**\*SIFT.cpp\***

**1. Code Explanation**

**1) overall explanation**

SIFT.cpp detects keypoints using detect() function, and computes the descriptors for a set of keypoints in each image “input1.jpg” and “input2.jpg”. After running SIFT descriptor, it performs the feature matching using nearest neighbor method, and then draw line between nearest neighbor pairs.

**2) code**

//Calculate euclid distance

//vec1, vec2: two matrices. we will calculate euclid distance between them

double euclidDistance(Mat& vec1, Mat& vec2) {

double sum = 0.0;

int dim = vec1.cols;

for (int a = 0; a < dim; a++) {

sum += (vec1.at<float>(0, a) - vec2.at<float>(0, a)) \* (vec1.at<float>(0, a) - vec2.at<float>(0, a));

}

return sqrt(sum);

}

// Find the index of nearest neighbor point from keypoints.

// vec: descriptor for fi, keypoints: g1~gm, descriptors: descriptors of g

int nearestNeighbor(Mat& vec, vector<KeyPoint>& keypoints, Mat& descriptors) {

int neighbor = -1;

double minDist = 1e6; //minimum distance between fi and gj

for (int i = 0; i < descriptors.rows; i++) {

Mat v = descriptors.row(i); // each row of descriptor

if (euclidDistance(vec, v) < minDist) {

minDist = euclidDistance(vec, v);

neighbor = i;

}

}

return neighbor;

}

//Find pairs of points with the smallest distace between them and save them in srcPoints, dstPoints

void findPairs(vector<KeyPoint>& keypoints1, Mat& descriptors1,

vector<KeyPoint>& keypoints2, Mat& descriptors2,

vector<Point2f>& srcPoints, vector<Point2f>& dstPoints, bool crossCheck, bool ratio\_threshold) {

for (int i = 0; i < descriptors1.rows; i++) {

KeyPoint pt1 = keypoints1[i];

Mat desc1 = descriptors1.row(i);

int nn = nearestNeighbor(desc1, keypoints2, descriptors2);

// Refine matching points using ratio\_based thresholding

if (ratio\_threshold) {

int neighbor = -1;

double secondMinDist = 1e6; //second minimum distance between fi and gj

for (int j = 0; j < descriptors2.rows; j++) {

Mat v = descriptors2.row(j); // each row of descriptor

Mat k1 = descriptors2.row(nn); //row that has minimum distance

if (euclidDistance(desc1, k1)<euclidDistance(desc1, v) < secondMinDist) {

secondMinDist = euclidDistance(desc1, v);

neighbor = j;

}

}

int nn2 = neighbor; //index of second nearest neighbor point

Mat k1 = descriptors2.row(nn); Mat k2 = descriptors2.row(nn2);

if ((double)(euclidDistance(desc1, k1) / euclidDistance(desc1, k2)) >= RATIO\_THR) //if ratio < threshold then it is reliable match, otherwise unreliable

continue;

}

// Refine matching points using cross-checking

if (crossCheck) {

Mat desc2 = descriptors2.row(nn);

int l = nearestNeighbor(desc2, keypoints1, descriptors1);

if (i != l) //if unreliable match- 해당 match를 저장하지 않고 다음 pair 로 넘어간다

continue;

}

KeyPoint pt2 = keypoints2[nn];

srcPoints.push\_back(pt1.pt);

dstPoints.push\_back(pt2.pt);

}

}

**3) analysis & explanation of the code**

1. Scale-space extrema detection
2. Keypoint localization
3. Orientation assignment
4. Keypoint descriptor
5. Keypoint matching: using nearest neighbor

**2. Results & analysis**

텍스트, 하늘, 실외, 빨간색이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

Case1) show all the nearest neighbors (crossCheck=false, ratio\_threshold=false)

텍스트, 하늘, 빨간색이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

Case2) Refine feature matching results with using cross-checking

(crossCheck=true, ratio\_threshold=false)

텍스트, 하늘, 빨간색이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

Case3) using both cross-checking and ratio-based thresholding

(crossCheck=true, ratio\_threshold=true)

**\*SURF\_Homography.cpp\***

**1. Code Explanation**

**1) overall explanation**

: This code detects keypoints using detect() function, and computes the descriptors for a set of keypoints in each image “input1.jpg” and “input2.jpg”. After running SURF descriptor, it performs perstpective transformation, and then draw line between the corneres

**2) code**

#include <stdio.h>

#include <iostream>

#include "opencv2/core/core.hpp"

#include "opencv2/highgui/highgui.hpp"

#include "opencv2/calib3d/calib3d.hpp"

#include "opencv2/features2d/features2d.hpp"

#include "opencv2/nonfree/nonfree.hpp"

using namespace cv;

void readme();

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);

//keypoints of img\_object and img\_scene

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;

//descriptors of img\_object and img\_scene

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); //feature matching\*\*\*\*\*\*\*\*\*\*\*\*\*

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;

//draw matches of keypoints from two images on '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 from img\_1 in img\_2

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); //computes the best-fit perspective transformation mapping obj to scene

//-- 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] = cvPoint(0, img\_object.rows);

std::vector<Point2f> scene\_corners(4);

perspectiveTransform(obj\_corners, scene\_corners, H); //performs perstpective transformation

//-- 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] + Point2f(img\_object.cols, 0), Scalar(0, 255, 0), 4);

line(img\_matches, scene\_corners[1] + Point2f(img\_object.cols, 0), scene\_corners[2] + Point2f(img\_object.cols, 0), Scalar(0, 255, 0), 4);

line(img\_matches, scene\_corners[2] + Point2f(img\_object.cols, 0), scene\_corners[3] + Point2f(img\_object.cols, 0), Scalar(0, 255, 0), 4);

line(img\_matches, scene\_corners[3] + Point2f(img\_object.cols, 0), scene\_corners[0] + Point2f(img\_object.cols, 0), Scalar(0, 255, 0), 4);

//-- Show detected matches

imshow("Good Matches & Object detection", img\_matches);

waitKey(0);

return 0;

}

/\*\*

\* @function readme

\*/

void readme()

{

std::cout << " Usage: ./SURF\_Homography <img1> <img2>" << std::endl;

}

**3) analysis & explanation of the code**

텍스트이(가) 표시된 사진

자동 생성된 설명

‘프로젝트 속성>구성 속성>디버깅>명령 인수’ 에 input1.jpg, input2.jpg의 경로명을 입력한다. 이는 argv[1], argv[2]로 전달된다.

1. Detect the keypoints using SURF Detector
2. Calculate descriptors
3. Matching descriptor vectors using FLANN matcher
4. Draw only "good" matches (i.e. whose distance is less than 3\*min\_dist )
5. computes the best-fit perspective transformation mapping obj to scene and then draw lines between the corners

**2. Results & analysis**

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

argv[1]=”input1.jpg”, argv[2]=”input2.jpg”

텍스트, 전자기기, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

argv[1]=”input2.jpg”, argv[2]=”input1.jpg”