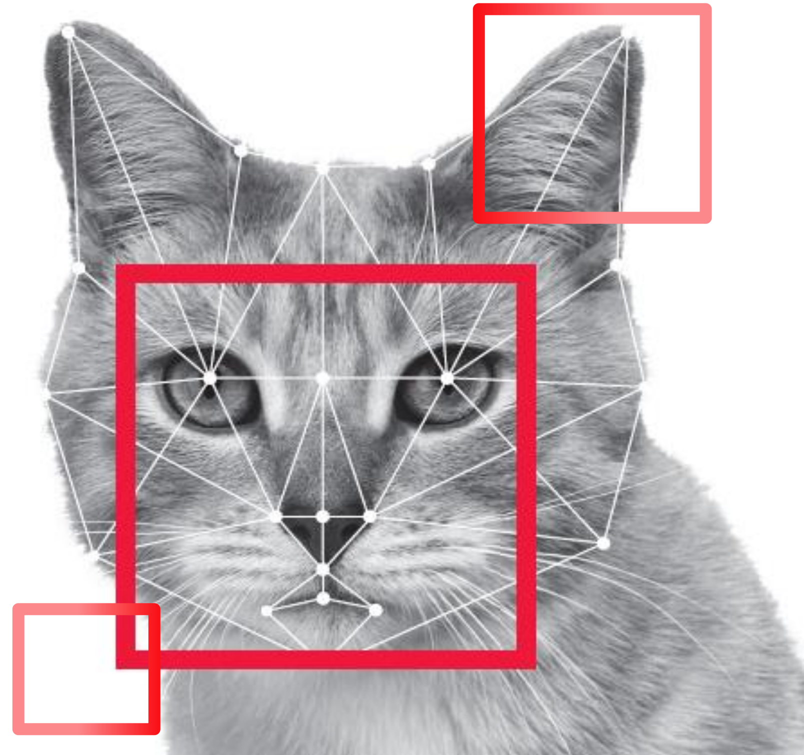
한빛아카데미
Hanbit Academy, Inc.



COMPUTER VISION 컴퓨터 비전

기본 개념부터 최신 모바일 응용 예까지

7장. 매칭

PREVIEW

매칭

- 어떤 대상을 다른 것과 비교하여 같은 것인지 알아내는 과정
- 둘 사이의 유사성 또는 거리를 측정해서 비교
- 여러 가지 문제를 해결하는 열쇠 (물체 인식, 자세 추정, 스테레오, 증강 현실 등)

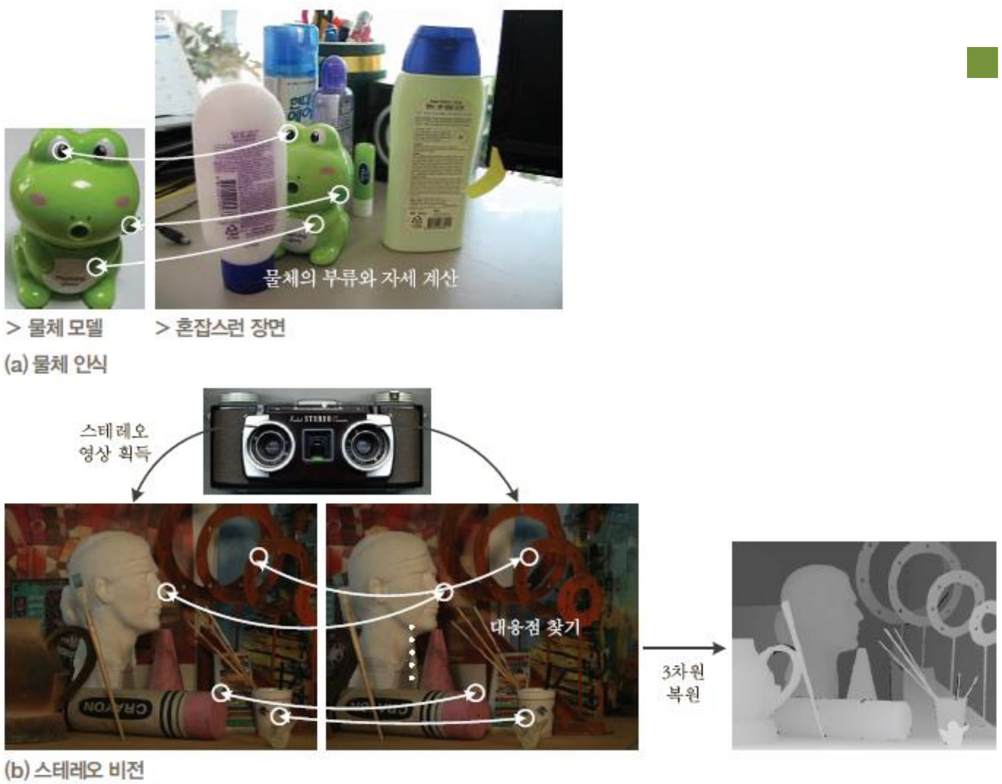


그림 7-1 매칭을 이용한 응용 문제 해결

생각해 볼 점

- 거짓 긍정을 어떻게 찾아 배제할 것인가?
- 매칭 속도
  - 두 영상의 특징점 개수가 m 과 n 이고 특징 벡터의 차원이 d 라면,

    두 영상을 매칭하는데 $\Theta(mnd)$ 시간 소요
  - 미리 인덱싱 해두면 실시간 처리가 가능할까?

각 절에서 다루는 내용

1. 매칭의 기초

→ 매칭에 사용하는 거리 척도와 매칭 전략, 성능을 분석하는 척도

2. 빠른 최근접 이웃 탐색

→ 매칭 속도를 올릴 수 있는 방법 : *kd* 트리와 해싱

3. 기하 정렬과 변환 추정

→ 신뢰도가 높은 매칭 쌍을 고르는 방법

4. 웹과 모바일 응용

→ 매칭을 활용한 파노라마, 사진 관광 응용 사례

7.1 매칭의 기초

7.1.1 거리 척도

- 유클리디안 거리 vs. 마할라노비스 거리

7.1.2 매칭 전략과 성능 분석

7.1.1 거리 척도

- 유클리디안 거리(euclidian distance) vs. 마할라노비스 거리(mahalanobis distance)

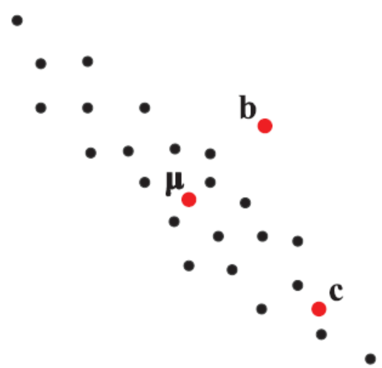


그림 7-2 확률 분포 속의 거리

← μ는 b와 c 중 누구와 더 가까운가?

- 유클리디안 거리

$$d_E(\mathbf{a},\mathbf{b}) = \|\mathbf{a}-\mathbf{b}\| = \sqrt{\sum_{i=1}^{d}(a_i-b_i)^2} \qquad (7.1)$$

- 마할라노비스 거리 : 공분산 행렬 **Σ**를 이용하여 확률 분포를 고려 (**c**가 더 가깝다.)

점 $\mathbf{a}$와 $N(\boldsymbol{\mu},\boldsymbol{\Sigma})$사이의 마할라노비스 거리 $d_M(\mathbf{a}) = \sqrt{(\mathbf{a}-\boldsymbol{\mu})\boldsymbol{\Sigma}^{-1}(\mathbf{a}-\boldsymbol{\mu})^{\mathrm{T}}}$ (7.2)

두 점 $\mathbf{a}$와 $\mathbf{b}$ 사이의 마할라노비스 거리 $d_M(\mathbf{a},\mathbf{b}) = \sqrt{(\mathbf{a}-\mathbf{b})\boldsymbol{\Sigma}^{-1}(\mathbf{a}-\mathbf{b})^{\mathrm{T}}}$ (7.3)

7.1.1 거리 척도

예제 7-1 **마할라노비스 거리**

[그림 7-3]은 네 개의 점 {(2,1), (1,3), (2,5), (3,3)}이 확률 분포를 이루는 간단한 상황이다. 먼저 이 분포를 무시하고 유클리디안 거리를 계산하면 $d_E(\mu,b)=2$, $d_E(\mu,c)=3$이므로 b가 c보다 μ에 더 가깝다.

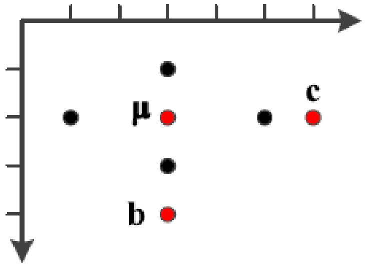


그림 7-3 마할라노비스 거리 예제

이제 확률 분포를 고려한 거리를 계산해 보자. 이 분포의 평균은 $\mu=(2,3)$이고 공분산 행렬은 $\Sigma=\begin{pmatrix}0.5 & 0\\ 0 & 2\end{pmatrix}$이다. Σ의 역행렬을 구하면, $\Sigma^{-1}=\begin{pmatrix}2 & 0\\ 0 & 0.5\end{pmatrix}$이다. 이들을 이용하여 두 점 b와 c에서 이 가우시안 분포까지의 거리를 계산하면 다음과 같다. b와 c는 각각 분포까지의 거리가 2.8284와 2.1213이므로 c가 b보다 가우시안 분포에 더 가깝다.

$$\text{b와 가우시안 분포 사이의 마할라노비스 거리 } d_M(b)=\sqrt{\begin{pmatrix}4-2 & 3-3\end{pmatrix}\begin{pmatrix}2 & 0\\ 0 & 0.5\end{pmatrix}\begin{pmatrix}4-2\\ 3-3\end{pmatrix}}=2.8284$$

$$\text{c와 가우시안 분포 사이의 마할라노비스 거리 } d_M(c)=\sqrt{\begin{pmatrix}2-2 & 6-3\end{pmatrix}\begin{pmatrix}2 & 0\\ 0 & 0.5\end{pmatrix}\begin{pmatrix}2-2\\ 6-3\end{pmatrix}}=2.1213$$

이제 두 점 b와 c 사이의 유클리디안 거리와 마할라노비스 거리를 계산해 보자.

$$\text{b와 c 사이의 유클리디안 거리 } d_E(b,c)=\sqrt{(4-2)^2+(3-6)^2}=3.6056$$

$$\text{b와 c 사이의 마할라노비스 거리 } d_M(b,c)=\sqrt{\begin{pmatrix}4-2 & 3-6\end{pmatrix}\begin{pmatrix}2 & 0\\ 0 & 0.5\end{pmatrix}\begin{pmatrix}4-2\\ 3-6\end{pmatrix}}=\ 3.5355$$

7.1.1 거리 척도

- 화이트닝 변환(whitening transform)
  - 공분산 행렬이 단위 행렬 **I**가 되도록 벡터 **x**를 **y**로 변환 → **Σ**=**I**이면 두 거리 척도가 같음

$$\mathbf{y}^{\mathrm{T}} = \boldsymbol{\Lambda}^{-\frac{1}{2}}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{x}^{\mathrm{T}} \qquad (7.4)$$

예제 7-2 화이트닝 변환

[예제 7-1]의 샘플을 재활용한다. 공분산 행렬 $\boldsymbol{\Sigma}=\begin{pmatrix}0.5 & 0\\ 0 & 2\end{pmatrix}$의 고유 벡터와 고유값을 계산하여, $\boldsymbol{\Phi}$와 $\boldsymbol{\Lambda}$를 구성하면 다음과 같다.

두 개의 고유값과 고유 벡터 : 0.5와 (1,0), 2.0과 (0,1)

$$\boldsymbol{\Phi}=\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix},\quad \boldsymbol{\Lambda}=\begin{pmatrix}0.5 & 0\\ 0 & 2.0\end{pmatrix},\quad \boldsymbol{\Lambda}^{-\frac{1}{2}}=\begin{pmatrix}1.4142 & 0\\ 0 & 0.7071\end{pmatrix}$$

$$\boldsymbol{\Lambda}^{-\frac{1}{2}}\boldsymbol{\Phi}^{\mathrm{T}}=\begin{pmatrix}1.4142 & 0\\ 0 & 0.7071\end{pmatrix}$$

Λ : 고유값 행렬

Φ : 고유벡터 행렬

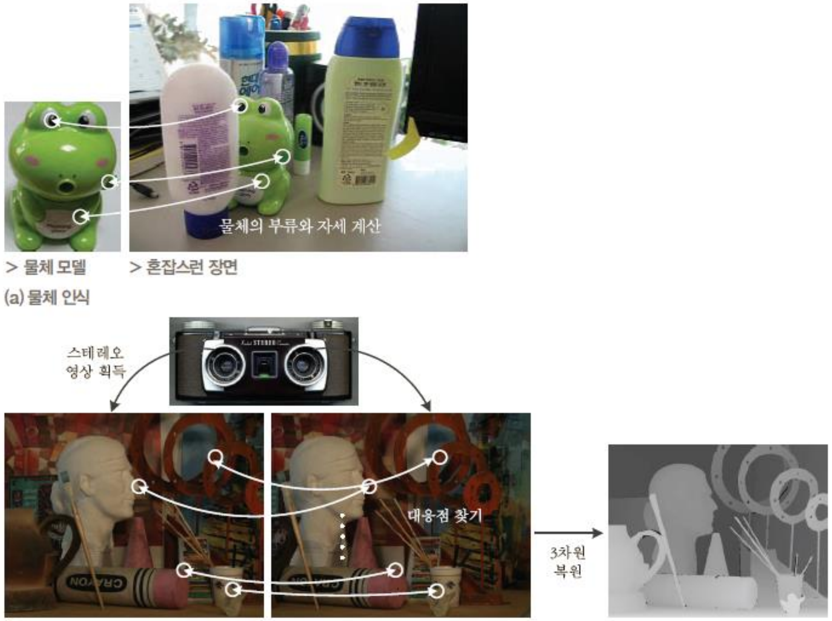
네 개의 샘플을 식 (7.4)로 변환하면 다음과 같다. 새로 얻은 네 점을 가지고 공분산 행렬을 구해 보면 단위 행렬 **I**가 되어, 화이트닝 변환이 적용되었음을 확인할 수 있다.

$$\begin{pmatrix}1.4142 & 0\\ 0 & 0.7071\end{pmatrix}\begin{pmatrix}2\\ 1\end{pmatrix}=\begin{pmatrix}2.8284\\ 0.7071\end{pmatrix},\quad \begin{pmatrix}1.4142 & 0\\ 0 & 0.7071\end{pmatrix}\begin{pmatrix}1\\ 3\end{pmatrix}=\begin{pmatrix}1.4142\\ 2.1213\end{pmatrix},$$

$$\begin{pmatrix}1.4142 & 0\\ 0 & 0.7071\end{pmatrix}\begin{pmatrix}2\\ 5\end{pmatrix}=\begin{pmatrix}2.8284\\ 3.5355\end{pmatrix},\quad \begin{pmatrix}1.4142 & 0\\ 0 & 0.7071\end{pmatrix}\begin{pmatrix}3\\ 3\end{pmatrix}=\begin{pmatrix}4.2426\\ 2.1213\end{pmatrix}$$

7.1.2 매칭 전략과 성능 분석

- 매칭을 활용하는 여러 가지 상황
  - 파노라마 영상 제작
    - 두 영상이 동등한 입장에서 참여
  - 물체 인식 또는 증강 현실
    - 모델 영상은 깨끗한 배경 위에 물체가 놓임
    - 장면 영상은 심한 혼재와 가림이 발생



(a) 물체 인식
(b) 스테레오 비전
그림 7-1 매칭을 이용한 응용 문제 해결

- 단순한 매칭 전략
  - 두 영상의 특징 벡터를 $\mathbf{a}_i$ (i=1,2,…,m)와 $\mathbf{b}_j$ (j=1,2,…,n)라 표기할 때,
    식 (7.5)를 만족하면 매칭 성공

$$d(\mathbf{a}_i, \mathbf{b}_j) < T \tag{7.5}$$

7.1.2 매칭 전략과 성능 분석

- ROC(Receiver Operating Characteristic) 곡선을 이용한 성능 분석
  - if) 낮은 *T* → 큰 *FN* (거짓 부정, 맞는 데 다르다고 판정) → 낮은 참 긍정률

    낮은 *T* → 낮은 *FP* (거짓 긍정, 다른 데 맞다고 판정) → 낮은 거짓 긍정률
  - if) 높은 *T* → 높은 참 긍정률, 높은 거짓 긍정률

$$\text{참 긍정률 } TPR = \frac{TP}{(TP+FN)}$$

$$\text{거짓 긍정률 } FPR = \frac{FP}{(FP+TN)}$$

  - *T* 를 점점 키우면서 측정한 거짓 긍정률과 참 긍정률을 나타낸 그래프가 ROC
  - 왼쪽 위 구석에 가까울수록 좋은 성능
  - 성능 지표 : AUC (Area Under Curve, 곡선 아래 면적)

표 1–1 부류가 두 개인 경우의 혼동 행렬

참 부류 \ 분류 결과	ω_1	ω_2
ω_1	n_{11}(TP)	n_{12}(FN)
ω_2	n_{21}(FP)	n_{22}(TN)

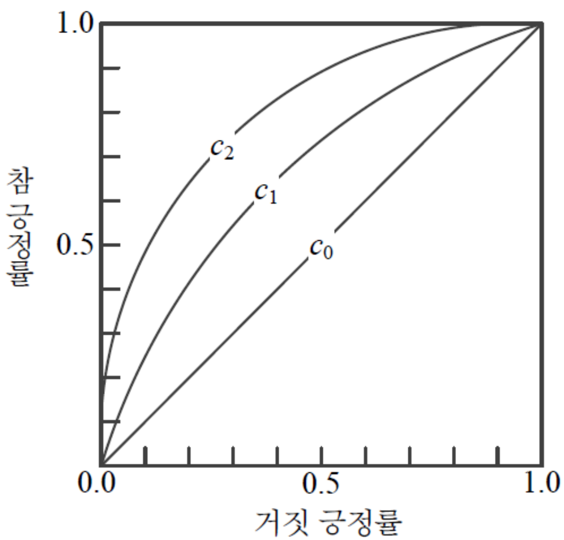


그림 7–4 ROC 성능 분석

7.1.2 매칭 전략과 성능 분석

- 또 다른 매칭 전략
  - 최근접 이웃 전략
    - $\mathbf{a}_i$의 최근접 이웃 $\mathbf{b}_j$가 $d(\mathbf{a}_i, \mathbf{b}_j) < T$를 만족하면 매칭 성공
  - 최근접 거리 비율 전략
    - 최근접 $\mathbf{b}_j$와 두 번째 최근접 $\mathbf{b}_k$가 식 (7.7)을 만족하면 매칭 성공

$$\frac{d(\mathbf{a}_i, \mathbf{b}_j)}{d(\mathbf{a}_i, \mathbf{b}_k)} < T \qquad (7.7)$$

애가 재보다 월등히 작아야 함

- 실험에 따르면 이 전략이 가장 높은 성능 (예, SIFT [Lowe2004])

7.2 빠른 최근접 이웃 탐색

7.2.1 *k*d 트리

7.2.2 해싱

7.2 빠른 최근접 이웃 탐색

- 순진한 알고리즘
  - 모든 쌍을 일일이 검사 → 시간이 넉넉한 상황에서만 활용 가능

- 특징 벡터를 미리 인덱싱해 두는 효율적인 알고리즘
  - 두 알고리즘: *k*d 트리와 위치의존 해싱
  - 일반적인 탐색 기법으로 데이터마이닝, 빅데이터, 생물 정보학 등에서도 활용

7.2.1 *kd* 트리

- 이진검색 트리 (BST, binary search tree)
  - 루트를 기준으로 왼쪽 부분 트리는 루트보다 작은 값, 오른쪽은 큰 값을 갖는 이진 트리 (이 성질이 부분 트리에 재귀적으로 반복 적용)

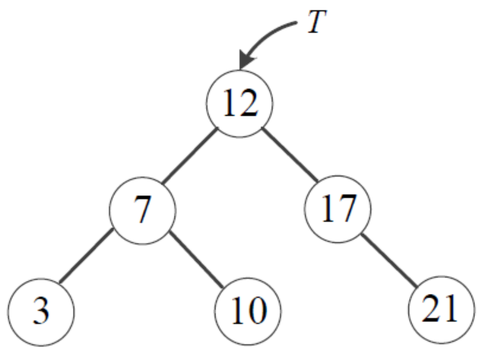


그림 7-5 이진검색 트리

균형이 잡힌 경우

탐색시간 $O(\log n)$

- BST를 적용할 수 있나?
  - 검색 키(특징 벡터)가 여러 개의 실수로 구성된 벡터임
  - 동일한 값을 갖는 노드를 찾는 것이 아니라, 최근접 이웃을 찾음
- BST를 그대로 적용 불가능 → *kd* 트리
  - 두 가지 다른 점을 수용할 수 있게 BST를 확장한 기법 [Bently75]

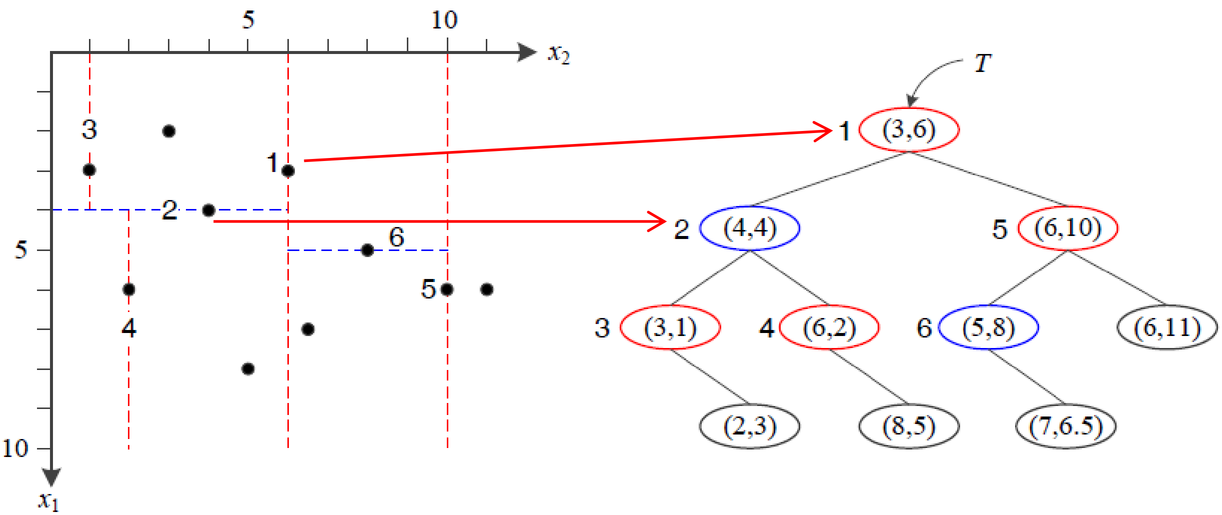
7.2.1 *k*d 트리

- 표기
  - n개의 벡터를 가지고 *k*d 트리 구축 $X=\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$
  - $\mathbf{x}_i$는 d 차원 벡터
- *k*d 트리의 원리
  - 루트 노드는 X를 두 개의 부분 집합 X_{left}와 X_{right} 나눔
  - 이때 분할 기준을 어떻게 선택하나?
    - d 개의 차원(축) 중에 어느 것을 쓸 것인가?
      - 분할 효과를 극대화하려면 각 차원의 분산을 계산한 후, 최대 분산을 갖는 축 k를 선택
    - 축을 선택했다면 n개의 샘플 중 어느 것을 기준으로 X를 분할할 것인가?
      - X_{left}와 X_{right}의 크기를 갖게 하여 균형 잡힌 트리를 만들어야 함.
      - X를 차원 k로 정렬하고, 그 결과의 중앙 값을 분할 기준으로 삼는다.
  - X를 X_{left}와 X_{right}로 분할한 후, 각각에 같은 과정을 재귀적으로 반복하면 *k*d트리 완성

7.2.1 *kd* 트리

예제 7-3 ***kd* 트리 만들기**

설명을 쉽게 하기 위해 $d=2$로 한정하고, $X=\{\mathbf{x}_1=(3,1),\ \mathbf{x}_2=(2,3),\ \mathbf{x}_3=(6,2),\ \mathbf{x}_4=(4,4),\ \mathbf{x}_5=(3,6),\ \mathbf{x}_6=(8,5),\ \mathbf{x}_7=(7,6.5),\ \mathbf{x}_8=(5,8),\ \mathbf{x}_9=(6,10),\ \mathbf{x}_{10}=(6,11)\}$이라 하자. [그림 7-6(a)]는 주어진 특징 벡터의 집합을 보여준다.



(a) 특징 벡터의 집합

(b) 완성된 *kd* 트리

그림 7-6 *kd* 트리

먼저, 루트 노드로 결정할 만한 분할 기준을 찾아보자. 두 개의 차원은 각각 3, 2, 6, 4, …, 6과 1, 3, 2, 4, …, 11의 값을 가진다. 이들의 분산을 구해 보면 두 번째가 더 크다. 따라서 [알고리즘 7-4]의 11행에서 k는 2가 된다. 두 번째 차원을 기준으로 X를 정렬하면 $X_{sorted}=\{\mathbf{x}_1,\ \mathbf{x}_3,\ \mathbf{x}_2,\ \mathbf{x}_4,\ \mathbf{x}_6,\ \mathbf{x}_5,\ \mathbf{x}_7,\ \mathbf{x}_8,\ \mathbf{x}_9,\ \mathbf{x}_{10}\}$이 된다. 이 리스트의 중앙값 $\mathbf{x}_5$를 기준으로 좌우를 분할하면, $X_{left}=\{\mathbf{x}_1,\ \mathbf{x}_3,\ \mathbf{x}_2,\ \mathbf{x}_4,\ \mathbf{x}_6\}$, $X_{right}=\{\mathbf{x}_7,\ \mathbf{x}_8,\ \mathbf{x}_9,\ \mathbf{x}_{10}\}$이 된다. 이제 14~16행에서 노드를 하나 할당받아 값을 채운다. 이렇게 만들어진 노드가 [그림 7-6]에서 T가 가리키는 루트 노드이다.

이 루트 노드의 물리적인 의미를 해석해 보자. [그림 7-6(a)]에서 1 옆의 빨간색 선이 이 노드의 역할을 보여준다. 이 노드는 $k=2$에 해당하는 x_2축을 기준으로 공간을 둘로 분할한다. 이때 왼쪽 영역에 있는 점들이 X_{left}가 되고 오른쪽 영역은 X_{right}가 된다. 이제 X_{left}와 X_{right} 각각에 같은 과정을 재귀적으로 반복하면 [그림 7-6(b)]와 같은 *kd* 트리가 완성된다. 그림에서는 기준이 되는 축을 쉽게 구분할 수 있도록 각각 다른 색으로 표시하였다. x_1축이 기준이라면 파란색, x_2축이 기준이라면 빨간색이다.

7.2.1 *kd* 트리

kd 트리에서 최근접 이웃 탐색

- 새로운 특징 벡터 **x**가 입력되면 **x**의 최근접 이웃을 어떻게 찾을까?
- 예) **x**가 =(7, 5.5)
  - 루트가 x_2축을 기준으로 하므로 5.5를 6과 비교하고 작으므로 왼쪽으로 분기
  - (4,4) 노드가 x_1축을 기준으로 하므로 7과 4를 비교하고 크므로 오른쪽으로 분기
  - 반복하면 (8,5) 노드에 도착

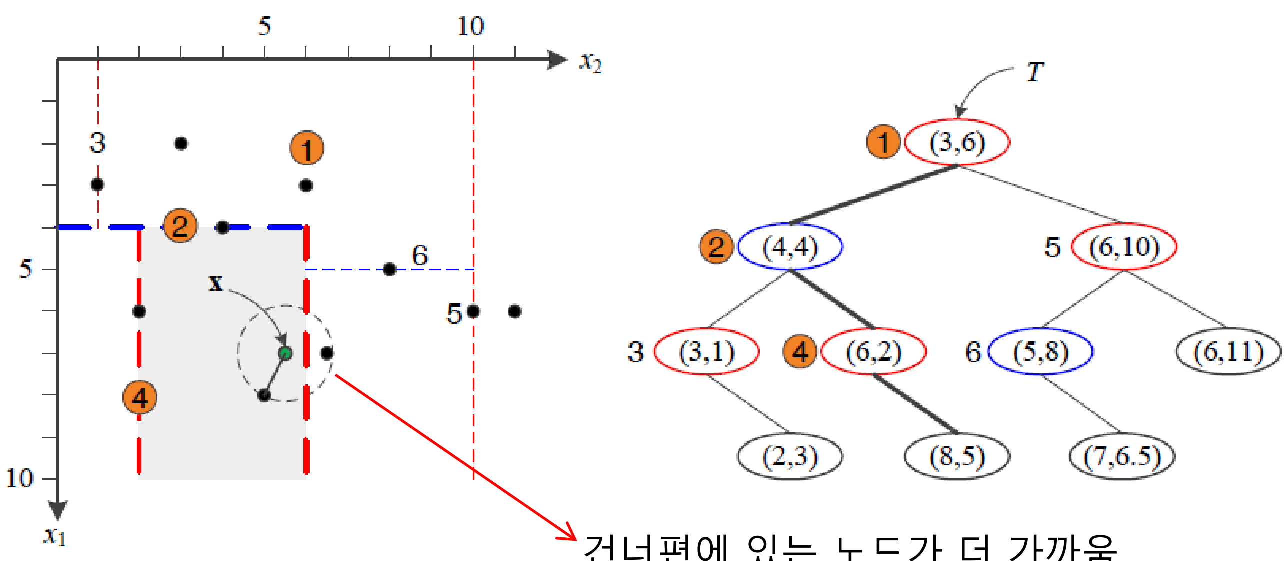


그림 7-7 *kd* 트리에서 검색하는 예

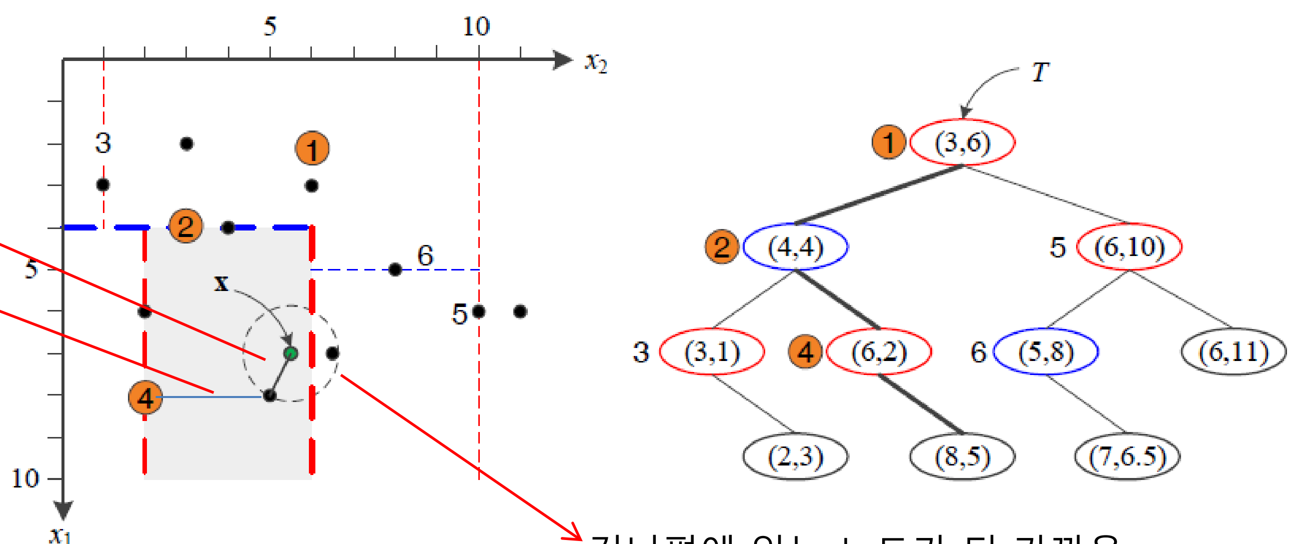
7.2.1 *kd* 트리

kd 트리를 이용한 최근접 이웃 탐색

- 리프 노드 (8,5)를 답으로 취하면 될까?
  - 최근접일 가능성이 있지만 반드시 그렇진 않다.
  - 분할 평면의 건너편에 더 가까운 노드가 있을 수 있음
- 스택을 이용한 백트래킹
  - 한정 분기(branch-and-bound)를 적용하면서 ④→②→① 순으로 처리

current_best를 설정하고

기준 노드와의 거리 비교



건너편에 있는 노드가 더 가까움

그림 7-7 *kd* 트리에서 검색하는 예

7.2.1 *kd* 트리

- 백트래킹때문에 속도 저하 발생
- d > 10 → 모든 특징 벡터와 비교하는 단순한 알고리즘과 비슷한 낮은 속도
  - 어떻게 시간 효율을 회복할 수 있을까?
- 근사 최근접 이웃(approximate nearest neighbor) 탐색
  - 최적 칸 우선 탐색
    - 스택 대신 우선순위 큐(priority queue)인 힙(heap)을 사용
    - 거리를 우선순위로 사용하는데, 백트래킹할 때 가까운 것부터 조사하도록 해줌
    - 예) 그림 7-7에서, ①→④→② 순으로 처리
  - 미리 설정해 놓은 값 *try_allowed* 에 따라 조사 횟수를 제한함
    - 근사 최근접에서 멈춤 (최적칸 우선을 적용했기 때문에 최근접 찾을 확률 높음)
    - 대신 시간 효율 얻음
- 성능 : SIFT의 실험 예 [Lowe2004]
  - *try_allowed* =200으로 했을 때, n=100,000, d=128인 상황에서 최근접 95%, 근사 최근접 5%
  - 속도는 100배 빨라짐

7.2.2 해싱(hashing)

해싱의 원리

- 임의의 길이의 입력 메세지를 고정된 길이의 출력값으로 압축시키는 함수
- 많은 메모리를 사용하는 대신 계산 시간이 짧게 걸리는 검색 기법
- 해시 함수(hash function)는 키 값을 해시 테이블의 주소로 변환
- 테이블에 골고루 배치할수록 좋은 해시 함수
- 충돌(collision) 해결책 필요

해시 함수 $h(x)=x \bmod 13$

0	
1	27
2	
3	
4	147
5	
6	19
7	
8	8
9	
10	23
11	1311
12	

그림 7-8 해싱의 원리

7.2.2 해싱

매칭에 적용

- 일반 해싱과 다른 점
  - 키는 단일 값이 아니라 실수 여러 개로 구성된 특징 벡터 $\mathbf{x}=\{x_1, x_2, \ldots, x_d\}$임
  - 동일한 요소가 아니라 최근접 이웃을 찾음
- 가장 크게 다른 점 : 일반 해싱과 정반대 목표
  - 일반 해싱은 데이터를 골고루 배치하는 반면, 매칭에서는 가까운 벡터들은 같은 통에 담길 확률이 높아야 함

  → 위치의존 해싱(locality-sensitive hashing)으로 해결

7.2.2 해싱

- 위치의존 해싱 [Andoni2008]
  - 하나의 해시 함수가 아니라, 해시 함수 집합 H 에서 여러 개를 임의 선택하여 사용
  - H 에 속한 해시 함수 h 가 식 (7.8)을 만족하면, H 는 위치의존적

$$\begin{gathered}\text{임의의 두 점 } \mathbf{a}\text{와 } \mathbf{b}\text{에 대해,}\\ \|\mathbf{a}-\mathbf{b}\| \le R\text{이면, } p(h(\mathbf{a})=h(\mathbf{b})) \ge p_1\text{이고}\\ \|\mathbf{a}-\mathbf{b}\| \ge cR\text{이면, } p(h(\mathbf{a})=h(\mathbf{b})) \le p_2\text{이다.}\\ \text{이때 } c>1,\ \ p_1>p_2\end{gathered} \tag{7.8}$$

  - 위치의존의 의미는?

가까운 두 벡터는 같은 통에 담길 (해시 함수 값이 같을) 확률이 크고, 먼 벡터는 같은 통에 담길 확률이 작음

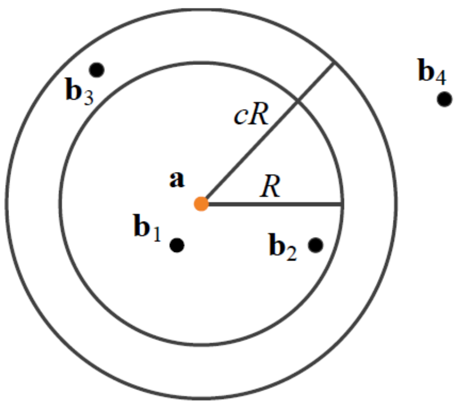


그림 7-9 조건식 (7.8)의 의미

7.2.2 해싱

- H 를 어떻게 만드나?
  - 여러 가지 알고리즘 존재, 식 (7.9)는 그 중 하나
  - 난수로 **r**과 b를 설정하여 원하는 수만큼 함수 생성 가능

$$h(\mathbf{x}) = \left\lfloor \frac{\mathbf{r} \cdot \mathbf{x} + b}{w} \right\rfloor \tag{7.9}$$

- 해시 함수 h 의 동작
  - d 차원 공간을 **r**에 수직인 초평면으로 분할
  - w 는 구간의 간격으로서, 작으면 촘촘하게 크면 듬성듬성 분할

7.2.2 해싱

예제 7-4 **위치의존 해시 함수**

특징 벡터는 $\mathbf{x}=(x_1, x_2)$로 표현되는 2차원이라 가정한다. w는 2로 설정되어 있고, 난수를 생성하여 $\mathbf{r}=(1,2)$, $b=0.6$을 얻었다고 하자. 식 (7.9)에 따른 해시 함수는 다음과 같다.

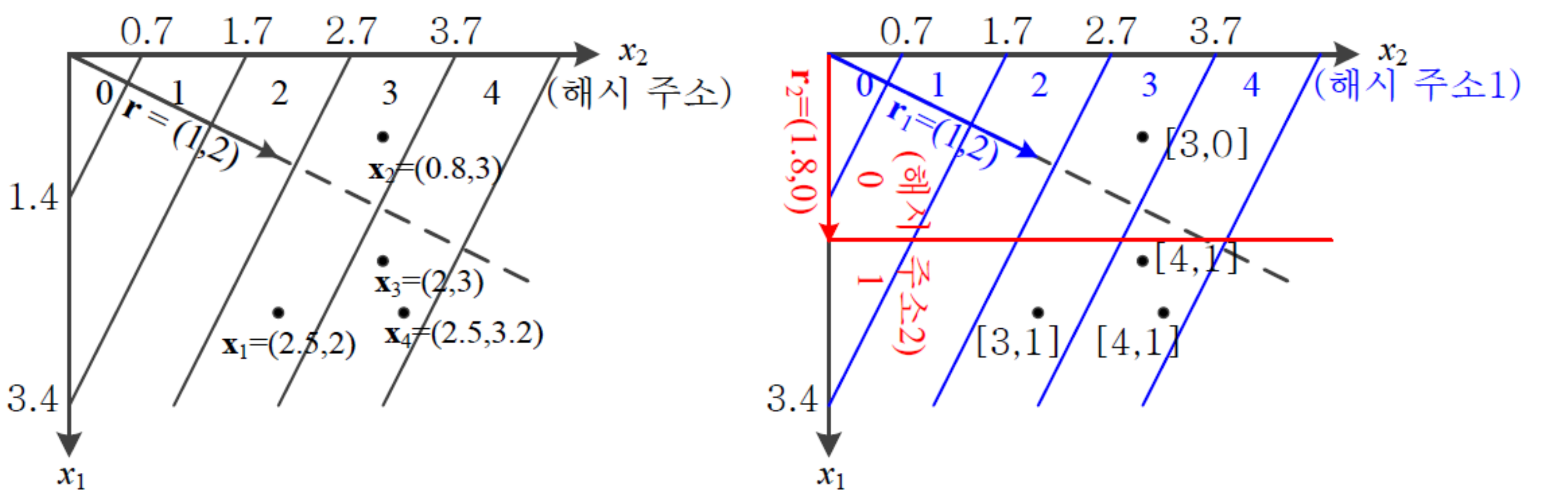
$$h(\mathbf{x}) = \left\lfloor \frac{x_1 + 2x_2 + 0.6}{2} \right\rfloor$$

floor 함수

몇 개의 점을 대상으로 이 해시 함수가 특징 벡터를 어떤 주소로 매핑해 주는지 살펴보자. 해시 함수는 2차원 공간을 띠 모양의 영역으로 분할하는데, 원점에서 오른쪽으로 진행하며 0, 1, 2, …라는 주소를 부여한다. [그림 7-10]은 네 개의 특징 벡터 $\mathbf{x}_1=(2.5, 2)$, $\mathbf{x}_2=(0.8, 3)$, $\mathbf{x}_3=(2,3)$, $\mathbf{x}_4=(2.5, 3.2)$가 어떤 영역으로 매핑되는지 보여준다. [그림 7-10(a)]를 보면 결과적으로 이들은 각각 주소가 3, 3, 4, 4인 통에 담긴다.

$$h(2.5,2) = \left\lfloor \frac{7.1}{2} \right\rfloor = 3,\;\; h(0.8,3) = \left\lfloor \frac{7.4}{2} \right\rfloor = 3,\;\; h(2,3) = \left\lfloor \frac{8.6}{2} \right\rfloor = 4,\;\; h(2.5,3.2) = \left\lfloor \frac{9.5}{2} \right\rfloor = 4$$

7.2.2 해싱



(a) 해시 함수 한 개 사용 (b) 해시 함수 두 개 사용

그림 7-10 해시 함수의 공간 분할과 주소 매핑

이제 두 개의 해시 함수 h_1과 h_2를 사용하는 [그림 7-10(b)]로 관심을 옮겨 보자. h_1은 이전과 같이 $\mathbf{r}_1=(1,2)$, $b=0.6$으로 정의되고, h_2는 $\mathbf{r}_2=(1.8,0)$, $b=0$으로 정의한다고 하자. 이 상황에서는 값 두 개로 주소가 정해진다. 예를 들어 점 $\mathbf{x}_1=(2.5,2)$는 주소 [3,1]을 가진다. 나머지 점의 주소도 계산해 보면, 그림에 표시된 주소를 갖는다. 이때 해시 함수를 두 개 사용한 효과를 관찰해 보자. 왼쪽 그림에서는 $\mathbf{x}_1$과 $\mathbf{x}_2$가 멀리 떨어져 있음에도 불구하고 주소3에 같이 담겨있다. 하지만 두 개의 해시 함수를 사용하는 오른쪽 그림에서는 이들이 각각 [3,1]과 [3,0]이라는 주소를 가져 다른 통에 담겨있는 것을 확인할 수 있다. 서로 가까운 $\mathbf{x}_3$와 $\mathbf{x}_4$는 여전히 같은 통 [4,1]에 들어 있다.

7.2.2 해싱

- 위치의존을 만족하는 해시 함수 여러 개를 쓰면,
  - 가까운 벡터가 같은 통에 담길 확률이 충분히 높을까? → 그렇지 않다.
- 확률을 높이는 추가적인 방안
  - 해시 테이블을 여러 개 사용

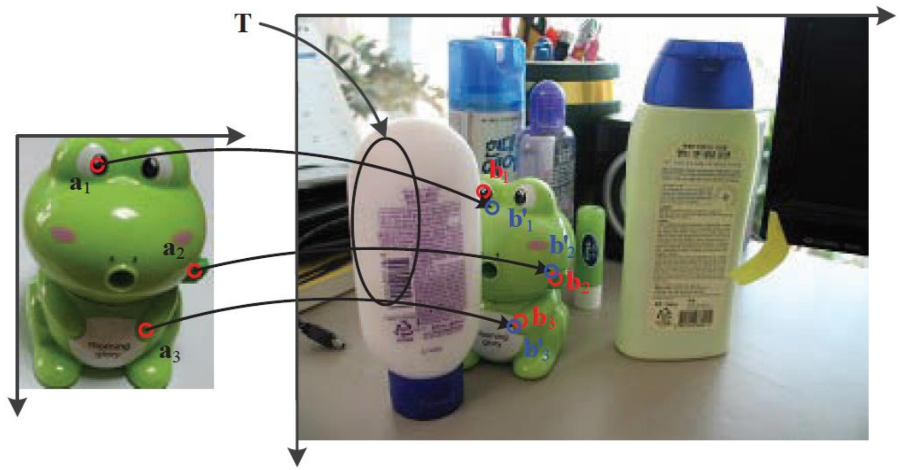
해시 함수 6개를 한꺼번에 쓰면?

해시 함수 2개를 쓰는 해시 테이블 3개

  - 두 벡터가 여러 테이블 중 하나에라도 같은 통에 있으면 매칭으로 분류

7.3 기하 정렬과 변환 추정

- 점 대 점의 매칭이 아니라 강체(rigid object)가 옮겨가는 것을 변환으로 설정하고, 변환식을 추정



(a) 모델 영상 (b) 장면 영상

그림 7-13 최소제곱법으로 물체의 자세 T를 알아내는 사례

7.3.1 최소제곱법과 강인한 추정 기법

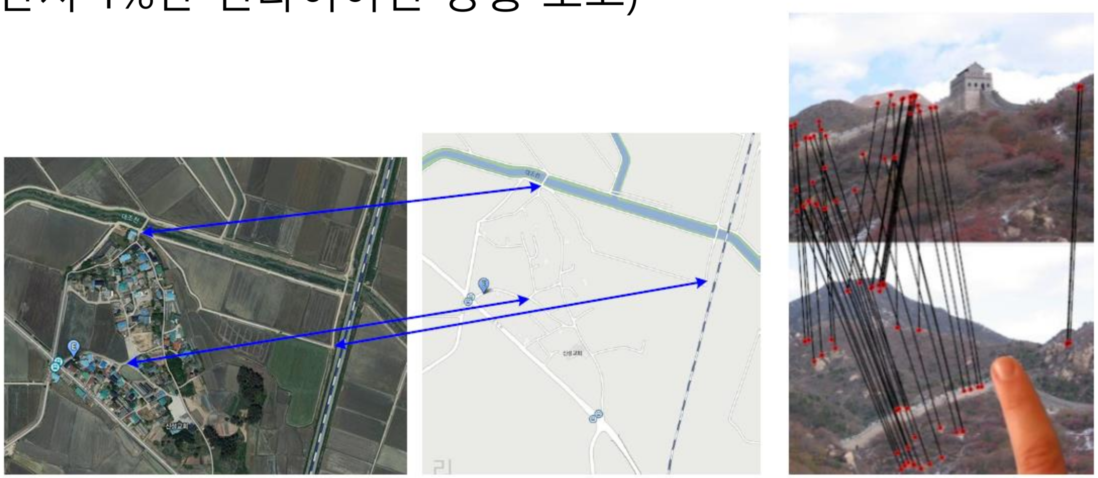
7.3.2 RANSAC

7.3 기하 정렬과 변환 추정

- 지금까지(지역 정보만 사용)는,
  - 특징 벡터로 개별적으로 매칭을 수행 → 아웃라이어 매칭 (거짓 긍정) 발생

  → 기하 정렬(geometrical alignment)을 이용하여 인라이어 집합을 찾아내고, 변환 행렬을 추정해야 함

- 여러 상황
  1. 사람이 개입하여 아웃라이어 없는 경우 : 항공 사진 비교, 의료 영상 정합 등
  2. 아웃라이어 있는 경우 : 파노라마 영상 제작 등 (예, [Lowe2004]는 심한 혼재와 가림이 있는 경우 단지 1%만 인라이어인 상황 보고)



(a) 대응 쌍이 모두 옳음

(b) 거짓 긍정이 포함된 경우

그림 7-11 대응 쌍의 여러 가지 상황

7.3.1 최소제곱법과 강인한 추정 기법

최소 제곱법(least square method)

- 오래 전부터 수학과 통계 분야에서 사용된 기법
- 예) $\boldsymbol{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$를 가장 잘 대표하는 직선을 찾아라.
  - 직관적으로 l_1이 더 좋음 → 수학으로 어떻게 설명하나?

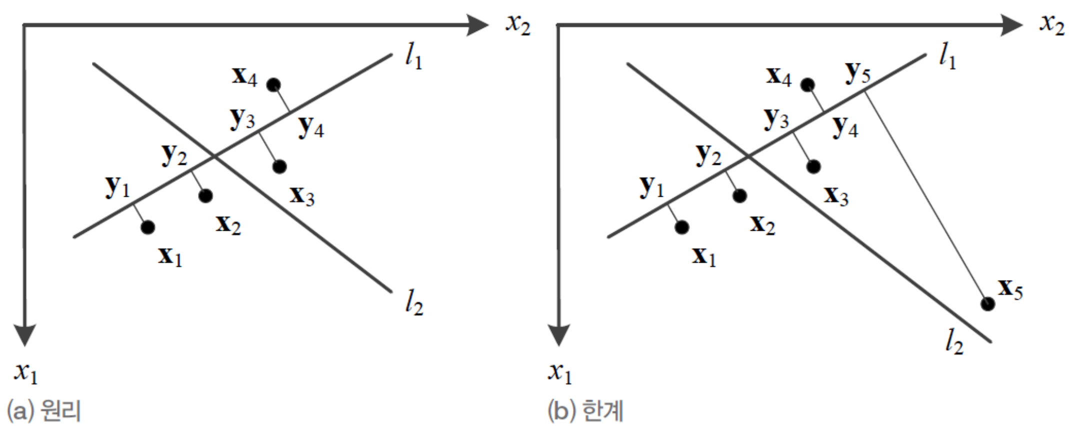


그림 7-12 최소제곱법

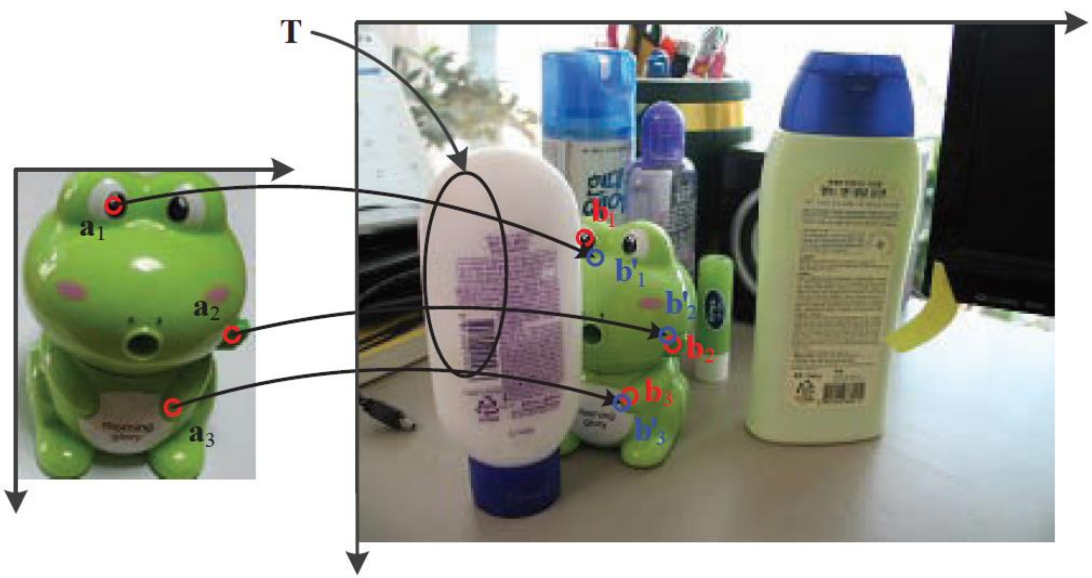
- 직선 l까지의 거리의 합을 오차로 공식화
  - $E(l)$을 최소화하는 l을 찾아라.

$$E(l) = \sum_{i=1}^{n} r_i^2 = \sum_{i=1}^{n} d(\mathbf{x}_i, l)^2 = \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{y}_i\|^2$$

7.3.1 최소제곱법과 강인한 추정 기법

매칭 문제로 확장

- 입력은 매칭 쌍 집합 $\boldsymbol{X}$ ={($\mathbf{a}_1$, $\mathbf{b}_1$), ($\mathbf{a}_2$, $\mathbf{b}_2$), … , ($\mathbf{a}_n$, $\mathbf{b}_n$)}
- 모델은? → 변환 행렬 (이동, 크기, 회전 모두 포함) $\mathbf{T} = \begin{pmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{pmatrix}$



(a) 모델 영상 (b) 장면 영상

그림 7-13 최소제곱법으로 물체의 자세 T를 알아내는 사례

빨간색 점 : 실제 매칭점

파란색 점 : 추정한 T로 예측한 점

7.3.1 최소제곱법과 강인한 추정 기법

$$\begin{pmatrix} b_{i1}' & b_{i2}' \end{pmatrix} = b_i'\begin{pmatrix} y & x \end{pmatrix}$$

$$\begin{pmatrix} a_{i1} & a_{i2} \end{pmatrix} = a_i\begin{pmatrix} y & x \end{pmatrix}$$

- 오차 함수 $E(\mathbf{T})$

$$\mathbf{b}_i' = \mathbf{a}_i\mathbf{T}, \quad \text{풀어쓰면} \quad \begin{pmatrix} b_{i1}' & b_{i2}' & 1 \end{pmatrix} = \begin{pmatrix} a_{i1} & a_{i2} & 1 \end{pmatrix}\begin{pmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{pmatrix} \quad (7.12)$$

$$E(\mathbf{T}) = \sum_{i=1}^{n}\left\|\mathbf{b_i} - \mathbf{b_i}'\right\|^2 = \sum_{i=1}^{n}\left(\left(b_{i1} - (t_{11}a_{i1} + t_{21}a_{i2} + t_{31})\right)^2 + \left(b_{i2} - (t_{12}a_{i1} + t_{22}a_{i2} + t_{32})\right)^2\right) \quad (7.13)$$

- $E(\mathbf{T})$를 최소화하는 $\mathbf{T}$는?
  - E를 t_{ij}로 미분한 도함수 $\frac{\partial E}{\partial t_{ij}} = 0$으로 두고 풀어보면,

T의 요소들

$$\begin{pmatrix} \sum_{i=1}^{n} a_{i1}^2 & \sum_{i=1}^{n} a_{i1}a_{i2} & \sum_{i=1}^{n} a_{i1} & 0 & 0 & 0 \\ \sum_{i=1}^{n} a_{i1}a_{i2} & \sum_{i=1}^{n} a_{i2}^2 & \sum_{i=1}^{n} a_{i2} & 0 & 0 & 0 \\ \sum_{i=1}^{n} a_{i1} & \sum_{i=1}^{n} a_{i2} & \sum_{i=1}^{n} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sum_{i=1}^{n} a_{i1}^2 & \sum_{i=1}^{n} a_{i1}a_{i2} & \sum_{i=1}^{n} a_{i1} \\ 0 & 0 & 0 & \sum_{i=1}^{n} a_{i1}a_{i2} & \sum_{i=1}^{n} a_{i2}^2 & \sum_{i=1}^{n} a_{i2} \\ 0 & 0 & 0 & \sum_{i=1}^{n} a_{i1} & \sum_{i=1}^{n} a_{i2} & \sum_{i=1}^{n} 1 \end{pmatrix}\begin{pmatrix} t_{11} \\ t_{21} \\ t_{31} \\ t_{12} \\ t_{22} \\ t_{32} \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n} a_{i1}b_{i1} \\ \sum_{i=1}^{n} a_{i2}b_{i1} \\ \sum_{i=1}^{n} b_{i1} \\ \sum_{i=1}^{n} a_{i1}b_{i2} \\ \sum_{i=1}^{n} a_{i2}b_{i2} \\ \sum_{i=1}^{n} b_{i2} \end{pmatrix} \quad (7.14)$$

7.3.1 최소제곱법과 강인한 추정 기법

- 최소제곱법은 아웃라이어(outlier) 있으면 오작동
  - 예) 아웃라이어 $\mathbf{x}_5$가 포함되면, l_2를 선호

    → 이런 경우에는 강인한 추정 기법 필요

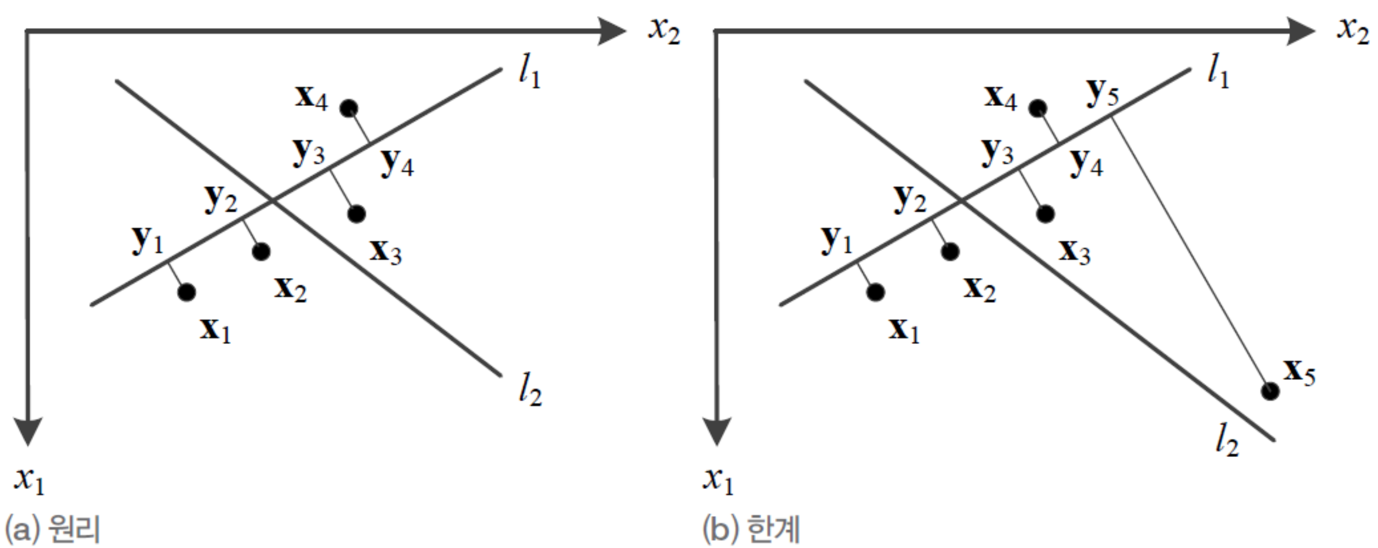


그림 7-12 최소제곱법

7.3.1 최소제곱법과 강인한 추정 기법

강인한 추정 기법

- 최소제곱법을 최적화 문제로 다시 써보면,

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{n} r_i^2 \tag{7.15}$$

- 아웃라이어의 영향력을 약화시키는 함수 $\rho(.)$를 사용하는 M-추정(M-estimator)

$$\text{M - 추정} : \hat{\theta} = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{n} \rho(r_i) \tag{7.16}$$

$$\rho(r) = \begin{cases} \frac{1}{2} r^2, & |r| \le c \\ \frac{1}{2} c(2|r| - c), & |r| > c \end{cases} \tag{7.18}$$

- 아웃라이어는 중앙값 계산하는 단계까지만 참여하는 최소제곱중앙값(LMedS, Least-median of Squares)

$$\text{최소제곱중앙값} : \hat{\theta} = \underset{\theta}{\operatorname{argmin}} \underset{i}{med}\, r_i^2 \tag{7.17}$$

7.3.2 RANSAC(RANdom Sample Consensus)

- 원리
  - 직선 검출하는 3장의 그림 3-31과 같은 원리

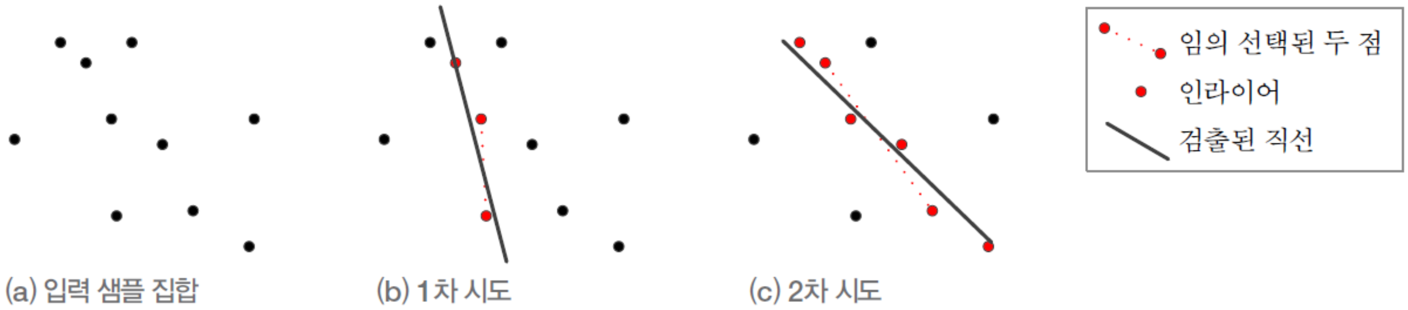


그림 3-31 RANSAC의 원리

- 여기서는,
  - 매칭 쌍 집합 $\boldsymbol{X}$={($\mathbf{a}_1$, $\mathbf{b}_1$), ($\mathbf{a}_2$, $\mathbf{b}_2$), … , ($\mathbf{a}_n$, $\mathbf{b}_n$)}을 처리할 수 있게 확장

7.3.2 RANSAC

알고리즘 7-9 기하 변환을 추정하기 위한 RANSAC

입력 : $X=\{(\mathbf{a}_i,\mathbf{b}_i), i=1,2,\cdots,n\}$ // 매칭 쌍 집합

반복 횟수 k, 인라이어 판단 t, 인라이어 집합의 크기 d, 적합 오차 e

출력 : 기하 변환 행렬 **T**

$Q=\emptyset$;

for(j=1 to k) {

  X에서 세 개 대응점 쌍을 임의로 선택한다.

  이들 세 쌍을 입력으로 식 (7.14)를 풀어 $\mathbf{T}_j$를 추정한다.

  이들 세 쌍으로 집합 *inlier*를 초기화한다.

  for(이 세 쌍을 제외한 X의 요소 p 각각에 대해) {

    if(p가 허용 오차 t 이내로 $\mathbf{T}_j$에 적합) p를 *inlier*에 넣는다.

  }

  if($|inlier| \geq d$) // 집합 *inlier*가 d개 이상의 샘플을 가지면

    *inlier*에 있는 모든 샘플을 가지고 새로운 $\mathbf{T}_j$를 계산한다.

  if($\mathbf{T}_j$의 적합 오류$<e$) $\mathbf{T}_j$를 집합 Q에 넣는다.

}

Q에 있는 변환 행렬 중 가장 좋은 것을 **T**로 취한다.

7.3.2 RANSAC

- PROSAC [Chum2005]
  - 행 3에서 대응 쌍의 품질에 따라 선택 확률을 결정하여 성능 향상 꾀함

3 | 매칭 점수가 높을수록 선택 확률이 높은 방식에 따라, X에서 세 쌍을 선택한다.

- 반복 횟수 k를 결정하는 방법
  - 옳은 답을 기대할 수 없는 확률을 p_{thres}보다 낮게 유지하고 싶다면,

$$(1-q^3)^k < p_{thres}$$

$$\text{양변에 log를 취하면, } k\log(1-q^3) < \log(p_{thres}) \tag{7.19}$$

$$\text{따라서 } k > \frac{\log(p_{thres})}{\log(1-q^3)}$$

7.4 웹과 모바일 응용

7.4.1 파노라마 영상 제작

7.4.2 사진 관광

7.4 웹과 모바일 응용

- 웹과 모바일 환경에서 '인터넷 비전' 연구 분야 태동
  - 방대한 영상 발생 (예, Flickr에 하루에 올라오는 영상은 350만장)
  - 새로운 응용 분야 창출 (예, 파노라마, 사진 관광, 증강 현실 등)
  - 문제를 푸는 새로운 접근 방법 개발 (예, SNS 정보 활용한 얼굴 인식 성능 향상)

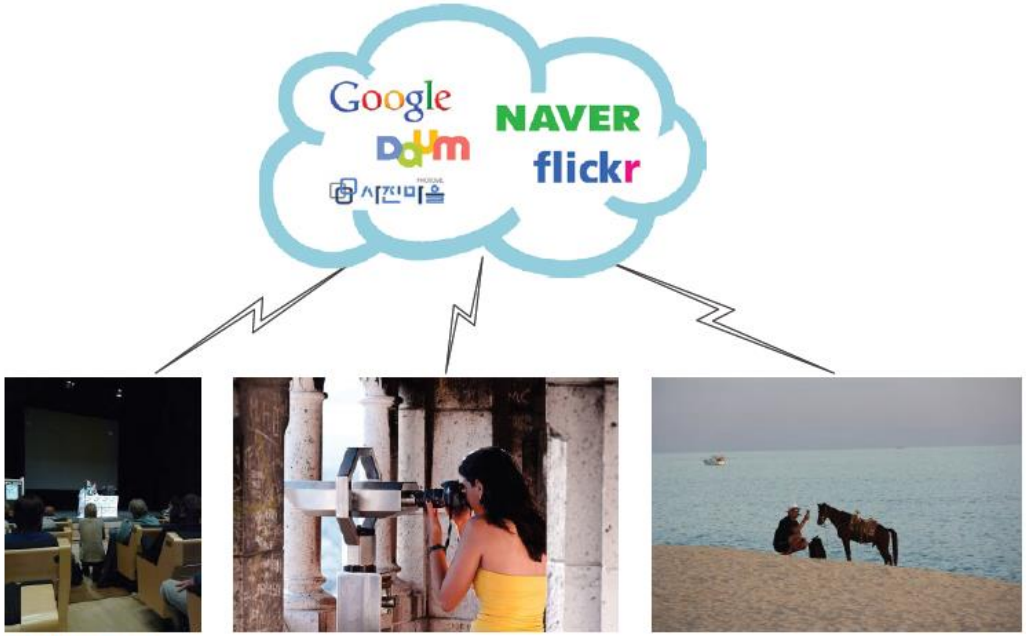


그림 7-14 인터넷에 쌓이는 영상

7.4.1 파노라마 영상 제작

- 제작 사례

그림 7-15 파노라마 영상

- 스마트폰 앱
  - Photosynth
  - AutoStitch

7.4.1 파노라마 영상 제작

■ 제작 과정

그림 7-16 파노라마 제작 과정

7.4.1 파노라마 영상 제작

■ 알고리즘

- RANSAC은 이웃한 두 영상 사이의 변환을 추정해 줌
- 번들 조정은 영상 집합 전체에 대해 변환 행렬을 미세 조정함
- 이어 붙이기는 다중 밴드 결합 알고리즘 [Burt83b] 사용

알고리즘 7-10 파노라마 영상 제작

입력 : 같은 장면을 찍은 영상 집합 f_i, $1 \le i \le k$ // 시점이 $i=1, 2, \cdots, k$ 순서라고 가정

출력 : 파노라마 영상 p

```
k개의 모든 영상에서 지역 특징을 추출한다. // 예를 들어 SIFT
for(i=1 to k−1) {  // i와 i+1번째 영상을 이어 붙인다.
   kd 트리 또는 위치의존 해싱을 이용하여 f_i와 f_{i+1} 사이의 대응점을 찾는다.
   [알고리즘 7-9(RANSAC)]를 이용하여 f_i와 f_{i+1} 사이의 변환 행렬 T_i를 추정한다.
}
번들 조정을 수행하여 T_i, i=1, 2, ···, k−1을 보다 정확한 값으로 조정한다.
T_i 정보를 이용하여 k개의 영상을 이어 붙인다.
```

7.4.2 사진 관광

구조 추정 문제

- 같은 장면을 여러 시점에서 찍은 영상들로부터 3차원 정보 복원
  - 장면에 나타난 물체의 자세 정보 복원
  - 카메라 시점 정보 복원
- 다양한 응용 문제에 적용 가능 → 예) 사진 관광, 증강 현실 등

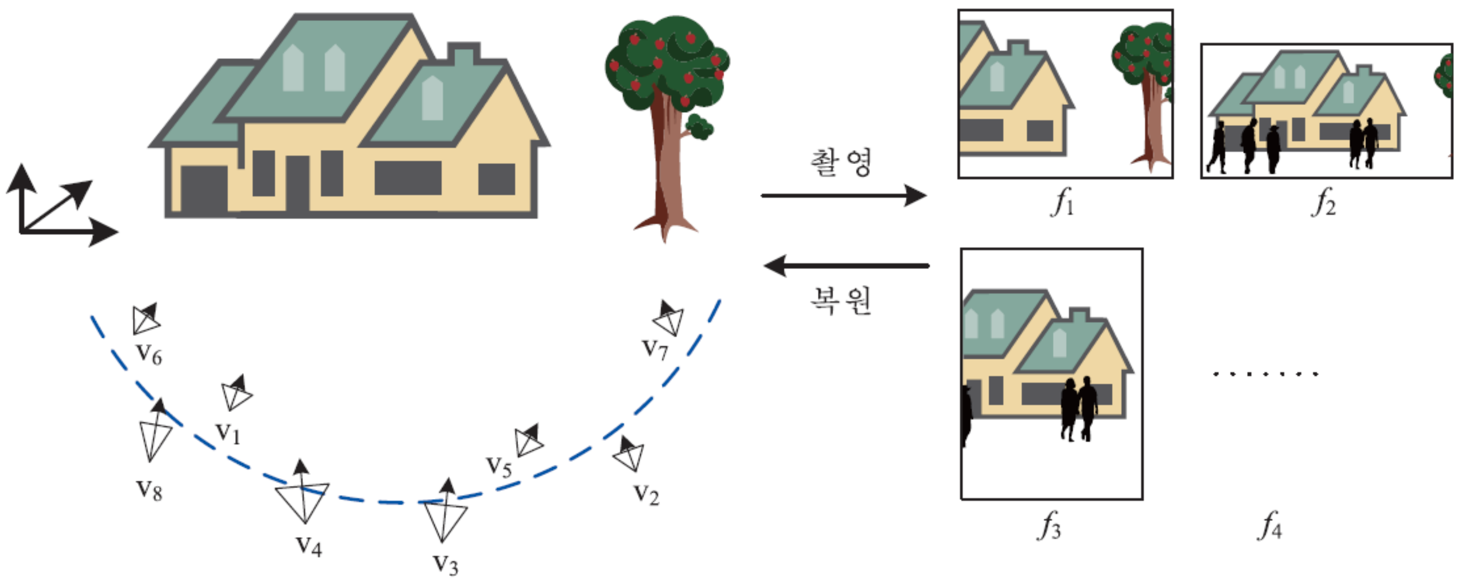


그림 7-17 같은 장소를 여러 시점에서 촬영 – 시점과 3차원 장면을 복원할 수 있을까?

7.4.2 사진 관광

- 사진 관광 : 3차원 둘러보기
  - 파란 삼각형은 복원된 카메라 시점
  - 검은 선은 경로 계획 알고리즘으로 계산한 매끄러운 경로 (빨간 선분은 카메라가 바라보는 방향)

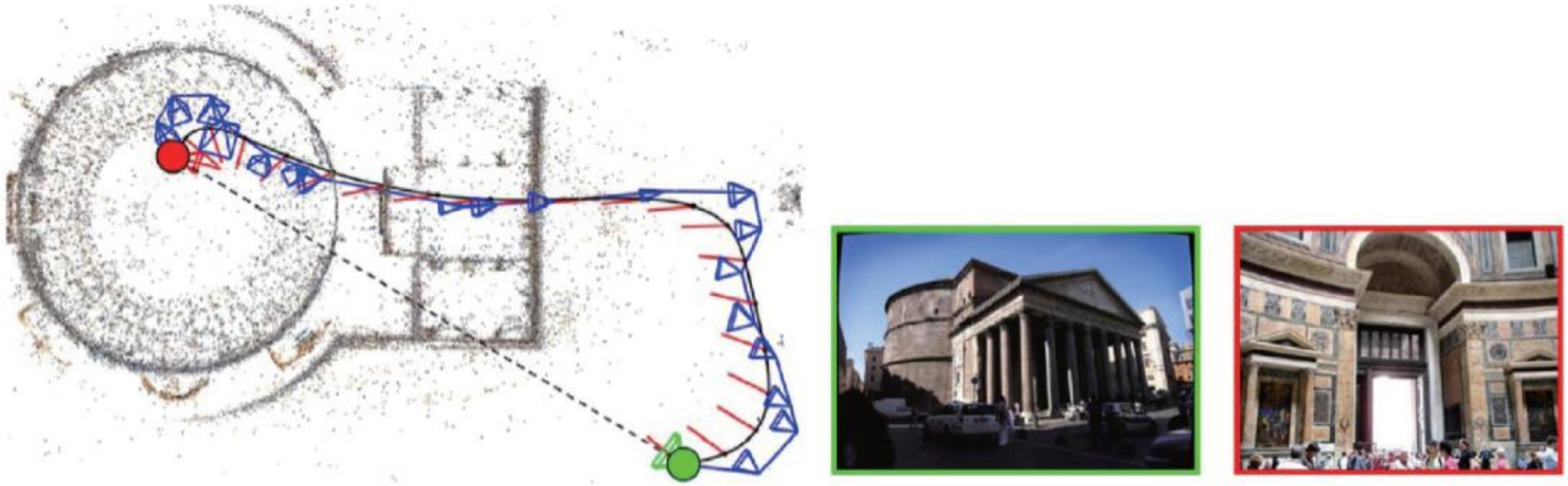


그림 7-18 사진 관광

7.4.2 사진 관광

■ 사진 관광 : 자동 주석 붙이기

- 모델 영상에 주석을 입력해 두면, 새로운 영상에 대해 주석 위치를 자동 추정하여 보여줌 (일종의 증강 현실)

그림 7-19 자동 주석 붙이기

7.4.2 사진 관광

- 사진 관광에 필요한 정보를 추정하는 알고리즘
  - 시점 변화가 크므로,
    - 영상 간의 겹침 관계 알아내는 과정 필요
    - 겹침 정보를 그래프로 표현
  - 그래프의 연결요소를 찾고, 각각에 대해 번들 조정을 수행

7.4.2 사진 관광

알고리즘 7-11 사진 관광에 필요한 정보 추정

입력 : 같은 장면을 찍은 영상 집합 f_i, $1 \le i \le k$, 임계값 t와 c

출력 : 카메라 시점과 3차원 물체

```
모든 영상에서 지역 특징을 추출한다. // 예를 들어 SIFT
kd 트리 또는 위치의존 해싱을 이용하여 대응점을 찾는다.
for(i=1 to k)
  for(j=1 to k)
    if(i≠j이고 f_i와 f_j 사이에 대응점이 t개 이상이면) { // 겹침 조사
      RANSAC을 적용하여 변환 행렬 T를 구한다.
      T의 신뢰도가 c 이상이면 f_i와 f_j 사이에 에지를 부여한다.
    }
for(그래프의 연결요소 각각에 대해)
  번들 조정을 수행하여 카메라 시점과 3차원 점을 구한다.
```