Indexing Sub-Vector Distance for High-Dimensional Feature Matching

Heng Yang, Qing Wang, Zhoucan He
School of Computer Science and Engineering
Northwestern Polytechnical University
Xi'an 710072, P. R. China
qwang@nwpu.edu.cn

Abstract

High-dimensional feature matching based on nearest neighbors search is a core part of many image-matching based problems in computer vision which are solved by local invariant features. In this paper, we propose a new indexing structure for the high-dimensional feature matching, which is based on the distance of the sub-vectors. In addition, we employ an effective image-similarity measure of two images based on the exponential distribution of the Euclidean distance between matched feature vectors. Experimental results have demonstrated the efficiency and effectiveness of the proposed methods in extensive image matching and image retrieval applications.

1 Introduction

Using local invariant features for image matching plays an important role in many computer vision applications, such as image retrieval, object recognition, panoramas building, scene reconstruction and video data mining [1-4]. In these applications, firstly local invariant features are detected individually in each image, then they are characterized by invariant descriptors and finally the feature descriptors in one image are matched to the features of other images. The matched features can be used in the subsequent procedures such as representing an object, voting for an image or being used for estimating the geometry parameters.

Related work. Lowe [5] proposed an object recognition method that are composed by three main components: (1) local feature detector based on the Difference-of-Gaussian filter; (2) the SIFT descriptor, which is highly distinctive and robust over common image deformations caused by changes in camera pose and lighting; (3) a fast matching algorithm, called BBF (best-bin-first) for the high-dimensional vector searching. These components have been widely used in many vision problems. Especially the SIFT descriptor performs so well that it still seems to be the most appealing descriptor for practical applications nowadays although various refinements based on this scheme have been proposed [6]. At the same time, a lot of data structures for the matching of the feature descriptors have been reported, which can be generally divided into five classes [7], (a) Exhaustive search; (b) Hashing and indexing; (c) Static space partitioning; (d) Dynamic space partitioning; (e) Randomized algorithms. For

BMVC 2008 doi:10.5244/C.22.70

detailed discussions on these algorithms please refer to [7]. In this paper, we only focus on the static space partitioning methods. K-d tree is a static space partitioning strategy based on a k-dimensional binary search tree and has been successfully used in ANN (approximate nearest neighbor) search for low dimensional cases. However, the k-d tree search often performs poorly in high-dimensional spaces. The BBF algorithm [5], also based on the binary searching tree, can deal with the high-dimensional searching issue and it has been widely used in matching image feature vectors. Each feature vector is considered as a point. BBF firstly selects a dimension as the key one, which has the largest variance, to divide each node if it is not a leaf node. Then for every point, it is added to the left child if the key dimension is smaller than that of the root and to the right child if bigger. The process is recursively executed on the left and right children until only one point remains. By this way, it will cost relatively less time when carrying out the ANN search process. Yu et al. [8] proposed the iDistance (indexing the distance) algorithm which is a B+ tree based indexing method used for ANN search in high dimensional data space. The core idea of iDistance is that it only searches the points which have the similar distance to the reference point as the query point. Our algorithm is also inspired by this idea. Sivic et al. [4] employed vocabulary tree to represent large number of the subsets of the feature vectors, and they consider the features associated with a particular subset as matches to each other. Nister et al. [9] designed a hierarchical vector quantization method based on the vocabulary tree and they matched individual feature descriptors by comparing paths of features down to the vocabulary tree. However, the memory occupancy needed to build a vocabulary tree is very high. In [9], about 143MB memory is used for building a vocabulary tree with 6 levels and 10 branches.

The first and the main goal of this paper is to propose an efficient and effective indexing mechanism for the feature vectors matching in high-dimensional space. Unlike kd-tree based methods, we employ sub-vectors (containing multi-dimensions, not only some one dimension) of the feature descriptor to build indexing structures for that multiple dimensions could contain more information than one. We consider feature descriptor matched if they contain particular numbers of the similar sub-vectors and the sub-vectors could be regarded as similar when their distances to a reference point are similar. Based on assumptions above, we could build a simple indexing using a string of binary bits to indicate the distance information of each sub-vector to a reference point. We will demonstrate our method more efficient and effective than the commonly used BBF algorithm by extensive experimental results. The second goal of this paper is to present a simple yet effective image-similarity measure which can be used in the image matching based applications.

The remainder of this paper is organized as follows. Section 2 details the proposed feature matching strategy and section 3 introduces the presented image-similarity measure. The experimental results and analyses are drawn in section 4. Finally, the conclusion and perspective are summarized in section 5.

2 High-Dimensional Feature Matching

To make searching for NNs (nearest neighbors) more efficient, we propose a new indexing structure based on the distances of sub-vector (seemed as a point) to the reference point (the origin point in our algorithm), called iSVD (indexing Sub-Vector

Distance). In our algorithm, each feature vector is divided into n parts equally and each part is termed as a sub-vector with equal dimensions.

2.1 Basic Idea of iSVD

There are two basic assumptions for iSVD.

Assumption 1: *If two feature vectors are similar (measured by Euclidean distance), then the correspondent sub-vectors of the two features should be similar respectively.*

This assumption can be formulated as following,

$$Sim(\mathbf{f},\tilde{\mathbf{f}}) = Sim(\mathbf{f}_1,\tilde{\mathbf{f}}_1) \wedge Sim(\mathbf{f}_2,\tilde{\mathbf{f}}_2) \wedge \ldots \wedge Sim(\mathbf{f}_m,\tilde{\mathbf{f}}_m) \quad (1)$$

where $Sim(\mathbf{f},\tilde{\mathbf{f}})$ denotes a binary variable which equals to 1 if the correspondent feature vectors $\mathbf{f}$ and $\tilde{\mathbf{f}}$ are similar and equals to 0 if not. $\mathbf{f}_i$ denotes the *ith* sub-vector of the feature descriptor and *m* is the number of sub-vectors used for indexing.

Assumption 2: *If two sub-vectors (seemed as points in high-dimensional space) are similar, then they have similar distance to the same reference point.*

This idea is also employed by [8], and it reveals the fact that the two points which are approximately located in a sphere centred at the reference point may be similar with each other. In our algorithm, we simply choose the origin point of high-dimensional space as the reference point. Therefore, the distance from a sub-vector point to the original point equals to the L2 norm of the sub-vector (termed as *norm_i* and *i* denotes the sequence number of the sub-vector in a feature descriptor, $1 \le i \le m$).

Suppose we build indexing structure using *m* sub-vectors of feature descriptors extracted in one image. For the illustration purpose, we resort to the binary split tree with *m* depth (see Figure 1 a), which in fact does not need in our algorithm. At each node of the tree in *ith* level, a divided value *div* is defined to divide the features in this node into 2 groups: "left" group if *norm_i* is smaller than *div* and "right" group if larger. The same process is recursively applied to each group until all the *m* sub-vectors are compared for every feature descriptor. We index each feature descriptor according to the path to denote the way it is down to the tree, i.e. assign '0' to denote it down to the left group and '1' denotes to right group (see Figure 1 a). Finally the feature descriptors can be indexed by a binary string or an equivalence decimal integer. The binary split tree discovers the essential idea of our indexing algorithm. In practical use, the tree structure is not required in our algorithm. We actually only need to calculate a binary string and the equivalent integer for indexing purpose (the red rectangle in Figure 1 b).

In order to improve the NN searching accuracy, we introduce a probability factor α ($0 \le \alpha \le 1$), which defines an ambiguity region. If there are *num* feature descriptors in the current group in *ith* level and they are sorted ascendingly according to *norm_i*, then we define S(*j*) function, denoting the *jth* ($1 \le j \le num$) L2 norm of sub-vector of all the features in this group. Thus, the ambiguity region can be defined as following,

$$amb\_reg = [v1, v2]$$
$$v1 = \mathrm{S}\left(\left\lceil \frac{1-\alpha}{2} * num \right\rceil\right) \quad , \quad \mathrm{v2} = \mathrm{S}\left(\left\lfloor \frac{1+\alpha}{2} * num \right\rfloor\right) \quad (2)$$

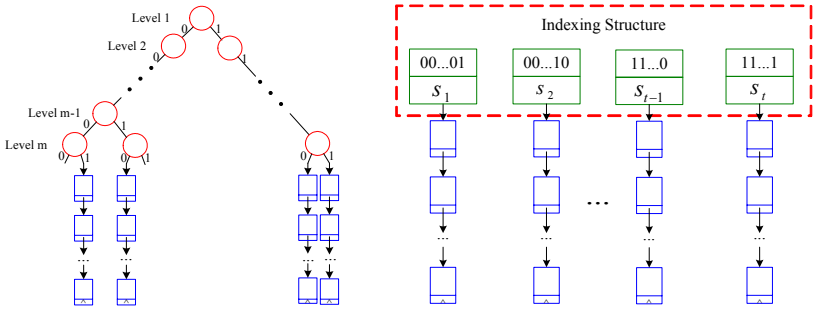
For each feature in this group, we define the following rules to build the indexing,

If *norm_i*<*v*1, add the feature to the "left" group in next level, i.e. assign '0' to the current bit of the binary string;

If *norm_i*>*v*2, add the feature to its "right" group in next level, i.e. assign '1' to the current bit of the binary string;

Otherwise, add the feature to both its "left" and "right" group, i.e. copy the feature indexing structure first, and then assigned '0' to the current bit of the binary string of original one and assigned '1' to that of the copied one. In other words, there will be producing two different indexes to indicate the same feature descriptor in this process.

The ambiguity region improves the hit probability for searching the accurate NN of the query feature and the parameter α will be discussed in detail in section 2.3.



(a) (b)

Figure 1: The proposed indexing structure iSVD for high-dimensional feature search. The red circles denote the tree nodes and the blue rectangles denote the feature descriptors. (a) the tree indexing structure; (b) the actually used indexing structure.

2.2 The Procedure of iSVD Algorithm

The proposed iSVD algorithm for high-dimensional feature match is concluded and organized as follows:

Step 1: Building. An index structure is built for all features extracted in one image.

Sub-step 1.1 Assign a binary string containing *m* bits and an unsigned decimal integer to each feature for indexing.

Sub-step 1.2 Divide each feature descriptor equally into *n* sub-vectors and use the first *m* ($1 \le m \le n$) sub-vectors for indexing.

In our experiment, the SIFT descriptor with 128 dimensions is divided into *n*=16 sub-vectors and each sub-vector contains 128/16=8 dimensions. Thus, *m* can be set from 1 to 16 and we choose *m*=8 in our subsequent experiments to get good balance between matching speed and accuracy.

Sub-step 1.3 For the current group of feature descriptors in *ith* level, the ambiguity region is first calculated (see Eq.2). Then for each feature, we employ the aforementioned rules in section 2.1. In addition, the median value of the *norm_i* of features in the current group is recorded in an array *array_divided* which will be used as the divide-values in the searching process.

The sub-step 1.3 is recursively applied to each group of feature descriptors level by level until the level is up to *m*, i.e. the first *m* sub-vectors of every feature descriptor are all compared in turn.

As a result, each feature has one or more binary array(s) and the equivalent decimal integer(s) for indexing. Then, we sort the indexing integers by ascending order. Thus, for every indexing integer, there are several feature descriptors indexed by it (see Figure 1 b). Furthermore, it is worth noticed that the number of feature descriptors indexed by an integer is nearly same due to our building scheme. This balance property will improve the searching speed.

Step 2: Searching. For a given feature in another image, firstly, we also assign a binary string containing *m* bits and an unsigned short integer to it for indexing. Then, we compare L2 norm of its first *m* sub-vectors in turn to the correspondent value in *array_divided* computed in sub-step 1.3. If it is smaller than the median value stored in *array_divided*, assign '0' to the *ith* position of the indexing binary string array; if it is larger or equal, assign '1'. This process is executed repeatedly until all the *m* sub-vectors have been compared. Then, we can calculate the equivalent integer *x* according to the indexing binary string. Next, we search the integer indexing structure *IS* built in Step 1, which is in ascending order. If $x_{i-1} < x = x_i < x_{i+1}$ ($x_i \in IS$), set $\tilde{x} = x$; If $x_{i-1} < x < x_i$, set $\tilde{x} = x_i$. Finally, we search all the features that indexed by $\tilde{x}$ one-by-one and find the nearest neighbor (1-NN) and the second one (2-NN) to the query feature among them.

Step 3: Matching. We only consider feature matches in which the distance ratio of 1-NN to the 2-NN is less than a ratio threshold T ($0 < T < 1$). This measure [5] performs well and effectively, since correct matches should have the closest neighbors significantly closer than the closest incorrect match.

2.3 Discussion on the Probability Factor α

The probability factor α is very important in iSVD algorithm and it is designed to improve the searching accuracy. Figure 2 shows the average searching accuracy and time cost along with different α values on the "Boat" image set [10], which contains 10 test images varying in different scales and rotations of the same scene. We match the feature sets between the first image to all the other 9 images respectively and calculate the average accuracy and time cost on the same value of α. The searching accuracy denotes the accuracy of the nearest neighbor.

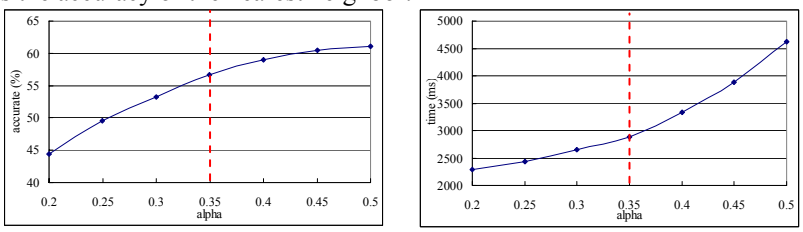


Figure 2: The curves of searching accuracy and time cost vs. α.

From Figure 2, we can see that both searching accuracy and time cost increase when α increases for that α defines an ambiguous region. If the L2 norm of the sub-vector of the query feature locates in this region, the feature indexing structure will be copied, which means the feature will be indexed by more than one integer. In other words, the number of feature indexed by one integer can be added. The larger the α is, the more number of features will be indexed by one integer, which will certainly

result in higher searching accuracy and time cost for searching. From Figure 2 (a), we can find that the improvement of searching accuracy becomes slowly when α is bigger than 0.35, while the time cost increases sharply when α exceeds 0.35. Therefore, we set α =0.35 in our subsequent experiments to get good balance between searching accuracy and time cost.

2.4 Memory Cost of iSVD

The iSVD algorithm only needs much smaller memory occupancy compared to vocabulary tree [9]. The memory cost contains two parts: one is for feature descriptors; the other is for the extra indexing structures. The total memory occupancy can be calculated approximately as following,

$$Mem = ND + N(1+\alpha)^m U = N[D + (1+\alpha)^m U] \tag{3}$$

where N is the feature number extracted in one image. For D-dimensional descriptor represented as char type it needs approximate DN bytes. For m sub-vectors building indexing needs approximate $N(1+\alpha)^m U$ bytes, where U denotes the bytes for each indexing structure containing a char array with m bytes and a short integer type with 2 bytes. In our case, D=128, m=8, U=m+2, α =0.35, N is averagely 8000, resulting in 1.8MB memory. When performing matching images, we can build indexing structures in one image using about 1.8MB memory and release the memory before performing another two image matching.

3 Similarity between Two Images

Matching of local invariant features enables image matching robust to background clutter and occlusion. When finish matching local feature descriptors between query image and the reference images, each match could translate to a vote for a particular reference image. The vote value can be considered as an image-similarity measure. For a successful voting scheme, a large value of votes should be assigned to the matching reference images, while only smaller value of votes can be assigned to the unrelated reference images. One natural way to measure the similarity of two images I_1 and I_2 is to use the matching number of the local features. However, in some situations, especially in the wide-baseline image matching, only matching feature number is not robust enough since the local feature descriptor is not as distinctive as that in the narrow-baseline case. Yao [3] proposed the image-similarity both using the number of matched features $N(I_1, I_2)$ and the mean distance of all matched features $\bar{d}(I_1, I_2)$,

$$Sim(I_1, I_2) = \beta \frac{N(I_1, I_2)}{N_{\max}} + (1-\beta)\frac{\bar{d}_{\max} - \bar{d}(I_1, I_2)}{\bar{d}_{\max}}, \quad \beta \in [0,1] \tag{4}$$

where $N_{\max}$ and $\bar{d}_{\max}$ are the highest $N(I_1, I_2)$ and largest $\bar{d}(I_1, I_2)$ among all image pairs, and β is a weighted parameter.

Here, we present a simple but effective image-similarity measure. Generally, the probability of two features being true match (i.e. correspond to the same local region R) is a monotonic function h(•) of the Euclidean distance between the two feature descriptors f^i and $\tilde{f}^i$,

$$P[(f^i, \tilde{f}^i) \in R] = \mathrm{h}(\| f^i - \tilde{f}^i \|_2) \quad (5)$$

And this probability can be simulated as an exponential distribution, i.e. $\mathrm{h}(x) = e^{-x}$. Therefore, the proposed image-similarity measure can be calculated as following,

$$Sim(I_1, I_2) = \sum_{i=1}^{K} e^{-\|f^i - \tilde{f}^i\|_2} \quad (6)$$

where K is the total number of matched features between two images I_1 and I_2.

The proposed image-similarity is related with the distance of the matching feature descriptors and the total number of matching features. It is simple yet effective, which will be verified in our subsequent experiments.

4 Experimental Results and Analysis

In this section, we present extensive experimental results to evaluate the performance of the proposed iSVD algorithm and image-similarity measure in image matching and image retrieval applications. In our experiments, we use SIFT algorithm [5] to create the invariant SIFT descriptors with 128 dimensions for each local regions. All the experiments are executed on a PC with Pentium IV 2.80 GHz CPU and 768M memory.

4.1 Searching Methods Comparison

In order to evaluate the proposed searching method iSVD completely, we choose 8 groups of image sets from the image database [10] which consists of a large number of various types of scenes. The image sets used in this experiment are listed in Table 1.

	Boat	Bricks	Cars	East_Park	Ensimag	Graffiti	Resid	Inria
Image Num	10	6	6	11	11	11	11	11
Total Feature Num	50050	62754	16554	43615	43311	42779	27863	30338

Table 1: The image number and feature number for each group of image sets.

		Boat	Bricks	Cars	East_Park	Ensimag	Graffiti	Resid	Inria
exhaustive search	Time	27512	77609	6543	13020	15866	8626	6095	8958
BBF	Accu 1	46.03	41.83	66.49	58.22	53.71	69.04	71.26	69.98
	Accu 2	19.90	18.04	36.16	33.04	30.82	51.55	49.74	46.21
	Time	2263	5159	878	1205	1399	755	804	1164
iSVD	Accu 1	57.60	61.49	71.75	71.85	61.88	76.32	79.29	77.03
	Accu 2	32.87	35.39	39.42	53.13	41.92	61.78	62.66	59.96
	Time	2024	2987	503	983	1087	610	612	927

Table 2: The average value of the accuracy (%) and time cost (ms) for 8 groups of image sets.

Each image set contains several related images of the same scene. About 300,000 SIFT features are extracted from these test images. For each image set, we take the first image as a query image to match all the other ones in this group respectively. In every two images matching procedure, we search the 1-NN and 2-NN of features of the first image in the indexing structure built in the second image. We record the average

searching accuracy of 1-NN, 2-NN and the average time cost for each image group. The results of exhaustive algorithm, well-known BBF algorithm and the proposed iSVD algorithm are listed in Table 2, respectively. We can see that the iSVD can obtain higher searching accuracy than BBF (the accuracy of 1-NN and 2-NN can averagely improve 19% and 43% than BBF respectively). Moreover, iSVD can be executed faster than BBF for feature matching (the speed of iSVD is 36% faster than BBF on average).

4.2 Feature Matching Performance Comparison

In this experiment, we evaluate the feature matching performance for the proposed iSVD algorithm using recall-(1-precision) graph [6], which captures the fact that we want to increase the number of correct positives while minimizing the number of false positives. We obtain the curves along with the variation of the matching ratio threshold T (see section 2.2 Step 3). Figure 3 presents the results of exhaustive searching, BBF and iSVD on images with different transformations, from which we can see that iSVD performs obviously better than BBF over various transformation types of images due to its higher accuracy in NN searching.

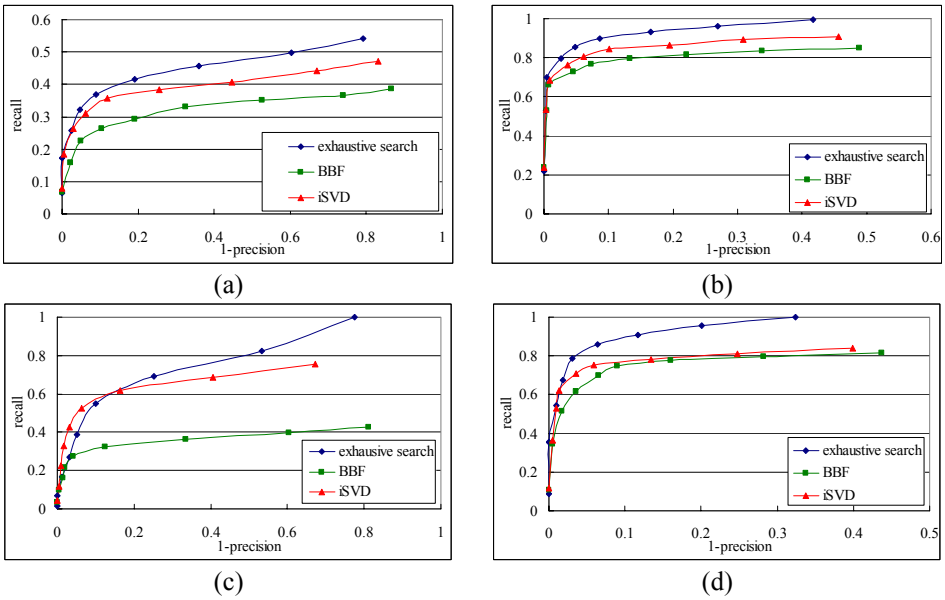


Figure 3: iSVD vs. BBF on matching tasks under different conditions. (a) rotation of 65 degree and scale of 4; (b) 12 degree viewpoint change; (c) corrupted by Gaussian noise; (d) 50% intensity scales.

4.3 Image-Similarity Validation

We evaluate the proposed image-similarity measure in an image retrieval experiment on a small dataset [11] which contains 30 images with 10 groups of different items. Our image retrieval experiment is similar to that conducted by Ke et al. [6]. They performed image matching between every two images and regard the number of matched features as a similarity measure between images. For each image, the top 2 images with most

matched number are returned. If the returned 2 images are both in the same group of the query image, the algorithm is awarded 2 points. If only one image is in the same group, it is awarded 1 point. Otherwise, it is given no point. Therefore, the full mark is 60. Here, we follow Ke's scoring rules, and use BBF searching method with three different image similarity measures which are matched number (termed as *sim_num*), weighed sum of matched number and distances [3] (termed as *sim_sum*) and the proposed method in section 3 (termed as *sim_exp*). The score results are listed in Table 3 and we can find that *sim_exp* is the most effective, whose score is much higher than the other two measures. In addition, *sim_sum* performs only a little better than *sim_num*, which is not as good as our expectation.

	sim_num	*sim_sum*	*sim_exp*
Scores	26	28	38

Table 3: Score results using three different image-similarity measures in an image retrieval test.

4.4 Image Retrieval Experiment

In this experiment, we evaluate the performance of the proposed feature matching method iSVD and image-similarity measure on a challenging image data set of recognition benchmark images provided by [12], which contains 10200 images in groups of four that belong together. We match every two images of the first 1000 images and perform scoring by counting how many of the four images in the same group (including the query image itself). The results of iSVD with *sim_exp* image-similarity measure compared to iSVD with *sim_num* and BBF with *sim_num* are presented in Figure 4. We can see that iSVD algorithm performs much better than BBF method with the same image-similarity measure (*sim_num*). Moreover, jointly using iSVD and *sim_exp* can obtain the best performance.

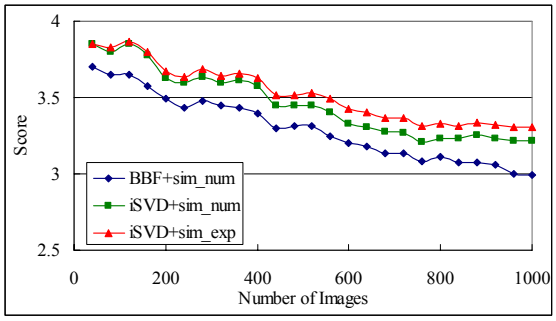


Figure 4: Performance comparison of image retrieval on the database [12].

5 Conclusion

The main contribution of this paper is to propose a novel indexing structure for high-dimensional feature matching, which builds indexing structure based on sub-vector distances. Furthermore, a simple yet effective image-similarity measure is also presented. We have demonstrated our approach efficient and effective on extensive

image matching and image retrieval experiments. In future work, we aim at applying our method in some other computer vision applications, such as object recognition and scene reconstruction from images.

Acknowledgments

This work is supported by National Hi-Tech Development Programs of China under grant No. 2007AA01Z314, National Natural Science Fund (60403008) and Program for New Century Excellent Talents in University (NCET-06-0882), P. R. China.

References

[1] Noah Snavely, Steven M. Seitz, Richard Szeliski, Photo tourism: Exploring photo collections in 3D. ACM Transactions on Graphics, 25(3):835-846, 2006.

[2] M. Brown and D.G. Lowe. Automatic Panoramic Image Stitching Using Invariant Features. IJCV, 74(1):59-73, 2007.

[3] J. Yao and W.K. Cham. Robust multi-view feature matching from multiple unordered views. Pattern Recognition, 40:3081-3099, 2007.

[4] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In ICCV, pages 1470-1477, 2003.

[5] D.G. Lowe. Distinctive image features from scale-invariant keypoints. IJCV, 60(2):91-110, 2004.

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

[7] Sameer A. Nene, Shree K. Nayar. A Simple Algorithm for Nearest Neighbor Search in High Dimensions. IEEE PAMI, (19) 9:989-1003, 1997.

[8] C. Yu, B. C. Ooi, K. L. Tan, H.V. Jgadish. Indexing the Distance: An Efficient Method to KNN Processing. Proceedings of the 27th VLDB Conference, pages 421-430, 2001.

[9] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In CVPR, pages 2161-2168, 2006.

[10] Test images for image matching. http://lear.inrialpes.fr/people/mikolajczyk/

[11] A small image set. http://www.cs.cmu.edu/~yke/pcasift/

[12] Object recognition benchmark. http://vis.uky.edu/˜stewe/ukbench/