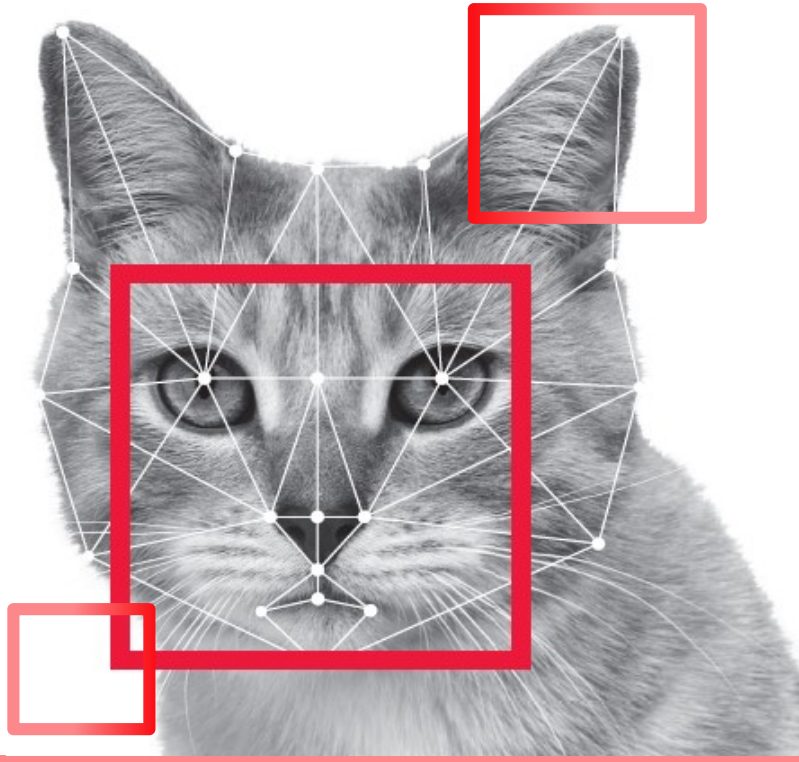
한빛아카데미 Hanbit Academy, Inc.

# COMPUTER VISION 컴퓨터 비전

기본 개념부터 최신 모바일 응용 예까지



## 7장. 매칭

# PREVIEW

## 매칭

- 어떤 대상을 다른 것과 비교하여 같은 것인지 알아내는 과정
- 여러 가지 문제를 해결하는 열쇠 (물체 인식, 자세 추정, 스테레오, 증강 현실 등)

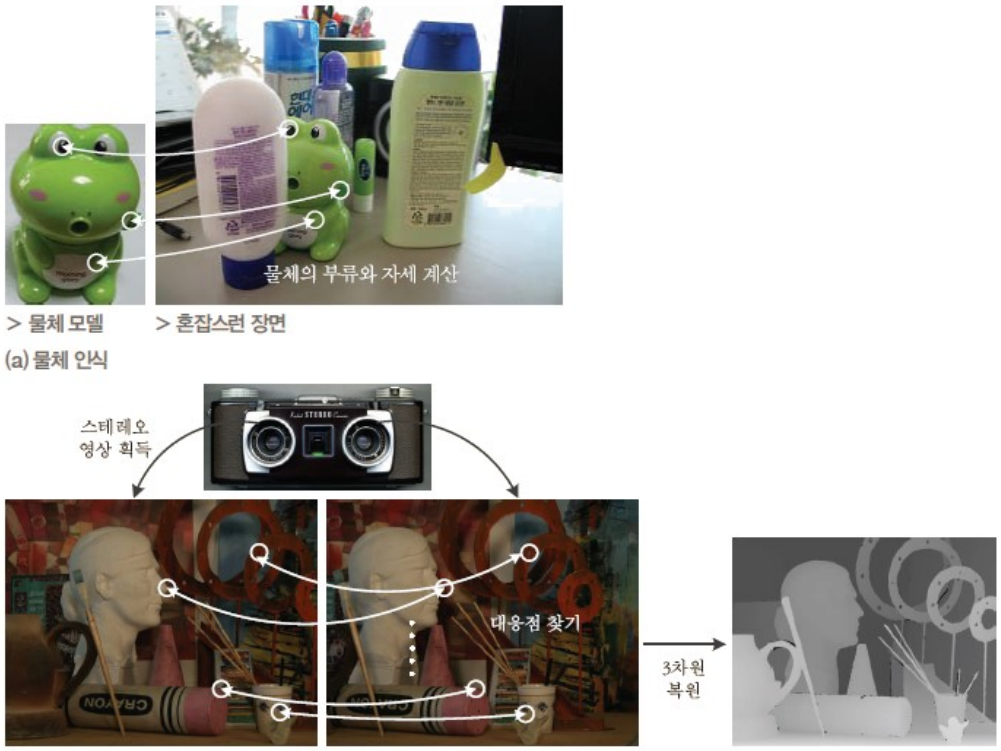



그림 7-1 매칭을 이용한 응용 문제 해결

# PREVIEW

- 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 척도에 관해 살펴본다.

2. 빠른 최근접 이웃 탐색

   → 매칭 속도를 올릴 수 있는 방법으로 kd 트리와 해싱에 대해 살펴본다.

3. 기하 정렬과 변환 추청

   → 신뢰도가 높은 매칭 쌍을 고르는 방법을 다룬다.

4. 웹과 모바일 응용

   → 매칭을 활용한 파노라마, 사진 관광 응용 사례를 살펴본다.

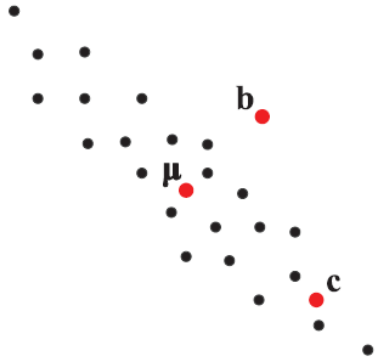
# 7.1 매칭의 기초

## 7.1.1 거리 척도

## 7.1.2 매칭 전략과 성능 분석

# 7.1.1 거리 척도

- 유클리디안 거리 vs. 마할라노비스 거리




← **μ는 b와 c 중 누구와 더 가까운가?**

그림 7-2 확률 분포 속의 거리

  - 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.)

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

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

# 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 $d_E(\boldsymbol{\mu},\mathbf{b})=2$, $d_E(\boldsymbol{\mu},\mathbf{c})=3$이므로 b가 c보다 $\boldsymbol{\mu}$에 더 가깝다.

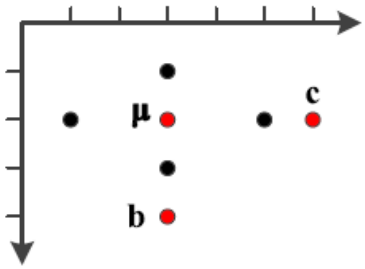



그림 7-3 마할라노비스 거리 예제

이제 확률 분포를 고려한 거리를 계산해 보자. 이 분포의 평균은 $\boldsymbol{\mu}=(2,3)$이고 공분산 행렬은 $\boldsymbol{\Sigma}=\begin{pmatrix}0.5 & 0\\ 0 & 2\end{pmatrix}$이다. $\boldsymbol{\Sigma}$의 역행렬을 구하면, $\boldsymbol{\Sigma}^{-1}=\begin{pmatrix}2 & 0\\ 0 & 0.5\end{pmatrix}$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.

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

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

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

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

$$\text{b와 c 사이의 마할라노비스 거리 } d_M(\mathbf{b},\mathbf{c}) = \sqrt{(4-2 \quad 3-6)\begin{pmatrix}2 & 0\\ 0 & 0.5\end{pmatrix}\begin{pmatrix}4-2\\ 3-6\end{pmatrix}} = 4.6368$$

# 7.1.1 거리 척도

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

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

**예제 7-2 화이트닝 변환**

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

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

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

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

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.

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

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

# 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

- 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

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

# 7.1.2 매칭 전략과 성능 분석

## ROC를 이용한 성능 분석

- 임계값 $T$가 낮으면 거짓 부정 많아지고, 높으면 거짓 긍정이 많아짐
  - $T$를 점점 키우면서 측정한 거짓 긍정률과 참 긍정률을 나타낸 그래프가 ROC
- 왼쪽 위 구석에 가까울수록 좋은 성능
- AUC (곡선 아래 면적): 성능을 하나의 수치로 표현할 때 사용

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

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

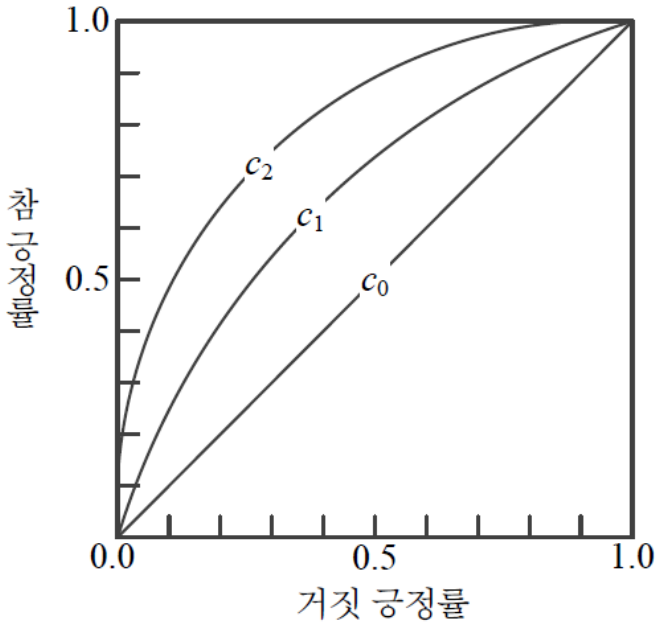



그림 7-4 ROC 성능 분석

# 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

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

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

# 7.2 빠른 최근접 이웃 탐색

## 7.2.1 kd 트리

## 7.2.2 해싱

# 7.2 빠른 최근접 이웃 탐색

## 순진한 알고리즘

- 모든 쌍을 일일이 검사 → 시간이 넉넉한 상황에서만 활용 가능

**알고리즘 7-1 순진한 매칭 알고리즘**

**입력 :** 첫 번째 영상의 특징 벡터 $\mathbf{a}_i$, $1 \le i \le m$, 두 번째 영상의 특징 벡터 $\mathbf{b}_j$, $1 \le j \le n$, 거리 임계값 $T$

**출력 :** 매칭 쌍 리스트 $mlist$

```
1  mlist=∅;
2  for(i=1 to m) {
3    shortest=∞;
4    for(j=1 to n) {
5      dist = d(a_i, b_j);
6      if(dist<shortest) {match=j; shortest = dist;}
7    }
8    if(shortest<T) mlist=mlist ∪(a_i, b_match);
9  }
```

# 7.2 빠른 최근접 이웃 탐색

- 특징 벡터를 미리 인덱싱해 두는 효율적인 알고리즘
  - 두 알고리즘: *kd* 트리와 위치의존 해싱

**알고리즘 7-2 빠른 매칭 알고리즘**

**입력 :** 첫 번째 영상의 특징 벡터 $\mathbf{a}_i$, $1 \le i \le m$, 두 번째 영상의 특징 벡터 $\mathbf{b}_j$, $1 \le j \le n$, 거리 임계값 $T$

**출력 :** 매칭쌍 리스트 $mlist$

```
1  mlist = ∅;
2  for(i=1 to m) {
3    kd트리나 해싱 알고리즘으로 a_i의 최근접 또는 근사 최근접 이웃 b_match를 탐색한다.
4    if(d(a_i, b_match)<T) mlist = mlist ∪ (a_i, b_match);
5  }
```

  - 일반적인 탐색 기법으로 데이터마이닝, 빅데이터, 생물 정보학 등에서도 활용

# 7.2.1 *kd* 트리

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

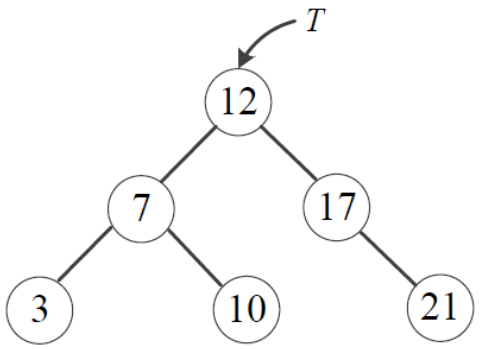


그림 7–5 이진검색 트리

균형이 잡힌 경우
탐색시간 $O(\log n)$

**알고리즘 7–3 이진검색 트리의 검색**

**입력 :** 이진검색 트리 $T$, 검색 키 $v$

**출력 :** 검색 결과

```
1   t=search(T, v);
2   if(t=Nil) T 안에 v가 없음을 알린다.
3   else t를 검색 결과로 취한다.
4   function search(t,v) {
5     if(t=Nil or t.key=v) return t;
6     else {
7       if(v<t.key) return search(t.leftchild, v);
8       else return search(t.rightchild, v);
9     }
10  }
```

# 7.2.1 *k*d 트리

- BST를 적용할 수 있나?
  - 매칭 문제
    - 검색 키(특징 벡터)가 여러 개의 실수로 구성된 벡터임
    - 동일한 값을 갖는 노드를 찾는 것이 아니라, 최근접 이웃을 찾음

    →BST를 그대로 적용 불가능

- *k*d 트리
  - 두 가지 다른 점을 수용할 수 있게 BST를 확장한 기법 [Bently75]

# 7.2.1 *k*d 트리

## 표기

- $n$개의 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 $X$를 두 개의 부분 집합 $X_{left}$와 $X_{right}$ 나눔
- 이때 분할 기준을 어떻게 선택하나?
  - $d$개의 차원(축) 중에 어는 것을 쓸 것인가?
    - 분할 효과를 극대화하려면 각 차원의 분산을 계산한 후, 최대 분산을 갖는 축 $k$를 선택
  - 축을 선택했다면 $n$개의 샘플 중 어는 것을 기준으로 $X$를 분할할 것인가?
    - $X_{left}$와 $X_{right}$의 크기를 갖게 하여 균형 잡힌 트리를 만들어야 함.
    - $X$를 차원 $k$로 정렬하고, 그 결과의 중앙 값을 분할 기준으로 삼는다.
- $X$를 $X_{left}$와 $X_{right}$로 분할한 후, 각각에 같은 과정을 재귀적으로 반복하면 $k$d트리 완성

# 7.2.1 kd 트리

**알고리즘 7-4 kd 트리 만들기**

**입력 :** 특징 벡터 집합 $X=\{\mathbf{x}_i, i=1,2,\cdots,n\}$

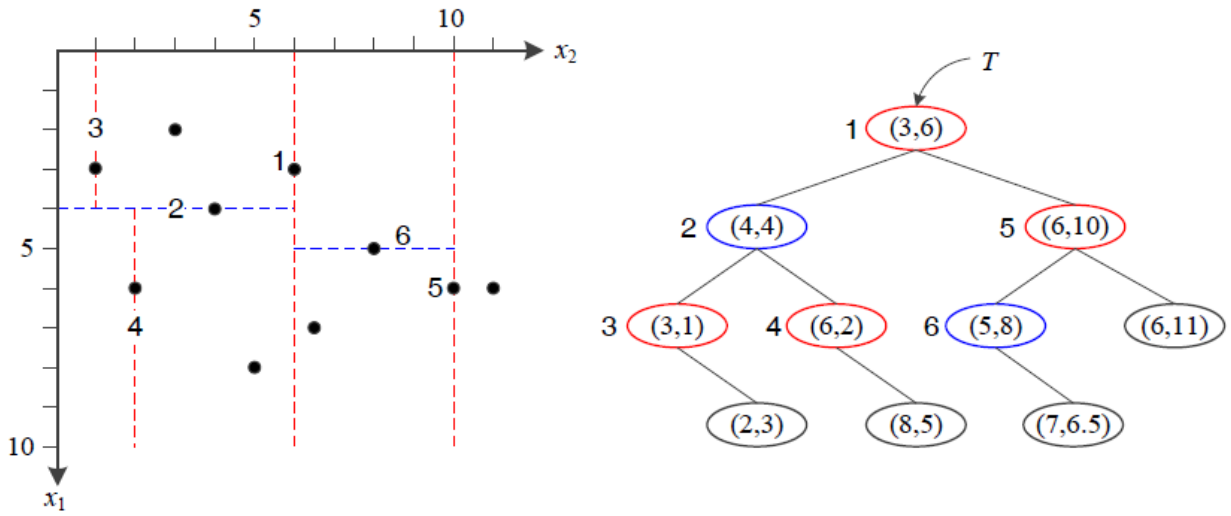
**출력 :** kd 트리 $T$

```
1   T=make_kdtree(X);
2   function make_kdtree(X) {
3     if(X=Ø) return Nil;
4     else if(|X|=1) {  // 단말 노드
5       트리 노드 node를 생성한다.
6       node.vector = x_m;  // x_m은 X에 있는 벡터
7       node.leftchild = node.rightchild =Nil;
8       return node;
9     }
10    else {
11      d개의 차원 각각에 대해 X의 분산을 구하고 최대 분산을 갖는 차원을 k라 하자.
12      X를 k차원을 기준으로 정렬하여 리스트 X_sorted를 만든다.
13      X_sorted에서 중앙값을 x_m, 왼쪽 부분집합을 X_left, 오른쪽 부분집합을 X_right라 하자.
14      트리 노드 node를 생성한다.
15      node.dim=k;  // 어느 차원으로 분할하는지
16      node.vector = x_m;  // 어떤 특징 벡터로 분할하는지
17      node.leftchild =make_kdtree(X_left);
18      node.rightchild =make_kdtree(X_right);
19      return node;
20    }
21  }
```

# 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 $k$는 2가 된다. 두 번째 차원을 기준으로 $X$를 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 $T$가 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 $x_2$축을 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. $x_1$축이 기준이라면 파란색, $x_2$축이 기준이라면 빨간색이다.

# 7.2.1 *k*d 트리

- *k*d 트리에서 최근접 이웃 탐색
  - 새로운 특징 벡터 **x**가 입력되면 **x**의 최근접 이웃을 어떻게 찾을까?
  - 예) **x**가 =(7, 5.5)
    - 루트가 $x_2$축을 기준으로 하므로 5.5를 6과 비교하고 작으므로 왼쪽으로 분기
    - (4,4) 노드가 $x_1$축을 기준으로 하므로 7과 4를 비교하고 크므로 오른쪽으로 분기
    - 반복하면 (8,5) 노드에 도착

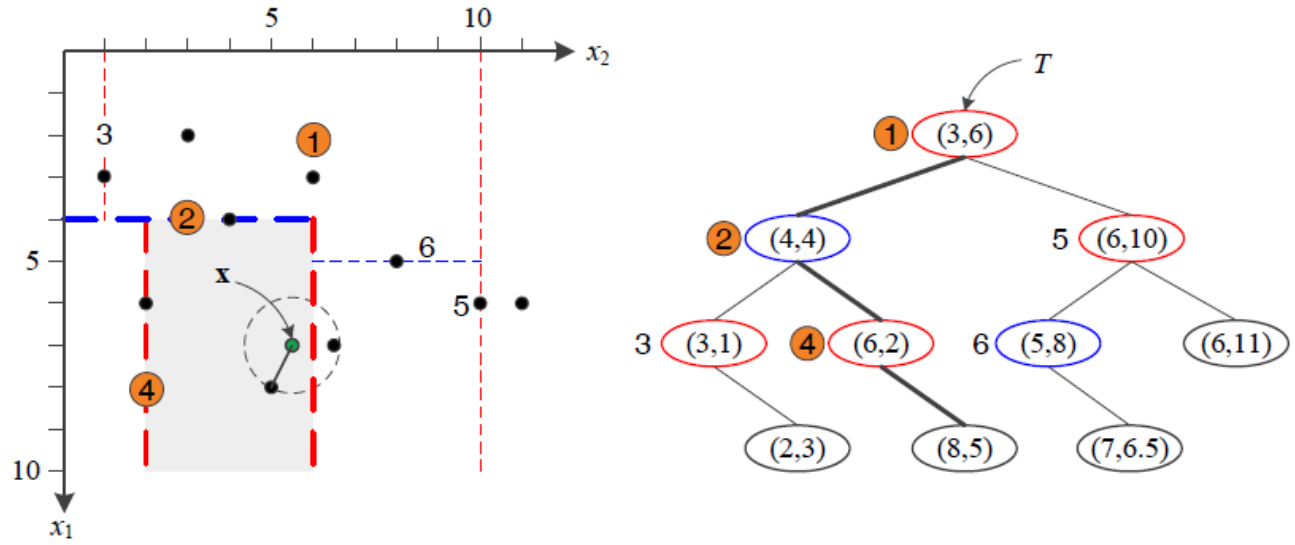



그림 7-7 kd 트리에서 검색하는 예

# 7.2.1 *k*d 트리

- *k*d 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 분기를 적용하면서 ④→②→① 순으로 처리

- *d*=10을 넘으면 순진한 알고리즘과 비슷한 낮은 속도
  - 어떻게 시간 효율을 회복할 수 있을까?

# 7.2.1 *kd* 트리

**알고리즘 7–5 kd 트리에서 최근접 찾기**

**입력 :** *kd* 트리 *T*, 탐색할 특징 벡터 **x**

**출력 :** 최근접 이웃 *nearest*

```
1  stack s=Ø; // 백트래킹을 위해, 지나온 노드를 저장할 스택을 생성한다.
2  current_best=∞;
3  nearest=Nil;
4  search_kdtree(T,x);
5  function search_kdtree(T,x) {
6    t=T;
7    while(not is_leaf(t)) { // 단말 노드를 찾는다.
8      if(x[t.dim]<t.vector[t.dim]) {s.push(t,"right"); t=t.leftchild;}
9      else {s.push(t,"left"); t=t.rightchild;}
10   }
11   d=dist(x,t.vector);
12   if(d<current_best) {current_best=d; nearest=t;}
13   while(s≠Ø) {   // 거쳐온 노드 각각에 대해 최근접 가능성 여부를 확인(백트랙킹)
14     (t1,other_side)=s.pop();
15     d=dist(x,t1.vector);
16     if(d<current_best) {current_best=d; nearest=t1;}
17     if(x에서 t1의 분할 평면까지 거리<current_best) {  // 건너편에 더 가까운 것 있을 수 있음
18       t1의 other_side 자식 노드를 tother라 하자.
19       search_kdtree(tother,x);
20     }
21   }
22 }
```

# 7.2.1 *k*d 트리

- 근사 최근접 이웃 탐색
  - 최적 칸 우선 탐색
    - 스택 대신 우선순위 큐인 힙을 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

# 7.2.1 *kd* 트리

**알고리즘 7-6 kd 트리에서 근사 최근접 이웃 찾기**

**입력 :** kd 트리 $T$, 탐색할 특징 벡터 **x**, 허용된 최대 조사 횟수 *try_allowed*
**출력 :** 근사 최근접 이웃 *nearest*

```
1  heap s=∅; // 백트래킹을 위해, 지나온 노드를 저장할 힙을 생성한다.
2  current_best=∞;
3  nearest=Nil;
4  try=0;  // 비교 횟수를 센다.
5  search_kdtree(T,x)
6  function search_kdtree(T,x) {
7    t=T;
8    while(not is_leaf(t)) {  // 단말 노드를 찾는다.
9      if(x[t.dim]< t.vector[t.dim]) {s.push(t,"right"); t=t.leftchild;}
10     else {s.push(t,"left"); t=t.rightchild;}
11   }
12   d=dist(x,t.vector);
13   if(d<current_best) {current_best=d; nearest=t;}
14   try++;
15   if(try>try_allowed) 알고리즘을 끝낸다.
16   while(s≠∅) {  // 거쳐온 노드 각각에 대해 최근접 가능성 여부를 확인(백트래킹)
17     (t1,other_side)=s.pop();
18     d=dist(x,t1.vector);
19     if(d<current_best) {current_best=d; nearest=t1;}
20     if(x에서 t1의 분할 평면까지 거리<current_best) { // 건너편에 더 가까운 것 있을 수 있음
21       t1의 other_side 노드를 t_other라 하자.
22       search_kdtree(t_other,x);
23     }
24   }
25 }
```

허용된 만큼만 백트랙킹

# 7.2.2 해싱

## 해싱의 원리

- 해시 함수는 키 값을 해시 테이블의 주소로 변환
- 테이블에 골고루 배치할수록 좋은 해시 함수
- 충돌 해결책 필요

해시 함수 $h(x)=x \bmod 13$

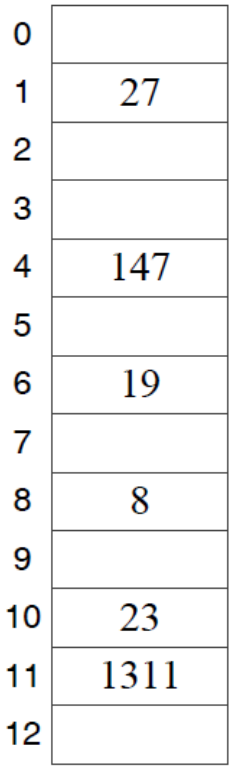



그림 7-8 해싱의 원리

# 7.2.2 해싱

- 매칭에 적용
  - 일반 해싱과 다른 점
    - 키는 단일 값이 아니라 실수 벡터임
    - 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 해싱으로 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

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

  - 위치의존의 의미는?

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

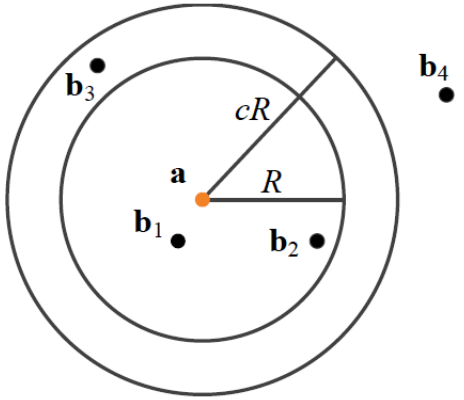



그림 7-9 조건식 (7.8)의 의미

# 7.2.2 해싱

- *H*를 어떻게 만드나?
  - 여럿 개발되어 있는데, 식 (7.9)는 그 중 하나
  - 난수로 **r**과 *b*를 설정하여 원하는 수만큼 함수 생성 가능

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

- 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**

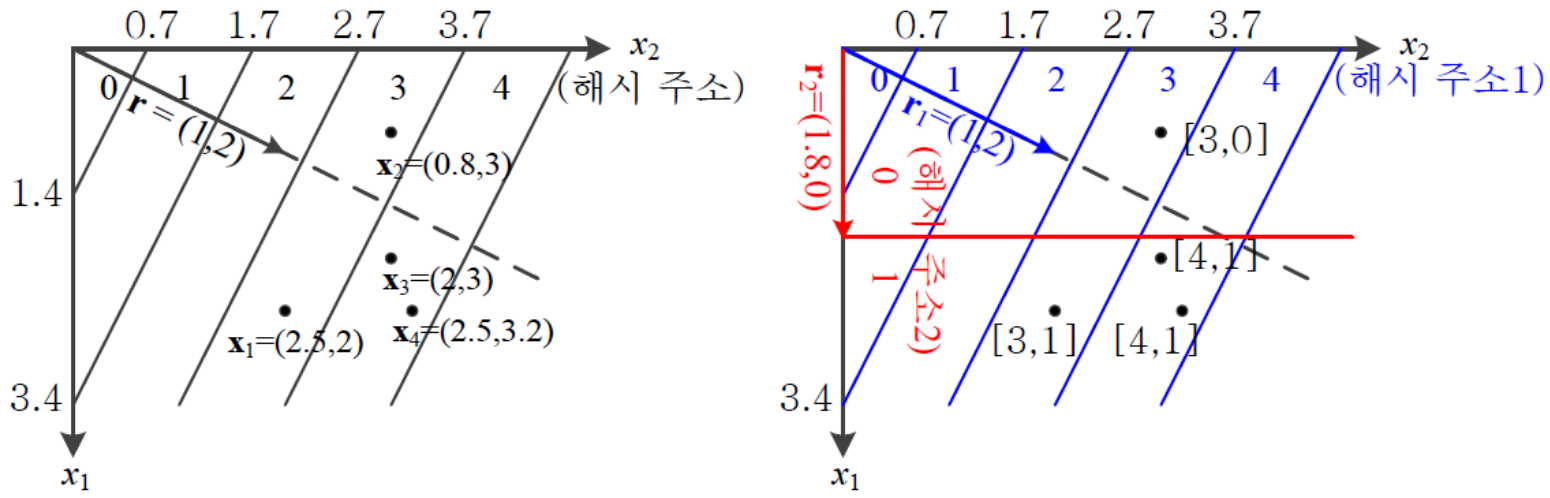
특징 벡터는 $\mathbf{x}=(x_1, x_2)$로 표현되는 2차원이라 가정한다. $w$는 2로 설정되어 있고, 난수를 생성하여 $\mathbf{r}=(1, 2)$, $b=0.6$을 얻었다고 하자. 식 (7.9)에 따른 해시 함수는 다음과 같다.

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

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.

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

# 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 $h_1$과 $h_2$를 사용하는 [그림 7-10(b)]로 관심을 옮겨 보자. $h_1$은 이전과 같이 $\mathbf{r}_1=(1,2)$, $b=0.6$으로 정의되고, $h_2$는 $\mathbf{r}_2=(1.8,0)$, $b=0$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
  - 가까운 두 벡터가 여러 테이블 중 하나에라도 같은 통에 있으면 성공

**알고리즘 7-7 위치의존 해시 테이블의 구축**

**입력** : 특징 벡터 집합 $X=\{\mathbf{x}_i, i=1,2,\cdots,n\}$, 해시 테이블이 사용하는 해시 함수의 개수 $k$, 해시 테이블의 개수 $L$

**출력** : $L$개의 해시 테이블

```
1  for(j=1 to L) {
2    for(i=1 to k) {
3      가우시안 분포에 따른 난수를 생성하여 (r_i, b_i)를 설정한다.
4      (r_i, b_i)로 해시 함수 h_i를 만든다.
5    }
6    해시 함수 g_j=(h_1, h_2, ···, h_k)를 만든다.
7  }
8  for(i=1 to n)
9    for(j=1 to L) x_i를 g_j로 해싱하여 해당 주소의 통에 담는다.
```

# 7.2.2 해싱

## 검색 알고리즘

**알고리즘 7-8 위치의존 해시 테이블에서 검색**

**입력** : $L$개의 해시 테이블, 특징 벡터 $\mathbf{x}$, 매개변수 $R$과 $N$

**출력** : 근사 최근접 이웃 $\mathbf{x}_{nearest}$

```
1  Q=Ø; // 근사 최근접 이웃을 저장
2  for(j=1 to L) {
3    j번째 해시 테이블에서 주소 g_j(x)인 통을 조사한다.
4    이 통에 있는 점들 중 x와 거리가 R 이내인 것을 Q에 추가한다.
5    Q의 크기가 N을 넘으면 break; // 이 행을 제거하면 R 이내인 모든 점을 Q에 저장
6  }
7  Q에서 거리가 가장 짧은 것을 x_nearest로 취한다.
```

# 7.3 기하 정렬과 변환 추정

## 7.3.1 최소제곱법과 강인한 추청 기법

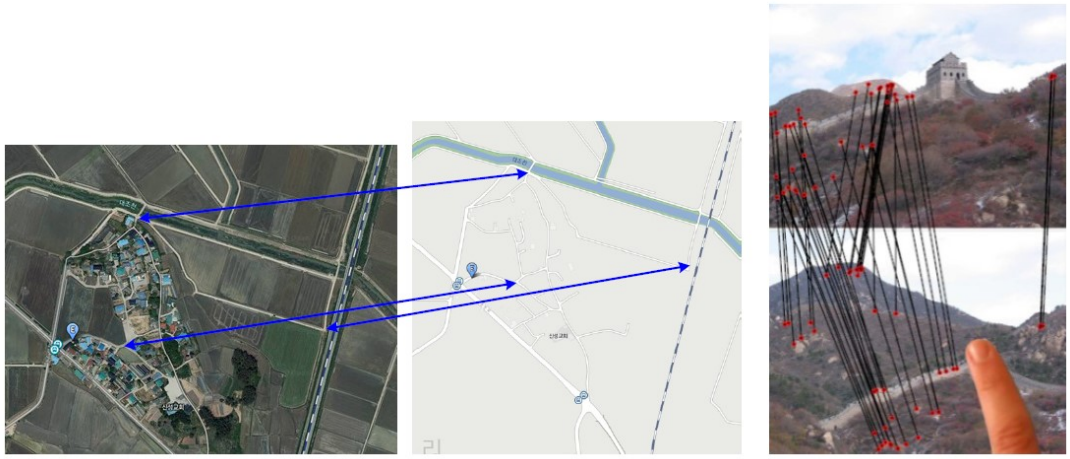
## 7.3.2 RANSAC

# 7.3 기하 정렬과 변환 추정

- 지금까지는,
  - 특징 벡터가 개별적으로 매칭을 수행 → 아웃라이어 매칭 (거짓 긍정) 발생

  → 기하 정렬을 이용하여 인라이어 집합을 찾아내고, 변환 행렬을 추정해야 함

- 여러 상황
  - 사람이 개입하여 아웃라이어 없는 경우: 항공 사진 비교, 의료 영상 정합 등
  - 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

- 오래 전부터 수학과 통계 분야에서 사용된 기법
- 예) $X=\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$를 가장 잘 대표하는 직선을 찾아라. ← 회귀 문제
  - 직관적으로 $l_1$이 더 좋음 → 수학으로 어떻게 설명하나?

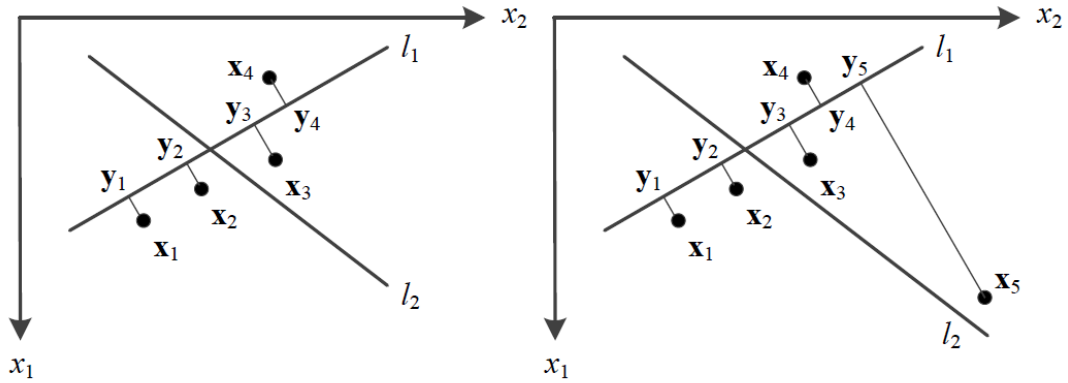



그림 7-12 최소제곱법

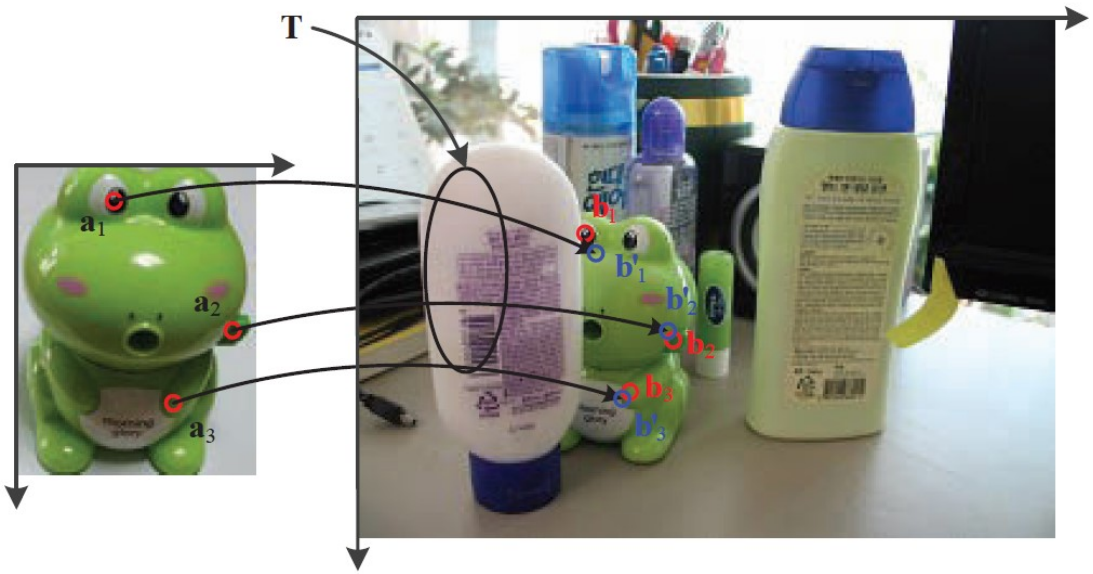
- 직선 $l$까지의 거리의 합을 오차로 공식화
  - $E(l)$을 최소화하는 $l$을 찾아라.

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

# 7.3.1 최소제곱법과 강인한 추정 기법

- 매칭 문제로 확장
  - 입력은 매칭 쌍 집합 $X=\{(\mathbf{a}_1,\mathbf{b}_1),(\mathbf{a}_2,\mathbf{b}_2),\ldots,(\mathbf{a}_n,\mathbf{b}_n)\}$
  - 모델은? → 변환 행렬

$$\mathbf{T}=\begin{pmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{pmatrix}$$




(a) 모델 영상 (b) 장면 영상

그림 7-13 최소제곱법으로 물체의 자세 T를 알아내는 사례

# 7.3.1 최소제곱법과 강인한 추정 기법

- 오차 함수 $E(\mathrm{T})$

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

$$\begin{aligned} E(\mathbf{T}) &= \sum_{i=1}^{n} \| \mathbf{b}_i - \mathbf{b}'_i \|^2 \\ &= \sum_{i=1}^{n} ((b_{i1} - (t_{11}a_{i1} + t_{21}a_{i2} + t_{31}))^2 + (b_{i2} - (t_{12}a_{i1} + t_{22}a_{i2} + t_{32}))^2) \end{aligned} \tag{7.13}$$

# 7.3.1 최소제곱법과 강인한 추정 기법

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

**T의 요소들**

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

# 7.3.1 최소제곱법과 강인한 추정 기법

- 최소제곱법은 아웃라이어 있으면 오작동
  - 예) 아웃라이어 $\mathbf{x}_5$가 포함되면, $l_2$를 선호
    → 이런 경우에는 강인한 추정 기법 필요

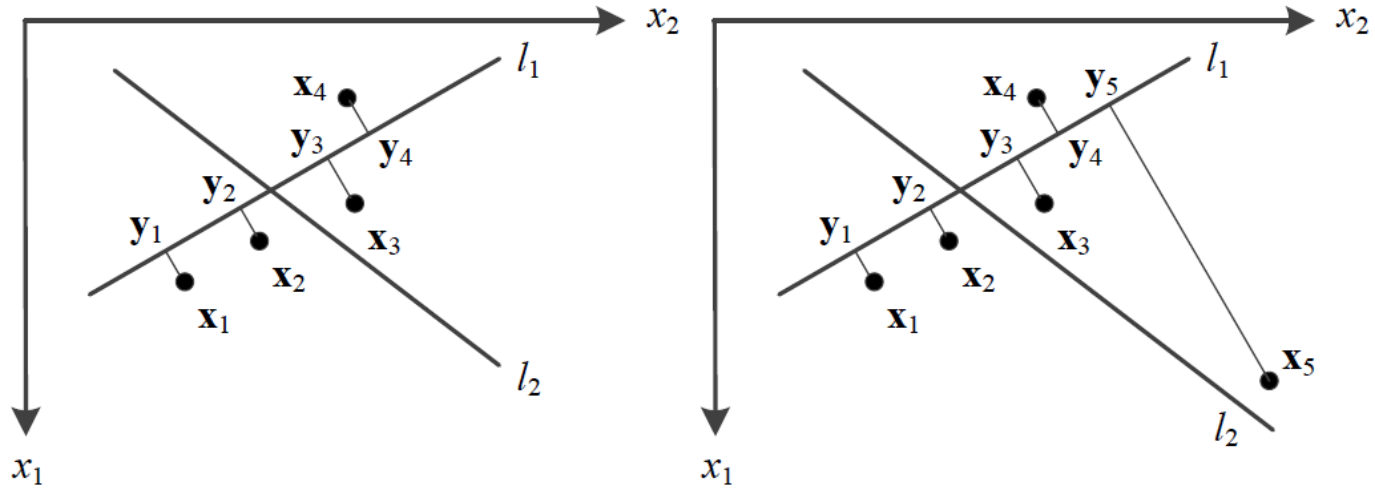



그림 7-12 최소제곱법

# 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

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

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

- 아웃라이어는 중앙값 계산하는 단계까지만 참여하는 최소제곱중앙값

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

# 7.3.2 RANSAC

- 원리
  - 직선 검출하는 3장의 그림 3-31과 같은 원리

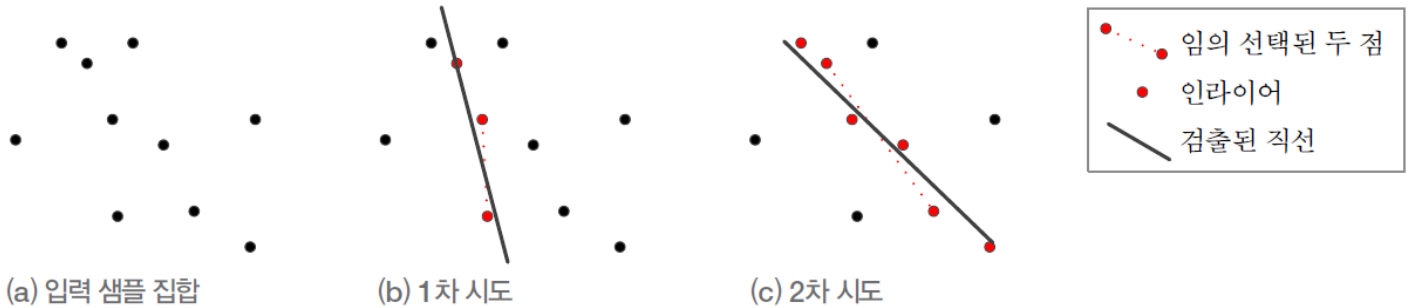



그림 3-31 RANSAC의 원리

- 여기서는,
  - 매칭 쌍 집합 $X=\{(\mathbf{a}_1,\mathbf{b}_1),(\mathbf{a}_2,\mathbf{b}_2),\ldots,(\mathbf{a}_n,\mathbf{b}_n)\}$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 $\mathbf{T}$

```
1   Q = ∅;
2   for(j=1 to k) {
3     X에서 세 개 대응점 쌍을 임의로 선택한다.
4     이들 세 쌍을 입력으로 식 (7.14)를 풀어 T_j를 추정한다.
5     이들 세 쌍으로 집합 inlier를 초기화한다.
6     for(이 세 쌍을 제외한 X의 요소 p 각각에 대해) {
7       if(p가 허용 오차 t 이내로 T_j에 적합) p를 inlier에 넣는다.
8     }
9     if(|inlier|≥d)  // 집합 inlier가 d개 이상의 샘플을 가지면
10      inlier에 있는 모든 샘플을 가지고 새로운 T_j를 계산한다.
11    if(T_j의 적합 오류<e ) T_j를 집합 Q에 넣는다.
12  }
13  Q에 있는 변환 행렬 중 가장 좋은 것을 T로 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, $X$에서 세 쌍을 선택한다.

- 반복 횟수 $k$를 결정하는 방법
  - 옳은 답을 기대할 수 없는 확률을 $P_{thres}$보다 낮게 유지하고 싶다면,

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

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

$$\text{따라서 } k > \frac{\log(p_{thres})}{\log(1 - q^3)} \tag{7.19}$$

# 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

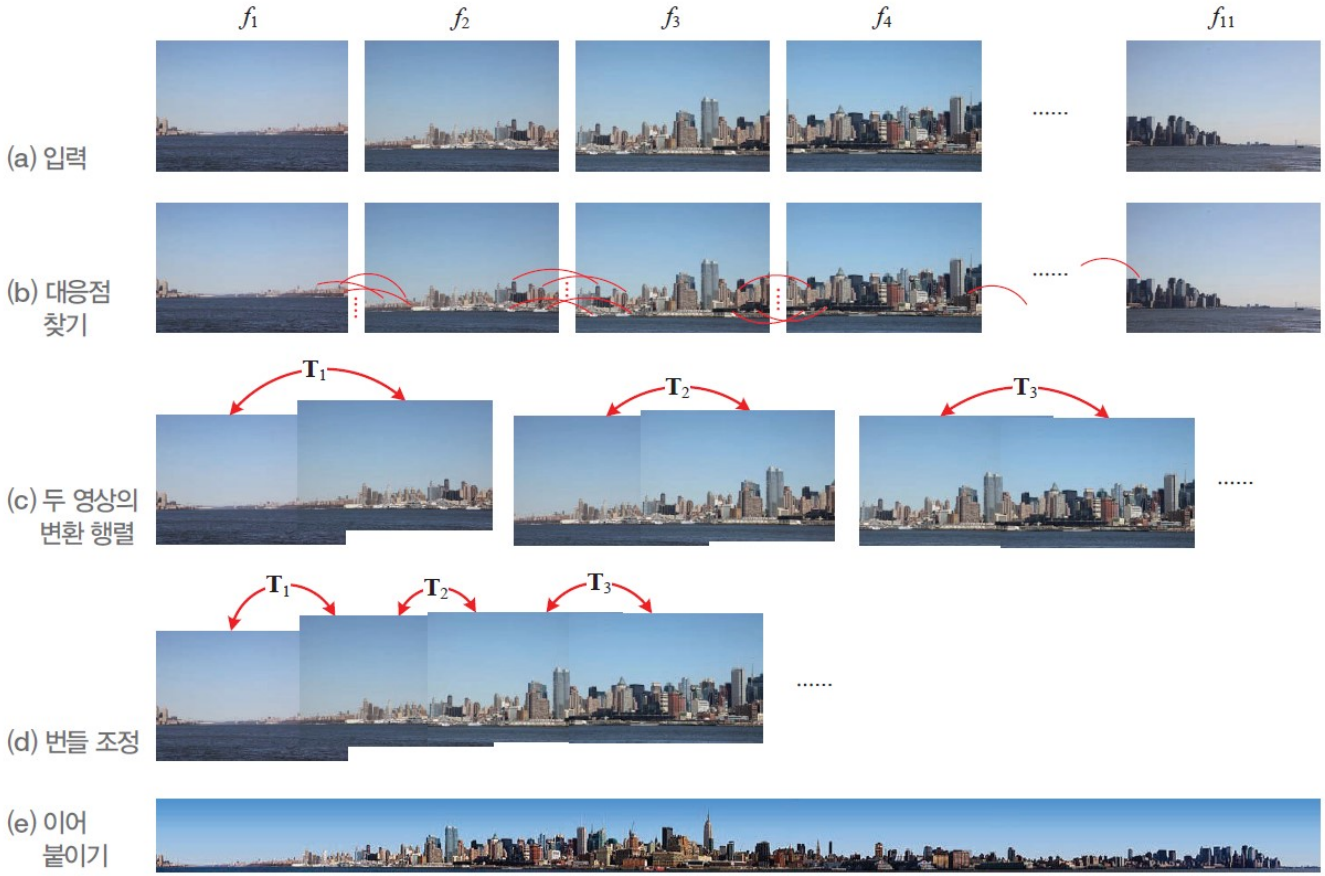



그림 7-16 파노라마 제작 과정

# 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 $p$

```
1  k개의 모든 영상에서 지역 특징을 추출한다. // 예를 들어 SIFT
2  for(i=1 to k-1) {  // i와 i+1번째 영상을 이어 붙인다.
3     kd 트리 또는 위치의존 해싱을 이용하여 f_i와 f_{i+1} 사이의 대응점을 찾는다.
4     [알고리즘 7-9(RANSAC)]를 이용하여 f_i와 f_{i+1} 사이의 변환 행렬 T_i를 추정한다.
5  }
6  번들 조정을 수행하여 T_i, i=1,2,…,k-1을 보다 정확한 값으로 조정한다.
7  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

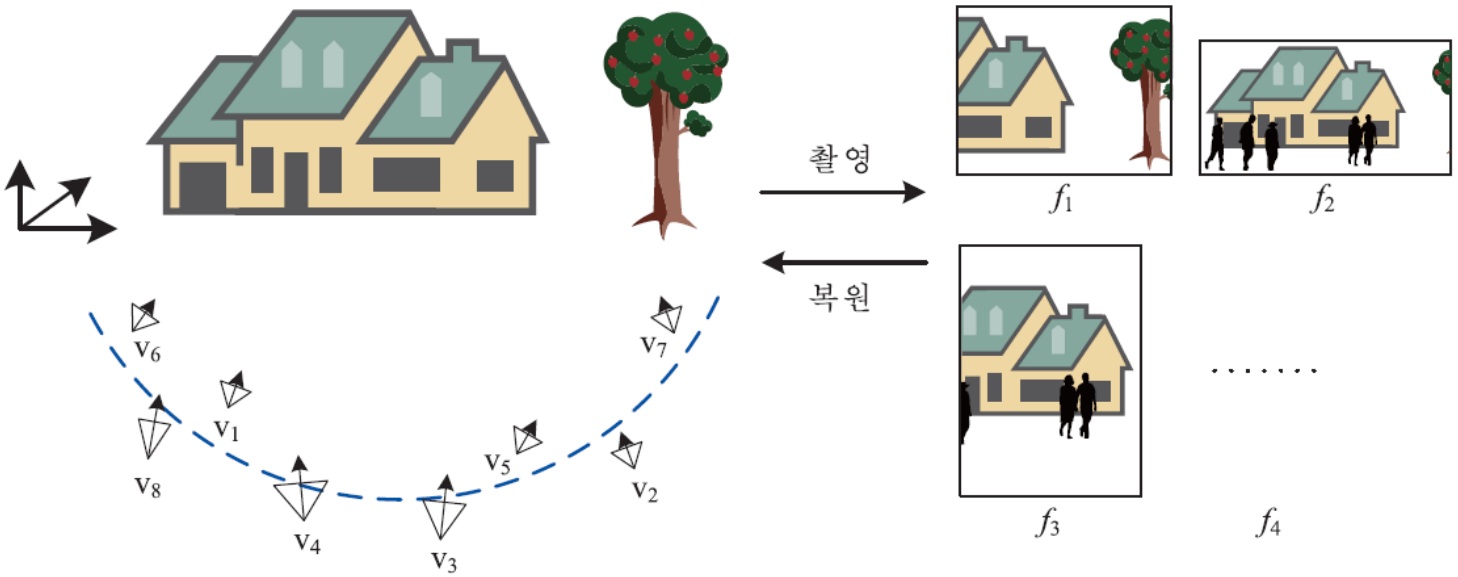



그림 7-17 같은 장소를 여러 시점에서 촬영 – 시점과 3차원 장면을 복원할 수 있을까?

# 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)

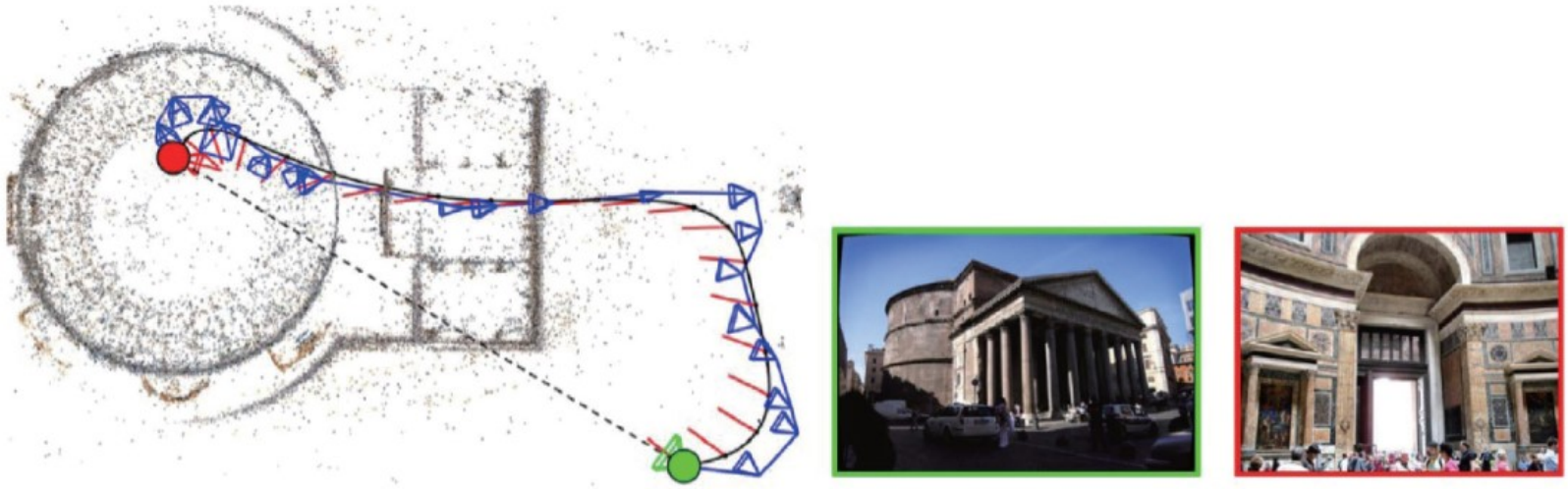


그림 7-18 사진 관광

# 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)

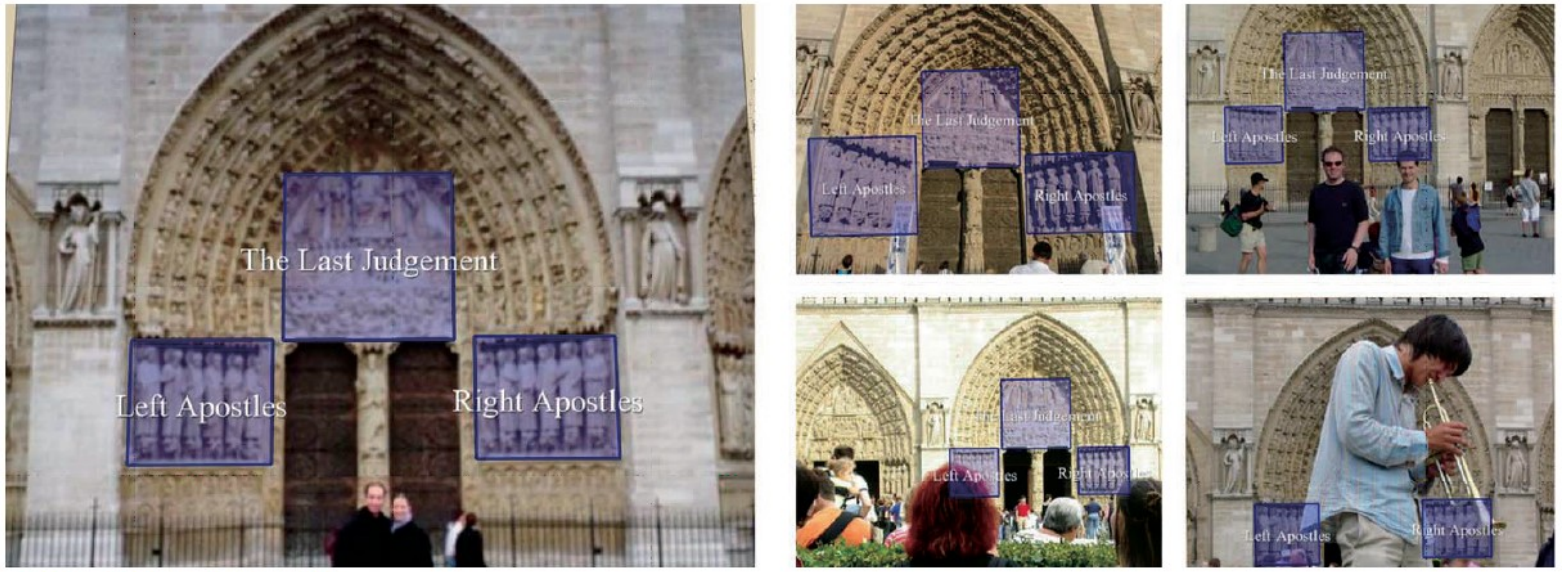



그림 7-19 자동 주석 붙이기

# 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 $t$와 $c$

**출력 :** 카메라 시점과 3차원 물체

```
1   모든 영상에서 지역 특징을 추출한다.  // 예를 들어 SIFT
2   kd 트리 또는 위치의존 해싱을 이용하여 대응점을 찾는다.
3   for(i=1 to k)
4     for(j=1 to k)
5       if(i≠j이고 f_i와 f_j 사이에 대응점이 t개 이상이면) {  // 겹침 조사
6         RANSAC을 적용하여 변환 행렬 T를 구한다.
7         T의 신뢰도가 c 이상이면 f_i와 f_j 사이에 에지를 부여한다.
8       }
9   for(그래프의 연결요소 각각에 대해)
10    번들 조정을 수행하여 카메라 시점과 3차원 점을 구한다.
```