Technical Report

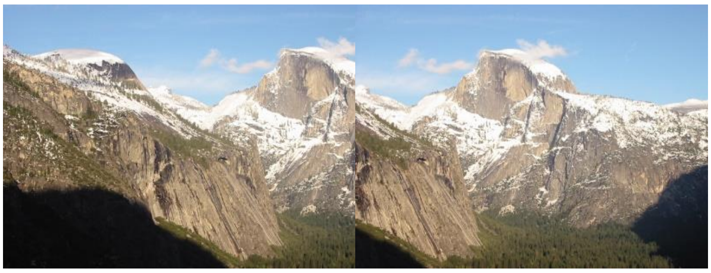
1829008 김민영

<1> Practice 1 : SIFT

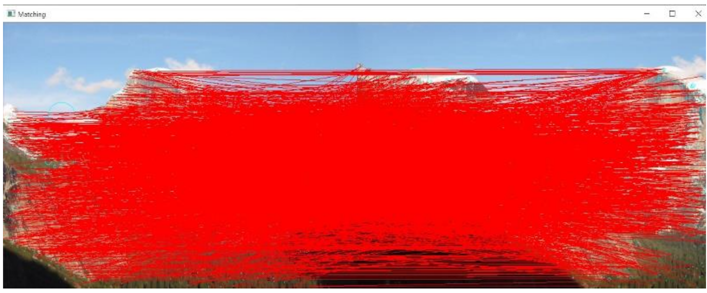
Run the SIFT descriptor using the function provided by OpenCV.

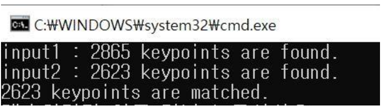
1. result image

INPUT) input1, input2



CASE1)

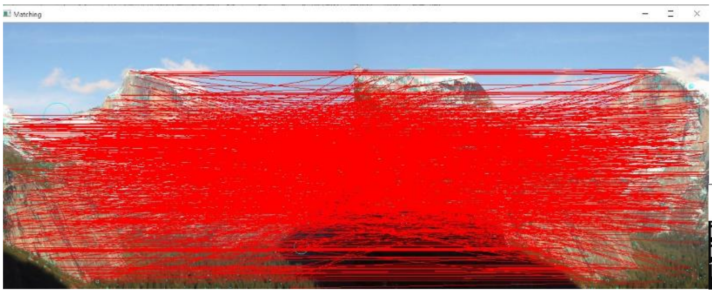


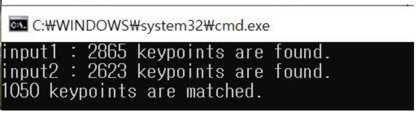


C:₩WINDOWS₩system32₩cmd.exe

```
input1 : 2865 keypoints are found.
input2 : 2623 keypoints are found.
2623 keypoints are matched.
```

CASE2)

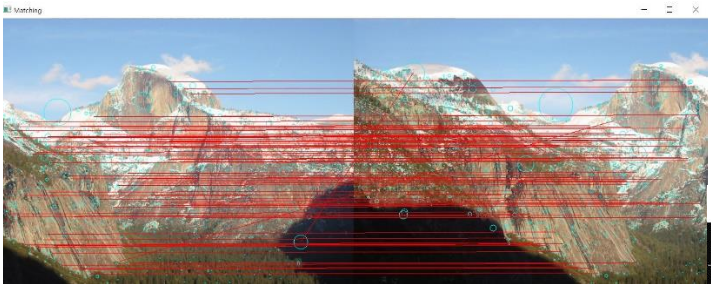


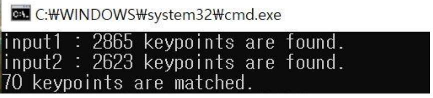


C:₩WINDOWS₩system32₩cmd.exe

```
input1 : 2865 keypoints are found.
input2 : 2623 keypoints are found.
1050 keypoints are matched.
```

CASE3)





C:₩WINDOWS₩system32₩cmd.exe

```
input1 : 2865 keypoints are found.
input2 : 2623 keypoints are found.
70 keypoints are matched.
```

2. Explanation of Code

Case1) `bool crossCheck = false; bool ratio_threshold = false;`

1. Run SIFT descriptor for two images I_1 and I_2.
2. Perform the feature matching using NN for $I_1 \rightarrow I_2$ and $I_2 \rightarrow I_1$
   Without using threshold T, show all the nearest neighbors for $I_1 \rightarrow I_2$ and $I_2 \rightarrow I_1$

Case2) `bool crossCheck = true; bool ratio_threshold = false;`

1. Run SIFT descriptor for two images I_1 and I_2.
2. Perform the feature matching using NN for $I_1 \rightarrow I_2$ and $I_2 \rightarrow I_1$
3. Refine the feature matching results using cross-checking for $I_1 \rightarrow I_2$ and $I_2 \rightarrow I_1$

Case3) `bool crossCheck = true; bool ratio_threshold = true;`

1. Run SIFT descriptor for two images.
2. Perform the feature matching using NN for $I_1 \rightarrow I_2$ and $I_2 \rightarrow I_1$
3. Refine the feature matching results using both cross-checking and ratio-based thresholding for $I_1 \rightarrow I_2$ and $I_2 \rightarrow I_1$

Main 함수 : siftFeatureDetector, SiftDescriptorExtractor를 만든다. Keypoints 와 descriptor를 출력 화면에 보여준다. Nearest neighbor pairs를 찾고, 그 쌍을 그려 matching image를 출력한다.

euclidDistance 함수 : 유클리디안 거리를 계산하고 이를 반환한다.

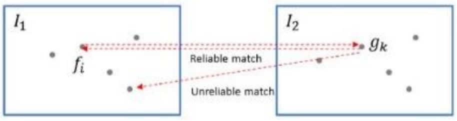
nearestNeighbor1, nearestNeighbor2 함수: Find the index of nearest neighbor point from keypoints.

findPairs 함수 : Find pairs of points with the smallest distace between them

Cross-checking) if i=l, the matching is assumed to be reliable. Otherwise, the matching is unreliable.

$$k = \min_j dist(f_i, g_j) \rightarrow NN(f_i) = g_k$$

$$l = \min_i dist(f_i, g_k) \rightarrow NN(g_k) = f_l$$



Threshold ratio)

$$k_1 = \min_j dist(f_i, g_j)$$

$$k_2 = \text{second} \min_j dist(f_i, g_j)$$

$$\frac{dist(f_i, g_{k_1})}{dist(f_i, g_{k_2})} < T_r \rightarrow NN(f_i) = g_{k_1}$$

3. Analysis

SIFT의 전체적인 과정

<detector>

1. Scale-Space Extrema detection
2. Key Point Localization

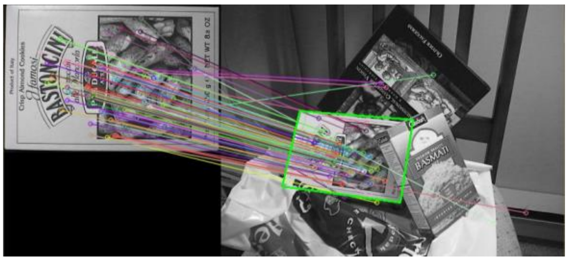
<descriptor>

3. Orientation Assignment
4. Key Point Descriptor Construction

<2> Practice2 : Simple Application

Run the following code and analyze what the code is doing in the technical report

➔ SURF 방법으로 feature matching을 수행한다.



1. SURF 를 이용하여 keypoints를 detect 하고 descriptors를 extract 한다.

```
int minHessian = 400;
Ptr<SURF> detector = SURF::create(minHessian);
std::vector<KeyPoint> keypoints_object, keypoints_scene;
Mat descriptors_object, descriptors_scene;
detector->detectAndCompute( img_object, Mat(), keypoints_object, descriptors_object);
detector->detectAndCompute( img_scene, Mat(), keypoints_scene, descriptors_scene);
```

2. FLANN matcher를 이용하여 descripteor vectors를 매칭시킨다.

```
FlannBasedMatcher matcher;
std::vector< DMatch > matches;
matcher.match(descriptors_object, descriptors_scene, matches);
double max_dist = 0; double min_dist = 100;
```

3. keypoints 사이의 최대, 최소 distance 계산

```
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;
}
```

4. good matches만 그리기(3*min_dist보다 작은)

```
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),
std::vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);
```

4. Localize the object

```
std::vector<Point2f> obj;
std::vector<Point2f> scene;
for (size_t 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, RANSAC);
```

5. Iamge1으로부터 corner 가져오기 (detect할 object)

```
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);
```

6. corner 사이에 선 그리기 (secne에서 mapped 된 object-image2)

```
 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);
```

7. Show matching image

```
 imshow("Good Matches & Object detection", img_matches);
```

Good Matches & Object detection 창은 image1 과 image2 를 붙여서 보여주며 매칭되는 포인트를 선으로 연결하고 찾은 object 를 초록색 사각형으로 그려준다.