



# Fast feature matching based on $r$ -nearest $k$ -means searching

ISSN 2468-2322

Received on 26th September 2018

Revised on 17th October 2018

Accepted on 21st October 2018

doi: 10.1049/trit.2018.1041

www.ietdl.org

Ke Wang<sup>1</sup> ✉, Ningyu Zhu<sup>1</sup>, Yao Cheng<sup>2</sup>, Ruifeng Li<sup>1</sup>, Tianxiang Zhou<sup>1</sup>, Xuexiong Long<sup>1</sup>

<sup>1</sup>State Key Laboratory of Robotics and System, School of Mechatronics, Harbin Institute of Technology, Harbin 150001, People's Republic of China

<sup>2</sup>China Mobile (Hangzhou) Information Technology Co., Ltd, Hangzhou 311100, People's Republic of China

✉ E-mail: wangke@hit.edu.cn

**Abstract:** Feature matching has been frequently applied in computer vision and pattern recognition. In this paper, the authors propose a fast feature matching algorithm for vector-based feature. Their algorithm searches  $r$ -nearest neighborhood clusters for the query point after a  $k$ -means clustering, which shows higher efficiency in three aspects. First, it does not reformat the data into a complex tree, so it shortens the construction time twice. Second, their algorithm adopts the  $r$ -nearest searching strategy to increase the probability to contain the exact nearest neighbor (NN) and take this NN as the global one, which can accelerate the searching speed by 170 times. Third, they set up a matching rule with a variant distance threshold to eliminate wrong matches. Their algorithm has been tested on large SIFT databases with different scales and compared with two widely applied algorithms, priority search km-tree and random kd-tree. The results show that their algorithm outperforms both algorithms in terms of speed up over linear search, and consumes less time than km-tree. Finally, they carry out the CFI test based on ISKLRS database using their algorithm. The test results show that their algorithm can greatly improve the recognition speed without affecting the recognition rate.

## 1 Introduction

Feature matching has played a critical role in the field of computer vision such as image indexing, local feature-based object recognition and video retrieval [1]. It is a process of  $k$ -nearest neighbour (NN) searching, which returns  $k$  exact NNs of a query point [2]. In most cases, the database contains millions of points, thus the NN searching becomes the most computationally expensive part in the feature matching process. Linear search is the most accurate way to deal with this issue [3]; however, it queries all points in the database, and its searching time soars linearly with the increasing database scale. To speed up the feature matching process, Approximate NN (ANN) [4] has been widely used. ANN speeds up the matching process by substituting the exact NN with an approximate one, which has a high probability to be the point we are searching for.

To improve the matching speed and precision, we have developed a novel matching algorithm. It is based on  $r$ -nearest  $k$ -means clusters to solve the problem of vector-based feature matching in large-scale database such as scale-invariant feature transform (SIFT) [5] and speeded-up robust features (SURF) [6]. The key concept is to locate a small region of the database that has a high probability to contain the NN and then takes the local NN in this region as the global one. After the database is partitioned into  $k$  clusters by  $k$ -means clustering, the  $r$ -nearest clusters to the query point are selected as local region. Experimental results show that our algorithm outperforms two widely applied algorithms: priority search km-tree and kd-tree. It achieves near two orders of magnitude of speed ups over linear search with a precision of 95%. The organisations of our work can be summarised in three aspects. First, it has a faster structure construction process. Unlike priority search km-tree, which needs to allocate a complex tree structure and conducts recursively  $k$ -means clustering in every cluster, our algorithm establishes a general  $k$ -means manipulation instead of a tree structure. Therefore, our structure is greatly simplified with less construction time than km-tree. Second, it is composed of a faster query procedure. Different from km-tree which searches for the NN in a queue of tree nodes, our  $r$ -nearest

searching strategy only locates and searches the  $r$ -nearest clusters to the query point in the database, rather than the whole database. The remarkable reduction of the number of searching points will shorten search time and achieve impressive speed up over linear search. Third, our algorithm adopts a matching rule of higher precision, which is different from the widely applied feature matching rule. We set up a variant distance threshold to adapt to the  $r$ -nearest search strategy, so as to eliminate wrong matches caused by the local NNs.

The organisation of this paper is as follows. In Section 2, we present the technical details of our approach. Section 3 shows the experimental results obtained by our algorithm. Section 4 shows the test results of clustering of features and images (CFI) optimisation based on images of State Key Laboratory of Robotics and System (ISKLRS) database.

Feature matching is a typical application of the ANN method [7]. The only linear search returns the exact NN of the query point but it generally costs too much time. To speed up the NN searching process, many ANN search algorithms have been proposed. According to the way of constructing the structure of the database, most ANN methods can be classified as three categories: partitioning trees, hashing techniques and neighbouring graph techniques [8]. Hashing techniques are always used for binary features such as oriented FAST and rotated BRIEF (ORB) [9] and binary robust independent elementary features (BRIEF) [10]. Partitioning tree is more efficient for vector-based feature with high dimension such as SIFT and SURF. NN graph methods usually build a graph structure, where points are connected to its NNs by edges, and it is really time-consuming.

A famous tree-based ANN method is kd-tree, which is widely used with good performance for low-dimensional search [11]. It recursively partitions the space through hyperplanes and forms a tree structure. When searching for the NN, the query point is compared with the splitting value of each tree node to find the NN. Locality sensitive hashing (LSH) adopts several hash functions of the same type to create the corresponding hash value for each point of dataset [12]. Each function reduces data dimension by projecting them to random vectors and recursive hashing step. The query point is mapped by

the hash function and the projected points in the same bin are returned as nearest candidates. Hajebi built a  $k$ -NN graph in an offline manner and then used a hill-climbing step to find the nearest point from a random graph node [13]. It has been empirically proved that this graph-based method outperforms LSH and kd-tree. However, building a  $k$ -NN graph for each point in the database may cost much more time than other algorithms.

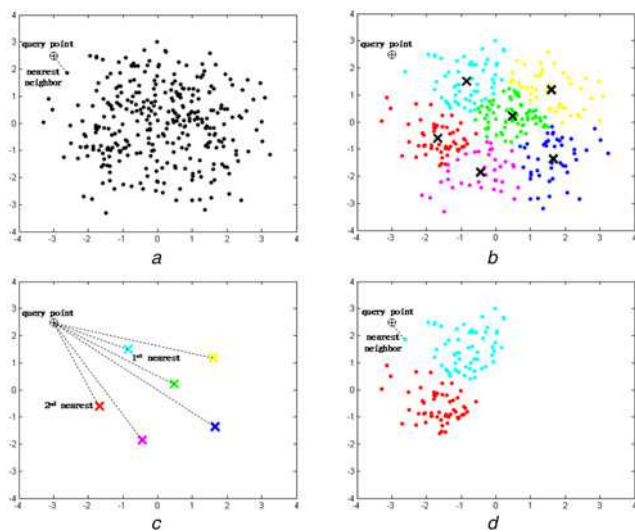
Muja did a lot of work in ANN search [14], and our algorithm is mainly compared with priority search km-tree. Marius discussed two algorithms: randomised kd-tree and priority search km-tree. He empirically illustrated that these two algorithms outperformed other methods for vector-based feature. The priority search km-tree is constructed by partitioning data into  $k$  distinct regions using  $k$ -means clustering at each level, and then adopts a recursive method in each region. These two algorithms achieve speed up at about two orders of magnitudes, one is with precision 0.8 over linear search and the other one is with precision 0.98. Marius built a fast library for ANN search (FLANN) [15] including LSH, random kd-tree, priority search km-tree and hierarchical clustering tree. In this paper, it is adopted to test km-tree and kd-tree to make a comparison with our algorithm.

The main idea for our proposed method is that our algorithm will partition the database into  $k$  clusters first and ranks the centres by their distance to the query point. The NN will be more possible to be found in the region near the query point compared with the region far from the query point. If the whole database to be searched can be scaled to a small region, it will remarkably accelerate the searching process. When it refers to the feature matching among large-scale database, fast feature matching is always adopted for its remarkable matching speed. Compared to km-tree and kd-tree, our fast feature matching algorithm based on  $r$ -nearest  $k$ -means clusters has a simpler structure with less construction time, a flexible matching rule, and higher searching speed and precision. Therefore, our fast feature matching algorithm is a better choice for feature matching with the large scale of vector-based feature such as SIFT and SURF.

## 2 $r$ -Nearest $k$ -means clusters

Fig. 1 shows how to search the  $r$ -nearest clusters for the NN of a query point in two-dimensional feature space after  $k$ -means clustering.

If the linear search is applied, it is necessary to visit all points in the whole feature space, which is computationally too expensive. In our algorithm, the database is clustered by  $k$ -means clustering first as



**Fig. 1** Illustration for ANN search algorithm based on  $r$ -nearest  $k$ -means clusters proposed in this work

a global linear search  
b  $k$  clusters ( $k=6$ )  
c  $r$ -nearest search ( $r=2$ )  
d local linear search

shown in Fig. 1b, in which case it is supposed that  $k=6$ . Points in the database will be clustered around  $k$  centres with  $k$  clusters, and there are clusters near the query point while others are not. When searching for the NN, the query point will be first matched to  $k$  centres by linear search as shown in Fig. 1c, and this would not cost much time since  $k \ll n$ . To reduce the searching time, our algorithm sets searching factor  $r$  to represent the number of selected clusters. Only  $r$  clusters near to the query point will be selected as the searching region. We use Euclidean distance to measure the distance for vector-based feature. This search strategy is called  $r$ -nearest search strategy. In Fig. 1d, two nearest clusters will be searched for the ANN. When the distance between the centre of a cluster and the query point increases, or the searching factor  $r$  becomes bigger, the probability of finding the NN in these  $r$  clusters will become higher.

### 2.1 Construction of $k$ -means clusters

When constructing data structure, we apply  $k$ -means clustering as Algorithm 1 (see Fig. 2) depicted. We add some variables to this  $k$ -means process.  $n\_C\_index$  is used to index the local number of a feature by its global number, while  $index\_C\_n$  is used to index it back. Here,  $k$  is the branching factor which determines the number of clusters a database that will be partitioned into.  $I_{max}$  denotes the number of iterations conducted in  $k$ -means process.  $C_{alg}$  represents the way to initialise the seed centre position. After clustering, the database will be divided into  $k$  clusters. Different from priority search km-tree method, which will recursively conduct clustering in each part again and again until the number of the points is less than the given threshold, our algorithm does not apply such recursive construction step but clusters only once.

### 2.2 $r$ -Nearest search and feature matching

The fast feature matching process based on  $r$ -nearest  $k$ -means clusters is shown in Algorithm 2 (see Fig. 3). This algorithm

---

**Input:** feature dataset  $D$ , branching factor  $k$ , maximum iterations  $I_{max}$ , center initialization algorithm  $C_{alg}$   
**Output:**  $k$ -means clusters, with  $C$  recording the centers,  $C_d$  recording the points in each cluster,  $n\_C\_index$  recording the center index and index in clusters of points in  $D$ , and  $index\_C\_n$  recording the index in  $D$  of points in each clusters.

```

1: if  $|D| < k$  then
2:   create a cluster with the descriptors in  $D$ 
3:    $C \leftarrow$  means of descriptors in  $D$ 
4: else
5:    $C \leftarrow$  select  $k$  points from  $D$  using the  $C_{alg}$  algorithm
6:   converged  $\leftarrow$  false
7:   iterations  $\leftarrow 0$ 
8:   while not converged and iterations  $< I_{max}$  do
9:      $C_d \leftarrow$  cluster the points in  $D$  around nearest centers  $P$ 
10:     $C_{new} \leftarrow$  means of clusters in  $C_d$ 
11:    if  $C = C_{new}$  then
12:      converged  $\leftarrow$  true
13:    end if
14:     $C \leftarrow C_{new}$ 
15:    iterations  $\leftarrow$  iterations + 1
16:  endwhile
17:  $n\_C\_index \leftarrow$  center index and index in clusters of descriptors in  $D$ 
18:  $index\_C\_n \leftarrow$  index in  $D$  of descriptors in each clusters

```

---

**Fig. 2** Algorithm 1: Clustering dataset by  $k$ -means

---

**Input:**  $k$ -means clusters, searching factor  $r$ , query descriptor  $Q$   
**Output:** *MatchInfo*, the information of the matched descriptor in database  $D$

- 1: *match*  $Q$  to clustering centers  $C$
- 2:  $P \leftarrow$  cluster index of  $r$  nearest centers to  $Q$
- 3:  $C \leftarrow$  means of descriptors in  $D$
- 4:  $d_{\min} \leftarrow \infty$
- 5:  $d_{\text{nextMin}} \leftarrow \infty$
- 7:  $i \leftarrow 0$
- 6: **while**  $i \leq r$  **do**
- 7:  $d_{\min, \text{new}} \leftarrow$  the nearest distance of  $Q$  to descriptors in the  $i$ -th cluster
- 8:  $d_{\text{nextMin}, \text{new}} \leftarrow$  the next nearest distance of  $Q$  to descriptors in the  $i$ -th cluster
- 9:  $d_{\min} \leftarrow \min\{d_{\min}, d_{\text{nextMin}}, d_{\min, \text{new}}, d_{\text{nextMin}, \text{new}}\}$
- 10:  $d_{\text{nextMin}} \leftarrow \text{nextMin}\{d_{\min}, d_{\text{nextMin}}, d_{\min, \text{new}}, d_{\text{nextMin}, \text{new}}\}$
- 11: *MatchInfo*  $\leftarrow$  the information of the descriptor with the nearest distance
- 12: **endwhile**
- 13: **if**  $d_{\min}$  and  $d_{\text{nextMin}}$  satisfy the matching condition
- 14: **return** *MatchInfo*
- 15: **else**
- 16: **return** false
- 17: **endif**

---

**Fig. 3** Algorithm 2:  $r$ -nearest search and feature matching

approximately takes local NNs in  $r$ -nearest clusters as the global NNs in  $k$  clusters. It remarkably reduces the number of points to be searched and achieves speed up over linear search. ANN search process is conducted following  $r$ -nearest searching strategy. Searching factor  $r$  represents the number of nearest clusters selected as the candidate region. As illustrated in Figs. 1c and d, two nearest clusters are selected as the region to search. Then, the linear search method is used to search NNs in this region. Finally, the NNs to the query point will be approximately returned. When  $r=0$ , no points will be searched, the algorithm returns *false*. When  $r=k$ , all points in the database will be searched and our algorithm degenerated into a linear search method with an exact NN.

### 2.3 Feature matching with a variant threshold

Traditional linear search matching generally uses rules recommended by Lowe [16]; it adopts fixed thresholds for nearest distance  $d_{\text{st}}$ , the ratio  $d_{\text{st}}/d_{\text{rd}}$  ( $d_{\text{st}}$  represent the nearest distance, and  $d_{\text{rd}}$  is the second nearest distance), or a combination of these two values. Here,  $d_{\text{st}}/d_{\text{rd}}$  is easier to satisfy Lowe's rule for that  $d_{\text{rd}}$  is definitely equal or greater than the exact value, whereas  $d_{\text{st}}$  is more possible to be the exact one. It leads to the fact that when the traditional matching rule for linear search is applied in ANN search methods, it will generate more incorrect matches than linear search. To filter those incorrect matches, our method defines a novel matching rule to adapt to our  $r$ -nearest search strategy.

Our matching rule combines Lowe's rule with a variant distance threshold changing with the index of the cluster, which contains the NN in the chosen  $r$ -nearest clusters. The new rule is defined as

$$\begin{cases} d_{\text{st}} < D_{\min} \times \left(1 - \frac{i}{\alpha \times r}\right) \\ \frac{d_{\text{st}}}{d_{\text{rd}}} < \text{ratio} \end{cases} \quad (1)$$

where  $i \in \{1, 2, \dots, r\}$  means the  $i$ th nearest cluster to the query point. Initial nearest distance threshold  $D_{\min}$  is set to be near the largest value for matched features in linear search. The new matching rule reveals that the smaller  $i$  is, the bigger the threshold will be. If the

ANN is detected in a cluster relatively far away from the query point, it means that  $i$  is comparatively big, a smaller-nearest distance threshold is then used to eliminate wrong matches. Here,  $\alpha$  is a coefficient which decides the importance of the variant threshold that is attached to  $i$  and  $r$  is the searching factor. The existence of  $D_{\min} \times [1 - i/(\alpha \times r)]$  is empirically proved to be very helpful to improve precision.

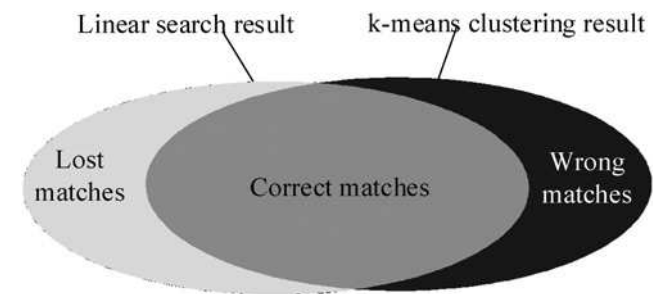
### 2.4 Computational complexity

The computational complexity of our algorithm involves structure construction complexity and searching time complexity. Our method conducts  $k$ -means clustering for original data only once, so construction time complexity is  $O(ndkI)$ . Here,  $n$  denotes the number of points in the database,  $d$  is the feature dimension,  $k$  represents the branching factor or the number of clusters and  $I$  means the number of clustering iterations. The construction complexity of km-tree is  $O(ndkI(\log n/\log k))$  [17] since it recursively performs  $k$ -means clustering in the clusters until the number of points in each cluster is below the threshold. Our  $r$ -nearest  $k$ -means clustering process is faster than that of priority search km-tree by  $(\log n/\log k)$  when constructing data structure, since  $\log n \gg \log k$ . With the assumption that the whole database is partitioned evenly, the proposed method will search for  $k + nr/k$  points while linear search method queries all points. The searching time complexity of our method will be  $O[(k + nr/k)d]$ , while it is  $O[Ld(\log n/\log k)]$  for km-tree and  $O(nd)$  for linear search.  $L$  denotes the number of points that km-tree will query until it stops, and it is similar to  $k + nr/k$ . This indicates that our method in this work requires less construction and searching time than priority search km-tree by  $(\log n/\log k)$ . Compared to linear search, our algorithm costs less searching time by  $1/(k/n + r/k)$ , where  $n \gg k$  and  $k \gg r$ .

## 3 Experimental results

Our algorithm is aimed at improving the speed of a high-dimensional vector-based feature matching process. We mainly compare the algorithm with km-tree and kd-tree, both of which have been proved to be one of the best ANN algorithms for vector-based feature. We conduct a series of experiments to test the proposed algorithm to illustrate that it performs better than these two widely used algorithms.

In our experiments, we take precision and search speed into consideration. As shown in Fig. 4, precision in this work is defined as  $p = (\text{correct} - \text{wrong})/(\text{lost} + \text{correct})$ , where  $\text{lost} + \text{correct}$  is the number of matches detected by linear search,  $\text{correct}$  means the matches detected by our algorithm that are the same as that of linear search. Here,  $\text{lost}$  is the number of matches detected by linear search but lost by our algorithm, while  $\text{wrong}$  denotes the number of matches not detected by linear search but detected by our algorithm. When a pair of matches is detected by both linear search and our algorithm, it will be counted as a correct match;  $\text{correct}$  is the true positive instance, while  $\text{wrong}$  is the false positive instance which indicates the disadvantage of the system.



**Fig. 4** Relationship between matching results of the linear search and  $r$ -nearest  $k$ -means clusters search

In the following experiments, the basic database is formed by SIFT features extracted from image database indoor environment under changing conditions (INDECS), which contains 3252 images [18]. It is evenly divided into train set and query set, and images are scaled to  $64 \times 48$ ,  $128 \times 96$ ,  $256 \times 192$  and  $512 \times 384$  to form databases with different sizes, as shown in Table 1. In the following paragraph, set\_1, set\_2, set\_3 and set\_4 will be used to represent these four sets. Experiments are conducted on a ThinkPad laptop with a central processing unit of Core(TM) i5 M 520 with 8 GB random access memory. The kd-tree and km-tree are provided by FLANN. The random method is used to initialise the cluster centres for  $k$ -means clustering.

### 3.1 Iterations

Iteration is an important factor for  $k$ -means clustering which influences the precision of  $k$ -means clustering and it is denoted by  $I_{\max}$  in this work. When iterating enough times, it converges to fixed cluster centres. In many cases,  $k$ -means clustering will cost too much time because its computational time is linear over  $I_{\max}$ . In this work,  $k$ -means clustering can be set to converge or iterate for required times, and it may affect final precision. This part of the experiment is conducted on set\_3 with increasing  $I_{\max}$  to test the influence of  $I_{\max}$  on precision and data structure construction time. The result is shown in Fig. 5. It implies that when iteration times reach 11, the precision is almost the same as the final convergence; it costs only about 10% of construction time compared with clustering. Therefore, in the following experiments, we will let  $I_{\max} = 11$ .

### 3.2 Branching factor $k$ and searching factor $r$

There are two parameters, branching factor  $k$  and searching factor  $r$ . They directly affect construction time and search performance of our algorithm. Its computational complexity is  $O(ndkl)$ , which means that its construction time is linear to  $k$ ,  $k$  is related to precision.

Searching factor  $r$  represents the number of selected nearest clusters to the query point and determines how many clusters will be searched. It is obvious that searching time is linear to  $r$  with the assumption that the database is partitioned evenly. In this part of the experiment, we use parameter grid to optimise  $k$  and  $r$  to get optimum values. All of the four SIFT sets are clustered by  $k$ -means with different branching factor  $k$ . Feature matching is conducted with increasing the searching factor  $r$ .

Results are presented in Fig. 6, each subfigure is for a different database. No matter what database size is, as searching factor  $r$  increases, matching precision will grow with decreasing the speed

ups. However, this growth will become smoother since most of the NNs exist in the first few clusters. For  $r$ -nearest  $k$ -means clusters in this work, as query point gets closer to the selected cluster centre, the probability of the NN being in the cluster will become higher.

When we add clusters which are relatively far away from the query point into the searched region, it would not increase the precision but enlarges the searched region and costs more searching time. That is the main reason why we only search the  $r$ -nearest clusters. When  $r$  is close to  $k$ , its precision approaches 1 but its speed up will be lower than 1. This can be explained by its search computational complexity  $O[(k + nr/k)d]$ , which is better than linear search by  $1/(k/n + r/k)$ . Given  $k \ll n$ , if  $r = k$ , then  $1/(k/n + r/k) < 1$ , the searching process of our algorithm will be slower than linear search.

Concentrating on speed ups of required precision, we can find the corresponding best parameters. In this work, the precisions were set to be 0.8, 0.9, 0.95 and 0.98, and their best speed ups are shown in Fig. 7. It implies that if the size of SIFTs data is about 280 k, our algorithm achieves speed up by 170 times with precision 0.8 and speed up by 78 times with a precision 0.95. It is more than twice as much as that of km-tree. The optimal value of  $r$  for required precision will be the critical value to satisfy the requirement. Fig. 6 also implies that with the increase of  $k$ , the database will be partitioned into more clusters, and its speed up will increase to the extremum and then drop, indicating that there is an optimal  $k$  for different databases to obtain the best speed up. Since when  $k$  increases, the search procedure has to query more cluster centres in the linear search process. It can be explained by the searching complexity  $O[(k + nr/k)d]$ . Here,  $(k + nr/k)$  is the number of points to be searched, and it will first decrease with small  $k$ , for that  $nr/k$  is the main part and then increase with  $k$  since  $k$  becomes the main part, while the speed up can be affected oppositely. The theoretical equilibrium point for  $k$  will be  $k = \sqrt{nr}$ , where  $(k + nr/k)$  reaches the extremum. The best parameters based on these databases for our algorithm are summarised in Table 2.

### 3.3 Comparison to km-tree and kd-tree

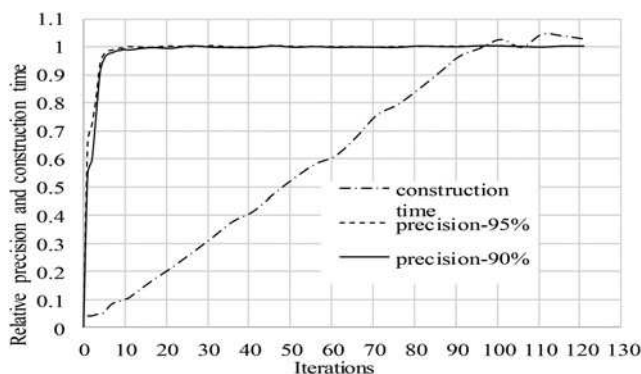
Marius has tested and compared km-tree and kd-tree with some other algorithms and proved that km-tree and kd-tree are better for high-dimensional vector feature. He explained that priority search km-tree outperforms kd-tree when it refers to data with high dimension or it requires high precision but kd-tree costs less construction time. In this part, km-tree and kd-tree will be compared with our algorithm. The ways to get precision and matching rule for km-tree and kd-tree in this part of the experiment are the same as the former part, and iteration times of km-tree are also 11.

We did similar experiments as Section 3.2 to test km-tree and kd-tree. Parameter grid is used to search the best parameters for km-tree and kd-tree. Clustering factor and index factor [19, 20] are the main parameters of the two algorithms, which are similar to  $k$  and  $r$  of our algorithm. The experimental results of km-tree are shown in Fig. 8. Compared to Fig. 6, we can see that the speed of km-tree is not very high but it decreases slowly and the precision rises slowly. Besides, we can get Fig. 9 from Fig. 8. The shapes of curves in Fig. 9 are different from that of Fig. 7, and the speed up of km-tree is only about half that of our algorithm. Fig. 8 shows that the precision of the algorithm will decrease if the indexing factor becomes smaller. The precision of set\_4 is below 0, it is because that the number of wrong matches is more than that of correct matches. In fact, when the indexing factor is small, the number of correct matches of km-tree is equivalent to the algorithm proposed in this paper but too many wrong matches are generated, making the precision below 0. It shows that our algorithm is a better choice for vector-based feature matching than km-tree.

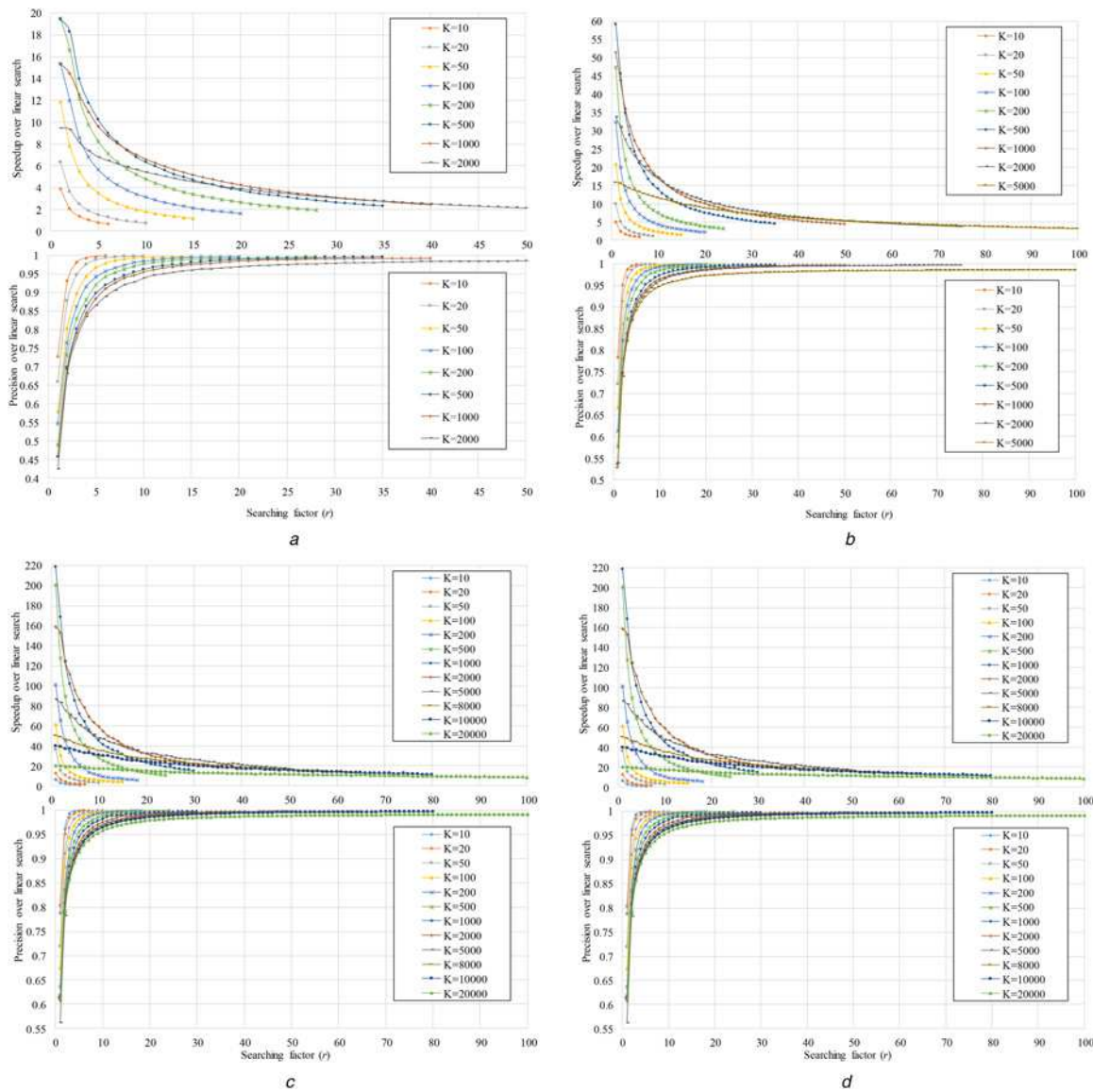
The experimental results of kd-tree are shown in Fig. 10. We can see that with the increase of the indexing factor, the speed up of kd-tree decreases and the precision increases. Similar to km-tree, the precision of kd-tree is below zero when the indexing factor is

**Table 1** Testing database sizes

| N_SIFTs   | set_1  | set_2  | set_3   | set_4   |
|-----------|--------|--------|---------|---------|
| train set | 20,414 | 53,727 | 127,371 | 278,043 |
| query set | 23,557 | 62,387 | 181,037 | 720,321 |

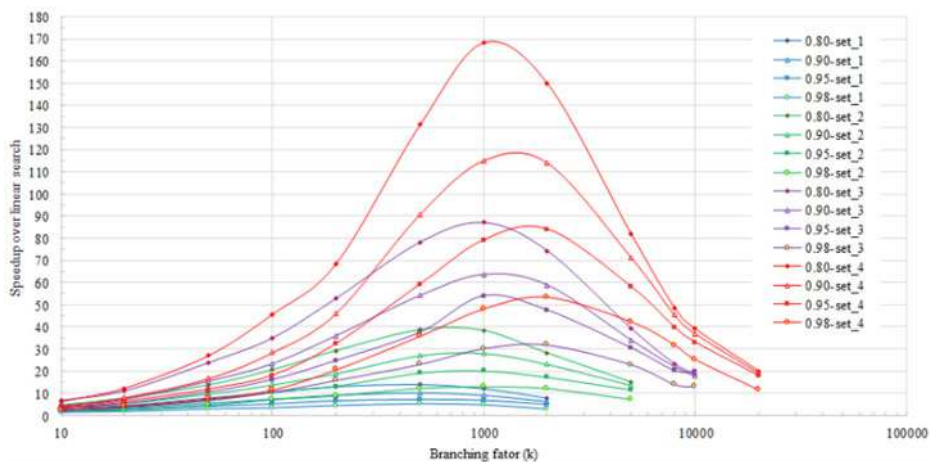


**Fig. 5** Relative search precision and construction time over iteration times



**Fig. 6** Precision and speed up of  $r$ -nearest  $k$ -means clusters with different branching factor  $k$  and searching factor  $r$

- a Performance of  $r$ -nearest  $k$ -means clusters with set\_1
- b Performance of  $r$ -nearest  $k$ -means clusters with set\_2
- c Performance of  $r$ -nearest  $k$ -means clusters with set\_3
- d Performance of  $r$ -nearest  $k$ -means clusters with set\_4



**Fig. 7** Speed ups for certain precisions of different databases

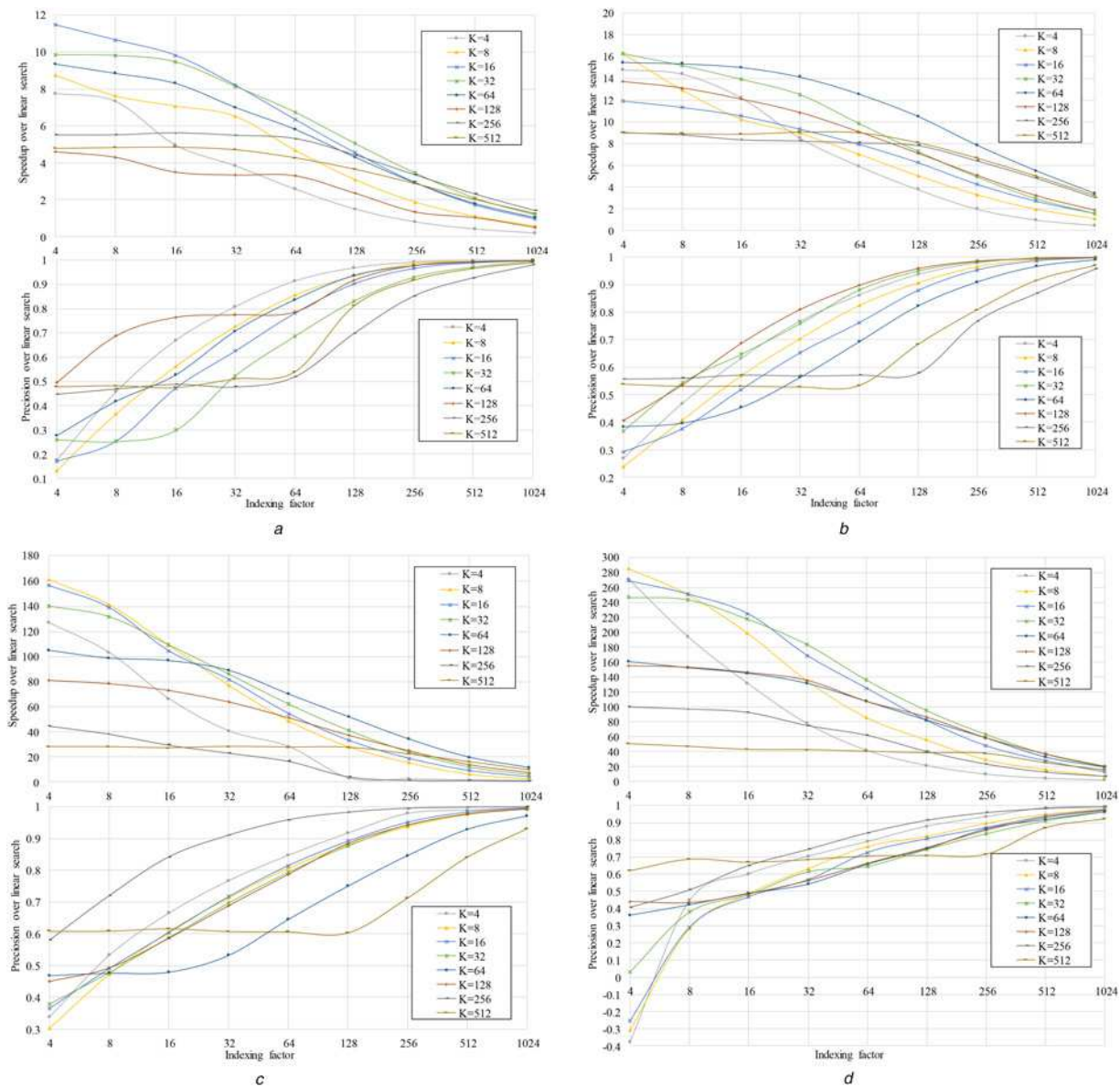
**Table 2** Optimal parameters for  $r$ -nearest  $k$ -means clusters

| $(k, r)$ | set_1    | set_2     | set_3     | set_4     |
|----------|----------|-----------|-----------|-----------|
| 0.8      | (480,3)  | (700,3)   | (900,2)   | (1100,2)  |
| 0.9      | (500,5)  | (780,5)   | (1100,4)  | (1500,4)  |
| 0.95     | (550,8)  | (840,8)   | (1200,6)  | (1800,6)  |
| 0.98     | (600,15) | (1000,15) | (1500,11) | (2000,11) |

small. We can also get Fig. 11 from Fig. 10. The results show that the speed up of kd-tree is lower than that of km-tree for different precisions, which means that our algorithm in this paper is superior to kd-tree.

These three algorithms are tested on empirically estimated best parameters. Fig. 12 presents the experimental results, which records their speed ups and construction time. All of the figures in Fig. 12 imply that when required precisions increase, their speed up process becomes slower, no matter what algorithm is adopted. We can also see that speed up of  $r$ -nearest  $k$ -means clusters searching is always much higher than that of km-tree and kd-tree,

which indicates that our algorithm outperforms km-tree and kd-tree. Besides, it validated the conclusion that km-tree performs better than kd-tree when it requires high precision. The km-tree is more suitable for the higher-dimensional feature than kd-tree. While construction time issue is ignored, it is about half of the speed ups of our algorithm. It shows that the larger the database scale is, the higher speed up our algorithm may achieve, which can be explained by its computational complexity that  $1/(k/n + r/k)$  will become larger as  $n$  increases. The construction time of each algorithm is recorded in the below figure of each subfigure in Fig. 12. It indicates that kd-tree costs the least construction time, and the construction time of our algorithm is less than km-tree. It has been theoretically discussed in Section 3.2 that construction computational complexity of the proposed algorithm is lower than that of km-tree, which indicates that it can be a better choice than km-tree for the invariant database, and the kd-tree is more suitable for variant feature database. Therefore, if database changes and requires frequent construction process, kd-tree will be a better choice. However, when the database keeps the same for a long time, our algorithm may be a better choice than km-tree.

**Fig. 8** Precision and speed up of km-tree clusters with different branching factor  $k$  and searching factor  $r$ 

- a Performance of km-tree clusters with set\_1
- b Performance of km-tree clusters with set\_2
- c Performance of km-tree clusters with set\_3
- d Performance of km-tree clusters with set\_4

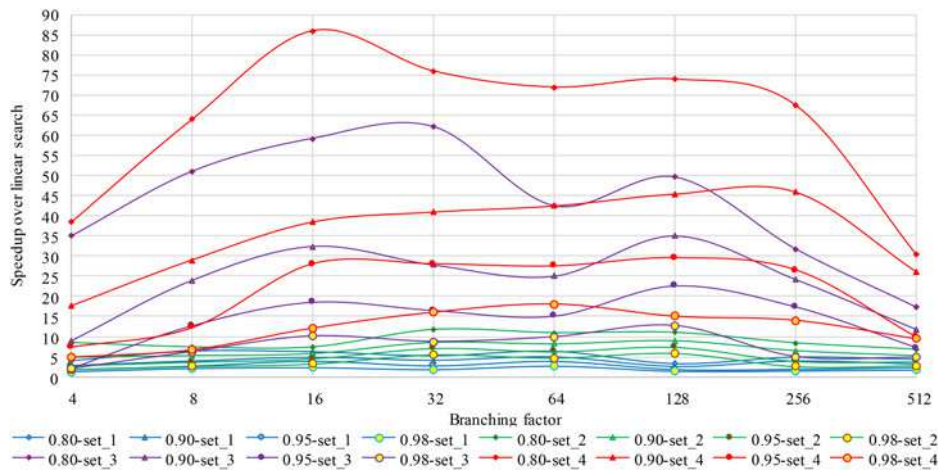


Fig. 9 Speed ups for certain precisions of different databases

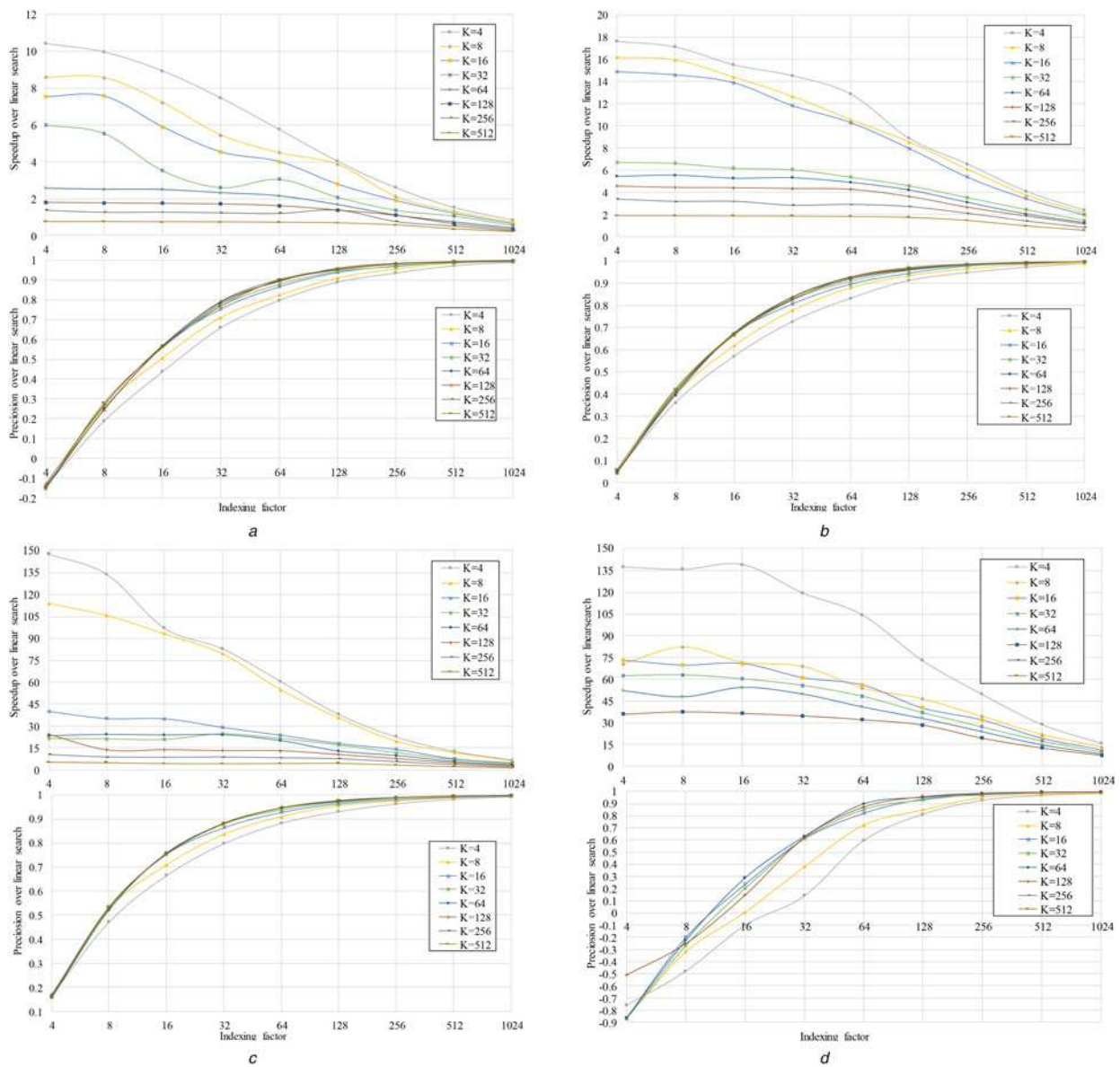


Fig. 10 Precision and speed up of kd-tree clusters with different branching factor  $k$  and searching factor  $r$

- a Performance of kd-tree clusters with set\_1
- b Performance of kd-tree clusters with set\_2
- c Performance of kd-tree clusters with set\_3
- d Performance of kd-tree clusters with set\_4

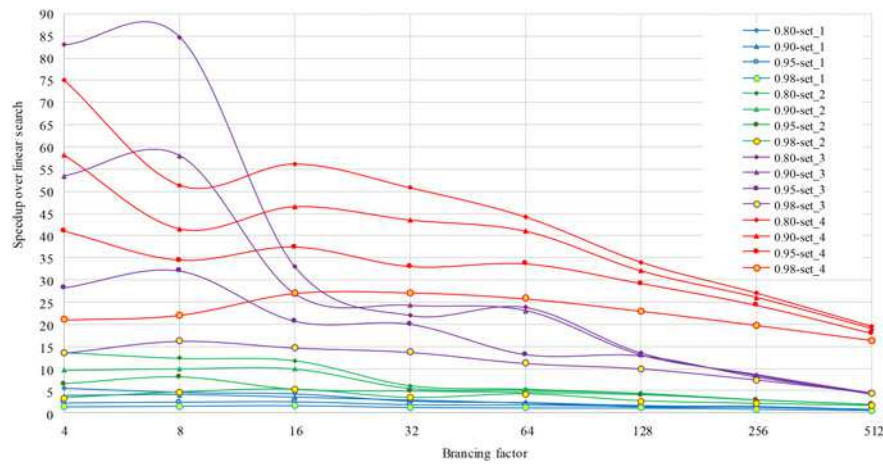


Fig. 11 Speed ups for certain precisions of different databases

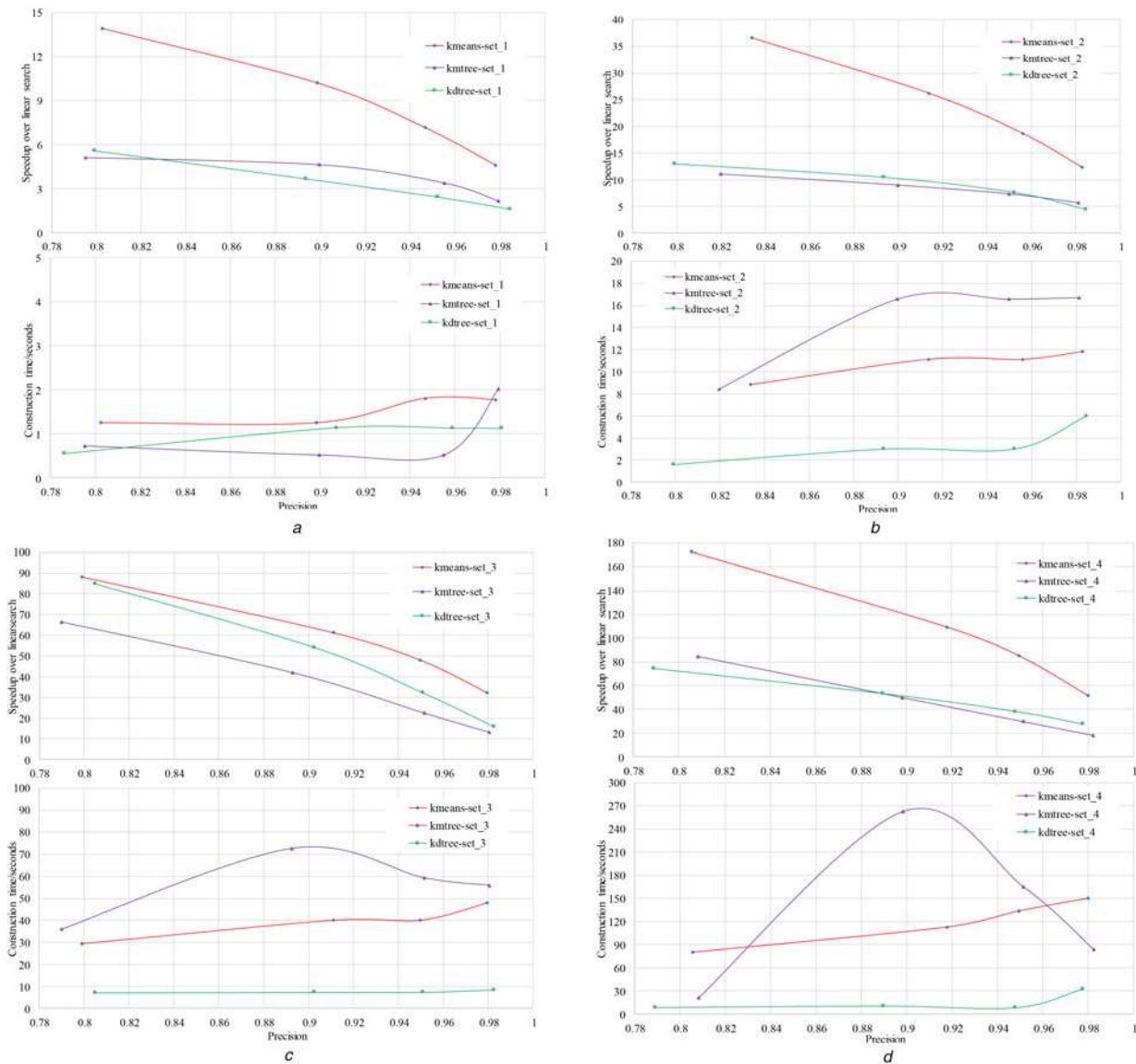


Fig. 12 Comparisons over three algorithms on four different databases

- a Speed ups and construction time for set\_1
- b Speed ups and construction time for set\_2
- c Speed ups and construction time for set\_3
- d Speed ups and construction time for set\_4

#### 4 CFI optimisation based on $r$ -nearest $k$ -means searching

The image clustering process of CFI algorithm is the matching process of image feature points and large-scale feature points set. In the CFI process, the image clustering process costs 64 ms, which accounts for 74% of the whole time. After adopting the  $r$ -nearest  $k$ -means searching algorithm, the process can achieve acceleration by 5–20 times while maintaining a matching precision of 98%, and the whole process of CFI can achieve acceleration by 2–3 times. In this part, the linear searching algorithm of CFI is replaced by  $r$ -nearest  $k$ -means to observe the matching performance.

In the experiments, the basic database is formed by SIFT features and SURF features extracted from image dataset ISKLRS, which contains 5280 images. We use different train set ratios to get datasets of different scales. The numbers of feature points with different train set ratios are shown in Fig. 13. The number of

feature points decreases with the decrease of train set ratio, so different branching factors  $k$  and searching factors  $r$  are needed for different train set ratios. According to the results in Table 2, the parameters for the feature databases of different train set ratios are shown in Table 3.

The experimental results of SIFT feature points are shown in Fig. 14. When the train set ratio is only 1/9, CFI can still maintain a recognition rate of more than 98%, and the recognition speed can reach 15.87 fps. Fig. 14b shows that the recognition rate of CFI is almost unaffected by the  $r$ -nearest  $k$ -means searching and matching process, and the recognition speed increases by several times. We can see that the recognition speed of CFI is almost not affected by the size of feature database after  $r$ -nearest  $k$ -means is used for image clustering because the time feature matching process costs will not increase with the increase of the size of feature database.

The experimental results of SURF feature points are shown in Fig. 15. We can see that it can achieve a recognition speed

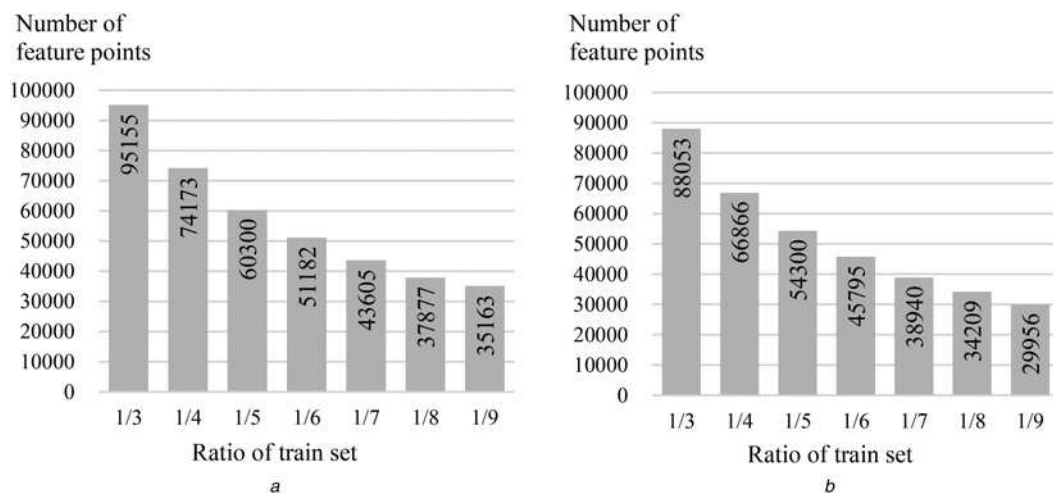


Fig. 13 Number of ISKLRS feature points

a SIFT database  
b SURF database

Table 3 Parameters for  $r$ -nearest  $k$ -means clusters of different train set ratios

| Ratio | 1/3      | 1/4      | 1/5      | 1/6     | 1/7      | 1/8      | 1/9      |
|-------|----------|----------|----------|---------|----------|----------|----------|
| SIFT  | (1200,6) | (1100,6) | (1000,8) | (900,8) | (800,10) | (720,10) | (680,10) |
| SURF  | (1200,5) | (1100,5) | (1000,7) | (900,7) | (800,9)  | (720,9)  | (680,9)  |

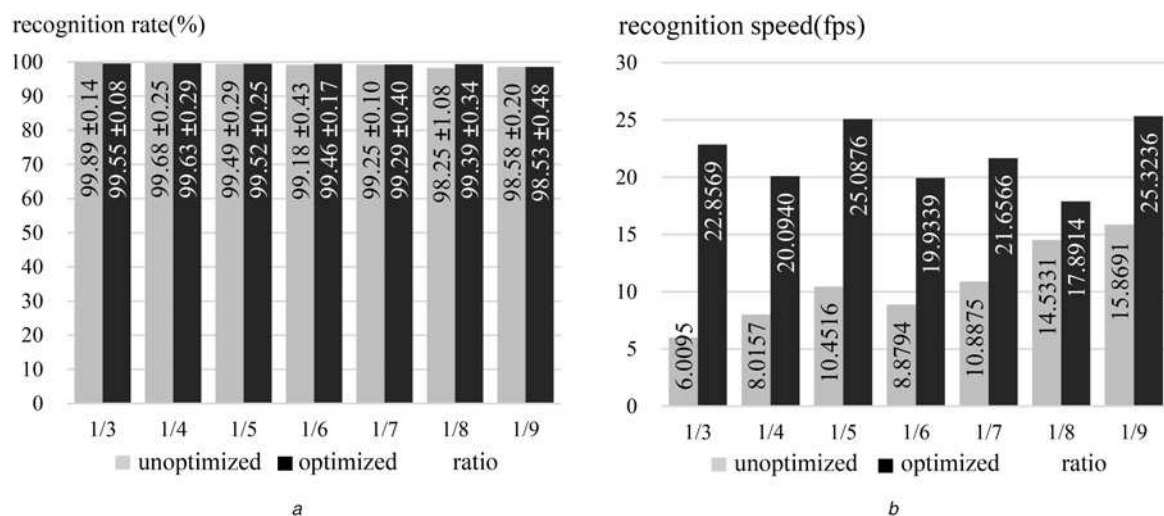
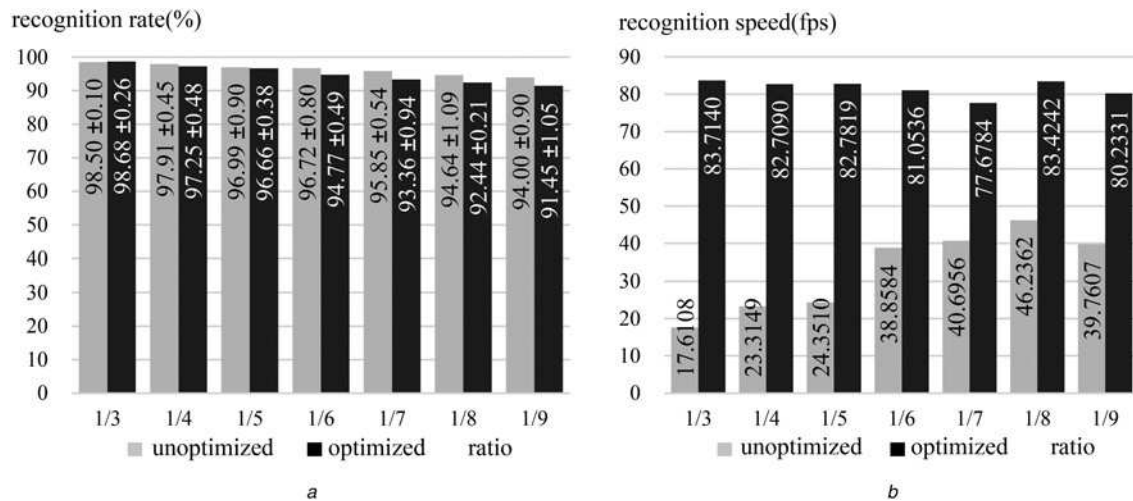


Fig. 14 SIFT optimisation performance

a SIFT recognition rate  
b SIFT recognition speed



**Fig. 15** SURF optimisation performance

a SURF recognition rate  
b SURF recognition speed

of 50–80 fps and the recognition rate is not greatly affected. SURF can achieve a faster recognition speed and searching speed than SIFT, so the overall speed of CFI based on SURF will be higher than that of SIFT. However, the robustness of SURF is not so good as that of SIFT, so the recognition rate will be lower than that of SIFT, especially when the ratio of the train set is small.

## 5 Conclusion

In this paper, we propose a fast feature matching algorithm based on  $r$ -nearest  $k$ -means clusters searching, which is suitable for high-dimensional invariant feature matching problem. It clusters the database into  $k$  clusters by non-iterative  $k$ -means clustering and adopts  $r$ -nearest search strategy that only searches matching features in the  $r$ -nearest clusters of the  $k$  clusters. Therefore, our algorithm can ensure a high probability to contain the NN for the query point. Furthermore, we have improved precision significantly by changing the general matching rule to adapt to our searching strategy to eliminate wrong matches. In addition, we have conducted experiments to compare our algorithm with two widely applied ones. Experiment results show that our algorithm improves matching speed by 78 times and achieves precision to 95% for a 280 k SIFT database, which is superior to two most popular methods, priority search km-tree and random kd-tree, especially that of priority search km-tree which is also based on  $k$ -means clustering for the invariant database. Our new methods generate double speed up over km-tree while requires much less construction time. Finally, we carry out CFI optimisation experiment based on ISKLRS dataset. The results show that the recognition speed of CFI can increase by 2–3 times with  $r$ -nearest  $k$ -means searching, and it has little impact on recognition rate. In summary, we have developed a faster and more precise method than priority search km-tree and random kd-tree for high-dimensional invariant feature matching.

Currently, our algorithm is being applied in the research of dangerous scenes recognition in the home environment. It will improve the speed and precision of fire detection, which can effectively prevent fire and reduce fire losses.

## 6 Acknowledgment

This work was supported by the Joint Research Fund for The Ministry of Education of China and China Mobile (MCM20170208). This manuscript is an extension of a conference paper presented at the 13th Chinese Conference on Intelligent Robotics.

## 7 References

- [1] Li, Y., Snavely, N., Huttenlocher, D.P.: 'Location recognition using prioritized feature matching'. European Conf. Computer Vision, Hersonissos, Greece, 2010, vol. 6312, pp. 791–804
- [2] Liu, L., Yu, M., Shao, L.: 'Unsupervised local feature hashing for image similarity search', *IEEE Trans. Cybern.*, 2015, **46**, (11), pp. 2548–2558
- [3] Muja, M., Lowe, D.G.: 'Scalable nearest neighbor algorithms for high dimensional data', *IEEE Trans. Pattern Anal. Mach. Intell.*, 2014, **36**, (11), pp. 2227–2240
- [4] Liu, T., Moore, A.W., Gray, A., et al.: 'An investigation of practical approximate nearest neighbor algorithms'. Int. Conf. Neural Information Processing Systems, Calcutta, India, 2004, vol. 17, pp. 825–832
- [5] Guo, F., Yang, J., Chen, Y., et al.: 'Research on image detection and matching based on SIFT features'. 2018 Third Int. Conf. Control and Robotics Engineering (ICCRE), Nagoya, Japan, 2018, pp. 130–134
- [6] Fu, Q., Wang, H.: 'A fast image stitching algorithm based on SURF'. IEEE/CIC Int. Conf. Communications in China, Qingdao, China, 2017, pp. 1–4
- [7] Cai, Z., Xiao, D.G.: 'Feature matching algorithm based on KAZE and fast approximate nearest neighbor search'. Int. Conf. Computer Science and Service System, Bangkok, Thailand, 2014, pp. 270–273
- [8] Sinkar, S.V., Deshpande, A.M.: 'Object recognition with plain background by using ANN and SIFT based features'. Int. Conf. Information Processing IEEE, Pune, India, 2016, pp. 575–580
- [9] Rublee, E., Rabaud, V., Konolige, K., et al.: 'ORB: an efficient alternative to SIFT or SURF'. 2011 Int. Conf. Computer Vision, Barcelona, 2011, pp. 2564–2571
- [10] Calonder, M., Lepetit, V., Strecha, C., et al.: 'BRIEF: binary robust independent elementary features'. European Conf. Computer Vision, Hersonissos, Greece, 2010, vol. 6314, pp. 778–792
- [11] Jia, Y., Wang, J., Zeng, G., et al.: 'Optimizing kd-trees for scalable visual descriptor indexing', *Comput. Vis. Pattern Recognit.*, 2010, **119**, (5), pp. 3392–3399
- [12] Wang, C., Lu, Z., Liao, Q., et al.: 'Locality sensitive hashing based deep matching for optical flow estimation'. 2017 IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP), New Orleans, LA, USA, 2017, pp. 1472–1476
- [13] Hajebi, K., Abbasi-Yadkori, Y., Shahbazi, H., et al.: 'Fast approximate nearest-neighbor search with  $k$ -nearest neighbor graph'. IJCAI 2011 Proc. Int. Joint Conf. Artificial Intelligence, Barcelona, Catalonia, Spain, 2009, pp. 1312–1317
- [14] Muja, M.: 'Fast approximate nearest neighbors with automatic algorithm configuration'. Int. Conf. Computer Vision Theory and Application, Kyoto, Japan, 2009, vol. 1, pp. 331–340
- [15] Muja, M.: 'FLANN -fast library for approximate nearest neighbors user manual', 2013
- [16] Lowe, D.G., Lowe, D.G.: 'Distinctive image features from scale-invariant keypoints', *Int. J. Comput. Vis.*, 2004, **60**, (2), pp. 91–110
- [17] Rosten, E., Drummond, T.: 'Machine learning for high-speed corner detection'. European Conf. Computer Vision 2006, Graz, Austria, 2006, pp. 430–443
- [18] Pronobis, A., Caputo, B.: 'The  $k$ th-INDECS database' (KTH Royal Institute of Technology, Stockholm, Sweden, 2005)
- [19] Zhang, L., Zhen, X., Shao, L.: 'Learning object-to-class kernels for scene classification', *IEEE Trans. Image Process.*, 2014, **23**, (8), pp. 3241–3253
- [20] Ullah, M.M., Pronobis, A., Caputo, B., et al.: 'The cold database' (Idiap, Martigny, Switzerland, 2007)