

## 들어가기 전에

### 선수 연계 과목

고등학교 수준의 수학 실력을 갖추었다면 이 책을 보는 데 큰 어려움이 없을 것이다. 추가로 선형대수, 미적분학, 확률 통계와 자료구조, 알고리즘에 대한 기초 지식이 있으면 좀 더 빠르게 내용을 이해할 수 있다. 만약 모르는 내용이 나오더라도 필요한 주제를 선별적으로 공부하면 된다. 또한 부록 A에 있는 프로그래밍 실습을 따라 하려면 C, C++, Java 언어 중 하나로 프로그래밍이 가능해야 한다.

### 소스 코드와 강의 보조 자료

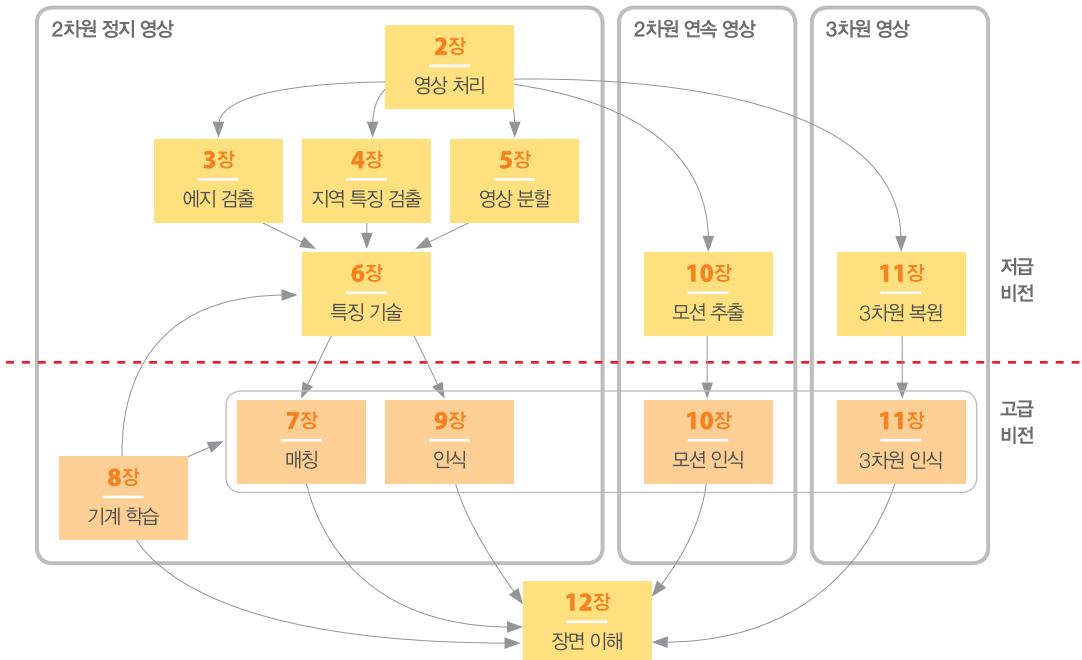
이 책의 부록A를 실습하는 데 필요한 자료 및 소스 코드는 다음 주소에서 내려받을 수 있다.

<http://www.hanbit.co.kr/exam/4121>

다음 사이트에 교수 회원으로 가입하신 교수/강사분에게 교수용 자료를 제공합니다.

<http://www.hanbit.co.kr>

### 이 책에서 다루는 내용



# Chapter 07

# 매칭

1 매칭의 기초

2 빠른 최근접 이웃 탐색

3 기하 정렬과 변환 추정

4 웹과 모바일 응용

연습문제

# Preview

매화꽃 피면

그대 오신다고 하기에

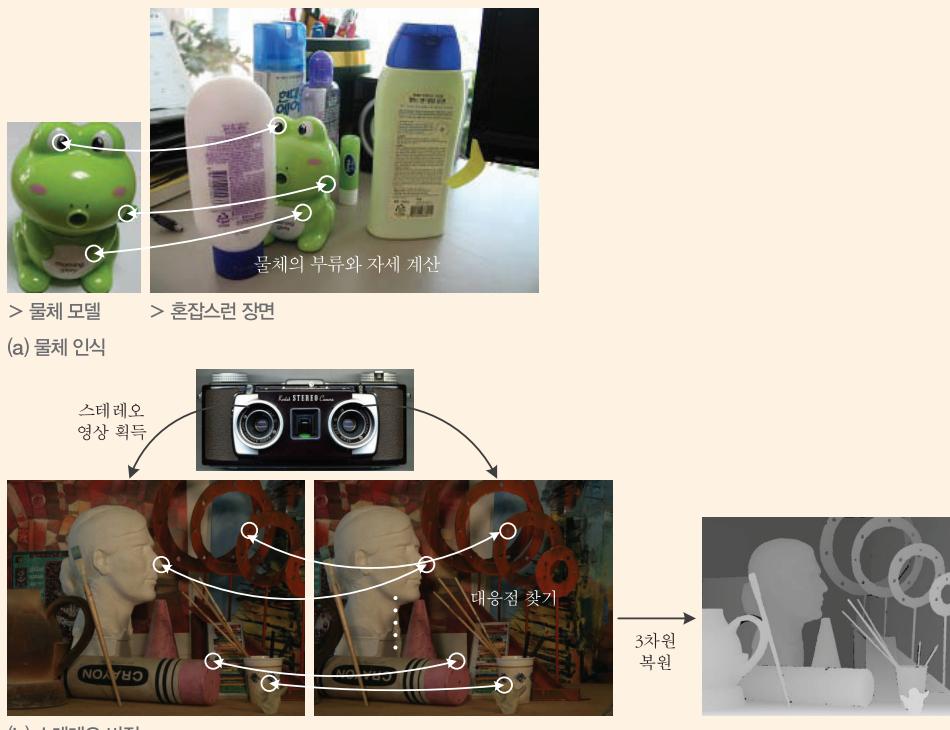
매화더러 피지 마라고 했지요

그냥, 지금처럼

피우려고만 하라구요

\_김용택 '그리움'

매칭이란 어떤 대상을 다른 것과 비교하여 그들이 같은 것인지 알아내는 과정을 일컫는다. 보통 둘 사이의 유사성 또는 거리를 측정해서 비교한다.



[그림 7-1(a)]는 물체 모델과 혼잡스런 장면을 매칭하여 물체를 인식하는 것을 보여준다. 세 점 이상의 올바른 매칭 쌍을 찾으면 물체의 자세pose(위치와 방향)를 알아낼 수 있다. [그림 7-1(b)]는 위치 차이가 약간 나는 두 대의 카메라로 찍은 영상을 이용해 물체까지 거리를 추정하는 스테레오 문제이다. 카메라의 위치 차이를 정확히 알고 있으므로, 매칭 점을 찾으면 그 점까지 거리를 삼각비로 쉽게 계산할 수 있다. 4장의 도입부에서 제시하였던 파노라마 영상의 경우에도 매칭 쌍을 찾은 후 영상의 이음선을 봉합하여 제작한 것이다. 이와 같이 서로 다른 둘 또는 그 이상의 영상을 매칭하여 대응점을 찾는 문제는 여러 가지 응용 문제를 해결하는 중요한 열쇠이다. 이때 이런 의문을 가질 수 있다. 무엇을 매칭할 것인가?

3~5장에서는 매칭에 쓸 특징을 검출하는 방법에 대해 공부하였다. 3장의 에지, 4장의 지역 특징, 5장의 영역이 매칭에 사용된다. 6장에서는 매칭에 쓸 풍부한 정보를 추출하고 그것을 특징 벡터(기술자)로 표현하는 방법을 다루었다. 이제는 특징 벡터를 비교하여 대응점을 찾아내면 된다. 이때 특징별로 매칭에 참여하는 방식에 차이가 있다. 7장에서 제시하는 매칭 방법은 4장에서 공부한 지역 특징을 대상으로 한다. 5장에서 공부한 영역 또는 전체 영상에서 추출한 특징 벡터는 그것이 속할 부류가 정해져 있는 상황이 대부분이다. 예를 들어 영역 또는 영상을 도로, 건물, 나무, 해변, 실내의 다섯 부류 중 하나로 분류하는 응용 문제이다. 이런 상황은 특징을 직접 비교하는 접근 방법 대신 8장에서 공부할 기계 학습이나 9장의 인식 알고리즘을 주로 사용한다.

매칭이라는 연산은 단순하다. 특징점 사이의 유사도를 계산한 후, 가장 큰 값을 갖는 쌍을 대응점으로 결정하면 된다. 또는 거리를 계산하고 가장 작은 값을 갖는 쌍을 찾는다. 이런 관점에서 바라보면, 매칭은 더 이상 기술적인 문제가 없는 듯 이 보인다. 하지만 현실로 들어가면 그렇지 않다. 첫째, 서로 다른데 우연히 유사도가 높은 잘못된 매칭(거짓 긍정)이 발생 할 수도 있고 반대로 서로 같은 특징점인데 유사도가 낮아 매칭이 이루어지지 않는 문제(거짓 부정)도 발생한다. 이러한 상황에서 [그림 7-1]과 같은 응용 문제를 어떻게 풀 것인가? 대부분 많은 매칭 쌍을 찾은 후 검증 과정에서 신뢰도가 높은 매칭 쌍 집합을 골라내는 접근 방법을 사용한다.

둘째는 속도이다. [그림 7-1]의 영상은 각각 수백~수천 개의 특징점을 가질 것이다. 두 영상의 특징점의 개수를  $m$ 과  $n$ 이라 하고 특징 벡터의 차원을  $d$ 라 하면, 두 영상을 매칭하는 데 걸리는 시간은  $\Theta(mnd)$ 이다. 파노라마 영상을 만들기 위해 몸을 돌려 10여 장의 사진을 찍는 수고를 감수했는데, 파노라마 제작에 수십 초가 걸리면 사용자는 짜증이 날 것이며 다음에는 다른 브랜드의 카메라를 구입할 것이다.

#### ▶ 각 절에서 다룬는 내용

**7.1절\_** 매칭에 사용하는 거리 척도와 매칭 전략, 성능을 분석하는 척도에 대해 살펴본다.

**7.2절\_** 매칭 속도를 올릴 수 있는 방법으로 kd 트리와 해싱에 대해 살펴본다.

**7.3절\_** 신뢰도가 높은 매칭 쌍을 고르는 방법을 다룬다.

**7.4절\_** 매칭을 활용한 파노라마, 사진 관광 응용 사례를 살펴본다.

# 1

## 매칭의 기초

두 특징점을 매칭하려면 유사도 또는 거리를 측정하는 척도가 필요하다. 7.1.1절에서 이 주제를 다룬다. 응용에 따라 틀린 매칭을 좀더 허용하더라도 옳은 매칭을 많이 찾아야 하는 상황도 있을 수 있고, 반대로 틀린 매칭을 최소로 허용하면서 옳은 매칭 몇 개만 찾으면 되는 상황도 있을 수 있다. 따라서 매칭 알고리즘은 다양한 상황에 적응할 수 있는 매개변수를 가져야 한다. 7.1.2절에서는 이러한 매개변수를 조절하는 방법과 매칭 성능을 측정하는 방법을 다룬다.

### 1. 거리 척도

6장에서는 관심점이나 영역으로부터 특징 벡터(기술자)  $\mathbf{x}$ 를 추출하였다.  $\mathbf{x}$ 는  $d$ 차원 공간의 한 점이다. 예를 들어, SIFT 기술자의 경우  $\mathbf{x}$ 는 128차원 공간의 점이다. 이 공간에 있는 두 점  $\mathbf{a}$ 와  $\mathbf{b}$ 를 매칭하기 위해서는 이들이 얼마나 떨어져 있는지 측정하는 거리 척도를 마련해야 한다. 여기에서는 거리 척도가 잘 작동하도록 전처리 과정으로 수행하는 화이트닝 변환에 대해서도 공부할 것이다.

**TIP** 이들 특징 벡터는 상황에 따라  $\mathbf{x}, \mathbf{y}$  또는  $\mathbf{a}, \mathbf{b}, \mathbf{c}$ 로 표기한다. 여러 개의 특징 벡터를 구별할 필요가 있을 때는  $\mathbf{x}_1, \mathbf{a}_1$ 와 같이 표기한다.

두 점을  $\mathbf{a}=(a_1, a_2, \dots, a_d)$ 와  $\mathbf{b}=(b_1, b_2, \dots, b_d)$ 로 표기할 때 이들 사이의 거리는 식 (7.1)로 측정할 수 있다. 유클리디안 거리euclidean distance라 부르는데, 가장 널리 사용하는 거리 척도이다.

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

하지만 상황이 [그림 7-2]와 같다면 이야기는 달라진다. 그림에서  $\mathbf{b}$ 와  $\mathbf{c}$  중에 어느 것이  $\mu$ 에 더 가까운가? 이들 점이 속하는 확률 분포를 고려하지 않으면 당연히  $\mathbf{b}$ 가 더 가깝다. 하지만 확률 분포에 따르면  $\mathbf{c}$ 는  $\mathbf{b}$ 에 비해 발생할 확률이 훨씬 크다. 평균 점인  $\mu$ 까지 거리를 따진다면  $\mathbf{c}$ 가  $\mathbf{b}$ 보다 가깝다고 말해야 합리적이다.

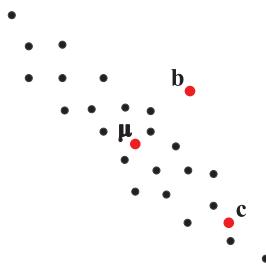


그림 7-2 확률 분포 속의 거리

인도의 통계학자인 마할라노비스는 이러한 상황을 고려한 거리를 제안하였다. 점  $\mathbf{a}$ 에서 가우시안 분포  $N(\mu, \Sigma)$ 까지를 이르는 마할라노비스 거리mahalanobis distance는 식 (7.2)와 같이 정의한다. 이때  $\mu$ 는 평균 벡터이고  $\Sigma$ 는 공분산 행렬이다. 식 (7.3)은 두 점  $\mathbf{a}$ 와  $\mathbf{b}$  사이의 마할라노비스 거리이다.

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

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

[예제 7-1]을 살펴보며 마할라노비스 거리를 직관적으로 이해해보자.

### 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  $b$ 가  $c$ 보다  $\mu$ 에 더 가깝다.

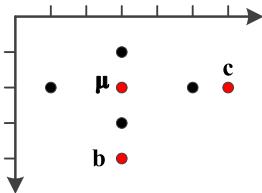


그림 7-3 마할라노비스 거리 예제

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.  $\Sigma$ 의 역행렬을 구하면,  $\Sigma^{-1} = \begin{pmatrix} 2 & 0 \\ 0 & 0.5 \end{pmatrix}$ 이다. 이들을 이용하여 두 점  $b$ 와  $c$ 에서 이 가우시안 분포까지의 거리를 계산하면 다음과 같다.  $b$ 와  $c$ 는 각각 분포까지의 거리가 2.8284와 2.1213이므로  $c$ 가  $b$ 보다 가우시안 분포에 더 가깝다.

$$\begin{aligned} b \text{와 가우시안 분포 사이의 마할라노비스 거리 } d_M(b) &= \sqrt{(4-2)(3-3)} \begin{pmatrix} 2 & 0 \\ 0 & 0.5 \end{pmatrix} \begin{pmatrix} 4-2 \\ 3-3 \end{pmatrix} = 2.8284 \\ c \text{와 가우시안 분포 사이의 마할라노비스 거리 } d_M(c) &= \sqrt{(2-2)(6-3)} \begin{pmatrix} 2 & 0 \\ 0 & 0.5 \end{pmatrix} \begin{pmatrix} 2-2 \\ 6-3 \end{pmatrix} = 2.1213 \end{aligned}$$

이제 두 점  $b$ 와  $c$  사이의 유클리디안 거리와 마할라노비스 거리를 계산해 보자.

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

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

공분산 행렬이  $\Sigma = \mathbf{I}$ 인 특수한 경우는 유클리디안 거리와 마할라노비스 거리가 같다.  $\Sigma^{-1} = \mathbf{I}$ 이므로 식 (7.3)에서  $\Sigma^{-1}$ 을 생략해도 되기 때문이다. 하지만 실제 세계에서는 공분산 행렬이 단위 행렬  $\mathbf{I}$ 가 되는 경우는 많지 않은데, 전처리 단계에서 화이트닝 whitening transform을 적용하면 공분산 행렬을 단위행렬로 만들 수 있다. 이런 변환을 추가한다면 마할라노비스 거리 대신 유클리디안 거리를 사용하면 된다. 화이트닝 변환은 식 (7.4)로 수행할 수 있다.  $\Phi$ 는  $d \times d$  행렬로, 공분산 행렬에서 구한 고유 벡터를 담고 있다.  $\Phi$ 의  $i$ 번째 열의 값은  $i$ 번째 고유 벡터이다.  $\Lambda$ 는  $d \times d$  크기의 대각선 행렬로서  $i$ 번째 대각선 요소는  $i$ 번째 고유값이다. [예제 7-2]는 화이트닝 변환을 예로 보여준다.

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

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

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

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

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

네 개의 샘플을 식 (7.4)로 변환하면 다음과 같다. 새로 얻은 네 점을 가지고 공분산 행렬을 구해 보면 단위 행렬  $I_2$ 가 되어, 화이트닝 변환이 적용되었음을 확인할 수 있다.

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

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

## 2. 매칭 전략과 성능 분석

매칭을 사용하는 응용 상황은 다양하다. [그림 7-1(b)]의 스테레오 영상이나 여러 장의 영상을 이어서 붙이는 파노라마 영상을 제작하는 경우에는 두 장의 영상이 등등한 입장에서 매칭에 참여 한다. 한 영상에서 추출한 특징의 수는 장면의 복잡도에 따라 다르지만 적계는 수백에서 많게는 수 천에 이른다. 한편, 물체 인식이나 증강현실과 같은 응용에서는 [그림 7-1(a)]에서 볼 수 있듯이 모델 영상과 장면 영상이 구분된다. 모델은 대상 물체만 가지므로 수십에서 수백 개 정도의 비교적 적은 수의 특징 벡터를 갖지만, 장면 영상은 여러 가지 물체와 배경이 혼재되어 있으므로 수천 개를 가질 가능성이 높다. 한 장의 장면 영상에 의자, 책상, 화분, TV, 시계, 계단 등이 있는지 인식하고자 할 때는 여러 가지 물체의 모델 각각에 매칭을 수행해야 한다. 이와 같이 일대일 매칭이 아닌, 일대다 매칭이 벌어지는 경우에는 일대일 매칭을 여러 번 수행하면 된다. 지금부터는 두 장의 영상을 매칭하는 작업을 살펴본다.

두 영상을 매칭하는 경우, 서로 구분하기 위해 첫 번째 영상의 특징 벡터를  $\mathbf{a}_i$  ( $1 \leq i \leq m$ )라 하고 두 번째 영상은  $\mathbf{b}_j$  ( $1 \leq j \leq n$ )라 표기하자. 가장 단순한 매칭 전략은 고정 임계값을 사용하는 것이다. 서로 다른 두 영상에서 추출한 두 점  $\mathbf{a}_i$ 와  $\mathbf{b}_j$ 는 식 (7.5)를 만족하면 성공적으로 매칭이 되었다고 판단하고 매칭 쌍으로 저장한다. 이 식에서 거리 척도  $d(\cdot)$ 는 7.1절에서 제시한 유클리디안 거

리  $d_E(\cdot)$  또는 마할라노비스 거리  $d_M(\cdot)$ 을 사용하면 된다. 이러한 매칭 전략을 쓰면 하나의 점에 여러 개의 점이 대응될 수 있다.

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

이 전략에서 가장 신경쓸 문제는 임계값  $T$ 를 정하는 것이다.  $T$ 를 작게 하면 아주 가까운 쌍만 매칭이 성공하므로, 진짜 매칭 쌍인데 실패하는 경우가 발생할 수 있다. 즉, 거짓 부정(FN=False Negative)이 많이 발생한다. 반대로  $T$ 를 크게 하면 거리가 먼 쌍도 매칭에 성공할 수 있으므로, 가짜 쌍인데 맺어지는 경우가 발생할 수 있다. 즉, 거짓 긍정(FP=False Positive)이 많이 발생한다.

$T$ 를 조금씩 증가시키며 수집한 점 ( $FPR, TPR$ )을 이은 곡선을 ROC(Receiver Operating Characteristic) 곡선이라 부른다. [그림 7-4]는 ROC 곡선의 예이다.  $T$ 를 아주 낮게 하면  $FP$ 가 줄어들어 거짓 긍정률은 0에 가깝게 된다. 또한  $FN$ 이 커지므로 참 긍정률도 0에 가까워진다.  $T$ 를 조금씩 증가시키면 거짓 긍정률이 증가하는데, 참 긍정률도 함께 증가한다.

**TIP** 식(7.6)은 참 긍정률과 거짓 긍정률이다. 1.3.4절에서 이미 제시하였는데, 편의상 여기에 다시 제시한다.

$$\begin{aligned} \text{참 긍정률 } TPR &= \frac{TP}{(TP + FN)} \\ \text{거짓 긍정률 } FPR &= \frac{FP}{(FP + TN)} \end{aligned} \quad (7.6)$$

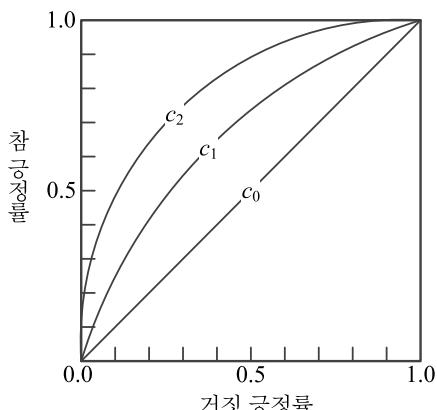


그림 7-4 ROC 성능 분석

[그림 7-4]의 ROC 곡선은 매개변수에 따라 거짓 긍정률과 참 긍정률이 어떻게 변하는지 한눈에 보여주므로 성능 분석에 자주 활용된다.<sup>1</sup> 두 종류의 특징이 있다고 가정해 보자. 이들이 각각  $c_1$ 과  $c_2$ 라는 그래프를 만들었다면, 어떤 특징이 더 훌륭할까? 당연히  $c_2$ 가 더 낫다. 같은 거짓 긍정률일 때  $c_2$ 가  $c_1$ 보다 참 긍정률이 높기 때문이다. 곡선이 왼쪽 위 구석에 가까울수록 더 좋다. 왼쪽 위 구석을 지나는 곡선은 이상적인 성능을 뜻한다. 대각선에 해당하는 곡선  $c_0$ 는 아무 특징 정보도 사용하지 않고 무턱대고 판단하는 임의 추정<sup>random guess</sup>에 해당한다.

어떤 상황에서는 곡선 자체 대신 수치 하나로 성능을 표현해야 한다. 이런 때는 곡선 아래에 있는 영역의 면적을 성능 지표로 사용하면 된다. 이 면적을 AUC(Area Under Curve)라 부르는데, AUC가 클수록 좋다. ROC에 대해 보다 상세하게 알고 싶은 독자는 [Fawcett2006]을 참고하기 바란다.

지금까지 고정 임계값을 사용하여 매칭하는 전략을 공부하였다. 또 다른 전략은 최근접 이웃 nearest neighbor을 찾는 것이다. 첫 번째 영상의 특징 벡터  $\mathbf{a}_i$ 의 대응점을 찾는다고 하자. 두 번째 영상에 있는 특징 벡터 중에  $\mathbf{a}_i$ 와 가장 가까운 것, 즉 최근접에 해당하는  $\mathbf{b}_j$ 를 찾는다. 둘 사이의 거리가  $T$  이내이면 둘은 대응 쌍이 된다.

세 번째 전략으로 최근접 거리 비율이 있다. 이 전략은 가장 가까운 점  $\mathbf{b}_j$ 와 두 번째 가까운 점  $\mathbf{b}_k$ 를 구한다. 이때 두 점이 식 (7.7)을 만족하면 점  $\mathbf{a}_i$ 와  $\mathbf{b}_j$ 가 대응 쌍이 된다. 여러 연구자들의 실험 결과에 따르면 최근접 거리 비율 전략이 가장 높은 성능을 보인다[Mikolajczyk2005a]. SIFT도 이 전략을 사용한다[Lowe2004].

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

---

<sup>1</sup> 여기서는 어떤 특징의 매칭 성능 분석에 ROC를 사용하는데, 이진 분류기의 성능을 분석하는 데에도 많이 활용된다.

## 2 빠른 최근접 이웃 탐색

[알고리즘 7-1]은 첫 번째 영상의 특징 벡터 각각에 대해 두 번째 영상에서 최근접 이웃을 구한 후, 이들을 매칭 쌍으로 취하여 *mlist*에 저장한다. 7.1.2절에서 언급한 최근접 이웃 전략을 충실히 게 수행해 주는 알고리즘이다. 시간이 별로 중요하지 않은 상황이라면 그대로 적용하면 되지만, 실시간 처리가 필요한 상황이라면 문제가 생긴다. 이 알고리즘의 매칭 시도 횟수는  $mn$ 번이다. 만일 각각의 영상에서 1,000개의 점이 추출되었다면, [알고리즘 7-1]은 매칭을 백만 번 시도한다. 따라서 이 순진한 알고리즘은  $m$ 과  $n$ 이 크고 실시간 처리가 필요한 응용에는 적절하지 않다.

### 알고리즘 7-1 순진한 매칭 알고리즘

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

출력 : 매칭 쌍 리스트 *mlist*

```
1   mlist= $\emptyset$ ;  
2   for(i=1 to m) {  
3       shortest= $\infty$ ;  
4       for(j=1 to n) {  
5           dist=d( $\mathbf{a}_i$ ,  $\mathbf{b}_j$ );  
6           if(dist<shortest) {match=j; shortest = dist}  
7       }  
8   }
```

```

8   if(shortest < T) mlist = mlist ∪ (ai, bmatch);
9 }

```

이 절의 주제는 최근접 이웃의 빠른 탐색이다. [알고리즘 7-1]의 3~7행이 수행하는 최근접 이웃 탐색을 어떻게 빨리 할지가 주된 관심사로, 적절한 자료구조를 설계하여 특징 벡터 집합을 미리 인덱싱 하면 훨씬 빠른 시간 안에 최근접 점을 찾을 수 있다. 이진검색 트리binary search tree는 이를 때 쓸 수 있는 자료구조 중 하나이다. 7.2.1절에서는 이진검색 트리를 개조한 kd 트리를 공부한다. 자료구조 과목에서 배운 또 다른 빠른 검색 방법은 해싱이다. 7.2.2절에서는 일반 해싱 방법을 개조한 위치의존 해싱을 다룬다. 이 두 기법은 [알고리즘 7-1]에 비해 수십~수백 배 빠르게 최근접 또는 최근접에 아주 가까운 근사 최근접 이웃을 찾는다. 이들은 다차원 벡터를 빠르게 찾는 일반적인 기법이므로 컴퓨터 비전뿐 아니라 데이터마이닝, 빅데이터, 정보 검색, 생물 정보학, 웹 마이닝, 계산 언어학 등 아주 다양한 분야에 활용된다.<sup>2</sup>

이제부터 공부할 두 가지 기법은 [알고리즘 7-2]의 틀에서 작동한다. [알고리즘 7-2]는 7.1.2 절의 세 가지 매칭 전략 중 두 번째의 최근접 이웃을 채택하고 있는데, 세 번째의 최근접 거리 비율 전략을 쓸 수도 있다. 이 경우 3행에서 최근접을 찾은 후, 그것을 배제하고 다시 최근접을 찾아 두 번째 최근접을 구한다.

### 알고리즘 7-2 빠른 매칭 알고리즘

**입력 :** 첫 번째 영상의 특징 벡터  $a_i$ ,  $1 \leq i \leq m$ , 두 번째 영상의 특징 벡터  $b_j$ ,  $1 \leq j \leq n$ , 거리 임계값  $T$

**출력 :** 매칭상 리스트  $mlist$

```

1   mlist = ∅;
2   for(i=1 to m) {
3       kd트리나 해싱 알고리즘으로 ai의 최근접 또는 근사 최근접 이웃 bmatch를 탐색한다.
4       if(d(ai, bmatch) < T) mlist = mlist ∪ (ai, bmatch);
5   }

```

<sup>2</sup> 다차원 공간 탐색을 가장 광범위하게 다룬 문헌을 원하면 [Samet2006]을 참고하라.

## 1. kd 트리

[그림 7-5]는 이진검색 트리의 사례이다. 이 트리에서는 어떤 노드를 기준으로 왼쪽에 있는 모든 노드는 기준 노드의 값보다 작고 오른쪽의 노드는 크다. 예를 들어 12를 갖는 루트 노드의 왼쪽 트리는 7, 3, 10을 가지고 오른쪽 트리는 17, 21을 가지므로 이 성질을 만족한다. 루트 노드의 자식 노드를 기준 노드로 할 때도 이 성질을 만족한다. 이제 트리가 주어졌으니, 검색하는 문제를 생각해 보자. 이 트리에서 10을 검색 키 search key라 하자. 루트와 비교해 보니 작으므로 왼쪽 자식 노드로 이동한다. 그곳의 7보다 크므로 오른쪽 자식으로 이동한다. 그곳에서 10을 찾았다. 이와 같은 재귀 과정으로 검색 키를 빠른 속도로 찾아낼 수 있다. [알고리즘 7-3]은 이진검색 트리  $T$ 에서 키  $v$ 를 검색하는 과정을 보여준다.

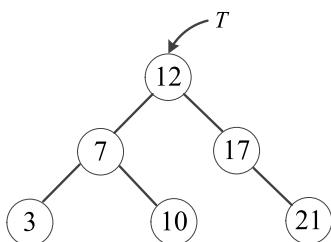


그림 7-5 이진검색 트리

### 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 }
```

[알고리즘 7-3]은  $T$ 가 평형을 이룬다면 즉, 왼쪽과 오른쪽 트리의 크기가 대략 같다면  $O(\log(n))$  시간에 검색을 마친다.  $n$ 은 노드의 개수이다. 트리의 깊이는 대략  $\log(n)$ 이고 한 번 비교를 할 때마다 한 단계씩 트리 아래쪽으로 전진하기 때문이다.

## kd 트리 만들기

이 절의 주제인 빠른 최근접 탐색을 위해 이진검색 트리의 아이디어를 활용할 수 있다. 하지만 몇 가지 측면에서 이진검색 트리와 다른 점이 있다. 첫째, 검색 키  $v$ 가 하나의 값이 아니라 여러 개의 실수로 구성된 벡터이다. 둘째,  $v$ 와 같은 값을 갖는 노드를 찾는 것이 아니라  $v$ 와 가장 가까운 최근접 이웃 노드를 찾아야 한다. 예를 들어, [그림 7-5]의 트리에서 6을 찾는다면 답은 7이다. 하지만 탐색 과정에서 7에서 바로 멈출 수는 없다. 그 밑에 7보다 더 가까운 것이 있을지도 모르기 때문이다.

kd 트리는 이러한 문제들을 해결한 알고리즘으로, 벤트리가 1975년에 고안하였다[Bently75].  $n$ 개의 특징 벡터  $X=\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ 을 가지고 kd 트리를 만든다고 하자. 하나의 특징 벡터는  $\mathbf{x}_i=(x_1, x_2, \dots, x_d)$ 과 같이 표현되는  $d$ 차원 벡터이다. kd 트리는 하나의 벡터가 하나의 노드가 된다. 루트 노드는  $X$ 를 두 개의 부분 집합  $X_{left}$ 와  $X_{right}$ 로 분할한다. 이때 분할의 기준을 선택하는 일이 중요하다.

두 가지 기준을 선택해야 하는데, 첫 번째로  $d$ 개의 차원(축) 중에 어느 것을 쓸 것인지 선택해야 한다. 이때 분할 효과를 극대화하기 위해 각 차원의 분산을 계산한 후 최대 분산을 갖는 축  $k$ 를 선택한다. 축을 선택한 이후에는  $n$ 개의 샘플 중 어느 것을 기준으로  $X$ 를 분할할 것인지 결정한다. 이때  $X_{left}$ 와  $X_{right}$ 의 크기를 같게 하여 균형 잡힌 트리를 만들기 위해  $X$ 를 차원  $k$ 로 정렬하고 그 결과의 중앙값 median을 분할 기준으로 삼는다. 이렇게  $X$ 를  $X_{left}$ 와  $X_{right}$ 로 분할한 후 각각에 같은 과정을 재귀적으로 반복하면 kd 트리가 완성된다.

지금까지 설명한 과정을 정리하면 [알고리즘 7-4]와 같다. 뒤이어 나오는 [예제 7-3]은  $d=2$ 이고  $n=10$ 인 간단한 상황으로 알고리즘의 동작을 보여준다.

## 알고리즘 7-4 kd 트리 만들기

입력 : 특징 벡터 집합  $X = \{x_i, i=1, 2, \dots, n\}$

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=xm; // xm은 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 Xsorted를 만든다.
13      Xsorted에서 중앙값을 xm, 왼쪽 부분집합을 Xleft, 오른쪽 부분집합을 Xright라 하자.
14      트리 노드 node를 생성한다.
15      node.dim=k; // 어느 차원으로 분할하는지
16      node.vector=xm; // 어떤 특징 벡터로 분할하는지
17      node.leftchild=make_kdtree(Xleft);
18      node.rightchild=make_kdtree(Xright);
19      return node;
20    }
21  }
```

### 예제 7-3 kd 트리 만들기

설명을 쉽게 하기 위해  $d=2$ 로 한정하고,  $X=\{x_1=(3,1), x_2=(2,3), x_3=(6,2), x_4=(4,4), x_5=(3,6), x_6=(8,5), x_7=(7,6.5), x_8=(5,8), x_9=(6,10), x_{10}=(6,11)\}$ 이라 하자. [그림 7-6(a)]는 주어진 특징 벡터의 집합을 보여준다.

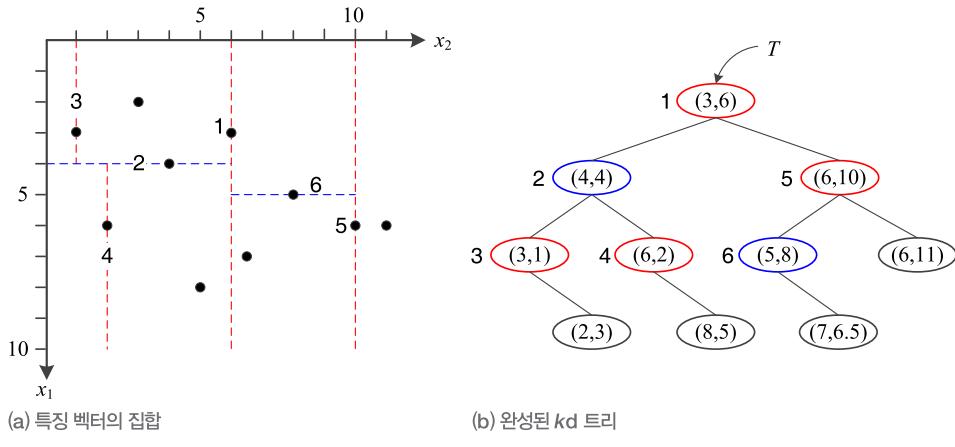


그림 7-6 kd 트리

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  $X_{sorted}=\{x_1, x_3, x_2, x_4, x_6, x_5, x_7, x_8, x_9, x_{10}\}$ 이 된다. 이 리스트의 중앙값  $x_5$ 를 기준으로 좌우를 분할하면,  $X_{left}=\{x_1, x_3, x_2, x_4, x_6\}$ ,  $X_{right}=\{x_7, x_8, x_9, x_{10}\}$ 이 된다. 이제 14~16행에서 노드를 하나 할당받아 값을 채운다. 이렇게 만 들어진 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.

### kd 트리를 이용한 최근접 이웃 탐색

지금까지 kd 트리를 만드는 알고리즘을 살펴보았다. 이제 이러한 트리가 주어질 때, 새로운 특징 벡터  $\mathbf{x}$ 를 입력하면 그것의 최근접 이웃을 어떻게 찾을지 생각해 보자. [그림 7-7]이 보여주는 바와 같이, 녹색으로 표시한  $\mathbf{x}=(7,5.5)$ 의 최근접 이웃을 찾는다고 하자.  $T$ 의 루트 노드는 두 번째 차원( $x_2$ 축)으로 분할하며, 분할 기준값은 6이다. 5.5가 6보다 작으므로 왼쪽 노드로 이동한다. 이 노드는 첫 번째 차원( $x_1$ 축)으로 분할하는데, 기준값은 4이다. 7이 4보다 크므로 오른쪽으로 이동한다. 이런 과정을 단말 노드에 도착할 때까지 반복하면, [그림 7-7]에서 회색 표시된 칸에 도착한

다. 이 칸에는  $\mathbf{x}_6=(8,5)$ 가 자리잡고 있다. 이 특징 벡터는  $\mathbf{x}$ 의 최근접일 가능성이 크다. 하지만 가능성이 큰 것이지 반드시 최근접이라는 보장은 없다. 분할 평면의 건너편에 더 가까운 점이 있을 수 있기 때문이다. [그림 7-7]은 1번 분할 평면의 건너편에 더 가까운 점  $\mathbf{x}_7$ 이 놓여있는 상황을 보여준다.

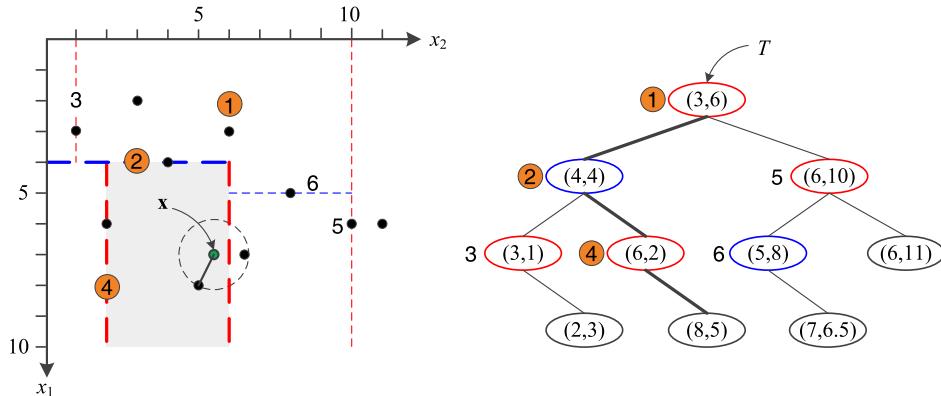


그림 7-7 kd 트리에서 검색하는 예

[알고리즘 7-5]는 이런 상황을 고려한 최근접 탐색 알고리즘으로, 거쳐온 노드들을 추가로 탐색한다. [그림 7-7]에서 주황색 원기호는 거쳐온 노드를 나타낸다. 이 알고리즘은 거쳐온 노드들을 스택에 저장한 후(7~10행), 백트래킹한다(13~21행). 이때 현재까지 찾은 가장 가까운 점을 저장해 놓고, 그것보다 먼 가지는 더 이상 탐색하지 않는 한정분기<sup>branch-and-bound</sup> 기법을 적용한다.

[그림 7-7]을 가지고 이 과정을 설명해 보자. 단말 노드에 도착하면,  $\mathbf{x}$ 까지의 거리를 계산하여 *current\_best*를 초기화한다. 그림에서 점선 원의 반지름이 이 값이 된다. 이제 백트래킹을 시작한다. 스택에서 노드 하나를 꺼내면 4번 노드가 나온다.  $\mathbf{x}$ 에서 4번 노드의 분할 평면까지 거리가 *current\_best*보다 크다. 따라서 4번 노드를 기준으로 왼쪽에 위치한 노드는 검사할 필요가 없다. 스택에서 하나를 또 꺼내면 2번 노드가 나오는데, 역시 미찬가지이다. 이제 스택에서 1번 노드가 나온다.  $\mathbf{x}$ 에서 1번 노드의 분할 평면까지 거리는 *current\_best*보다 작다. 따라서 건너편에 더 가까운 점이 존재할 가능성이 있으므로, 1번 노드의 오른쪽 건너편에 위치한 노드들을 추가로 검사해야 한다. 19행은 추가로 검사해야 하는 노드를 대상으로 kd 트리 탐색을 재귀적으로 반복한다.

### 알고리즘 7-5 kd 트리에서 최근접 찾기

입력 : kd 트리  $T$ , 탐색할 특징 벡터  $\mathbf{x}$

출력 : 최근접 이웃  $\text{nearest}$

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

지금까지 kd 트리에서 최근접 이웃을 찾는 알고리즘을 살펴보았다. 이 검색 알고리즘의 효율을 생각해 보자. 만일 백트래킹이 없다면  $\log(n)$ 보다 적은 시간에 검색이 완료된다. 노드의 개수를 의미하는  $n$ 의 값이 1,000,000이라 하더라도,  $\log(n)=20$ 이므로 매우 빠른 알고리즘이다. 하지만 kd 트리에서는 단말 노드에 도달한 이후에 백트래킹으로 추가적인 탐색을 한다. 여러 실험 결과에 따르면 10차원 이상이 되면 모든 특징 벡터와 비교를 수행하는 [알고리즘 7-1]과 비슷해져서 낮은 효율을 보인다.

## kd 트리를 이용한 근사 최근접 이웃 탐색

이러한 비효율을 벗어나는 현실적인 길은 최근접 이웃 대신 '근사 최근접 이웃 approximate nearest neighbor'을 찾는 것이다. 이러한 근사 최근접 이웃은 최근접 이웃과 같거나 아주 가까운 이웃이 되어 많은 응용에서 큰 성능 저하 없이 사용할 수 있다. [알고리즘 7-6]은 근사 최근접 이웃을 찾도록 개조한 버전이다[Beis97]. 이 알고리즘은 스택 대신 우선순위 큐priority queue인 힙heap을 사용한다. 이 힙은  $\mathbf{x}$ 로부터의 거리를 우선순위로 사용하는데, 백트래킹할 때 거리가 가까운 노드부터 시도할 수 있게 한다. [그림 7-7]과 같은 상황을 다시 예로 들어보자. 스택을 사용하던 이전 버전은  $4 \rightarrow 2 \rightarrow 1$  순으로 처리하였는데, [알고리즘 7-6]은  $1 \rightarrow 4 \rightarrow 2$  순으로 조사한다. 거리가 가깝다는 말은 최근접 이웃이 놓여있을 가능성성이 크다는 뜻이므로 그곳을 먼저 찾아보는 전략은 매우 효과적이다. 이러한 이유 때문에 최적 칸 우선 탐색best-bin-first search이라고 부른다. 또한 힙에 있는 모든 노드를 조사하는 대신, 매개변수  $try\_allowed$ 를 사용하여 조사 횟수를 제한한다.  $try\_allowed$  번만 조사하여 그때까지 찾은 것을 답으로 취한다.

### 알고리즘 7-6 kd 트리에서 근사 최근접 이웃 찾기

입력 : kd 트리  $T$ , 탐색할 특징 벡터  $\mathbf{x}$ , 허용된 최대 조사 횟수  $try\_allowed$

출력 : 근사 최근접 이웃  $nearest$

```
1 heap s = Ø; // 백트래킹을 위해, 지나온 노드를 저장할 힙을 생성한다.
2 current_best = ∞;
3 nearest = Nil;
4 try = 0; // 비교 횟수를 센다.
5 search_kdtree( $T, \mathbf{x}$ )
6 function search_kdtree( $T, \mathbf{x}$ ) {
7      $t = T$ ;
8     while(not is_leaf( $t$ )) { // 단말 노드를 찾는다.
9         if( $\mathbf{x}[t.dim] < t.vector[t.dim]$ ) {s.push( $t, "right"$ );  $t = t.leftchild$ ;}
10        else {s.push( $t, "left"$ );  $t = t.rightchild$ ;}
11    }
12    d = dist( $\mathbf{x}, t.vector$ );
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(백트랙킹)
17     (t,other_side)=s.pop();
18     d=dist(x,t,vector);
19     if(d<current_best) {current_best=d;nearest=t,;}
20     if(x에서 t,의 분할 평면까지 거리<current_best) { // 건너편에 더 가까운 것 있을 수 있음
21       t,의 other_side 노드를 t_other라 하자.
22       search_kdtree(t_other,x);
23     }
24   }
25 }

```

---

[알고리즘 7–6]은 매칭을 빠르게 처리해야 하는 응용에 적합하다.<sup>3</sup> [알고리즘 7–6]의 여러 가지 변형이 개발되어 있다. Muja는 대표적인 알고리즘을 대상으로 벤치마킹한 결과를 발표하였다 [Muja2009].<sup>4</sup> 또한 주어진 데이터에 대해 가장 빠른 알고리즘을 자동으로 선정하고 매개변수를 자동으로 설정하는 방법도 제시하였다. 반드시 최근접 이웃이 필요한 응용에서는 [알고리즘 7–5]를 사용해야 하며 속도가 중요하다면 GPU를 이용하여 병렬 처리를 해야 한다.

## 2. 해싱

해싱[hashing]은 실제 필요한 용량보다 많은 메모리를 사용하는 대신 계산 시간이 짧게 걸리는 검색 기법이다. 적절한 해시 함수와 충돌 해결 방안을 사용한다면,  $n$ 개의 요소가 담겨 있는 테이블에서  $n$ 과 무관하게  $\Theta(1)$ 이라는 상수 시간에 데이터를 삽입, 삭제, 검색할 수 있다. 해시 함수hash function는 검색 키의 값을 해시 테이블의 주소값으로 매핑해 준다.

---

**3** 근사 최근접 이웃을 사용한 유명한 예는 SIFT이다[Lowe2004]. 이 논문의 실험에 따르면, [알고리즘 7–6]에서  $try\_allowed=200$ 으로 설정하였을 때,  $n=100,000$ ,  $d=128$ 인 상황에서 95%는 정확한 최근접 이웃을 찾았고 5%만 근사 최근접 이웃을 찾았다. 대신 속도는 100배 이상 빨랐다.

**4** FLANN(Fast Library for Approximate Nearest Neighbors 홈페이지에서 공개 소프트웨어를 다운받을 수 있다.  
<http://www.cs.ubc.ca/research/flann/>

## 해싱의 기본 원리

[그림 7-8]은 해시 기법을 설명한다. 이 예는 검색 키  $x$ 가 정수이고 해시 함수는  $h(x)=x \bmod 13$ 이라 가정한다. 어떤 검색 키  $x$ 가 들어오면 그것을 13으로 나눈 나머지를 해시 테이블의 주소로 취한다. 예를 들어  $x=23$ 이라면  $h(23)=23 \bmod 13=10$ 므로 23은 10번지 통에 삽입된다. 검색도 같은 요령으로 한다.  $x=19$ 를 검색하고 싶다면  $19 \bmod 13=6$ 으로 주소 6을 조사한다. 찾으려는 19가 있으므로 그것을 반환하면 된다. 16을 검색하는 경우에는  $16 \bmod 13=3$ 인데 주소 3이 비어 있으므로 없다는 신호를 반환하면 된다. 이때 해시 함수를 한 번만 계산하면 되기 때문에  $\Theta(1)$ 이라는 상수 시간에 검색이 가능하다고 말할 수 있다.

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

그림 7-8 해싱의 원리

해싱 과정에서 충돌<sup>collision</sup>이 발생할 수 있다. 예를 들어 [그림 7-8]의 해시 테이블에 14를 삽입한다면,  $14 \bmod 13=1$ 므로 주소 1에 넣어야 한다. 하지만 그곳에 이미 27이 들어 있으므로 충돌이 발생한다. 알고리즘 교과서를 보면 여러 가지 유용한 해시 함수와 충돌 해결 기법이 제시되어 있다[문병로2007]. 해시 함수는 데이터를 해시 공간에 골고루 배치해 준다. 해싱은 충돌을 피할 수 있지만 충돌 가능성은 줄이고 적절한 충돌 해결 기법을 쓰면 매우 효율적인 검색 기법이 된다.

## 매칭에 적용

이제 컴퓨터 비전 문제로 관심을 옮겨 보자. 앞에서 설명했듯이 처리해야 할 대상은 하나의 키 값이 아니라 실수 여려 개로 구성된 특징 벡터  $\mathbf{x}=(x_1, x_2, \dots, x_d)$ 이다. 또한 동일한 것을 찾는 것이

아니라 최근접 이웃 또는 그것을 대신할 근사 최근접 이웃을 찾아야 한다. 가장 먼저 이해해야 할 것은 해시 함수의 특성이 앞에서와 반대라는 점이다. 이제는 골고루 퍼뜨리면 안 된다. 반대로 서로 가까운 특징 벡터들은 같은 통에 담길 확률이 높아야 하고 먼 벡터는 낮아야 한다. 다시 말해 가까운 벡터들이 충돌을 일으킬 확률을 일부러 높여야 한다. 그래야만 같은 통에서 최근접 이웃을 찾을 수 있기 때문이다.

이러한 성질을 만족하는 가장 널리 쓰이는 기법은 위치의존 해싱<sup>locality-sensitive hashing</sup>이다 [Andoni2008]. 일반 해싱은 골고루 배치하기만 하면 된다. 그 외에는 위치에 대한 어떤 조건도 없다. 하지만 위치의존 해싱에서는 서로 가까운 벡터는 같은 통에 담겨야 한다는 조건이 따라야 하므로 위치의존이라는 용어를 덧붙여 사용한다.

$H$ 를 해시 함수의 집합이라 하자. 위치의존 해싱은 하나의 해시 함수만 쓰지 않고  $H$ 에서 여러 개를 선택하여 결합해서 사용한다.  $H$ 에서 뽑힌 해시 함수  $h$ 가 식 (7.8)에 있는 조건을 만족하면  $H$ 를 위치 의존적이라 말한다. 여기에서  $p(\cdot)$ 는 확률을 뜻하고,  $\|\mathbf{a} - \mathbf{b}\|$ 는  $\mathbf{a}$ 와  $\mathbf{b}$  사이의 유clidean 거리이다.

임의의 두 점  $\mathbf{a}$ 와  $\mathbf{b}$ 에 대해,

$$\begin{aligned} \|\mathbf{a} - \mathbf{b}\| \leq R \text{이면, } p(h(\mathbf{a}) = h(\mathbf{b})) &\geq p_1 \text{이고} \\ \|\mathbf{a} - \mathbf{b}\| \geq cR \text{이면, } p(h(\mathbf{a}) = h(\mathbf{b})) &\leq p_2 \text{이다.} \end{aligned} \quad (7.8)$$

이 때  $c > 1$ ,  $p_1 > p_2$

[그림 7-9]를 보면서 식 (7.8)이 어떤 의미를 갖는지 생각해 보자.  $\|\mathbf{a} - \mathbf{b}\| \leq R$ 이란 두 벡터가 가깝다는 뜻이고,  $\|\mathbf{a} - \mathbf{b}\| \geq cR$ 은 상대적으로 멀다는 뜻이다. 따라서 두 벡터가 가까우면 같은 통에 담길(즉, 해시 함수의 값이 같을) 확률이 크고, 반대로 멀면 같은 통일 확률이 작아야 한다는 의미이다. [그림 7-9]의 예에서  $\mathbf{a}$ 는  $\mathbf{b}_1$ 이나  $\mathbf{b}_2$ 와 같은 통에 담길 확률은 크고  $\mathbf{b}_1$ 과 같은 통일 확률은 작아야 한다. 위치의존 해싱의 기본 원리는 이 식을 토대로 한다. 그런데 단 하나의 해시 함수만 쓴다면  $p_1$ 은 1에 아주 가까운 값,  $p_2$ 는 0에 가까운 값을 설정해야 한다. 하지만 이런 해시 함수를 설계하는 것은 어려울 뿐 아니라 합리적이지도 않다. 실제 구현에서는  $p_1$ 과  $p_2$ 의 격차를 적절한 정도로 설정해둔다. 대신  $H$ 에서 임의로 여러 개의 해시 함수를 골라낸 후 그것들을 조합해 근사 최근접 이웃을 찾을 확률을 높게 유지한다.

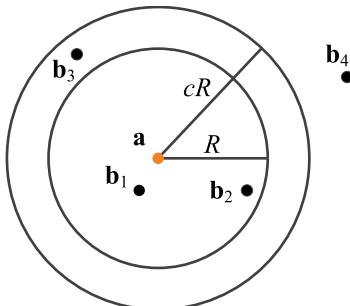


그림 7-9 조건식 (7.8)의 의미

이제 위치의존 해싱이 취하는 접근 방법과 알고리즘으로 구현하기 위해 고려해야 할 사항이 꽤 분명해졌다. 식 (7.8)을 만족하는 해시 함수 군을 어떻게 만들 것인가? 여러 가지 종류가 개발되어 있는데, 여기서는 널리 쓰이는 함수 하나를 소개한다. 함수의 정의는 식 (7.9)와 같다. 이 식에서  $\mathbf{r} \cdot \mathbf{x}$ 는 벡터  $\mathbf{r}$ 과  $\mathbf{x}$ 의 내적이고, 연산자  $\lfloor q \rfloor$ 는 실수  $q$ 를 넘지 않는 가장 큰 정수를 반환하는 연산자이다. 여기에서  $\mathbf{r}$ 과  $b$ 를 임의로 바꾸면 무수히 많은 해시 함수를 만들 수 있다. 이런 잠재적인 함수들이 집합  $H$ 를 형성한다. 벡터  $\mathbf{r}$ 은 특징 벡터와 마찬가지로  $d$ 차원인데,  $d$ 개 요소의 값을 가우시안 함수로부터 임의로 생성한다.  $b$ 는  $(0, w)$  사이의 난수이다.

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

[예제 7-4]를 살펴보면서 식 (7.8)과 (7.9)의 의미를 보다 구체적으로 생각해 보자.

#### 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  $0, 1, 2, \dots$ 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.

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

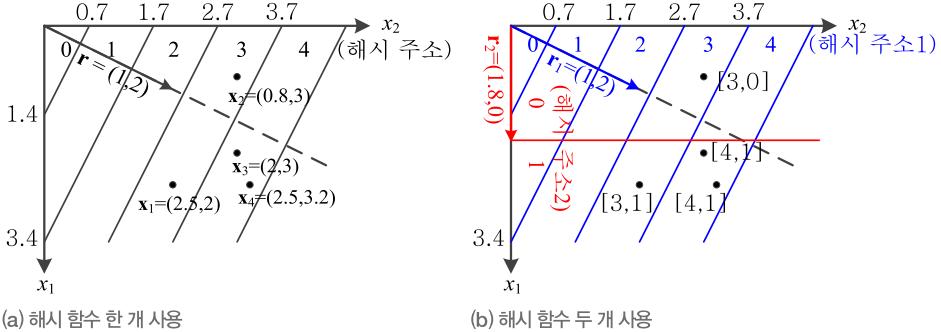


그림 7-10 해시 함수의 공간 분할과 주소 매핑

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  $r_1=(1,2)$ ,  $b=0.6$ 으로 정의되고,  $h_2$ 는  $r_2=(1.8,0)$ ,  $b=0$ 으로 정의한다고 하자. 이 상황에서는 값 두 개로 주소가 정해진다. 예를 들어 점  $\mathbf{x}_1=(2.5,2)$ 은 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.

예제를 통해 기본 원리를 살펴봤는데, 몇 가지 부가적인 설명을 덧붙이기로 하자. [그림 7-10]에서 해시 함수는 2차원 공간을 벡터  $\mathbf{r}$ 에 수직인 직선으로 나눈다. 3차원에서는 평면, 4차원 이상에서는 초평면hyperplane이 공간을 나눈다. 이때  $w$ 는 구간의 간격인데, 작으면 촘촘하게 나뉘고 크면 듬성듬성 나뉜다. 함수  $h$ 는 식 (7.8)의 위치 의존성을 만족할까? 직관적으로 생각해 보면, 가까운 벡터는 같은 구간에 놓일 확률이 크다. 하지만 경계 부근에서는 가까이 있더라도 다른 구간으로 나뉠 수 있다. 예를 들어 [그림 7-10]의 오른쪽 상황에서  $\mathbf{r}_2=(2.2,0)$ 이라면,  $\mathbf{x}_3$ 와  $\mathbf{x}_4$ 는 각각  $[4,0]$ 과  $[4,1]$ 로 매핑되어 다른 통에 담기게 된다. 하지만 가까운 두 점은 먼 두 점에 비해 같은 주소가 될 확률이 더 크다는 것이 입증되어 있다. 즉, 위치 의존성을 만족한다.

가까운 특징 벡터는 같은 통에 담길 확률이 충분히 높아야 하지만, 실제 구현에서 [그림 7-10(b)]와 같이 여러 개의 함수를 조합하여 해시 테이블을 구성하므로 그렇지 않은 경우가 발생한다. 어떻게 해야 확률을 올릴 수 있을까? 간단한 대책은 해시 테이블을 여러 개 구성하는 것이다. 어떤 해시 테이블에서는 가까운 점이 다른 통에 들어 있지만 다른 해시 테이블에서는 같은 통에 담겨 있다면 최근접 이웃을 찾는 데 문제가 되지 않는다. 이러한 방식을 사용하면 만족스러운 근사 최근접 이웃을 찾을 수 있다는 사실이 증명되어 있다[Datar2004].

지금까지 설명한 원리와 과정을 알고리즘으로 정리해 보자. [알고리즘 7-7]은 해시 테이블을 만드는 과정이다. [알고리즘 7-8]은 특징 벡터  $\mathbf{x}$ 가 주어졌을 때 해시 테이블에서  $\mathbf{x}$ 의 근사 최근접 이웃을 검색하는 알고리즘이다.

#### 알고리즘 7-7 위치의존 해시 테이블의 구축

입력 : 특징 벡터 집합  $X = \{\mathbf{x}_i, i=1, 2, \dots, n\}$ , 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  $h_i$ 를 만든다.  
5     }  
6     해시 함수  $g_j = (h_1, h_2, \dots, h_k)$ 를 만든다.  
7   }  
8   for(i=1 to n)  
9     for(j=1 to L)  $\mathbf{x}_i$ 를  $g_j$ 로 해싱하여 해당 주소의 통에 담는다.
```

#### 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 =  $\emptyset$ ; // 근사 최근접 이웃을 저장  
2   for(j=1 to L) {  
3     j번째 해시 테이블에서 주소  $g_j(\mathbf{x})$ 인 통을 조사한다.  
4     이 통에 있는 점들 중  $\mathbf{x}$ 와 거리가  $R$  이내인 것을 Q에 추가한다.  
5     Q의 크기가  $N$ 을 넘으면 break; // 이 행을 제거하면  $R$  이내인 모든 점을 Q에 저장  
6   }  
7   Q에서 거리가 가장 짧은 것을  $\mathbf{x}_{nearest}$ 로 취한다.
```

매개변수  $k, L, R, N$ 의 영향이나 식 (7.9)이외의 해시 함수 등에 관심이 있는 독자는 [Andoni 2008]을 참고하기 바란다. 이 논문의 저자 그룹이 정밀하게 코딩한 프로그램이 E2LSH라는 이름의 오픈 소스로 공개되어 있다[Andoni 2005]. 이 프로그램은 [알고리즘 7-7]의 입력인 특징 벡터 집합  $X$ 를 분석하여 적절한 값으로  $k$ 와  $L$ 을 자동으로 설정해 준다. [Shakhnarovich 2003]은 최적의 해시 함수 집합을 찾는 방법을 제안하였다.

# 3

## 기하 정렬과 변환 추정

앞 절에서 살펴본 매칭 기법은 지역 정보만 사용했다고 말할 수 있다. 첫 번째 영상에서 검출된 특징 벡터는 두 번째 영상의 특징 벡터들 중에서 오로지 자신과 가장 가까운 것을 찾아 대응 쌍을 이루기 때문이다. 이 대응 쌍은 어떤 2차원 기하 변환 관계를 가진다. 이때 특징 벡터가 검출된 물체가 강체<sup>rigid object</sup>라고 가정하자. 그러면 같은 물체에 속하는 다른 대응 쌍이 있을 때, 이 쌍의 기하 변환은 앞의 것과 비슷해야 한다. 같은 물체가 서로 다른 기하 변환을 일으킬 수 없기 때문이다. 하지만 앞 절의 매칭 알고리즘은 이런 기하 정렬<sup>geometrical alignment</sup><sup>5</sup> 조건을 전혀 고려하지 않았다. 지역 정보만 사용한 매칭 알고리즘은 오류를 내포할 수 밖에 없다. 이러한 한계를 극복하기 위해, 광역 정보를 활용하는 기하 정렬을 수행해야 한다. 이 절은 잡음, 가림, 그리고 혼재에도 불구하고 강건하게 작동하는 기하 정렬 알고리즘을 소개한다.

**TIP** 사람과 같은 유연한 물체<sup>non rigid object</sup>는 팔과 다리 등의 부품이 서로 다른 기하 변환을 가질 수 있다.

7.2절에서 구한 대응 쌍은 정확도에 따라 여러 상황으로 구분할 수 있다. [그림 7-11(a)]는 모든 쌍이 옳은 상황으로, 사람이 개입하여 대응 쌍을 검증하는 경우에 가능하다. 즉, 거짓 긍정률이 0이다. 국토지리원에서 땅의 용도를 추적하는 데 항공사진을 활용하는 경우를 예로 들어보자. 새로 찍

---

<sup>5</sup> 컴퓨터 과학에서 정렬<sup>sorting</sup>이라는 용어는 요소들을 크기 순서로 재 배열하는 작업을 뜻할 때 사용한다. 하지만 여기서는 여러 대응 쌍이 같은 기하 변환 관계를 공유함을 뜻한다.

은 영상과 이전 영상(또는 지도)을 비교하는 작업을 하는 데 지리학자가 대응 쌍을 일일이 지정하거나 시스템이 찾은 대응 쌍을 검증할 수 있다. 또 다른 응용은 의료 영상 처리이다. 몇 달 전에 찍은 영상과 현재 영상을 비교하기 위해서 사람이 개입하여 정확한 대응 쌍을 찾아내는 일은 병원에서 자주 벌어진다.



그림 7-11 대응 쌍의 여러 가지 상황

두 번째 상황은 [그림 7-11(b)]와 같이 파노라마 영상을 제작하는 경우로, 이때는 완전 자동으로 수행해야 한다. 카메라가 파노라마 사진을 제작하는 데 사람의 도움을 요청할 수는 없다. 이런 경우 대응 쌍 중에는 거짓 긍정이 다수 포함될 수 있다. 다행스런 점은 이들 중 참 긍정이 여럿 있을 것이라는 사실이다. 그렇다면 어떻게 많은 대응 쌍 무리에서 참 긍정만을 골라낼 수 있을까? 상황에 따라서 참 긍정이 적어 꽤 어려운 문제가 되기도 한다. 예를 들어, [Lowe2004]의 실험에서 혼재와 가림이 심한 영상의 경우 참 긍정이 1% 가량에 불과하였다.

이 절에서 소개하는 여러 기법은 각자 고유한 특성을 지니므로, 상황에 맞게 적용해야 한다. 예를 들어 7.3.1절의 최소제곱법은 거짓 긍정이 없는 [그림 7-11(a)]에는 적절한데, 거짓 긍정이 다수인 [그림 7-11(b)]의 상황에는 적용할 수 없다. 후자의 경우에는 최소제곱법을 확장한 강인한 추정 기법 또는 7.3.2절의 RANSAC을 적용해야 한다.

## 1. 최소제곱법과 강인한 추정 기법

### 최소제곱법

최소제곱법 least square method 은 컴퓨터 비전이 등장하기 훨씬 이전부터 통계학자와 수학자가 연구해온 오래된 기법으로서 아주 다양한 분야에 활용된다. [그림 7-12(a)]는 간단한 상황으로 이 기법의 원리를 설명한다. 주어진 네 점의 집합  $X=\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$ 를 가장 잘 대표하는 직선을 구하는 문제를 생각해 보자. 이때  $l_1$ 과  $l_2$  중에 어느 것이 더 좋을까? 직관적으로 판단하면  $l_1$ 이 더 좋다. 왜일까? 그 이유를 수학으로 설명할 수 있을까?

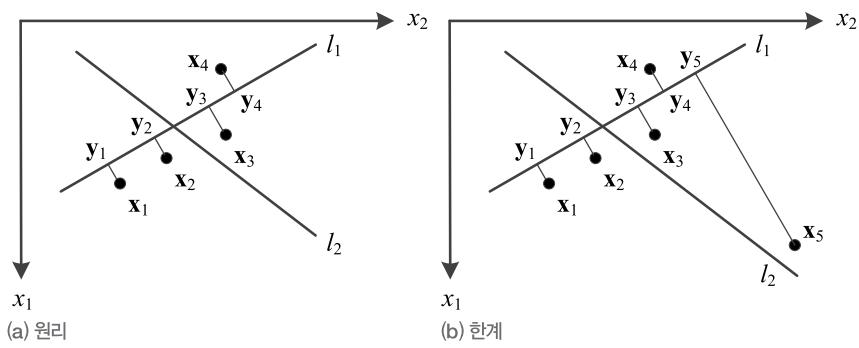


그림 7-12 최소제곱법

[그림 7-12(a)]는 네 개의 점 각각에 대해  $l_1$ 까지 거리를 보여준다. 식 (7.10)은 이 거리의 제곱 합을 오류로 취한다. 그림에서 주어진 점  $\mathbf{x}_i$ 를 직선으로 근사한 점을  $\mathbf{y}_i$ 로 표기하였다.  $\mathbf{x}_i$ 와  $\mathbf{y}_i$ 는 각각 관찰된 값과 모델이 예측한 값이라 말할 수 있다. 이때 직선이 모델 역할을 한다. 이 모델은  $x_i = ax_2 + b$ 로 표현할 수 있으며, 두 개의 매개변수  $a$ 와  $b$ 를 갖는다. 이제 최소제곱법이 풀어야 하는 문제를 수학적으로 정의할 수 있다. 식 (7.10)의 오류  $E$ 를 최소로 하는 모델의 매개변수를 추정하는 문제이다. 이렇게 정의된 문제에서 보면,  $l_1$ 은  $l_2$ 보다 최적해에 더 가깝다고 말할 수 있다.

$$E(l) = \sum_{i=1}^n r_i^2 = \sum_{i=1}^n d(\mathbf{x}_i, l)^2 = \sum_{i=1}^n \|\mathbf{x}_i - \mathbf{y}_i\|^2 \quad (7.10)$$

지금까지 점 집합이 주어졌을 때 직선을 모델로 하는 시나리오를 가지고 최소제곱법을 설명하였다. 이제 컴퓨터 비전 문제로 관심을 돌려 보자. 주어진 데이터는 점의 집합이 아니라 대응 쌍의 집합이다. 즉  $X=\{(\mathbf{a}_1, \mathbf{b}_1), (\mathbf{a}_2, \mathbf{b}_2), \dots, (\mathbf{a}_n, \mathbf{b}_n)\}$ 이다. 이 상황에서 무엇이 모델일까? 이들 대응 쌍이 같은 물체에서 발생했다면, 이들은 같은 기하 변환을 겪었을 것이다. 기하 변환은 식 (7.11)의 행

렬  $\mathbf{T}$ 로 표현할 수 있다. 이 변환 행렬은 이동과 크기, 회전 변환을 모두 포함하므로 세 변환이 동시에 일어난 상황을 다룰 수 있다.

$$\mathbf{T} = \begin{pmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{pmatrix} \quad (7.11)$$

변환 행렬  $\mathbf{T}$ 는 식 (7.12)에 따라 점  $\mathbf{a}_i$ 를  $\mathbf{b}'_i$ 로 매핑한다. 이때 실제 측정된 주어진 점  $\mathbf{b}_i$ 와  $\mathbf{T}$ 로 매핑된 점  $\mathbf{b}'_i$  사이에 차이가 발생할 수 있다. 이 차이가 오류이다. 식으로 표현하면 식 (7.13)i 된다.

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

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

당면한 문제는 오류  $E$ 를 최소로 하는  $\mathbf{T}$ 를 찾는 것이다. 즉,  $\mathbf{T}$ 가 가지고 있는 여섯 개의 매개변수를 알아내는 일이다. 이 목적을 달성하기 위해, 오류  $E$ 를 여섯 개의 매개변수  $t_{ij}$ 로 미분하여 얻은 도함수  $\frac{\partial E}{\partial t_{ij}}$ 를 0으로 놓으면 총 여섯 개의 식을 얻는데, 그것을 행렬 형태로 표현하면 식 (7.14)가 된다.

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

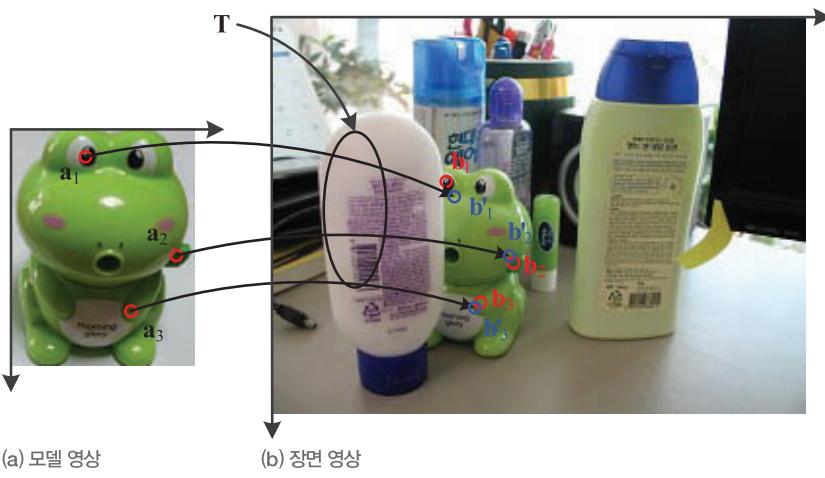


그림 7-13 최소제곱법으로 물체의 자세  $T$ 를 알아내는 사례

[그림 7-13]은 세 개의 대응점 쌍이 주어졌을 때, 식 (7.14)를 풀어  $\mathbf{T}$ 를 구한 예를 보여준다. 그림에서 빨간색 점은 관찰된(주어진) 점이고, 파란색 점은 추정한  $\mathbf{T}$ 로 예측한 점들이다. 이 예에서는 세 개의 점을 가지고 변환 행렬을 추정하였는데, 현실적으로 대응 쌍이 약간씩 위치 오류를 포함하고 있으므로  $n$ 이 클수록 보다 정확한  $\mathbf{T}$ 를 얻게 된다. 변환 행렬이 구해지면,  $X$ 에 없는 다른 점에 대해서도 그것이 나타날 지점을 예측할 수 있다. 예를 들어, [그림 7-13]에서 개구리의 코나 발 등이 장면 영상의 어디에 나타날지 예측할 수 있다.

지금까지 설명한 최소제곱법은 위치 오류가 정규 분포(가우시안 분포)를 따르는 경우에 제대로 작동한다. 이제 [그림 7-12(b)]에 관심을 가져 보자. 이 그림은 점  $\mathbf{x}_5$ 가 추가되어 상황이 조금 바뀌었다. 이 상황에서  $\mathbf{x}_5$ 는 어떤 종류의 오류로 인해 발생한 아웃라이어(outlier)라 볼 수 있다. 이제 식 (7.10)의 오류 측정치에 따르면  $I_1$ 과  $I_2$  중에 어느 것이 더 좋을까? 답은  $I_2$ 가 될 것이다. 이러한 현상이 뜻하는 바는 최소제곱법은 아웃라이어에 민감하다는 사실이다. 최소제곱법은 아웃라이어가 없다는 확신이 있는 상황에 한정해서 사용할 수 있다. 하지만 컴퓨터 비전에서 다루는 문제들은 아웃라이어를 피할 수 없다.

## 강인한 추정 기법 : M-추정과 최소제곱중앙값

다행히 아웃라이어에 대처할 수 있게 최소제곱법을 확장한 여러 변형이 있다. 여기서는 M-추정 M-estimator과 최소제곱중앙값 LMedS(Least-median of Squares)을 소개한다.<sup>6</sup> 이미 살펴보았던 최소제곱법을 다시 적어보자. 오류  $E$ 를 최소화하는 매개변수를 찾는 문제이니 식 (7.15)와 같이 쓸 수 있다.

$$\hat{\theta} = \operatorname{argmin}_{\theta} \sum_{i=1}^n r_i^2 \quad (7.15)$$

이렇게 전개된 식을 이용하여 최적값을 찾으면  $n$ 개의 점 전체가 동일한 자격으로 오류 항에 참여하므로 아웃라이어의 영향이 클 수밖에 없다. 결국 아웃라이어에 민감해진다. 아웃라이어에 대처하려면 그것들의 영향을 배제하거나 약화시키는 어떤 원리가 추가되어야 한다. 다음의 M-추정식 (7.16)과 최소제곱중앙값 식 (7.17)은 그런 기능을 가진 방법이다.

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

$$\text{최소제곱중앙값} : \hat{\theta} = \operatorname{argmin}_{\theta} \operatorname{med}_i r_i^2 \quad (7.17)$$

M-추정의 기본 원리는 식 (7.16)의 함수  $\rho(r_i)$ 에 있다. 여기서  $\rho(r_i)=r_i^2$ 이면 최소제곱법이 된다.  $\rho(r_i)$ 는  $r_i$ 가 어느 정도 이상이면 함수의 크기를 떨어뜨려 영향력을 줄인다. 식 (7.18)은 이런 특성을 가진 여러 함수 중의 하나로서 Huber 함수라 부른다. 이 함수는  $r$ 의 값이 작을 때( $|r| \leq c$ )는 2차 곡선을 사용하지만, 클 때는 1차 곡선을 사용하여 영향력을 누그러뜨린다.

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

식 (7.17)이 설명하는 최소제곱중앙값은  $n$ 개의 값 중에 중앙값을 취하고 그것을 최소로 하는 매개변수를 찾는다. 따라서 중앙값보다 큰 50%의 점들은 중앙값을 결정하는 단계까지만 개입하고 그 이후 오류를 계산하는 과정에서 아예 배제되는 셈이다. 여기서는 이 두 가지 강인한 추정 방법에 대한 개략적인 아이디어만 제시하였다. 이들을 컴퓨터 비전 문제에 적용하는 데 필요한 세부적인 사항은 [Stewart99, Meer91, Rousseeuw87]을 참고하기 바란다.

---

<sup>6</sup> 이들 두 기법은 각각 [Huber81]과 [Rousseeuw84]가 제안하였다.

지금까지 공부한 세 가지 기법을 봉괴점breakdown point과 계산 효율 측면에서 비교해 보자. 봉괴점이란 그 이상이 되면 추정 알고리즘이 더 이상 작동하지 않는 아웃라이어 비율을 뜻한다. 예를 들어 최소제곱법은 아웃라이어가 하나라도 포함되면 오작동을 할 수 있으므로 봉괴점은 0%이다. 최소제곱중앙값은 50%이고 M-추정은 (여러 가지 상황에 따라 다른데) 다른 두 기법 사이에 위치한다. 최소제곱중앙값은 중앙값을 찾는 연산 때문에 다른 두 기법에 비해 느리다.

## 2. RANSAC

이 기법은 3.5.3절에서 공부한 적이 있다. 그때는 에지 점들로부터 직선을 추정하는 데 RANSAC을 활용하였다. 기본 원리는 같지만, 매칭에서는 입력이 대응점 쌍이고 출력이 기하 변환 행렬이다. [알고리즘 7-9]는 기하 변환을 추정해 주는 RANSAC이다.

### 알고리즘 7-9 기하 변환을 추정하기 위한 RANSAC

**입력 :**  $X = \{(a_i, b_i), i=1, 2, \dots, n\}$  // 매칭 쌍 집합  
 반복 횟수  $k$ , 인라이어 판단  $t$ , 인라이어 집합의 크기  $d$ , 적합 오차  $\epsilon$   
**출력 :** 기하 변환 행렬  $T$

```

1   Q = Ø;
2   for(j=1 to k) {
3       X에서 세 개 대응점 쌍을 임의로 선택한다.
4       이들 세 쌍을 입력으로 식 (7.14)를 풀어  $T_j$ 를 추정한다.
5       이들 세 쌍으로 집합  $inlier$ 를 초기화한다.
6       for(이 세 쌍을 제외한 X의 요소  $p$  각각에 대해) {
7           if( $p$ 가 허용 오차  $t$  이내로  $T_j$ 에 적합)  $p$ 를  $inlier$ 에 넣는다.
8       }
9       if( $|inlier| \geq d$ ) // 집합  $inlier$ 가  $d$ 개 이상의 샘플을 가지면
10           $inlier$ 에 있는 모든 샘플을 가지고 새로운  $T_j$ 를 계산한다.
11          if( $T_j$ 의 적합 오류  $< \epsilon$ )  $T_j$ 를 집합 Q에 넣는다.
12      }
13  Q에 있는 변환 행렬 중 가장 좋은 것을  $T$ 로 취한다.

```

알고리즘 작동 원리는 3.5.3절의 [알고리즘 3-8]과 같기 때문에, 그때 사용한 [그림 3-31]을 다시 살펴보기 바란다. [알고리즘 7-9]는 3행에서 임의로 세 개의 대응 쌍을 선택한다. 즉  $n$ 개의 대응 쌍을 동등하게 취급하는 셈이다. 하지만 매칭 과정을 잘 들여다보면  $n$ 개가 동등한 입장은 아니

다. 대응 쌍은 식 (7.5) 또는 식 (7.7)과 같은 척도에 따라 맷어진다. 이때 이들 값이 작을수록 두 특징 벡터는 더 가깝고, RANSAC 알고리즘 관점에서 볼 때 인라이어일 가능성이 높다. RANSAC을 개조한 PROSAC은 3행을 다음과 같이 수정하여 이런 점을 고려한다. 매칭 점수를 산정하고 그것을 확률로 바꾸는 구체적인 방법은 [Chum2005]를 참고하기 바란다.

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

RANSAC은 동작 과정에서 난수를 생성하므로 수행할 때마다 다른 답을 만들어 주는 비결정적 인 알고리즘이다. 게다가 시간을 더 주면 더 좋은 품질의 답을 생성하는 특성을 가진다. [알고리즘 7-9]는  $k$ 번 반복하는데,  $k$ 에 따른 답의 품질을 따져보자.  $q$ 를  $X$ 의 인라이어 비율( $q=\text{인라이어 개수}/n$ )이라 하자. 3행은 세 개의 대응 쌍을 선택하는데, 이들 셋이 모두 인라이어일 때만 옳은 답을 만들어낼 가능성이 생긴다. 이 경우를 성공 후보라 부르면 세 점이 성공 후보가 될 확률은  $q^3$ 이고, 실패 후보일 확률은  $1-q^3$ 이다. 따라서  $k$ 번 반복했을 때 모두 실패 후보일 확률, 즉 옳은 답을 전혀 기대할 수 없는 확률은  $(1-q^3)^k$ 이다. 예를 들어,  $q=0.5$ 라면  $0.875^k$ 이다. 만일 열 번( $k=10$ ) 반복했다면 0.263이다. 옳은 답을 기대할 수 없는 확률을  $p_{\text{thres}}$  이하로 낮추고 싶다면  $k$ 를 식 (7.19)보다 크게 해야 한다.

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

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

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

계산 시간을 더 주면 더 좋은 품질의 해를 찾아주는 특성은 단점일 수도 있지만, 충분한 계산 시간을 확보할 수 있는 상황에서는 장점이 될 수 있다. [Choi2009]는 RANSAC의 여러 변형과 성능 평과 결과를 제시한다. RANSAC을 적용하여 응용 문제를 푼 사례로는 같은 관광 명소를 찍은 다수의 2차원 영상으로부터 3차원 장면을 구성하고 3차원 렌더링을 수행하는 사진 관광 [Snavely2006], 증강 현실[Wagner2010, Gordon2006], 파노라마 영상 제작[Brown2003] 등이 있다.

**TIP** 7.4절은 이들 응용을 다룬다.

# 4

## 웹과 모바일 응용

디지털 카메라는 1975년에 처음 발명된 후 급속도로 퍼져나가, 이제는 100만 화소 이상의 카메라가 장착된 모바일 기기도 흔해졌다. 장소에 상관 없이 인터넷에 접속하는 일도 보편화되면서 [그림 7-14]와 같이 디지털 카메라로 찍은 사진을 바로 인터넷에 올리는 것도 일상이 되었다. 2013년 3월에 나온 Flickr에 관한 보고서를 보면 하루에 Flickr에 올라오는 사진은 350만 장으로 알려져 있다.<sup>7</sup>



그림 7-14 인터넷에 쓰이는 영상

<sup>7</sup> Flickr는 2004년에 개설된 영상 호스팅 서비스이다. <http://www.flickr.com/>

이러한 현상이 컴퓨터 비전과 무슨 관련이 있을까? ‘인터넷 비전’이라는 새로운 연구 주제가 만들어졌으며 2008년에는 최초로 인터넷 비전 학술대회가 개최되었다. 인터넷 비전은 크게 두 가지 길로 나누어볼 수 있다. 첫 번째는 새로운 응용 분야 창출이고, 두 번째는 인식과 같은 문제를 푸는 새로운 접근 방법의 개발이다[Avidan2010]. 이 절은 주로 웹 또는 모바일 플랫폼을 통해 제공되는 새로운 응용 분야 두 가지를 살펴본다.<sup>8</sup> 두 번째 길은 [Avidan2010]이 소개하는 논문을 참고하기 바란다.<sup>9</sup>

## 1. 파노라마 영상 제작

[그림 7-15]는 여러 장의 영상을 이어 붙여 만든 파노라마 영상이다.<sup>10</sup> 현재 파노라마 영상은 제작 기술이 보편화되어 디지털 카메라나 스마트폰 앱으로도 손쉽게 만들 수 있다.



그림 7-15 파노라마 영상

<sup>8</sup> 컴퓨터 비전을 모바일 환경에 적용하는 일에 관심이 있는 독자는 IEEE Workshop on Mobile Vision을 참고하기 바란다.

1차(2010년) : <http://www.cs.stevens.edu/~ghua/ghweb/IWMVProgram.htm>

2차(2011년) : <http://www.cs.stevens.edu/~ghua/ghweb/IWMVProgram2011.htm>

3차(2013년) : [http://www.cs.stevens.edu/~ghua/ghweb/IWMV/CFP\\_Third\\_IEEE\\_Mobile\\_Vision\\_Workshop.html](http://www.cs.stevens.edu/~ghua/ghweb/IWMV/CFP_Third_IEEE_Mobile_Vision_Workshop.html)

<sup>9</sup> 예를 들어, Stone은 현재 얼굴 인식 프로그램이 낮은 성능을 보이는 사례를 제시하고 정확도를 개선할 목적으로 페이스북과 같은 사회관계망 서비스(SNS)에서 발생하는 정보를 활용하는 기법을 제안하였다[Stone2010]. 이 기법에서는 영상을 찍은 사람, 찍은 GPS 위치, 친구 관계 등의 정보를 인식 과정에 추가로 활용한다.

<sup>10</sup> 영상을 이어 붙이는 알고리즘을 가장 자세히 설명한 문헌으로 [Szeliski2006, Zitova2003]을 추천한다.

파노라마 영상을 제작하려는 사람은 [그림 7-16(a)]와 같이 방향을 조금씩 변화시키며 같은 장면을 겹치게 찍는다. 이렇게 찍은 영상을  $f_i$ ,  $i=1, 2, \dots, k$ 라 하자. 이제 알고리즘은 이들의 이음선을 찾은 후 자국이 남지 않도록 정교하게 이어 붙여야 한다. 파노라마 영상을 만들어 주는 알고리즘은 다음과 같으며, [그림 7-16]은 [알고리즘 7-10]의 수행 과정을 보여준다.

### 알고리즘 7-10 파노라마 영상 제작

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

**출력 :** 파노라마 영상  $p$

- ```

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

```

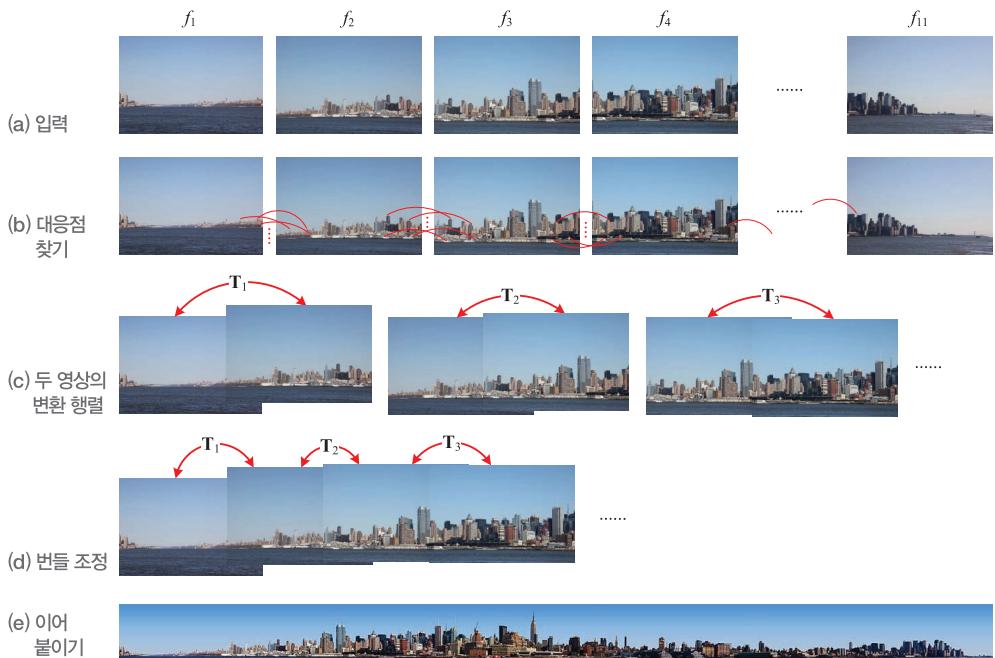


그림 7-16 파노라마 제작 과정

파노라마 제작에서 가장 중요한 점은 영상을 이어 붙였다는 사실을 보는 사람이 인지하지 못할 정도로 사실성을 확보하는 것이다. 그러기 위해서는 카메라에 대한 변환을 보다 정교하게 표현하고 더욱 정확하게 추정해야 한다. 카메라 변환은 카메라 위치를 나타내는 세 개의 매개변수, 회전을 나타내는 세 개의 매개변수, 초점 거리를 나타내는 하나의 매개변수로 표현된다.<sup>11</sup>

[알고리즘 7-10]의 4행은 이웃한 두 영상에 대해 RANSAC을 이용하여 일관성을 갖춘 대응점 집합을 찾아낸다. 즉, 아웃라이어에 해당하는 대응점을 제거한다. 이렇게 추정한 변환 행렬은 얼마 정도의 오류를 피할 수 없다. 6행은  $k-1$  개의 변환 행렬을 동시에 살펴으로써 최적의 변환 행렬을 추정한다. 이 과정을 번들 조정bundle adjustment이라 부른다.<sup>12</sup>

[그림 7-16]에서 번들 조정이 된 영상을 살펴보면 위치, 회전, 크기, 조명 등에서 변화가 발생하였다. 이제 이음선을 눈치채지 못하도록 정교하게 이어 붙이는 방법을 사용해야 한다. Brown은 영상의 중앙은 1, 가장자리는 0이라는 값을 주고 중앙에서 가장자리로 가면서 선형적으로 줄어드는 가중치 함수  $w()$ 를 사용해 두 영상을 선형 결합하였다[Brown2003]. 하지만 이런 단순한 결합은 예지 부근에서 블러링 현상을 일으키므로 Burt와 Adelson이 개발한 다중 밴드 결합 알고리즘을 추가로 사용한다[Burt83b].

스마트폰용 앱 스토어에 두 종류의 유명한 파노라마 제작 앱이 있다. 이들 앱은 SIFT를 창안한 Lowe 교수가 창업한 벤처 업체에서 제작한 AutoStitch와 마이크로소프트가 제작한 Photosynth이다.<sup>13</sup> 이들에 대해 좀더 자세히 공부하려는 독자에게 [Brown2003, Brown2007]과 [Snavely2010]을 추천한다.

## 2. 사진 관광

요즘 여행객들은 디지털 카메라와 스마트폰을 가지고 다니며 찍은 영상을 [그림 7-14]와 같이 인터넷에 올린다. 그 결과 유명한 장소는 수천~수백만 장의 사진이 인터넷에 쌓인다.

[그림 7-17]은 같은 건물을 서로 다른 여덟 개 지점에서 촬영한 상황을 보여준다. 이때 여덟 장

<sup>11</sup> 카메라 기하에 대해 보다 근본적인 공부를 원하는 사람에게 [Hartley2000, Moons2010]을 추천한다.

<sup>12</sup> 번들 조정에 대한 훌륭한 논문이 있다[Triggs99].

<sup>13</sup> 이들의 공식 홈페이지는 각각 <http://www.cloudburstresearch.com/>과 <http://photosynth.net/>이다. Autostitch는 아이폰과 안드로이드용이 있으며 유료이다. Photosynth는 아이폰용만 있으며 무료이다(2014년 5월 기준).

의 영상  $f_1, f_2, \dots, f_8$ 이 얻어질 것이다. 이들 영상은 같은 장소를 찍었다는 공통점이 있지만, 카메라 시점(카메라의 위치와 방향)camera viewpoint은 모두 다르다. 또한 카메라의 초점 거리와 노출도 다르고, 다양한 전경(대부분 사람)을 포함한다. 게다가 시간과 날씨 역시 변동이 크다.

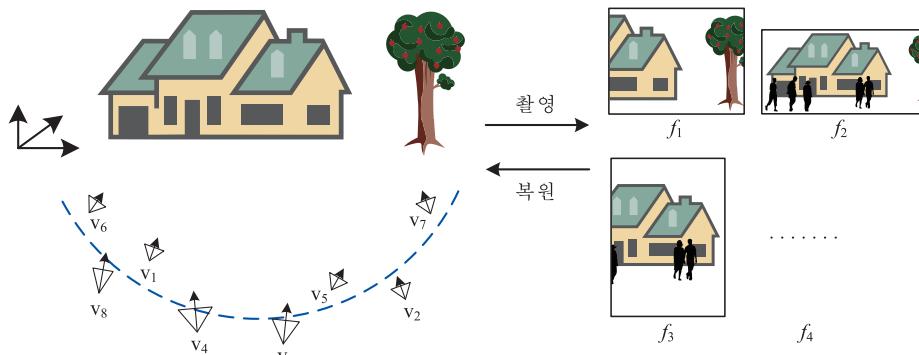


그림 7-17 같은 장소를 여러 시점에서 촬영 – 시점과 3차원 장면을 복원할 수 있을까?

인터넷에서 [그림 7-17] 오른쪽과 같은 영상을  $k$ 장 구했다고 하자. 이들 영상으로 카메라 시점 viewpoint과 장면에 나타난 물체의 3차원 모양을 복원할 수 있을까? 이러한 복원 작업을 모션에서 구조 추정structure from motion이라 부른다. 이러한 정보를 복원하면 어떤 일에 활용할 수 있을까?

카메라 시점과 3차원 정보가 복원되었다고 가정하고, 활용에 대해 먼저 살펴보자. [그림 7-17]에 있는 파란색 점선은 복원한 시점과 가깝게 지나는 궤적이다. 관찰자는 마우스로 궤적을 따라 원하는 곳을 돌아다닐 수 있다. 시스템은 현재 관찰자 시점에 가장 가까운 2~3개의 시점을 찾는다. 그런 다음 이들 시점에 해당하는 영상을 보간하여 현재 시점의 영상을 제작하고 화면에 보여준다. 시스템이 실시간으로 영상을 제작하거나 미리 제작해 둔 영상을 보여준다면 관찰자는 3차원 공간을 옮겨 다니며 가상 관광을 하는 느낌을 받게 된다. 이러한 응용을 사진 관광photo tourism이라 부른다[Snavely2006, Snavely2010]. [그림 7-18]은 로마에 있는 팬테온의 사진 관광이다. 녹색 점은 건물 바깥인데 궤적을 따라 건물 안에 있는 빨간색 점까지 이동할 수 있다. 파란색 선은 복원된 21개의 시점을 단순히 직선으로 이어놓은 것이다. 이 초기 궤적은 매끄럽지 못하므로, 경로 계획 알고리즘을 이용하여 부드럽게 움직이는 검은색 선 궤적을 생성한다. 검은색 선에 붙어있는 빨간색 선분은 시점이 바라보는 방향을 나타낸다.<sup>14</sup>

<sup>14</sup> 웹에서 사진 관광을 경험해보길 권한다. <http://photosynth.net>

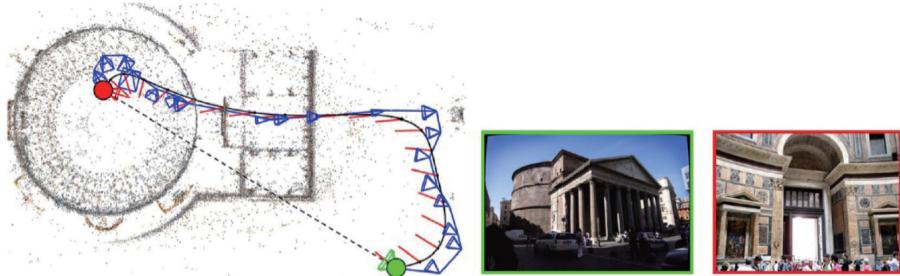


그림 7-18 사진 관광

[그림 7-19]는 또 다른 재미있는 응용 사례를 보여준다. 맨 왼쪽은 중요한 장면에 사람이 주석을 붙인 결과이다. 이렇게 한 장의 사진에 주석을 붙여 놓으면, 시스템은 다른 사진의 대응하는 지점에 자동으로 주석을 붙여 준다. 일종의 증강 현실이다. 이러한 기능이 어떻게 가능한지 이해하려면 [그림 7-17]에 주목해 보자. TIP 증강 현실의 또 다른 예를 [그림 9-20]에서 볼 수 있다.

누군가가  $f_2$  영상 속 굴뚝에 ‘100년 전 모양이 그대로 보존된 것으로 섬세한 무늬가 돋보인다’라는 주석을 붙였다고 하자. 시스템은 다른 영상이  $f_2$ 와 어떤 기하 변환 관계를 가지는지에 대한 정보를 정확히 알고 있으므로 다른 영상의 같은 지점에 동일한 주석을 자동으로 붙일 수 있다. 이외에도 여러 가지 흥미로운 작업에 응용이 가능한데 자세한 내용은 [Snavely2010]을 참고하기 바란다.

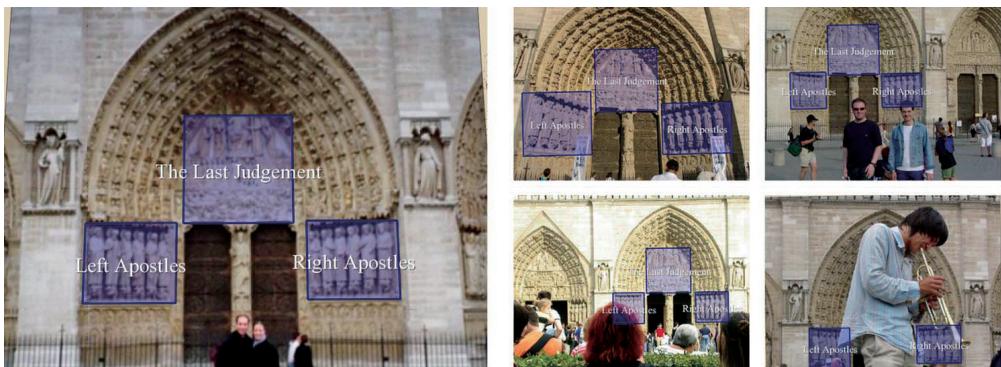


그림 7-19 자동 주석 붙이기

이제 카메라 시점과 3차원 물체 정보를 복원하는 방법에 대해 살펴보자. 전체 처리 과정 중 많은 부분이 앞 절의 파노라마 제작과 비슷하고, 몇 가지가 추가된다. 파노라마와 가장 큰 차이는 여러 사람이 다른 시간에 다른 관심을 가지고 찍었기 때문에 시점의 변화가 훨씬 크다는 것이다. 보통 파노라마를 제작하는 사람은 좋은 영상을 얻기 위해 일관성 있게 시점을 변화시킨다. 또한 사진 관

광에서는 날씨 및 찍은 시간(낮/밤), 전경에 넣는 사람이나 물체 등에 따라 변화가 크게 발생한다. [알고리즘 7-11]은 사진 관광 시스템을 구축하는 처리 과정을 설명한다. 파노라마와 달리 겹치는 영상 집합을 찾아내기 위해 그래프를 사용한다.

### 알고리즘 7-11 사진 관광에 필요한 정보 추정

입력 : 같은 장면을 찍은 영상 집합  $f_i$ ,  $1 \leq i \leq k$ , 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( $i \neq j$ 이고  $f_i$ 와  $f_j$  사이에 대응점이  $t$ 개 이상이면) { // 겹침 조사
6       RANSAC을 적용하여 변환 행렬  $T$ 를 구한다.
7        $T$ 의 신뢰도가  $c$  이상이면  $f_i$ 와  $f_j$  사이에 에지를 부여한다.
8     }
9   for(그래프의 연결요소 각각에 대해)
10    번들 조정을 수행하여 카메라 시점과 3차원 점을 구한다.
```

이 알고리즘의 입력은 [그림 7-16]이 아니라 [그림 7-17] 또는 [그림 7-19]와 같은 상황이다. 3~8행은 모든 영상 쌍에 대해 돌 사이에 겹침이 있는지 조사하고, 그렇다면 두 영상에 해당하는 노드 사이에 에지를 부여하여 그래프를 구성한다. 9~10행은 그래프에 존재하는 연결요소 각각에 대해 번들 조정을 적용하여 카메라 시점과 장면의 3차원 점의 좌표를 계산한다. 이렇게 구한 정보는 [그림 7-18], [그림 7-19]와 같은 응용에 활용한다.

## 연습문제

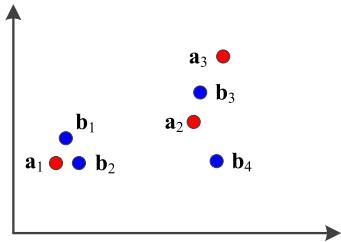
1 [예제 7-1]에서 점 (2,2)가 추가되어 다섯 개의 점이 확률 분포를 이룬다고 가정하자.

(1) [그림 7-3]과 같은 방식으로 새로운 분포를 그리시오.

(2) 새로운 분포에 대해 [예제 7-1]의 계산을 하시오.

2 문제 1의 새로운 분포에 대해 [예제 7-2]의 계산을 하시오.

3 같은 물체를 포함하는 두 장의 영상이 있는데, 첫 번째 영상에서 특징 벡터  $a_1 \sim a_3$ , 두 번째 영상에서  $b_1 \sim b_4$ 를 추출하였다고 가정하자. 이들 특징 벡터는 다음 그림과 같은 값을 가진다. 이 상황에서 식 (7.5)와 식 (7.7)의 전략 각각에 대해, 매칭에 성공할 가능성 순서에 따라  $a_1, a_2, a_3$ 를 나열하시오.



4 식 (7.7)을 사용하는 최근접 거리 비율 전략에 대해 생각해 보자.  $T$ 를 크게 했을 때, 매칭 쌍의 개수, 거짓 긍정, 거짓 부정이 어떻게 변할지 설명하시오.

5 [알고리즘 7-2]를 식 (7.7)의 최근접 거리 비율 전략을 사용하는 버전으로 바꾸어 쓰시오.

6 [알고리즘 7-6]은 힙heap을 사용한다.

(1) 힙이 수행하는 두 종류의 연산이 무엇인지 쓰시오.

(2) 일반적인 1차원 배열 대신 힙을 사용할 때 얻는 이점을 두 종류의 연산과 연결지어 설명하시오.

- 7 Chum은 RANSAC을 개선한 PROSAC을 제안하였다[Chum2005]. 핵심 아이디어는 7.3.2절에서 설명한 “매칭 점수가 높을수록 선택 확률이 높은 방식에 따라,  $X$ 에서 세 쌍을 선택한다”에 있다. PROSAC이 이 아이디어를 어떻게 구현하는지 구체적으로 설명하시오.
- 8 파노라마를 제작할 때 인접한 두 영상을 표시가 나지 않게 붙여야 하는데, 이때 [Burt83b]의 다중 밴드 결합 알고리즘을 사용한다. 이 알고리즘의 원리를 조사하고 설명하시오.
- 9 <http://photosynth.net>에 접속하면 Photosynth로 제작한 파노라마 영상을 감상할 수 있다. 파노라마 영상을 세심하게 살펴보면서 부자연스럽게 접합된 부분이 있는지 확인하시오. 만일 그런 곳을 발견하였다면, 화면을 캡처하여 제시하시오.