Matching with detectors and descriptors.

Introduction

This practical course uses OpenCv to illustrates how matching interest points can be uses to detect objects in images. The core of the algorithm is : 1) the detector, 2) the descriptor and 3) the matcher. OpenCv already proposes several methods for each step. A Set of images (objets and full images) are provided for experiments. Moreover, a skeleton of a C++ source file is provided.

1 Matching two images

The following code read two images and find matches from interest points.

```
//
//  main.cpp
//  openCV1
//
//
#include <stdio.h>
#include <iostream>
#include <opencv2/features2d.hpp>
#include <opencv2/xfeatures2d.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/opencv.hpp>
#include <vector>
#include <iostream>

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


/** @function main */
int main( int argc, char** argv )
{
// Read object and scene images
Mat img_object = imread( "img/ip.jpg", CV_LOAD_IMAGE_GRAYSCALE );
Mat img_scene = imread( "img/test_ip.jpg", CV_LOAD_IMAGE_GRAYSCALE );

if( !img_object.data || !img_scene.data )
{ std::cout<< " --(!) Error reading images " << std::endl; return -1; }

//-- Step 1: Detect the keypoints using SURF Detector
//int minHessian = 400;

Ptr<Feature2D> detector = SIFT::create();
```

```
std::vector<KeyPoint> keypoints_object, keypoints_scene;

double t = (double)getTickCount();
detector->detect( img_object, keypoints_object );
detector->detect( img_scene, keypoints_scene );


//-- Step 2: Calculate descriptors (feature vectors)


Mat descriptors_object, descriptors_scene;

detector->compute( img_object, keypoints_object, descriptors_object );
detector->compute( img_scene, keypoints_scene, descriptors_scene );


//-- Step 3: Matching descriptor vectors using FLANN matcher
//Ptr<DescriptorMatcher> matcher = DescriptorMatcher::create("FlannBased");
BFMatcher matcher(NORM_L2);

std::vector< DMatch > matches;
matcher.match( descriptors_object, descriptors_scene, matches );
t = ((double)getTickCount() - t)/getTickFrequency();
cout << "Times passed in seconds: " << t << endl;


double max_dist = 0; double min_dist = 100;

//-- Quick calculation of max and min distances between keypoints
for( int i = 0; i < descriptors_object.rows; i++ )
{ double dist = matches[i].distance;
if( dist < min_dist ) min_dist = dist;
if( dist > max_dist ) max_dist = dist;
}

printf("-- Max dist : %f \n", max_dist );
printf("-- Min dist : %f \n", min_dist );

//-- Draw only "good" matches (i.e. whose distance is less than n*min_dist )
unsigned int n=2;
std::vector< DMatch > good_matches;
for( int i = 0; i < descriptors_object.rows; i++ )
{ if( matches[i].distance < n*min_dist )
{ good_matches.push_back( matches[i]); }
}

Mat img_matches;
drawMatches( img_object, keypoints_object, img_scene, keypoints_scene,
good_matches, img_matches, Scalar::all(-1), Scalar::all(-1),
vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS );
```

```
drawKeypoints( img_scene, keypoints_scene, img_scene, Scalar::all(-1), DrawMatchesFlags::DEFAULT );
drawKeypoints( img_object, keypoints_object, img_object, Scalar::all(-1), DrawMatchesFlags::DEFAULT );

//-- Show detected matches
imshow( "Scene Keypoints", img_scene );
imshow( "Object Keypoints", img_object );
imshow( "Good Matches & Object detection", img_matches );

waitKey(0);
return 0;
}
```

1. Find in this code the detector, descriptor and matcher and explain the associated parameters.
2. Compare the following detectors in terms of computation time and performances (on the couple of images im1.png and IMG_1326.jpg) : SIFT, SURF, BRISK, ORB. The following instructions are useful to get the computation time.

   ```
   double t = (double)getTickCount();
   // do something ...
   t = ((double)getTickCount() - t)/getTickFrequency();
   cout << "Times passed in seconds: " << t << endl;
   ```

3. Compare the execution time of FlannBasedMatcher with BFMatcher. You may use the virtual class DescriptorMatcher :

   ```
   Ptr<DescriptorMatcher> matcher = DescriptorMatcher::create("FlannBased");
   ```

   Conclude
4. test with im1.png and IMG_1326bis.jpg to show the invariance to a rotation around the optical axis.

Warning : when using binary descriptor, the matching distance must be adapted to binary vectors (Hamming distance) : `BFMatcher matcher(NORM_HAMMING)`

2 Matching with a video sequence

1. Modify the program to match an image with the webcam or with a video sequence.
2. Compare invariance of several descriptors to scale variation, rotation and motion blur.