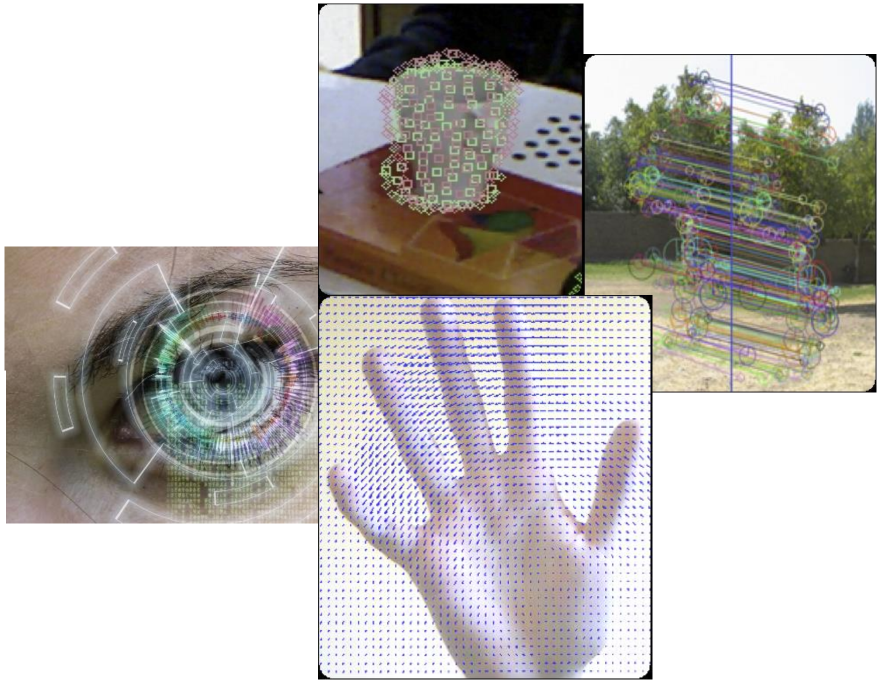
Intelligent Vision Processing Lab

2023, Fall

COMPUTER VISION 비젼 프로그래밍



Dept. of IT Engineering, Sookmyung Women's University

Prof. Byung-Gyu Kim

chap.9. Interesting point extraction and matching (관심점 검출 및 매칭)(Practice)

Basic Header Files for Exercises

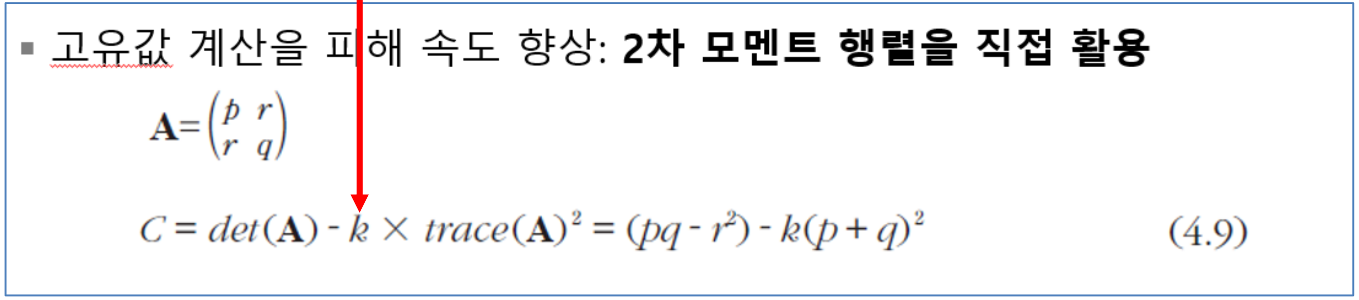
```
#include <stdio.h>
#include <iostream>
#include <stdio.h>
#include <iostream>
#include "opencv2/core.hpp"
#include "opencv2/opencv.hpp"
#include "opencv2/calib3d.hpp"
#include "opencv2/features2d.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/xfeatures2d.hpp "

using namespace std;
using namespace cv;
using namespace cv::xfeatures2d;
```

Harris Corner Extractor

- Harris Corner 연산자 API
  - void **cornerHarris**(InputArray **src**, OutputArray **dst**, int blockSize, int ksize, double k, int borderType=BORDER_DEFAULT)

Parameters:

- **src** – Input single-channel 8-bit or floating-point image.
- **dst** – Image to store the Harris detector responses. It has the type CV_32FC1 and the same size as src .
- **blockSize** – Neighborhood size
- **ksize** – Aperture parameter for the Sobel() operator.
- **k** – Harris detector free parameter. See the formula below.
- **borderType** – Pixel extrapolation method.



- 고유값 계산을 피해 속도 향상: **2차 모멘트 행렬을 직접 활용**

$$\mathbf{A}=\begin{pmatrix} p & r \\ r & q \end{pmatrix}$$

$$C = det(\mathbf{A}) - k \times trace(\mathbf{A})^2 = (pq - r^2) - k(p+q)^2 \qquad (4.9)$$

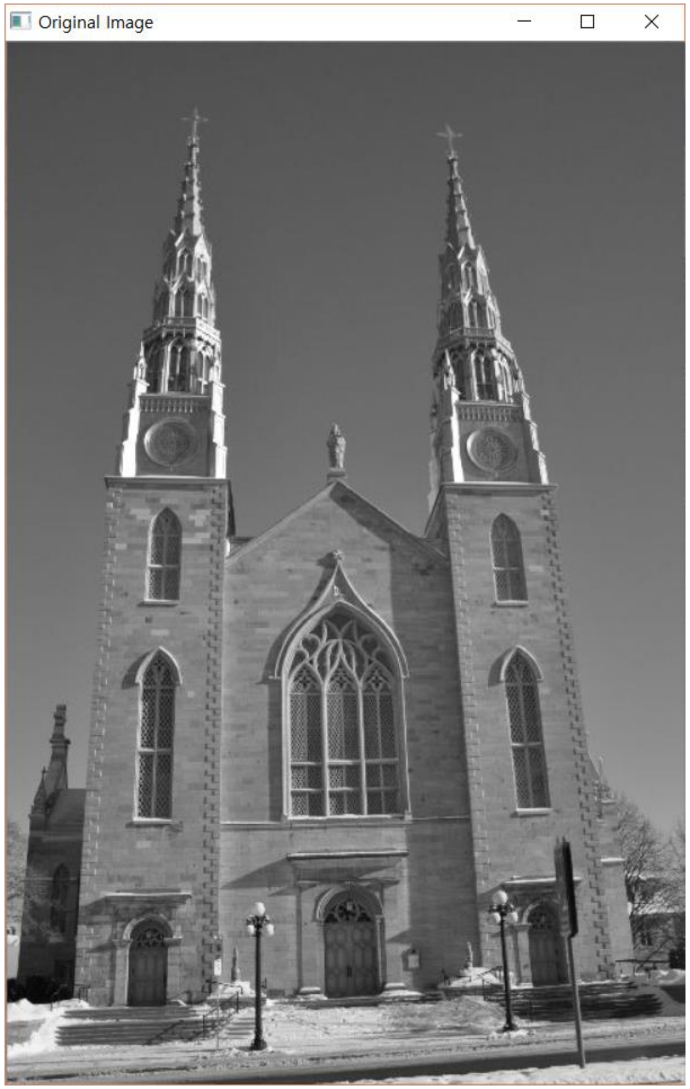
Harris Corner Extractor

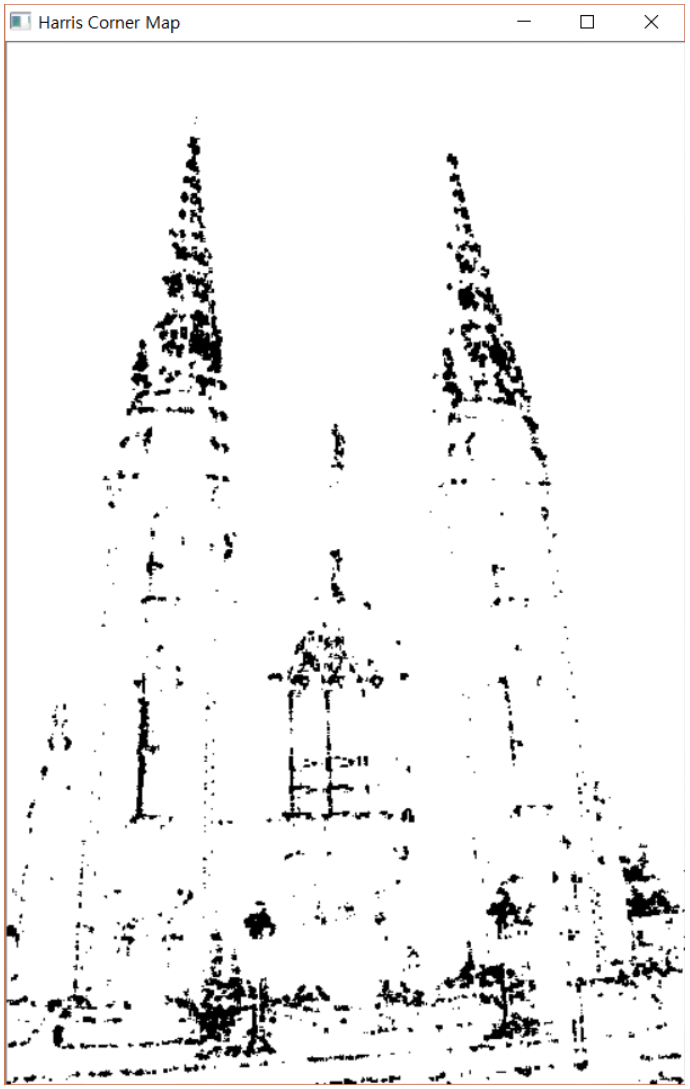
Harris Corner Extractor 연산자 기본 예제

```
int main()
{
        // Read input image
        cv::Mat image= cv::imread("church01.jpg",0);
        if (!image.data) return 0;

         // Display the image
        cv::namedWindow("Original Image");
        cv::imshow("Original Image",image);

        // Detect Harris Corners
        cv::Mat cornerStrength;
        cv::cornerHarris(image, cornerStrength,                              // Corner score 계산
                  3,      // neighborhood size
         3,      // aperture size
         0.01); // Harris parameter

        // threshold the corner strengths
        cv::Mat harrisCorners;
        double threshold= 0.0001;          // C: corner value
        cv::threshold(cornerStrength, harrisCorners,    threshold, 255, cv::THRESH_BINARY_INV); // 큰 Score만 선정

         // Display the corners
        cv::namedWindow("Harris Corner Map");
        cv::imshow("Harris Corner Map",harrisCorners);

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

Harris Corner Extractor

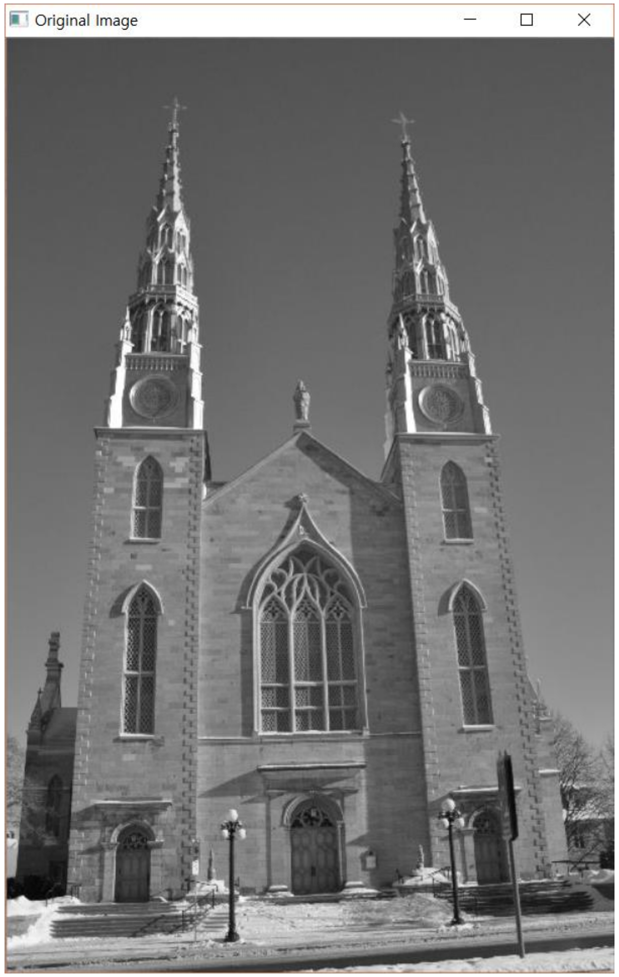
- 수행 결과:

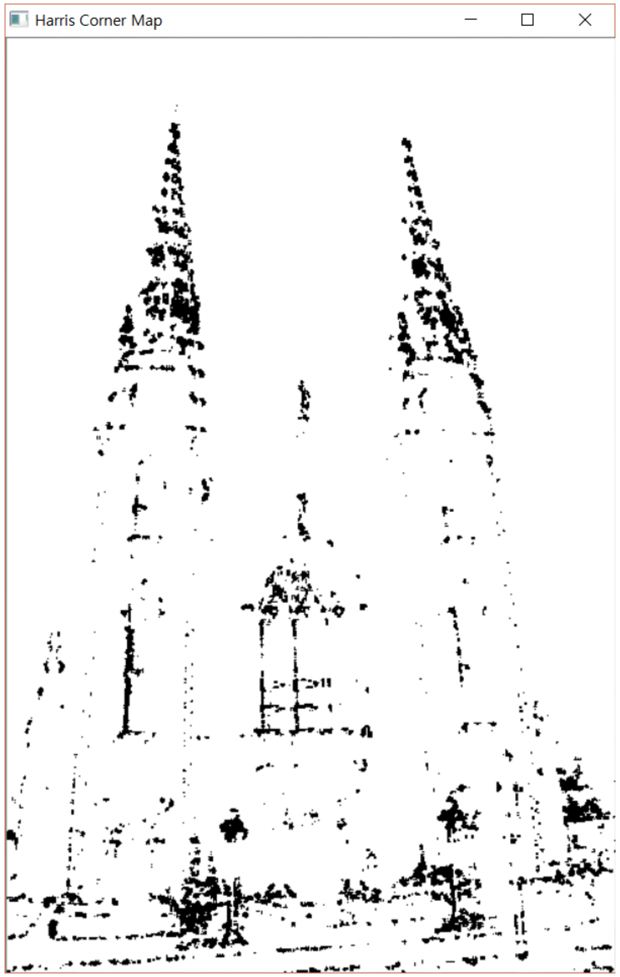


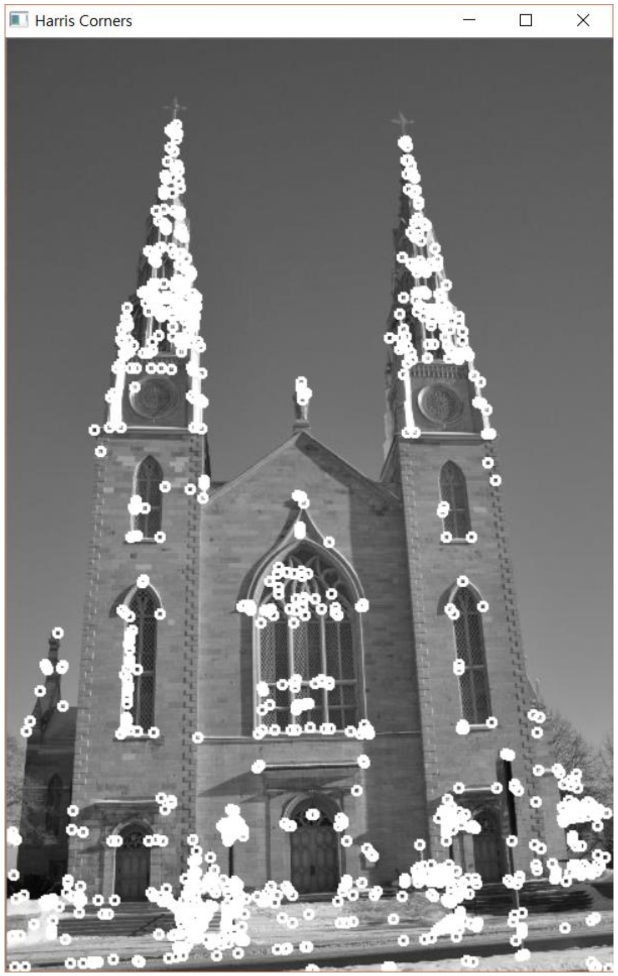


Harris Corner Extractor- Drawing harris corners

- 수행 결과:







Harris Corner Extractor - Keypoint 클래스의 활용

- goodFeaturesToTrack API 활용
  - Determines **strong corners** on an image.
  - void goodFeaturesToTrack(**image**, OutputArray **corners**, int **maxCorners**, double **qualityLevel**, double **minDistance**, InputArray **mask**=noArray(), int **blockSize**=3, bool **useHarrisDetector**=false, double **k**=0.04)

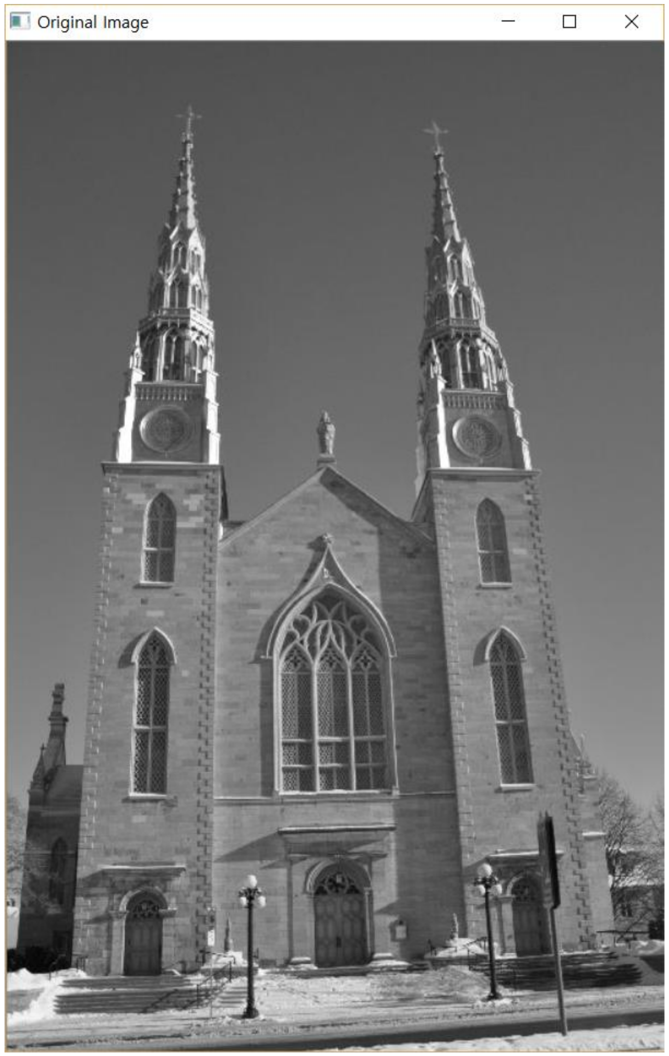
Parameters:	•**image** – Input 8-bit or floating-point 32-bit, single-channel image. •**eig_image** – The parameter is ignored. •**temp_image** – The parameter is ignored. •**corners** – Output vector of detected corners. •**maxCorners** – Maximum number of corners to return. If there are more corners than are found, the strongest of them is returned. •**qualityLevel** – Parameter characterizing the minimal accepted quality of image corners. The parameter value is multiplied by the best corner quality measure, which is the minimal eigenvalue (see cornerMinEigenVal()) or the Harris function response (see cornerHarris()). The corners with the quality measure less than the product are rejected. For example, if the best corner has the quality measure = 1500, and the qualityLevel=0.01 , then all the corners with the quality measure less than 15 are rejected. •**minDistance** – Minimum possible Euclidean distance between the returned corners. •**mask** – Optional region of interest. If the image is not empty (it needs to have the type CV_8UC1 and the same size as image), it specifies the region in which the corners are detected. •**blockSize** – Size of an average block for computing a derivative covariation matrix over each pixel neighborhood. See cornerEigenValsAndVecs() . •**useHarrisDetector** – Parameter indicating whether to use a Harris detector (see cornerHarris()) or cornerMinEigenVal(). •**k** – Free parameter of the Harris detector.

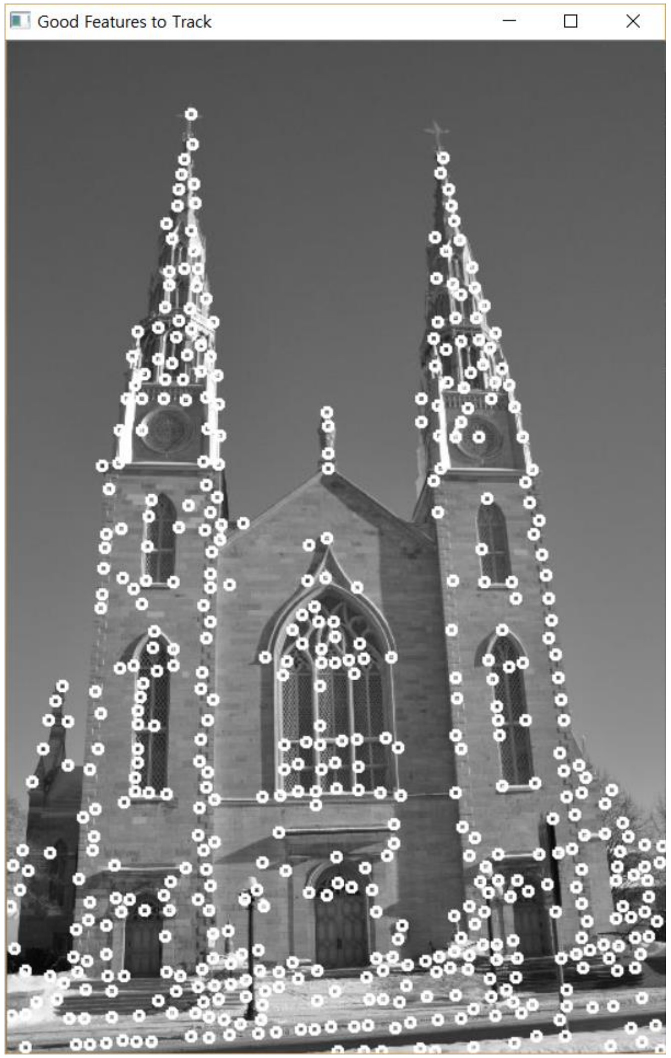
Harris Corner Extractor - Keypoint 클래스의 활용

```
int main()
{
          cv::Mat image= cv::imread("church01.jpg", 0);
          cv::namedWindow("Original Image");
          cv::imshow("Original Image",image);


         // Compute good features to track
         std::vector<cv::Point2f> corners;
         cv::goodFeaturesToTrack(image,corners,
         500,// maximum number of corners to be returned
         0.01,// quality level
         10);// minimum allowed distance between points

         // for all corners
         std::vector<cv::Point2f>::const_iterator it= corners.begin();
         while (it!=corners.end()) {

         // draw a circle at each corner location
         cv::circle(image,*it,3,cv::Scalar(255,255,255),2);
         ++it;
         }

             // Display the corners
         cv::namedWindow("Good Features to Track");
         cv::imshow("Good Features to Track",image);

             cv::waitKey(0);

     return 0;
}
```

Harris Corner Extractor - Keypoint 클래스의 활용

- 수행 결과: Harris corner 검출과 거의 동일





FAST(Feature From Accelerated Segment Test) 특징:실습

FAST API 활용법

- Detects corners using the FAST algorithm.
  - void **FAST**(InputArray **image**, vector<KeyPoint>& **keypoints**, int **threshold**, bool **nonmaxSuppression**=true)

Parameters:	•**image** – grayscale image where keypoints (corners) are detected. •**keypoints** – keypoints detected on the image. •**threshold** – threshold on difference between intensity of the central pixel and pixels of a circle around this pixel. •**nonmaxSuppression** – if true, non-maximum suppression is applied to det ected corners (keypoints). •**type** – one of the three neighborhoods as defined in the paper: FastFeatur eDetector::TYPE_9_16, FastFeatureDetector::TYPE_7_12, FastFeatureDetector:: TYPE_5_8

Harris Corner Extractor – Common Interface for Keypoints Detection

- FAST 알고리즘 활용법

```
//FAST Keypoint implementation for extraction
Ptr<FastFeatureDetector> detector = FastFeatureDetector::create(50, true);

// Compute keypoints and descriptor from the source image in advance
vector<KeyPoint> keypoints2;
Mat descriptors2;

// detecting and computing keypoints and descriptors
detector->detect(img_gray, keypoints2);

printf(" ==> original image:%d keypoints are found.₩n", (int)keypoints2.size());
```

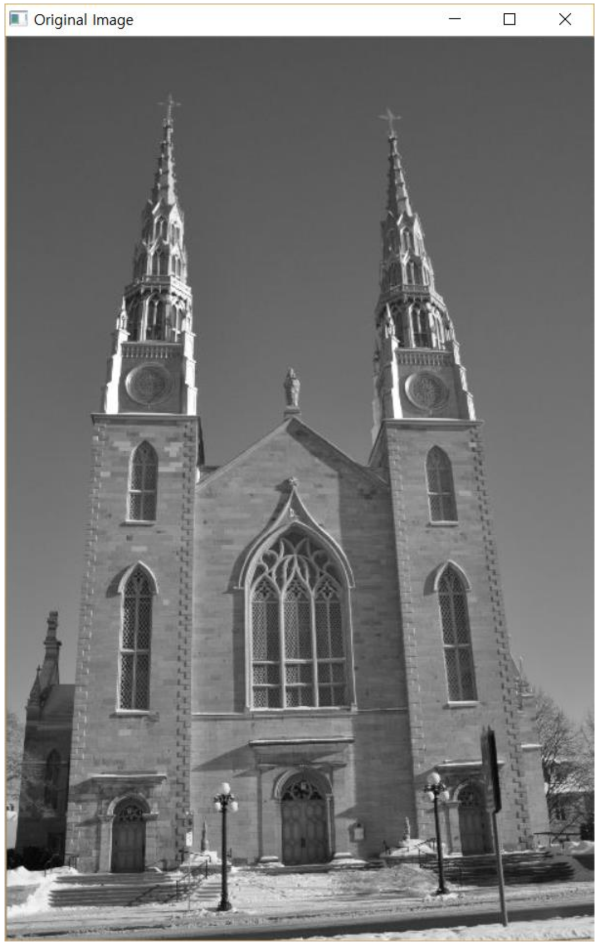
FAST(Feature From Accelerated Segment Test) 특징:실습

- FAST 알고리즘 활용 예제

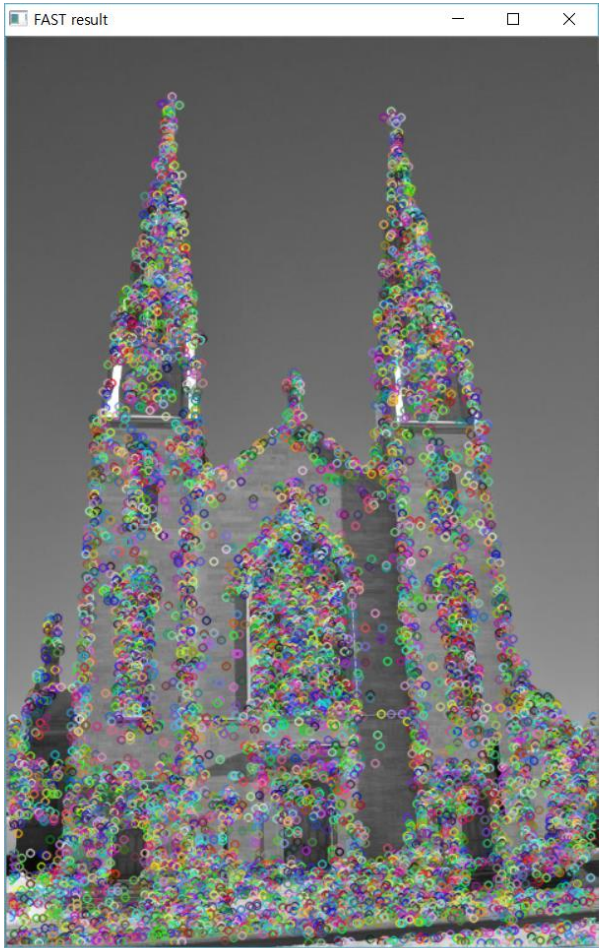
```
int main(int argc, char** argv){
        if (argc != 2){ readme(); return -1;
        }

        Mat img_object = imread(argv[1], IMREAD_COLOR);
        Mat img_gray;
        cvtColor(img_object,img_gray, COLOR_BGR2GRAY);

        //FAST Keypoint implementation for extraction
        Ptr<FastFeatureDetector> detector = FastFeatureDetector::create(50, true);

        // Compute keypoints and descriptor from the source image in advance
        vector<KeyPoint> keypoints2;
        Mat descriptors2;

        // detecting and computing keypoints and descriptors
        detector->detect(img_gray, keypoints2);
        printf(" ==> original image:%d keypoints are found.₩n", (int)keypoints2.size());

        for (int i = 0; i < keypoints2.size(); i++) {
                KeyPoint kp = keypoints2[i];
                circle(img_object, kp.pt, cvRound(kp.size*0.25), Scalar(255, 255, 0), 1, 8, 0);
        }

        namedWindow("FAST Keypoints");
        imshow("FAST Keypoints", img_object);

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

FAST(Feature From Accelerated Segment Test) 특징:실습

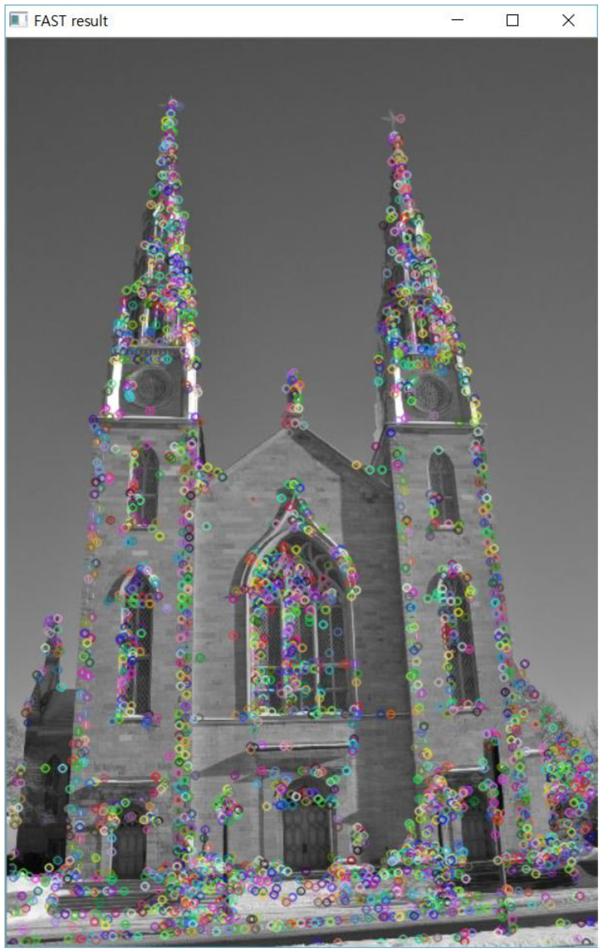
- 수행 결과: FAST corner 검출 결과



원본



Thres=10



Thres=30

SURF(Speeded-Up Robust Features) 특징:실습

- **SURF** API 활용법
  - SurfFeatureDetector 객체 활용
  - 사용법:

```
SurfFeatureDetector detector( minHessian );
```

  - ***Remind !!!!!***

$$C = det(\mathbf{H}) = d_{yy}(\sigma)\, d_{xx}(\sigma) - d_{yx}(\sigma)^2 \qquad (4.12)$$

where

$$d_{yy}(\sigma) = \frac{\partial^2}{\partial y^2}(G(\sigma) \circledast f) = \left(\frac{\partial^2}{\partial y^2} G(\sigma)\right) \circledast f \qquad (4.22)$$

⇩

행렬식을 빠르게 계산하기 위해, d_{yy}, d_{xx}, d_{yx}를 **9*9 마스크로 근사 계산**

SURF(Speeded-Up Robust Features) 특징:실습

SurfFeatureDetector *detector*(**minHessian**);

The **`minHessian`** is **a threshold** to decide from which value you are willing to accept keypoints.

- The higher the `minHessian`, the fewer keypoints you will obtain,
- On the other hand, the lower the `minHessian`, the more keypoints you get, but they may be more noisy.
  - In usual images, a value between 400 and 800 works well.

SURF(Speeded-Up Robust Features) 특징:실습

drawMatches 활용법

- **Draws the found matches of keypoints from two images.**
- void drawMatches(const Mat& **img1**, const vector<KeyPoint>& **keypoints1**, const Mat& **img2**, const vector<KeyPoint>& **keypoints2**, const vector<DMatch>& **matches1to2**, Mat& **outImg**, const Scalar& **matchColor**=Scalar::all(-1), const Scalar& **singlePointColor**=Scalar::all(-1), const vector<char>& **matchesMask**=vector<char>(), int **flags**=DrawMatchesFlags::DEFAULT)

Parameters:	•**img1** – First source image. •**keypoints1** – Keypoints from the first source image. •**img2** – Second source image. •**keypoints2** – Keypoints from the second source image. •**matches1to2** – Matches from the first image to the second one, which means that keypoints1[i] has a corresponding point in keypoints2[matches[i]] . •**outImg** – Output image. Its content depends on the flags value defining what is drawn in the output image. See possible flags bit values below. •**matchColor** – Color of matches (lines and connected keypoints). If matchColor==Scalar::all(-1) , the color is generated randomly. •**singlePointColor** – Color of single keypoints (circles), which means that keypoints do not have the matches. If singlePointColor==Scalar::all(-1) , the color is generated randomly. •**matchesMask** – Mask determining which matches are drawn. If the mask is empty, all matches are drawn. •**flags** – Flags setting drawing features. Possible flags bit values are defined by DrawMatchesFlags

SURF(Speeded-Up Robust Features) 특징:실습

- DrawMatchesFlags
  - DEFAULT = 0, : // Output image matrix will be created (Mat::create), // i.e. existing memory of output image may be reused. // Two source images, matches, and single keypoints // will be drawn. // For each keypoint, only the center point will be // drawn (without a circle around the keypoint with the // keypoint size and orientation).
  - DRAW_OVER_OUTIMG = 1, // Output image matrix will not be // created (using Mat::create). Matches will be drawn // on existing content of output image.
  - NOT_DRAW_SINGLE_POINTS = 2, // Single keypoints will not be drawn.
  - DRAW_RICH_KEYPOINTS = 4 // For each keypoint, the circle around // keypoint with keypoint size and orientation will // be drawn.

SURF(Speeded-Up Robust Features) 특징:실습

drawKeypoints 활용법

- Draws the found matches of keypoints from two images.
- void drawKeypoints(const Mat& **image**, const vector<KeyPoint>& **keypoints**, Mat& **outImage**, const Scalar& **color**=Scalar::all(-1), int **flags**=DrawMatchesFlags::DEFAULT)
-

Parameters:	•**image** – Source image. •**keypoints** – Keypoints from the source image. •**outImage** – Output image. Its content depends on the flags value definin g 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()

SURF(Speeded-Up Robust Features) 특징: 특징 추출 실습

- SURF 알고리즘 활용 예제

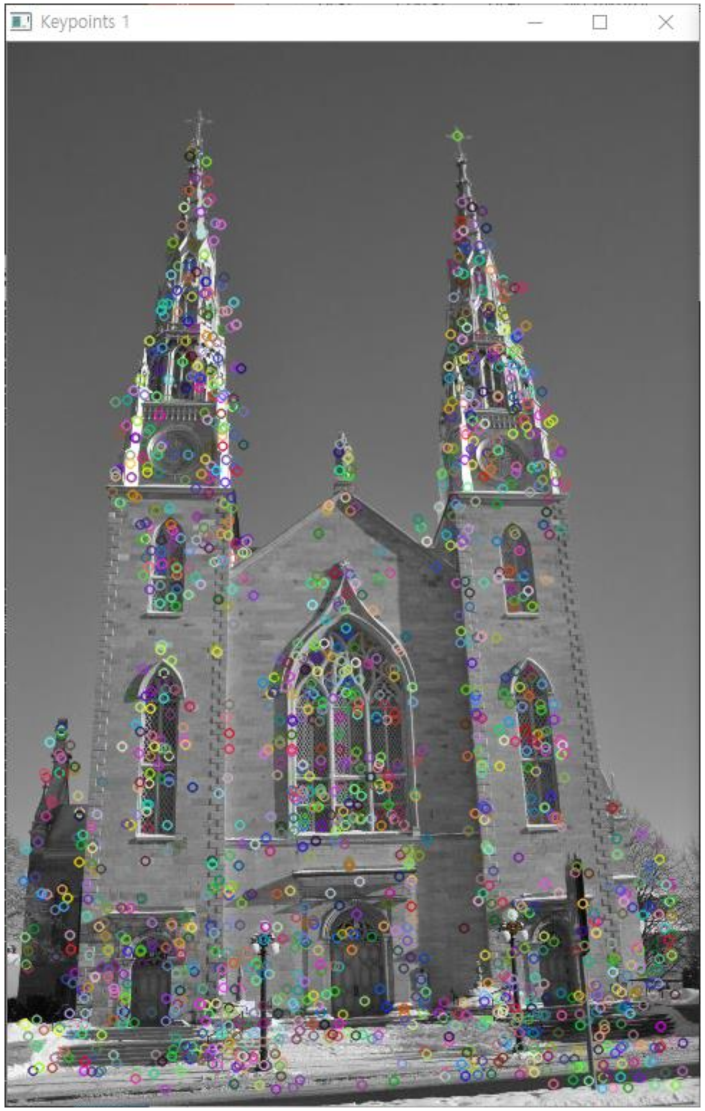
```
 #include <stdio.h>
#include <iostream>
#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
//#include "opencv2/nonfree/features2d.hpp“
#include "opencv2/xfeatures2d.hpp"
#include "opencv2/highgui/highgui.hpp"
//#include "opencv2/nonfree/nonfree.hpp"

using namespace cv;

/** @function main */
int main( int argc, char** argv )
{
        Mat img_1 = imread("C:/Temp/Images/church01.jpg",0);
        Mat img_2 = imread( "C:/Temp/Images/church02.jpg",0 );

        if( !img_1.data || !img_2.data )
        { std::cout<< " --(!) Error reading images " << std::endl; return -1; }
```
(계 속)

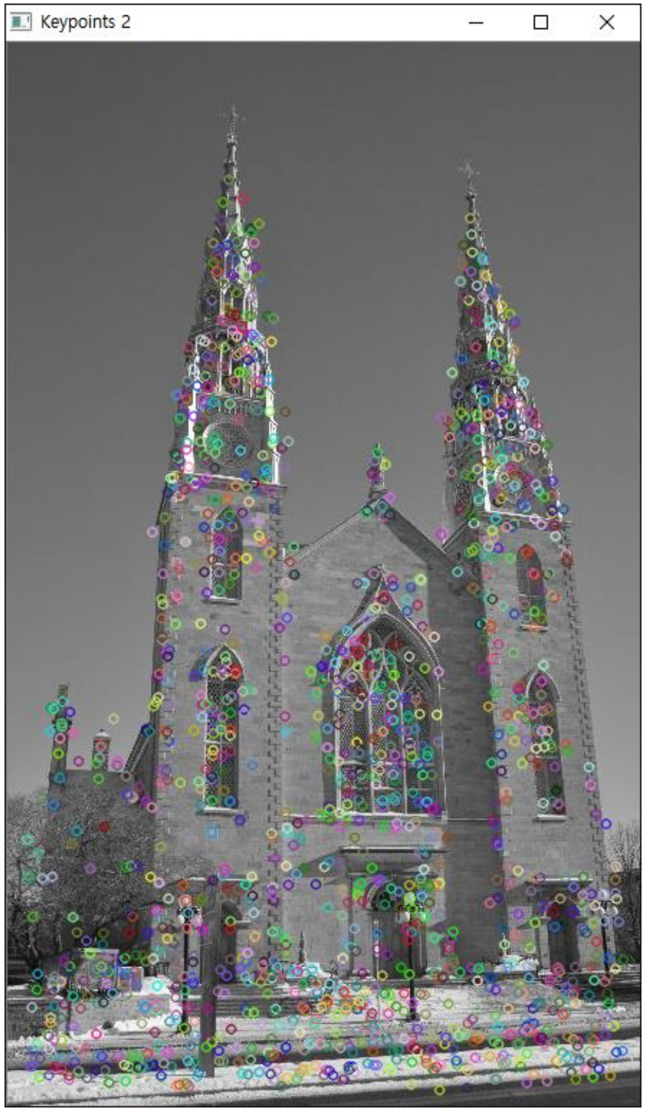
SURF(Speeded-Up Robust Features) 특징: 특징 추출 실습

```
if (argc != 3)
{
        readme(); return -1;
}

Mat img_1 = imread(argv[1], IMREAD_GRAYSCALE);
Mat img_2 = imread(argv[2], IMREAD_GRAYSCALE);

if (!img_1.data || !img_2.data)
{
        std::cout << " --(!) Error reading images " << std::endl; return -1;
}

//-- Step 1: Detect the keypoints using SURF Detector
int minHessian = 400;

Ptr<SURF> detector = SURF::create(minHessian);// detector(minHessian);

std::vector<KeyPoint> keypoints_1, keypoints_2;

detector->detect( img_1, keypoints_1);
detector->detect( img_2, keypoints_2);

//-- Draw keypoints
Mat img_keypoints_1; Mat img_keypoints_2;

drawKeypoints( img_1, keypoints_1, img_keypoints_1, Scalar::all(-1),
DrawMatchesFlags::DEFAULT);
drawKeypoints( img_2, keypoints_2, img_keypoints_2, Scalar::all(-1),
DrawMatchesFlags::DEFAULT);

//-- Show detected (drawn) keypoints
imshow("Keypoints 1", img_keypoints_1);
imshow("Keypoints 2", img_keypoints_2);
```

SURF(Speeded-Up Robust Features) 특징: 특징 추출 실습

- 수행 결과: SURF 특징 결과





SURF(Speeded-Up Robust Features) 특징: 매칭 실습

- **DescriptorExtractor 활용법**
  - *class* DescriptorExtractor
  - Abstract base class for computing descriptors for image keypoints.

```
class CV_EXPORTS DescriptorExtractor {
public:
	virtual ~DescriptorExtractor();
	void compute( const Mat& image, vector<KeyPoint>& keypoints, Mat& descriptors ) con
	void compute( const vector<Mat>& images, vector<vector<KeyPoint> >
				& keypoints, vector<Mat>& descriptors ) const;
	virtual void read( const FileNode& );
	virtual void write( FileStorage& ) const;
	virtual int descriptorSize() const = 0;
	virtual int descriptorType() const = 0;
	static Ptr<DescriptorExtractor> create( const string& descriptorExtractorType );
protected: ...
};
```

SURF(Speeded-Up Robust Features) 특징: 매칭 실습

BFMatcher 활용법

- *class* **BFMatcher**
- **BFMatcher::BFMatcher**(int **normType**=NORM_L2, bool **crossCheck**=false)
- **Brute-force** matcher constructor.

  → 완전 탐색 알고리즘

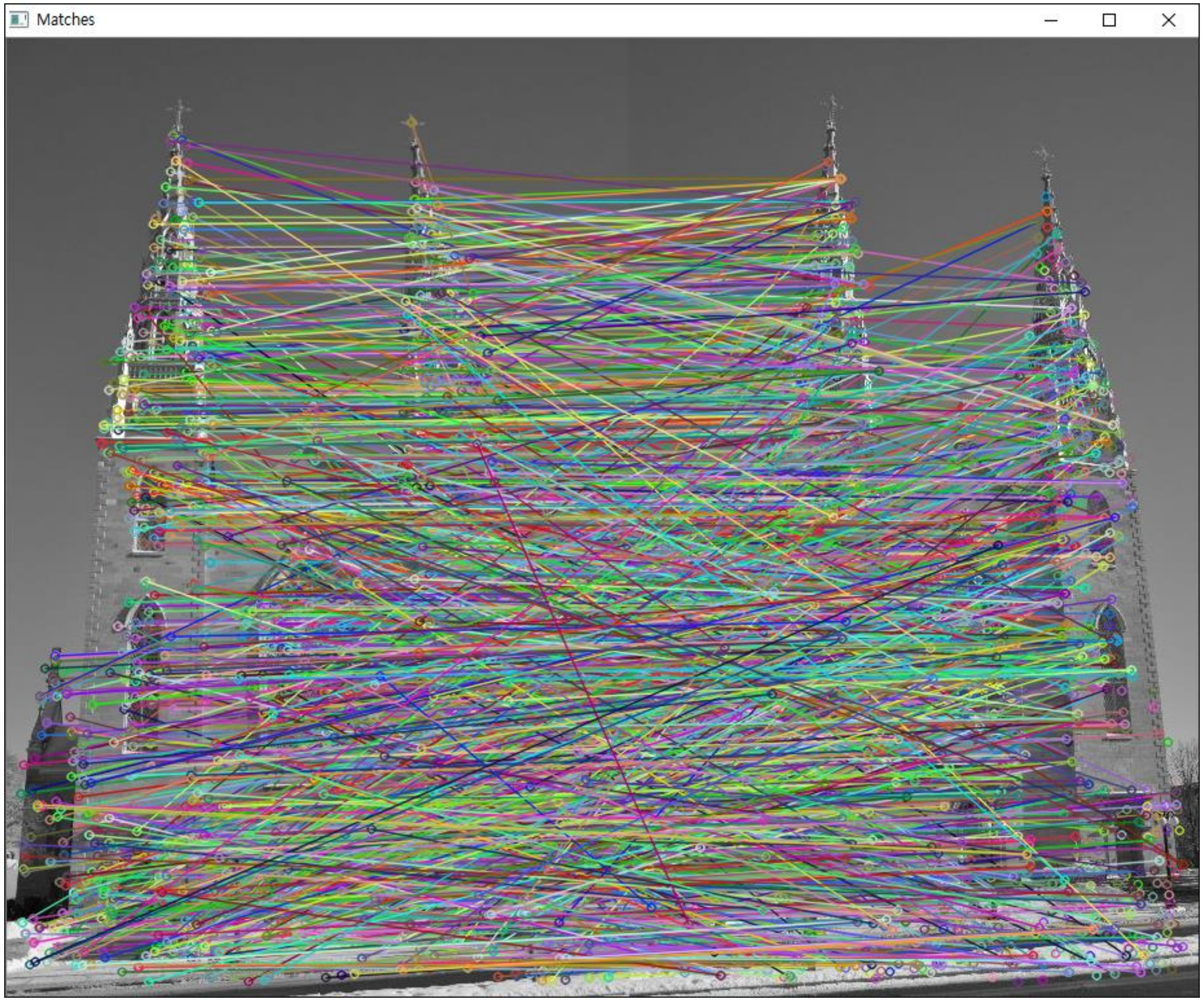
Parameters:	•**normType** – One of NORM_L1, NORM_L2, NORM_HAMMING, NORM_HAMMING2. L1 and L2 norms are preferable choices for SIFT and SURF descriptors, NORM_HAMMING should be used with ORB, BRISK and BRIEF, NORM_HAMMING2 should be used with ORB when WTA_K==3 or 4 (see ORB::ORB constructor description). •**crossCheck** – If it is false, this is will be default BFMatcher behaviour when it finds the k nearest neighbors for each query descriptor. **If crossCheck==true, then the knnMatch() method with k=1 will only return pairs (i,j) such that for i-th query descriptor the j-th descriptor in the matcher's collection is the nearest and vice versa,** i.e. the BFMatcher will only return consistent pairs. Such technique usually produces best results with minimal number of outliers when there are enough matches. This is alternative to the ratio test, used by D. Lowe in SIFT paper.

SURF(Speeded-Up Robust Features) 특징: 매칭 실습

```
 // Construction of the SURF descriptor extractor
Ptr<SURF> extractor = SURF::create();

// Extraction of the SURF descriptors
cv::Mat descriptors1, descriptors2;
extractor->compute(img_1, keypoints_1, descriptors1);
extractor->compute(img_2, keypoints_2, descriptors2);
std::cout << "descriptor matrix size: " << descriptors1.rows << " by "
                                                  << descriptors1.cols << std::endl;
// Construction of the matcher
cv::BFMatcher matcher( cv::NORM_L2, false );

// Match the two image descriptors
std::vector<cv::DMatch> matches;
matcher.match(descriptors1,descriptors2, matches);
std::cout << "Number of matched points: " << matches.size() << std::endl;

//-- Draw matches---//
 cv::Mat img_matches;
 cv::drawMatches( img_1, keypoints_1, img_2, keypoints_2, matches, img_matches );

//-- Show detected matches
imshow("Matches", img_matches );

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

SURF(Speeded-Up Robust Features) 특징: 매칭 실습

- 수행 결과: SURF 특징 매칭 결과



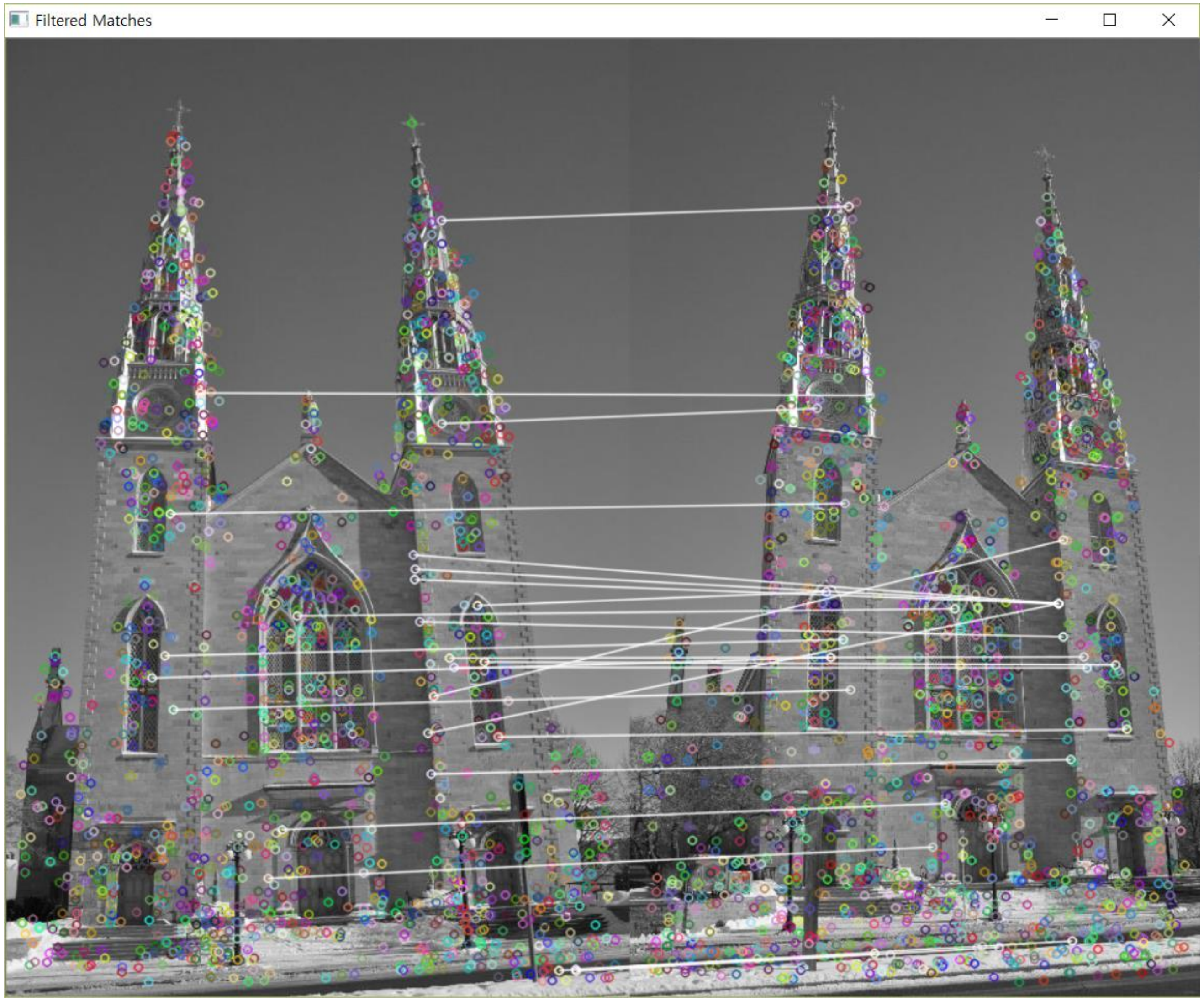
SURF(Speeded-Up Robust Features) 특징: 필터링+매칭 실습

- 매칭 후 Filtering 샘플 코드

```
   (Matching process 후)

 //--- Filtering loop ---//
std::nth_element(matches.begin(),     // initial position
                    matches.begin()+24, // position of the sorted element
                    matches.end());      // end position
// remove all elements after the 25th
matches.erase(matches.begin()+25, matches.end());

cv::Mat imageMatches1;
cv::drawMatches(img_1,keypoints_1,  // 1st image and its keypoints
                    img_2,keypoints_2,  // 2nd image and its keypoints
                    matches,// the matches
                    imageMatches1,// the image produced
                   cv::Scalar(255,255,255)); // color of the lines
cv::namedWindow("Filtered Matches");
cv::imshow("Filtered Matches",imageMatches1);

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

SURF(Speeded-Up Robust Features) 특징: 필터링+매칭 실습

- 25개로 제한하여 중요 특징 매칭한 결과



SIFT(Scale-invariant feature transform) 특징:실습

- **SIFT** API 활용법
  - *class* SIFT : *public* Feature2D.
    - Class for extracting keypoints and computing descriptors using the Scale Invariant Feature Transform (SIFT) algorithm by D. Lowe [Lowe04].
    - Constructor
      - SIFT::SIFT(int **nfeatures**=0, int **nOctaveLayers**=3, double **contrastThreshold**=0.04, double **edgeThreshold**=10, double **sigma**=1.6)

Parameters:	•**nfeatures** – The number of best features to retain. The features are ranked by th eir scores (measured in SIFT algorithm as the local contrast) •**nOctaveLayers** – The number of layers in each octave. 3 is the value used in D. Lowe paper. The number of octaves is computed automatically from the image re solution. •**contrastThreshold** – The contrast threshold used to filter out weak features in s emi-uniform (low-contrast) regions. The larger the threshold, the less features are produced by the detector. •**edgeThreshold** – The threshold used to filter out edge-like features. Note that t he its meaning is different from the contrastThreshold, i.e. the larger the edgeThr eshold, the less features are filtered out (more features are retained). •**sigma** – The sigma of the Gaussian applied to the input image at the octave #0. If your image is captured with a weak camera with soft lenses, you might want t o reduce the number.

SIFT(Scale-invariant feature transform) 특징:실습

- **SIFT::create ()**
  - Extract features and computes their descriptors using SIFT algorithm
  - void SIFT::create()(InputArray **img**, InputArray **mask**, vector<KeyPoint>& **keypoints**, OutputArray **descriptors**, bool **useProvidedKeypoints**=false)

Parameters:	•**nfeatures** – The number of best features to retain. The features are ranked by th eir scores (measured in SIFT algorithm as the local contrast) •**nOctaveLayers** – The number of layers in each octave. 3 is the value used in D. Lowe paper. The number of octaves is computed automatically from the image re solution. •**contrastThreshold** – The contrast threshold used to filter out weak features in s emi-uniform (low-contrast) regions. The larger the threshold, the less features are produced by the detector. •**edgeThreshold** – The threshold used to filter out edge-like features. Note that t he its meaning is different from the contrastThreshold, i.e. the larger the edgeThr eshold, the less features are filtered out (more features are retained). •**sigma** – The sigma of the Gaussian applied to the input image at the octave #0. If your image is captured with a weak camera with soft lenses, you might want t o reduce the number.

SIFT(Scale-invariant feature transform) 특징:실습

- SIFT 알고리즘 활용 예제

```
 int main(  )
{
      //source image
     char* img1_file = "F:/Temp/Images/monument1.jpg";
     char* img2_file = "F:/Temp/Images/monument2.jpg";

     // image read
     Mat img_scene = cv::imread(img1_file, 1);
     Mat img_object = cv::imread(img2_file, 1);

     if( !tmp.data || !in.data ){
     std::cout<< " --(!) Error reading images " << std::endl;
     return -1;
     }

     // SIFT feature detector and feature extractor
     cv::Ptr<SIFT> sift;
     sift = SIFT::create(0, 4, 0.04, 10, 1.6);
```

(계 속)

SIFT(Scale-invariant feature transform) 특징:실습

```
// Compute keypoints and descriptor from the source image in advance
vector<KeyPoint> keypoints1, keypoints2;
Mat descriptors1, descriptors2;


sift->detect(img_object, keypoints1);
sift->compute(img_object, keypoints1, descriptors1);

printf("original image:%d keypoints are found.₩n", (int)keypoints1.size());

for (int i = 0; i < keypoints1.size(); i++) {
KeyPoint kp = keypoints1[i];
circle(img_object, kp.pt, cvRound(kp.size*0.25), Scalar(255, 255, 0), 1, 8, 0);
}

namedWindow("SIFT Keypoints-src");
imshow("SIFT Keypoints-src", img_object);
```

Computes the descriptors for a set of keypoints detected in an image (first variant) or image set (second variant).

(계 속)

SIFT(Scale-invariant feature transform) 특징:실습

```
sift->detect(img_scene, keypoints2);
sift->compute(img_scene, keypoints2, descriptors2);

printf("original image:%d keypoints are found.₩n", (int)keypoints2.size());

for (int i = 0; i < keypoints2.size(); i++) {
KeyPoint kp = keypoints2[i];
circle(img_scene, kp.pt, cvRound(kp.size*0.25), Scalar(255, 255, 0), 1, 8, 0);
}

namedWindow("SIFT Keypoints-tgt");
imshow("SIFT Keypoints-tgt", img_scene);

//-- Step 3: Matching descriptor vectors using FLANN matcher
FlannBasedMatcher matcher;
std::vector< DMatch > matches;

matcher.match(descriptors1, descriptors2, matches);
```

(계 속)

SIFT(Scale-invariant feature transform) 특징:실습

```
double max_dist = 0; double min_dist = 100;

//-- Quick calculation of max and min distances between keypoints
for (int i = 0; i < descriptors1.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 < descriptors1.rows; i++)
{
        if (matches[i].distance < 3 * min_dist){
                good_matches.push_back(matches[i]);
        }
}
```

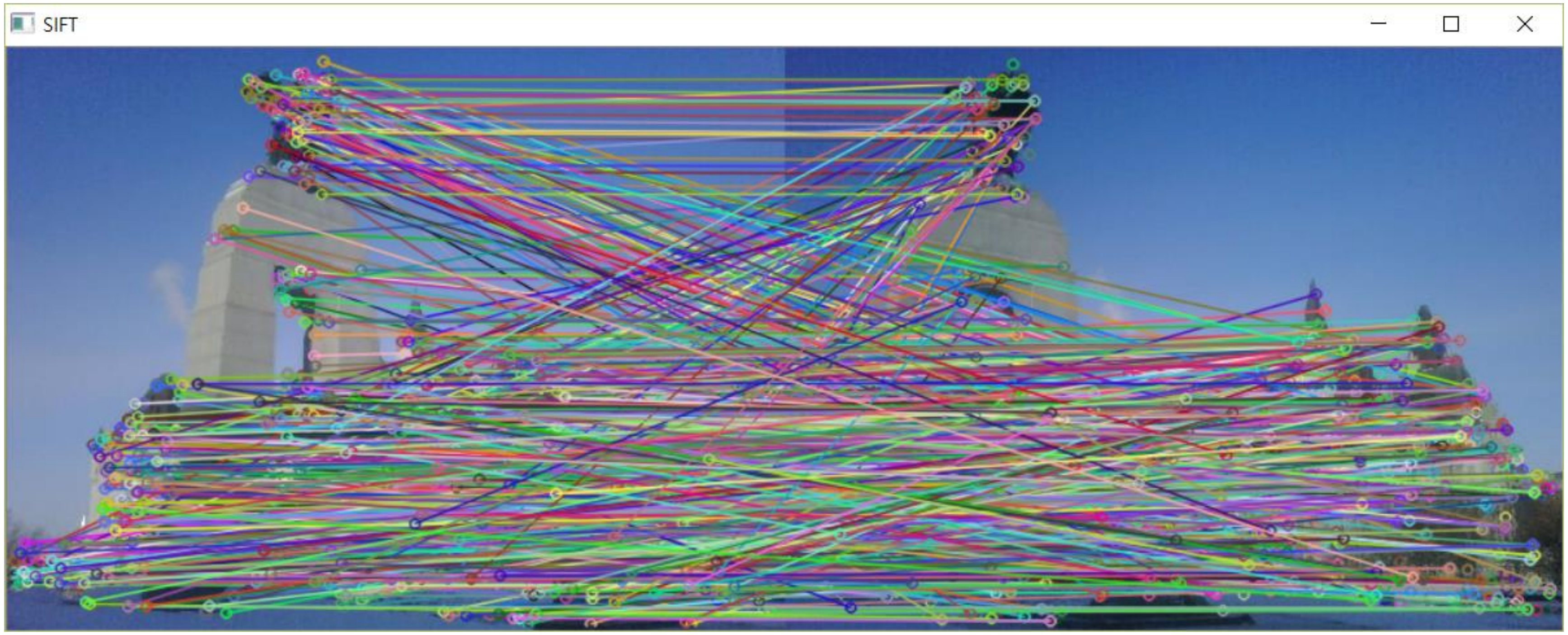
(계 속)

SIFT(Scale-invariant feature transform) 특징:실습

```
        Mat img_matches;
        drawMatches(img_object, keypoints1, img_scene, keypoints2,
        good_matches, img_matches, Scalar::all(-1), Scalar::all(-1),
        std::vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);

        imshow("Matched Image", img_matches);


        waitKey(0);

        return 0;
}
```

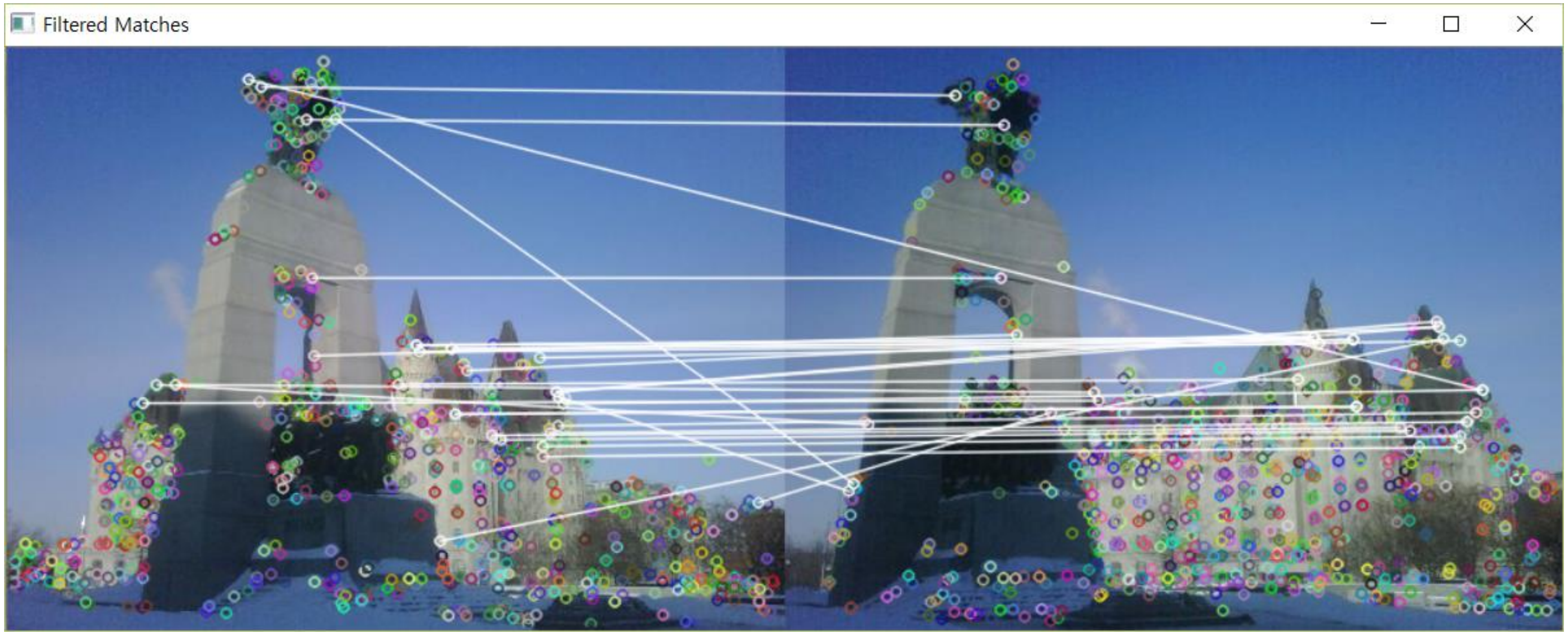
SIFT(Scale-invariant feature transform) 특징:실습

- 수행 결과: SIFT 특징 검출 및 매칭 결과
  - 약 818개, 907개 추출되어 매칭됨



SIFT(Scale-invariant feature transform) 특징:실습

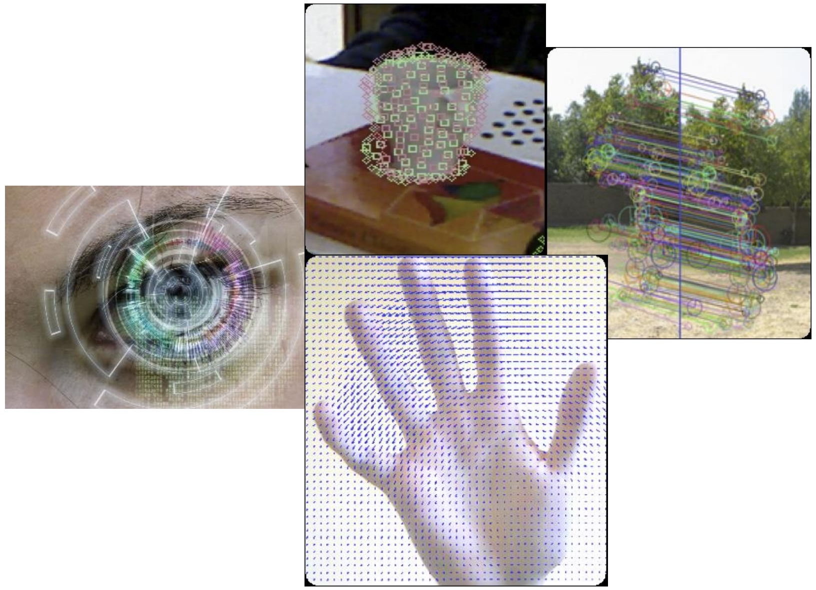
- 수행 결과: SIFT 특징 검출 및 매칭 결과
  - 약 30개로 제한하여 추출 및 매칭 결과



```
//--- Filtering loop ---//
std::nth_element(matches.begin(),     // initial position
                 matches.begin()+30, // position of the sorted element
                 matches.end());      // end position
// remove all elements after the 31th
matches.erase(matches.begin()+31, matches.end());
```

COMPUTER VISION 비젼 프로그래밍

Thank you and Question?



IVPL
Intelligent Vision Processing Lab