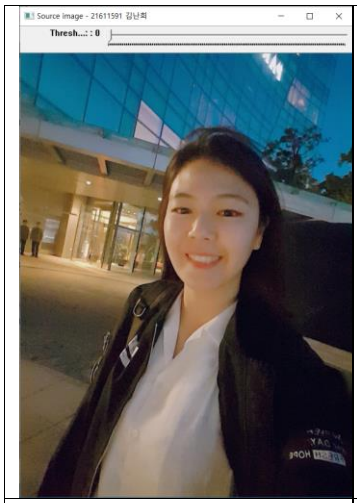
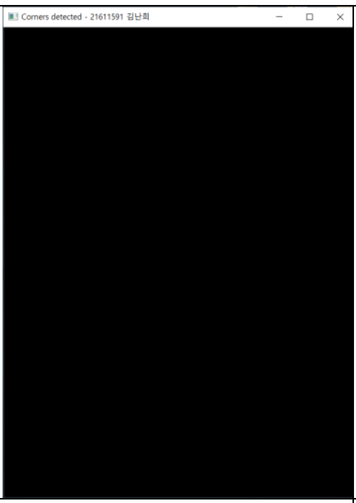
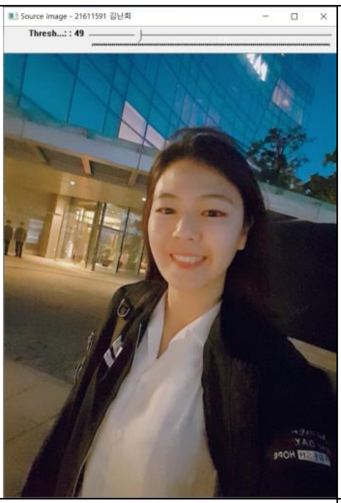
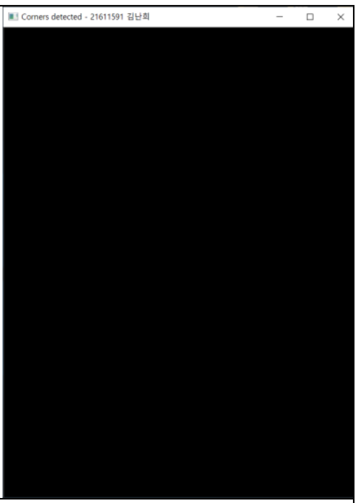
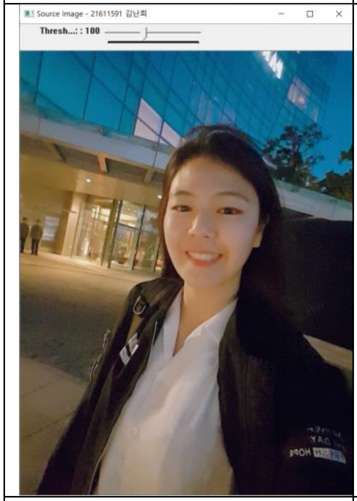
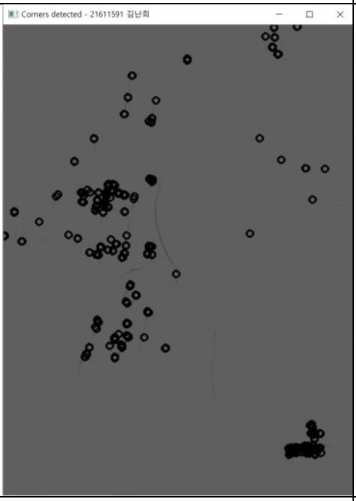
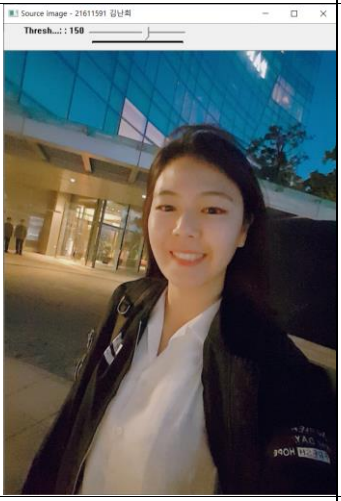
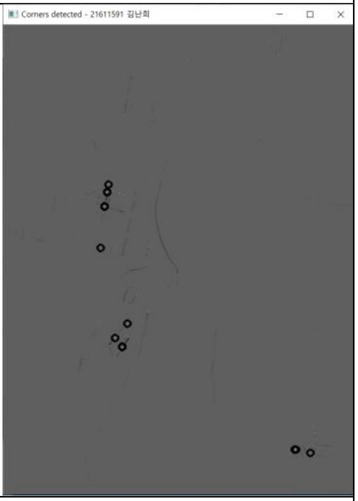
[Computer Vision Programming]

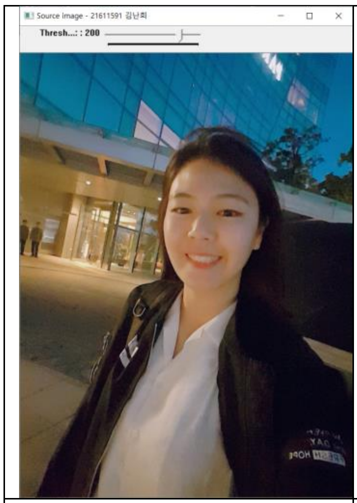
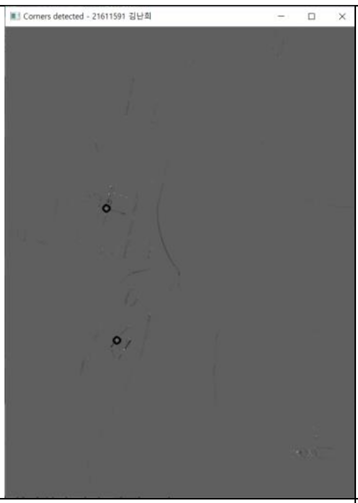
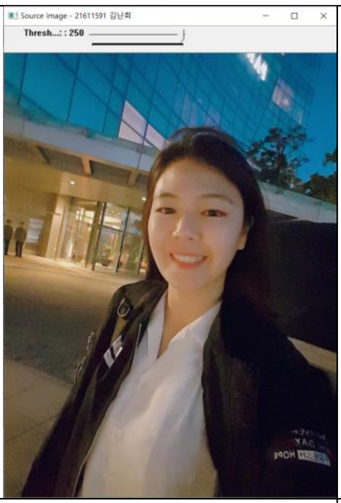
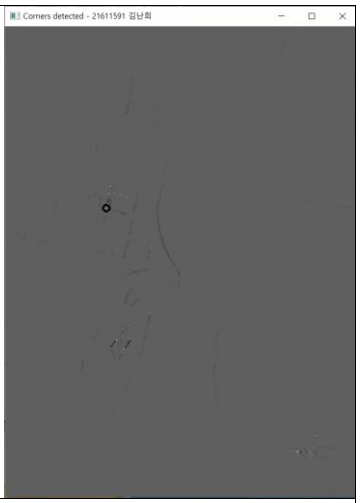
Lab 6. Corner and Blob detector

전자공학과 21611591 김난희

Lab 1. Harris Corner Detector

-결과 사진

			
Source Image : Threshold 0	Corners Detected Image(연산량이 너무 많아서 검은색 image가 뜸)	Source Image : Threshold 50	Corners Detected Image(연산량이 너무 많아서 검은색 image가 뜸)
			
Source Image : Threshold 100	Corners Detected Image	Source Image : Threshold 150	Corners Detected Image

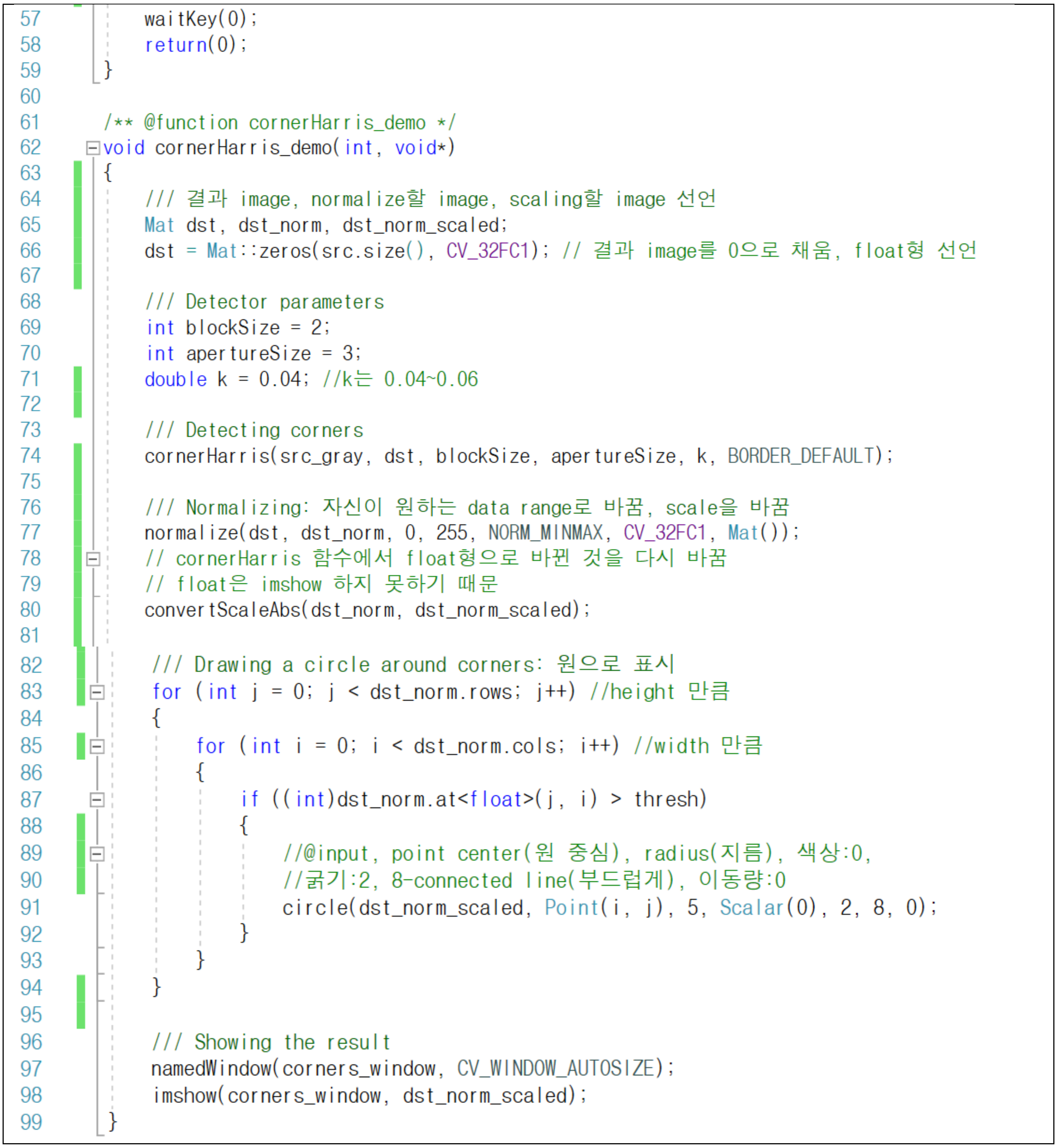
			
Source Image : Threshold 200	Corners Detected Image	Source Image : Threshold 250	Corners Detected Image

-Discursion

Threshold 값을 내릴수록 Detected Corners가 많아졌다. 100 이하가 되었을 때는 연산량이 너무 많아져 검은색 이미지로 표시되었다. 이는 모두 corner로 표시된 것인지, 아니면 오류가 난 것인지 알 수 없다. 상대적으로 corner가 적게 표시되는 image로 테스트를 따로 해보았을 때는 80이하부터 검은색 이미지로 표시되었다. Demo를 실행하지 않고 곧바로 threshold 값을 50으로 낮추어 imshow만 했을 때도 마찬가지였다.

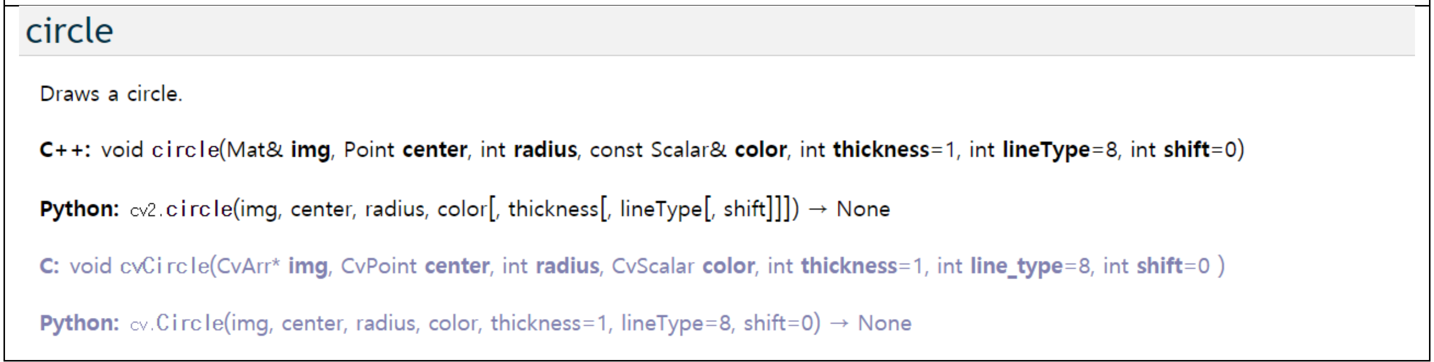
-소스 코드

```cpp
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
using namespace cv;

//***** Global variables for Lab.1
Mat src, src_gray; //원본, gray image
int thresh = 200, max_thresh = 255; //threshold
String source_window = "Source image - 21611591 김난희";
String corners_window = "Corners detected - 21611591 김난희";

/// Function header
void cornerHarris_demo(int, void*);
int main(void)
{
	//***** ***** Lab.1 Simple Canny Edge Detector ***** *****//
	/// Load source image, resize 1/2 of size and convert it to gray
	src = imread("myimg2.jpg");
	resize(src, src, Size(src.size().width/2, src.size().height / 2));
	cvtColor(src, src_gray, CV_BGR2GRAY);

	/// Create a window and a trackbar
	namedWindow(source_window, CV_WINDOW_AUTOSIZE);
	createTrackbar("Threshold: ", source_window, &thresh, max_thresh,cornerHarris_demo);

	/// Show image and keep running
	imshow(source_window, src);
	cornerHarris_demo(0, 0);
```



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

/** @function cornerHarris_demo */
void cornerHarris_demo(int, void*)
{
    /// 결과 image, normalize할 image, scaling할 image 선언
    Mat dst, dst_norm, dst_norm_scaled;
    dst = Mat::zeros(src.size(), CV_32FC1); // 결과 image를 0으로 채움, float형 선언

    /// Detector parameters
    int blockSize = 2;
    int apertureSize = 3;
    double k = 0.04; //k는 0.04~0.06

    /// Detecting corners
    cornerHarris(src_gray, dst, blockSize, apertureSize, k, BORDER_DEFAULT);

    /// Normalizing: 자신이 원하는 data range로 바꿈, scale을 바꿈
    normalize(dst, dst_norm, 0, 255, NORM_MINMAX, CV_32FC1, Mat());
    // cornerHarris 함수에서 float형으로 바뀐 것을 다시 바꿈
    // float은 imshow 하지 못하기 때문
    convertScaleAbs(dst_norm, dst_norm_scaled);

    /// Drawing a circle around corners: 원으로 표시
    for (int j = 0; j < dst_norm.rows; j++) //height 만큼
    {
        for (int i = 0; i < dst_norm.cols; i++) //width 만큼
        {
            if ((int)dst_norm.at<float>(j, i) > thresh)
            {
                //@input, point center(원 중심), radius(지름), 색상:0,
                //굵기:2, 8-connected line(부드럽게), 이동량:0
                circle(dst_norm_scaled, Point(i, j), 5, Scalar(0), 2, 8, 0);
            }
        }
    }

    /// Showing the result
    namedWindow(corners_window, CV_WINDOW_AUTOSIZE);
    imshow(corners_window, dst_norm_scaled);
}
```

<circle 함수>



circle

Draws a circle.

C++: void circle(Mat& **img**, Point **center**, int **radius**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, int **shift**=0)

Python: cv2.circle(img, center, radius, color[, thickness[, lineType[, shift]]]) → None

C: void cvCircle(CvArr* **img**, CvPoint **center**, int **radius**, CvScalar **color**, int **thickness**=1, int **line_type**=8, int **shift**=0)

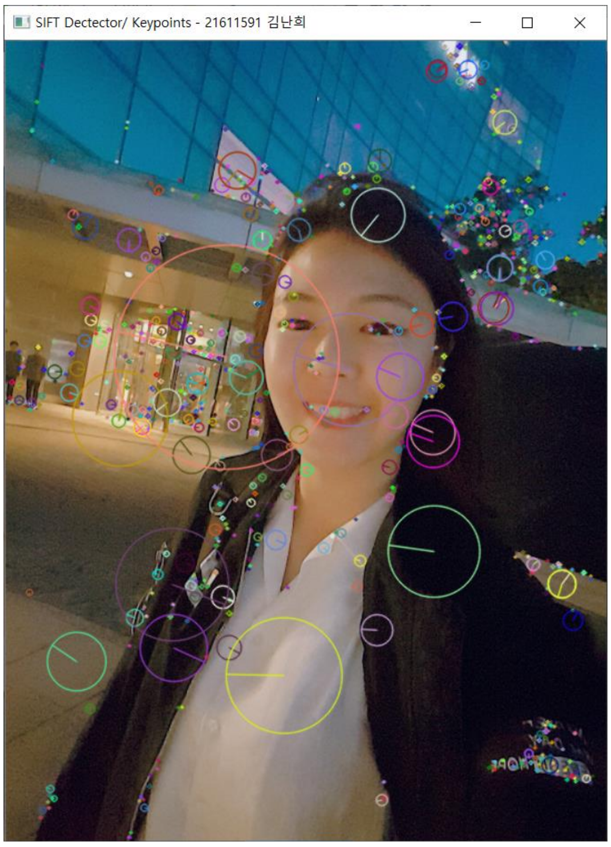
Python: cv.Circle(img, center, radius, color, thickness=1, lineType=8, shift=0) → None

Parameters:
- **img** – Image where the circle is drawn.
- **center** – Center of the circle.
- **radius** – Radius of the circle.
- **color** – Circle color.
- **thickness** – Thickness of the circle outline, if positive. Negative thickness means that a filled circle is to be drawn.
- **lineType** – Type of the circle boundary. See the `line()` description.
- **shift** – Number of fractional bits in the coordinates of the center and in the radius value.

The function `circle` draws a simple or filled circle with a given center and radius.

Lab 2. SIFT(blob) detector

-결과 사진



-Discursion

SIFT (Scale-Invariant Feature Transform)은 이미지의 크기와 회전에 불변하는 특징을 추출하는 알고리즘이다. 자세한 알고리즘 설명에 대해서는 참고 사이트에 작성해 놓았다.

특징들을 추출할 때, 원으로 크기와 방향으로 표시할 수 있었다. 어느 정도 edge를 따라 가며 keypoint들이 그려진 것을 확인할 수 있다.

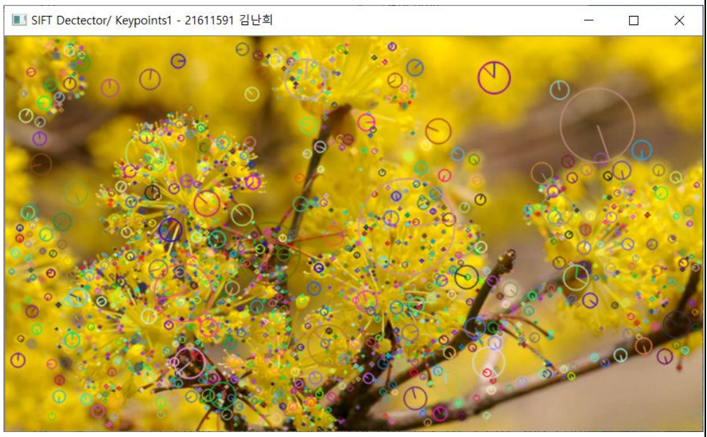
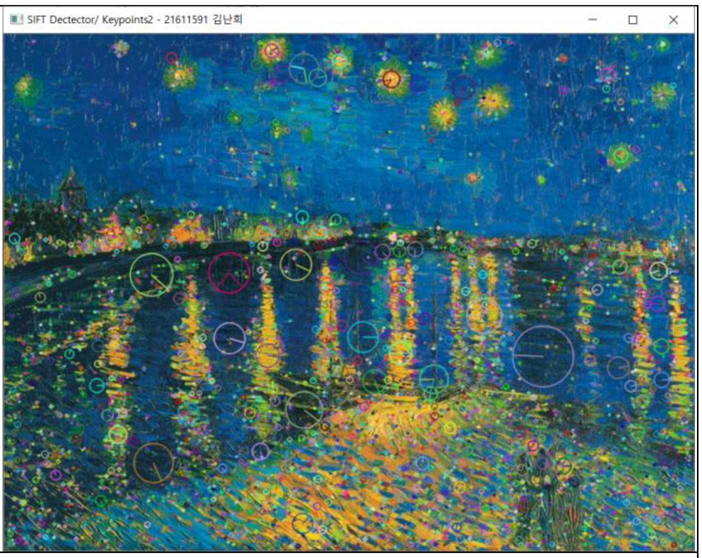
-소스 코드

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <opencv2/nonfree/features2d.hpp> // for Lab.2: SIFT Detector

using namespace cv;
```

Main문 내부

```cpp
	//***** ***** Lab.2 SIFT(blob) detector ***** *****//

	/// SIFT feature detector: SIFT 객체 생성
	SiftFeatureDetector detector;

	/// Feature detection: Keypoints 객체 생성
	vector<KeyPoint> keypoints;

	/// using detect func: image에서 keypoints를 계산하고 추출,
	/// 일부분의 keypoints 검출을 위해서는 mask 인자를 전달하면 됨.
	detector.detect(src, keypoints);

	/// draw Keypoints: image에 keypoints의 위치를 원으로 표시
	/// Scalar::all(-1)로 여러 색상으로 표시하며, DRAW_RICH_KEYPOINTS를 지정하면 크기와 방향성도 함께 표시
	drawKeypoints(src, keypoints, src, Scalar::all(-1), DrawMatchesFlags::DRAW_RICH_KEYPOINTS);

	/// Show detected (drawn) keypoints
	imshow("SIFT Dectector/ Keypoints - 21611591 김난희", src);

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

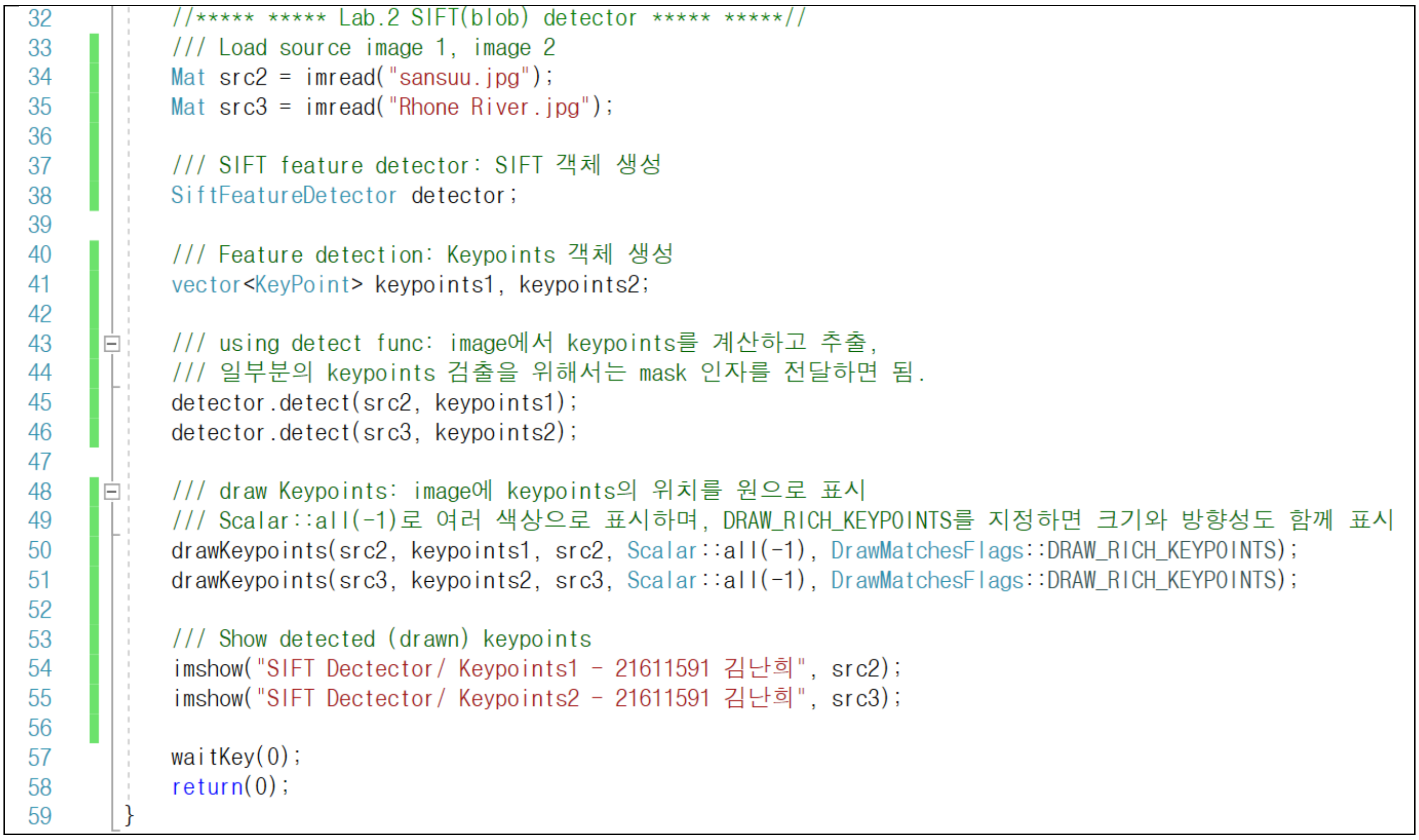
-결과 사진(추가 IMAGE)

	
(추가 IMAGE) 첫 번째 결과 사진: 산수유	(추가 IMAGE) 두 번째 결과 사진: Rhone River

-소스 코드(추가 IMAGE)

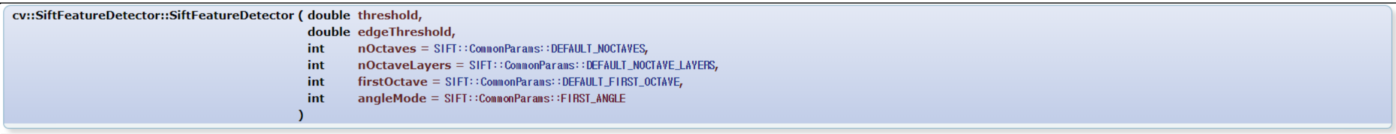
```cpp
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <opencv2/nonfree/features2d.hpp> // for Lab.2: SIFT Detector

using namespace cv;
```

Main문 내부

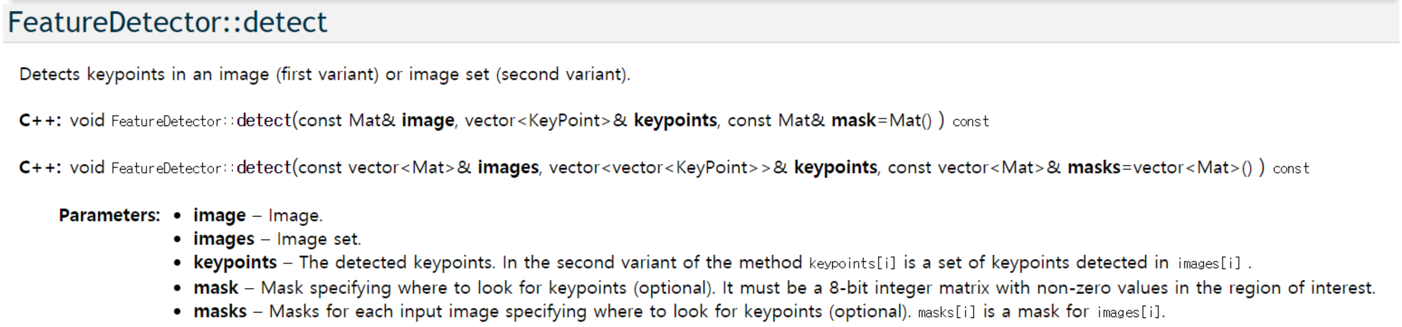


```
//***** ***** Lab.2 SIFT(blob) detector ***** *****//
/// Load source image 1, image 2
Mat src2 = imread("sansuu.jpg");
Mat src3 = imread("Rhone River.jpg");

/// SIFT feature detector: SIFT 객체 생성
SiftFeatureDetector detector;

/// Feature detection: Keypoints 객체 생성
vector<KeyPoint> keypoints1, keypoints2;

/// using detect func: image에서 keypoints를 계산하고 추출,
/// 일부분의 keypoints 검출을 위해서는 mask 인자를 전달하면 됨.
detector.detect(src2, keypoints1);
detector.detect(src3, keypoints2);

/// draw Keypoints: image에 keypoints의 위치를 원으로 표시
/// Scalar::all(-1)로 여러 색상으로 표시하며, DRAW_RICH_KEYPOINTS를 지정하면 크기와 방향성도 함께 표시
drawKeypoints(src2, keypoints1, src2, Scalar::all(-1), DrawMatchesFlags::DRAW_RICH_KEYPOINTS);
drawKeypoints(src3, keypoints2, src3, Scalar::all(-1), DrawMatchesFlags::DRAW_RICH_KEYPOINTS);

/// Show detected (drawn) keypoints
imshow("SIFT Dectector/ Keypoints1 - 21611591 김난희", src2);
imshow("SIFT Dectector/ Keypoints2 - 21611591 김난희", src3);

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

<Detector 객체 선언에 대한 것과 detect()함수>



```
cv::SiftFeatureDetector::SiftFeatureDetector ( double threshold,
                                               double edgeThreshold,
                                               int    nOctaves = SIFT::CommonParams::DEFAULT_NOCTAVES,
                                               int    nOctaveLayers = SIFT::CommonParams::DEFAULT_NOCTAVE_LAYERS,
                                               int    firstOctave = SIFT::CommonParams::DEFAULT_FIRST_OCTAVE,
                                               int    angleMode = SIFT::CommonParams::FIRST_ANGLE
                                             )
```



FeatureDetector::detect

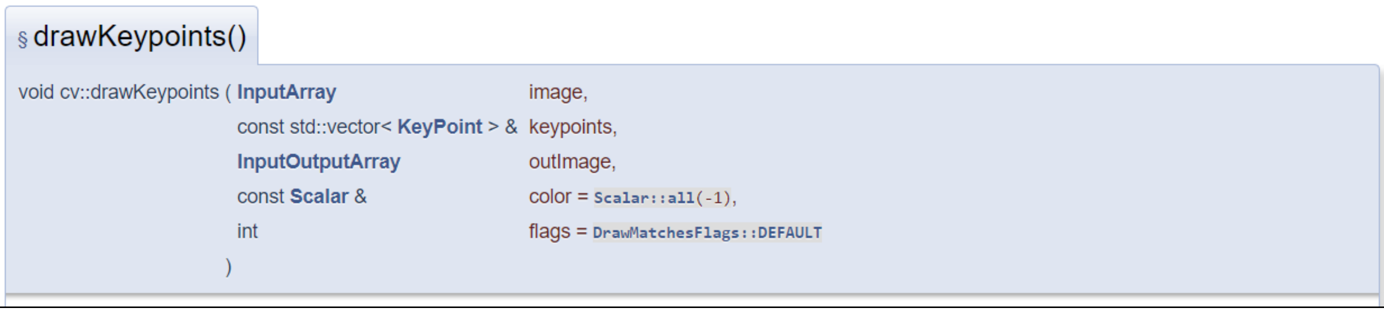
Detects keypoints in an image (first variant) or image set (second variant).

C++: void FeatureDetector::detect(const Mat& **image**, vector<KeyPoint>& **keypoints**, const Mat& **mask**=Mat()) const

C++: void FeatureDetector::detect(const vector<Mat>& **images**, vector<vector<KeyPoint> >& **keypoints**, const vector<Mat>& **masks**=vector<Mat>()) const

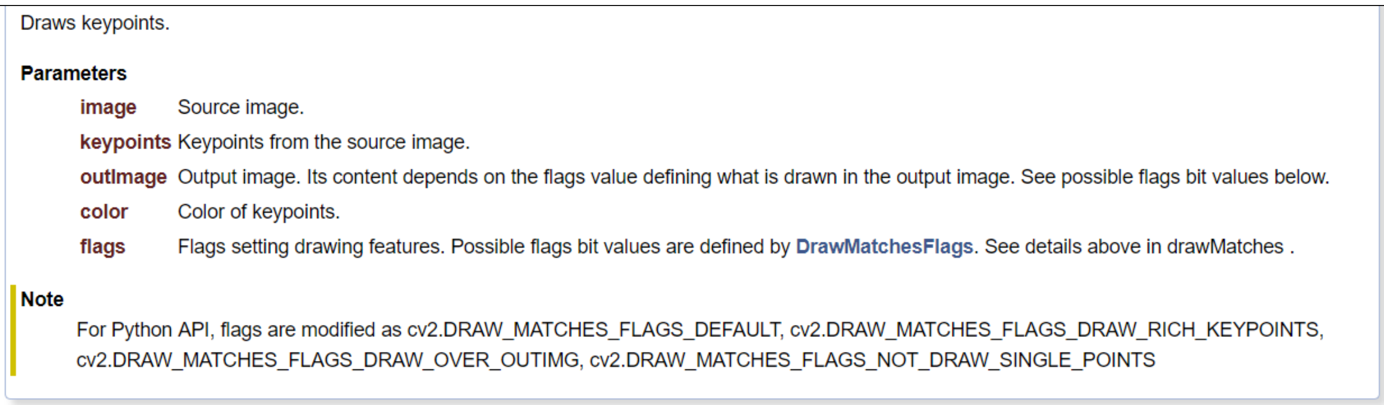
Parameters:
- **image** – Image.
- **images** – Image set.
- **keypoints** – The detected keypoints. In the second variant of the method keypoints[i] is a set of keypoints detected in images[i] .
- **mask** – Mask specifying where to look for keypoints (optional). It must be a 8-bit integer matrix with non-zero values in the region of interest.
- **masks** – Masks for each input image specifying where to look for keypoints (optional). masks[i] is a mask for images[i].

<drawKeypoints() 함수>



§ drawKeypoints()

```
void cv::drawKeypoints ( InputArray                      image,
                         const std::vector< KeyPoint > & keypoints,
                         InputOutputArray                outImage,
                         const Scalar &                  color = Scalar::all(-1),
                         int                             flags = DrawMatchesFlags::DEFAULT
                       )
```


Draws keypoints.

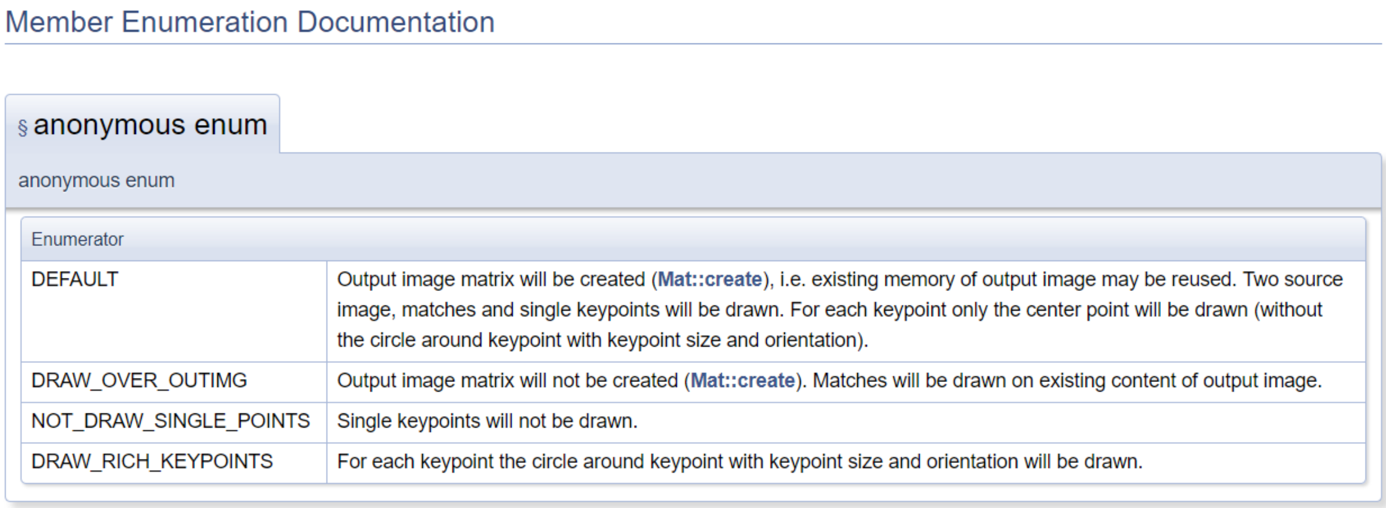
Parameters

image	Source image.
keypoints	Keypoints from the source image.
outImage	Output image. Its content depends on the flags value defining what is drawn in the output image. See possible flags bit values below.
color	Color of keypoints.
flags	Flags setting drawing features. Possible flags bit values are defined by **DrawMatchesFlags**. See details above in drawMatches .

Note

For Python API, flags are modified as cv2.DRAW_MATCHES_FLAGS_DEFAULT, cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS, cv2.DRAW_MATCHES_FLAGS_DRAW_OVER_OUTIMG, cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS

Flags parameter에 대한 자세한 설명


Member Enumeration Documentation

§ anonymous enum

anonymous enum

Enumerator	
DEFAULT	Output image matrix will be created (**Mat::create**), i.e. existing memory of output image may be reused. Two source image, matches and single keypoints will be drawn. For each keypoint only the center point will be drawn (without the circle around keypoint with keypoint size and orientation).
DRAW_OVER_OUTIMG	Output image matrix will not be created (**Mat::create**). Matches will be drawn on existing content of output image.
NOT_DRAW_SINGLE_POINTS	Single keypoints will not be drawn.
DRAW_RICH_KEYPOINTS	For each keypoint the circle around keypoint with keypoint size and orientation will be drawn.

전체 소스 코드

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <opencv2/nonfree/features2d.hpp> // for Lab.2: SIFT Detector

using namespace cv;

//***** Global variables for Lab.1
Mat src, src_gray; //원본, gray image
int thresh = 200, max_thresh = 255; //threshold
String source_window = "Source image - 21611591 김난희";
String corners_window = "Corners detected - 21611591 김난희";

/// Function header
void cornerHarris_demo(int, void*);

int main(void)
{
        //***** ***** Lab.1 Simple Canny Edge Detector ***** *****//
        /// Load source image, resize 1/2 of size and convert it to gray
        src = imread("myimg2.jpg");
        resize(src, src, Size(src.size().width/2, src.size().height / 2));
        cvtColor(src, src_gray, CV_BGR2GRAY);
        /// Create a window and a trackbar
        namedWindow(source_window, CV_WINDOW_AUTOSIZE);
        createTrackbar("Threshold: ", source_window, &thresh, max_thresh,cornerHarris_demo);

        /// Show image and keep running
        imshow(source_window, src);
        cornerHarris_demo(0, 0);

        //***** ***** Lab.2 SIFT(blob) detector ***** *****//

        /// SIFT feature detector: SIFT 객체 생성
        SiftFeatureDetector detector;

        /// Feature detection: Keypoints 객체 생성
        vector<KeyPoint> keypoints;

        /// using detect func: image에서 keypoints를 계산하고 추출,
        /// 일부분의 keypoints 검출을 위해서는 mask 인자를 전달하면 됨.
        detector.detect(src, keypoints);

        /// draw Keypoints: image에 keypoints의 위치를 원으로 표시
        /// Scalar::all(-1)로 여러 색상으로 표시하며, DRAW_RICH_KEYPOINTS를 지정하면 크기와 방향성도 함께
표시
        drawKeypoints(src, keypoints, src, Scalar::all(-1), DrawMatchesFlags::DRAW_RICH_KEYPOINTS);

        /// Show detected (drawn) keypoints
        imshow("SIFT Dectector/ Keypoints - 21611591 김난희", src);

        waitKey(0);
        return(0);
}

/** @function cornerHarris_demo */
void cornerHarris_demo(int, void*)
{
        /// 결과 image, normalize할 image, scaling할 image 선언
        Mat dst, dst_norm, dst_norm_scaled;
        dst = Mat::zeros(src.size(), CV_32FC1); // 결과 image를 0으로 채움, float형 선언

        /// Detector parameters
        int blockSize = 2;
        int apertureSize = 3;
        double k = 0.04; //k는 0.04~0.06

        /// Detecting corners
        cornerHarris(src_gray, dst, blockSize, apertureSize, k, BORDER_DEFAULT);

        /// Normalizing: 자신이 원하는 data range로 바꿈, scale을 바꿈
        normalize(dst, dst_norm, 0, 255, NORM_MINMAX, CV_32FC1, Mat());
        // cornerHarris 함수에서 float형으로 바뀐 것을 다시 바꿈
        // float은 imshow 하지 못하기 때문
        convertScaleAbs(dst_norm, dst_norm_scaled);

        /// Drawing a circle around corners: 원으로 표시
        for (int j = 0; j < dst_norm.rows; j++) //height 만큼
        {
                for (int i = 0; i < dst_norm.cols; i++) //width 만큼
                {
                        if ((int)dst_norm.at<float>(j, i) > thresh)
                        {
                                //@input, point center(원 중심), radius(지름), 색상:0,
                                //굵기:2, 8-connected line(부드럽게), 이동량:0
                        circle(dst_norm_scaled, Point(i, j), 5, Scalar(0), 2, 8, 0);
                    }
                }
            }

            /// Showing the result
            namedWindow(corners_window, CV_WINDOW_AUTOSIZE);
            imshow(corners_window, dst_norm_scaled);
                                            }
```

참고 사이트

(Lab1) Circle 함수 설명

https://docs.opencv.org/2.4/modules/core/doc/drawing_functions.html

(Lab2) SIFT 알고리즘 설명

https://bskyvision.com/21

(Lab2) SiftFeatureDetector 설명

https://physics.nyu.edu/grierlab/manuals/opencv/classcv_1_1SiftFeatureDetector.html

https://docs.opencv.org/2.4/modules/features2d/doc/common_interfaces_of_feature_detectors.html

(Lab2) SiftFeatureDetector 사용 예제

https://thinkpiece.tistory.com/246

(Lab2) drawKeypoints() 함수 설명

https://docs.opencv.org/3.3.1/d4/d5d/group__features2d__draw.html

http://blog.naver.com/PostView.nhn?blogId=samsjang&logNo=220643446825&parentCategoryNo=&categoryNo=66&viewDate=&isShowPopularPosts=false&from=postView