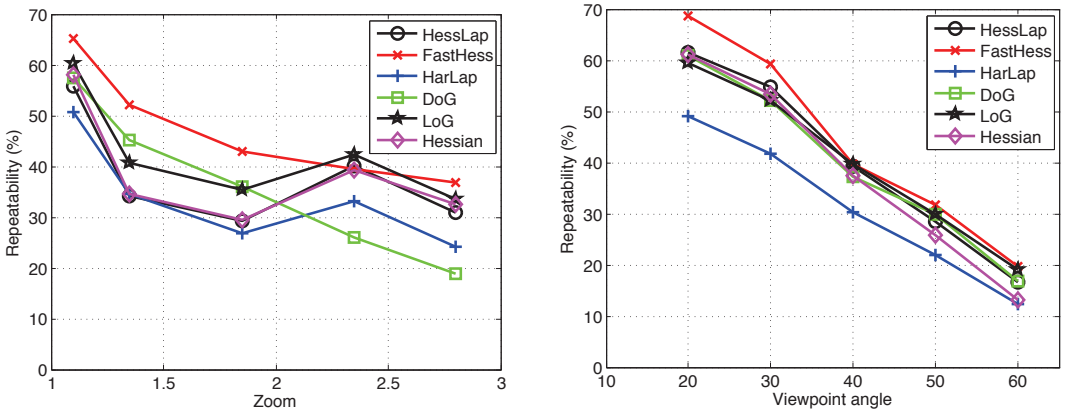
(d) Boat (the 4th)

Figure 1.2: Testing image series from VGG [19]. Wall and Boat image series, which are under affine transformations, scaling and rotation.

local structures. The sign of Eqn. 1.3 reflects the general direction of gradient field either pointing outward or inward. Usually structure with negative sign will never match to structure with positive sign in Eqn. 1.3. This simple fact leads to a good property which is helpful in many contexts. For example, in keypoint matching, computation costs can be reduced if we restrict that one patch can only match to another patch with the same sign. At the same time, the false matching rate is expected to reduce to its half. In terms of Hessian function (Eqn. 1.5), although Eqn. 1.3 is not on the shelf, only few extra efforts are required to derive Eqn. 1.3 when the second order derivatives are ready. As a consequence, the sign of Laplacian turns out to be an additional property available to all the scale invariant detectors that are discussed here. User can refer to Section 3.1 for more operational details.

1.10 Performance Evaluation on Detectors

In this section, the performance of keypoint detectors is evaluated in two scenarios. In the first scenario, detectors are tested with object recognition task. In particular, the test image series from VGG [7] have been adopted for evaluation (sample images are shown in Fig. 1.2). The repeatability score [1] (the higher the better) for each



(a) Incremental scale changes on Boat series

(b) Incremental viewpoint changes on Wall series

Figure 1.3: Performance of Difference-of-Gaussian, Harris-Laplacian, Hessian-Laplacian, Fast Hessian, Laplacian-of-Guassian using detectors extracted by LIP-VIREO.

Table 1.1: Similar Image Retrieval Performance (mAP) on Holidays [13]. Processing efficiency is tested on a PC with 2.4GHz CPU setup.

	Harris	Harlap	Hesslap	Hessian	LoG	DoG	Fast-Hessian	Dense	HesAff [1]
VLAD* [12]	55.3	73.9	72.7	68.2	71.3	73.3	60.9	77.2	70.6
BoW [14]	-	62.0	62.6	57.3	60.3	60.4	54.0	-	61.2
Time (s)	0.92	2.0	1.23	1.15	1.15	0.64	0.56	2.01	0.88

detector is reported, which basically reveals the robustness of the detector against image transformations (such as affine, scaling and rotation).

In the second scenario, the detectors have been tested under two different image retrieval frameworks, namely BoW [14] and Fisher Vector (VLAD* [12]). In this experiment, Holidays dataset [13] has been adopted, which consists of $1,491$ images. Among them, 500 images are selected as the queries. The extracted keypoints from each detector have been described with rootSIFT [4]. For VLAD*, the vocabulary size is fixed to 64. While for BoW, the vocabulary size is fixed to 32k. Vocabularies are trained respectively for features from different detector. During the training, an independent image set has been used. The performance of all detectors that are integrated with **LIP-VIREO** is compared with Hessian-Affine (HesAff) [1] from [7]. The results are shown in Table 1.1. In the last row of the table, the average time for processing one image (sized of 512×320) for each detector is reported. Notice that the time cost includes the time for keypoint detection and descriptor extraction.

Chapter 2

Descriptors

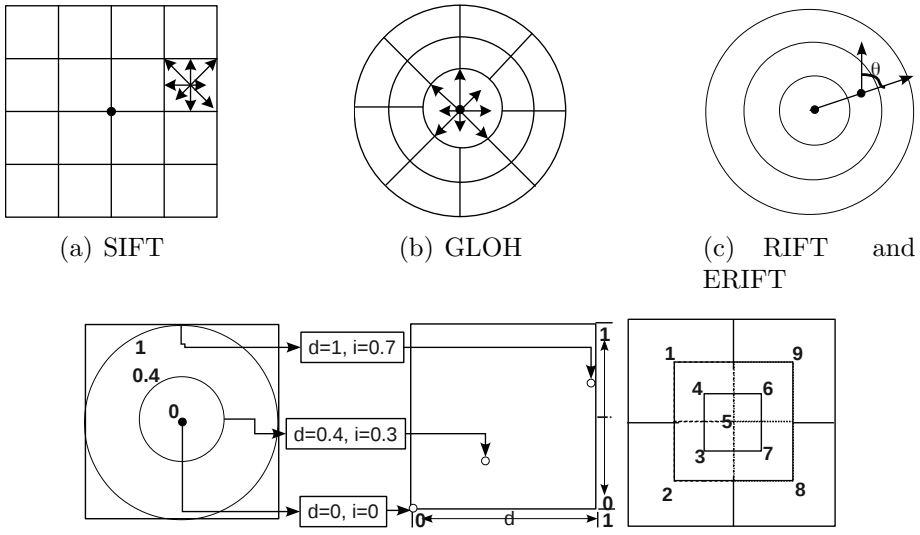
There are ten descriptors integrated with **LIP-VIREO**. They are SIFT and its variants, namely, flip invariant SIFT (F-SIFT), PCA-SIFT and FIND. Additionally, one can also find implementation of descriptors such as SURF, AoD, SPIN, RIFT, ERIFT and Steerable Filters in the package. In this section, we review these descriptors and our implementation in brief. Finally, a brief evaluation on these descriptors is presented.

2.1 SIFT

SIFT is proposed by D. Lowe in [2]. It has been proved to be useful for various tasks such as object classification, panorama generation, wide baseline matching and ND image identification. Figure 2.1 shows the partition scheme of SIFT. Basically, SIFT applies grid partitioning on the local patch around a keypoint. Gradient of a pixel within each partition block has been quantized according to its orientation. Typically, the number of quantization bins is 8. According to [2], this setting reaches to the best performance. Before the quantization, this local patch is rotated to its dominant orientation. This operation enables the extracted feature to be invariant to rotation transformation. The dominant orientation is basically estimated also based on orientation quantization in the gradient field of this local patch. Usually, the region used for dominant orientation calculation is small portion of the local patch (e.g. concentric region of the patch with half radius length). To achieve better performance, the histogram are further weighted firstly by its gradient length, secondly by a Gaussian window centering around the keypoint. Our implementation of SIFT is every bit like what described in [2], except that we do not truncate the feature dimension that is higher than 0.2 after l2-normalization. We find that truncating these strong signals is in general detrimental to the performance. In particular, if one wants use root-SIFT [4], feature without truncating the large signals shows much better performance.

2.2 Flip invariant SIFT

In version 1.06 and later, the flip invariant SIFT is incorporated, which is proposed by us. After an comprehensive studies with near-duplicate detection and object recognition



(a) SIFT (b) GLOH (c) RIFT and ERIFT

(d) SPIN (e) FIND

Figure 2.1: Partition schemes for SIFT, GLOH, RIFT and ERIFT.

tasks, we show that this new descriptor is able to maintain similar performances as SIFT feature for cases flip is not involved, while outperforms SIFT apparently on the cases that flip operation is observed. Users are referred to our recent technical report [15] for more details.

2.3 FIND

FIND [18] is proposed as a variant of SIFT which is enabled with flip invariance. Similar to SIFT, histogram on gradients is generated. Comparing with SIFT, FIND has a quite different partition scheme, which allows overlapping between blocks (as shown in Figure 2.1(e)). As indicated in the figure, the number of pixels is not distributed evenly in each block. The flip invariance is achieved by first characterizing patches as either left-pointing or right-pointing. A flip indicator (left or right) is thus kept in the descriptor for each patch. when a left-pointing patch matches with a right-pointing patch, feature vector from one of them is reversed. Then the flip invariance is actually achieved during the matching. In our implementation, we find such kind of indicator is unnecessary once if we normalize (flip) the patch to one orientation (e.g., left-pointing). As a result, FIND becomes quite similar to F-SIFT except they rely on different information for flip normalization and they adopt different partition schemes. Comparing with F-SIFT, either the partition scheme or orientation estimation adopted by FIND is less stable. In terms of computation cost, the latter is cheaper to obtain.

2.4 Steerable Filters

Steerable filters has been used in many contexts. It is also known as local jet and textons. They are actually partial derivatives obtained based on series of Gaussian kernels. We follow the implementation described in [3] in which a 41×41 Gaussian window centering around the keypoint is employed. We calculate the derivatives up-to the forth order. This results in 15 dimensional feature vector including pixel value of the keypoint itself. The intensity value and different order of derivatives are arranged in following order.

v, dx, dy, dxx, dxy, dyy, $dxxx$, $dxxy$, $dxyy$, $dyyy$, $dxxxx$, $dxxxy$, $dxxyy$, $dxyyy$, $dyyyy$

2.5 SPIN

The intensity domain SPIN image incorporated in `LIP-VIREO` is a two-dimensional histogram encoding the distribution of image brightness values in the neighborhood of a particular reference (center) point. The quantization scheme is illustrated in Figure 2.1(d). The implementation is every bit like the description in [5]. The two dimensions of the histogram are d, distance from the center point, and i, the intensity value. The "slice" of the spin image corresponding to a fixed d is simply the histogram of the intensity values of pixels located at a distance d from the center. Since the d and i parameters are invariant under orthogonal transformations of the image neighborhood, spin images offer an appropriate degree of invariance for representing affine normalized patches. In our implementation, we used 10 bins for distance and 10 for intensity value, resulting in 100-dimensional descriptors.

We implement the SPIN image as a "soft histogram" where each pixel within the support region contributes to more than one bin. Specifically, the contribution of a pixel located in x to the bin indexed by (d, i) is given by Eqn. 2.1.

$$exp(-\frac{(|x - x_0| - d)^2}{2\times\alpha^2} - \frac{(I(x) - i)^2}{2\times\beta^2}), \tag{2.1}$$

where x_0 is the location of the center pixel, and α and β are the parameters representing the "soft width" of the two-dimensional histogram bin. Note that the soft histogram can be seen as a set of samples from the Parzen estimate (with Gaussian windows) of the joint density of intensity values i and distances d.

According to our observation, SPIN shows excellent performances in the context that only rotation and flip transformations are involved. Its performance drops dramatically when it faces images with scale transformation. In addition, extracting SPIN feature could be very slow (roughly 10 times slower than that of SIFT).

2.6 SURF and AoD

In general, SURF descriptor shares the same partition scheme as SIFT. However, in each block, in stead of generating histogram on gradients, SURF aggregates on Haar wavelets of different channels. As SURF detector, box filters are applied also for speed efficiency. The aggregation is performed on dx, $|dx|$, dy and $|dy|$. This results in a *64* dimensional vector. In the original implementation, the author also discussed the variant of separating negative wavelets from positive ones, however this flexibility is not incorporated in our implementation.

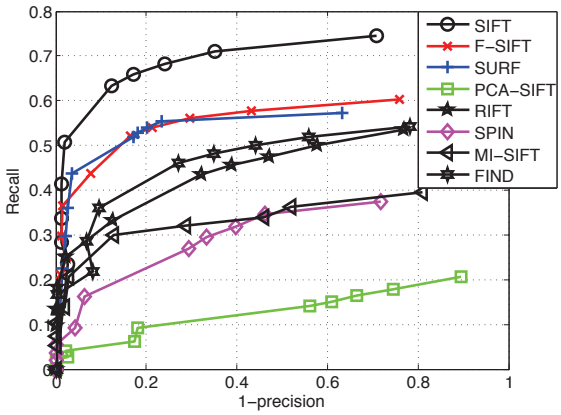
The implementation of AoD is inspired by SURF descriptor. Comparing with SURF, the only difference is that the box filters have been replaced with derivatives of each pixel in the patch. Due to the higher cost of calculating derivatives, AoD is less efficient than SURF.

We evaluated our SURF implementation by matching near-duplicate image pairs. However, the performance is unsatisfactory although we try to follow every detail described in the original paper [10]. The performance difference can be attributed to difference in implementation details which cannot be found in the paper or inappropriate parameter tuning. Currently, the performance from Fast Hessian combined with SURF cannot compete with the original implementation. For this reason, if one wants to perform evaluation related to SURF in his scientific study, choosing the original implementation is strongly recommended. Nevertheless, we have to point out again, when *Fast Hessian* combined with other descriptors such as *SIFT* or *F-SIFT*, it performs fairly well. So it is safe to choose this combination if speed efficiency is not the major concern.

2.7 RIFT and ERIFT

To obtain a complementary representation of local appearance of normalized patches, Lazebnik and et. al have developed a rotation-invariant descriptor that generalizes Lowe's SIFT [2]. The normalized circular local patch has been partitioned into concentric rings (as shown in Figure 2.1(c)). Rings share the same width. For this reason, different ring covers different number of pixels. In other word, the sizes of histogram are different. Apparently, the outer rings have larger coverage. To maintain rotation invariance, this orientation is measured at each point relative to the direction pointing outward from the center. We use four rings and eight histogram orientations. It finally yields 32-dimensional descriptors.

ERIFT is short for Enhanced Rotation Invariant Feature Transform (ERIFT). It is an enhanced version of Rotation Invariant Feature Transform [5]. ERIFT follows the same partition scheme as RIFT on a local patch. But different from RIFT, we carefully set the width of each ring, to make sure these rings cover equal number of pixels. Similar to RIFT, the gradient orientation histogram is computed within each ring. To maintain rotation invariance, this orientation is measured at each point relative to the direction pointing outward from the center. We use eight rings and eight histogram orientations. In order to make it more distinctive, the quantization is performed on



(a) VIREO

Figure 2.2: Performance eight different descriptors with DoG detector on 8 image pairs which cover transformations such as scale, rotation, viewpoint changes, blur, changes in JPEG compression rate and lighting changes.

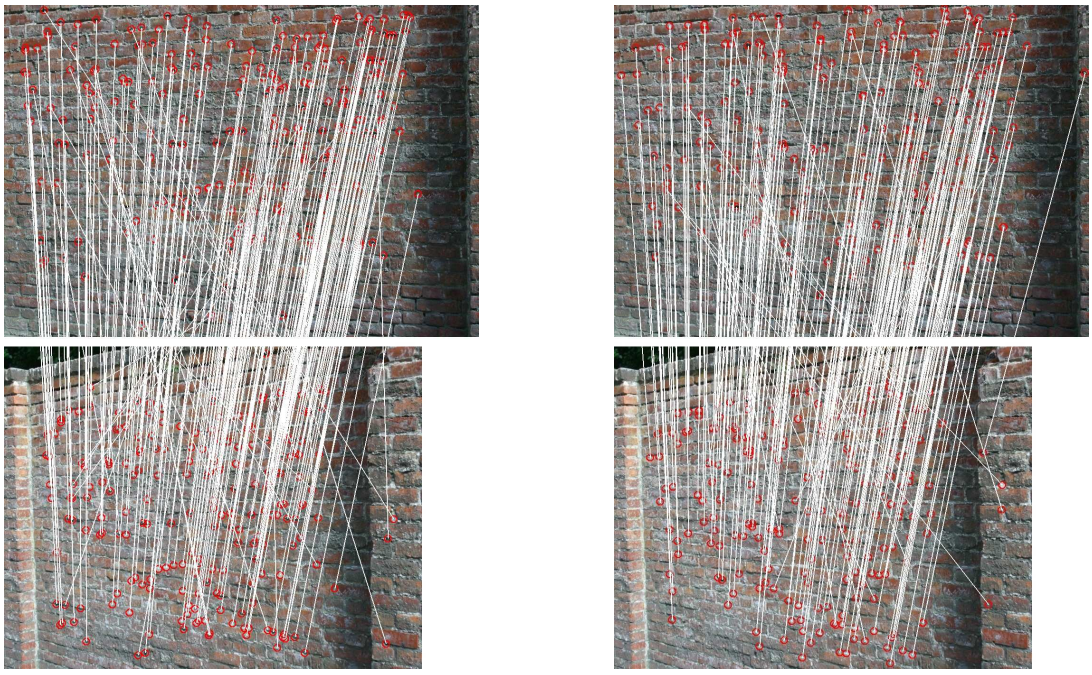
two perpendicular directions for each gradient value. It finally yields 128-dimensional descriptors. To enable flip invariance, for each pixel, in addition to quantizing along its gradient direction, we quantize it to the flipped gradient direction. ERIFT demonstrates better performance than RIFT in general cases when scale transformation is observed.

Comments Comparing with SIFT descriptor, both RIFT and ERIFT loose the constraint on pixel location, especially, along the log-polar direction. This enables RIFT and ERIFT to be invariant to both flip and rotation. However, as a side-effect, pixels are free to move around along this direction, while the final representation of RIFT or ERIFT keeps the same all the time. For this reason, the distinctiveness of the descriptors has been undermined. According to our experimental observation, neither RIFT nor ERIFT can be as discriminative as SIFT feature.

2.8 Performance Evaluation on Descriptors

Similar to keypoint detectors, we choose dataset [19] as the evaluation benchmark. Precision-recall curve is plotted for each descriptor, which basically shows the proportion of local regions between two similiar image have been correctly matched with each descriptor. We mainly demonstrate the effectiveness of descriptors from **LIP-VIREO**. For all the local features, we choose DoG as the detector. Figure 2.2 shows performances of SIFT, PCA-SIFT, PSIFT and GLOH. PCA-SIFT is obtained from Yanke while GLOH comes from VGG.

In Figure 2.3, the cros-matching results between images in wall series are presented. In the experiment, One-to-one and symmetric matching (OOS) has been used. We tested the cross-matching with DoG and Hessian detectors respectively in combination with SIFT descriptor.



(a) DoG+SIFT (b) Hessian+SIFT

Figure 2.3: OOS matching results from two different detectors.

Chapter 3

Command Line Options

3.1 Keypoint detection

Command as follows could produce keypoints for input image or images.

lip-vireo [**-img**|**-dir**] path **-d** [`dog`|`hess`|`harr`|`log`|`harlap`|`hesslap`|`dsurf`|`hesaff`|`dense`|`non`] [**-kpdir**] path **-c** file.conf

- **-img** `path`
- **-dir** `path`

For the above two options, user could only choose one at once. For '**-dir**' option, **LIP-VIREO** batchfully extracts keypoints for each image file in the specified folder while option '**-img**' processes only one image at once. Currently, the program only accepts images formatted in '**pgm**', '**ppm**', '**png**', '**bmp**' and '**jpg**'. Moreover, the file name should be suffixed with '*.pgm*', '*.ppm*', '*.png*', '*.bmp*' and '*.jpg*' correspondingly, otherwise, image will be simply ignored by **LIP-VIREO**.

- **-d** [`hess`|`harr`|`dog`|`log`|`harlap`|`hesslap`|`hesaff`|`dsurf`|`dense`|`non`]

This option allows user to choose among detectors such as Harris ('`harr`'), Hessian ('`hess`'), Hessian-Laplacian ('`hesslap`'), DoG ('`dog`'), Laplacian of Gaussian ('`log`'), Harris-Laplacian ('`harlap`'), Fast Hessian ('`dsurf`') and Dense sampling ('`dense`'). For all these detectors, only Harris is not scale-invariant since we only localize the keypoint in X-Y space only.

For detector '`dense`', this tool simply samples the pixels on the image densely. By default, one pixel is chosen for every 5 pixels. This default setting can be reset by put 'step=n' in the configuration file. Once it has been re-configured, the sampling rate becomes one point for every n pixels.

By specifying '`non`' detector, one is allowed to input keypoints detected by his own algorithms for display or descriptor extraction. When this special detector is selected, the '-kpdir' **MUST BE** specified to point out where user's keypoint files located.

- **-kpdir** `path`

Option '**-kpdir**' is used to indicate the destine directory where the features are saved. By default, the output keypoints file has the same name as the input image but with different suffix. We fix the suffix to '.keys'. It is a text file. Each line in the file writes in following format.

x_cord y_cord a b c iscale angle score

One line in the resulting file gives description for one keypoint. There are totally 8 items and are arranged in 8 columns. 'x_cord' and 'y_cord' are basically x and y location of the keypoint. 'a', 'b' and 'c' are parameters for the adapted affine region (ellipse shape). 'iscale' is the detected integration scale for the point. 'angle' indicates the dominant orientation of the local patch around the keypoint. While 'score' is the function value of detected point according to specified detector. In general, this value indicates the saliency of one keypoint, the higher of this value the more salient the point is.

- **-c** file.conf

This option is always required. In configuration file 'file.conf', several settings for additional options are provided. It is formatted as follows.

option1=value1
option2=value2

Basically, item before '=' (e.g. 'option1') is the option to be specified, setting is put after '='. Valid values can be assigned to the corresponding option. Configuration on five options, '**pcamat**', '**topk**', '**scale**', '**angle**' and '**affine**', are explained as follows. While the complete list of available options are shown in Table 3.1.

Option 'topk' allows user to specify the maximum number of keypoint that returns from one image. Once '**topk**' has been set, detector tries to choose top '**topk**' number of keypoints according to their saliency scores. When '**scale**' option is specified to 'yes', **LIP-VIREO** outputs the characteristic scale detected for each keypoint to the descriptor file. Similarly, when '**angle**' option is specified to 'yes', **LIP-VIREO** outputs the dominant orientation estimated for each keypoint to the descriptor file. Noted that 'scale' and 'angle' are options for descriptor extraction. For keypoint detection option '-kpdir', the scale and dominant orientation will be generated despite these two options are enabled or not.

Different from 'scale' and 'angle' options, 'affine' option takes effect for keypoint detection, descriptor extraction and keypoint display. When it is set to 'yes', detector estimates the affine region for each keypoint. Note that for the detectors, such as 'DoG', 'dsurf' and 'harr', which do not support affine adaptation, **LIP-VIREO** simply outputs an identity matrix. Finally, the detected affine regions will be outputed to keypoint files ('.keys') and the descriptors will be extracted from an ellipse instead of from a circle

Table 3.1: Check List of Available Options for Configuration

Option	Value	Default value	Take effect with option(s)
scale	yes/no	yes	'-d *'
angle	yes/no	yes	'-d *'
topk	integer	750	'-d *'
step	integer	10	'-d dense'
circle	yes/no	yes	'-drdir'
affine	yes/no	no	'-d *', '-p *', '-drdir'
format	vireo/vgg	vireo	'-d *', '-p *'
log	string (path of log file)	empty	'-d *', '-p *'

*: applies to all available choices for the option.

region. If '-drdir' option is specified, keypoints are displayed with ellipse scales in the image.

In current versions, an empty configuration file must be provided even user doesn't prepare to make any additional settings.

3.2 Descriptor extraction

Command as follows extracts descriptors from image or images.

lip-vireo [**-img**|**-dir**] path **-d** [*detector*] **-p** [$SIFT|LJET|SPIN|ERIFT|FIFT|FIND|SURF|AOD$] [**-dsdir**] path **-c** file.conf

Settings for options '**-img**', '**-dir**', '**-d**' and '**-c**' are the same as keypoint detection. So please refer to Section 3.1 for details. The focus of this section is on descriptor options: '**-p**' and '**-dsdir**'.

First of all, for descriptor extraction, a proper detector must be chosen for '**-d**'. This time, option '-kpdir' becomes optional. However, there is one exception. When detector option is set to '-d non', '-kpdir' is required to specify, i.e., user has to offer **LIP-VIREO** with the location of keypoint file(s). At the same time, one has to make sure the file is correctly formatted (refer to Appendix III) and the file name is suffixed with '.keys'.

- **-p** `SIFT|LJET|SPIN|ERIFT|FIFT|FIND|SURF|AOD`

In addition, one is required to specify the descriptor by '-p'. Available choices are SIFT, FIFT, RIFT, ERIFT, SPIN, SURF, AoD and LJet (Local Jet feature). User is free to choose either normalized the descritpors (l_2 norm) or features. To generate normalized descriptor, one can put character 'N' before the wanted descriptor option. For example, if '-p SIFT' is set, raw SIFT feature will be generated. Alternatively, if '-p NSIFT' is specified, **LIP-VIREO** outputs normalized feature vector.

- **-dsdir** `path`

Corresponding to '**-img**' and '**-dir**' options, '**-dsdir**' option specifies destine directory where to save the keypoint descriptor files.

By default (if 'affine=yes' is not specified in the configuration file), the descriptor is calculated within a circle region centering around the keypoint. Otherwise, the descriptor is calculated within a normalized affine region.

In addition to descriptor for each keypoint, in the descriptor file, user is allowed to keep affiliated information such as scale and rotation along with each detected keypoint. To achieve this, user needs to put following configurations in the configure file before the detection.
#output scale
scale=yes
#output angle
angle=yes
Similar to location 'x' and 'y', 'scale' and 'angle' are treated as affiliated properties of one keypoint and have been placed right after 'x' and 'y'. **Note that when user chooses to output scale and angle together, the output order for these two properties in the descriptor file is fixed to <scale, angle>.** For the format of the descriptor file, see details in Appendix I.

3.3 Visualization

3.3.1 Display Local Interest Point on the Image

In version 1.05 and later, **LIP-VIREO** allows user to display the keypoints on the input image with both location (with a cross in red) and the region (with an ellipse in yellow). The command is as follows.

lip-vireo [**-img**|**-dir**] path **-d** [`dog`|`hess`|`harr`|`log`|`harlap`|`hesslap`|`hesaff`|`dsurf`|`dense`|`non`] [**-kpdir** kpdir] **-drdir** imgdir **-c** file.conf

As shown in the command, in order to display points on the image(s), user has to specify the option '`-drdir`' to indicate where to save the image(s) with keypoints plotted. Actually, this option also invokes the displaying routine. Other options are the same as keypoint detection.

Note that the local interest point regions that are displayed can be either in circle or ellipse depending on both the option for detector and whether setting 'affine=yes' has been enabled in the file 'file.conf'. For detectors 'harlap', 'hesslap' and 'log', the regions are displayed in ellipses only when the setting 'affine=yes' has been set. For detector 'dog', the plotted regions are circles all the time since no affine estimation available with it. For Fast Hessian, the regions are displayed with squares all the time, which follows the tradition of original implementation. Similar to original implementation, the squares are rotated to their dominant orientation as well.

In version 1.06 and later, user is allowed to display keypoints with 'cross' only or

'cross' along with 'circle'. By default, only crosses are plotted in the image. In order to display local region with circle (or ellipse), user is required to specify 'circle=yes' in the configuration file.

For detector option '-d non', user MUST specify option '-kpdir' to point out the location of keypoint file(s). while for other detector options, '-kpdir' is optional.

3.3.2 Display Normalized Local Patches in Rows

In version 1.12 and later, **LIP-VIREO** allows user to display the detected keypoints in one image. The patches are extracted centering around the keypoint and normalized (to 41×41 patch) with its characteristic scale. The command is as follows.

lip-vireo [**-img**|**-dir**] path **-d** [`dog`|`hess`|`harr`|`log`|`harlap`|`hesslap`|`hesaff`|`dsurf`|`dense`|`non`] **-p sift -dvdir** dstdir **-c** file.conf

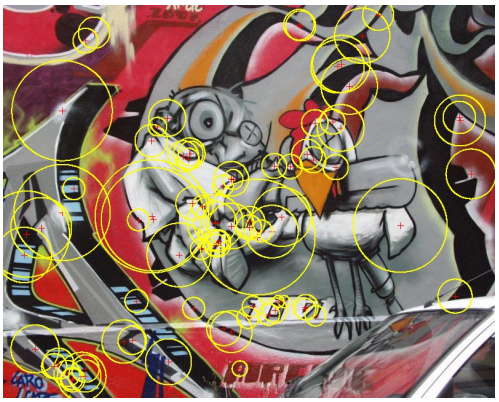
As shown in above command, function of displaying patches can be specified by '**-dvdir**' option. Once it has been chosen, all the detected keypoints from the input image will be displayed in one image. The normalized patches are organized into columns and rows. The image(s) is/are saved to 'dstdir'. Normalized patches of Harris-Laplacian and Fast Hessian points are shown in Figure 3.1(c) and (d) respectively.

(a) Harris-Laplacian points

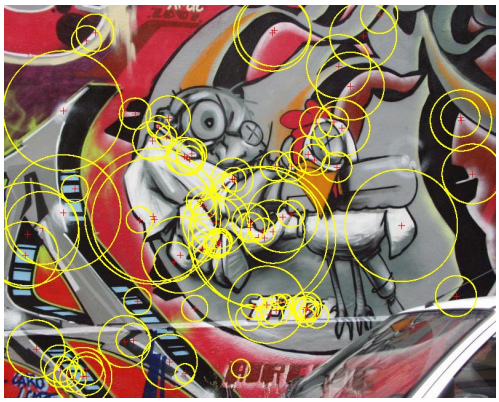
(b) Fast Hessian points

(c) Normalized Harris-Laplacian patches

(d) Normalized Fast Hessian patches



(e) Hessian points



(f) Hessian-Laplacian points

(g) Normalized Hessian patches

(h) Normalized Hessian-Laplacian patches

Figure 3.1: Display of keypoints from various detectors and their corresponding patch views. They are obtained with option: '-drdir dstdir' and '-dvdir dstdir' respectively.

Chapter 4

MISC

4.1 Important Notice

Features extracted by **LIP-VIREO** are not necessarily compatible with the features derived from tools by VGG [7], Prof. D. G. Lowe [2] or Dr. Dorko [9]. Therefore, the performance is not guaranteed when there is any mixture use of these detectors or descriptors.

4.2 Copyright

4.3 Acknowledgements

We would like to express our sincere thanks to Dr. Yanke from CMU, USA who provides us PCA-SIFT source code. His assistance has been invaluable. We are very grateful to Dr. Dorko from TUD, Germany who sends us many suggestions about how the dominant orientation of local patch is estimated. Gratefulness also reaches to Xiao-Jie Guo from University of Tianjin, China, whose informative suggestions and patience help the implementing of FIND.

4.4 Release Notes

4.4.1 V1.348 Release Note

1. The implementations of PCA-SIFT and MSER have been dropped from the project.
2. The PNG image format is supported in this new release
3. A fatal bug which causes memory leakage at descriptor calculation stage has been identified and removed.
4. The process of calculating descriptors have been sped-up, which causes minor loss in performance
5. The implementation for Hessian, Hessian-Laplacian, Difference-of-Gaussian, Laplacian of Gaussian and Dense detectors have been all optimized.

4.4.2 V1.26 Release Note

1. Fast Hessian detector has been integrated.
2. SURF descriptor has been integrated. (find more details in Section 2.6)
3. SURF-like descriptor 'AoD' (Aggregation on the first order Derivatives) has been integrated. (find more details in Section 2.6)
4. User is allowed to display keypoint as patches. (find more details in Section 3.3.2)
5. DoG detector has been optimized.
6. SIFT-like Flip invariant descriptor FIND has been integrated. (find more details in Section 2.3)

4.4.3 V1.067 Release Note

1. The performance of detector 'hess' has been optimized. It is able to achieve similar performance as 'hesslap' (hessian-laplacian).

4.4.4 V1.06 Release Note

1. The 'dense' detector has been integrated (check more details in Section 1.6).
2. 'ppm' formatted images are supported.
3. Flip invariant SIFT descriptor is integrated (check more details in Section 2.2).
4. In, keypoint feature display, user is allowed to either display 'cross' only or display 'cross' along with 'ellipses' (check more details in Section 3.3.1).

4.4.5 V1.055 Release Note

1. The MSER detector has been integrated.

2. A bug in loading JPEG file is identified and removed.

3. The '-d non' detector option is provided, which allows user to supply his own keypoints for extracting local interest descriptors. (see more details from Section 1.8)

4.4.6 V1.05 Release Note

1. Bugs in reading bmp file under Linux have been removed.

2. Bug causes irregular output of keypoints has been removed, which impacts the performances of hesslap and dog detectors in Version 1.042 a lot.

3. Function to plot keypoints onto the input image(s) has also been integrated. Please refer to Section 3.3.1 for more details.

4.4.7 V1.044 Release Note

1. We optimized the implementation of SIFT descriptor. The gradient-orientation bins with fewer energy are smoothed to zero. As a result, the SIFT feature becomes sparser. Based on our observation, such kind of modification results in better point-to-point matching results than previous versions. Note that, because of the modification, SIFT feature generated by this version is NOT COMPATIBLE to previous versions.

4.4.8 V1.042 Release Note

1. We optimized the parameters for descriptor calculation. As a result, features generated by this version may NOT be compatible with features generated by previous versions.

2. In this new version, user can obtain the affine region a keypoint by setting "affine=yes" in the configuration file. Please check details in Section 3.1.

3. We change the output format keypoint file (NOT the descriptor file). Please check details in Section 3.1.

4.4.9 V1.03 Release Note

1. SPIN descriptor is incorporated in this new version which is invariant to rotation and flip transformation. According to our experiment, it is also partially invariant to scaling. See more details in Section 2.5.

2. A novel descriptor ERIFT (Enhanced Rotation Invariant Feature Transform) which is invariant to scale, rotation and flip transformation has been incorporated. See details in Section 2.7.

3. Binary code which can be run under SunOS is also available for this new version.

4. Minor changes have been made on notations for descriptor options. For example, to choose un-normalized SIFT, user should choose 'SIFT' instead of 'UNSIFT'. Accordingly, normalized SIFT option has been changed to 'NSIFT'. Similarly, option for un-normalized LJet feature is changed to 'LJET', while for the normalized the LJet feature, the option is modified to 'NLJET' See details in Section 3.2.

4.4.10 V1.02 Release Note

1. In the descriptor file, user can choose to output scale and rotation information affiliated with each keypoint after proper configurations. Chech more details in Section 3.2.

2. The output format of descriptor file has been modified. For that reason, it becomes INCOMPATIBLE with previous versions. Check more details in Section 4.4.11.

4.4.11 V1.01 Release Note

1. **LIP-VIREO** accepts 'JPEG' formatted images as input.

2. Bug causes exception when inputting gray level 'jpeg' and 'bmp' has been identified and removed.

3. Bug causes exception when choosing 'ykpca' option has been identified and removed.

4. Performance of keypoint detector 'LoG' (`log`), 'Hessian-Laplacian' (`hesslap`) and 'Harris-Laplacian' (`harlap`) have been further improved. Comparing with previous versions, they produce less number of keypoints while achieving better performances.

Appendix

I. Format of descriptor file (Version 1.02 and later)

numkp	m	d		
p_1	p_2	...	p_m	
val_1	val_2	val_3	...	val_k
val_{k+1}	val_{k+2}	val_{k+3}	...	val_{2k}
...	...	...	...	...
val_{d-k+1}	val_{d-k+2}	val_{d-k+3}	...	val_d
p_1	p_2	...	p_m	
val_1	val_2	val_3	...	val_k
val_{k+1}	val_{k+2}	val_{k+3}	...	val_{2k}
...	...	...	...	...
val_{d-k+1}	val_{d-k+2}	val_{d-k+3}	...	val_d

◇ **numkp**: the number of features in the file

◇ **m**: the number affiliated properties, such as x, y, scale and dominant orientation. Especially, for local interest features. These properties have been put in a fixed order, i.e., x, y, scale and dominant orientation. If there is no affiliated properties, **m** is set to *0*.

◇ **d**: the dimension of the feature vector. The feature vector might be put in one line or several lines. Feature values that are in same line are separated by blank characters. The name of feature file is suffixed with fixed identifier: '**.pkeys**'.

II. Format of descriptor file (Version 1.01 & 1.00)

numkp	d	$nline$		
x_1	y_1			
val_1	val_2	val_3	...	val_k
val_{k+1}	val_{k+2}	val_{k+3}	...	val_{2k}
...	...	...	...	...
val_{d-k+1}	val_{d-k+2}	val_{d-k+3}	...	val_d
x_2	y_2			
val_1	val_2	val_3	...	val_k
val_{k+1}	val_{k+2}	val_{k+3}	...	val_{2k}
...	...	...	...	...
val_{d-k+1}	val_{d-k+2}	val_{d-k+3}	...	val_d

◇ **numkp**: the number of features in that file

◇ **d**: the dimension of the feature vector. The feature vector can be put in one line or several lines. Values presented appear in same line are separated by blank characters. The name of feature file is suffixed with '**.pkeys**' all the time.

◇ **nline**: it indicates how many lines that the descriptor feature takes up for one keypoint besides coordinate information (x, y).

III. Format of input keypoint file for 'non' detector option (Version 1.05 and later)

numkp *col*

x_1	y_1	a_1	b_2	c_3	*scale*	*angle*
x_1	y_1	a_1	b_2	c_3	*scale*	*angle*
...	...	...	...	...	...	...
x_n	y_n	a_n	b_n	c_n	*scale*	*angle*

◇ **numkp**: the number of features in this file.

◇ **col**: the number of columns in each line.

◇ **x and y**: X and Y location of a keypoint.

◇ **a, b and c**: parameters for ellipse region, if it is a circle, it is specified as $a = 1$, $b = 0$ and $c = 1$.

◇ **scale**: the characteristic scale of a keypoint, the radius of the local patch used for extracting descriptor.

◇ **angle**: the dominant orientation of the local patch.
Please be noted that the file name MUST BE suffixed with '.keys' all the time.

Bibliography

[1] M. K. and C. Schmid, "Scale and affine invariant interest point detectors," *International Journal of Computer Vision*, Vol. 60, pp. 63–86, 2004.

[2] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal on Computer Vision*, Vol. 60, no. 2, pp. 91–110, 2004.

[3] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 27, no. 10, pp. 1615–1630, 2005.

[4] R. Arandjelovic and A. Zisserman, "Three things everyone should know to improve object retrieval," in *CVPR*, Jun. 2012.

[5] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce, "A sparse texture representation using local affine regions", in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2005, Vol. 27, pp. 1265–1278.

[6] Linderberg, "Feature detection with automatic scale selection," *International Journal of Computer Vision*, Vol. 30, no. 2, pp. 79–116, 1998.

[7] A. Zissermanin and etc, "Visual Geomtry Group" in `http://www.robots.ox.ac.uk/~vgg/`.

[8] TREC Video Retrieval Evaluation (TRECVID), in `http://www-nlpir.nist.gov/projects/trecvid/`.

[9] Dorko, "Keypoint Detectors & Descriptors," in `http://lear.inrialpes.fr/people/dorko/downloads.html`.

[10] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, "SURF: Speeded up robust features," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[11] Christopher Evans: Notes on the OpenSURF Library, 2009.

[12] J. Delhumeau, P.-H. Gosselin, H. Jégou, and P. Pérez, "Revisiting the VLAD image representation," in *ACM Multimedia*, pp. 653–656, 2013.

[13] H. Jégou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search," in *ECCV*, pp. 304–317, Oct. 2008.

[14] Sivic, J. and Zisserman, A., "Video Google: A Text Retrieval Approach to Object Matching in Videos," *Proceedings of the International Conference on Computer Vision*, Vol. 2, pp. 1470–1477, oct, 2003.

[15] W.-L. Zhao, C.-W. Ngo, "Flip-Invariant SIFT for Copy and Object Detection", IEEE Trans. on Image Processing, 22(3), pp. 980–991, 2013.

[16] H. Jégou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search," in *European Conf. on Computer Vision*, 2008.

[17] W.-L. Zhao, X. Wu and C.-W. Ngo, "On the Annotation of Web Videos by Efficient Near-duplicate Search," *IEEE. Trans. on Multimedia*, vol. 12, no. 5, pp. 448–461, 2010.

[18] X. Guo and X. Cao, "FIND: A neat flip invariant descriptor," in *International Conf. on Pattern Recognition*, 2010, pp. 515–518.

[19] Test image from VGG, in `http://www.robots.ox.ac.uk/~{}vgg/data/data-aff.html`

[20] Patent of SIFT, in `http://www.google.com/patents?vid=6711293`