[Computer Vision Programming] Lab 7. Fitting

전자공학과 21611591 김난희

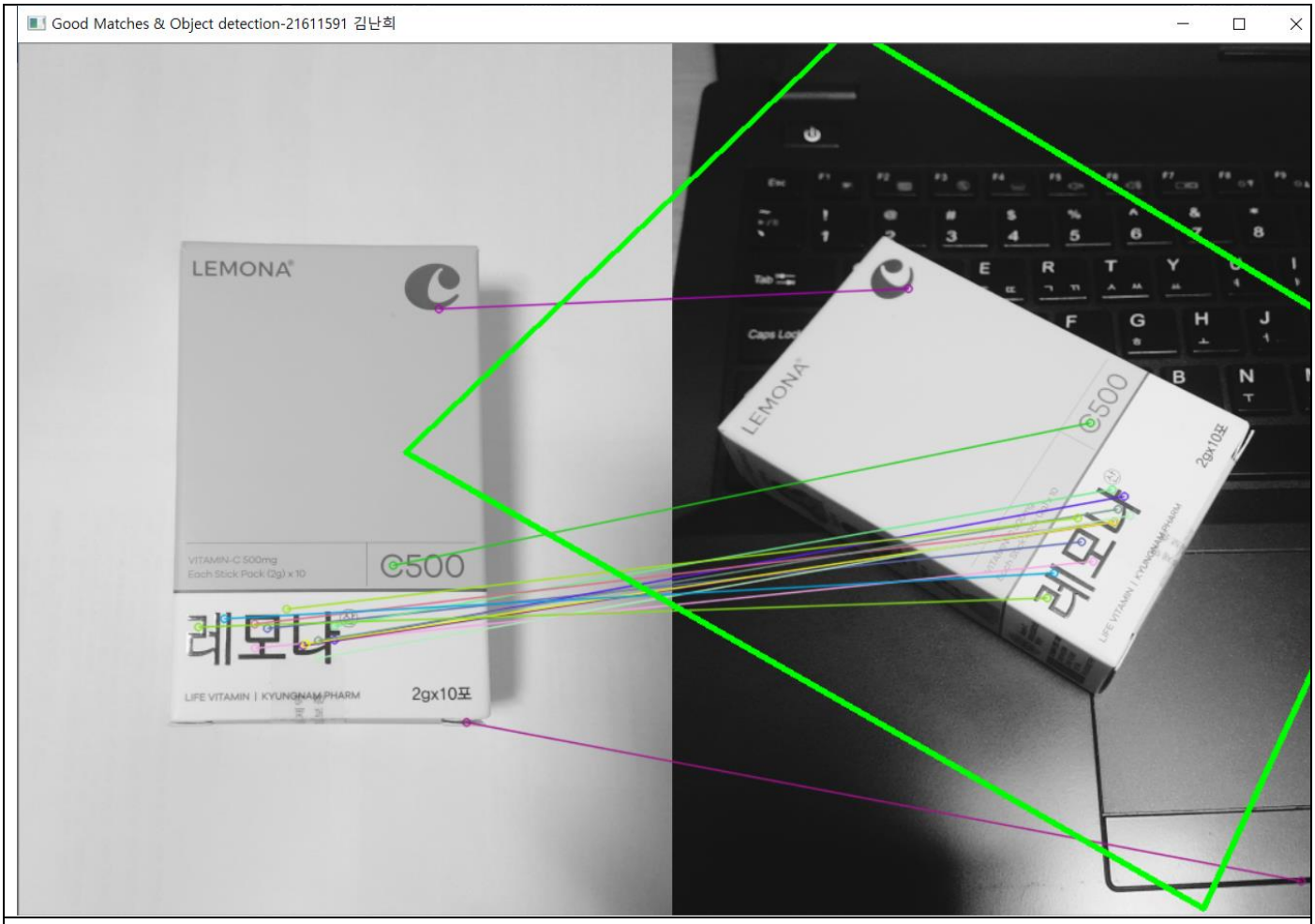
To do 1. 스마트폰으로 특정 물체를 다른 위치에서 2 장 촬영할 것.
To do 2. findHmography 에서 flag (0, CV_RANSAC, CV_LMEDS)의 의미 조사할 것.
To do 3. 각 flag 에 따른 매칭 특성 비교 분석할 것.

Lab 1. RANSAC-based Homography Estimation

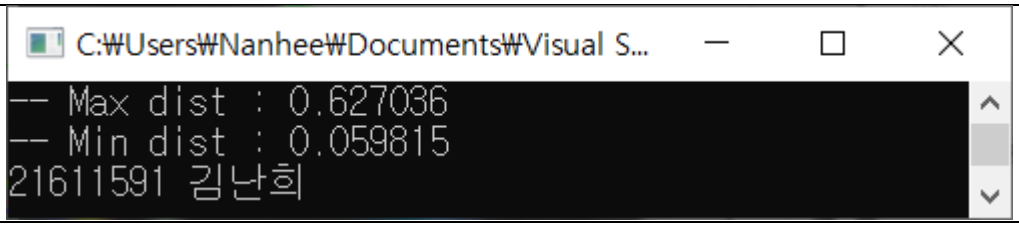
-결과 사진

Matching할 Object

matching할 object가 있는 배경



키보드부분에 잘 못 매칭된 것을 제외하고는, 매칭이 아주 잘되었다.
회전한 만큼과 함께 코너도 잘 그려졌다.



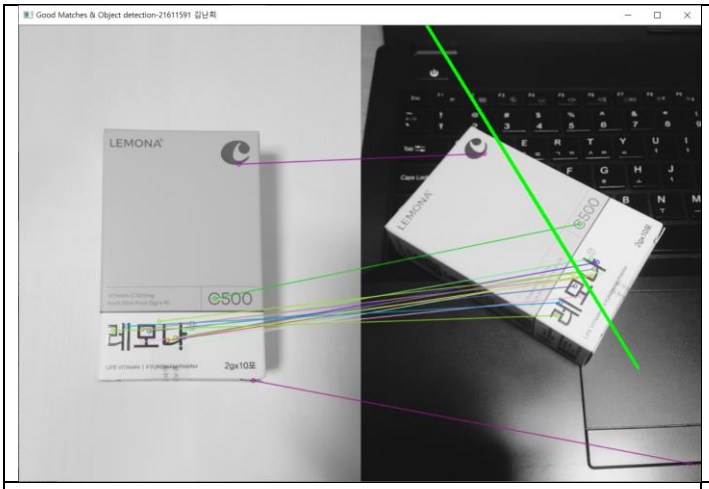
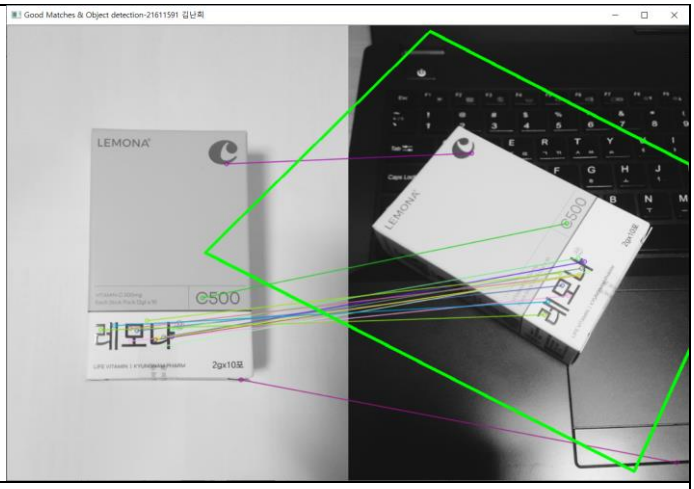
Min 거리 3배 이하에 들어오는 것만 매칭을 하여,
좋은 매칭만 뽑아낸다.

Lab 2. Homography Estimation using Other Method

-결과 사진1(findHomography function만 수정하였을 때)

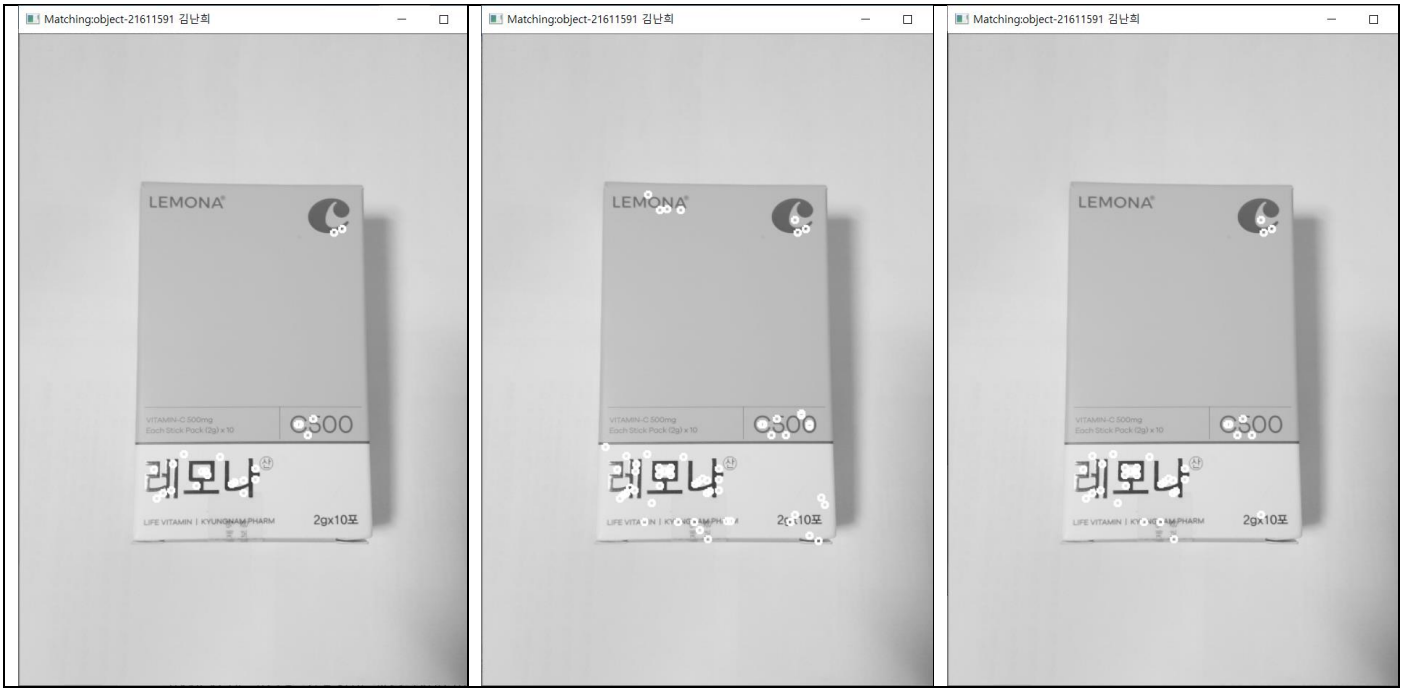
findHomography 함수의
Method에 0을 넣었을 때

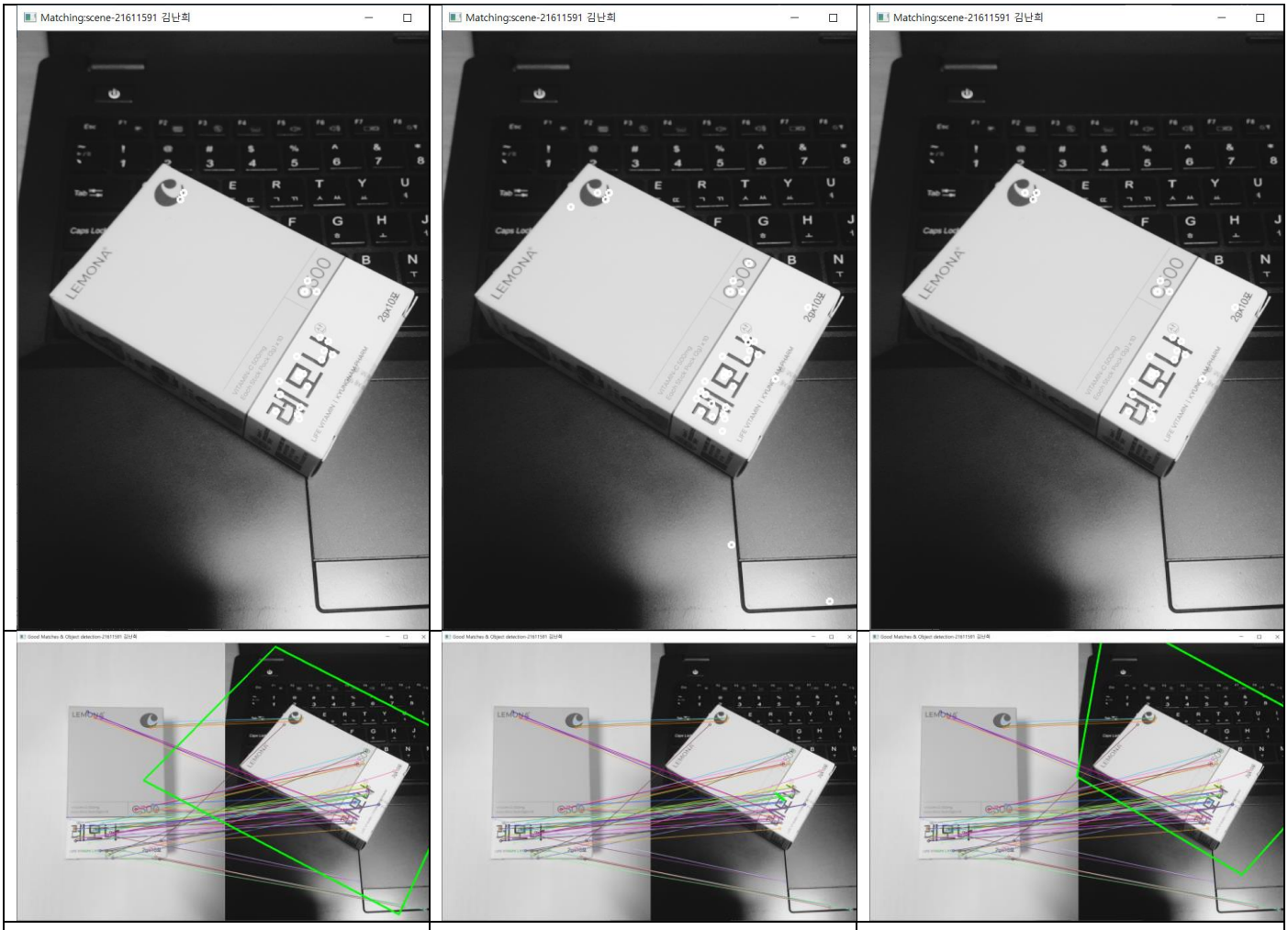
findHomography 함수의
Method에 LMEDS을 넣었을 때

	
findHomography 함수의 Method에 0을 넣었을 때	findHomography 함수의 Method에 LMEDS을 넣었을 때

-결과 사진2(findHomography function 수정 + 거리 이하를 3→5 수정하였을 때)

거리 이하를 5배까지 하여 좀더 느슨하게 테스트해보았다. 이론상으로 가장 좋은 결과가 나올 것은 3배이지만, 3 flags간의 큰 차이를 보기 위해 5배까지로 수정하였다.





RANSAC	0	LMEDS

fineHomography function의 Method의 0, RANSAC, LMEDS는 각각 무엇을 의미할까?

0	RANSAC	LMEDS
regular method using all the points (모든 점을 fitting edge로 잡는다.)	RANdom SAmple Consensus -based robust method (점 몇 개 fitting 잘되는 것만 선택한다.) RANSAC은 컨센서스가 최대인, 즉 가장 많은 수의 Data로부터 지지를 받는 모델을 선택하는 방법이다. RANSAC의 기준을 사용하면 관측 데이터에 이상한 애들(outlier)이 많더라도 데이터 근사가 가능하다는 점이다	Least-Median robust method (Error를 sorting하여 median을 찾아 점점 작아지도록 상위 몇 개만 선택한다.) 그렇기 때문에 Outlier가 50% 이하일때만 적용이 가능하다.

위의 조사한 바를 통해 결과 사진으로부터 매칭 특성을 해석해보자면,

1. Min dist, Max dist

우선 min dist와 max dist 값은 Homography 함수를 계산하기 전이어서 값이 3 flags 모두 같게 나온다.

2. Matching object와 scene 각각의 이미지

(결과 사진1) 거리 이하를 3배만 하여 좋은 매칭을 골라서 하였을 때는, 세 flags(0, RANSAC, LMEDS)간의 별다른 차이가 없었다. 0일 때 모든 점을 Matching하기 때문에 그림상에서 Object 밖의 Outlier를 하나 더 잡았다

(결과 사진2) 잡은 Inliers(물체 내부의 점) 개수는 0>LMEDS>RANSAC 으로 많았고, Outlier(물체 외부의 점) 개수는 0>LMEDS=RANSAC으로 표현되었다.

즉, 0은 모든 점을 fitting하기 때문에 LMEDS와 RANSAC보다 잡은 점의 수는 많은 것을 알 수 있었고, detect하기 위한 것으로도 정확성이 떨어지는 것을 볼 수 있다. RANSAC은 최상의 점만 선택하기 때문에 가장 적은 점을 선택했다. LMEDS는 Median의 결과로 나타나는 점이다. 그림상에서는 LMEDS와 RANSAC 모두 Outliers가 없었지만, 다른 이미지로 실험해본다면, 아마도 RANSAC이 LMEDS보다 적은 Outliers 수를 가져올 것이다.

3. Object detection 이미지

(참고)각각 matching detection된 알록달록한 선들은 findHomography function을 사용하기 전에 matching하며 미리 그린 것으로 동일하다. 그 후 findHomography function을 사용하여 각각의 object, scene 이미지에 circle을 그려주었고, 마지막으로 corners를 계산하여 그려주었다.

(결과 사진1) Object detection을 잡은 후에도, RANSAC과 LMEDS는 별다른 차이가 없었으나, 0는 모든 점을 잡았기에 corners가 일직선으로 표현되었다.

(결과 사진2) 결과사진1과 결론은 비슷하다. 0은 일직선으로 표현되었다. 점 모두를 fitting하기 위한 결과로 나타났다. RANSAC이 물체의 직사각형 모양에 가장 비슷하게 결과가 나왔고, LMEDS는 삐뚤어진 결과가 나왔다.

-소스 코드

```
#include <stdio.h> //매칭 점의 MAX, MIN 거리를 구함
#include <iostream> //vector 사용하기 위함
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/core/core.hpp"
#include <opencv2/nonfree/features2d.hpp> // for 2.4.13
#include "opencv2/calib3d/calib3d.hpp"

using namespace cv;
using namespace std;

int main(int argc, char** argv)
{
        Mat img_object = imread("LEMONA0.jpg", IMREAD_GRAYSCALE); //배경이 특징이 없고 똑바로 잘 선 이미지
        Mat img_scene = imread("LEMONA2.jpg", IMREAD_GRAYSCALE); //아무곳에서 변경된 각도로 둔 물체 이미지
```

```
	//이미지를 절반으로 줄인다.
	resize(img_object, img_object, Size(img_object.size().width / 2, img_object.size().height / 2));
	resize(img_scene, img_scene, Size(img_scene.size().width / 2, img_scene.size().height / 2));

	if (!img_object.data || !img_scene.data) //이미지 READ에 실패하면 뜨는 에러 체크
	{
		std::cout << " --(!) Error reading images " << std::endl; return -1;
	}

	//-- Step 1: Detect the keypoints using SURF Detector
	int minHessian = 400;
	SurfFeatureDetector detector(minHessian);
	vector<KeyPoint> keypoints_object, keypoints_scene;
	detector.detect(img_object, keypoints_object); //점 추출
	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); //key point 위치에 따라
descriptors 계산
	extractor.compute(img_scene, keypoints_scene, descriptors_scene);

	//-- Step 3: Matching descriptor vectors using FLANN matcher
	FlannBasedMatcher matcher;
	vector< DMatch > matches; //매칭된 점
	matcher.match(descriptors_object, descriptors_scene, matches); //매칭 함수
	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; //거리 max, min 찾음, 쌍으로 나옴
		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);
	printf("21611591 김난희 ₩n");

	//-- Draw only "good" matches (i.e. whose distance is less than 3*min_dist )
	vector< DMatch > good_matches; //거리 이하 들어오는 것만 뽑아낼 수 있음
	for (int i = 0; i < descriptors_object.rows; i++)
	{
		if (matches[i].distance < 5 * min_dist) //lab2에서 3 --> 5로 수정
		{
			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
	vector<Point2f> obj;
	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);
        }
        vector<uchar> inliers(obj.size(), 0);
        //fine a perspective transformation between two planes
        Mat H = findHomography(obj, scene, inliers, RANSAC);//RANSAC based homography estimation for lab 1
        //Mat H = findHomography(obj, scene, inliers, 0); //Other Method for lab2 : 0 or LMEDS

        // Draw the inlier points
        vector<Point2f>::const_iterator itPts = obj.begin(); //object 이미지에 원을 그리기 위함
        vector<uchar>::const_iterator itIn = inliers.begin();
        while (itPts != obj.end()) {
                // draw a circle at each inlier location
                if (*itIn)
                        circle(img_object, *itPts, 3, Scalar(0, 255, 0), 2); //색상이 더욱 잘 보이도록
변경함 → 원래 255, 255, 255로 흰색
                ++itPts;
                ++itIn;
        }
        itPts = scene.begin(); //scene 이미지에 원을 그리기 위함
        itIn = inliers.begin();
        while (itPts != scene.end()) {
                // draw a circle at each inlier location
                if (*itIn)
                        circle(img_scene, *itPts, 3, Scalar(0, 255, 0), 2); //색상이 더욱 잘 보이도록
변경함 → 원래 255, 255, 255로 흰색
                ++itPts;
                ++itIn;
        }
        imshow("Matching:object-21611591 김난희", img_object);
        imshow("Matching:scene-21611591 김난희", img_scene);

        //-- Get the corners from the image_1 ( the object to be "detected" )
        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);
        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] +
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-21611591 김난희", img_matches);

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

참고 사이트

https://darkpgmr.tistory.com/61