Feature Detection

Feature

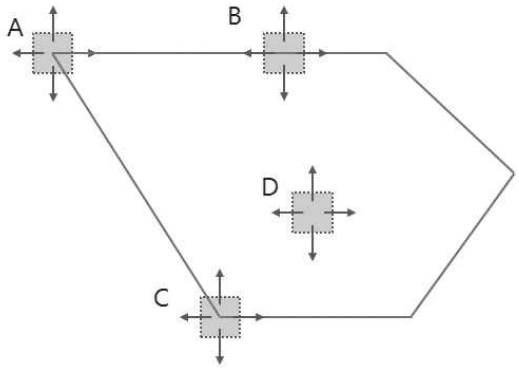
- 영상 특징 (feature)
  - 영상으로 추출할 수 있는 유용한 정보
  - 평균밝기, 히스토그램, 에지, 직선성분, 코너 등
- 지역 특징 (local feature)
  - 영상 전체가 아닌 일부 영역에서 추출할 수 있는 특징
  - 에지, 직선성분, 코너 등
- 특징점 (feature point)
  - 한 점의 형태로 표현할 수 있는 특징
  - 코너 등
  - Keypoint, interest point

Feature

- 특징의 분별력 비교
  - 부분 영상으로 전체영상에서의 위치를 판별하는 경우
  - 코너는 평탄영역(flat region) 이나 에지에 비하여 변별력이 높음

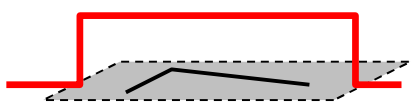
Harris Corner Detection

- Harris Corner
  - 1988, C.Harris
  - 코너점 아이디어를 수학적으로 잘 정의한 방법
- Corner point
  - (아이디어) 주변과 밝기 차이가 큰 점
  - (수식)



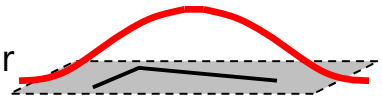
$$E(u,v)=\sum_y\sum_x G(x,y)\cdot(I(x+u,y+v)-I(x,y))^2 \qquad (x,y)\in W$$

Window function *G(x,y)* =



1 in window, 0 outside

or



Gaussian

Harris Corner Detection

- Taylor series

$$I(x+u, y+v) \cong I(x, y) + v d_y(x, y) + u d_x(x, y)$$

$$E(u, v) \cong \sum_y \sum_x G(x, y) \cdot \left(v d_y(x, y) + u d_x(x, y) \right)^2$$

$$\begin{aligned} E(u, v) &\cong \sum_y \sum_x G(x, y) \cdot (v d_y + u d_x)^2 \\ &= \sum_y \sum_x G(x, y) \cdot (v^2 d_y^2 + u^2 d_x^2 + 2 v u d_x d_y) \\ &= \sum_y \sum_x G(x, y) \cdot (u \; v) \begin{pmatrix} d_x^2 & d_x d_y \\ d_x d_y & d_y^2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} \\ &= (u \; v) M \begin{pmatrix} u \\ v \end{pmatrix}, \qquad M = \sum_y \sum_x G(x, y) \begin{pmatrix} d_x^2 & d_x d_y \\ d_x d_y & d_y^2 \end{pmatrix} \end{aligned}$$

특징 가능성을 직접 계산하는 대신 행렬 *M*의 식으로 정리

Harris Corner Detection

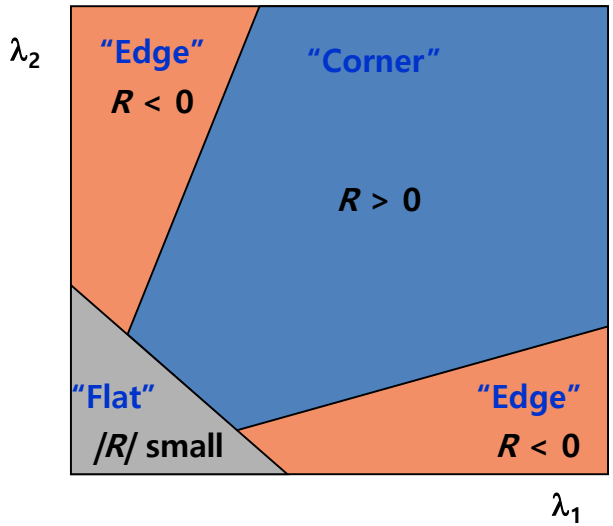
- Eigenvalue Analysis

$$M=\begin{pmatrix} d_x^2 & d_x d_y \\ d_x d_y & d_y^2 \end{pmatrix}=\begin{pmatrix} a & c \\ c & b \end{pmatrix}$$

$$R=det(M)-k\cdot trace(M)^2=(ab-c^2)-k\cdot(a+b)^2$$

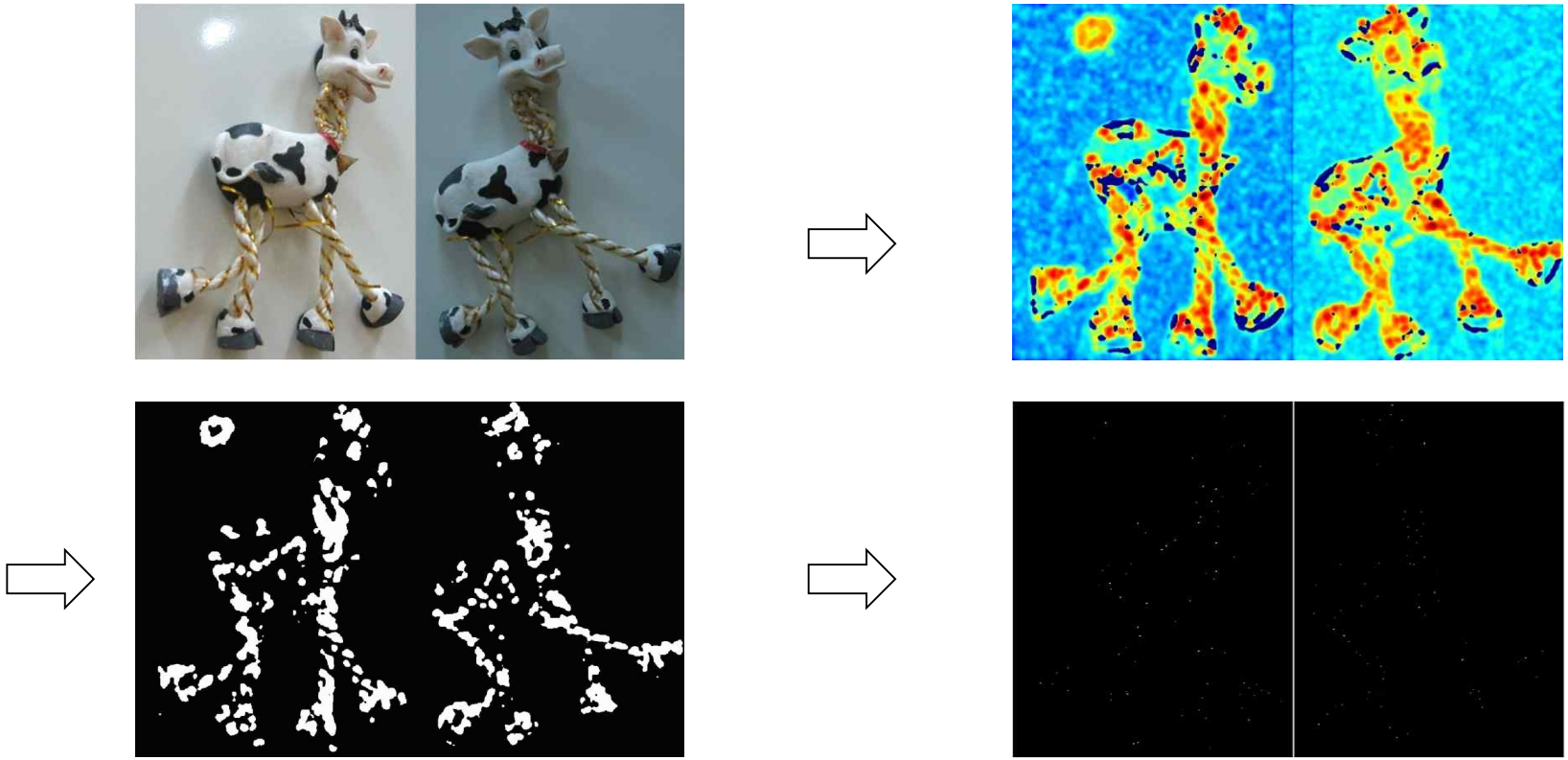
$$R=\lambda_1,\lambda_2-k\cdot(\lambda_1+\lambda_2)^2$$

k=0.04~0.06



Harris Corner Detection

- Algorithm
  1) Find points with large corner response function R ($R >$ threshold)
  2) Take the points of local maxima of R



Harris Corner Detection

- OpenCV 함수

```
void cornerHarris(InputArray src, OutputArray dst, int blockSize,
                  int ksize, double k, int borderType = BORDER_DEFAULT);
```

• src	입력 영상. CV_8UC1 또는 CV_32FC1
• dst	해리스 코너 응답 함수 값을 저장할 행렬. src와 크기가 같고 CV_32FC1 타입입니다.
• blockSize	행렬 M 연산에 사용할 이웃 픽셀 크기. 픽셀 주변 blockSize×blockSize 윈도우를 설정하여 행렬 M을 계산합니다.
• ksize	소벨 연산자를 위한 커널 크기
• k	해리스 코너 검출 상수
• borderType	가장자리 픽셀 확장 방식

Harris Corner Detection

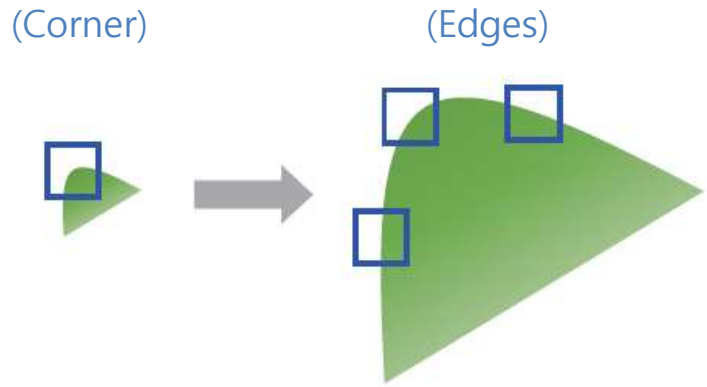
코드 14-1 해리스 코너 검출 예제 [ch14/corners]

```
01  void corner_harris()
02  {
03      Mat src = imread("building.jpg", IMREAD_GRAYSCALE);
04
05      if (src.empty()) {
06          cerr << "Image load failed!" << endl;
07          return;
08      }
09
10      Mat harris;
11      cornerHarris(src, harris, 3, 3, 0.04);
12
13      Mat harris_norm;
14      normalize(harris, harris_norm, 0, 255, NORM_MINMAX, CV_8U);
15
16      Mat dst;
17      cvtColor(src, dst, COLOR_GRAY2BGR);
18
19      for (int j = 1; j < harris.rows - 1; j++) {
20          for (int i = 1; i < harris.cols - 1; i++) {
21              if (harris_norm.at<uchar>(j, i) > 120) {
22                  if (harris.at<float>(j, i) > harris.at<float>(j - 1, i) &&
23                      harris.at<float>(j, i) > harris.at<float>(j + 1, i) &&
24                      harris.at<float>(j, i) > harris.at<float>(j, i - 1) &&
25                      harris.at<float>(j, i) > harris.at<float>(j, i + 1) ) {
26                      circle(dst, Point(i, j), 5, Scalar(0, 0, 255), 2);
27                  }
28              }
29          }
30      }
31
32      imshow("src", src);
33      imshow("harris_norm", harris_norm);
34      imshow("dst", dst);
35
36      waitKey(0);
37      destroyAllWindows();
38  }
```

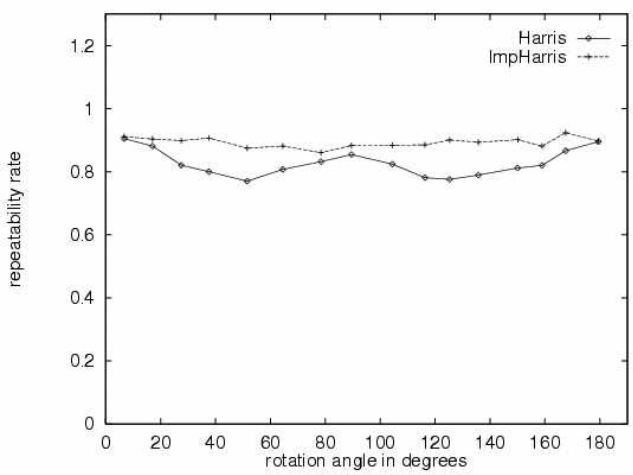
Harris Corner Detection

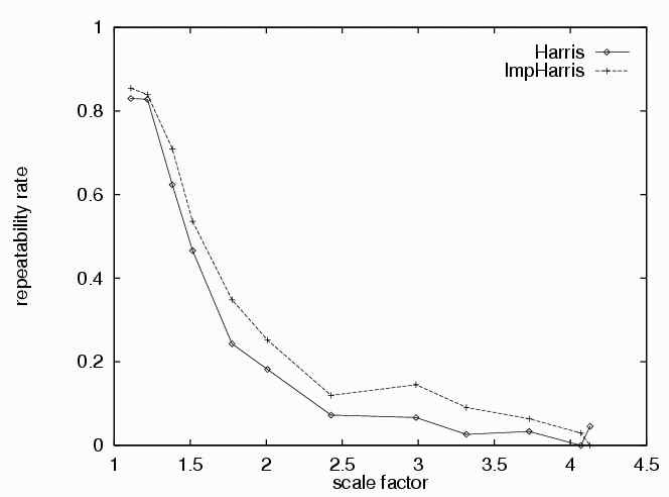
Harris Corner Detection

- 특징
  - 회전 변화에 강인
  - 크기 변화에 취약



$$\text{Repeatability rate} = \frac{\#\ \text{correspondences}}{\#\ \text{possible correspondences}}$$

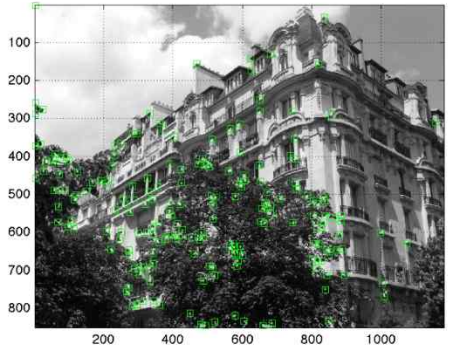


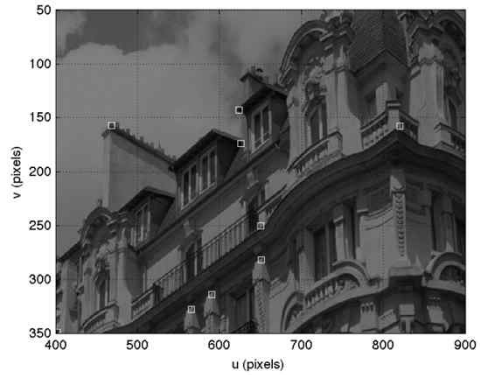


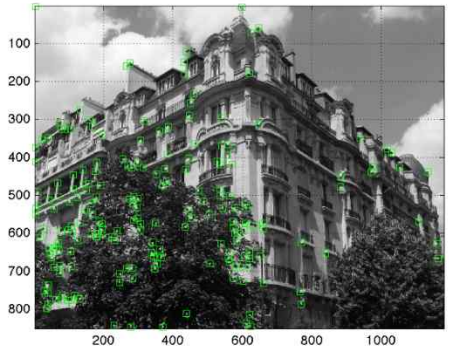
Harris Corner Detection

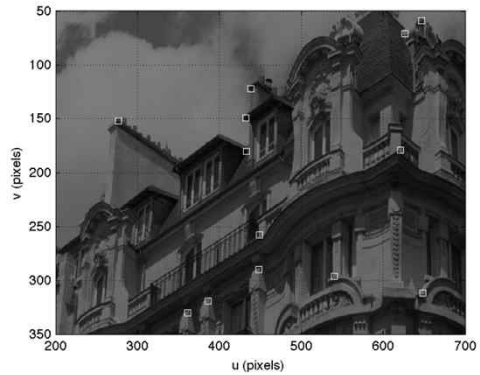
View1

View2









Original

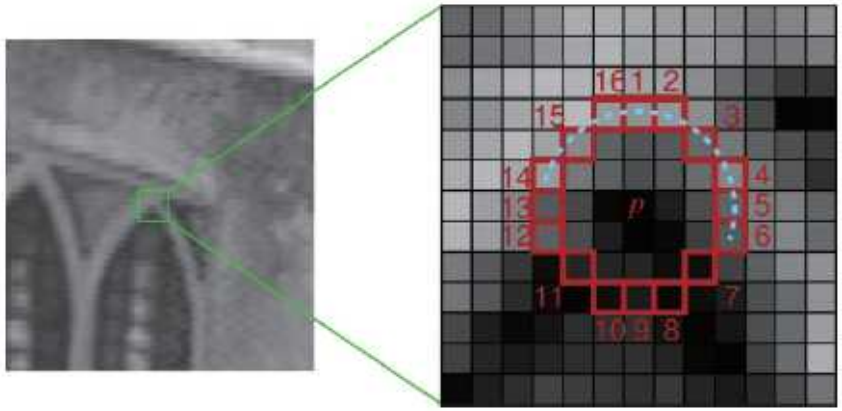
Zoomed

FAST

- FAST
  - Features from Accelerated Segment Test
  - 2006, ECCV, E.Rosten & T.Drummond
  - 단순한 픽셀 값 비교 방법을 통해 코너를 검출
  - 매우 빠르게 동작하는 코너 검출 방법

FAST

- 방법
  - 픽셀을 둘러싸고 있는 16개의 주변 픽셀과 밝기를 비교하여 코너 여부를 판별



- 주변 16개의 픽셀 중에서 그 값이 $I_p + t$ 보다 큰 픽셀이 아홉 개 이상 연속으로 나타나면 점 p는 어두운 영역이 뾰족하게 돌출되어 있는 코너
- 주변 16개의 픽셀 중에서 그 값이 $I_p - t$ 보다 작은 픽셀이 아홉 개 이상 연속으로 나타나면, 점 p는 밝은 영역이 돌출되어 있는 코너

  I_p : 점 p 의 밝기, t : threshold

- 비 최대치 억제 (non-maximal suppression)
  - 코너 점 주변 픽셀도 코너로 선택되는 현상 제거
  - 인접 코너 중 코너 점수가 가장 큰 코너 선택

FAST

- OpenCV 함수

```
void FAST(InputArray image, std::vector<KeyPoint>& keypoints,
          int threshold, bool nonmaxSuppression = true);
```

• image	입력 그레이스케일 영상
• keypoints	검출된 특징점을 표현하는 KeyPoint 객체의 벡터. KeyPoint::pt 멤버 변수에 코너 점 좌표가 저장됩니다.
• threshold	중심 픽셀 값과 주변 픽셀 값과의 차이 임계값
• nonmaxSuppression	비최대 억제 수행 여부. true이면 비최대 억제를 수행합니다.

cv::KeyPoints

Public Attributes

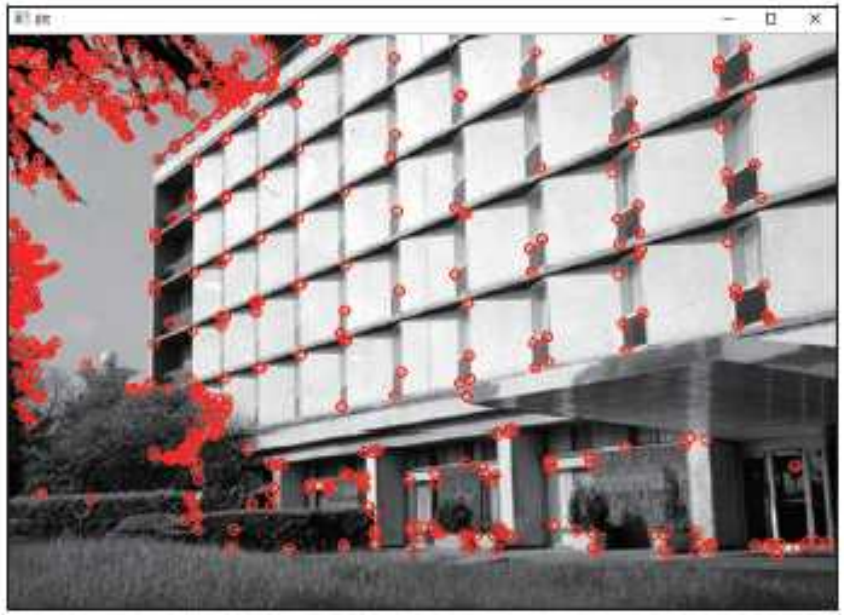
float	**angle**
int	**class_id** object class (if the keypoints need to be clustered by an object they belong to) More...
int	**octave** octave (pyramid layer) from which the keypoint has been extracted More...
Point2f	**pt** coordinates of the keypoints More...
float	**response** the response by which the most strong keypoints have been selected. Can be used for the further sorting or subsampling More...
float	**size** diameter of the meaningful keypoint neighborhood More...

FAST

코드 14-2 FAST 코너 검출 예제 [ch14/corners]

```
01  void corner_fast()
02  {
03      Mat src = imread("building.jpg", IMREAD_GRAYSCALE);
04
05      if (src.empty()) {
06          cerr << "Image load failed!" << endl;
07          return;
08      }
09
10      vector<KeyPoint> keypoints;
11      FAST(src, keypoints, 60, true);
12
13      Mat dst;
14      cvtColor(src, dst, COLOR_GRAY2BGR);
15
16      for (KeyPoint kp : keypoints) {
17          Point pt(cvRound(kp.pt.x), cvRound(kp.pt.y));
18          circle(dst, pt, 5, Scalar(0, 0, 255), 2);
19      }
20
21      imshow("src", src);
22      imshow("dst", dst);
23
24      waitKey(0);
25      destroyAllWindows();
26  }
```

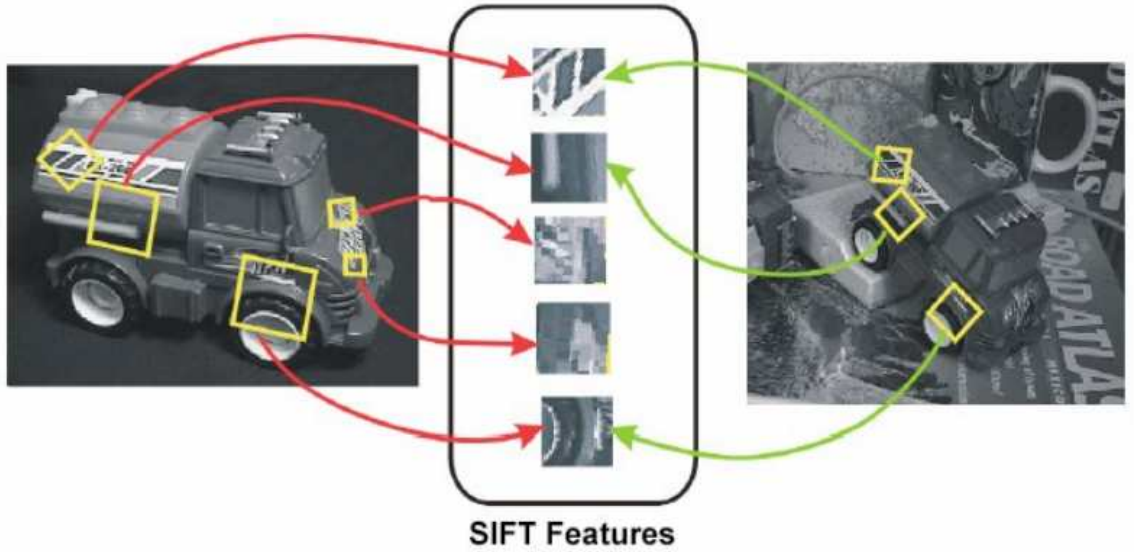
FAST



- 빠르다 (Harris corner 대비 20배)
- 잡음에 취약
- 임계값(t) 에 의존적

SIFT

- SIFT
  - Scale Invariant Feature Transform
  - 2004, D.Lowe
  - 영상의 회전 및 크기 변화에 무관하게 특징점 추출



SIFT

- Overview

  (Detector)

  1. Find scale-space extrema
  2. Keypoint localization & filtering

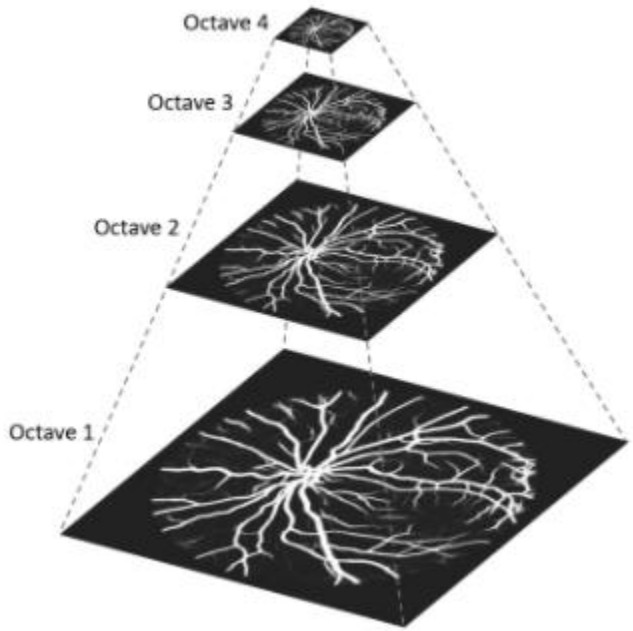
  (Descriptor)

  1. Orientation assignment
  2. Create descriptor

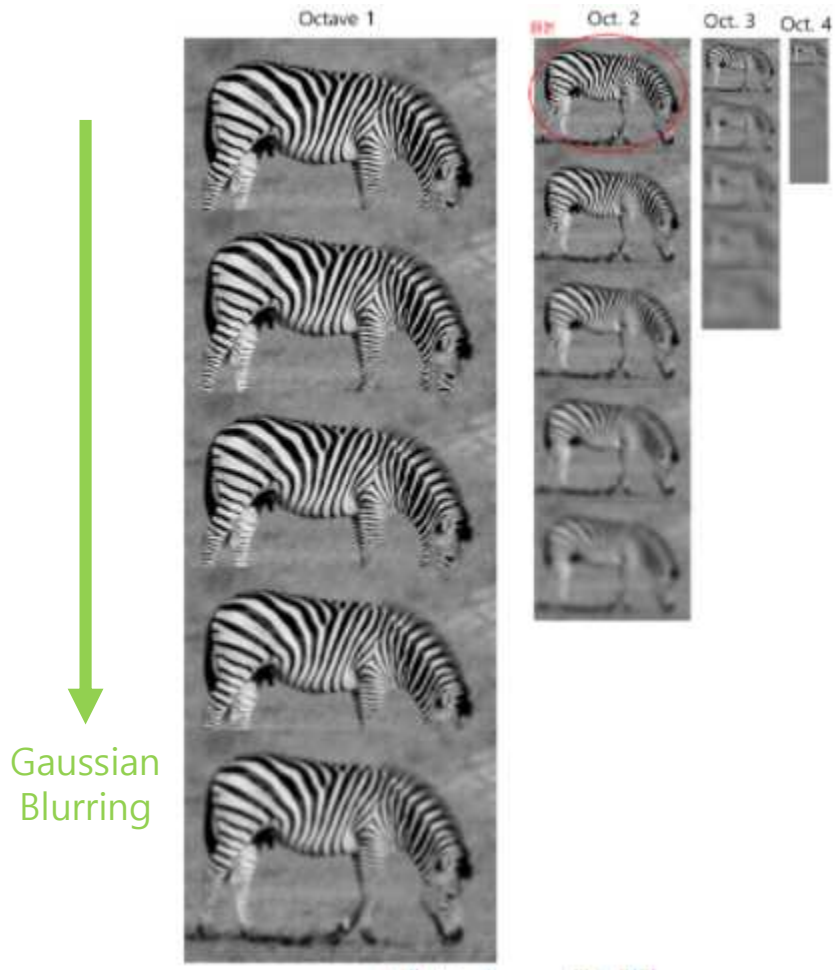
SIFT

1. Scale-space extrema

Image pyramid



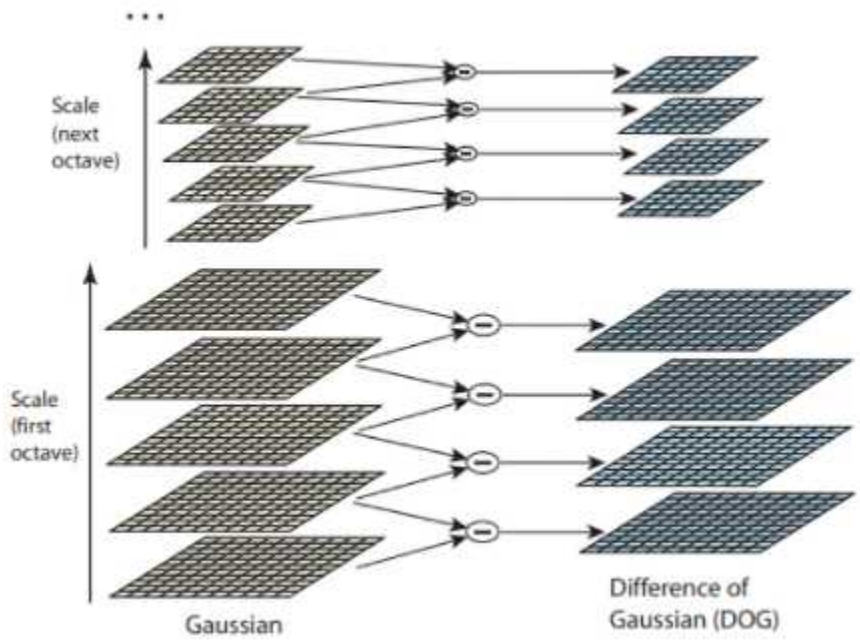
Scale space



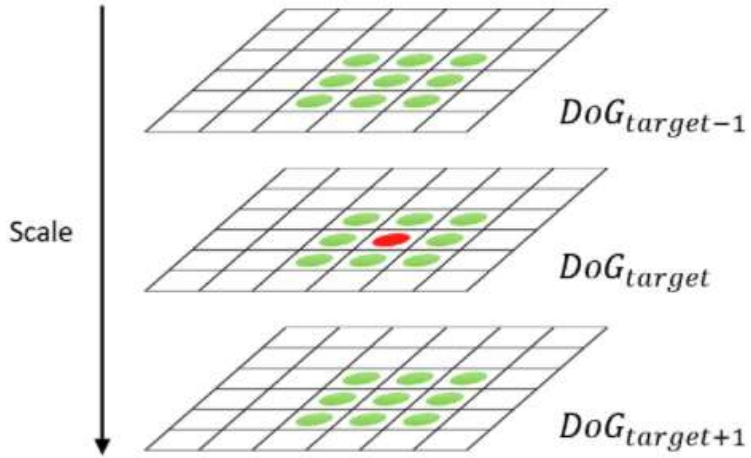
SIFT

1. Scale-space extrema

DoG (Difference of Gaussian)



Extrema



Choose all extrema within 3x3x3 neighborhoods

SIFT

2. Keypoint Localization & Filtering

– 우수한 extrema 선별 단계

832 extrema

729 extrema

536 extrema

SIFT

3. Orientation Assignment

– Keypoint 의 방향을 결정하는 단계

1) Compute gradient magnitude & orientation

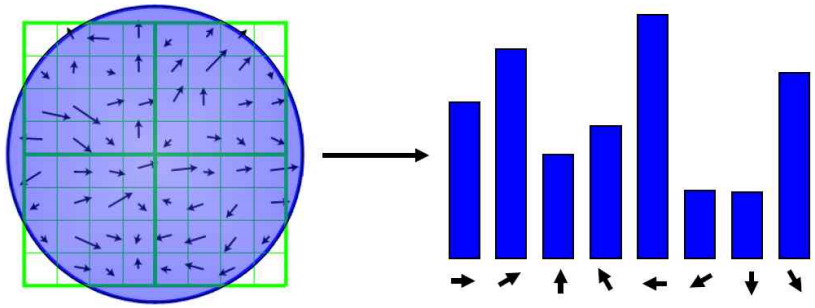
$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = \tan^{-1}\left(\frac{(L(x,y+1) - L(x,y-1))}{(L(x+1,y) - L(x-1,y))}\right)$$

단, $L(x,y) = G(x,y,\sigma) * I(x,y)$

Gaussian blurred image

2) Create gradient histogram



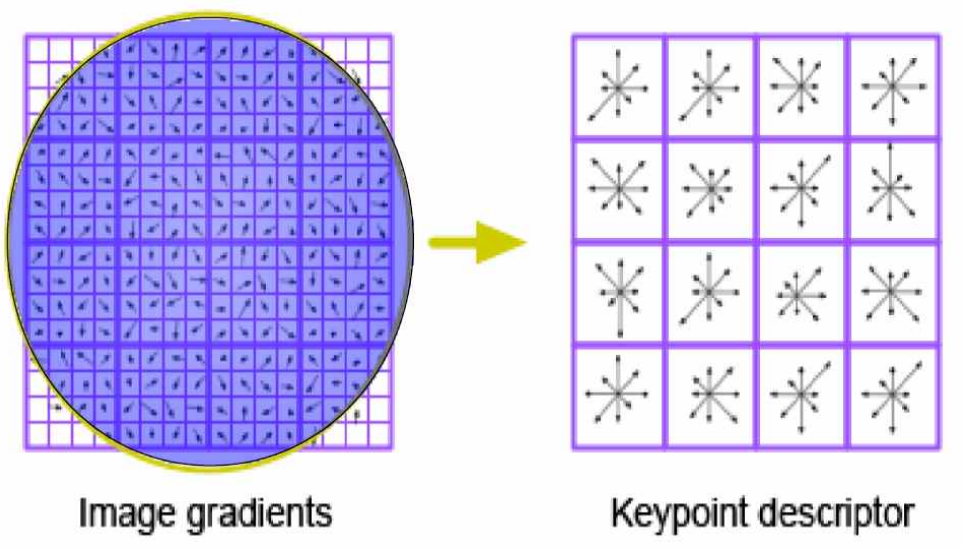
- 가장 높은 값을 갖는 bin 을 해당 Keypoint 의 방향으로 설정
- 가장 높은 값을 갖는 bin 의 80% 이상 갖는 bin 이 존재하면, 그 방향의 Keypoint 추가

keypoint1 : (x좌표, y좌표, scale factor, octave, orientation) = (10, 20, 2.0, 3, 95)

keypoint2 : (x좌표, y좌표, scale factor, octave, orientation) = (10, 20, 2.0, 3, 205)

SIFT

4. Keypoint Descriptor



1 개의 Keypoint
→ 4 x 4 gradient windows
→ histogram of 4 x 4 samples per window in 8 directions
→ 4 x 4 x 8 = 128 feature vectors

SIFT

- Performance
  - 영상의 크기, 회전 등의 변환뿐만 아니라 촬영 시점 변화에도 충분히 강인하게 동작함
  - 잡음의 영향과 조명 변화가 있어도 특징점을 반복적으로 잘 찾아냄
  - 객체 인식, 파노라마 영상 이어 붙이기, 3차원 장면 인식 등의 분야에서 효과적으로 사용됨

  - 계산량이 많아 SLAM (이동로봇, 자율주행) 등 실시간 응용에 한계

Scale Invariant 2D Features

- SURF (Speed-Up Robust Features) <2008>
  - SIFT 의 속도와 성능 개선
  - DoG 영상을 단순한 이진패턴으로 근사화
- KAZE <2012>
  - SIFT 의 크기, 회전, 잡음 변형에 대한 Repeatability 향상
  - Gaussian filter 대신 nonlinear diffusion filter 적용
- BRIEF (Binary Robust Independent Elementary Features) <2010>
  - Binary vector 를 사용하여 Descriptor 생성
  - 이진연산으로 빠른 계산 시간
- ORB (Oriented FAST and Rotated BRIEF) <2011>
  - FAST 기반 Keypoint & Descriptor 검출 → 빠른 계산 시간
  - 영상의 크기 변화에 대응하기 위하여 Pyramid 영상을 구축하여 Keypoint 추출
  - 방향을 고려한 BRIEF 알고리즘 적용하여 기술자 (Descriptor) 계산

OpenCV

- KeyPoint Class

코드 14-3 간략화한 KeyPoint 클래스 정의

```
01  class KeyPoint
02  {
03  public:
04      KeyPoint();
05      KeyPoint(Point2f _pt, float _size, float _angle = -1, float _response = 0,
06          int _octave = 0, int _class_id = -1);
07      ...
08
09      Point2f pt;
10      float   size;
11      float   angle;
12      float   response;
13      int     octave;
14      int     class_id;
15  };
```

KeyPoint::pt 멤버 변수는 특징점 좌표를 나타냅니다.

KeyPoint::size 멤버 변수는 특징점 크기(지름)를 나타냅니다.

KeyPoint::angle 멤버 변수는 특징점의 주된 방향(각도)을 나타냅니다.

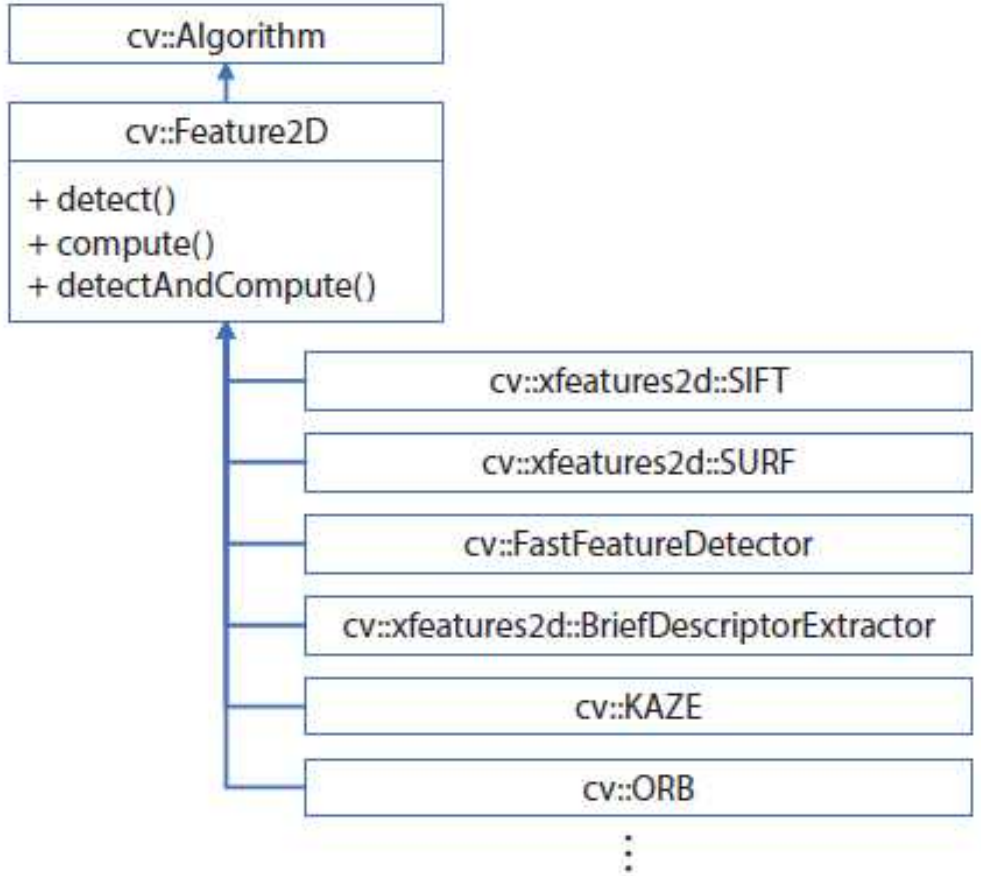
KeyPoint::response 멤버 변수는 특징점 반응성을 나타내며, 좋은 특징점을 선별하는 용도로 사용할 수 있습니다.

KeyPoint::octave 멤버 변수는 특징점이 추출된 옥타브(피라미드 단계)를 나타냅니다.

KeyPoint::class_id 멤버 변수는 특징점이 포함된 객체 번호를 나타냅니다.

OpenCV

- Keypoint Detection Class



OpenCV

- ORB 객체 생성

```
static Ptr<ORB> ORB::create(int nfeatures = 500, float scaleFactor = 1.2f,
                            int nlevels = 8, int edgeThreshold = 31,
                            int firstLevel = 0, int WTA_K = 2,
                            ORB::ScoreType scoreType = ORB::HARRIS_SCORE,
                            int patchSize = 31, int fastThreshold = 20);
```

• nfeatures	검출할 최대 특징 개수
• scaleFactor	피라미드 생성 비율(영상 축소 비율)
• nlevels	피라미드 단계 개수
• edgeThreshold	특징을 검출하지 않을 영상 가장자리 픽셀 크기
• firstLevel	항상 0을 지정해야 합니다.
• WTA_K	BRIEF 기술자 계산 시 사용할 점의 개수. 2, 3, 4 중 하나를 지정해야 합니다.
• scoreType	특징점 점수 결정 방법. ORB::HARRIS_SCORE 또는 ORB::FAST_SCORE 둘 중 하나를 지정합니다.
• patchSize	BRIEF 기술자 계산 시 사용할 패치 크기
• fastThreshold	FAST 코너 검출 방법에서 사용되는 임계값
• 반환값	ORB 객체를 참조하는 Ptr 스마트 포인터 객체

```
Ptr<ORB> feature = ORB::create();
Ptr<Feature2D> feature = ORB::create();
```

OpenCV

- Detect Keypoint

```
virtual void Feature2D::detect(InputArray image,
                               std::vector<KeyPoint>& keypoints,
                               InputArray mask = noArray());
```

- image 입력 영상
- keypoints 검출된 키포인트 정보
- mask 마스크 행렬. 마스크 행렬 원소가 0이 아닌 위치에서만 특징점을 검출합니다.

```
Mat src = imread("lenna.bmp", IMREAD_GRAYSCALE);

Ptr<Feature2D> feature = ORB::create();

vector<KeyPoint> keypoints;
feature->detect(src, keypoints);
```

OpenCV

- Compute Descriptor

```
virtual void Feature2D::compute(InputArray image,
                                std::vector<KeyPoint>& keypoints,
                                OutputArray descriptors);
```

- image 입력 영상
- keypoints 미리 검출해 둔 키포인트 정보
- descriptors 계산된 기술자 행렬. i번째 행은 i번째 키포인트의 기술자를 나타냅니다.

OpenCV

- Detect Keypoint & Compute descriptor

```
virtual void Feature2D::detectAndCompute(InputArray image, InputArray mask,
                                         std::vector<KeyPoint>& keypoints,
                                         OutputArray descriptors,
                                         bool useProvidedKeypoints = false);
```

• image	입력 영상
• mask	마스크 행렬. 마스크 행렬 원소가 0이 아닌 위치에서만 특징점을 검출합니다.
• keypoints	검출된 키포인트 정보
• descriptors	계산된 기술자 행렬
• useProvidedKeypoints	이 값이 true이면 keypoints 인자로 전달된 키포인트 정보를 이용하여 기술자를 계산합니다.

OpenCV

- Draw Keypoint

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

- image 입력 영상
- keypoints 입력 영상에서 검출된 키포인트
- outImage 키포인트가 그려진 출력 영상
- color 키포인트 색상. 이 값이 Scalar::all(-1)이면 각 특징점을 임의의 색상으로 그립니다.
- flags 키포인트 그리기 방법. DrawMatchesFlags 열거형 상수 중 하나를 지정합니다.

DrawMatchesFlags 열거형 상수	설명
DEFAULT	기본 방식. 검출된 모든 특징점에 작은 크기의 원을 그리고, 서로 매칭된 특징점끼리 직선을 그립니다.
DRAW_OVER_OUTIMG	출력 영상을 새로 생성하지 않고 전달된 영상 위에 그립니다.
NOT_DRAW_SINGLE_POINTS	drawMatches() 함수와 함께 사용되며, 매칭되지 않은 특징점은 그리지 않습니다.
DRAW_RICH_KEYPOINTS	키포인트의 크기와 방향 정보를 함께 나타냅니다.

OpenCV

코드 14-4 키포인트 검출 예제 [ch14/keypoints]

```
01  void detect_keypoints()
02  {
03      Mat src = imread("box_in_scene.png", IMREAD_GRAYSCALE);
04
05      if (src.empty()) {
06          cerr << "Image load failed!" << endl;
07          return;
08      }
09
10      Ptr<Feature2D> feature = ORB::create();
11
12      vector<KeyPoint> keypoints;
13      feature->detect(src, keypoints);
14
15      Mat desc;
16      feature->compute(src, keypoints, desc);
17
18      cout << "keypoints.size(): " << keypoints.size() << endl;
19      cout << "desc.size(): " << desc.size() << endl;
20
21      Mat dst;
22      drawKeypoints(src, keypoints, dst, Scalar::all(-1),
                      DrawMatchesFlags::DRAW_RICH_KEYPOINTS);
23
24      imshow("src", src);
25      imshow("dst", dst);
26
27      waitKey();
28      destroyAllWindows();
29  }
```

OpenCV