Feature Matching with FLANN

Prev Tutorial: **Feature Description**

Next Tutorial: **Features2D + Homography to find a known object**

Goal

In this tutorial you will learn how to:

- Use the **cv::FlannBasedMatcher** interface in order to perform a quick and efficient matching by using the **Clustering and Search in Multi-Dimensional Spaces** module

Warning

You need the OpenCV contrib modules to be able to use the SURF features (alternatives are ORB, KAZE, ... features).

Theory

Classical feature descriptors (SIFT, SURF, ...) are usually compared and matched using the Euclidean distance (or L2-norm). Since SIFT and SURF descriptors represent the histogram of oriented gradient (of the Haar wavelet response for SURF) in a neighborhood, alternatives of the Euclidean distance are histogram-based metrics (χ^2, Earth Mover's Distance (EMD), ...).

Arandjelovic et al. proposed in **[11]** to extend to the RootSIFT descriptor:

> a square root (Hellinger) kernel instead of the standard Euclidean distance to measure the similarity between SIFT descriptors leads to a dramatic performance boost in all stages of the pipeline.

Binary descriptors (ORB, BRISK, ...) are matched using the Hamming distance. This distance is equivalent to count the number of different elements for binary strings (population count after applying a XOR operation):

$$d_{hamming}\left(a,b\right) = \sum_{i=0}^{n-1} \left(a_i \oplus b_i\right)$$

To filter the matches, Lowe proposed in **[137]** to use a distance ratio test to try to eliminate false matches. The distance ratio between the two nearest matches of a considered keypoint is computed and it is a good match when this value is below a threshold. Indeed, this ratio allows helping to discriminate between ambiguous matches (distance ratio between the two nearest neighbors is close to one) and well discriminated matches. The figure below from the SIFT paper illustrates the probability that a match is correct based on the nearest-neighbor distance ratio test.

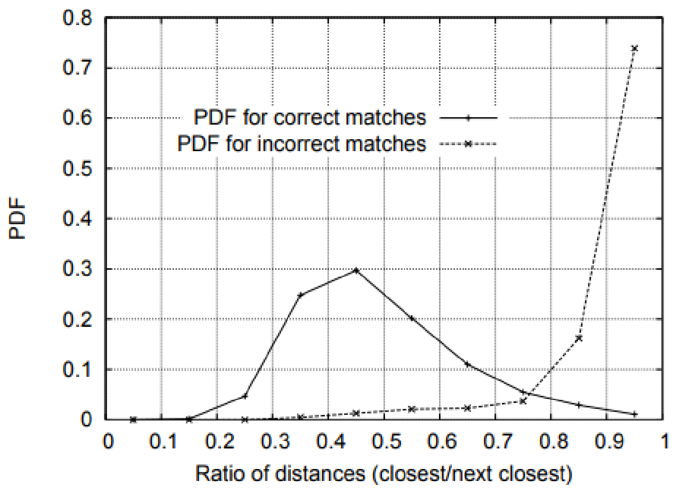


Figure 11: The probability that a match is correct can be determined by taking the ratio of distance from the closest neighbor to the distance of the second closest. Using a database of 40,000 keypoints, the solid line shows the PDF of this ratio for correct matches, while the dotted line is for matches that were incorrect.

Alternative or additional filterering tests are:

- cross check test (good match (f_a, f_b) if feature f_b is the best match for f_a in I_b and feature f_a is the best match for f_b in I_a)
- geometric test (eliminate matches that do not fit to a geometric model, e.g. RANSAC or robust homography for planar objects)

Code

C++ Java Python

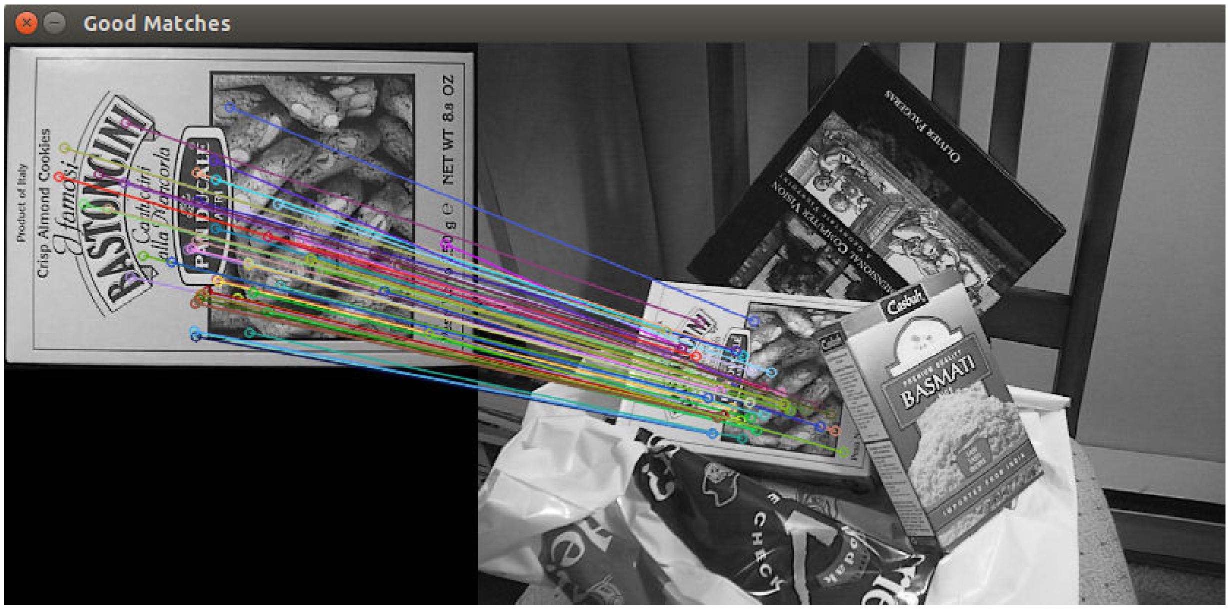
This tutorial code's is shown lines below. You can also download it from here

```
#include <iostream>
#include "opencv2/core.hpp"
#ifdef HAVE_OPENCV_XFEATURES2D
#include "opencv2/highgui.hpp"
#include "opencv2/features2d.hpp"
#include "opencv2/xfeatures2d.hpp"

using namespace cv;
using namespace cv::xfeatures2d;
using std::cout;
using std::endl;

const char* keys =
    "{ help h |                  | Print help message. }"
    "{ input1 | box.png          | Path to input image 1. }"
    "{ input2 | box_in_scene.png | Path to input image 2. }";

int main( int argc, char* argv[] )
{
    CommandLineParser parser( argc, argv, keys );
    Mat img1 = imread( samples::findFile( parser.get<String>("input1") ), IMREAD_GRAYSCALE );
    Mat img2 = imread( samples::findFile( parser.get<String>("input2") ), IMREAD_GRAYSCALE );
    if ( img1.empty() || img2.empty() )
    {
        cout << "Could not open or find the image!\n" << endl;
        parser.printMessage();
        return -1;
    }

    //-- Step 1: Detect the keypoints using SURF Detector, compute the descriptors
    int minHessian = 400;
    Ptr<SURF> detector = SURF::create( minHessian );
    std::vector<KeyPoint> keypoints1, keypoints2;
    Mat descriptors1, descriptors2;
    detector->detectAndCompute( img1, noArray(), keypoints1, descriptors1 );
    detector->detectAndCompute( img2, noArray(), keypoints2, descriptors2 );

    //-- Step 2: Matching descriptor vectors with a FLANN based matcher
    // Since SURF is a floating-point descriptor NORM_L2 is used
    Ptr<DescriptorMatcher> matcher = DescriptorMatcher::create(DescriptorMatcher::FLANNBASED);
    std::vector< std::vector<DMatch> > knn_matches;
    matcher->knnMatch( descriptors1, descriptors2, knn_matches, 2 );

    //-- Filter matches using the Lowe's ratio test
    const float ratio_thresh = 0.7f;
    std::vector<DMatch> good_matches;
    for (size_t i = 0; i < knn_matches.size(); i++)
    {
        if (knn_matches[i][0].distance < ratio_thresh * knn_matches[i][1].distance)
        {
            good_matches.push_back(knn_matches[i][0]);
        }
    }

    //-- Draw matches
    Mat img_matches;
    drawMatches( img1, keypoints1, img2, keypoints2, good_matches, img_matches, Scalar::all(-1),
                 Scalar::all(-1), std::vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS );

    //-- Show detected matches
    imshow("Good Matches", img_matches );

    waitKey();
    return 0;
}
#else
int main()
{
    std::cout << "This tutorial code needs the xfeatures2d contrib module to be run." << std::endl;
    return 0;
}
#endif
```

Explanation

Result

- Here is the result of the SURF feature matching using the distance ratio test:


Good Matches