Features2D + Homography to find a known object

Goal

In this tutorial you will learn how to:

- Use the function findHomography to find the transform between matched keypoints.
- Use the function perspectiveTransform to map the points.

Theory

Code

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

```
#include <stdio.h>
#include <iostream>
#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/calib3d/calib3d.hpp"
#include "opencv2/nonfree/nonfree.hpp"

using namespace cv;

void readme();

/** @function main */
int main( int argc, char** argv )
{
  if( argc != 3 )
  { readme(); return -1; }

  Mat img_object = imread( argv[1], CV_LOAD_IMAGE_GRAYSCALE );
  Mat img_scene = imread( argv[2], 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;

  SurfFeatureDetector detector( minHessian );

  std::vector<KeyPoint> keypoints_object, keypoints_scene;

  detector.detect( img_object, keypoints_object );
  detector.detect( img_scene, keypoints_scene );

  //-- Step 2: Calculate descriptors (feature vectors)
  SurfDescriptorExtractor extractor;

  Mat descriptors_object, descriptors_scene;
```

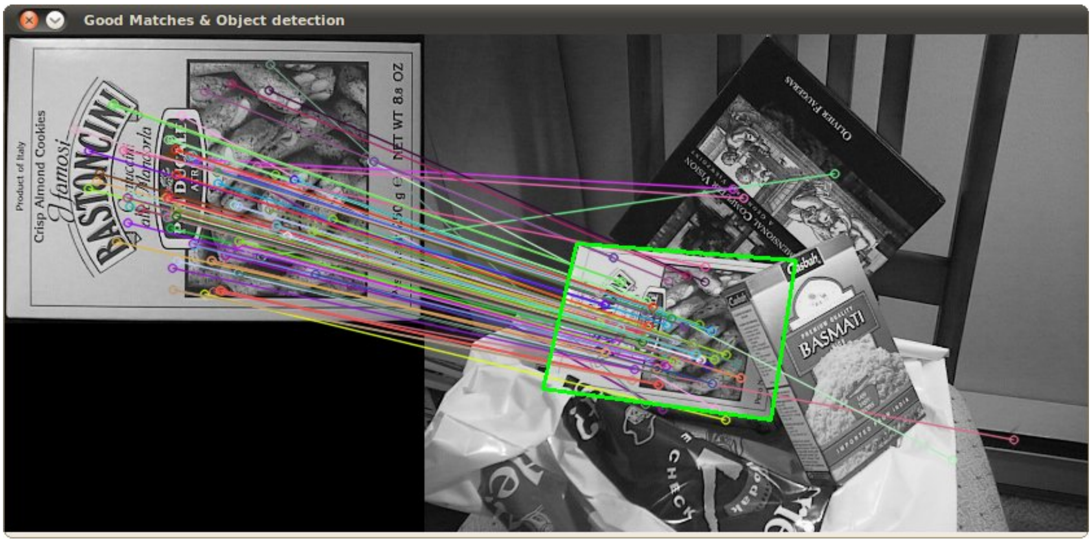
```
extractor.compute( img_object, keypoints_object, descriptors_object );
extractor.compute( img_scene, keypoints_scene, descriptors_scene );

//-- Step 3: Matching descriptor vectors using FLANN matcher
FlannBasedMatcher matcher;
std::vector< DMatch > matches;
matcher.match( descriptors_object, descriptors_scene, matches );

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 3*min_dist )
std::vector< DMatch > good_matches;

for( int i = 0; i < descriptors_object.rows; i++ )
{ if( matches[i].distance < 3*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 );

//-- Localize the object
std::vector<Point2f> obj;
std::vector<Point2f> scene;

for( int i = 0; i < good_matches.size(); i++ )
{
  //-- Get the keypoints from the good matches
  obj.push_back( keypoints_object[ good_matches[i].queryIdx ].pt );
  scene.push_back( keypoints_scene[ good_matches[i].trainIdx ].pt );
}

Mat H = findHomography( obj, scene, CV_RANSAC );

//-- Get the corners from the image_1 ( the object to be "detected" )
std::vector<Point2f> obj_corners(4);
obj_corners[0] = cvPoint(0,0); obj_corners[1] = cvPoint( img_object.cols, 0 );
obj_corners[2] = cvPoint( img_object.cols, img_object.rows ); obj_corners[3] = cvPoi
std::vector<Point2f> scene_corners(4);

perspectiveTransform( obj_corners, scene_corners, H);

//-- Draw lines between the corners (the mapped object in the scene - image_2 )
line( img_matches, scene_corners[0] + Point2f( img_object.cols, 0), scene_corners[1]
line( img_matches, scene_corners[1] + Point2f( img_object.cols, 0), scene_corners[2]
line( img_matches, scene_corners[2] + Point2f( img_object.cols, 0), scene_corners[3]
line( img_matches, scene_corners[3] + Point2f( img_object.cols, 0), scene_corners[0]

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

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

```
/** @function readme */
void readme()
{ std::cout << " Usage: ./SURF_descriptor <img1> <img2>" << std::endl; }
```

Explanation

Result

1. And here is the result for the detected object (highlighted in green)



Help and Feedback

You did not find what you were looking for?

- Ask a question on the **Q&A forum**.
- If you think something is missing or wrong in the documentation, please file a **bug report**.