2주차 chap05

연습문제_홀수

Chap 5. 지역 특징

5.1 발상

대응점 문제 : 두 영상에서 특징점을 추출하고 매칭을 통해 해당하는 특징점 쌍 찾기

아래 조건을 만족해야 대응점 문제를 푸는 데 유용 = 특징점 검출
충돌되는 부분이 있기에 적절히 조절이 필요

- **반복성**repeatability 같은 물체가 서로 다른 두 영상에 나타났을 때 첫 번째 영상에서 검출된 특징점이 두 번째 영상에서도 같은 위치에서 높은 확률로 검출되어야 한다.
- **불변성**invariance 물체에 이동, 회전, 스케일, 조명 변환이 일어나도 특징 기술자의 값은 비슷해야 한다. 불변성을 만족해야 다양한 변환이 일어난 상황에서도 매칭에 성공할 수 있기 때문이다. 이동과 회전에 대한 불변성은 5.2절, 스케일에 대한 불변성은 5.3~5.4절에서 다룬다.
- **분별력**discriminative power 물체의 다른 곳에서 추출된 특징과 두드러지게 달라야 한다. 그렇지 않다면 물체의 다른 곳에서 추출된 특징과 매칭될 위험이 있다.
- **지역성**locality 작은 영역을 중심으로 특징 벡터를 추출해야 물체에 가림occlusion이 발생해도 매칭이 안정적으로 동작한다.
- **적당한 양** 물체를 추적하려면 몇 개의 대응점만 있으면 된다. 하지만 대응점은 오류를 내포할 가능성이 있기 때문에 특징점이 더 많으면 더 정확하게 추적할 수 있다. 반면에 특징점이 너무 많으면 계산 시간이 과다해진다.
- **계산 효율** 계산 시간이 매우 중요한 응용이 많다. 예를 들어 선수를 추적하여 정보를 자동으로 표시하는 축구 중계의 경우 초당 몇 프레임 이상을 처리해야 하는 실시간 조건이 필수다.

5.2 이동과 회전 불변한 지역 특징

모라벡 알고리즘

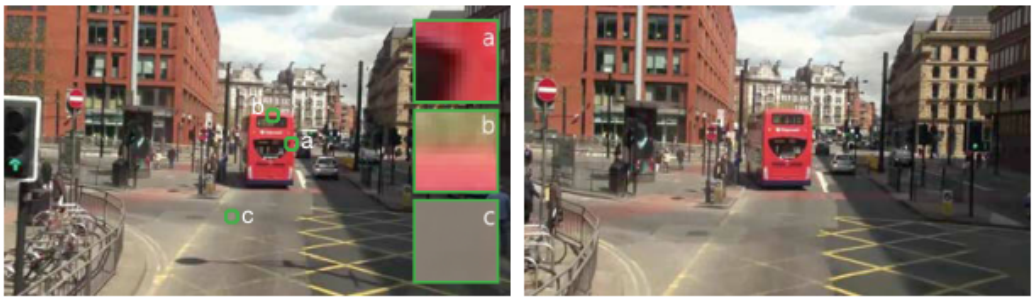


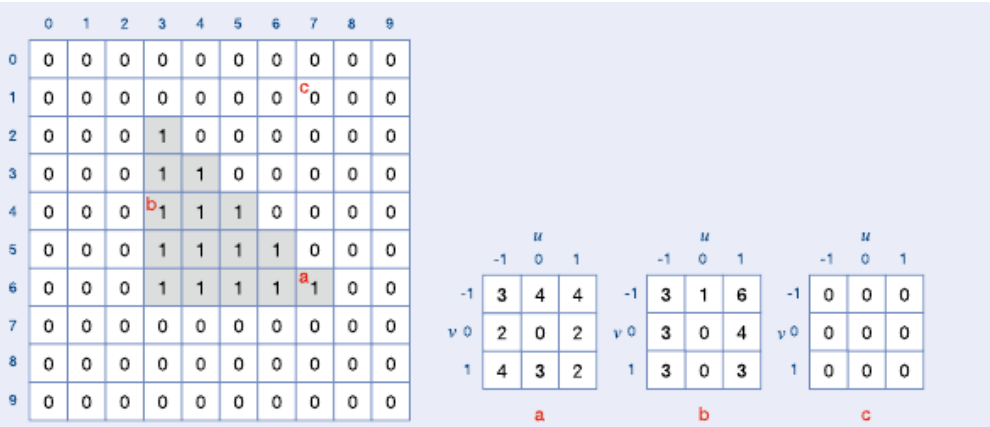
그림 5-3 대응점 찾기 인지 실험(MOT-17-14-SDP 동영상의 70번째와 83번째 영상)

a. 여러 방향으로 색상 변화가 있어 찾기 쉬움

b. 어느 방향으로도 밝기 변화가 미세해 찾기 어려움

제곱차의 합 SSD : 영상의 특징점 중 어느 점이 찾기 쉬운지, 어려운지 **찾기 쉬운 정도를 측정**하는 데 쓸 수 있음, 각 화소 v와 u를 각각 -1,0,1로 변화시켜 맵 생성

단점 : 현실적이지 않고 실제 영상은 단순하지 않기에 상하좌우 이웃만 보고 점수 부여는 한계



(a) 원래 영상 (b) 세 점에서 추출한 S 맵

그림 5-4 제곱차의 합 계산

아래 식은 v는 0, u는 1인 경우인 $S(0,1)$을 계산하는 과정이다.

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

(v,u)의 나머지 칸을 계산해 3×3 맵에 채우면 점 b의 맵은 [그림 5-4(b)]의 가운데가 된다.

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

▼ 지역 특징으로 **좋은 정도 측정**

$$C=\min\left(S(0,1),S(0,-1),S(1,0),S(-1,0)\right)$$

a. 원래 영상에서 모든 방향으로 변화가 있어 S 맵에서 8이웃이 모두 큰 값 보유

   → 지역 특징으로 손색 없음

b. 원래 영상에서 수직 방향으로 변화가 없고 수평 방향으로 변화가 있음, S맵에서 상하에 위치한 이웃 작은 값, 좌우 위치한 이웃 큰 값

   → 지역 특징으로 부족

c. 모든 방향에서 변화가 없어 S 맵의 모든 요소가 0

   → 지역 특징으로 자격 X

해리스 알고리즘

1. 가중치 제곱차의 합 (잡음 대처를 위해 가우시안 G 추가 적용)
2. 테일러 확장까지 따라 각 x,y 방향의 미분값도 적용

$$S(v,u)\cong\sum_y\sum_x G(y,x)\left(vd_y+ud_x\right)^2=\sum_y\sum_x G(y,x)\left(v^2d_y^2+2vud_yd_x+u^2d_x^2\right)$$

이후 컨볼루션 연산자 이용해 가중치 제곱차의 합은 행렬 A 도출 가능

행렬 A : 2차 모멘트 행렬

$$\mathbf{A}=\begin{pmatrix} G\circledast d_y^2 & G\circledast d_yd_x \\ G\circledast d_yd_x & G\circledast d_x^2\end{pmatrix}\qquad(5.7)$$

1. **실수**의 가중치 제곱차의 합 도출 가능
2. 어떤 화소 주위의 영상 구조를 표현하고 있어 행렬 A만 분석하면 **지역 특징여부 판단**

   고유값 이용해 적용 가능, 상수 k = 0.04 설정이 적절

$$C=\lambda_1\lambda_2-k\left(\lambda_1+\lambda_2\right)^2\qquad(5.8)$$

$$C=\left(pq-r^2\right)-k\left(p+q\right)^2 \qquad (5.9)$$

행렬의 고윳값 특징이용해서 행렬값으로 수식 대치

→ 맵을 생성하는 계산 과정 거치지 않아도 됨

3. 고윳값을 보고 **지역 특징으로 좋은 정도** 측정하는 기법 제안
   - 모두 0에 가까우면 특징으로 가치 없음
   - 하나는 큰데 다른 하나는 작은 경우 한 방향으로만 변화 있음
   - 모두 고윳값이 크면 지역 특징으로 good

	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

```
import cv2 as cv
import numpy as np

img=np.array([[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],
              [0,0,0,1,1,1,0,0,0,0],
              [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],
              [0,0,0,0,0,0,0,0,0,0]],dtype=np.float32) # 입력영상 생성

# 미분 필터 생성
ux=np.array([[-1,0,1]])
uy=np.array([-1,0,1]).transpose()
# 가우시안 필터 생성
k=cv.getGaussianKernel(3,1)
g=np.outer(k,k.transpose())

# dy, dx 도출
dy=cv.filter2D(img,cv.CV_32F,uy)
dx=cv.filter2D(img,cv.CV_32F,ux)

dyy=dy*dy
dxx=dx*dx
dyx=dy*dx

gdyy=cv.filter2D(dyy,cv.CV_32F,g)
gdxx=cv.filter2D(dxx,cv.CV_32F,g)
gdyx=cv.filter2D(dyx,cv.CV_32F,g)
```

```
# 특징가능성 맵 도출
C=(gdyy*gdxx-gdyx*gdyx)-0.04*(gdyy+gdxx)*(gdyy+gdxx)


# 비최대 억제 사용
for j in range(1,C.shape[0]-1):
    for i in range(1,C.shape[1]-1):
        if C[j,i]>0.1 and sum(sum(C[j,i]>C[j-1:j+2,i-1:i+2]))==8:
          # 극점이 되려면 C가 0.1보다 커야 하며 8개 이웃보다 커야 함
            img[j,i]=9      # 특징점을 원본 영상에 9로 표시

# 특징 가능성 맵 C 계산하는 데 필요한 미분 영상 출력
np.set_printoptions(precision=2)
print(dy)
print(dx)
print(dyy)
print(dxx)
print(dyx)
print(gdyy)
print(gdxx)
print(gdyx)

print(C)          # 특징 가능성 맵
print(img)          # 특징점을 9로 표시한 원본 영상

# 화소 확인 가능하게 16배로 확대해 윈도우에 보임
popping=np.zeros([160,160],np.uint8)
for j in range(0,160):
    for i in range(0,160):
        popping[j,i]=np.uint8((C[j//16,i//16]+0.06)*700)

cv.imshow('Image Display2',popping)
cv.waitKey()
cv.destroyAllWindows()
```

5.3 스케일 불변한 지역 특징

스케일 불변의 가능성을 증폭하게 하는 **스케일 공간 이론**

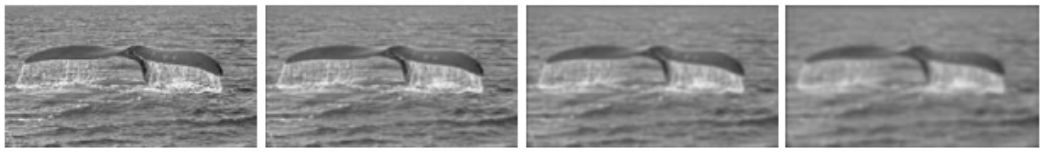
[알고리즘 5-1] 스케일 공간에서 특징점 검출

입력: 명암 영상 f

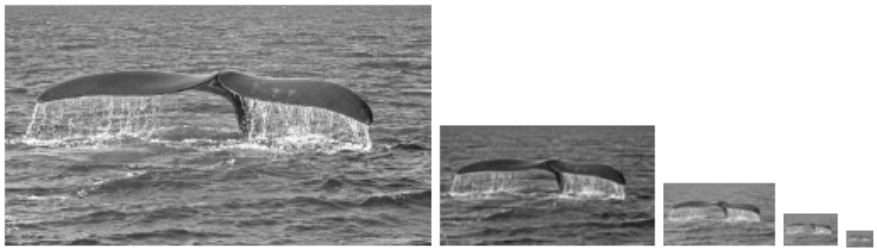
출력: 스케일에 불변한 특징점 집합

1. 입력 영상 f로부터 다중 스케일 영상 $\tilde{f}$를 구성한다.
2. $\tilde{f}$에 적절한 미분 연산을 적용하여 다중 스케일 미분 영상 $\tilde{f}'$를 구한다.
3. $\tilde{f}'$에서 극점을 찾아 특징점으로 취한다.

1. 가까이부터 멀리까지 본 장면 표현 필요



(a) 가우시안 스무딩 방법



(b) 피라미드 방법

1-1. 거리가 멀어지면 세부 내용이 점점 흐려지는 현상 모방

→ 표준편차 키우면서 가우시안 필터로 입력 영상 **스무딩**하여 흐리지는 현상 시물레이션

장점 : 표준편차를 연속된 값으로 조절할 수 있는 장점

1-2. 거리가 멀어짐에 따라 물체의 크기가 작아지는 현상 모방

→ 영상의 크기를 반씩 줄인 영상을 쌓은 **피라미드 영상**으로 이 현상 시물레이션

장점 : 연속 공간에서 유도한 수식과 알고리즘을 디지털 공간으로 변환해 사용 가능

5.4 SIFT

스케일 불변의 가능성을 증폭하게 하는 **스케일 공간 이론** 중 하나 SIFT

- 세 단계에 거쳐 특징점 검출
- 각 단계는 최적의 검출 정확도 달성하고 계산을 최대한 감소하는 연산 사용

1단계 : 다중 스케일 영상 구축

가우시안 + 피라미드로 다중 스케일 영상 구성

방법 : 아래 여섯장은 원래 영상에 가우시안 스무딩 적용 ⇒ 이렇게 적용된 영상 **옥타브**

아래 여섯장 옥타브0, 위 여섯장 옥타브 1

▼ 둘을 결합한 형태

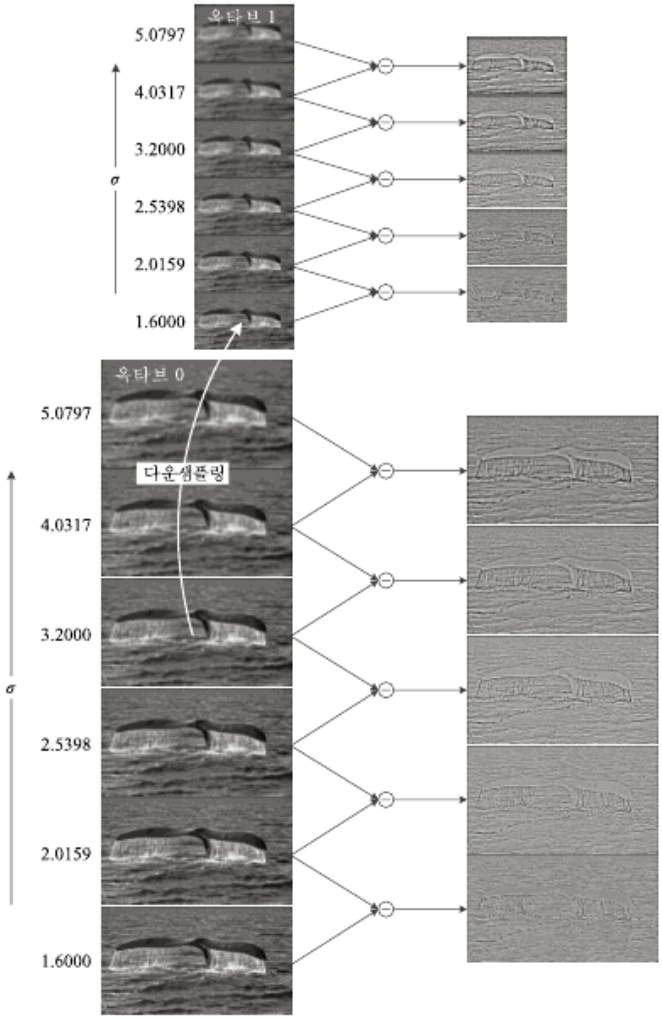


그림 5-9 SIFT의 다중 스케일 영상[오일석2014]

- 옥타브 0
  - 아래 여섯장
  - 원래 영상을 표준편차1 = 1.6 으로 스무딩한 영상에서 출발, 컨볼루션 6번 수행 (표준편차가 스무딩 역할)

    표준편차_(n+1) = k*표준편차n (n = 1,2,3,4,5)

    → n이 증가할수록 필터가 커서 시간이 오래 걸림

- 옥타브 1
  - 위쪽 여섯장
  - 원래 영상을 반으로 줄여 가우시안 적용해 생성
  - 옥타브 1의 첫번째 영상은 옥타브 0에서 3.2로 스무딩한 네번째 영상을 반으로 축소해 얻음

옥타브 0에서 옥타브 1을 얻는 과정을 반복 적용해 옥타브 2, 3, 4, 만들면 다중 스케일 영상 생성 ok

2단계 : 다중 스케일 영상에 미분 적용

== 스케일 정보 알아내기

정규 라플라시안 사용

$$\text{정규 라플라시안: } \nabla^2_{normal} f = \sigma^2 \mid d_{yy} + d_{xx} \mid \qquad (5.11)$$

→ 단점 : 큰 필터로 컨볼루션 수행하여 시간 많이 걸림

단점 보완, 유사기능 보유 **DOG** Different of Gaussian

- 이웃한 영상을 화소별로 빼면 되어 아주 빠르게 계산 가능

3단계 : 극점 검출

== 특징점의 위치 알아내기

i 번째 영상에서 X 표시된 화소의 극점 여부 조사

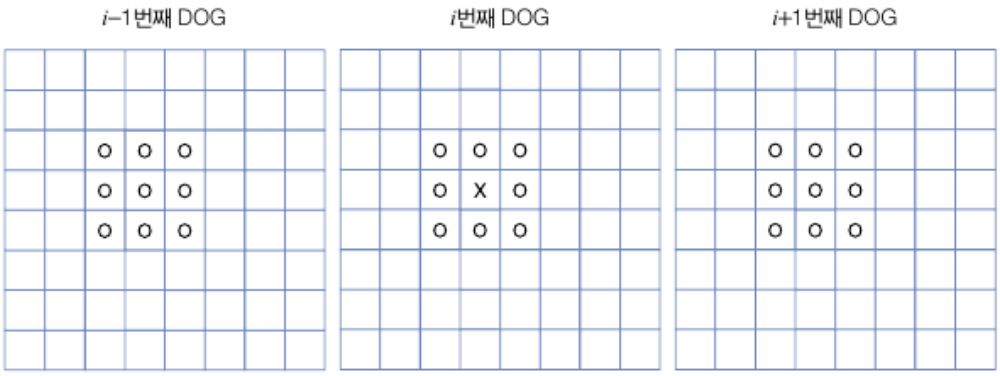


그림 5-10 3차원 구조의 DOG 영상에서 특징점(키포인트) 검출

→ 이웃 영상까지 조사해 18개 이웃 화소 조사

++ 기술사 추출 단계

2단계 3단계로 확인한 위치와 스케일 정보만으로 물체 매칭하는데 정보가 턱없이 부족

→ **특징점 주위를 살펴** 풍부한 정보를 가진 기술자 추출 단계

- **반복성**repeatability 같은 물체가 서로 다른 두 영상에 나타났을 때 첫 번째 영상에서 검출된 특징점이 두 번째 영상에서도 같은 위치에서 높은 확률로 검출되어야 한다.
- **불변성**invariance 물체에 이동, 회전, 스케일, 조명 변환이 일어나도 특징 기술자의 값은 비슷해야 한다. 불변성을 만족해야 다양한 변환이 일어난 상황에서도 매칭에 성공할 수 있기 때문이다. 이동과 회전에 대한 불변성은 5.2절, 스케일에 대한 불변성은 5.3~5.4절에서 다룬다.
- **분별력**discriminative power 물체의 다른 곳에서 추출된 특징과 두드러지게 달라야 한다. 그렇지 않다면 물체의 다른 곳에서 추출된 특징과 매칭될 위험이 있다.
- **지역성**locality 작은 영역을 중심으로 특징 벡터를 추출해야 물체에 가림occlusion이 발생해도 매칭이 안정적으로 동작한다.
- **적당한 양** 물체를 추적하려면 몇 개의 대응점만 있으면 된다. 하지만 대응점은 오류를 내포할 가능성이 있기 때문에 특징점이 더 많으면 더 정확하게 추적할 수 있다. 반면에 특징점이 너무 많으면 계산 시간이 과다해진다.
- **계산 효율** 계산 시간이 매우 중요한 응용이 많다. 예를 들어 선수를 추적하여 정보를 자동으로 표시하는 축구 중계의 경우 초당 몇 프레임 이상을 처리해야 하는 실시간 조건이 필수다.

위 조건 만족 필요

<알고리즘>

1. 가장 가까운 가우시안 영상 결정하고 거기서 기술자 추출
   → 스케일 불변성 달성
2. 기준 방향 설정, 기준 방향 중심으로 특징 추출
   → 회전 불변성 달성
3. 기술자 x를 단위 벡터로 변경 & 단위벡터에 0.2보다 큰 요소 있으면 0.2로 변경→ 단위 벡터
   → 조명 불변성 달성

```
import cv2 as cv

img=cv.imread('mot_color70.jpg') # 영상 읽기
gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)

sift=cv.SIFT_create() # SIFT 특징점 추출하는데 쓸 객체 생성
kp,des=sift.detectAndCompute(gray,None) # 특징점, 기술자 탐색
```

```
# 특징점 영상에 표시
gray=cv.drawKeypoints(gray,kp,None,flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

cv.imshow('sift', gray)

k=cv.waitKey()
cv.destroyAllWindows()
```

```
cv.SIFT_create(nfeatures=0,nOctaveLayers=3,contrastThreshold=0.04,edgeThreshold=10, sigma=1.6)
```

- 검출할 특징점 개수 지정
  - 0 : 검출한 특징점 모두 반환
  - 개수 : 신뢰도가 높은 순서로 지정한 만큼만 반환
- 옥타브 개수 지정
- 테일러 확장으로 미세조정할때 쓰는 매개변수
  - 값이 클수록 적은 수의 특징점 검출
- 에지에서 검출된 특징점을 걸러내는 데 쓰는 매개변수로 값이 클수록 덜 걸러내 더 많은 특징점 발생
- 옥타브 0의 입력 영상에 적용할 가우시안 표준편차

5.5 매칭

특징점이 많고 잡음이 섞인 기술자가 꽤 있다는 것을 고려해 **가장 유사한 특징점 찾아 쌍 맺기**

⇒ 같은 물체의 같은 곳에서 해당하는 쌍을 찾는 문제

- 첫번째 영상에서 추출한 기술자 집합 A = {a1, a2, a3, … , am}

  → 물체의 모델 영상(신뢰도가 높은 적은수의 기술자 검출)
- 두번째 영상에서 추출한 기술자 집합 B = {b1, b2, b3, … , bn}

  → 물체와 배경이 섞인 장면 영상 (기술자는 많고 잡음 심함 ㅠ)

mn 개의 쌍 각각에 대해 거리 계산, 거리가 임곗값보다 작은 쌍을 모두 취함

STEP1 거리 계산

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

→ 유클리디안 거리 사용해 두 기술자의 거리 계산

STEP2 매칭 전략

1. 고정 임곗값 방법

$$d(\mathbf{a}_i, \mathbf{b}_j) < T$$

   - 두 기술자 거리가 임곗값보다 작으면 매칭 !!
   - 임곗값 T 정하는 일이 중요

2. 최근접 이웃

   a_i는 B에서 거리가 가장 작은 b_j를 찾고 거리가 임계값보다 작으면 매칭 쌍으로 취함

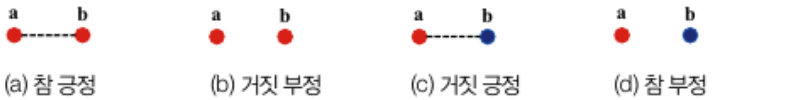
3. 최근접 이웃 거리 비율

$$\frac{d(\mathbf{a}_i, \mathbf{b}_j)}{d(\mathbf{a}_i, \mathbf{b}_k)} < T$$

   가장 가까운 bj 와 두번째 가까운 bk 찾아 비율로 확인

+++ 임계값 T를 작게하면 거짓긍정률은 작아지고, 크게 하면 거짓 긍정률이 커짐

STEP3 매칭 성능 측정 - 정확도



(a) 참 긍정 (b) 거짓 부정 (c) 거짓 긍정 (d) 참 부정

그림 5-14 매칭의 네 가지 경우(같은 색이 진짜 매칭 쌍이고 ----은 매칭 알고리즘이 맺어준 쌍)

색깔로 정답 표시

- 동일 색 : 진짜 매칭 쌍
- 다른 색 : 가짜 매칭 쌍

a. 긍정 예측-찐긍정

b. 거짓 예측-찐긍정

c. 긍정 예측 -찐부정

d. 부정 예측 - 찐부정

표 5-2 혼동 행렬

		정답(GT)	
		긍정	부정
예측	긍정	참 긍정(TP)	거짓 긍정(FP)
	부정	거짓 부정(FN)	참 부정(TN)

a,b,c,d의 경우의 빈도를 세어 혼동행렬 생성 → 성능 지표 계산 가능

$$\left.\begin{aligned} \text{정밀도} &= \frac{TP}{TP+FP} \\ \text{재현율} &= \frac{TP}{TP+FN} \\ F1 &= \frac{2\times \text{정밀도}\times \text{재현율}}{\text{정밀도}+\text{재현율}} \end{aligned}\right\} \quad (5.15)$$

$$\text{정확율} = \frac{TP+TN}{TP+TN+FP+FN} \quad (5.16)$$

정밀도 : 매칭 알고리즘이 긍정, 즉 매칭 쌍으로 예측한 개수 중 진짜 쌍인 비율

재현율 : 진짜 쌍 중에 알고리즘이 찾아낸 쌍의 비율

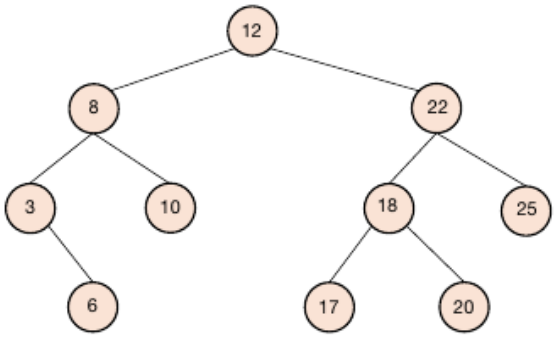
정확률 : 옳게 예측한 비율

STEP4 빠른 매칭 - 속도

속도 : 실기간 처리가 요구되는 상황에서 강한 조건

→ 대용량 데이터를 빠르게 탐색하는 자료구조 중 이진탐색트리 & 해싱

kd 트리



이진 탐색 트리

루트 노드(12) 기준

- 좌측 트리 모두 루트보다 작음
- 우측 트리 모두 루트보다 큼

→ 질의어를 탐색하는 일은 루트에서 시작해 루트보다 작으면 왼쪽, 크면 오른쪽으로 이동하는 일을 **재귀적 반복**으로 구성

이진탐색트리를 특징점 매칭에 적용하면 빠른 속도로 달성 가능

but 특징점 매칭의 독특한 성질 때문에 그대로 적용 불가

1. 특징점 매칭에서 여러값으로 구성된 기술자 (특징벡터) 비교
2. 이진 탐색 트리는 정확히 같은 값을 찾는 반면 특징점 매칭에서 최근접 이웃 찾아야 함

kd 트리 적용 과정

▼ 기술자 집합

$$X = x_1, x_2, ..., x_m$$

▼ i 번째 기술자

$$x_i = (x_i1, x_i2, ...x_id)$$

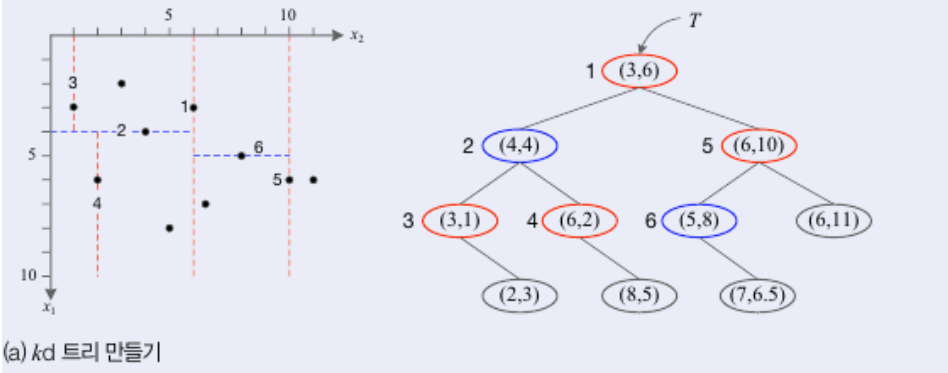
d개의 축 중 어느 것 사용할지 결정 → 축 각각의 분산 계산, 분산이 가장 큰 축 선택

분산이 클수록 트리 균형에 좋음

d=2, m=10 로 축은 각각 두개 축 존재

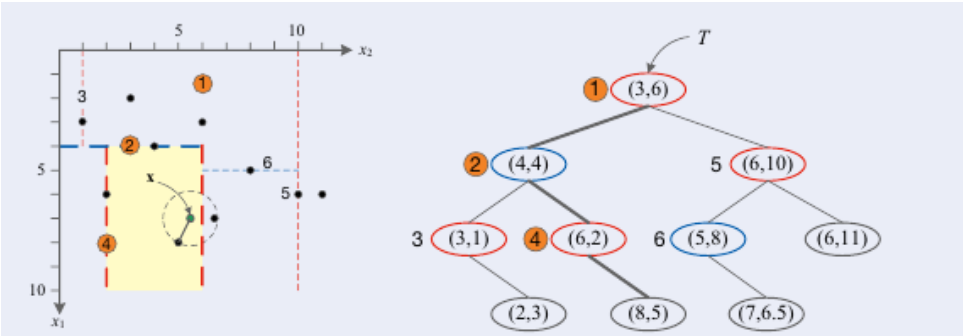
1. (3,2,6,4,…,6) (1,3,2,4,..11) ⇒ 두번째 축 분산이 더 커 두번째 축 선택
2. (1,3,2,4,..11) 내림차순 하여 중앙값 도출 시 j=5

   x_5 5번째 기술자가 분할 기준



(a) kd 트리 만들기

새로운 특징 벡터가 입력되었다고 가정하고 트리에서 최근접이웃 찾기



(b) kd 트리 탐색

그림 5-17 kd 트리 만들기와 kd 트리 탐색[오일석2014]

위치 의존 해싱

해시함수 h가 키값을 키가 저장될 칸의 번호로 매핑

ex. h(134) = 134%13 = 4

해시함수를 한번만 계산하면 키가 들어있는 칸을 찾을 수 있어 빠르게 탐색 가능

비슷한 특징 벡터가 같은 칸에 담길 확률을 최대화해야 함

+) **FLANN, FLAISS**

kd 트리의 다양한 변형을 보여주는 함수

```
import cv2 as cv
import numpy as np
```

```
import time

# 물체 모델 영상 정하기
img1=cv.imread('mot_color70.jpg')[190:350,440:560] # 버스를 크롭하여 모델 영상으로 사용
gray1=cv.cvtColor(img1,cv.COLOR_BGR2GRAY)

# 물체 장면 영상 정하기
img2=cv.imread('mot_color83.jpg')
gray2=cv.cvtColor(img2,cv.COLOR_BGR2GRAY)

sift=cv.SIFT_create()
kp1,des1=sift.detectAndCompute(gray1,None)
kp2,des2=sift.detectAndCompute(gray2,None)
print('특징점 개수:',len(kp1),len(kp2))

start=time.time()
flann_matcher=cv.DescriptorMatcher_create(cv.DescriptorMatcher_FLANNBASED) # FLANN 라이브러리 사용
knn_match=flann_matcher.knnMatch(des1,des2,2) # 최근접 두개 찾기

# 임계값 이용해 최근접 이웃거리 비율 전략 적용
T=0.7
good_match=[]
for nearest1,nearest2 in knn_match:
    if (nearest1.distance/nearest2.distance)<T:
        good_match.append(nearest1)
print('매칭에 걸린 시간:',time.time()-start)

# 매칭 결과 보여줄 ㄹ영상 생성
img_match=np.empty((max(img1.shape[0],img2.shape[0]),img1.shape[1]+img2.shape[1],3),dtype=np.uint8)
cv.drawMatches(img1,kp1,img2,kp2,good_match,img_match,flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

cv.imshow('Good Matches', img_match)

k=cv.waitKey()
cv.destroyAllWindows()
```

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

5.6 호모그래피 추정

아웃라이어(이상치) 걸러내는 과정 필요

매칭 쌍을 이용해 **물체 위치** 찾는 과정 추가

호모그래피 : 3차원에서 z 축을 무시해 변환한 3*3행렬로 표현되는 제한된 상황에서 3차원 점이 2차원 평면으로 변환되는 기하 관계인 **투영변환**을 평면 호모그래피

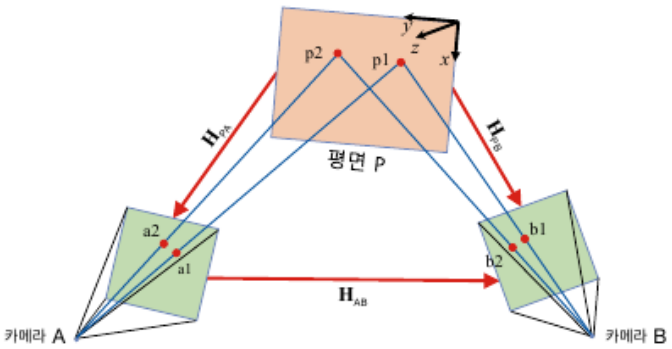


그림 5-18 호모그래피

평면P : 3차원 물체가 놓인 평면 / 카메라 A 영상 평면 / 카메라 B 영상 평면

호모그래피 행렬 H : 어떤 평면의 점을 다른 평면의 점으로 투영하는 변환행렬

앞서 구한 매칭쌍으로 호모그래피 행렬 도출

ex. 점 a → 점 b 위치로 되게 하는 것은 행렬 H

$$\mathbf{b}^{\mathrm{T}} = \begin{pmatrix} b_x \\ b_y \\ 1 \end{pmatrix} = \begin{pmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{pmatrix} \begin{pmatrix} a_x \\ a_y \\ 1 \end{pmatrix} = \mathbf{H}\mathbf{a}^{\mathrm{T}} \qquad (5.18)$$

$$\mathbf{B} = \mathbf{H}\mathbf{A} \qquad (5.20)$$

- H_PA : 평면 P를 카메라 A의 영상 평면으로 투영하는 호모그래피 행렬
- H_PB : 카메라 B의 영상평면으로 투영
- H_AB : 카메라 A의 평면을 카메라 B의 평면으로 투영

행렬 H구하는 알고리즘 - 최소평균제곱오차, RANSAC

최소평균제곱오차

$$E = \frac{1}{n}\sum_{i=1,n}\|\mathbf{H}\mathbf{a}_i^{\mathrm{T}} - \mathbf{b}_i\|_2^2 \qquad (5.21)$$

E가 최소인 H 찾기

모든 매칭쌍이 같은 자격으로 오류계산에 참여함

→ 이상치로 인해 호모그래피 정확도 감소할 수 있음

이상치 걸러내기

모든 쌍의 n개의 오차를 계산한 다음 정렬한 뒤, 가운데 위치한 오차(중앙값)를 E 취함 ↔ 평균 X

RANSAC

표본에 섞여 있는 아웃라이어 회피하며 최적해 구하는 기법

매칭 쌍 집합을 입력으로 받아 최적의 호모그래피 행렬 추정

[알고리즘 5-2] 호모그래피 추정을 위한 RANSAC

입력: 매칭 쌍 집합 $X=\{(\mathbf{a}_1,\mathbf{b}_1),(\mathbf{a}_2,\mathbf{b}_2),\cdots,(\mathbf{a}_n,\mathbf{b}_n)\}$,, 반복 횟수 m, 임곗값 t, d, e

출력: 최적 호모그래피 $\hat{\mathbf{H}}$

1. h=[]
2. for j=1 to m
3. X에서 네 쌍을 랜덤하게 선택하고 식 (5.21)을 풀어 호모그래피 행렬 $\mathbf{H}$를 추정한다.
4. 이들 네 쌍으로 *inlier* 집합을 초기화한다.
5. for (3행에서 선택한 네 쌍을 제외한 모든 쌍 p에 대해)
6. if (p가 허용 오차 t 이내로 $\mathbf{H}$에 적합하면) p를 *inlier*에 삽입한다.
7. if (*inlier*가 d개 이상의 요소를 가지면)
8. *inlier*의 모든 요소를 가지고 호모그래피 행렬 $\mathbf{H}$를 다시 추정한다.
9. if (8행에서 적합 오차가 e보다 작으면) $\mathbf{H}$를 h에 삽입한다.
10. h에 있는 호모그래피 중에서 가장 좋은 것을 $\hat{\mathbf{H}}$로 취한다.

1. 후보 호모그래피 저장할 리스트 초기화
2. 반복 횟수 수천으로 지정
3. 매칭 쌍 집합 X에서 4쌍을 임의로 선택하고 오류 E가 최소가 되도록 H 추정
4. 선택된 임의의 4쌍으로 inlier 집합 초기화
5. ~6. 나머지 매칭 쌍 중 H에 동의하는 쌍 찾아 inlier 집합에 추가

7. ~ 9. 모든 쌍 가지고 H추정 및 h 추가

import cv2 as cv
import numpy as np
import time

# 물체 모델 영상 정하기
img1=cv.imread('mot_color70.jpg')[190:350,440:560] # 버스를 크롭하여 모델 영상으로 사용
gray1=cv.cvtColor(img1,cv.COLOR_BGR2GRAY)

# 물체 장면 영상 정하기
img2=cv.imread('mot_color83.jpg')
gray2=cv.cvtColor(img2,cv.COLOR_BGR2GRAY)

sift=cv.SIFT_create()
kp1,des1=sift.detectAndCompute(gray1,None)
kp2,des2=sift.detectAndCompute(gray2,None)
print('특징점 개수:',len(kp1),len(kp2))

start=time.time()
flann_matcher=cv.DescriptorMatcher_create(cv.DescriptorMatcher_FLANNBASED) # FLANN 라이브러리 사용
knn_match=flann_matcher.knnMatch(des1,des2,2) # 최근접 두개 찾기

# 임계값 이용해 최근접 이웃거리 비율 전략 적용
T=0.7
good_match=[]
for nearest1,nearest2 in knn_match:
    if (nearest1.distance/nearest2.distance)<T:
        good_match.append(nearest1)

```
# good_match : 매칭 쌍 중 좋은 것을 골라 저장한 리스트
points1=np.float32([kp1[gm.queryIdx].pt for gm in good_match]) # 첫번째 영상 특징점 좌표
points2=np.float32([kp2[gm.trainIdx].pt for gm in good_match]) # 두번째 영상 특징점 좌표

H,_=cv.findHomography(points1,points2,cv.RANSAC) # 호모그래피 행렬 추정

h1,w1=img1.shape[0],img1.shape[1]     # 첫 번째 영상의 크기
h2,w2=img2.shape[0],img2.shape[1]     # 두 번째 영상의 크기

box1=np.float32([[0,0],[0,h1-1],[w1-1,h1-1],[w1-1,0]]).reshape(4,1,2) # 첫번째 영상을 포함하는 네 구석의 좌표 저장
box2=cv.perspectiveTransform(box1,H) # 첫번째 영상 좌표에 행렬 H 적용, 두번째 영상 투영한 결과 저장

img2=cv.polylines(img2,[np.int32(box2)],True,(0,255,0),8) # box2를 두번째 영상에 그리기

img_match=np.empty((max(h1,h2),w1+w2,3),dtype=np.uint8)
cv.drawMatches(img1,kp1,img2,kp2,good_match,img_match,flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

cv.imshow('Matches and Homography',img_match)

k=cv.waitKey()
cv.destroyAllWindows()
```