Multimedia Retrieval Exercise Course

11 Local Features: Feature Matching

Kimiaki Shirahama, D.E.

Research Group for Pattern Recognition
Institute for Vision and Graphics
University of Siegen, Germany

Overview of Today's Lesson

- Feature Matching by Nearest Neighbour (NN) Search
  - ❑ Brute Force NN Search

- Fast NN Search
  - ❑ k-d Tree
  - ❑ Randomised k-d Trees

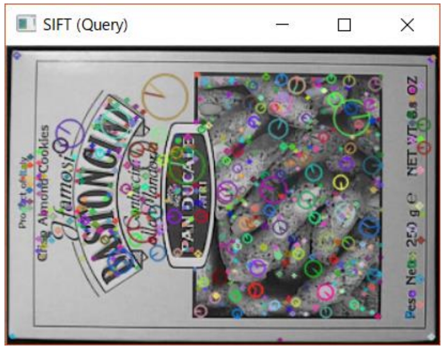
- Online Feature Matching

Feature Matching Task

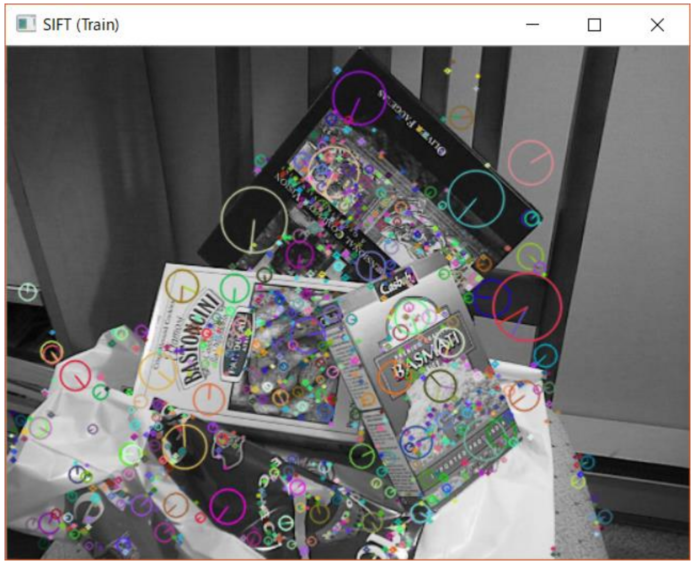
For each descriptor in the query set, find the most similar descriptor in the train set

NOTE: The "train set" is defined in OpenCV's cv::DescriptorMatcher:
https://docs.opencv.org/trunk/db/d39/classcv_1_1DescriptorMatcher.html#a0f046f47b68ec7074391e1e85c750cba
But, the meaning of the "train set" is not like training examples in the context of machine learning. The "train set" here is used to construct (train) *an index to perform enables fast feature matching*.

Query: 604 descriptors



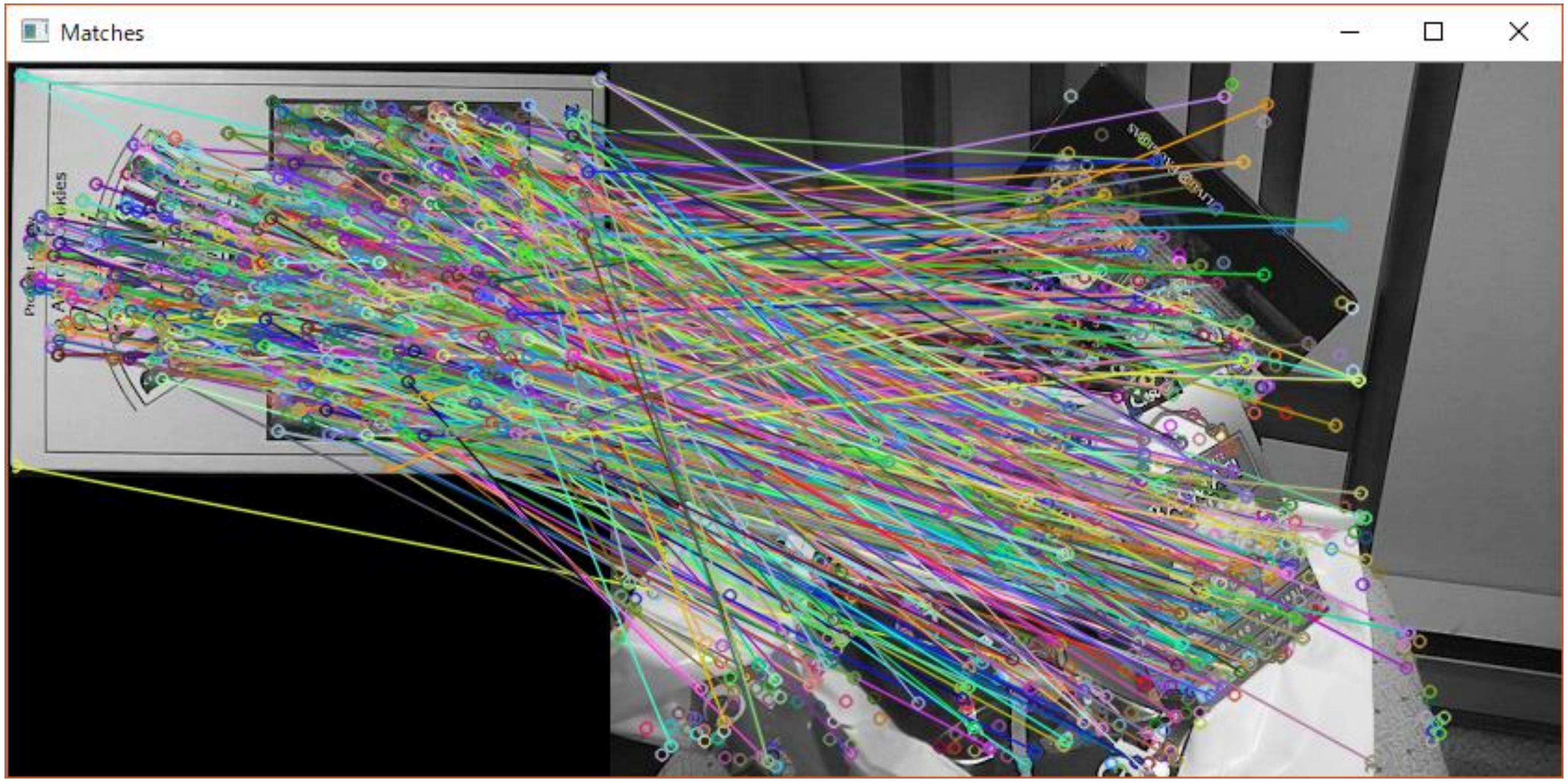
Train: 969 descriptors



Feature Matching Task

For each descriptor in the query set, find the most similar descriptor in the train set

NOTE: The "train set" is defined in OpenCV's cv::DescriptorMatcher:
https://docs.opencv.org/trunk/db/d39/classcv_1_1DescriptorMatcher.html#a0f046f47b68ec7074391e1e85c750cba
But, the meaning of the "train set" is not like training examples in the context of machine learning. The "train set" here is used to construct (train) *an index to perform enables fast feature matching*.

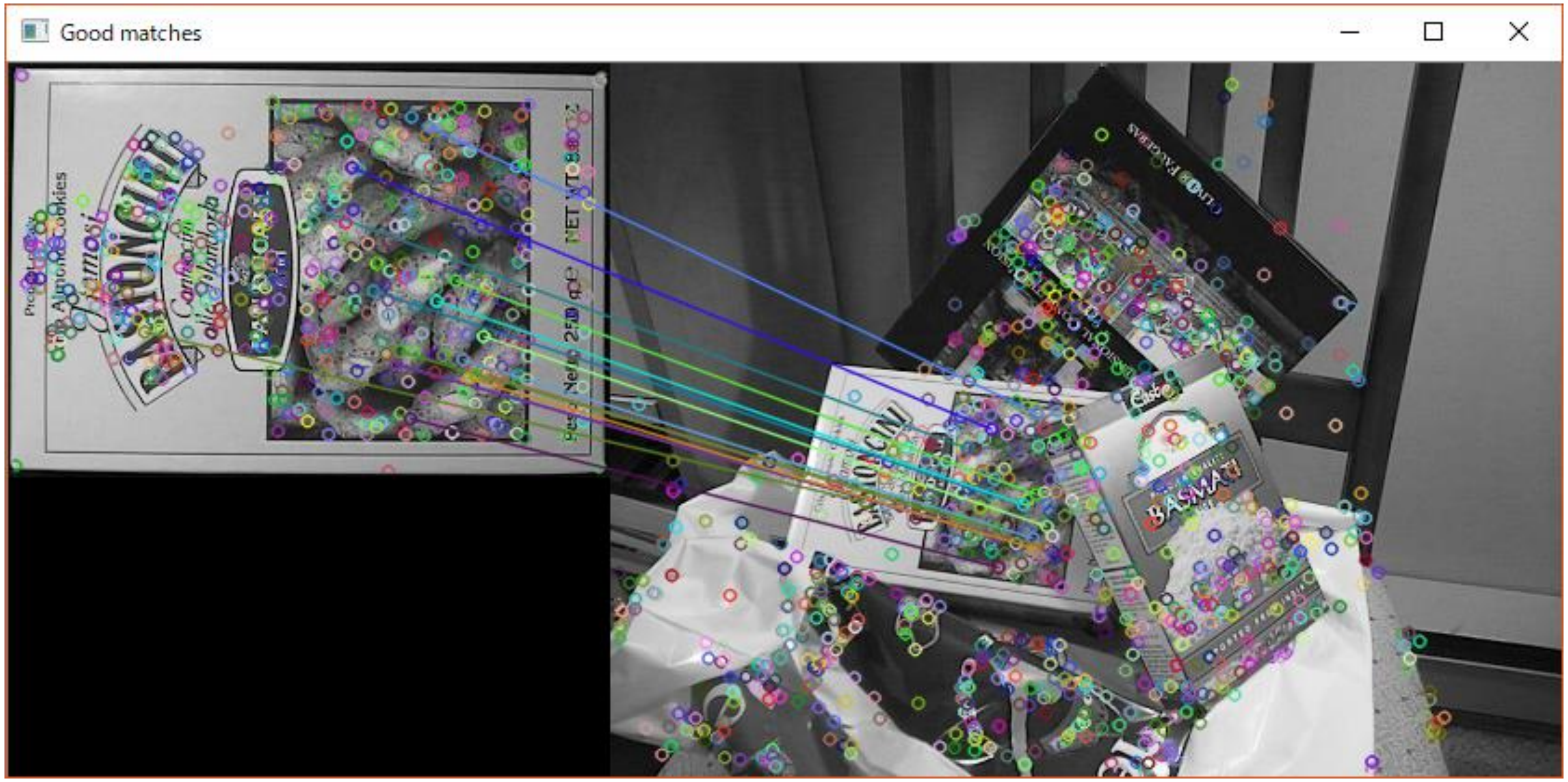


After filtering not-good matches…

Feature Matching Task

For each descriptor in the query set, find the most similar descriptor in the train set

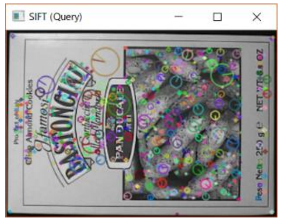
NOTE: The "train set" is defined in OpenCV's cv::DescriptorMatcher:
https://docs.opencv.org/trunk/db/d39/classcv_1_1DescriptorMatcher.html#a0f046f47b68ec7074391e1e85c750cba
But, the meaning of the "train set" is not like training examples in the context of machine learning. The "train set" here is used to construct (train) *an index to perform enables fast feature matching*.

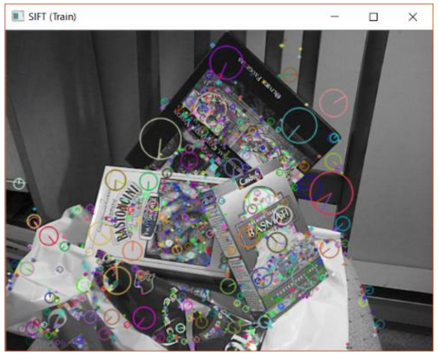


After filtering not-good matches…

Nearest Neighbour (NN) Search

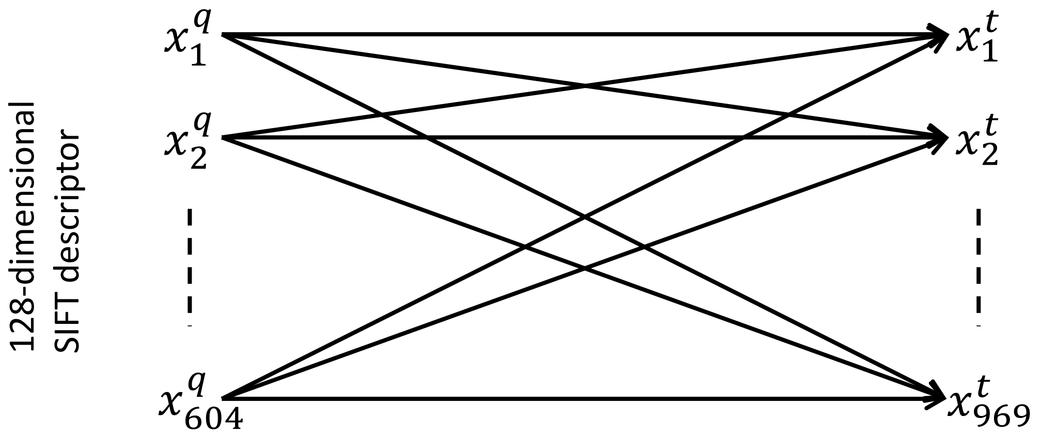
Brute force: For each descriptor in the query set, compute the distances to all the descriptors in the train set and find the most similar one.





Query: 604 descriptors

Train: 969 descriptors



In most cases, the brute force NN search is OK.

But, it is very slow when dealing with many (thousands of) descriptors.

Code of Feature Matching by Brute Force NN Search (1/2)

This code is based on https://docs.opencv.org/trunk/d5/dde/tutorial_feature_description.html

```
#include <iostream>
#include <chrono>
#include "opencv2/core.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/xfeatures2d.hpp"

using namespace std;
using namespace cv;
using namespace cv::xfeatures2d;

void readImage(string filename, Mat &img) {
   cout << ">> Read the image, " << filename << endl;
   img = imread(filename, cv::IMREAD_GRAYSCALE);
   if (!img.data) {
      cout << "Cannot read the image, " << filename << endl;
      int a; cin >> a; exit(0);
   }
}

int main(int argc, char** argv) {

   string img_filename1 = "<Some image filename 1>";
   string img_filename2 = "<Some image filename 2>";
   Mat img1, img2;
   readImage(img_filename1, img1);
   readImage(img_filename2, img2);

   vector<KeyPoint> keypoints1, keypoints2;
   Mat descriptors1, descriptors2;

   Ptr<SIFT> detector = SIFT::create();
   detector->detectAndCompute(img1, Mat(), keypoints1, descriptors1);
   detector->detectAndCompute(img2, Mat(), keypoints2, descriptors2);
   cout << ">> Query size: " << descriptors1.rows << " x " << descriptors1.cols << endl;
   cout << ">> Train size: " << descriptors2.rows << " x " << descriptors2.cols << endl;
   // (Continue to the next slide)
```

See the following pages for more details:
https://docs.opencv.org/trunk/d5/d3c/classcv_1_1xfeatures2d_1_1SIFT.html
https://docs.opencv.org/trunk/d0/d13/classcv_1_1Feature2D.html

Code of Feature Matching by Brute Force NN Search (2/2)

```
    chrono::system_clock::time_point start = chrono::system_clock::now();
    BFMatcher matcher(NORM_L2);
    vector< DMatch > matches;
    matcher.match(descriptors1, descriptors2, matches);
    chrono::system_clock::time_point end = chrono::system_clock::now();
    double elapsedTime = chrono::duration_cast<chrono::milliseconds>(end - start).count();
    cout << ">> Elapsed time = " << elapsedTime << " milliseconds" << endl;

    Mat img_matches;
    drawMatches(img1, keypoints1, img2, keypoints2, matches, img_matches);

    double min_dist = 1000;
    for (int i = 0; i < descriptors1.rows; i++) {
      cout << "-- " << matches[i].queryIdx << "th descriptor in the query set is matched with"
        << matches[i].trainIdx << "th descriptor in the train set with the distance " << matches[i].distance << endl;
      if (matches[i].distance < min_dist) min_dist = matches[i].distance;
    }
    cout << "-- Min dist : " << min_dist << endl;

    vector< DMatch > good_matches;
    for (int i = 0; i < descriptors1.rows; i++) {
      if (matches[i].distance <= 100.0)
        good_matches.push_back(matches[i]);
    }

    Mat img_good_matches;
    drawMatches(img1, keypoints1, img2, keypoints2, good_matches, img_good_matches);

    imshow("Matches", img_matches);
    imshow("Good matches", img_good_matches);
    waitKey(0);

    return 0;

}
```

BFMatcher: https://docs.opencv.org/trunk/d3/da1/classcv_1_1BFMatcher.html
match function: https://docs.opencv.org/trunk/db/d39/classcv_1_1DescriptorMatcher.html

DMatch: https://docs.opencv.org/trunk/d4/de0/classcv_1_1DMatch.html

If you want, you can use another filtering approach using "min_dist":

```
for( int i = 0; i < descriptors1.rows; i++ ) {
  if( matches[i].distance <= 2.0 * min_dist )
    good_matches.push_back( matches[i]);
}
```

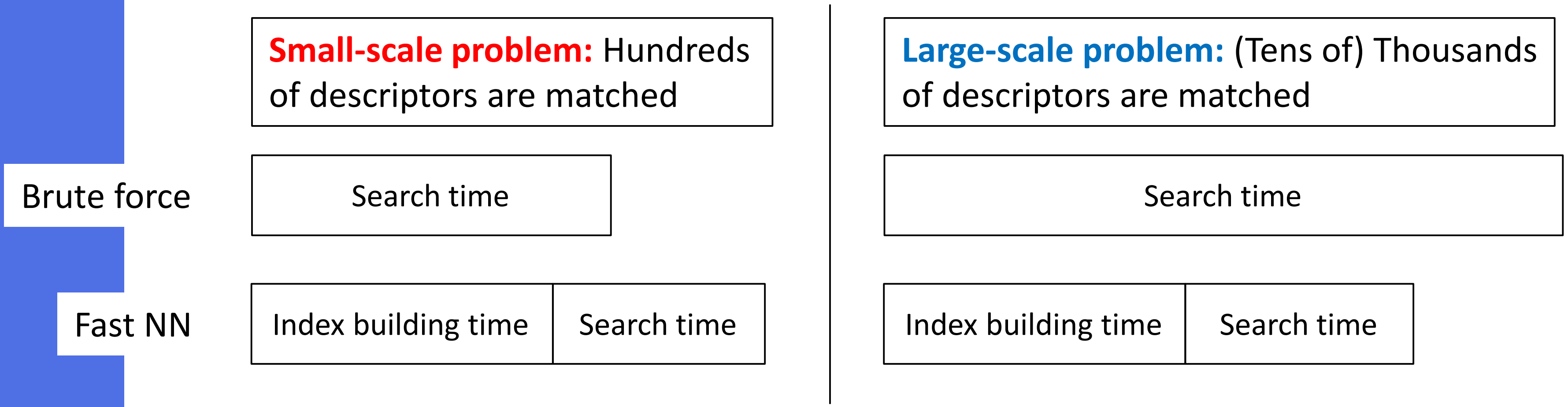
Also, an image retrieval process can be implemented based on the number of DMatch or the sum of DMatch's distance in good_matches

Fast NN Search

For each descriptor in the query set, find a very similar descriptor in the train set in a much shorter time.

In almost all cases, this is the most similar descriptor, but it is not guaranteed.

Fast NN approaches usually build an **index** to make the search process faster (also there is a memory overhead to allocate the index)



Small-scale problem: Hundreds of descriptors are matched

Large-scale problem: (Tens of) Thousands of descriptors are matched

In many cases, the brute force search is better than the fast NN search.
But, in very large-scale problems, the fast NN search is more than 100 times faster than the brute force search

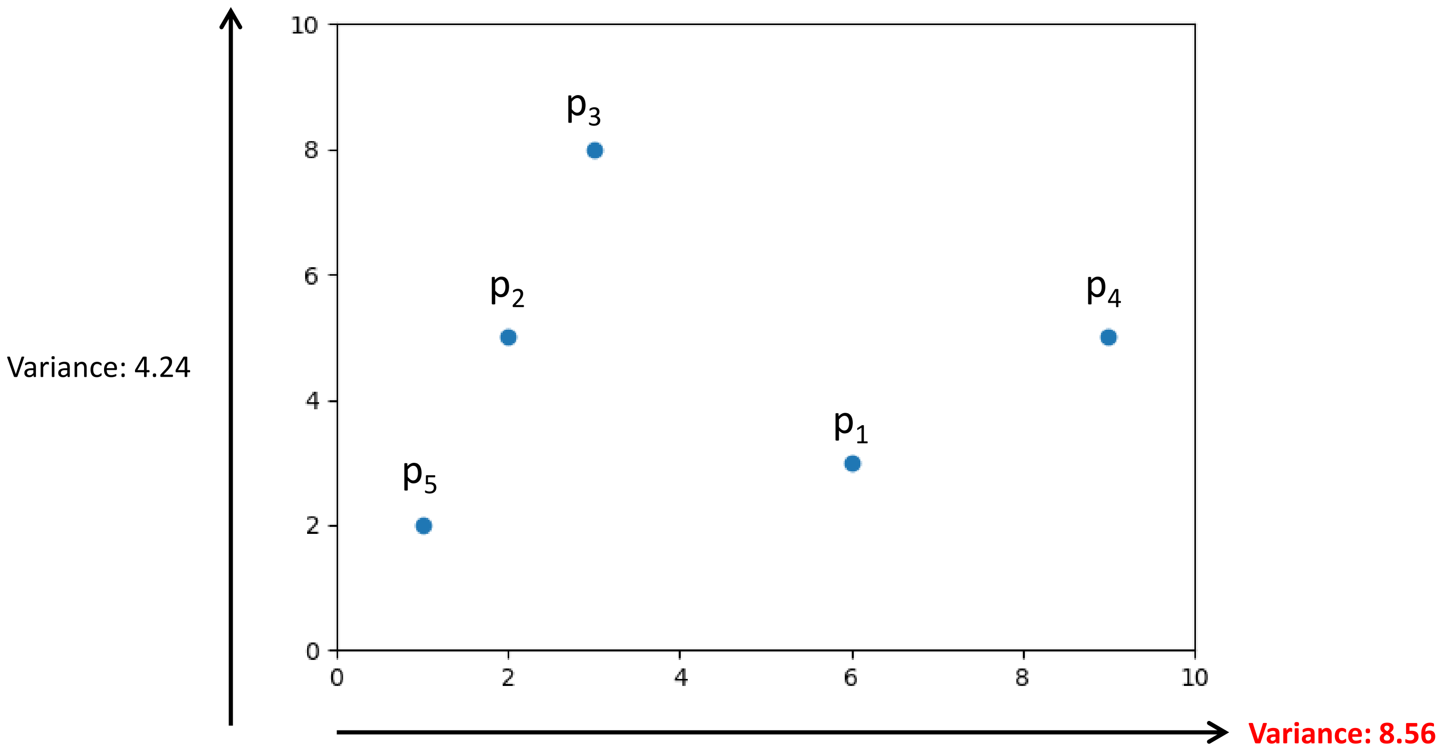
k-d (k-dimensional) Tree

Binary tree that recursively splits points (descriptors) into two sets along the dimension with the maximum variance

➡ Use the mean value on the split dimension

(Using the median seems more popular, but the mean is used in the OpenCV implementation, see “meanSplit” function in opencv-3.4.0\modules\flann\include\opencv2\flann\kdtree_index.h)

Five 2D points: p_1=(6, 3), p_2=(2, 5), p_3=(3, 8), p_4=(9, 5), p_5=(1, 2)



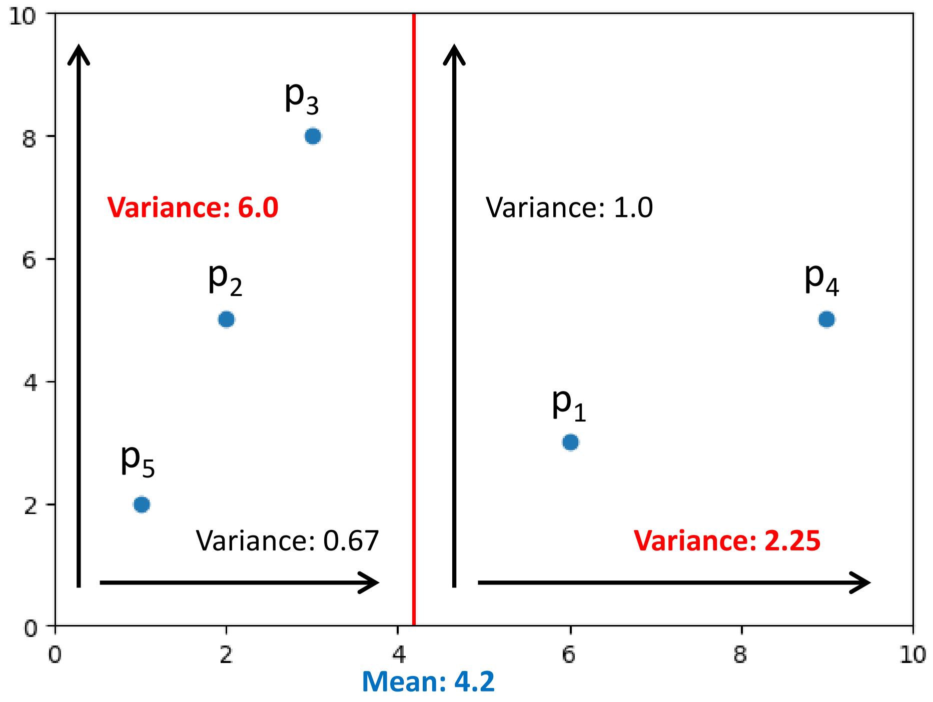
k-d (k-dimensional) Tree

Binary tree that recursively splits points (descriptors) into two sets along the dimension with the maximum variance

➡ Use the mean value on the split dimension

(Using the median seems more popular, but the mean is used in the OpenCV implementation, see "meanSplit" function in opencv-3.4.0\modules\flann\include\opencv2\flann\kdtree_index.h)

Five 2D points: p_1=(6, 3), p_2=(2, 5), p_3=(3, 8), p_4=(9, 5), p_5=(1, 2)



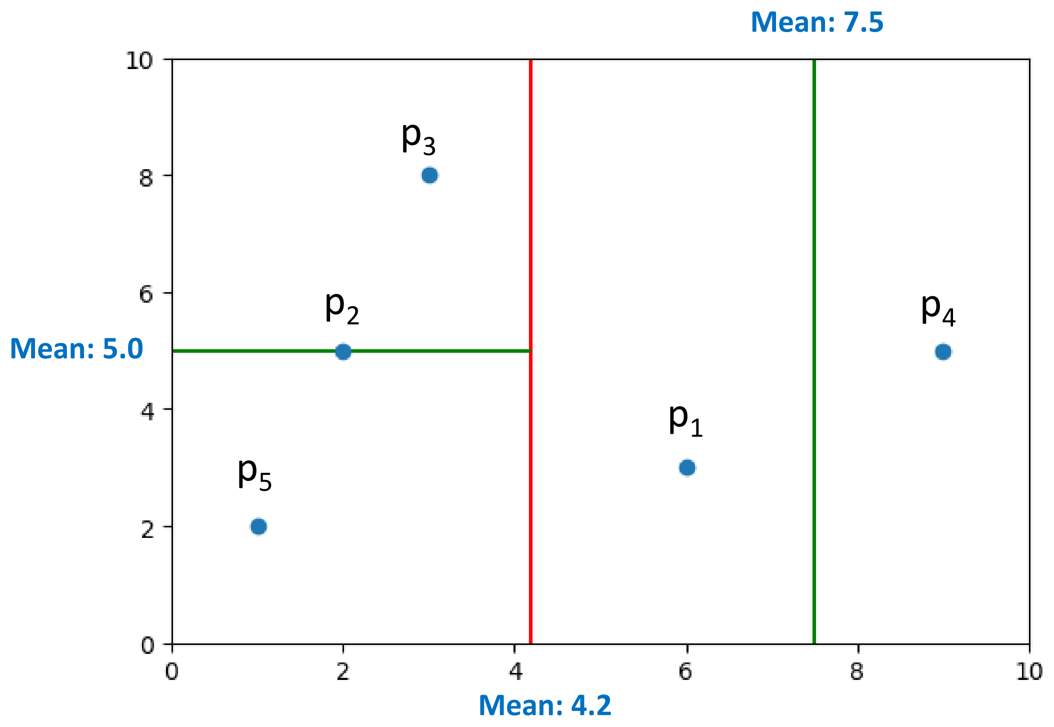
k-d (k-dimensional) Tree

Binary tree that recursively splits points (descriptors) into two sets along the dimension with the maximum variance

➡ Use the mean value on the split dimension

(Using the median seems more popular, but the mean is used in the OpenCV implementation, see “meanSplit” function in opencv-3.4.0\modules\flann\include\opencv2\flann\kdtree_index.h)

Five 2D points: p_1=(6, 3), p_2=(2, 5), p_3=(3, 8), p_4=(9, 5), p_5=(1, 2)



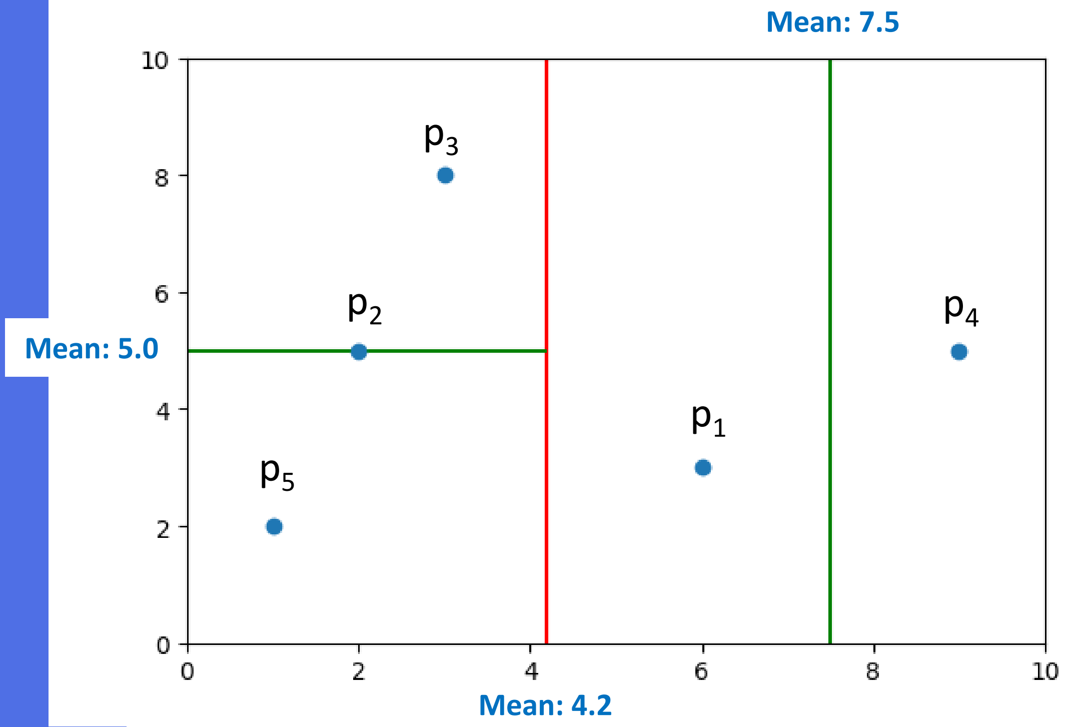
k-d (k-dimensional) Tree

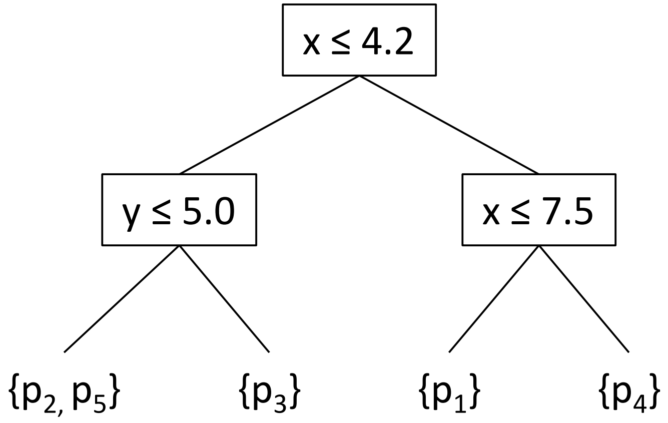
Binary tree that recursively splits points (descriptors) into two sets along the dimension with the maximum variance

⇨ Use the mean value on the split dimension

(Using the median seems more popular, but the mean is used in the OpenCV implementation, see “meanSplit” function in opencv-3.4.0\modules\flann\include\opencv2\flann\kdtree_index.h)

Five 2D points: $p_1=(6, 3)$, $p_2=(2, 5)$, $p_3=(3, 8)$, $p_4=(9, 5)$, $p_5=(1, 2)$





NN Search based on a k-d Tree (1/2)

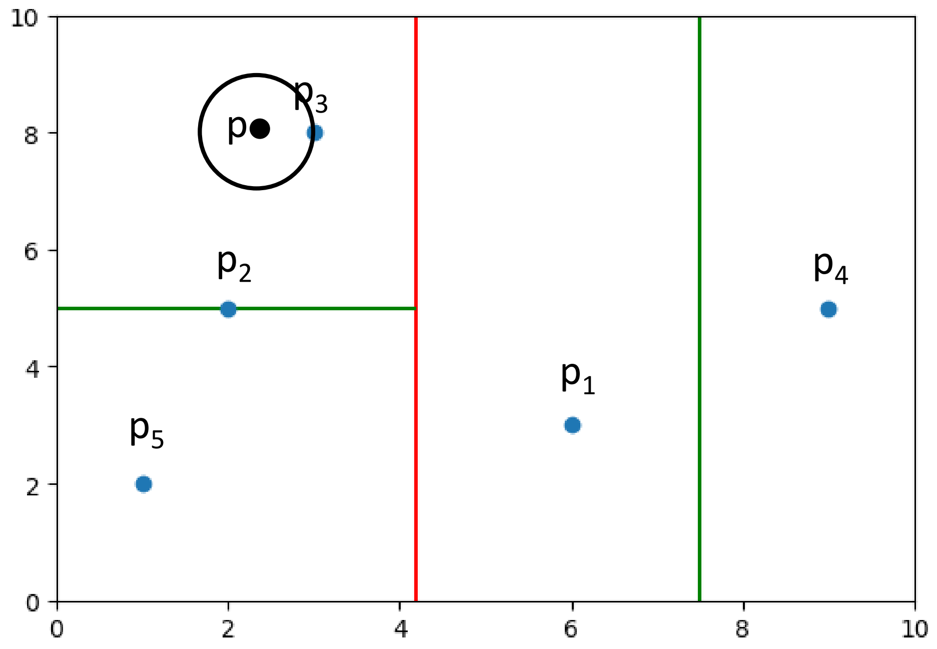
Give a query point p,

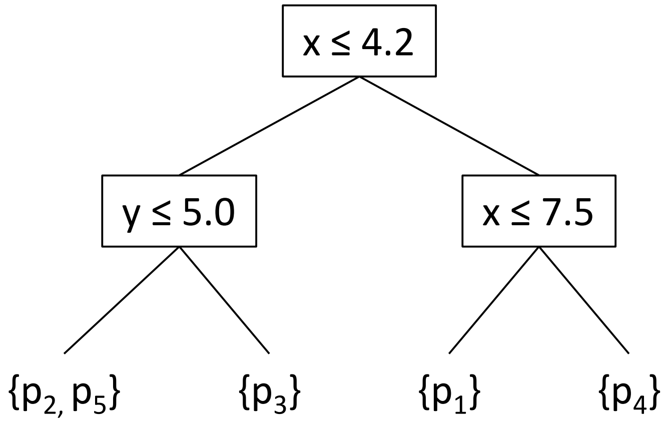
1. Find the leaf node including p, and compute the distance d* between p and its closest point p* in this node
2. Check whether the hypersphere (circle) around p with the radius d* is overlapping boundaries of other nodes
   - If there are overlapping nodes, they need to be recursively checked.
   - If there is no overlapping node, the current p* must be the closest in the whole set of points

   (For more detailed and formal explanation, you can find many papers and webpages on the Web)

Desirable case

Only one node needs to be checked → Very fast search





NN Search based on a k-d Tree (2/2)

Give a query point p,

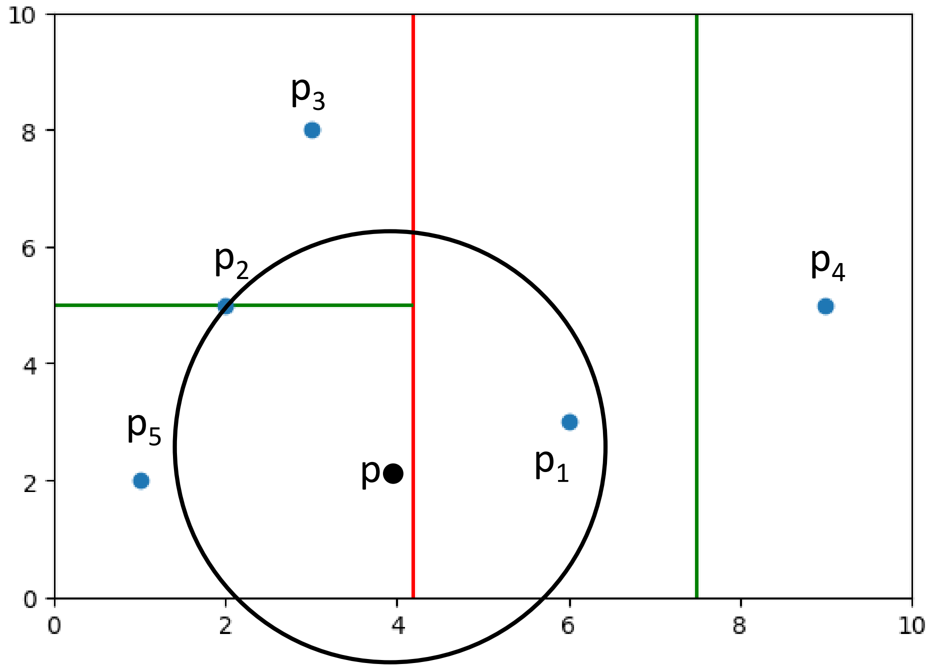
1. Find the leaf node including p, and compute the distance d* between p and its closest point p* in this node
2. Check whether the hypersphere (circle) around p with the radius d* is overlapping boundaries of other nodes
   - If there are overlapping nodes, they need to be recursively checked.
   - If there is no overlapping node, the current p* must be the closest in the whole set of points

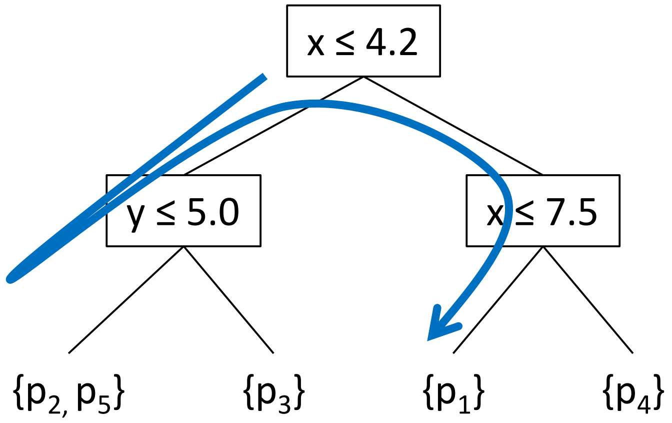
   (For more detailed and formal explanation, you can find many papers and webpages on the Web)

Undesirable case

Multiple nodes need to be checked → Slow down the search process

Especially, the number of overlapping nodes significantly increases for high-dimensional data (dimension-independent splitting cannot consider the distribution in the full feature space)distances → ***Curse of dimensionality***

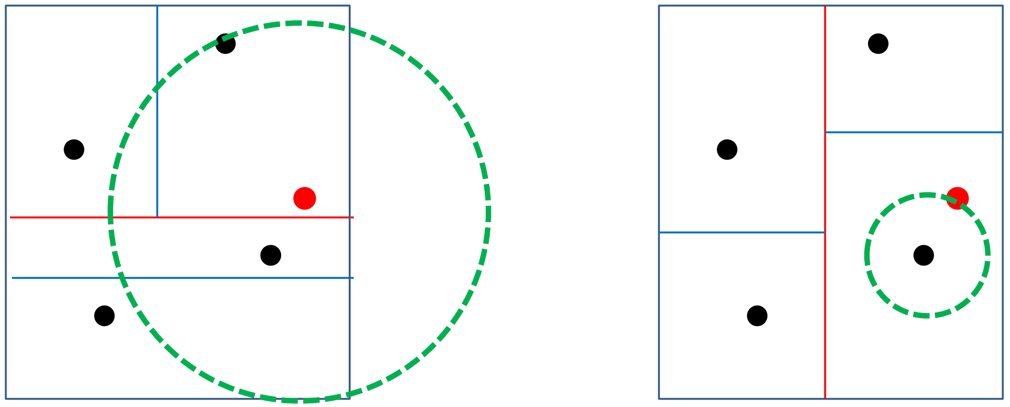




Randomised k-d Trees (k-d Forest)

Build N_t randomised k-d trees and search them in parallel

Each tree is constructed by recursively performing random selection of a split dimension among N_D dimensions with the highest variances



For a query point, an undesirable situation may occur in one randomised k-d tree.
But, another tree may offer a desirable situation for the same query point.

The probability of quickly finding the nearest neighbour is increased

Only N_c leaf nodes in all N_t trees are checked by assuming that a desirable situation occurs in one of those nodes

NOTE1: A priority queue is used to sort nodes in all N_t trees so that more promising nodes are first examined
NOTE2: In OpenCV, by default $N_t = 4$, $N_D = 5$, $N_c = 32$ (see kdtree_index.h and miniflann.cpp)

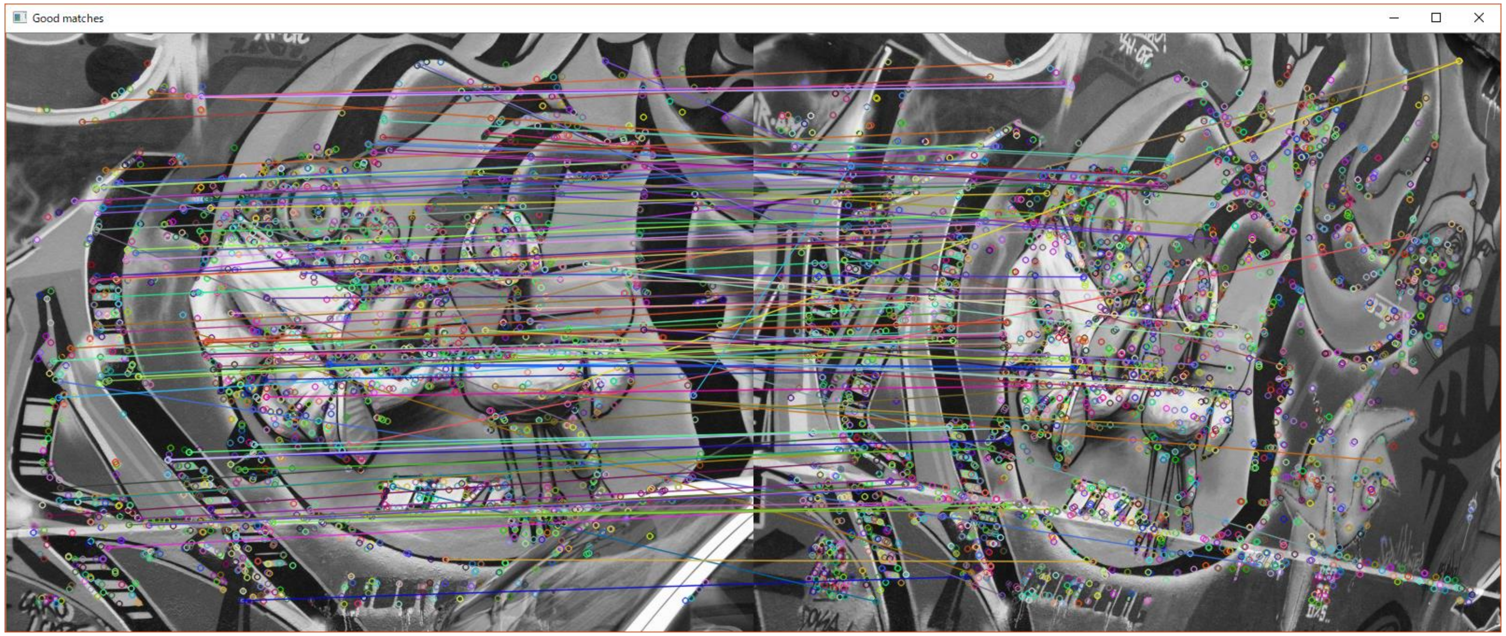
Code of Feature Matching by Randomised k-d Trees

You only have to make the following modification on the code of “Feature Matching by Brute Force NN Search” ☺

```
// BFMatcher matcher(NORM_L2); // Comment out this line
FlannBasedMatcher matcher; // Instead, use this line
// If you want, you can change the default parameters in the following way
// FlannBasedMatcher matcher(new flann::KDTreeIndexParams(1), new flann::SearchParams(4,0,true));
```

Also, you can get more accurate matching by changing SIFT to AKAZE (see https://docs.opencv.org/trunk/db/d70/tutorial_akaze_matching.html).

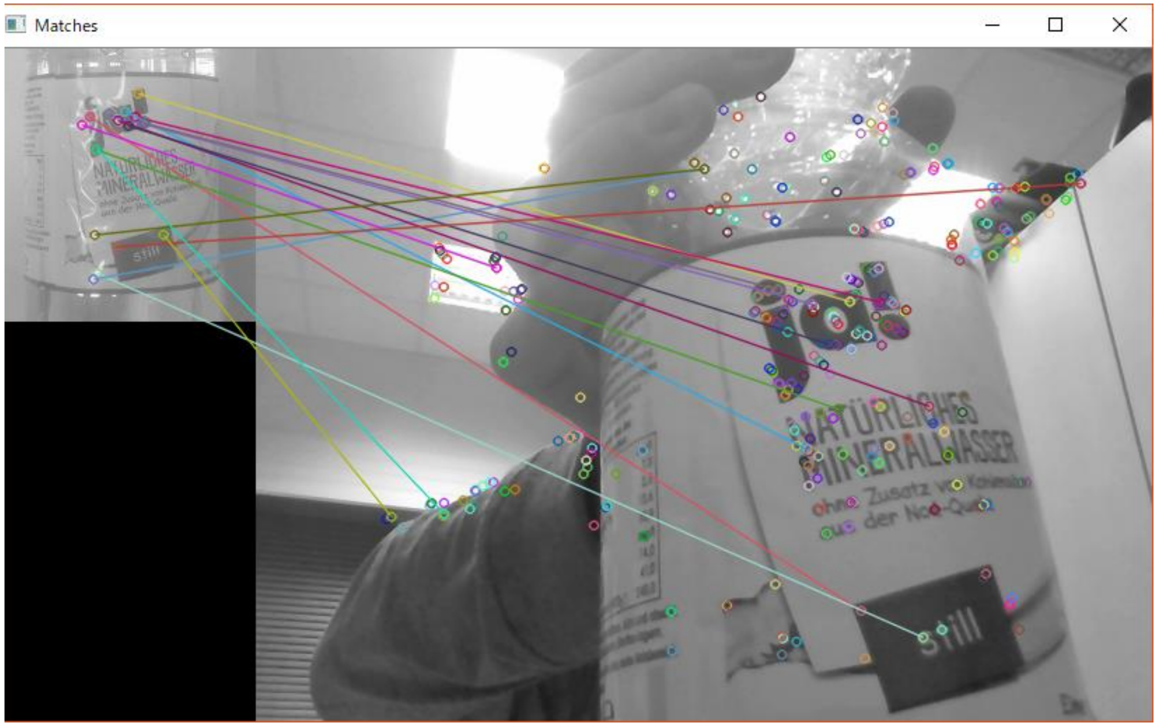
```
//Ptr<SIFT> detector = SIFT::create(); // Comment out this line
Ptr<AKAZE> detector = AKAZE::create(); // Instead, use this line
// The criterion using “min_dist” seems better to find good matches
```



Code of Online Feature Matching

You can change the image for “keypoints2” and “descriptor2” to video frames.

```
// Extract AKAZE descriptors from img1

cv::VideoCapture cap(0);
if (!cap.isOpened()) {
   cout << "!!! ERROR: Cannot open the video" << endl;
   int a; cin >> a; return 1;
}

BFMatcher matcher(NORM_L2); // Or FlannBasedMatcher

int frameID = 0;
while (1) {

   Mat frame;
   cap >> frame;
   cout << ">> Processing " << frameID << "th frame" << endl;
   Mat frame_gray;
   cvtColor(frame, frame_gray, CV_BGR2GRAY);
   // Extract AKAZE descriptors and perform matching
   // …

   cv::imshow("Matches", img_good_matches);
   int key = waitKey(1);
   if (key == 113)   // Terminate if “q” is pressed
      break;

   frameID++;

}
```



That’s all for Multimedia Retrieval exercise course!!! Thank you!!!