컴퓨터 비전과 딥러닝
챕터 5 정리

AI 학과 2444337 손주안

Chapter 5

5.1 발상

5.2 이동과 회전 불변한 지역 특징

5.3 스케일 불변한 지역 특징

5.4 SIFT

5.5 매칭

5.6 호모그래피 추정

발상 |5.1

출처: motchallenge.net의 MOT-17-14-SDP

- 2개의 프레임 속 버스를 추적하기 위해서는 특징점을 추출하고 매칭을 통해 해당하는 특징점 쌍을 찾아야 한다.

|불변성

- 물체에 이동, 회전, 스케일(크기) 변환이 발생하더라도 이러한 조건을 만족하는 특징

발상 |5.1

|반복성
같은 물체가 서로 다른 두 영상에서 나타났을 때, 첫 번째 영상에서 검출된 특징점이 두 번째 영상에서도 같은 위치에서 높을 확률로 검출돼야 한다.

|불변성
물체에 이동, 회전, 스케일, 조명 변환이 일어나도 특징 기술자의 값이 비슷해야 한다.

|분별력
물체의 다른 곳에서 추출된 특징과 두드러지게 달라야 한다.

|지역성
작은 영역을 중심으로 특징 벡터를 추출해야 한다.

|적당한 양
특징점이 많으면 더 정확하게 추적할 수 있으나 너무 많으면 계산 시간이 과다해진다.

|계산 효율
초당 몇 프레임 이상을 처리해야 하는 등의 조건이 있다.

|지역 특징

- 특징점을 물체의 모퉁이에서 찾는 것이 아닌, 물체의 같은 곳이 두 영상 모두에서 추출돼야 한다는 반복성을 중요하게 여기는 발상
- 다음 조건들은 **상충 관계**에 있다

이동과 회전 불변한 지역 특징 | 5.2 모라벡 알고리즘

- 여러 방향으로 색상 변화가 있는 특징점이 영상 내에서 찾기 쉽다.
- 다음 식은 각 화소마다 적용된다.
- V, u를 각각 -1, 0 –로 변화시켜 3*3 맵을 생성한다.

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

이동과 회전 불변한 지역 특징 | 5.2 모라벡 알고리즘

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	c 0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0
0	0	0	b 1	1	1	0	0	0	0
0	0	0	1	1	1	1	0	0	0
0	0	0	1	1	1	1	a 1	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

기존 영상

3	4	4
2	0	2
4	3	2

a

3	1	6
3	0	4
3	0	3

b

0	0	0
0	0	0
0	0	0

c

S맵

- 수직, 수평으로 모두 변화가 있는 a가 지역 특징으로 적합하다.
- 상하좌우 4개의 화소 중 최솟값을 C로 취하면 a가 가장 좋은 점수를 받게 된다.
- 하지만 실제 영상에 적용하기에 모라벡 알고리즘은 한계가 있다.

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

이동과 회전 불변한 지역 특징 | 5.2 해리스 특징점

- 위 식은 잡음에 대처하기 위해 기존 식에 가우시안 G를 적용한다.
- 이후 식을 정리하여 아래 식으로 정리할 수 있고, 행렬 **A를 따로 떼어** 쓴다.
- A는 정수 외 실수도 가능하며, 어떤 화소 주위의 영상 구조를 표현하여 A만 분석해도 지역 특징 여부를 판단할 수 있다.
- *는 컨볼루션 연산자, $d_y(y,x)$와 $d_x(y,x)$는 각각 y와 x 방향의 미분값이다.

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

$$S(v,u) \cong (v,u) \begin{pmatrix} G * d_y^2 & G * d_y d_x \\ G * d_y d_x & G * d_x^2 \end{pmatrix} \begin{pmatrix} v \\ u \end{pmatrix} = uAu^T$$

$$A = \begin{pmatrix} G * d_y^2 & G * d_y d_x \\ G * d_y d_x & G * d_x^2 \end{pmatrix}$$

이동과 회전 불변한 지역 특징 | 5.2 해리스 특징점

	a	b	c
2차 모멘트 행렬	$A = \begin{pmatrix} 0.52 & -0.2 \\ -0.2 & 0.52 \end{pmatrix}$	$A = \begin{pmatrix} 0.08 & -0.08 \\ -0.08 & 0.8 \end{pmatrix}$	$A = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
고윳값	$\lambda_1 = 0.72, \lambda_2 = 0.33$	$\lambda_1 = 0.81, \lambda_2 = 0.07$	$\lambda_1 = 0.0, \lambda_2 = 0.0$
특징 가능성 값	C = 0.1925	C = 0.0237	C = 0.0

- A가 두 고윳값이 모두 크므로 지역 특징으로 적합하다.

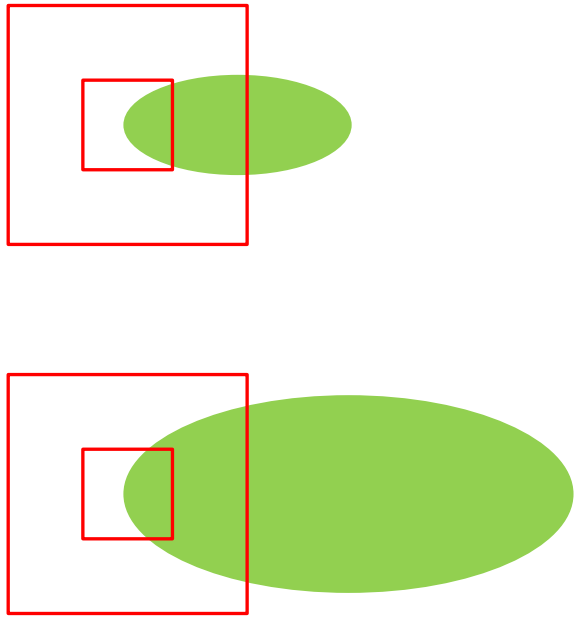
이동과 회전 불변한 지역 특징 | 5.2 해리스 특징점

- 위 식으로 지역 특징일 가능성을 정의한다.
- $A = \begin{pmatrix} p & r \\ r & q \end{pmatrix}$이라 하면 고윳값의 합은 $\lambda_1 + \lambda_2$ = p + q, 곱은 $\lambda_1\lambda_2 = pq - r^2$이므로 아래 식으로 사용이 가능하다.

$$C = \lambda_1\lambda_2 - k(\lambda_1+\lambda_2)^2$$

$$C = (pq - r^2 - k(p+q)^2$$

이동과 회전 불변한 지역 특징 | 5.2 해리스 특징점



- 해리스 특징점은 **이동과 회전에 있어 불변**이다.
- **크기 변환에 있어 등변**인데, 물체에 어떤 크기의 마스크를 적용하는지에 따라 달라지기 때문이다.

이동과 회전 불변한 지역 특징 | 5.2 실행 결과

```
import cv2
import numpy as np

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

ux = np.array([[-1,0,1]])
uy = np.array([-1,0,1]).transpose()
k = cv2.getGaussianKernel(3,1)
g = np.outer(k,k.transpose())

dy = cv2.filter2D(img, cv2.CV_32F, uy)
dx = cv2.filter2D(img, cv2.CV_32F, ux)
dyy = dy*dy
dxx = dx*dx
dyx = dy*dx
gdyy = cv2.filter2D(dyy,cv2.CV_32F,g)
gdxx = cv2.filter2D(dxx,cv2.CV_32F,g)
```

```
gdyx = cv2.filter2D(dyx,cv2.CV_32F,g)
C = (gdyy*gdxx - gdyx*gdyx) - 0.04*(gdyy+gdxx)*(gdyy+gdxx)

for j in range(1,C.shape[0]-1):
    for i in range(1,C.shape[1]-1):
        if C[j,i]>0.1 and sum(sum(C[j,i]>C[j-1:j+2,i-1:i+2])) == 8:
            img[j,i] = 9

np.set_printoptions(precision = 2)
print(dy)
print(dx)
print(dyy)
print(dxx)
print(dyx)
print(gdyy)
print(gdxx)
print(gdyx)
print(C)
print(img)

popping = np.zeros([160,160],np.uint8)

for j in range(0,160):
    for i in range(0,160):
        popping[j,i] = np.uint8((C[j//16,i//16]+0.06)*700)

cv2.imshow('imgage display2', popping)
cv2.waitKey()
cv2.destroyAllWindows()
```

이동과 회전 불변한 지역 특징 | 5.2 실행 결과

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

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

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

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

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

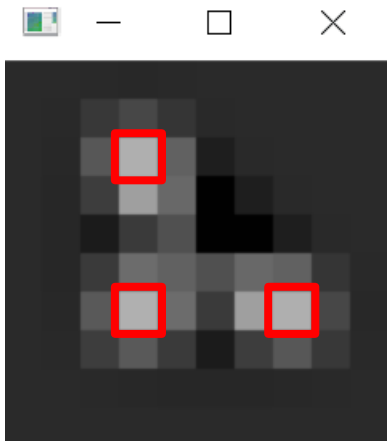
```
[[0.   0.   0.15 0.25 0.15 0.   0.   0.   0.   0.  ]
 [0.   0.   0.2  0.4  0.32 0.08 0.   0.   0.   0.  ]
 [0.   0.   0.2  0.53 0.6  0.32 0.08 0.   0.   0.  ]
 [0.   0.   0.08 0.32 0.6  0.6  0.32 0.08 0.   0.  ]
 [0.   0.   0.   0.08 0.32 0.6  0.6  0.32 0.08 0.  ]
 [0.   0.   0.08 0.2  0.35 0.6 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

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

이동과 회전 불변한 지역 특징 | 5.2 실행 결과

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

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



- 검출된 세 특징점은 모두 삼각형의 모퉁이지만 실제 영상에 적용할 경우 모퉁이뿐 아니라 둥근 모양의 **블롭**도 많이 검출한다.

스케일 불변한 지역 특징 |5.3

|스케일 공간

- 인간이 거리에 관계없이 같은 물체를 같다고 인식하는 능력을 컴퓨터 비전에 나타낸 것으로 SIFT가 있다.

- 다중 스케일 영상은 가까이부터 멀리까지 본 장면을 표현해야 하는데 입력은 1장의 명암 영상에 불과하기 때문에 여러 거리에서 보았을 때의 **현상을 흉내 내야** 한다.

- **첫 번째**는 거리가 멀어질수록 세부 내용이 **점점 흐려지는 현상**을 가우시안 필터로 입력 영상을 스무딩하여 시뮬레이션하는 것이다.

- **두 번째**는 거리가 멀어질수록 물체의 **크기가 작아지는 현**상을 영상의 크기를 반으로 줄여 피라미드 영상으로 시뮬레이션하는 것이다.

스케일 불변한 지역 특징 |5.3

피라미드 방법은 영상을 연속적인 스케일로 줄일 수 없지만 가우시안 스무딩은 연속된 값으로 조절할 수 있다.

스케일 공간의 미분은 라플라시안을 주로 사용한다. d_{yy}와 d_{xx}는 각각 y와 x로 2번 미분한 영상이다.

라플라시안은 스케일 공간에서 극점을 찾는 데 유리하다.

실제 구현에서는 정규 라플라시안을 이용한다.

라플라시안: $\nabla^2 f = d_{yy} + d_{xx}$

정규 라플라시안:
$\nabla^2_{normal} f = \sigma^2 |d_{yy} + d_{xx}|$

SIFT | 5.4

SIFT

- 스케일 공간에서 특징점을 검출하는 알고리즘의 다양한 변형 중 가장 성공적이고 지금까지 널리 쓰인다.

- 3단계를 거쳐 특징점을 검출한다.

SIFT | 5.4

1단계: 다중 스케일 영상 구축

- 가우시안 스무딩과 피라미드 방법을 결합해 다중 스케일 영상을 구성한다.
- 가우시안 스무딩을 적용한 영상들을 6장 만들고 **옥타브**라고 부른다.

- 스무딩을 적용한 영상에 $k\sigma$로 스무딩하여 6개의 영상을 얻는다.($k = 2^{1/3}$)
- 원래 영상에는 $\sigma_1 = 1.6$으로 스무딩하여 시작한다.

- 하지만 실제 구현에서는 원본 영상이 0.5만큼 스무딩되어 있다고 가정하여 $\sigma_1 = \sqrt{1.6^2 - 0.5^2}$를 사용한다.

- 이후 옥타브의 **4번째 영상을 반으로 축소**해 동일한 과정으로 새 옥타브를 얻는다.
- 이때, **첫 영상에는 추가 스무딩**을 적용하지 않는다.

SIFT | 5.4

2단계: 다중 스케일 영상에 미분 적용

- 정규 라플라시안 대신 매우 유사한 DOG를 사용하여 계산 시간을 줄인다.

- DOG는 이웃한 영상을 화소별로 빼서 만드므로 다섯 장의 영상이 생긴다.

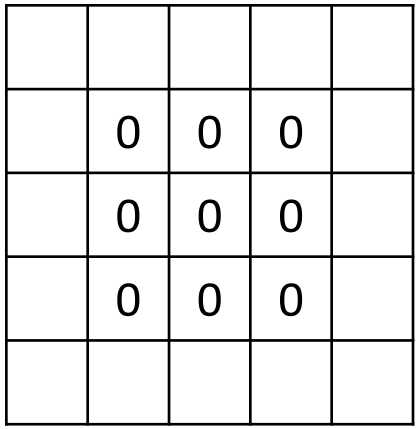
SIFT | 5.4

3단계: 극점 검출

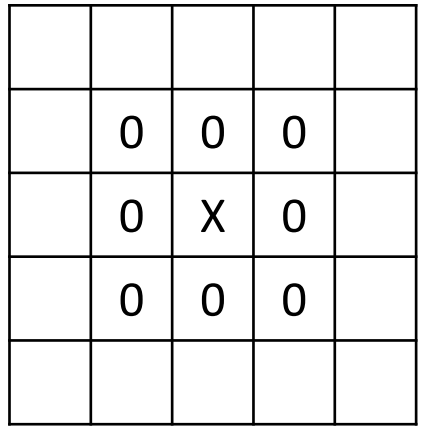
- X로 표시된 화소의 극점 여부를 조사하는데, 이웃 화소 26개보다 크면 극점으로 인정하여 특징점으로 취한다.
- SIFT는 특징점을 키포인트로 부른다.

- 검출된 특징점은 (y,x,o,i)로 표현한다.
- y,x: 화소의 위치를 나타냄
- O: 옥타브를 나타냄
- i: DOG 번호를 나타냄

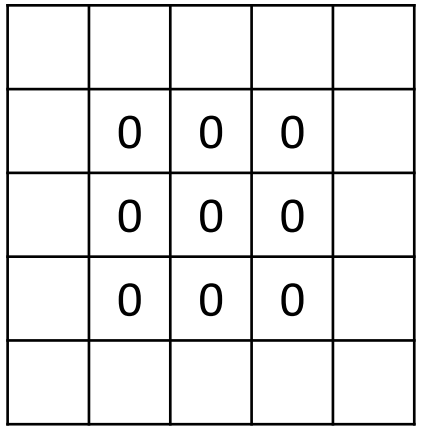
- 이 값들을 테일러 확장으로 미세 조정하여 최종 확정한다.



i-1번째 DOG



i번째 DOG



i+1번째 DOG

기술자

- 특징점 주위의 풍부한 정보
- 위치, 스케일 정보만으로는 물체 매칭에 부족하기 때문에 검출한다.

SIFT | 5.4 실행 결과

```
import cv2

img = cv2.imread('bus2.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

sift = cv2.SIFT_create()
kp, des = sift.detectAndCompute(gray,None)

gray = cv2.drawKeypoints(gray,kp,None,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv2.imshow('sift', gray)

k = cv2.waitKey()
cv2.destroyAllWindows()
```

매칭 |5.5

|A 영상(물체의 모델 영상)

- 깨끗한 배경에 물체가 홀로 있어 비교적 신뢰도가 높은 적은 수의 기술자가 검출된다.

|B 영상(물체와 배경이 섞인 영상

- 여러 물체가 혼재되어 기술자는 많고 잡음이 심하다.

|매칭

- 첫 번째 영상에서 추출한 기술자 집합A = $\{a_1, a_2, \dots a_m\}$과 두 번째 영상에서 추출한 기술자 B = $\{b_1, b_2, \dots b_n\}$ 집합으로 같은 물체의 같은 곳에 해당하는 쌍을 찾는 것이다.
- 가장 유사한 특징점을 찾아 쌍을 맺는 것
- 특징점이 상당히 많고 잡음이 섞인 기술자가 적지 않아 까다로움

- A, B를 조합한 mn개의 쌍 각각에 대해 거리를 계산해서 거리가 임곗값보다 작은 쌍을 모두 취하여 매칭 알고리즘을 만들 수 있으나 **거짓 긍정과 거짓 부정이 자주 발생**하며 m, n이 커서 시간이 오래 걸린다.

매칭 |5.5

|매칭

- 두 기술자 거리를 계산할 땐 보통 위 식으로 정의되는 유클리디안 거리를 사용한다.
- 이때 기술자는 d차원 벡터인데 SIFT 기술자는 d = 128이다.
- 두 기술자를 a = $\{a_1, a_2, \dots a_d\}$, b = $\{b_1, b_2, \dots b_d\}$로 표기한다.

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

매칭 | 5.5 세 가지 매칭 전략

고정 임곗값 방법

- 두 기술자 a_i, b_j의 거리가 임곗값보다 작으면 매칭되었다고 간주한다.
- 임곗값 T를 정하는 일이 매우 중요하다.

$$d(a_i - b_j) < T$$

최근접 이웃

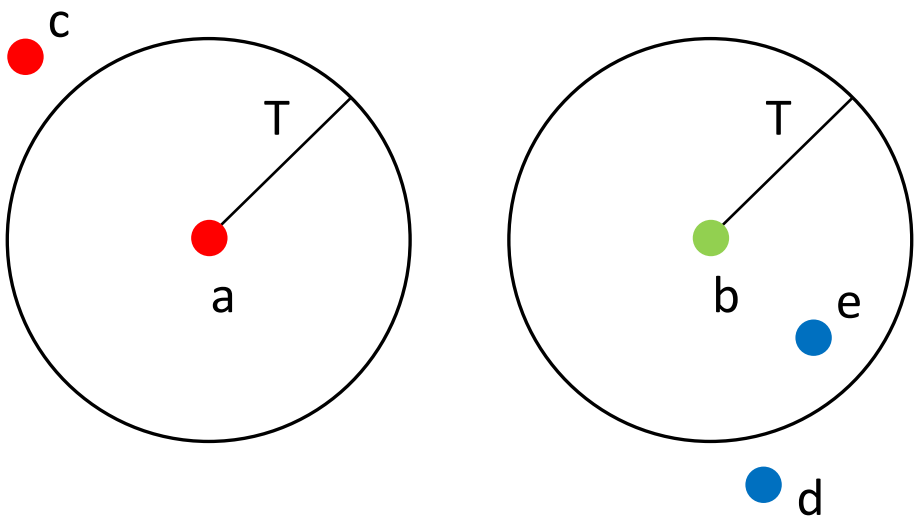
- a_i는 B에서 거리가 가장 작은 b_j를 찾고 a_i와 b_j가 옆 식을 만족하면 매칭 쌍으로 취한다.

최근접 이웃 거리 비율

- a_i는 B에서 가장 가까운 b_j와 2번째 가까운 b_k를 찾아 옆 식을 만족하면 a_i와 b_j는 쌍이 된다.

$$\frac{d(a_i - b_j)}{d(a_i - b_k)} < T$$

매칭 |5.5 세 가지 매칭 전략



|고정 임곗값 방법

- A와 c는 매칭되어야 하지만 인해 매칭이 일어나지 않아 거짓 부정이 되고 b와 e는 매칭되어 거짓 긍정이 된다.

|최근접 이웃 방법

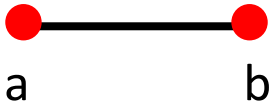
- A와 c는 옳게 쌍이 되어 참 긍정이 되고 b와 e는 거짓 긍정이 된다.

|최근접 이웃 거리 비율 방법

- A와 가장 가까운 두 이웃은 c,d인데 거리 비율이 작아 a와 c는 옳게 쌍이 되어 참 긍정이 되나, b는 두 최근접 이웃인 e와 d의 거리 비율이 커서 b와 e는 쌍이 안 되어 참 부정이 된다.

- 최근접 이웃 거리 비율 전략이 가장 성능이 좋으며 SIFT도 이를 사용한다.

매칭 | 5.5 정밀도와 재현율

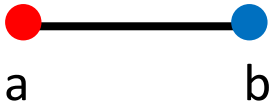


| 참 긍정

- 진짜 매칭 쌍을 긍정으로 예측함

| 거짓 부정

- 진짜 매칭 쌍을 부정으로 예측함



| 거짓 긍정

- 맺으면 안 되는데 맺음

| 참 부정

- 맺으면 안 되는데 맺지 않음

- 같은 색은 진짜 매칭 쌍이고 다른 색은 진짜 매칭 쌍이 아니다.

매칭 | 5.5 정밀도와 재현율

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

$$\text{정밀도} = \frac{TP}{TP+FP}$$

$$\text{재현율} = \frac{TP}{TP+FN}$$

$$\text{F1} = \frac{2 * \text{정밀도} * \text{재현율}}{\text{정밀도} + \text{재현율}}$$

| 혼동 행렬

- 매칭 알고리즘이 예측한 많은 수의 매칭 쌍에 대해 네 경우의 빈도를 세어 표에 채우면 혼동 행렬이 된다.
- 혼동 행렬로 다음 식의 성능 지표를 계산할 수 있다.

| 정밀도

- 매칭 알고리즘이 긍정, 즉 매칭 쌍으로 예측한 개수 중에 진짜 쌍인 비율

| 재현율

- 진짜 쌍 중에 알고리즘이 찾아낸 쌍의 비율

- F1은 정밀도와 재현율을 둘 다 고려하여 하나의 값으로 표현한다.

매칭 | 5.5 정밀도와 재현율

정확률

- 옳게 예측한 비율
- 특징점 매칭에선 부정이 긍정보다 월등히 많아 정확률의 의미가 없는 상황이 많다.
- 따라서 **주어진 상황에 딱 맞는** 성능 기준을 써야 한다.

$$\text{정확률} = \frac{TP + TN}{TP + TN + FP + FN}$$

매칭 |5.5

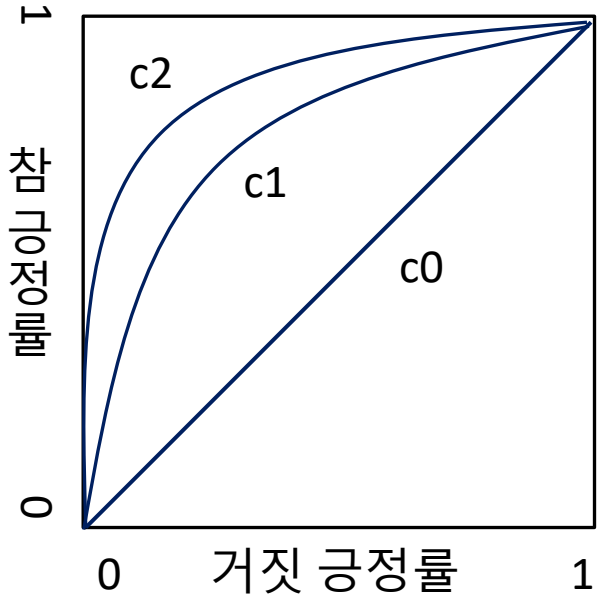
- 고정 임곗값 방법, 최근접 이웃 거리 비율 방법은 임곗값 T를 가지는데 T를 작게 하면 duv 식의 거짓 긍정률이 작아지고 T를 크게 하면 거짓 긍정률과 참 긍정률이 둘 다 커진다.

$$\text{참 긍정률} = \frac{TP}{TP+FN}$$

$$\text{거짓 긍정률} = \frac{FP}{TN+FP}$$

|ROC 곡선

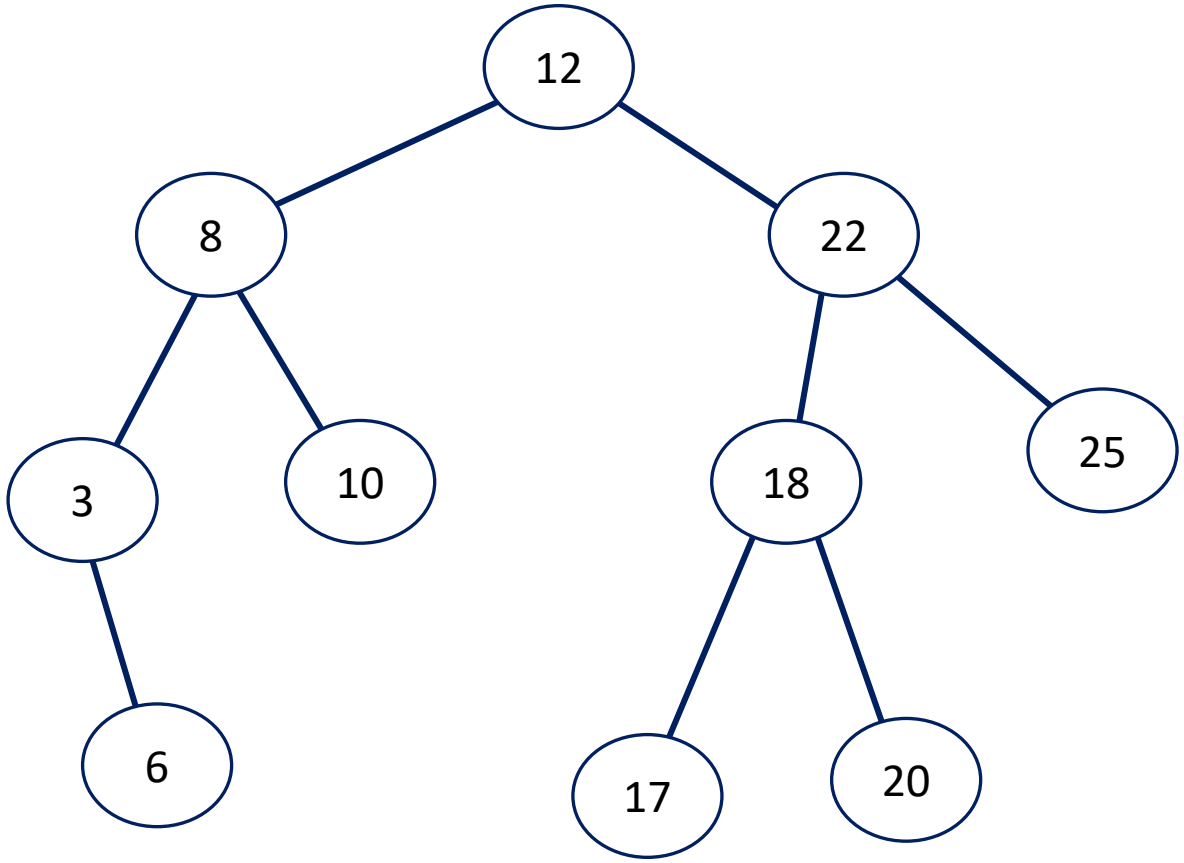
- 왼쪽 위 꼭지점에 가까운 c2의 성능이 가장 뛰어나다.
- 곡선이 **왼쪽 위 꼭지점**을 지난다면 오류를 전혀 범하지 않는 완벽한 매칭 알고리즘이다.



|AUC

- ROC 곡선의 아래 면적을 뜻한다.

매칭 | 5.5



| 이진 탐색 트리

- 이진 탐색 트리의 루트 노드는 12를 가진다.
- 왼쪽에 루트보다 작은 노드를, 오른쪽에 루트보다 큰 노드를 둔다.
- 이를 반복하여 질의어를 탐색한다.

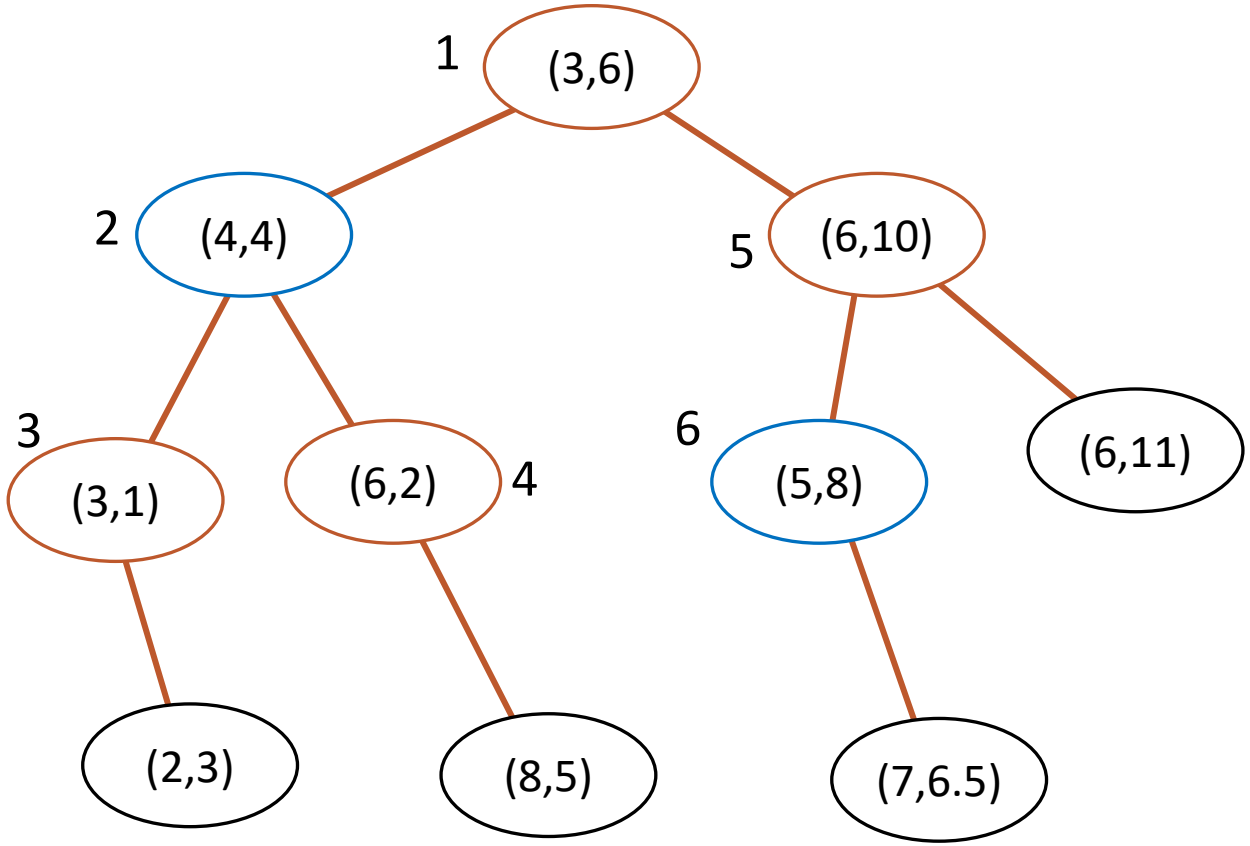
- 아주 빠른 탐색이 가능하다.

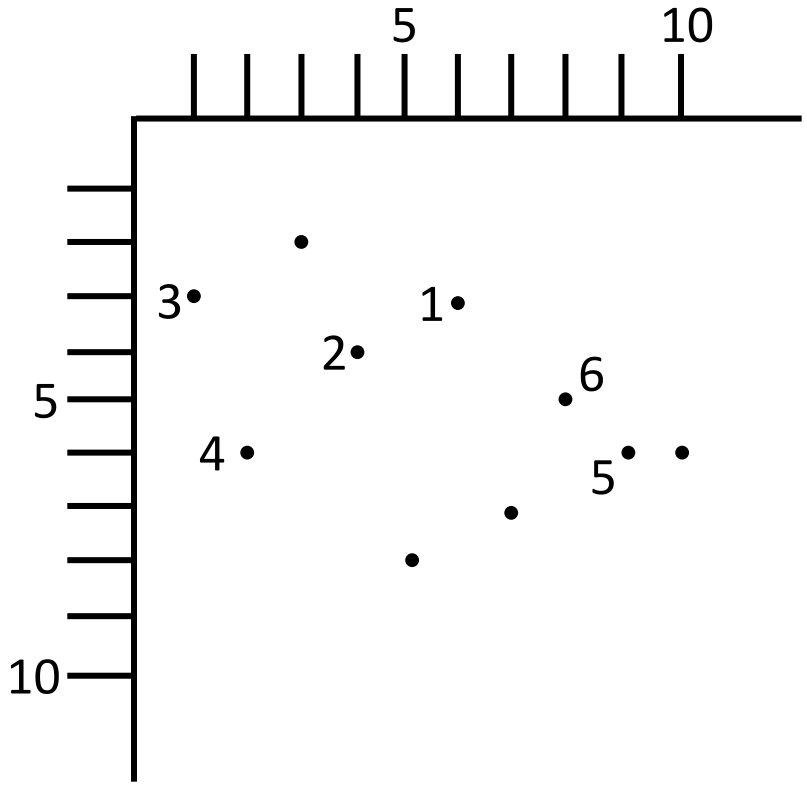
매칭 |5.5

|Kd 트리

- 특징점 매칭의 독특한 성질 때문에 이진 탐색 트리를 특징점 매칭에 그대로 적용할 수 없기 때문에 적합하게 개조한 것.
- $X = \{x_1 = (3,1), x_2 = (2,3), x_3 = (6,2), x_4 = (4,4), x_5 = (3,6), x_6 = (8,5), x_7 = (7,6.5), x_8 = (5,8), x_9 = (6,10), x_{10} = (6,11)$에서 두 축의 값을 각각 구하면 (3,2,6,…,6), (1,3,2,…,11)이 된다.
- 2번째 축의 분산이 더 크므로 그중 중앙값을 구해 분할 기준으로 x_5를 구한다.
- 빨간색은 2번 축, 파란색은 1번 축이다.

매칭 |5.5

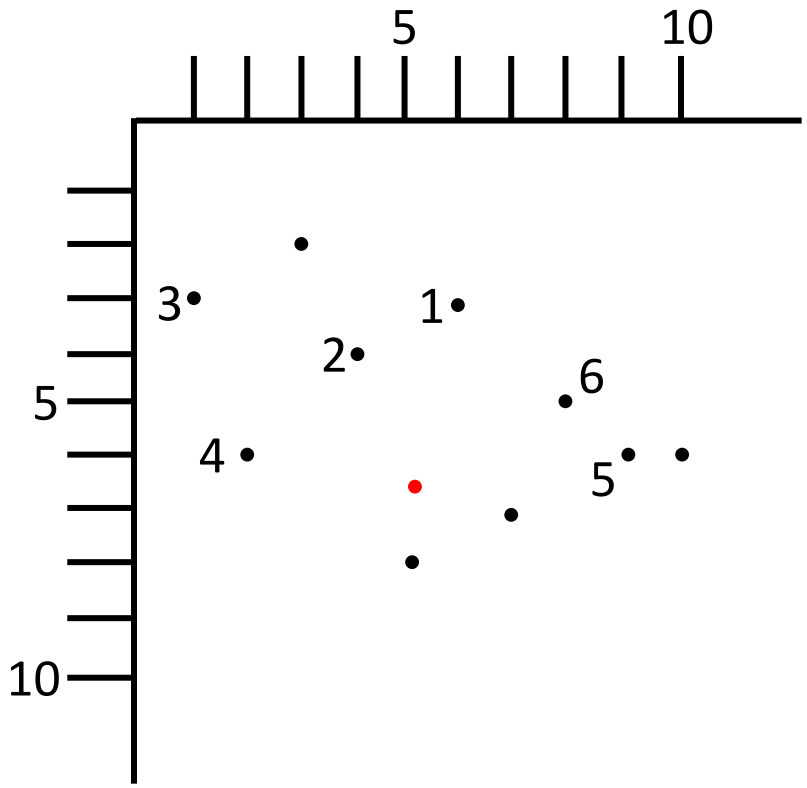




매칭 |5.5

- 새로운 벡터 x = (7,5.5)가 입력될 경우 값을 비교하며 (8,5)에 도달하게 된다.

- 이웃 칸에 더 가까운 점이 존재할 가능성이 있어 (8,5)를 최근접 이웃이라고 단언할 수 없고, 더 가까운 (7,6.5)가 있다.
- 따라서 타고 내려온 경로를 **역추적**하여 더 가까운 점을 찾는 추가 과정이 필요하다.



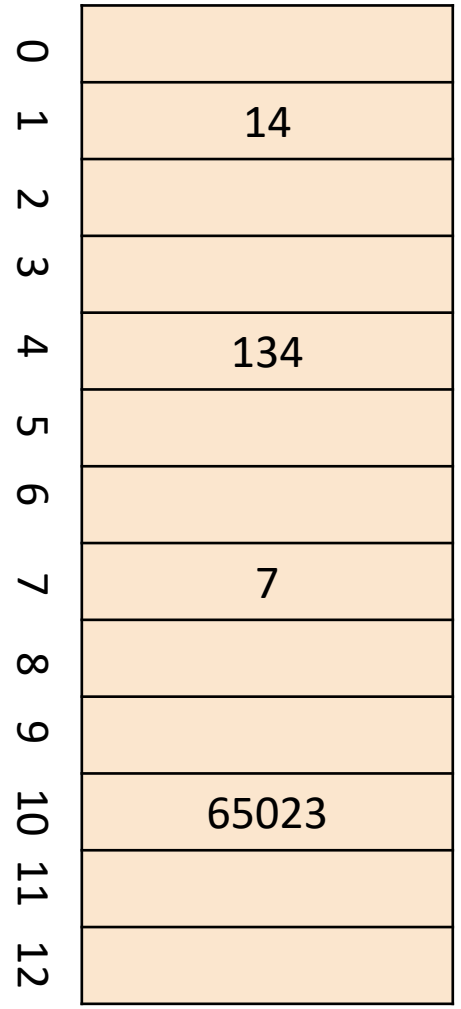
매칭 |5.5

|해싱

- 해시 함수 h가 키 값을 키가 저장될 칸의 번호로 매핑한다.

- **예)** 키 x가 134라면 h(134) = 134%13 = 4이므로 x를 4번 칸에 저장한다.

- 서로 다른 키가 같은 주소로 매핑되어 충돌이 일어날 수 있다.



매칭 |5.5

|FLANN 라이브러리

- Kd 트리의 다양한 변형을 구현한 라이브러리.
- SIFT를 고안한 로우 교수와 제자 무자가 공개함.

|FAISS 라이브러리

- 페이스북에서 공개한 최근접 이웃을 아주 빠르게 찾는 라이브러리.

매칭 | 5.5 실행 결과

```python
import cv2
import numpy as np
import time

img1 = cv2.imread('bus1.png')[250:400,550:700]
gray1 = cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
img2 = cv2.imread('bus2.png')
gray2 = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)

sift = cv2.SIFT_create()
kp1, des1 = sift.detectAndCompute(gray1,None)
kp2, des2 = sift.detectAndCompute(gray2,None)
print(f'특징점 개수: {len(kp1)}, {len(kp2)}')

start = time.time()
flann_matcher = cv2.DescriptorMatcher_create(cv2.DescriptorMatcher_FLANNBASED)
knn_match = flann_matcher.knnMatch(des1, des2, 2)

T = 0.7
good_match = []
```

```python
for nearest1, nearest2 in knn_match:
    if (nearest1.distance/nearest2.distance) <T:
        good_match.append(nearest1)
print(f'매칭에 걸린 시간: {time.time() - start}')

img_match = np.empty((max(img1.shape[0], img2.shape[0]), img1.shape[1] + img2.shape[1],3), dtype = np.uint8)
cv2.drawMatches(img1, kp1, img2, kp2, good_match, img_match, flags = cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

cv2.imshow('good matches', img_match)

k = cv2.waitKey()
cv2.destroyAllWindows()
```

```
특징점 개수: 286, 5989
매칭에 걸린 시간: 0.10975313186645508
```

매칭 |5.5 실행 결과

호모그래피 추정 | 5.6

투영 변환

- 3차원 점이 2차원 평면으로 변환되는 기하 관계
- 어파인 변환이 평행인 선분을 평행으로 유지하는 것과 다르게 먼 곳의 물체가 작게 나타나기 때문에 평행을 유지하지 못한다.

평면 호모그래피

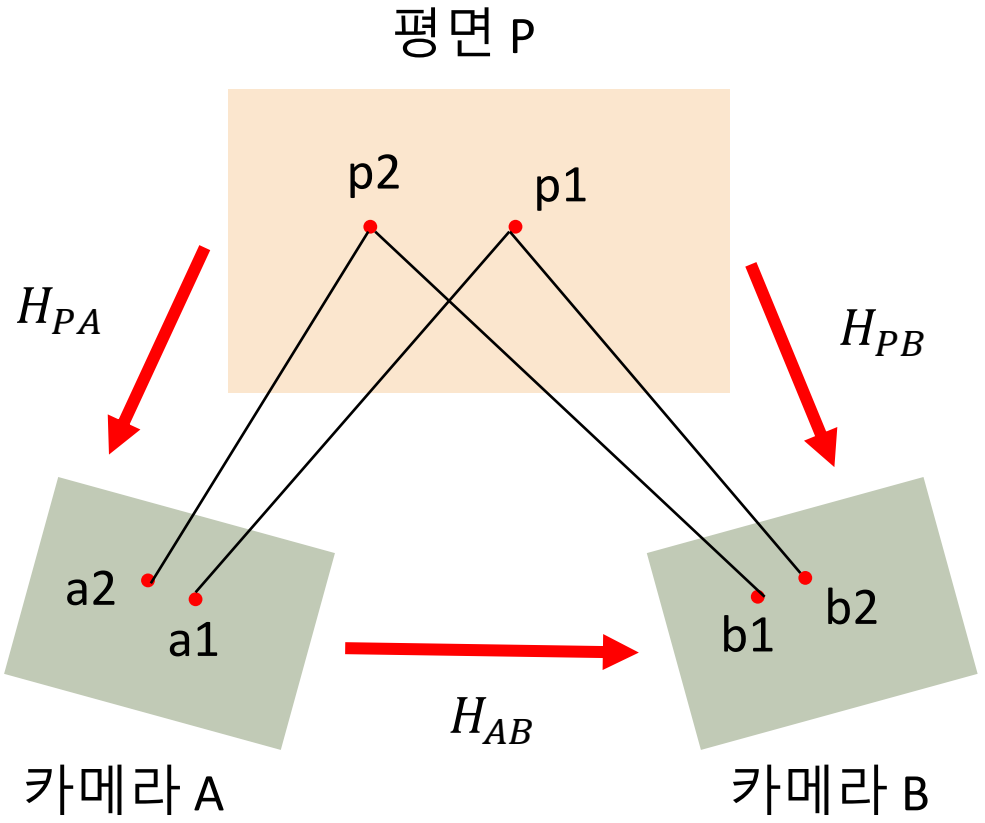
- 제한된 상황에서 이루어지는 투영 변환
- 줄여서 호모그래피라고 한다.

호모그래피 추정 | 5.6

| 투영 변환

- a와 b의 관계를 아래 식으로 표현할 수 있다.

$$b^T = \begin{pmatrix} b_x \\ b_y \\ 1 \end{pmatrix} = \begin{pmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{pmatrix} \begin{pmatrix} a_x \\ a_y \\ 1 \end{pmatrix} = Ha^T$$



| RANSAC

- 샘플에 섞여 있는 아웃라이어를 회피하면서 최적해를 구하는 일반 기법

호모그래피 추정 | 5.6 실행 결과

```python
import cv2
import numpy as np

img1 = cv2.imread('bus1.png')[250:400,550:700]
gray1 = cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
img2 = cv2.imread('bus2.png')
gray2 = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)

sift = cv2.SIFT_create()
kp1, des1 = sift.detectAndCompute(gray1,None)
kp2, des2 = sift.detectAndCompute(gray2,None)

flann_matcher = cv2.DescriptorMatcher_create(cv2.DescriptorMatcher_FLANNBASED)
knn_match = flann_matcher.knnMatch(des1, des2, 2)

T = 0.7
good_match = []

for nearest1, nearest2 in knn_match:
    if (nearest1.distance/nearest2.distance) < T:
        good_match.append(nearest1)

points1 = np.float32([kp1[gm.queryIdx].pt for gm in good_match])
points2 = np.float32([kp2[gm.trainIdx].pt for gm in good_match])
```

```python
H,_ = cv2.findHomography(points1, points2,cv2.RANSAC)

h1, w1 = img1.shape[0], img1.shape[1]
h2, w2 = img2.shape[0], img2.shape[1]

box1 = np.float32([[0,0],[0,h1-1],[w1-1,h1-1],[w1-1,0]]).reshape(4,1,2)
box2 = cv2.perspectiveTransform(box1, H)

img2 = cv2.polylines(img2, [np.int32(box2)], True, (0,255,0),8)

img_match = np.empty((max(h1,h2),w1+w2,3),dtype = np.uint8)
cv2.drawMatches(img1,kp1,img2,kp2,good_match, img_match,flags = cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

cv2.imshow('matches and homography', img_match)

k = cv2.waitKey()
cv2.destroyAllWindows()
```

호모그래피 추정 | 5.6 실행 결과