지역 특징

Computer Vision and Pattern Recognition

Byeongjoon Noh

powernoh@sch.ac.kr

SCH SOON CHUN HYANG UNIVERSITY

Most of the slides are available on Senseable AI Lab homepage: https://sailab.space/courses/

Contents

1. 이동과 회전 불변한 지역특징

영상에서의 특징

- 저수준 특징 (Low-level features)
  - edge, corners, points, segments, shapes, texture
- 중수준 특징 (Mid-level features)
  - contour (윤곽), local feature (SIFT, SURF)
- 고수준 특징 (High-level features)
  - object recognition, semantic segmentation, instance segmentation
- 기하학적 특징 (Geometry features)
  - 3D structure, position, orientation (direction)
- 전역 특징 (Global features)
  - histogram, moments

Getting started

- 대응점 찾기 (corresponding problem)
  - 같은 장면을 다른 시점에서 찍은 두 영상에서 대응하는 점의 쌍을 찾는 문제
  - 물체 인식, 추적, 카메라 calibration, stereo vision 등 컴퓨터 비전 문제 해결에 필수
  - ex) 파노라마 영상 제작에서 봉합 점을 결정

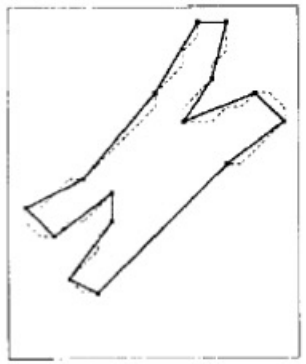
그림 5-1 지역 특징으로 대응점 문제를 해결해서 제작한 파노라마 영상

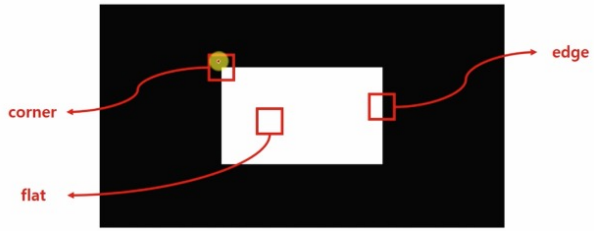
  - 저수준 특징 (edge / region)은 여러모로 부족
    - ➔ 지역 특징의 개념 등장

Getting started

- 물체 추적에서 대응점 찾기
  - ex) 다중 물체 추적 챌린지 MOT-17-14-SDP 동영상 데이터셋 (https://motchallenge.net/data/MOT17)
  - 반복성(repeatability)이 뛰어난 특징 필요

[그림 5-2] 대응점 찾기(MOT-17-14-SDP 동영상의 70번째와 83번째 영상)





- edge 경계선에서 corner를 찾기 시작
  - corner: edge 토막에서 곡률이 큰 지점
  - 특징점이 물체의 실제 corner에 해당해야 한다는 생각이 지배적 ➔ 지역 특징의 등장과 함께 ㅂㅂ

Recall: 특징의 불변성과 등변성

- 불변성 (Invariant)
  - 변환을 해도 값이 변하지 않는 특징
  - ex) 성별 ➔ 나이에 불변

- 등변성 (Equivariant)
  - 변환에 따라 값이 변하는 특징
  - ex) 면적 ➔ 축소에는 등변, 회전에는 등변이 아님

- 목적에 따라 특징 선택이 중요
  - ex) 물체 인식 후 물체를 집는 로봇 ➔ 회전에 등변인 특징을 사용해야 함

지역 특징의 조건

- 반복성 (repeatability)
- 불변성 (invariance)
- 분별력 (discriminating power)
- 지역성 (locality)
- 적당한 양
- 계산 효율

- ➔ 이들은 서로 상충 관계
  - 목적에 따라 적절한 조절 필요

지역 특징의 조건

- 이동과 회전에 불변한 지역 특징 검출 방법
  - Moravec algorithm
  - Harris corner detection

- 스케일에 불변한 지역 특징 검출 방법
  - SIFT
  - SURF

이동과 회전 불변한 지역 특징

- 간단한 인지 실험
  - 왼쪽 영상의 a, b, c 중 어느 것이 오른쪽 영상에서 찾기 쉬울까?
    - a가 가장 쉽고, c가 가장 어려움. Why?

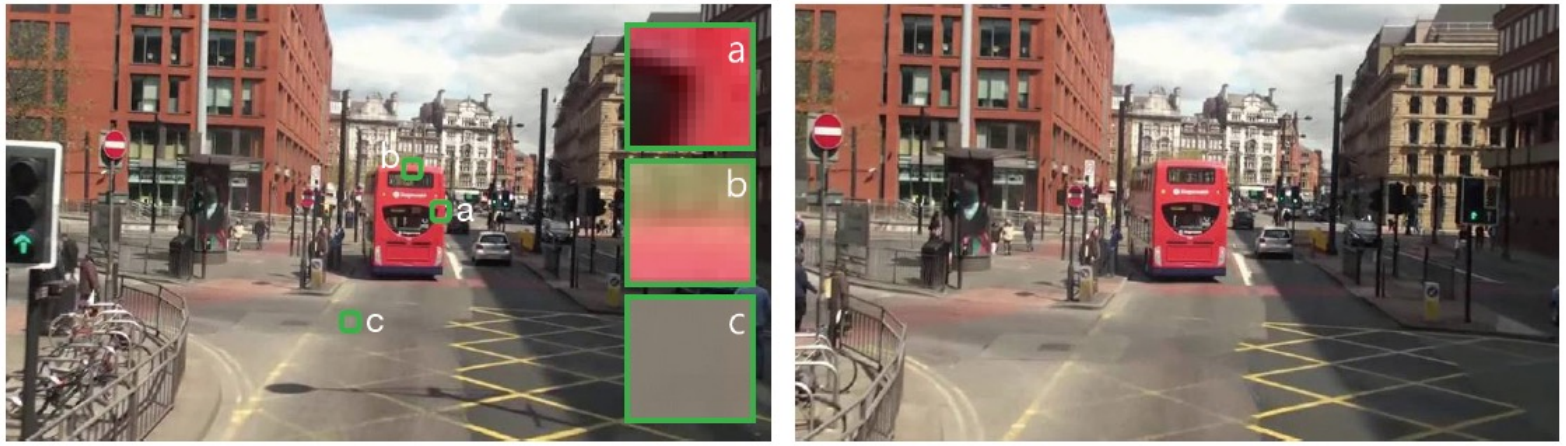
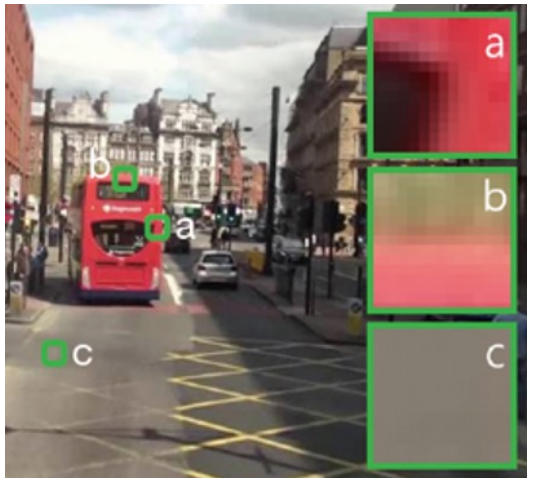


그림 5-3 대응점 찾기 인지 실험(MOT-17-14-SDP 동영상의 70번째와 83번째 영상)

  - idea: 사람에게 쉬운 곳이 컴퓨터에게도 쉽지 않을까? ➔ 좋은 정도를 어떻게 "수량화" 할까?

Moravec algorithm

- Moravec의 설명: 인지 실험에 주목
  - a는 여러 방향으로 색상 변화가 있음
  - c는 어느 방향으로도 미세한 변화만 있어 어려움



- 여러 방향에 대한 색상 변화를 측정(수량화)하는 방법을 제안
  - 중심을 기준으로 여러 방향의 색상의 제곱차의 합 (sum of squared difference, SSD)

$$S(v,u) = \sum_{y}\sum_{x}\big(f(y+v,x+u) - f(y,x)\big)^2$$

Moravec algorithm

- ex) 화소 위치 (y, x)가 (4, 3)인 경우

$$S(v,u)=\sum_{3\le y\le 5}\sum_{2\le x\le 4}\left(f(y+v,x+u)-f(y,x)\right)^2$$

➔ $S(0,1)=\sum_{3\le y\le 5}\sum_{2\le x\le 4}\left(f(y,x+1)-f(y,x)\right)^2=4$

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	c 0	0	0
2	0	0	0	1	0	0	0	0	0	0
3	0	0	0	1	1	0	0	0	0	0
4	0	0	0	b 1	1	1	0	0	0	0
5	0	0	0	1	1	1	1	0	0	0
6	0	0	0	1	1	1	1	a 1	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0

	u -1	u 0	u 1
-1	3	4	4
v 0	2	0	2
1	4	3	2

a

	u -1	u 0	u 1
-1	3	1	6
v 0	3	0	4
1	3	0	3

b

	u -1	u 0	u 1
-1	0	0	0
v 0	0	0	0
1	0	0	0

c

Moravec algorithm

- 상세 계산 과정

$$S(0,1) = \sum_{3\le y\le 5}\sum_{2\le x\le 4}\left(f(y,x+1)-f(y,x)\right)^2$$

$$= \sum_{3\le y\le 5}\left(\left(f(y,3)-f(y,2)\right)^2+\left(f(y,4)-f(y,3)\right)^2+\left(f(y,5)-f(y,4)\right)^2\right)$$

$$= \left(\left(f(3,3)-f(3,2)\right)^2+\left(f(3,4)-f(3,3)\right)^2+\left(f(3,5)-f(3,4)\right)^2\right)$$

1 0 1 1 0 1

$$+\left(\left(f(4,3)-f(4,2)\right)^2+\left(f(4,4)-f(4,3)\right)^2+\left(f(4,5)-f(4,4)\right)^2\right)$$

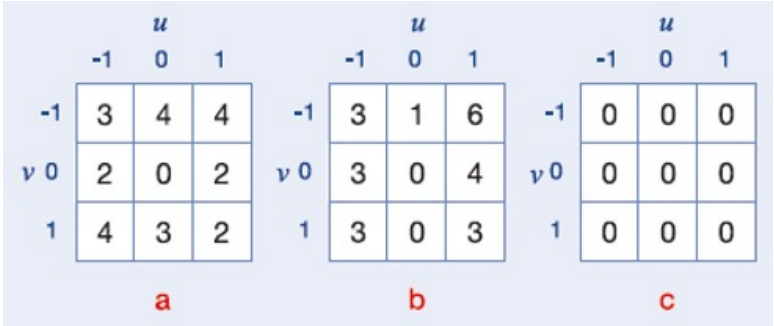
1 0 1 1 1 1

$$+\left(\left(f(5,3)-f(5,2)\right)^2+\left(f(5,4)-f(5,3)\right)^2+\left(f(5,5)-f(5,4)\right)^2\right)=4$$

1 0 1 1 1 1

Moravec algorithm

- 의미 및 의도
  - a
    - 모든 방향으로 변화가 있어 S맵에서 8-connectivity 모두 큰 값
    - 지역 특징으로서 훌륭함
  - b
    - 수평 방향만 변화가 있음
    - 지역 특징으로 부족
  - c
    - 모든 방향에서 변화가 없음
    - 지역 특징으로 자격이 없음



v \ u	-1	0	1
-1	3	4	4
0	2	0	2
1	4	3	2

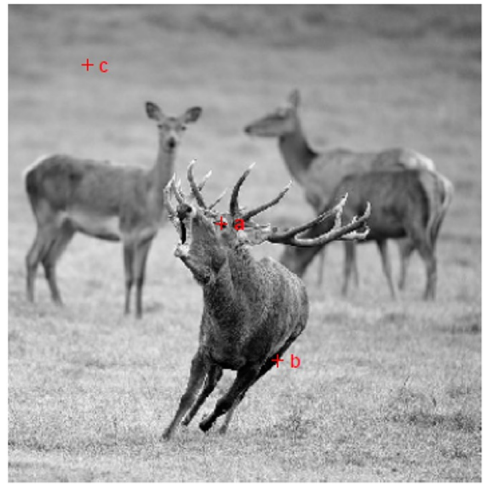
a

v \ u	-1	0	1
-1	3	1	6
0	3	0	4
1	3	0	3

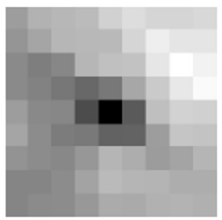
b

v \ u	-1	0	1
-1	0	0	0
0	0	0	0
1	0	0	0

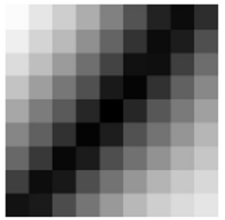
c



> 원래 영상



> a



> b

> c

Moravec algorithm

- Moravec algorithm을 통해 지역 특징의 길을 열었지만 현실적이지는 않음
  - 상하좌우만 보는 것으로는 부족
    - 28.5도, 34도 등 정방향 회전이 아닌 경우
  - Harris의 확장
    - ➔ 가중치를 두면 어떨까?

Harris corner detection

- “가중치” 제곱 차의 합 (weighted sum of squared difference, WSSD)를 이용한 noise 대처
  - 가중치 = Gaussian filter

$$S(v,u) = \sum_{y}\sum_{x} G(y,x)\big(f(y+v,x+u) - f(y,x)\big)^2$$

- 모든 방향에 관한 조사 ➔ Moravec의 4방향만 조사하는 한계점 해결
  - ➔ 미분 도입 (+Taylor expansion)

Harris corner detection

- (참고) Taylor expansion (Taylor series)
  - 어떤 함수를 그 함수의 도함수(derivatives)를 사용하여 다항식으로 근사하는 방법 by Brook Taylor
  - 함수의 값을 그 함수의 도함수들의 값과 어떤 지점에서의 함수값으로 표현 ➔ 복잡한 함수를 다루기 쉬운 다항식으로 근사할 수 있음

  - 함수 $f(x)$가 $x = a$에서 모든 차수의 도함수를 가지고 있을 때 ($x = a$ 근처에서 Taylor series로 근사)

$$f(x) \cong f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots$$

  - 함수 $f(x+a)$의 Taylor series 근사 (x를 기준점으로 $x+a$로 이동한 지점에서 근사)

$$f(x+a) \cong f(x) + af'(x) + \frac{a^2}{2!}f''(x) + \cdots$$

Harris corner detection

- (참고) Taylor expansion (Taylor series)
  - 함수 $f(x,y)$의 Taylor series 근사

$$f(x,y) \cong f(u,v) + \frac{\partial f}{\partial x}(u,v)(x-u) + \frac{\partial f}{\partial y}(u,v)(y-v)$$

$$+\frac{1}{2!}\left(\frac{\partial^2 f}{\partial x^2}(u,v)(x-u)^2 + 2\frac{\partial^2 f}{\partial x \partial y}(u,v)(x-u)(y-b) + \frac{\partial^2 f}{\partial y^2}(u,v)(y-v)^2\right) + \cdots$$

  - 함수 $f(y+u, x+v)$의 Taylor series 근사

$$f(v+y, u+x) \cong f(y,x) + v\frac{\partial f}{\partial y}(y,x) + u\frac{\partial f}{\partial x}(y,x)$$

$$f(v+y, u+x) \cong f(y,x) + vd_y(y,x) + ud_x(y,x)$$

Harris corner detection

- 대입, 정리

Recall: $f(v+y,u+x) \cong f(y,x) + vd_y(y,x) + ud_x(y,x)$

$$S(v,u) = \sum_y \sum_x G(y,x)\big(f(y+v,x+u) - f(y,x)\big)^2$$

$$\cong \sum_y \sum_x G(y,x)\left(vd_y(y,x) - ud_x(y,x)\right)^2$$

$$= \sum_y \sum_x G(y,x)\left(v^2 d_y^2 + 2vud_y d_x + u^2 d_x^2\right)$$

$$= \sum_y \sum_x G(y,x)(v \quad u)\begin{pmatrix} d_y^2 & d_y d_x \\ d_y d_x & d_x^2 \end{pmatrix}\begin{pmatrix} v \\ u \end{pmatrix}$$

$$= (v \quad u)\sum_y \sum_x G(y,x)\begin{pmatrix} d_y^2 & d_y d_x \\ d_y d_x & d_x^2 \end{pmatrix}\begin{pmatrix} v \\ u \end{pmatrix} = (v \quad u)\begin{pmatrix} \sum_y \sum_x G(y,x)d_y^2 & \sum_y \sum_x G(y,x)d_y d_x \\ \sum_y \sum_x G(y,x)d_y d_x & \sum_y \sum_x G(y,x)d_x^2 \end{pmatrix}\begin{pmatrix} v \\ u \end{pmatrix}$$

$$\mathbf{uAu^T}$$

$$\|$$

Harris corner detection

- 2차 모멘트 행렬 (구조행렬) $\mathbf{A}$
  - $\mathbf{A}$는 화소 주위의 영상 구조를 표현 ➔ $\mathbf{A}$만 분석하면 지역 특징 여부를 알 수 있음
    - (v, u)를 변화시키면서 맵을 생성할 필요 없음

- Harris의 특징 가능성 R 계산
  - $R = \mathbf{det}(\mathbf{A}) - k\big(\mathbf{trace}(\mathbf{A})\big) = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)$
    - $\lambda_1, \lambda_2 = \mathbf{A}$의 고유값

- R이 클 수록 코너일 가능성이 높음을 의미

표 5-1 [그림 5-4(a)]에서 세 점의 특징 가능성 측정

	a	b	c
2차 모멘트 행렬	$\mathbf{A} = \begin{pmatrix} 0.52 & -0.2 \\ -0.2 & 0.53 \end{pmatrix}$	$\mathbf{A} = \begin{pmatrix} 0.08 & -0.08 \\ -0.08 & 0.8 \end{pmatrix}$	$\mathbf{A} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
고윳값	λ_1=0.72, λ_2=0.33	λ_1=0.81, λ_2=0.07	λ_1=0.0, λ_2=0.0
특징 가능성 값	C=0.1925	C=0.0237	C=0.0

Harris corner detection

- Harris 특징점의 분석
  - 물체의 실제 모퉁이 뿐 아니라 blob에서도 검출 가능
  - Corner 대신 특징점(feature point) 또는 관심점(interest point)라고 부름
  - 이동과 회전에 불변
  - 스케일에는 가변

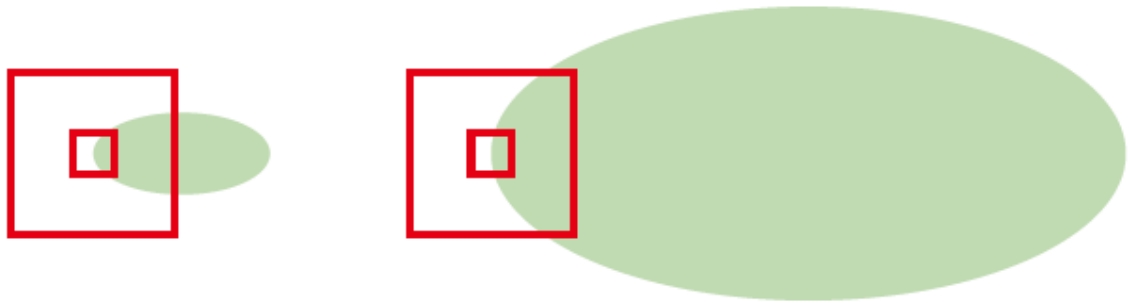


그림 5-6 물체의 크기에 따라 마스크의 크기를 적절하게 정해야 하는 상황

다양한 특징점 검출 알고리즘

- Moravec algorithm
- Harris corner (feature point) detection
- Hessian matrix

$$C = \det(A) - k \times trace(A)^2 = (pq - r^2) - k(p+q)^2$$

- LOG

$$C = \nabla^2 = trace(\mathbf{H}) = d_{yy}(\sigma) + d_{xx}(\sigma)$$

- SUSAN (Small Uni-value Segment Assimilating Nucleus, 슈산)

$$C = \begin{cases} q - usan\_area(r_o), & usan\_area(r_o) \le t \\ 0, & otherwise \end{cases}$$

프로그래밍 실습: Harris 특징점 검출

프로그램 5-1 해리스 특징점 검출 구현하기

```
01  import cv2 as cv
02  import numpy as np
03
04  img=np.array([[0,0,0,0,0,0,0,0,0,0],
05                [0,0,0,0,0,0,0,0,0,0],
06                [0,0,0,1,0,0,0,0,0,0],
07                [0,0,0,1,1,0,0,0,0,0],
08                [0,0,0,1,1,1,0,0,0,0],
09                [0,0,0,1,1,1,1,0,0,0],
10                [0,0,0,1,1,1,1,1,0,0],
11                [0,0,0,0,0,0,0,0,0,0],
12                [0,0,0,0,0,0,0,0,0,0],
13                [0,0,0,0,0,0,0,0,0,0]],dtype=np.float32)
14
15  ux=np.array([[-1,0,1]])
16  uy=np.array([-1,0,1]).transpose()
17  k=cv.getGaussianKernel(3,1)
18  g=np.outer(k,k.transpose())
19
```

프로그래밍 실습: Harris 특징점 검출

```
20  dy=cv.filter2D(img,cv.CV_32F,uy)
21  dx=cv.filter2D(img,cv.CV_32F,ux)
22  dyy=dy*dy
23  dxx=dx*dx
24  dyx=dy*dx
25  gdyy=cv.filter2D(dyy,cv.CV_32F,g)
26  gdxx=cv.filter2D(dxx,cv.CV_32F,g)
27  gdyx=cv.filter2D(dyx,cv.CV_32F,g)
28  C=(gdyy*gdxx-gdyx*gdyx)-0.04*(gdyy+gdxx)*(gdyy+gdxx)
29
30  for j in range(1,C.shape[0]-1):            # 비최대 억제
31      for i in range(1,C.shape[1]-1):
32          if C[j,i]>0.1 and sum(sum(C[j,i]>C[j-1:j+2,i-1:i+2]))==8:
33              img[j,i]=9                     # 특징점을 원본 영상에 9로 표시
34
```

프로그래밍 실습: Harris 특징점 검출

```
35  np.set_printoptions(precision=2)
36  print(dy) ①
37  print(dx) ②
38  print(dyy) ③
39  print(dxx) ④
40  print(dyx) ⑤
41  print(gdyy) ⑥
42  print(gdxx) ⑦
43  print(gdyx) ⑧
44  print(C) ⑨                              # 특징 가능성 맵
45  print(img) ⑩                            # 특징점을 9로 표시한 원본 영상
46
47  popping=np.zeros([160,160],np.uint8)    # 화소 확인 가능하게 16배로 확대
48  for j in range(0,160):
49      for i in range(0,160):
50          popping[j,i]=np.uint8((C[j//16,i//16]+0.06)*700)
51
52  cv.imshow('Image Display2',popping) ⑪
53  cv.waitKey()
54  cv.destroyAllWindows()
```

프로그래밍 실습: Harris 특징점 검출

①

```
[[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  1.  1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  1.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  1.  1.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  1.  1.  0.  0.]
 [ 0.  0.  0. -1. -1. -1. -1.  0.  0.  0.]
 [ 0.  0.  0. -1. -1. -1. -1. -1.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

②

```
[[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  1.  0. -1.  0.  0.  0.  0.  0.]
 [ 0.  0.  1.  1. -1. -1.  0.  0.  0.  0.]
 [ 0.  0.  1.  1.  0. -1. -1.  0.  0.  0.]
 [ 0.  0.  1.  1.  0.  0. -1. -1.  0.  0.]
 [ 0.  0.  1.  1.  0.  0.  0. -1. -1.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

③

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 1. 0. 0.]
 [0. 0. 0. 1. 1. 1. 1. 0. 0. 0.]
 [0. 0. 0. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

④

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 1. 1. 1. 1. 0. 0. 0. 0.]
 [0. 0. 1. 1. 0. 1. 1. 0. 0. 0.]
 [0. 0. 1. 1. 0. 0. 1. 1. 0. 0.]
 [0. 0. 1. 1. 0. 0. 0. 1. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

⑤

```
[[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. -1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. -1. -1.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. -1. -1.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0. -1. -1.  0.  0.]
 [ 0.  0.  0. -1. -0. -0. -0. -0. -0.  0.]
 [ 0.  0.  0. -0. -0. -0. -0. -0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

⑥

```
[[0.   0.   0.15 0.25 0.15 0.   0.   0.   0.   0.  ]
 [0.   0.   0.2  0.4  0.32 0.08 0.   0.   0.   0.  ]
 [0.   0.   0.2  0.53 0.6  0.32 0.08 0.   0.   0.  ]
 [0.   0.   0.08 0.32 0.6  0.6  0.32 0.08 0.   0.  ]
 [0.   0.   0.   0.08 0.32 0.6  0.6  0.32 0.08 0.  ]
 [0.   0.   0.08 0.2  0.35 0.6  0.73 0.48 0.12 0.  ]
 [0.   0.   0.2  0.53 0.73 0.8  0.8  0.52 0.15 0.  ]
 [0.   0.   0.2  0.53 0.73 0.73 0.65 0.4  0.12 0.  ]
 [0.   0.   0.08 0.2  0.27 0.27 0.27 0.2  0.08 0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]]
```

⑦

```
[[0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.08 0.12 0.15 0.12 0.08 0.   0.   0.   0.  ]
 [0.   0.2  0.4  0.52 0.48 0.32 0.08 0.   0.   0.  ]
 [0.   0.27 0.65 0.8  0.73 0.6  0.32 0.08 0.   0.  ]
 [0.   0.27 0.73 0.8  0.6  0.6  0.6  0.32 0.08 0.  ]
 [0.   0.27 0.73 0.73 0.35 0.32 0.6  0.6  0.32 0.15]
 [0.   0.2  0.53 0.53 0.2  0.08 0.32 0.53 0.4  0.25]
 [0.   0.08 0.2  0.2  0.08 0.   0.08 0.2  0.2  0.15]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]]
```

⑧

```
[[ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.   -0.08 -0.12 -0.08  0.    0.    0.    0.  ]
 [ 0.    0.    0.   -0.2  -0.4  -0.32 -0.08  0.    0.    0.  ]
 [ 0.    0.    0.   -0.2  -0.53 -0.6  -0.32 -0.08  0.    0.  ]
 [ 0.    0.    0.   -0.08 -0.32 -0.6  -0.6  -0.32 -0.08  0.  ]
 [ 0.    0.   -0.08 -0.12 -0.15 -0.32 -0.53 -0.4  -0.12  0.  ]
 [ 0.    0.   -0.12 -0.2  -0.12 -0.08 -0.2  -0.2  -0.08  0.  ]
 [ 0.    0.   -0.08 -0.12 -0.08  0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]]
```

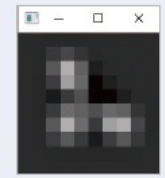
⑨

```
[[ 0.    0.   -0.   -0.   -0.    0.    0.    0.    0.    0.  ]
 [ 0.   -0.    0.02  0.04  0.02 -0.    0.    0.    0.    0.  ]
 [ 0.   -0.    0.07  0.19  0.08 -0.02 -0.    0.    0.    0.  ]
 [ 0.   -0.    0.03  0.17  0.09 -0.06 -0.02 -0.    0.    0.  ]
 [ 0.   -0.   -0.02  0.02  0.05 -0.06 -0.06 -0.02 -0.    0.  ]
 [ 0.   -0.    0.02  0.09  0.08  0.05  0.09  0.08  0.02 -0.  ]
 [ 0.   -0.    0.07  0.19  0.09  0.02  0.17  0.19  0.04 -0.  ]
 [ 0.   -0.    0.03  0.07  0.02 -0.02  0.03  0.07  0.02 -0.  ]
 [ 0.    0.   -0.   -0.   -0.   -0.   -0.   -0.   -0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]]
```

⑩

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 9. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]
 [0. 0. 0. 1. 1. 1. 1. 0. 0. 0.]
 [0. 0. 0. 9. 1. 1. 1. 9. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

⑪



2. 스케일 불변한 지역 특징

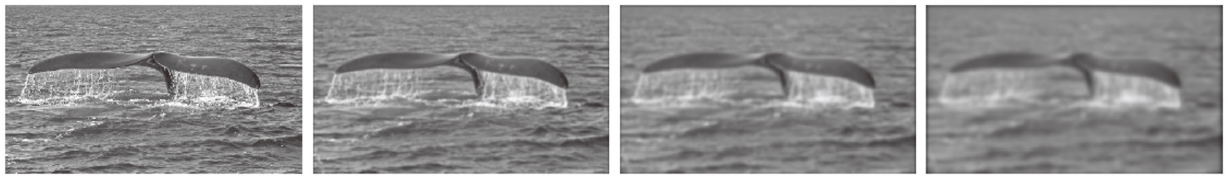
스케일 불변한 지역 특징

- 사람은 스케일 불변의 특징 사용
  - 거리에 상관없이 같은 물체를 같은 물체라고 인식
  - 거리에 따라 세세한 내용의 차이만 있음

- 스케일 공간 (scale space)이론
  - 스케일 공간에서 특징점 검출 ➔ 스케일에 관계없이 특징점을 검출할 수 있음
  - 전략
    - 1. 입력 영상 f로 부터 다중 스케일 영상 $\tilde{f}$를 구성
    - 2. $\tilde{f}$에 적절한 미분 연산을 적용하여 다중 스케일 미분영상 $\tilde{f}'$를 계산
    - 3. $\tilde{f}'$에서 극점을 찾아 특징점으로 취함

다중 스케일 영상의 구현 방법

- 입력 영상 (명암 영상) 한 장에 대해 여러 거리에서 보았을 때 나타나는 현상을 흉내 내는 것 처럼 구성
- Gaussian smoothing를 활용한 방법
  - 스케일에 해당하는 σ를 연속 공간에 정의



(a) 가우시안 스무딩 방법

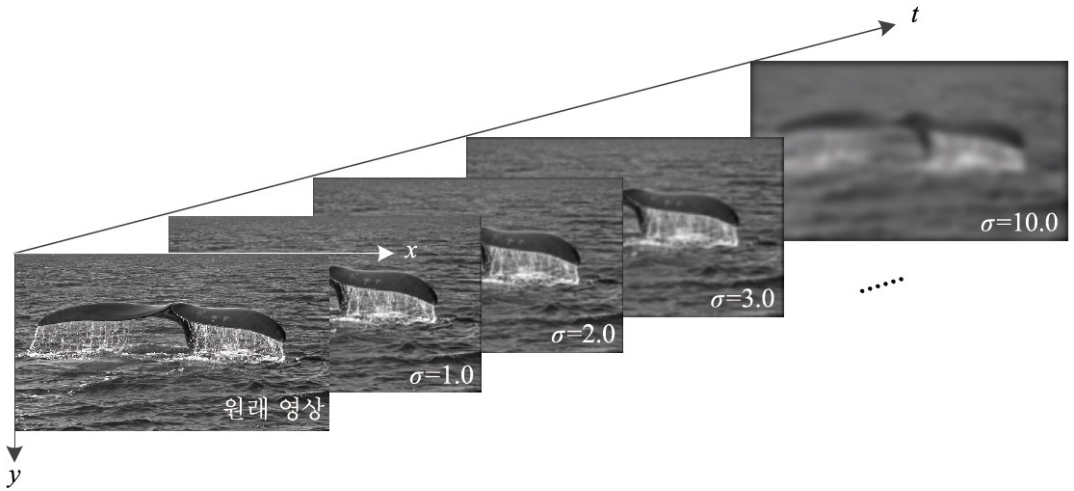
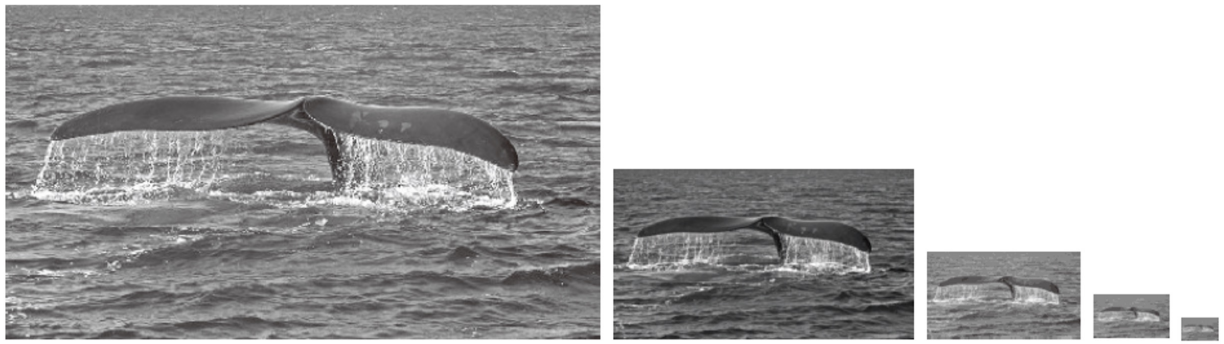


그림 5-8 스케일 공간 (y, x, t)에 정의된 다중 스케일 영상[오일석2014]

- 피라미드 방법
  - 이미지의 스케일 자체를 ½씩 줄임

    (이산적이라는 단점이 존재)

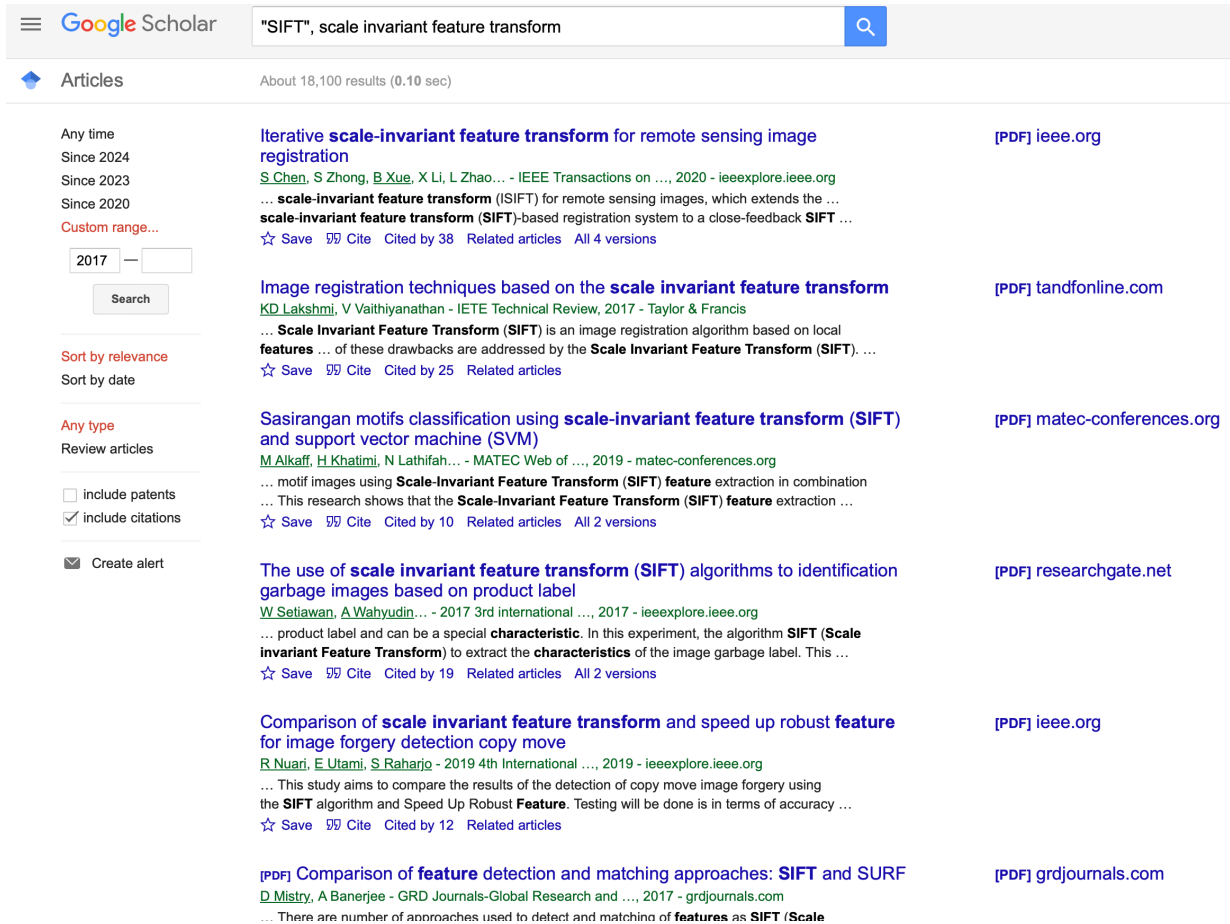


(b) 피라미드 방법

SIFT

- Scale-Invariant Feature Transform (SIFT)
  - 스케일 공간에서 스케일 불변인 특징점 (keypoint)을 검출하는 가장 성공적인 방법
  - 1999년 Prof. David Lowe
  - 2004년 IJCV에 확장된 논문 발표
  - 매우 뛰어난 성능
  - 현재에도 널리 사용
  - 다양한 변형

Distinctive image features from scale-invariant keypoints
DG Lowe - International journal of computer vision, 2004 - Springer
Abstract This paper presents a method for extracting distinctive invariant features from images that can be used to perform reliable matching between different views of an object or scene. The features are invariant to image scale and rotation, and are shown to provide ...
25413회 인용 관련 학술자료 전체 247개의 버전 Web of Science: 8507 인용 저장

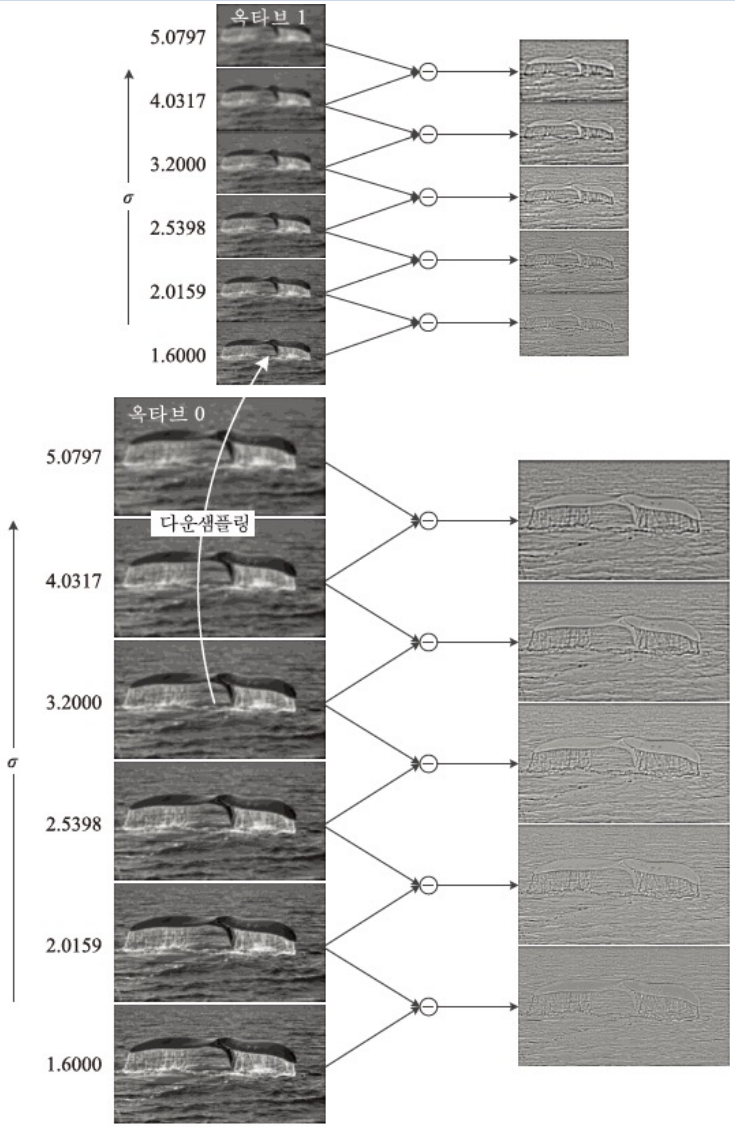


SIFT

- SIFT 알고리즘의 핵심 개념
  - 스케일 불변성
  - 방향 할당 (orientation assignment)
    - 검출된 각 특징점에 대해 주변 픽셀의 gradient 방향을 기반으로 방향을 할당 ➔ 회전에 대한 불변성 제공
  - 특징 기술자 (feature descriptor)
    - 각 특징점 주변의 지역적인 gradient 정보를 기반으로 특징 기술자를 생성
    - 특징점의 지역적인 특성을 요약한 것 ➔ 영상 매칭에서 사용

SIFT 검출

- 1단계: 다중 스케일 영상 구축
  - Gaussian smoothing과 Pyramid 방법을 결합해서 사용
  - 각 층은 여섯 영상의 묶음(Octave)로 구성
    - Octave이 영상은 σ_i로 smoothing
    - $\sigma_{i+1} = k\sigma_i$



(a) 다중 스케일 영상 (b) DOG

그림 5-9 SIFT의 다중 스케일 영상[오일석2014]

SIFT 검출

- 2단계: 다중 스케일 영상에 미분 적용
  - 참고: Laplacian, Normalized Laplacian, DOG
    - Laplacian: 함수 f 의 이차 미분의 합
      - $\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = d_{yy} + d_{xx}$ (∇^2: Laplacian 연산자)
    - Normalized Laplacian
      - $\nabla^2_{norm} f = \sigma^2 \left| \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \right| = \sigma^2 \left| d_{yy} + d_{xx} \right|$
    - DOG (Difference of Gaussian)
      - $DOG(\sigma_i) = G(\sigma_{i+1}) \odot f - G(\sigma_i) \odot f$

        $= G(k\sigma_i) \odot f - G(\sigma_i) \odot f = \big(G(k\sigma_i) - G(\sigma_i)\big) \odot f$
      - 이웃한 스케일의 영상과의 차이 ➔ 빠름

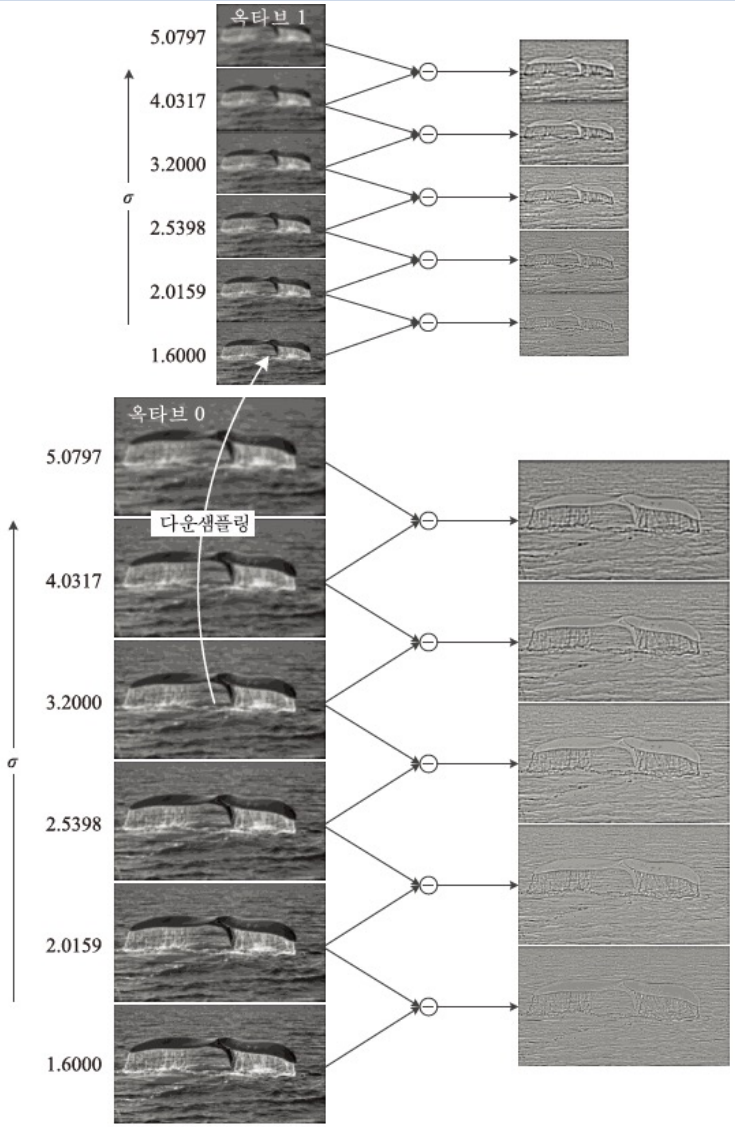
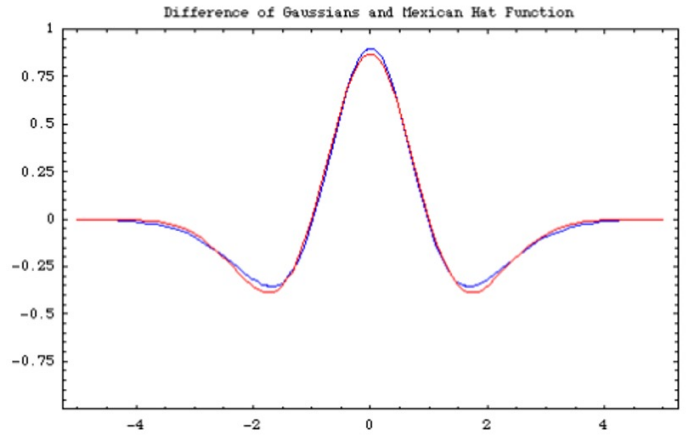


그림 5-9 SIFT의 다중 스케일 영상[오일석2014]

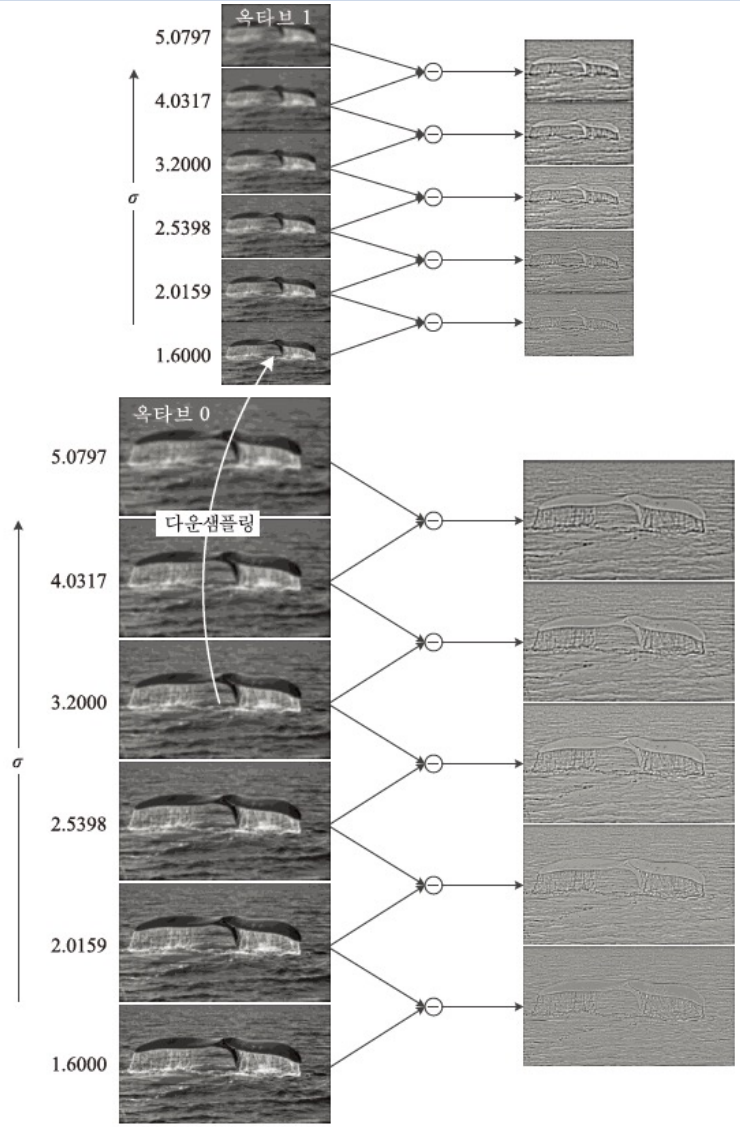
SIFT 검출

- 2단계: 다중 스케일 영상에 미분 적용
  - SIFT에서는 Laplacian 대신 DOG를 활용하여 시간 단축



DOG

정규 라플라시안



(a) 다중 스케일 영상 (b) DOG

그림 5-9 SIFT의 다중 스케일 영상[오일석2014]

SIFT 검출

- 3단계: 극점 검출
  - 미분한 다중 스케일 영상에서 극점 검출
  - 극점 == 특징점 (keypoint)
  - 3차원에서 NMS 적용
    - why 3차원? ➔ 2차원+scale차원
    - 주위 26개 이웃에 대해 NMS 적용

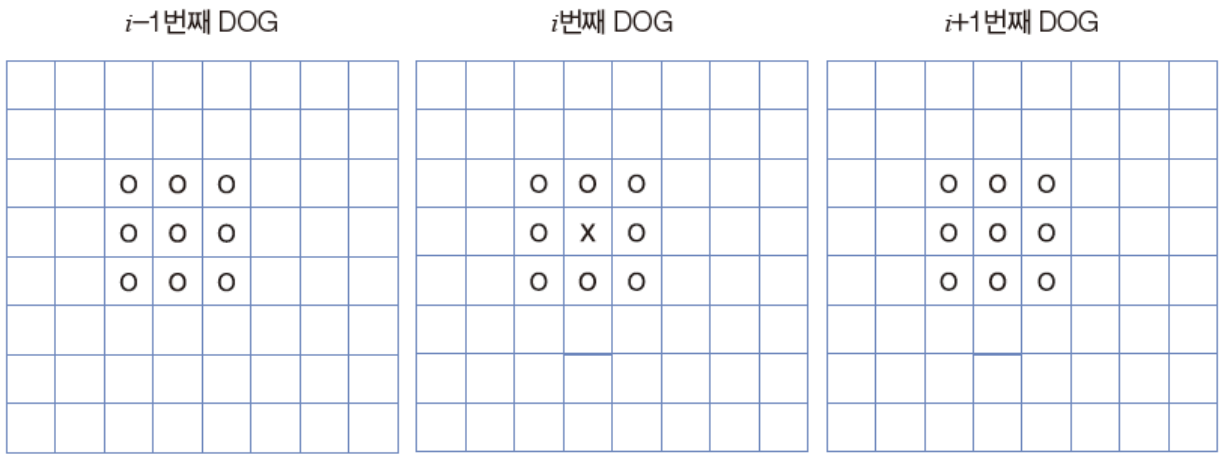
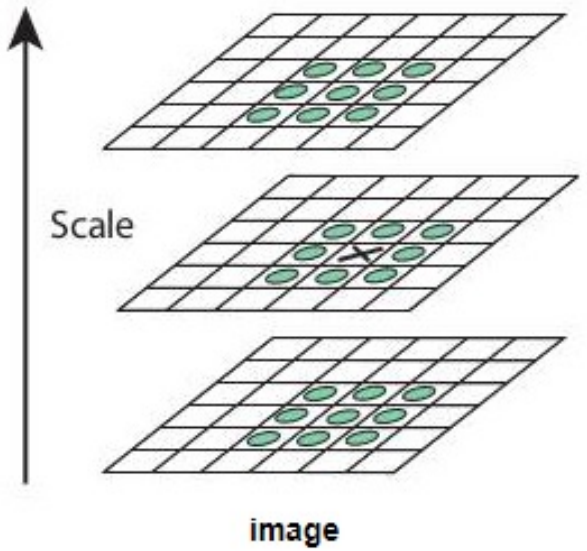


그림 5-10 3차원 구조의 DOG 영상에서 특징점(키포인트) 검출

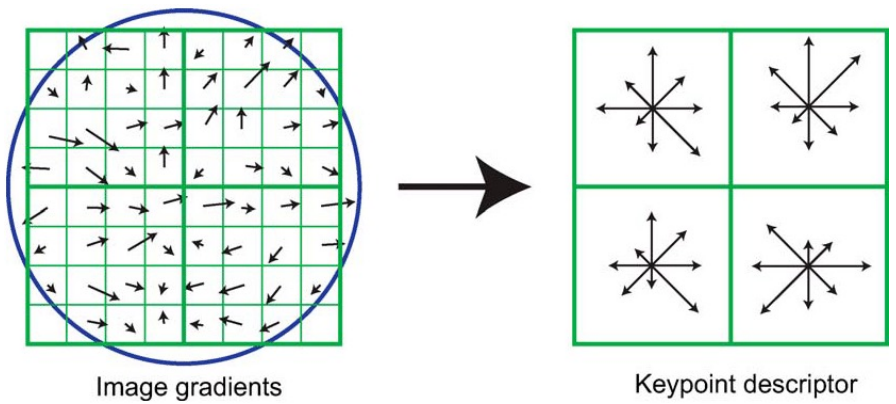


SIFT 검출

- 특징점 (keypoint)의 표현
  - $(pt, size, angle, response, octave, class\_id \ldots)$
    - pt : keypoint (x, y) 좌표
    - $size$: keypoint 직경
    - $angle$: keypoint 방향 (각도)
    - $response$: keypoint 응답 강도 (keypoint가 얼마나 두드러졌는지를 나타냄)
    - $octave$: keypoint가 발견된 이미지 피라미드의 옥타브
    - $class\_id$: keypoint가 속한 클래스 ID (일반적으로 사용되지 않음)

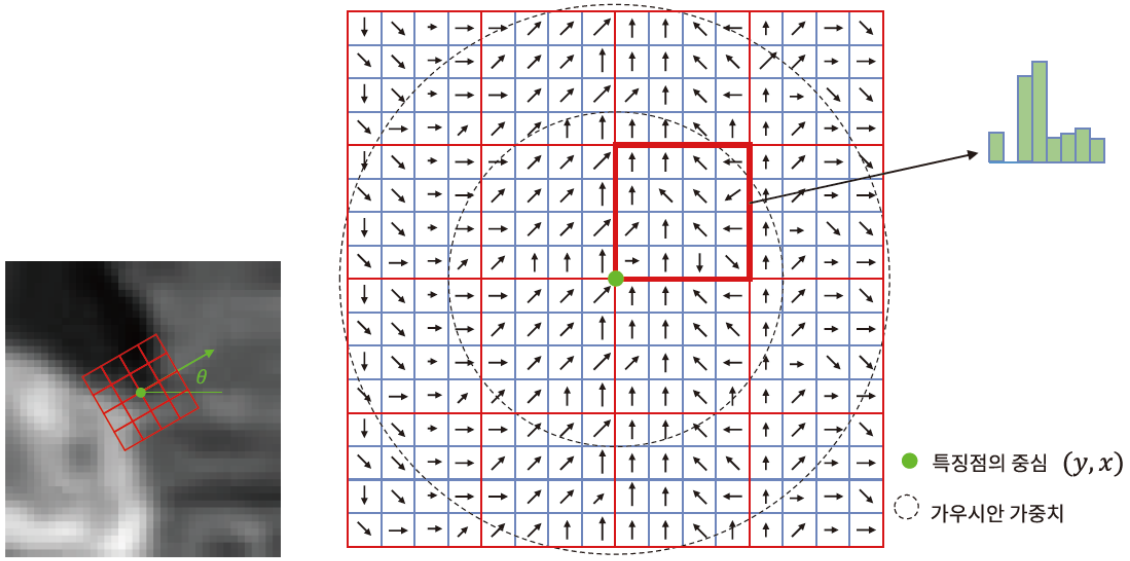
SIFT 기술자

- 기술자 (Descriptor)
  - 특징점(특징 벡터)에 대한 정보를 의미
  - 특징점과 그 주변에 대한 풍부한 정보를 포함
  - 불변성 달성이 매우 중요



SIFT 기술자

- 기술자 추출 알고리즘
  - o와 i로 가장 가까운 Gaussian을 결정한 후 기술자 추출 ➔ scale 불변성 달성
  - 기준 방향을 지정한 후 기준 방향을 중심으로 특징 추출 ➔ 회전 불변성 달성
  - Descriptor의 단위 벡터화 ➔ 조명 불변성 달성
- ➔ SIFT의 기술자는 128차원 벡터로 추출됨



(a) 지배적인 방향의 윈도우 (b) 16×16 부분 영역 샘플링과 기술자 추출

그림 5-11 SIFT 특징점에서 기술자 추출

프로그래밍 실습: SIFT 검출

프로그램 5-2 SIFT 검출

```
01  import cv2 as cv
02
03  img=cv.imread('mot_color70.jpg')              # 영상 읽기
04  gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)
05
06  sift=cv.SIFT_create()
07  kp,des=sift.detectAndCompute(gray,None)
08
09  gray=cv.drawKeypoints(gray,kp,None,flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_
    KEYPOINTS)
10  cv.imshow('sift', gray)
11
12  k=cv.waitKey()
13  cv.destroyAllWindows()
```

3. 매칭

Getting Started

- 아래 두 사진이 같은/다른 사진인지 판단하는 방법은?

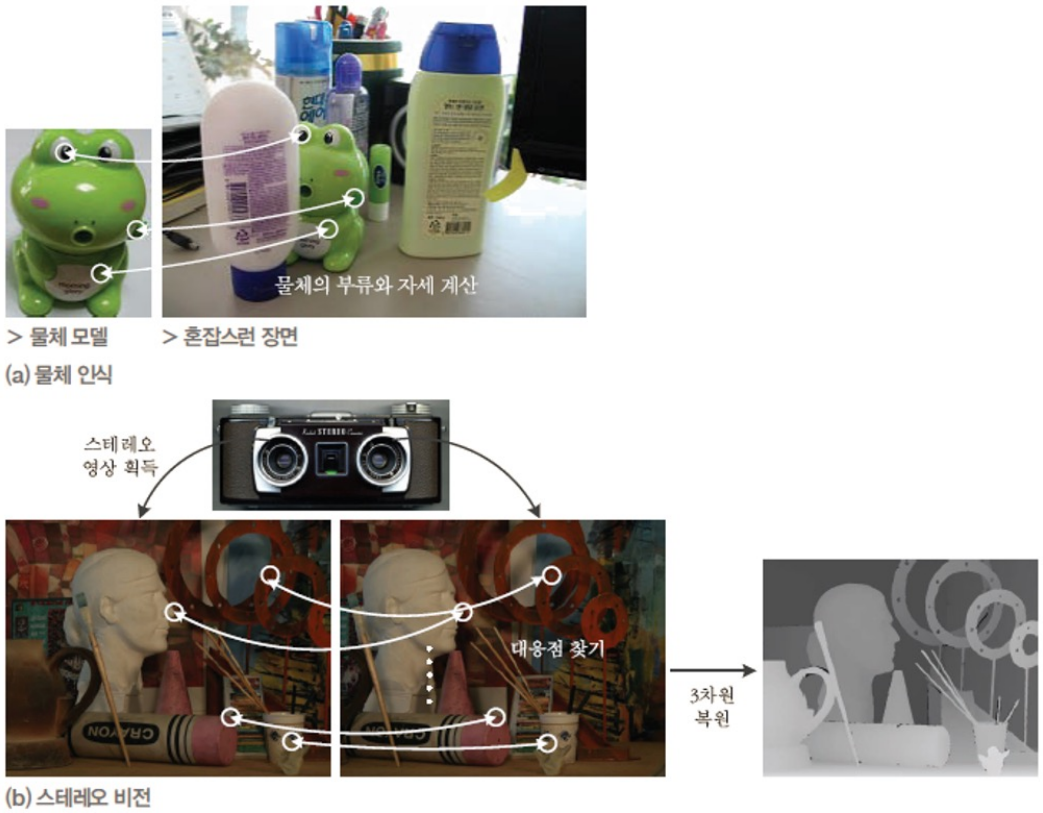
- 같은 위치에 있는 요소(특징)가 모두 같으면 같은 사진

특징: edge, local feature, region, etc.

Matching

매칭 전략

- Matching
  - 어떤 대상을 다른 것과 비교하여 같은 것인지 알아내는 과정
  - 매우 까다로운 문제
    - 두 영상 간 특징점 후보 쌍 다수
    - Noise가 섞인 기술자



매칭 전략

- 문제의 이해
  - 두 영상에서 추출한 집합 $A = \{\mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_m}\}$와 $B = \{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_n}\}$에서 같은 물체의 같은 곳에서 추출된 $\mathbf{a_i}$와 $\mathbf{b_i}$ 쌍을 모두 찾는 문제

- 매칭을 적용하는 다양한 상황
  - 물체 인식: 물체 영상 A, 장면 영상 B
  - 물체 추적 또는 스테레오: 두 영상이 동등한 입장

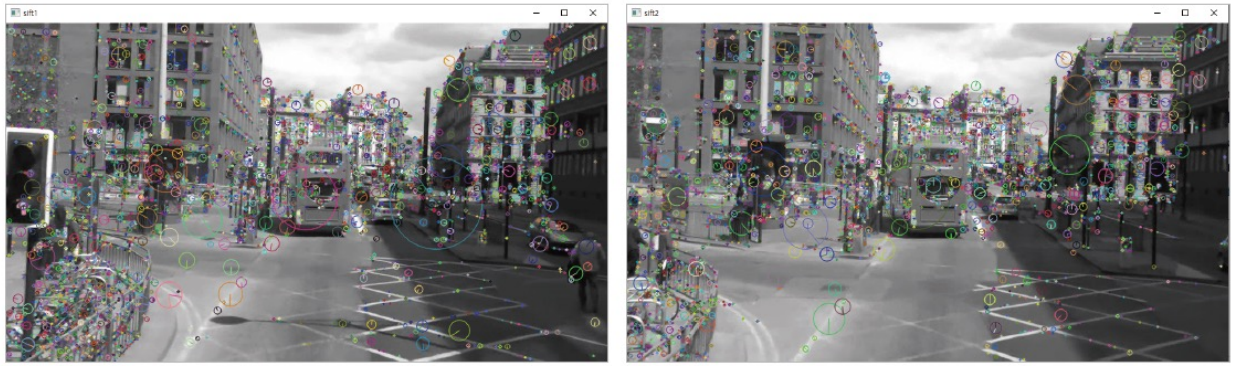


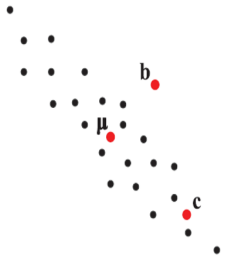
그림 5-12 두 장의 영상에서 추출한 SIFT 특징점을 어떻게 매칭할까?

- 가장 쉬운 접근 방법
  - mn개 쌍 각각에 대해 거리를 계산하고, 거리가 임계값 보다 작은 쌍을 모두 취함

매칭 전략

- 두 기술자(특징 벡터)간 거리 지표
  - Manhattan distance, Euclidean distance, Mahalanobis distance, Correlation distance, etc.

$$d(\mathbf{a},\mathbf{b}) = \sqrt{\sum_{k=1,d}(a_k - b_k)^2} = \|\mathbf{a}-\mathbf{b}\| \qquad (5.12)$$



← μ는 b와 c 중 누구와 더 가까운가?

- 임계값 구성 방법
  - 고정 임계값: 거리를 기준으로 특정 임계값 이하인 경우 모든 쌍을 매칭
  - 최근접 이웃: a_i는 가장 가까운 b_i를 찾고, $d(a_i, b_i) < T$를 만족하면 매칭
  - 최근접 이웃 거리 비율: 최근접 b_i와 두 번째 최근접 b_k가 $\frac{d(a_i,b_i)}{d(a_i,b_k)} < T$를 만족하면 매칭
    - ➔ 실험에 따르면 이 방법이 가장 좋은 성능을 보임
    - ➔ 어떻게 최근접 이웃 거리 비율을 빠르게 찾을 것인가?

매칭 성능 측정

- 성능 측정은 컴퓨터 비전에서 아주 중요한 관점임
  - 알고리즘 개선이나 최선의 알고리즘을 선택하는 기준
  - 현장 투입 여부를 결정하는 기준

- 성능 지표
  - Precision (정밀도) $= \frac{TP}{TP+FP}$
  - Recall (재현율, 민감도(Sensitivity)) $= \frac{TP}{TP+FN}$
  - Specificity (특이도) $= \frac{TN}{FP+TN}$
  - F1 score $= \frac{2*Precision*Recall}{Precision+Recall}$
  - Accuracy $= \frac{TP+TN}{TP+TN+FP+FN}$

		Predicted	
		Negative (**N**) -	Positive (**P**) +
Actual	Negative -	True Negatives (TN)	False Positives (FP) **Type I error**
	Positive +	False Negatives (FN) **Type II error**	True Positives (TP)

매칭 성능 측정

- 성능 지표
  - ROC (Receiver Operating Characteristics) 곡선 및 AUC (Area Under Curve)
    - X축(FPR): 1-specificity = $1 - \frac{TN}{FP+TN} = \frac{FP+TN-TN}{FP+TN} = \frac{FP}{FP+TN}$
      - FP = positive를 잘못 예측함 ➔ 실제로 negative
      - TN = negative를 negative로 잘 예측함 ➔ 실제로 negative
      - TPR = 실제로 negative인 것들 중에 positive로 잘못 예측한 비율
    - Y축(TPR): sensitivity $\frac{TP}{TP+FN}$
      - Sensitivity: True를 True로 잘 예측?
      - TP = positive를 positive로 잘 예측함 ➔ 실제로 positive
      - FN = negative를 잘못 예측함 ➔ 실제로 positive
      - FPR = 실제로 positive인 것들을 positive로 옳게 예측한 비율

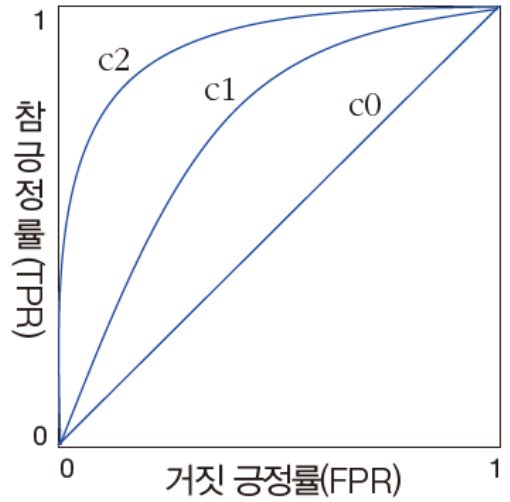
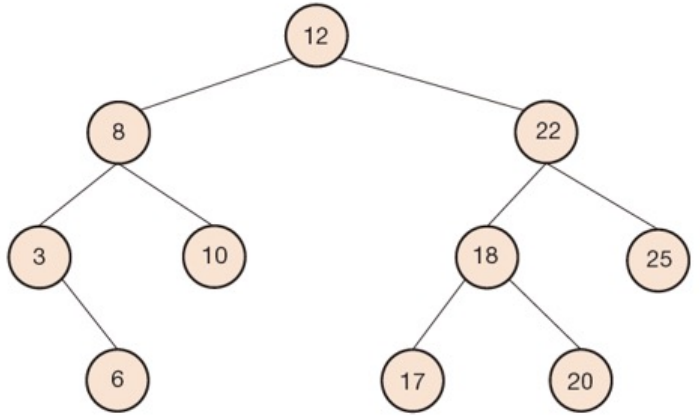


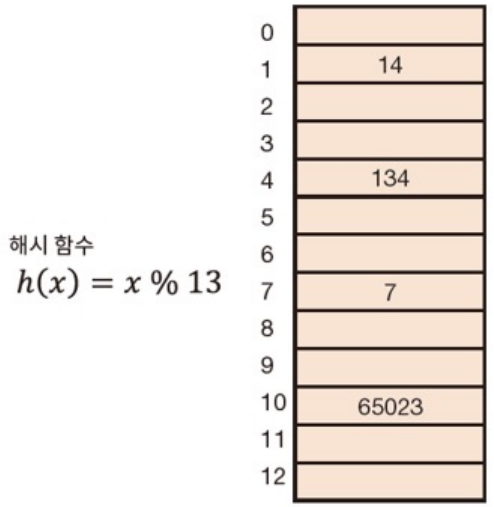
그림 5-15 ROC 곡선

매칭 성능 측정

- 성능 지표
  - 속도
    - 실시간성이 요구되는 응용에서 양보할 수 없는 강한 조건
    - Computer science에서 실시간성 ➔ “주어진 시간 내 처리”
    - Data structure 기반의 매칭 기법 ➔ 빠르게 최근접 이웃 비율 찾기
      - *k*d tree
      - hashing



(a) 이진 탐색 트리



(b) 해싱

(참고) *k*d tree

- 풀어야 하는 문제
  - 특징점 (keypoint)들의 최근접 이웃을 찾아야함
    - 특징점 (keypoint): 여러 값으로 구성된 벡터 형태
  - 특징점 매칭의 독특한 성질로 인해 이진 탐색 트리 (BST)에 그대로 적용할 수 없음

- kd tree는 이런 경우에 적합한 자료구조
  - 1) *k*d tree 만들기
  - 2) *k*d tree 탐색

(참고) *k*d tree

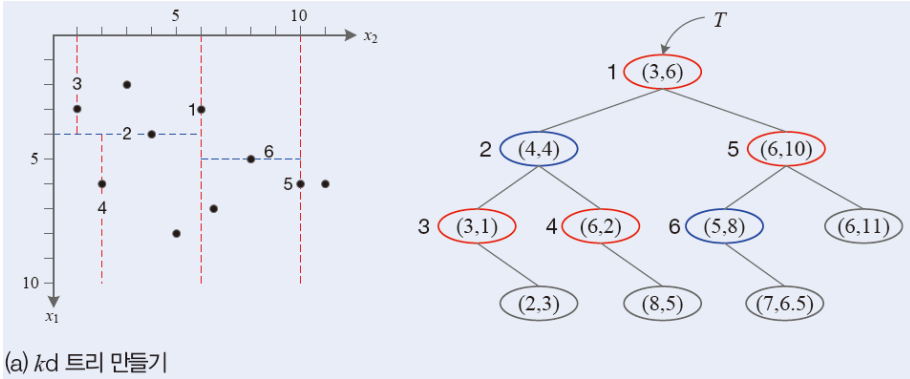
- Formulation
  - *k*d트리를 구성하는 n개의 벡터 $X = \{\mathbf{x_1}, \mathbf{x_2}, \dots, \mathbf{x_n}\}$; 각 $\mathbf{x}_i$는 d차원 벡터

- Algorithm
  - Root node는 X를 두 개의 부분집합 X_{left}, X_{right}로 분할
  - 분할 기준
    - 1) d개의 차원(축) 중 축 선택
      - ➔ 분할 효과를 극대화하려면 각 차원의 최대 분산을 갖는 축 *k*를 선택
    - 2) 축 선택 후 n개의 샘플 중 어떤 것을 기준으로 X를 분할할 것인가?
      - ➔ X_{left}, X_{right}의 크기 차이를 최소화하여 균형 잡힌 트리를 구성해야 함
      - X를 차원 *k*로 정렬 후 그 결과의 중앙값을 분할 기준으로 설정
    - 과정 1, 2를 재귀적으로 반복하여 *k*d트리 생성

(참고) kd tree

- 예시
  - $d = 2$인 특징 벡터 X

    $X = \{\mathbf{x_1} = (3,1), \mathbf{x_2} = (2,3), \mathbf{x_3} = (6,2), \mathbf{x_4} = (4,4), \mathbf{x_5} = (3,6),$
    $\mathbf{x_6} = (8,5), \mathbf{x_7} = (7,6.5), \mathbf{x_8} = (5,8), \mathbf{x_9} = (6,10), \mathbf{x_{10}} = (6,11)\}$



(a) kd 트리 만들기

  - 1) root node로 결정할 만한 분할 기준 탐색
    - 축 결정: 각 차원에 대한 분산 중 분산이 큰 쪽이 축으로 결정됨

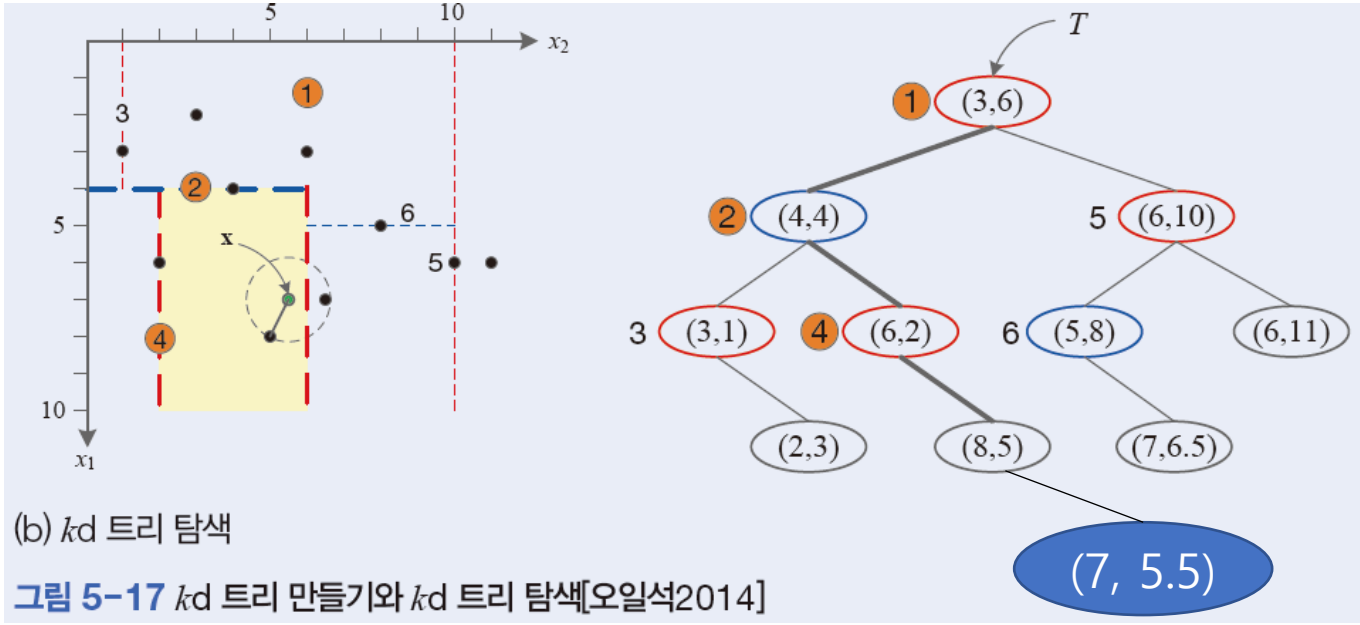
      {3, 2, 6, 4, 3, 8, 7, 5, 6, 6} vs {1, 3, 2, 4, 6, 5, 6.5, 8, 10, 11}

      ➔ 두번째 차원의 축이 더 큼 $k = 2$를 기준으로 정렬
  - 2) 정렬 후 중앙값 기준으로 분할 시작
    - 정렬 결과: $X_{sorted} = \{\mathbf{x_1}, \mathbf{x_3}, \mathbf{x_2}, \mathbf{x_4}, \mathbf{x_6}, \mathbf{x_5}, \mathbf{x_7}, \mathbf{x_8}, \mathbf{x_9}, \mathbf{x_{10}}\}$
    - 중앙값 $(\mathbf{x_5})$기준으로 분할 ➔ $X_{left} = \{\mathbf{x_1}, \mathbf{x_3}, \mathbf{x_2}, \mathbf{x_4}, \mathbf{x_6}\}$, $X_{right} = \{\mathbf{x_7}, \mathbf{x_8}, \mathbf{x_9}, \mathbf{x_{10}}\}$

(참고) kd tree

- 예시
  - 새로운 특징 벡터$\mathbf{x} = (7, 5.5)$입력 시
    - $k = 2$ ➔ x_2 축을 기준으로 5.5와 6 비교

      ➔ 왼쪽으로 ➔ 5.5와 4 비교 ➔ 오른쪽으로… 반복



(b) kd 트리 탐색

그림 5-17 kd 트리 만들기와 kd 트리 탐색[오일석2014]

프로그래밍 실습: FLANN 알고리즘

- Fast Library for Approximate Nearest Neighbors (FLANN) 알고리즘: 빠른 매칭을 보장함

프로그램 5-3 FLANN 라이브러리를 이용한 SIFT 매칭

```
01  import cv2 as cv
02  import numpy as np
03  import time
04
05  img1=cv.imread('mot_color70.jpg')[190:350,440:560] # 버스를 크롭하여 모델 영상으로 사용
06  gray1=cv.cvtColor(img1,cv.COLOR_BGR2GRAY)
07  img2=cv.imread('mot_color83.jpg')                  # 장면 영상
08  gray2=cv.cvtColor(img2,cv.COLOR_BGR2GRAY)
09
10  sift=cv.SIFT_create()
11  kp1,des1=sift.detectAndCompute(gray1,None)
12  kp2,des2=sift.detectAndCompute(gray2,None)
13  print('특징점 개수:',len(kp1),len(kp2)) ①
14
```

프로그래밍 실습: FLANN 알고리즘

- Fast Library for Approximate Nearest Neighbors (FLANN) 알고리즘: 빠른 매칭을 보장함

```
15  start=time.time()
16  flann_matcher=cv.DescriptorMatcher_create(cv.DescriptorMatcher_FLANNBASED)
17  knn_match=flann_matcher.knnMatch(des1,des2,2)
18
19  T=0.7
20  good_match=[]
21  for nearest1,nearest2 in knn_match:
22      if (nearest1.distance/nearest2.distance)<T:
23          good_match.append(nearest1)
24  print('매칭에 걸린 시간:',time.time()-start) ②
25
26  img_match=np.empty((max(img1.shape[0],img2.shape[0]),img1.shape[1]+img2.
    shape[1],3),dtype=np.uint8)
27  cv.drawMatches(img1,kp1,img2,kp2,good_match,img_match,flags=cv.
    DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
28
29  cv.imshow('Good Matches', img_match)
30
31  k=cv.waitKey()
32  cv.destroyAllWindows()
```

$\frac{d(a_i,b_i)}{d(a_i,b_k)} < T$ 최근접 이웃 거리 비율 적용

```
특징점 개수: 231 4096 ①
매칭에 걸린 시간: 0.03124260902404785 ②
```

End of slide