
8장. 영상 매칭과 추적

세부목차

1. **Matching with a similar image**
2. Feature and Key Point
3. Descriptor Extractor
4. Feature Matching
5. Tracking
6. Workshop

Matching with similar image

❖ **Average Hash Matching**

- 비슷한 그림 찾기 원리 이해 적합
- 평균 해쉬
 - 이미지를 평균 값으로 동일한 크기의 숫자 하나로 변환
 - 영상을 가로 세로 비율 무관하게 특정 크기로 축소
 - 픽셀 전체 평균 값 구해서 비교
 - $px > \text{평균} ? 1 : 0$
 - 1행으로 변환하면 동일한 크기의 2진수 숫자 하나로 변환
 - 16진수 등으로 변환 가능
- 유사도 거리
 - 두 값의 거리 측정해서 작으면 유사도 높음
 - 유클리드 거리(Euclidian distance)
 - 두 수의 차이를 거리로 계산
 - 예) $5:3 = 2, 5:8=3$
 - 해밍 거리(Hamming distance)
 - 두 수의 같은 자리 값이 서로 다른게 몇 개인지 나타내는 것
 - $12345:12354=2, 12345:92345=1$

Matching with similar image

❖ Average Hash Matching

```
import cv2
#영상 읽어서 그레이 스케일로 변환
img = cv2.imread('../img/pistol.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# 8x8 크기로 축소 ---①
gray = cv2.resize(gray, (16,16))
# 영상의 평균값 구하기 ---②
avg = gray.mean()
# 평균값을 기준으로 0과 1로 변환 ---③
bin = 1 * (gray > avg)
print(bin)
# 2진수 문자열을 16진수 문자열로 변환 ---④
dhash = []
for row in bin.tolist():
    s = ".join([str(i) for i in row])"
    dhash.append('%02x'%(int(s,2)))
dhash = ".join(dhash)
print(dhash)
cv2.namedWindow('pistol', cv2.WINDOW_GUI_NORMAL)
cv2.imshow('pistol', img)
cv2.waitKey(0)
```

[예제 8-2] 권총을 평균 해시로 변환(avg_hash .py)

Matching with similar image

❖ Average Hash Matching

g

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

fffff8000800080008000813fc1ffc1ffc07fc3ffc7ff87ff87ff87ffc7ff

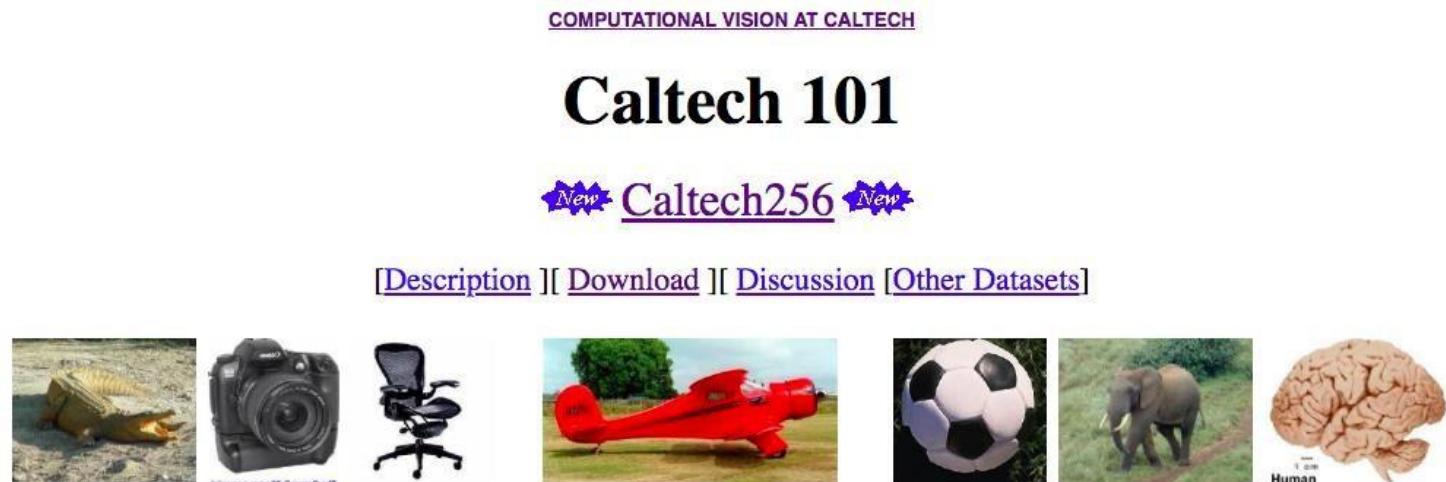


[그림 8-1] [예제 8-1]의 실행 결과

Matching with similar image

❖ Average Hash Matching

- 101 Image Dataset
 - 미국 캘리포니아 공대
 - 101 가지 물체를 담은 영상 이미지 셋
 - http://www.vision.caltech.edu/Image_Datasets/Caltech101/#Download



Description

Pictures of objects belonging to 101 categories. About 40 to 800 images per category. Most categories have about 50 images. Collected in September 2003 by Fei-Fei Li, Marco Andreetto, and Marc 'Aurelio Ranzato. The size of each image is roughly 300 x 200 pixels. We have carefully clicked outlines of each object in these pictures, these are included under the 'Annotations.tar'. There is also a matlab script to view the annotations 'show_annotations.m'

Matching with similar image

❖ Average Hash Matching

- 권총 이미지 찾기 <1/2>

```
import cv2
import numpy as np
import glob
# 영상 읽기 및 표시
img = cv2.imread('../img/pistol.jpg')
cv2.imshow('query', img)
# 비교할 영상들이 있는 경로 ---①
search_dir = '../img/101_ObjectCategories'
# 이미지를 16x16 크기의 평균 해쉬로 변환 ---②
def img2hash(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    gray = cv2.resize(gray, (16, 16))
    avg = gray.mean()
    bi = 1 * (gray > avg)
    return bi
# 해밍거리 측정 함수 ---③
def hamming_distance(a, b):
    a = a.reshape(1,-1)
    b = b.reshape(1,-1)
```

[예제 8-2] 사물 영상 중에서 권총 영상 찾기(avg_hash_matching.py)

Matching with similar image

❖ Average Hash Matching

- 권총 이미지 찾기 <2/2>

```
# 같은 자리의 값이 서로 다른 것들의 합  
distance = (a !=b).sum()  
return distance  
# 권총 영상의 해쉬 구하기 ---④  
query_hash = img2hash(img)  
# 이미지 데이터셋 딕토리의 모든 영상 파일 경로 ---⑤  
img_path = glob.glob(search_dir+'/**/*.{jpg}'  
for path in img_path:  
    # 데이터셋 영상 한개 읽어서 표시 ---⑥  
    img = cv2.imread(path)  
    cv2.imshow('searching...', img)  
    cv2.waitKey(5)  
    # 데이터셋 영상 한개의 해시 ---⑦  
    a_hash = img2hash(img)  
    # 해밍 거리 산출 ---⑧  
    dst = hamming_distance(query_hash, a_hash)  
    if dst/256 < 0.25: # 해밍거리 25% 이내만 출력 ---⑨  
        print(path, dst/256)  
        cv2.imshow(path, img)  
cv2.destroyAllWindows()  
cv2.waitKey(0)  
cv2.destroyAllWindows()
```

Matching with similar image

❖ Average Hash Matching

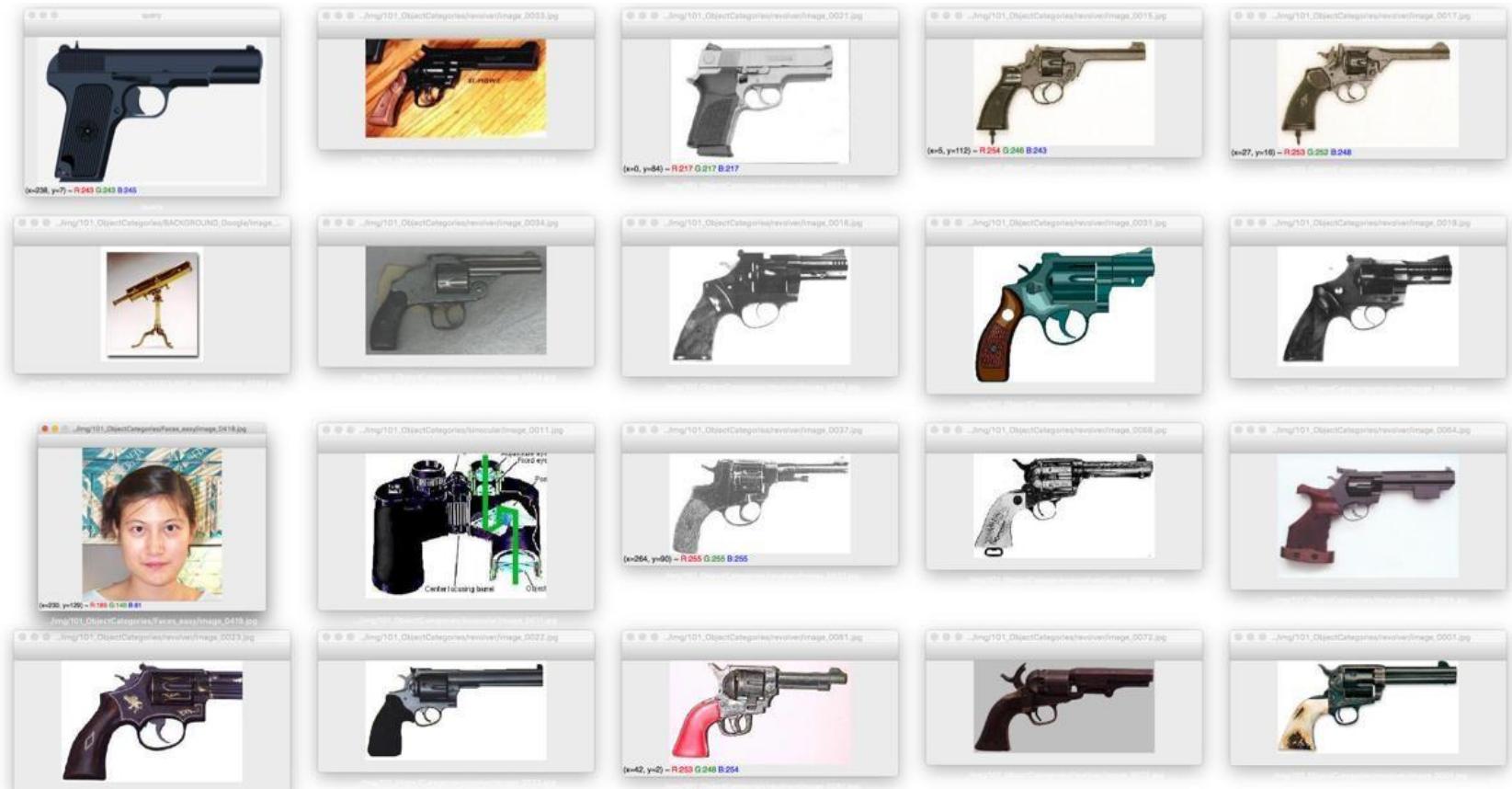
- 권총 이미지 찾기 <결과1>

```
./img/101_ObjectCategories/revolver/image_0033.jpg 0.2421875
./img/101_ObjectCategories/revolver/image_0019.jpg 0.23828125
./img/101_ObjectCategories/revolver/image_0031.jpg 0.21875
./img/101_ObjectCategories/revolver/image_0018.jpg 0.1953125
./img/101_ObjectCategories/revolver/image_0034.jpg 0.23046875
./img/101_ObjectCategories/revolver/image_0021.jpg 0.171875
./img/101_ObjectCategories/revolver/image_0037.jpg 0.2421875
./img/101_ObjectCategories/revolver/image_0023.jpg 0.21875
./img/101_ObjectCategories/revolver/image_0022.jpg 0.21484375
./img/101_ObjectCategories/revolver/image_0081.jpg 0.23046875
./img/101_ObjectCategories/revolver/image_0068.jpg 0.24609375
./img/101_ObjectCategories/revolver/image_0064.jpg 0.18359375
./img/101_ObjectCategories/revolver/image_0072.jpg 0.203125
./img/101_ObjectCategories/revolver/image_0001.jpg 0.2421875
./img/101_ObjectCategories/revolver/image_0015.jpg 0.24609375
./img/101_ObjectCategories/revolver/image_0017.jpg 0.23828125
./img/101_ObjectCategories/binocular/image_0011.jpg 0.23828125
./img/101_ObjectCategories/BACKGROUND_Google/image_0398.jpg 0.234375
./img/101_ObjectCategories/Faces_easy/image_0419.jpg 0.2421875
```

Matching with similar image

❖ Average Hash Matching

g

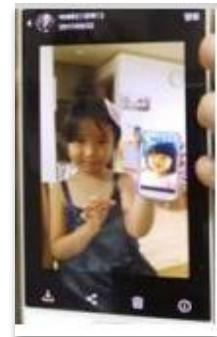


[그림 8-2] [예제 8-2]의 실행 결과

Matching with similar image

❖ **Template Matching**

- Object Detecting의 가장 기초적인 방법
- 찾고자 하는 이미지를 전체 이미지 검색
- cv2. matchTemplate(image, templ, method) : result
 - image : 입력영상
 - templ : 주적 대상
 - method : 매칭 알고리즘
 - cv2.TM_CCOEFF, cv2.TM_CCOEFF_NORMED
 - cv2.TM_CCORR, cv2.TM_CCORR_NORMED
 - cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED
 - 반환 : 매칭 비교 결과 값 배열,
 - TM_SQDIFF, TM_SQDIFF_NORMED 의 경우 min_loc
 - 나머지는 max_loc
- cv2.minMaxLoc(matched)
- 반환 : min_val, max_val, min_loc, max_loc



Matching with similar image

❖ Matching Method

(I : Image, T : Template, R : Result)

- method=CV_TM_SQDIFF

$$R(x, y) = \sum_{x', y'} (T(x', y') - I(x + x', y + y'))^2$$

- method=CV_TM_SQDIFF_NORMED

$$R(x, y) = \frac{\sum_{x', y'} (T(x', y') - I(x + x', y + y'))^2}{\sqrt{\sum_{x', y'} T(x', y')^2 \cdot \sum_{x', y'} I(x + x', y + y')^2}}$$

- method=CV_TM_CCORR

$$R(x, y) = \sum_{x', y'} (T(x', y') \cdot I(x + x', y + y'))$$

- method=CV_TM_CCORR_NORMED

$$R(x, y) = \frac{\sum_{x', y'} (T(x', y') \cdot I(x + x', y + y'))}{\sqrt{\sum_{x', y'} T(x', y')^2 \cdot \sum_{x', y'} I(x + x', y + y')^2}}$$

- method=CV_TM_CCOEFF

$$R(x, y) = \sum_{x', y'} (T'(x', y') \cdot I'(x + x', y + y'))$$

where

$$\begin{aligned} T'(x', y') &= T(x', y') - 1/(w \cdot h) \cdot \sum_{x'', y''} T(x'', y'') \\ I'(x + x', y + y') &= I(x + x', y + y') - 1/(w \cdot h) \cdot \sum_{x'', y''} I(x + x'', y + y'') \end{aligned}$$

- method=CV_TM_CCOEFF_NORMED

$$R(x, y) = \frac{\sum_{x', y'} (T'(x', y') \cdot I'(x + x', y + y'))}{\sqrt{\sum_{x', y'} T'(x', y')^2 \cdot \sum_{x', y'} I'(x + x', y + y')^2}}$$

Matching with similar image

❖ **Template Matching**

- Example <1/2>

```
import cv2
import numpy as np
# 입력이미지와 템플릿 이미지 읽기
img = cv2.imread('../img/figures.jpg')
template = cv2.imread('../img/taekwonv1.jpg')
th, tw = template.shape[:2]
cv2.imshow('template', template)
# 3가지 매칭 메서드 순회
methods = ['cv2.TM_CCOEFF_NORMED', 'cv2.TM_CCORR_NORMED', \
           'cv2.TM_SQDIFF_NORMED']
for i, method_name in enumerate(methods):
    img_draw = img.copy()
    method = eval(method_name)
    # 템플릿 매칭 ---①
    res = cv2.matchTemplate(img, template, method)
    # 최대, 최소값과 그 좌표 구하기 ---②
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
    print(method_name, min_val, max_val, min_loc, max_loc)
```

[예제 8-3] 템플릿 매칭으로 객체 위치 검출(template_matching .py)

Matching with similar image

❖ Template Matching

- Example <2/2>

```
# TM_SQDIFF의 경우 최소값이 좋은 매칭, 나머지는 그 반대 ---③
if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
    top_left = min_loc
    match_val = min_val
else:
    top_left = max_loc
    match_val = max_val
# 매칭 좌표 구해서 사각형 표시 ---④
bottom_right = (top_left[0] + tw, top_left[1] + th)
cv2.rectangle(img_draw, top_left, bottom_right, (0,0,255),2)
# 매칭 포인트 표시 ---⑤
cv2.putText(img_draw, str(match_val), top_left, \
            cv2.FONT_HERSHEY_PLAIN, 2,(0,255,0), 1, cv2.LINE_AA)
cv2.imshow(method_name, img_draw)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

출력 결과

```
cv2.TM_CCOEFF_NORMED-0.17781560122966766 0.5126562714576721 (42, 0) (208, 43)
cv2.TM_CCORR_NORMED 0.8271383047103882 0.9236084818840027 (85, 6) (208, 43)
cv2.TM_SQDIFF_NORMED 0.1706070452928543 0.36892083287239075 (208, 43) (86, 7)
```

Matching with similar image

❖ **Template Matching**

- Example <결과>



[그림 8-3] [예제 8-3]의 실행 결과

세부목차

1. Matching with a similar image

2. Feature and Key Point

3. Descriptor Extractor

4. Feature Matching

5. Tracking

6. Workshop

Feature and Key Point

❖ Corner Feature Detector

- Template Matching의 한계
 - Rotate, Scale, Brightness, Contrast, Hue, Affine 변환 대응 불가
- 이미지의 Corner :
 - 이미지의 특성 파악 용이
 - Corner 이외에는 인간도 구분하기 어려움
 - Image Feature(영상 특징) 검출

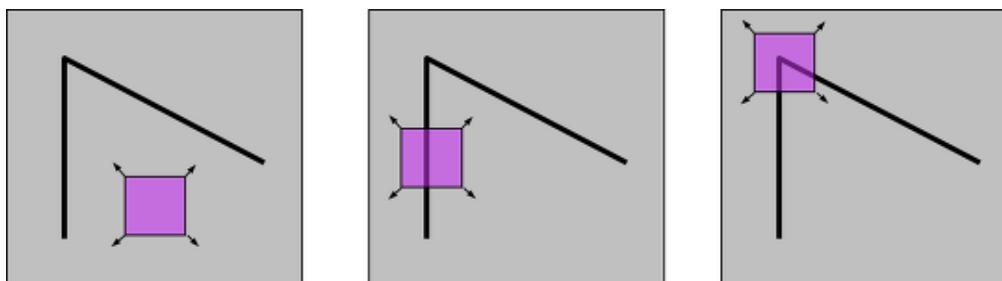


[그림 8-4] 코너 특징

Feature and Key Point

❖ Harris Corner Detection

- Chris harris, Mike Stephens의 논문, 1988
- 대표적인 Corner Keypoint 찾는 방법론
- kernel을 이동 했을때 모든 방향이 변화
- cv2.cornerHarris(img, blockSize, ksize, k)
 - img : 입력 영상, float32
 - blocksize : 이웃 픽셀 범위
 - ksize : Sobel에 사용할 kernel 크기
 - k : 코너검출 상수 (경험적 상수, 0.04~0.06)
 - 반환 : 입력영상과 동일한 크기
 - 변화량 값



[그림 8-5] 해리스 코너 검출

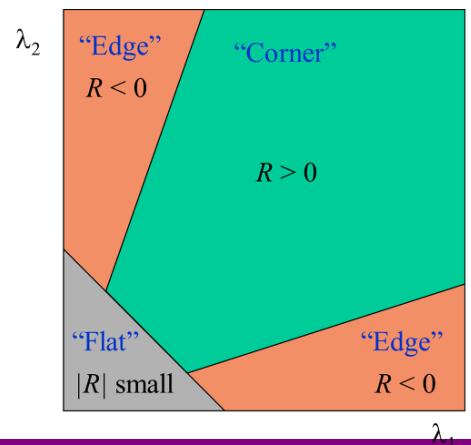
Structure tensor

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

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

$$R = \det(M) - k(\text{trace}(M))^2$$

코너응답함수



Feature and Key Point

❖ Harris Corner Detecti

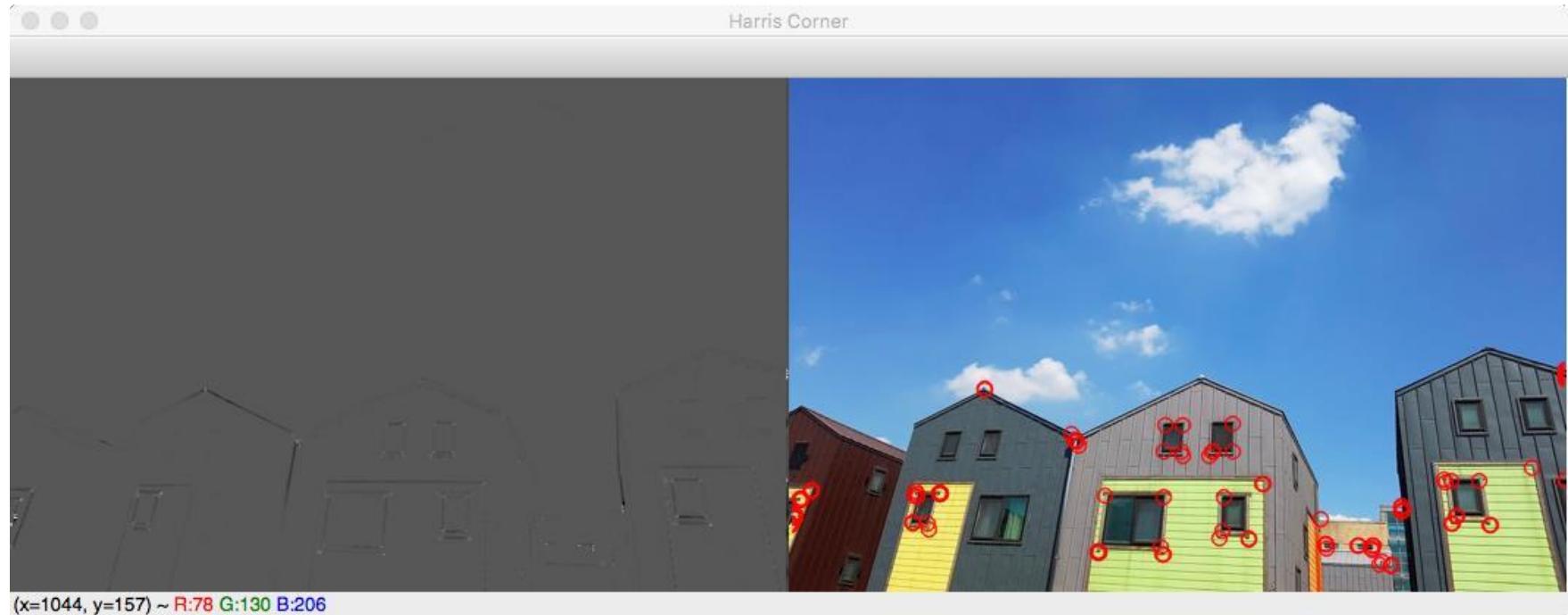
```
import cv2
import numpy as np
img = cv2.imread('../img/house.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# 해리스 코너 검출 ---①
corner = cv2.cornerHarris(gray, 2, 3, 0.04)
# 변화량 결과의 최대값 10% 이상의 좌표 구하기 ---②
coord = np.where(corner > 0.1* corner.max())
coord = np.stack((coord[1], coord[0]), axis=-1)
# 코너 좌표에 동그리미 그리기 ---③
for x, y in coord:
    cv2.circle(img, (x,y), 5, (0,0,255), 1, cv2.LINE_AA)
# 변화량을 영상으로 표현하기 위해서 0~255로 정규화 ---④
corner_norm = cv2.normalize(corner, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8U)
# 화면에 출력
corner_norm = cv2.cvtColor(corner_norm, cv2.COLOR_GRAY2BGR)
merged = np.hstack((corner_norm, img))
cv2.imshow('Harris Corner', merged)
cv2.waitKey()
cv2.destroyAllWindows()
```

[예제 8-4] 해리스 코너 검출(corner_harris.py)

Feature and Key Point

❖ Harris Corner Detection

- Example



[그림 8-6] [예제 8-4]의 실행 결과

Feature and Key Point

❖ Shi-Tomasi

- J.Shi, C. Tomasi 논문, 1994
- Optical Flow 등을 통해 추적에 사용할 특징점 추출 용도에 좋다
- Harris 방법에서 $\lambda_1\lambda_2$ 중에서 최소 값만 고려
- 결과 값에 Threshold 값 적용, 이 보다 크면 코너로 판단
- $R = \min(\lambda_1, \lambda_2) > T$
- cv2.goodFeaturesToTrack(img, maxCorner, qualityLev, minDistance)
 - img : 입력 영상
 - maxCorner : 얻고 싶은 코너 갯수, 강한것 순
 - qualityLev : 코너로 판단할 Threshold
 - minDistance : 코너간 최소 거리
 - mask : 검출에 제외할 마스크
 - blockSize=3 : 코너 주변 영역의 크기
 - useHarrisDetector=False : 코너 검출 방법 선택
 - True=해리스 코너 검출 방법, False=시와 토마시 검출 방법
 - k : 해리스 코너 검출 방법에 사용할 k 계수
 - corners : 코너 검출 좌표 결과, $N \times 1 \times 2$ 크기의 배열, 실수 값이기 때문에 정수로 변형 필요

Feature and Key Point

❖ Shi-Tomasi

- Example

```
import cv2
import numpy as np
img = cv2.imread('../img/house.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# 시-토마스의 코너 검출 메서드
corners = cv2.goodFeaturesToTrack(gray, 80, 0.01, 10)
# 실수 좌표를 정수 좌표로 변환
corners = np.int32(corners)
# 좌표에 동그라미 표시
for corner in corners:
    x, y = corner[0]
    cv2.circle(img, (x, y), 5, (0,0,255), 1, cv2.LINE_AA)
cv2.imshow('Corners', img)
cv2.waitKey()
cv2.destroyAllWindows()
```

[예제 8-5] 시-토마시 코너 검출(corner_goodFeature.py)

Feature and Key Point

❖ Shi-Tomasi

- Example <결과>

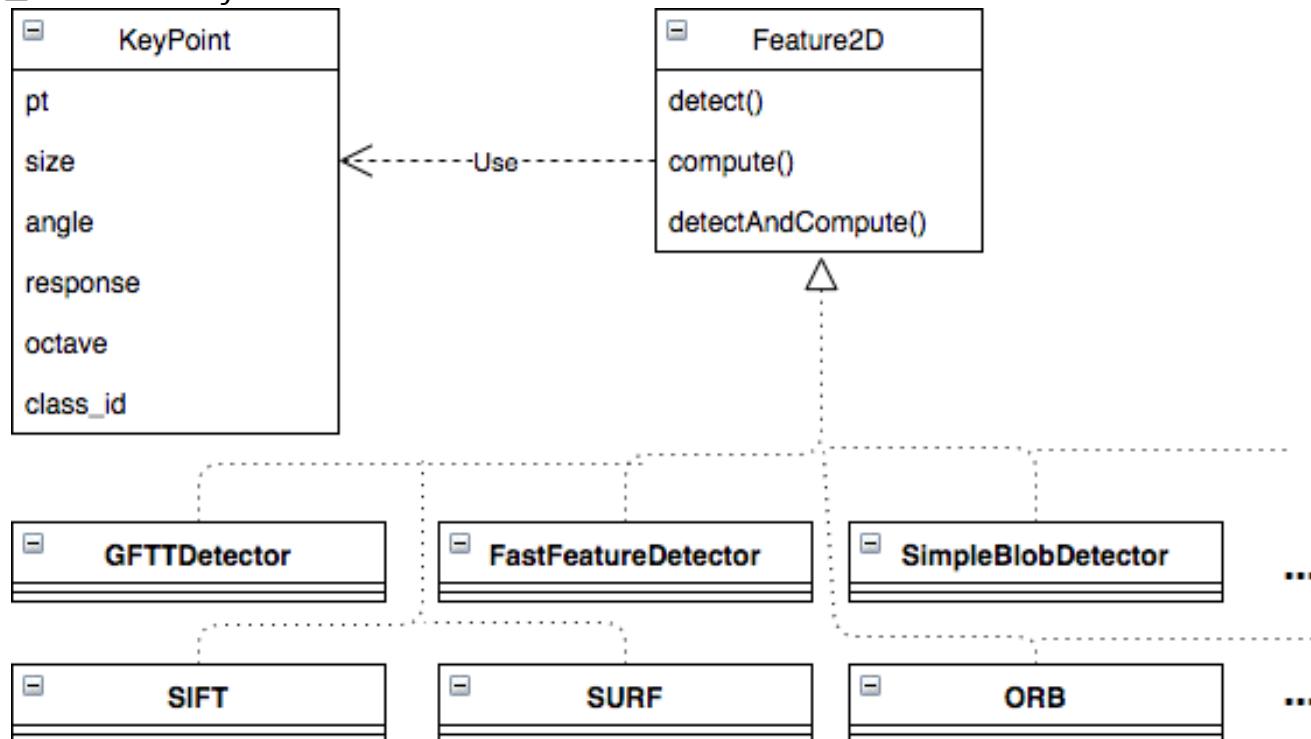


[그림 8-7] [예제 8-5]의 실행 결과

Feature and Key Point

❖ Key Point and Feature Detector

- 특징점 검출 알고리즘에 무관하게 통일된 인터페이스 제공
- 특징 검출기 : cv2.Feature2D 상속 받아 구현, 12가지(3.4.1버전 기준)
- 특징점 : cv2.KeyPoint



[그림 8-8] 특징 검출기 인터페이스 클래스 다이어그램

Feature and Key Point

❖ cv2.Feature2D

- keypoints = detector.detect(img [, mask]) : 키 포인트 검출 함수
 - img : 입력영상, 바이너리 스케일
 - mask : 검출 제외 마스크
 - keypoints : 특징점 검출 결과, KeyPoint의 리스트
- KeyPoint : 특징점 정보를 담는 객체
 - pt : 키 포인트 (x, y)좌표, Float 타입으로 정수로 변환 필요
 - size : 의미 있는 키포인트 이웃의 반지름
 - angle : 특징점 방향(시계방향, -1=의미 없음)
 - response : 특징점 반응 강도(추출기에 따라 다름)
 - octave : 발견된 이미지 피라미드 계층
 - class_id : 키 포인트가 속한 객체 ID

Feature and Key Point

❖ Key Point 표시

- outImg = cv2.drawKeypoints(img, keypoints, outImg[, color[, flags]])
img : 입력 이미지
- keypoints : 표시할 키 포인트 리스트
- outImg : 키 포인트가 그려진 결과 이미지
- color : 표시할 색상, (기본값 : 랜덤)
- flags : 표시 방법 선택 플래그
 - cv2.DRAW_MATCHES_FLAGS_DEFAULT : 좌표 중심에 동그라미만 그림, 기본 값
 - cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS : 동그라미의 크기를 size와 angle을 반영해서 그림

Feature and Key Point

❖ GFTTDetector

- cv2.goodFeaturesToTrack() 함수로 구현된 검출기
- 검출기 생성은 cv2.GFTTDetector_create 함수로 하되 검출하는데 사용하는 함수는 cv2.Feature2D의 detect() 함수와 동일
- detector = cv2.GFTTDetector_create([, maxCorners[, qualityLevel, minDistance, blockSize, useHarrisDetector, k]])
 - 인자의 모든 내용은 cv2.goodFeaturesToTrack()과 동일
 - KeyPoint객체의 pt 속성의 좌표 이외에 값 없음

Feature and Key Point

❖ GFTTDetector

```
import cv2
import numpy as np
img = cv2.imread("../img/house.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Good feature to track 검출기 생성 ---①
gftt = cv2.GFTTDetector_create()
# 키 포인트 검출 ---②
keypoints = gftt.detect(gray, None)
# 키 포인트 그리기 ---③
img_draw = cv2.drawKeypoints(img, keypoints, None) #openCV version 3.4.1
"openCV version 4.0
img2 = img.copy()
for marker in keypoints:
    img2 = cv2.drawMarker(img2, tuple(int(i) for i in marker.pt), color=(0, 255, 0))
cv2.imshow('GFTTDectector', img2)
cv2.imshow('GFTTDectector', img_draw)
# 결과 출력 ---④
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[예제 8-6] GFTTDetector로 키 포인트 검출(kpt_gftt.py)

Feature and Key Point

직선 그리기

OpenCV 함수 중에서 drawMarker() 함수는 직선 그리기 함수를 이용하여 다양한 모양의 마커(marker)를 그림

```
void drawMarker(InputOutputArray img, Point position, const Scalar& color,  
                int markerType = MARKER_CROSS, int markerSize=20, int thickness=1,  
                int line_type=8);
```

- `img` 입출력 영상
- `position` 마커 출력 위치
- `color` 선 색상
- `markerType` 마커 종류. `MarkerTypes` 열거형 상수 중 하나를 지정합니다.
- `markerSize` 마커 크기
- `thickness` 선 두께
- `line_type` 선 타입. `LINE_4`, `LINE_8`, `LINE_AA` 중 하나를 지정합니다.

Feature and Key Point

MarkerTypes 열거형 상수	설명
MARKER_CROSS	십자가 모양(+) 모양)
MARKER_TILTED_CROSS	45도 회전된 십자가 모양(× 모양)
MARKER_STAR	MARKER_CROSS 모양과 MARKER_TILTED_CROSS 모양이 합쳐진 형태(*) 모양)
MARKER_DIAMOND	마름모 모양(◇ 모양)
MARKER_SQUARE	정사각형 모양(□ 모양)
MARKER_TRIANGLE_UP	위로 뾰족한 삼각형(△ 모양)
MARKER_TRIANGLE_DOWN	아래로 뾰족한 삼각형(▽ 모양)

Feature and Key Point

❖ GFTTDetector



[그림 8-9] [예제 8-6]의 실행 결과

OpenCV 버전 다운하기

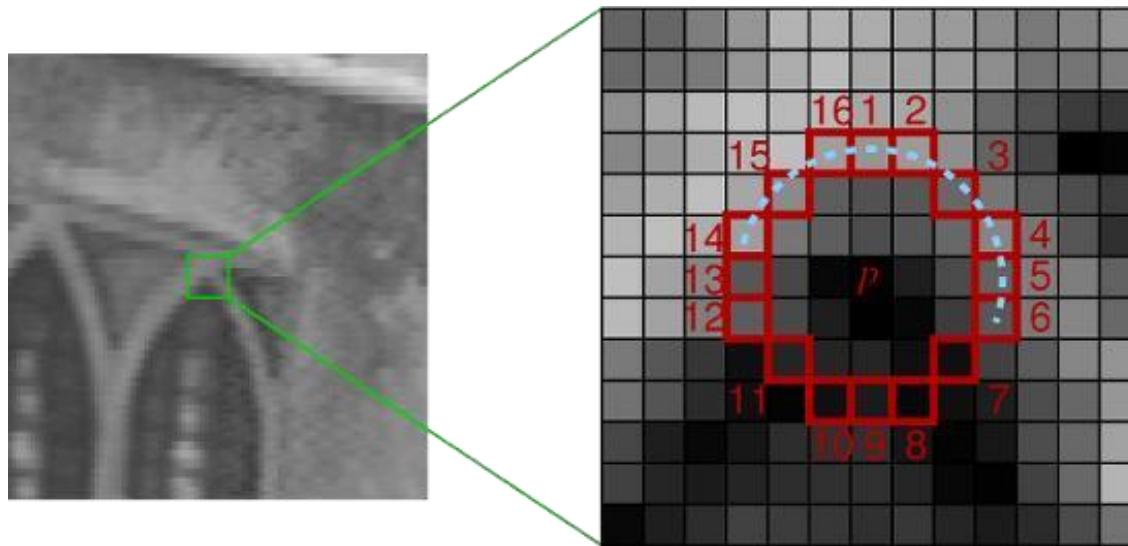
- openCV 3.4.2.6 이상에서는 SIFT와 SURF를 지원하지 않는다.



Feature and Key Point

❖ FAST(Feature from Accelerated Segment Test)

- Edward Rosten, Tom Drummon, Cambrige Univ, 2006
- 실시간 가능하게 속도 개선
 - 점 p를 중심으로 반지름 3인 원을 통과하는 16픽셀 확인
 - p보다 일정값(t) 이상 밝거나 어두운것이 n개 연속이면 코너로 판다
 - n : FAST-9, 10, 11, 12, 13, 14, 15, 16
- Descriptor는 없이 KeyPoint만 검출



[그림 8-10] FAST 알고리즘

Feature and Key Point

❖ **FAST(Feature from Accelerated Segment Test)**

- `detector = cv.FastFeatureDetector_create([threshold[, nonmaxSuppression, type]])`
 - threshold=10 : 코너 판단 임계 값
 - nonmaxSuppression = True : 최대 점수가 아닌 코너 억제
 - type : 엣지 검출 패턴
 - `cv2.FastFeatureDetector_TYPE_9_16` : 16개 중 9개 연속(기본 값)
 - `cv2.FastFeatureDetector_TYPE_7_12` : 12개 중 7개 연속
 - `cv2.FastFeatureDetector_TYPE_5_8` : 8개 중 5개 연속

Feature and Key Point

❖ **FAST(Feature from Accelerated Segment Test)**

```
import cv2
import numpy as np
img = cv2.imread('..../img/house.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# FAST 특징 검출기 생성 ---①
fast = cv2.FastFeatureDetector_create(50)
# 키 포인트 검출 ---②
keypoints = fast.detect(gray, None)
# 키 포인트 그리기 ---③
img = cv2.drawKeypoints(img, keypoints, None)
```

결과 출력 ---④

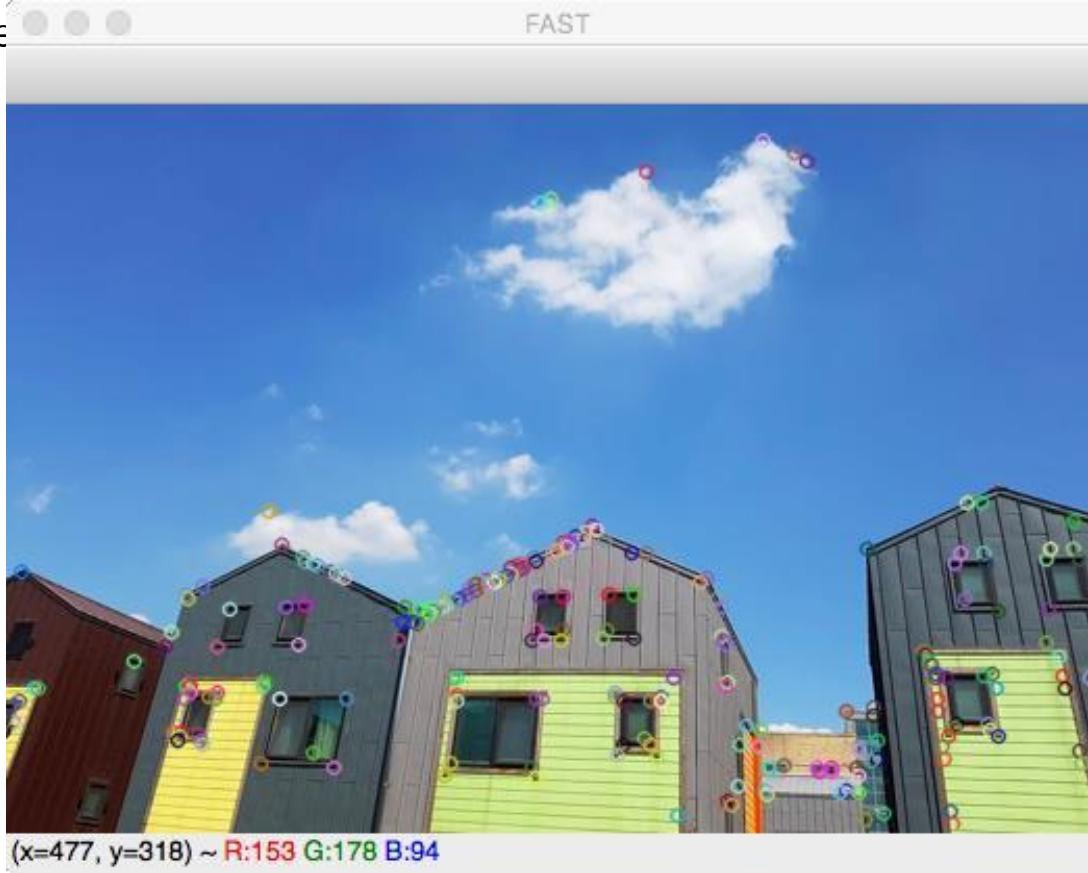
```
cv2.imshow('FAST', img)
cv2.waitKey()
cv2.destroyAllWindows()
```

[예제 8-7] FAST로 키 포인트 검출(kpt_fast.py)

Feature and Key Point

❖ FAST(Feature from Accelerated Segment Test)

- Example



[그림 8-11] [예제 8-7]의 실행 결과

Feature and Key Point

❖ **SimpleBlobDetector**

- BLOB(Binary Large Object)
 - 바이너리 스케일 이미지의 연결된 픽셀 그룹
 - 작은 객체는 노이즈로 판단
 - 특정 크기 이상의 객체에만 관심
- detector = cv2.SimpleBlobDetector_create([parameters]) : Blob 검출 기 생성자
 - parameters : Blob 검출 필터 인자 객체

Feature and Key Point

❖ SimpleBlobDetector Example

```
import cv2
import numpy as np

img = cv2.imread("../img/house.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# SimpleBlobDetector 생성 ---①
detector = cv2.SimpleBlobDetector_create()
# 키 포인트 검출 ---②
keypoints = detector.detect(gray)
# 키 포인트를 빨간색으로 표시 ---③

img = cv2.drawKeypoints(img, keypoints, None, (0,0,255), \
                       flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv2.imshow("Blob", img)
cv2.waitKey(0)
```

[예제 8-8] SimpleBlobDetector 검출기 | (kpt_blob.py)

Feature and Key Point

❖ SimpleBlobDetector Example



[그림 8-12] [예제 8-8]의 실행 결과

Feature and Key Point

❖ **SimpleBlobDetector**

- cv2.SimpleBlobDetector_Params()
 - minThreshold, maxThreshold, thresholdStep : Blob를 생성하기 위한 경계 값
 - minThreshold에서 maxThreshold를 넘지 않을 때 까지 thresholdStep 만큼 증가
 - minRepeatability : Blob에 참여하기 위한 연속된 경계값의 갯수
 - minDistBetweenBlobs : 두 blob를 하나의 blob로 간주한 거리
 - filterByArea : 면적 필터 옵션
 - minArea, maxArea : min~max 범위의 면적만 blob로 검출
 - filterByCircularity : 원형비 필터 옵션
 - minCircularity, maxCircularity : min ~ max 범위의 원형비율만 blob로 검출
 - filterByColor : 밝기를 이용한 필터 옵션
 - blobColor : 0=검정색 blob 검출, 255= 흰색 blob 검출
 - filterByConvexity : 볼록비율 필터 옵션
 - minConvexity, maxConvexity : min ~ max 범위의 볼록 비율만 Blob로 검출
 - filterByInertia : 관성비율 필터 옵션
 - minInertiaRatio, maxInertiaRatio: min ~ max 범위의 관성 비율만 Blob로 검출

Feature and Key Point

❖ **SimpleBlobDetector with Option Example <1/2**

```
import cv2
import numpy as np

img = cv2.imread("../img/house.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# blob 검출 필터 파라미터 생성 ---①
params = cv2.SimpleBlobDetector_Params()

# 경계값 조정 ---②
params.minThreshold = 10
params.maxThreshold = 240
params.thresholdStep = 5
# 면적 필터 켜고 최소 값 지정 ---③
params.filterByArea = True
params.minArea = 200
```

[예제 8-9] 필터 옵션으로 생성한 SimpleBlobDetector 검출기 (kpt_blob_param .py)

Feature and Key Point

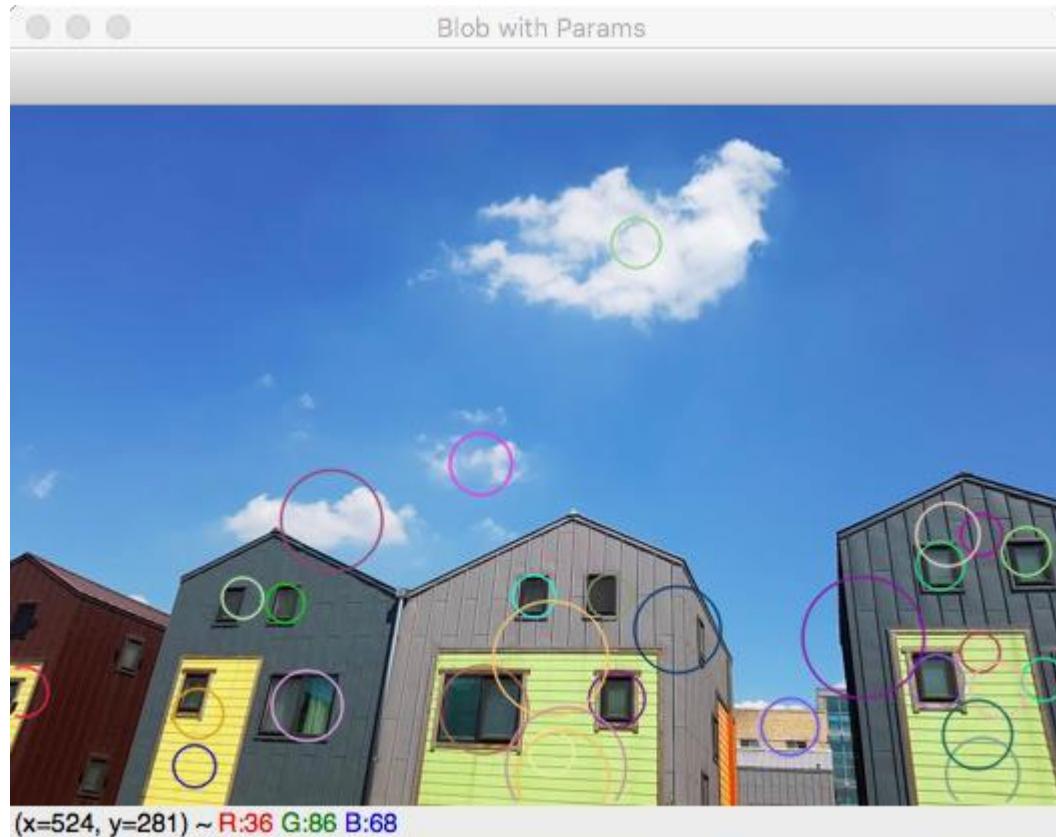
❖ SimpleBlobDetector with Option Example <2/2

```
# 컬러, 볼록 비율, 원형비율 필터 옵션 끄기 ---④
params.filterByColor = False
params.filterByConvexity = False
params.filterByInertia = False
params.filterByCircularity = False

# 필터 파라미터로 blob 검출기 생성 ---⑤
detector = cv2.SimpleBlobDetector_create(params)
# 키 포인트 검출 ---⑥
keypoints = detector.detect(gray)
# 키 포인트 그리기 ---⑦
img_draw = cv2.drawKeypoints(img, keypoints, None, None,\n                           cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
# 결과 출력 ---⑧
cv2.imshow("Blob with Params", img_draw)
cv2.waitKey(0)
```

Feature and Key Point

❖ **SimpleBlobDetector with option example**



[그림 8-13] [예제 8-9]의 실행 결과

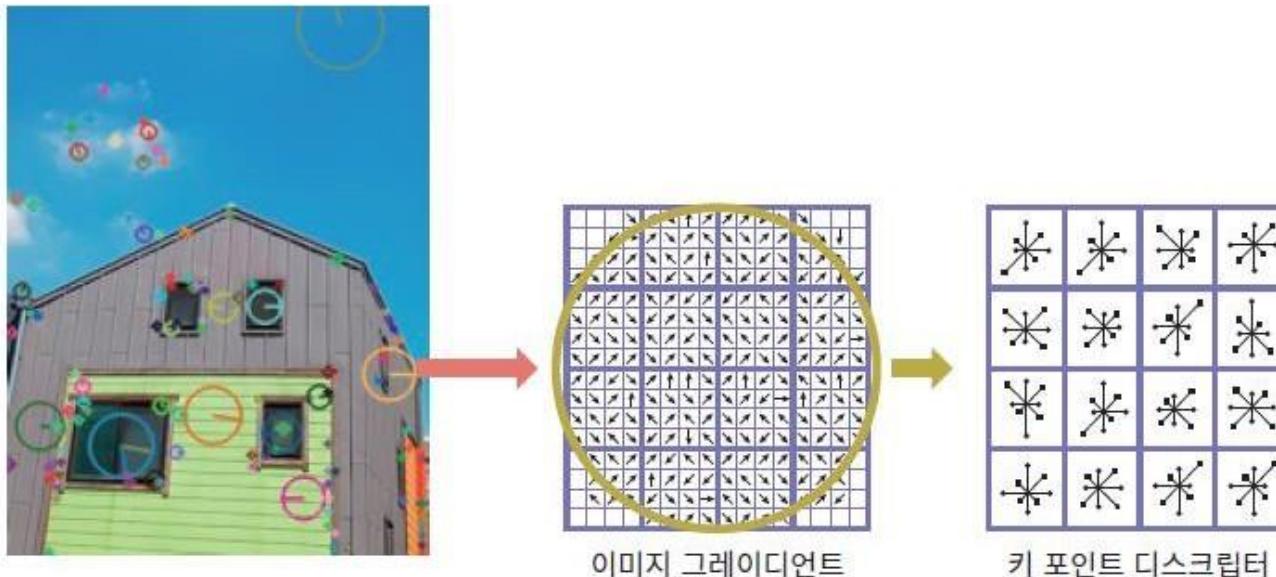
세부목차

1. Matching with a similar image
2. Feature and Key Point
- 3. Descriptor Extractor**
4. Feature Matching
5. Tracking
6. Workshop

Descriptor Extractor

❖ Feature Descriptor

- 회전, 크기, 방향에 영향이 없는 특징 서술자 요구
- Key Point 주변 픽셀을 일정한 크기의 블럭으로 나누어
- 블럭에 속한 픽셀의 그레이디언트 히스토그램을 계산한 것
- 일반적으로 8방향 경사도를 표현
 - 키 포인트 당 $4 \times 4 \times 8 = 128$ 개의 값으로 구성



[그림 8-14] 키 포인트와 특징 디스크립터

Descriptor Extractor

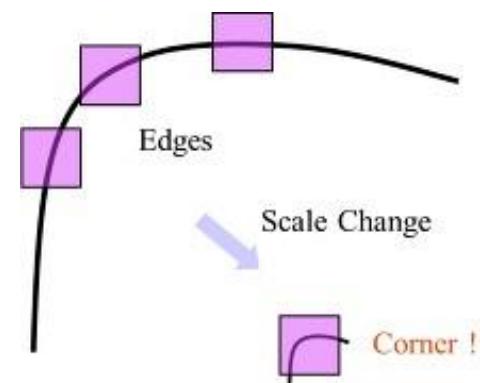
❖ **Descriptor Extractor Interface**

- cv2.Feature2D 추상 클래스
- keypoints, descriptors = detector.compute(image, keypoints[, descriptors]): 키 포인트 전달하면 특징 서술자를 계산해서 반환
- keypoints, descriptors = detector.detectAndCompute(image, mask[, descriptors, useProvidedKeypoints]): 키 포인트 검출과 특징 서술자 계산을 한번에 수행
 - image : 입력 영상
 - keypoints : 서술자 계산을 위해 사용할 키 포인트
 - descriptors : 계산된 서술자
 - mask : 키 포인트 검출에 사용할 마스크
 - useProvidedKeypoints : True 인 경우 키 포인트 검출을 수행하지 않음(사용 안함)

Descriptor Extractor

❖ SIFT (Scale-Invariant Feature Transform))

- D.Lowe, Univ of British Columbia, 2004
- Corner Matching Issue
 - Intensity : 밝기, 이동, 회전, 어핀 강인성
 - Scaling : 크기를 축소하면 더 많은 코너 검출
- Image Pyramid를 이용해서 Scale 이슈 해결
 - Laplacian 값이 극대 되는 점을 특징점으로 찾는다.
- 특히 때문에 학교와 연구 용으로만 사용가능
- OpenCV3.0 패키지 부터 기본 패키지 제외
 - Extra(contrib)패키지에 포함



Descriptor Extractor

❖ SIFT (Scale-Invariant Feature Transform)

- 객체 생성
- `detector = cv.xfeatures2d.SIFT_create([, nfeatures[, nOctaveLayers[, contrastThreshold[, edgeThreshold[, sigma]]]]])`
 - nfeatures : 검출 최대 특징 수
 - nOctaveLayers : 이미지 피라미드에 사용할 계층 수
 - contrastThreshold : 필터링할 빈약한 특징 문턱 값
 - edgeThreshold : 필터링할 엣지 문턱 값
 - sigma : 이미지 피라미드 0 계층에서 사용 할 가우시안 필터에 사용한 시그마 값

Descriptor Extractor

❖ SIFT (Scale-Invariant Feature Transform)

- Example

```
import cv2
import numpy as np
img = cv2.imread('../img/house.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# SIFT 추출기 생성
sift = cv2.xfeatures2d.SIFT_create()
# 키 포인트 검출과 서술자 계산
keypoints, descriptor = sift.detectAndCompute(gray, None)
print('keypoint:', len(keypoints), 'descriptor:', descriptor.shape)
print(descriptor)
# 키 포인트 그리기
img_draw = cv2.drawKeypoints(img, keypoints, None, \
                             flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
# 결과 출력
cv2.imshow('SIFT', img_draw)
cv2.waitKey()
cv2.destroyAllWindows()
```

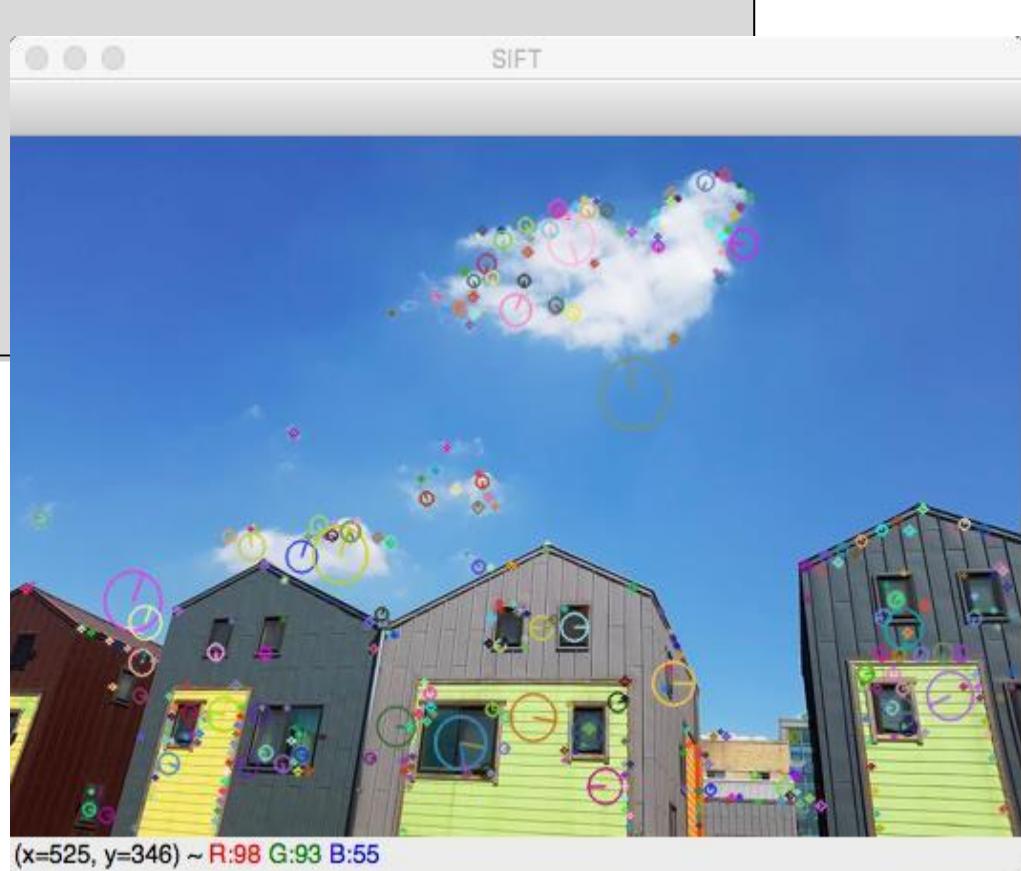
[예제 8-10] SIFT로 키 포인트 및 디스카립터 추출(desc_sift.py)

Descriptor Extractor

❖ SIFT (Scale-Invariant Feature Transform)

- Example <결과>

```
[[ 1.  1.  1. ...  0.  0.  1.]  
 [ 8. 24.  0. ...  1.  0.  4.]  
 [ 0.  0.  0. ...  0.  0.  2.]  
 ...  
 [ 1.  8.  71. ...  73. 127.  3.]  
 [ 35.  2.  7. ...  0.  0.  9.]  
 [ 36.  34.  3. ...  0.  0.  1.]]
```



[그림 8-15] [예제 8-10]의 실행 결과

Descriptor Extractor

❖ SURF (Speeded-Up Robust Feature)

- SIFT에 대한 성능 개선
 - Image Pyramid 사용 대신 필터의 크기를 바꾸는 방식
- 특히 때문에 학교와 연구 용으로만 사용 가능
- OpenCV3.0 패키지 부터 기본 패키지 제외, Extra(contrib)패키지에 포함

- `detector = cv.xfeatures2d.SURF_create([hessianThreshold, nOctaves, nOctaveLayers, extended, upright])`
 - `hessianThreshold` : 특징 추출 경계 값(100)
 - `nOctaves` : 이미지 피라미드 계층 수(3)
 - `extended` : 서술자 생성 플래그(False), True : 128개, False : 64개
 - `upright` : 방향 계산 플래그(False), True : 방향 무시, False: 방향 적용

Descriptor Extractor

❖ SURF (Speeded-Up Robust Feature)

- Example

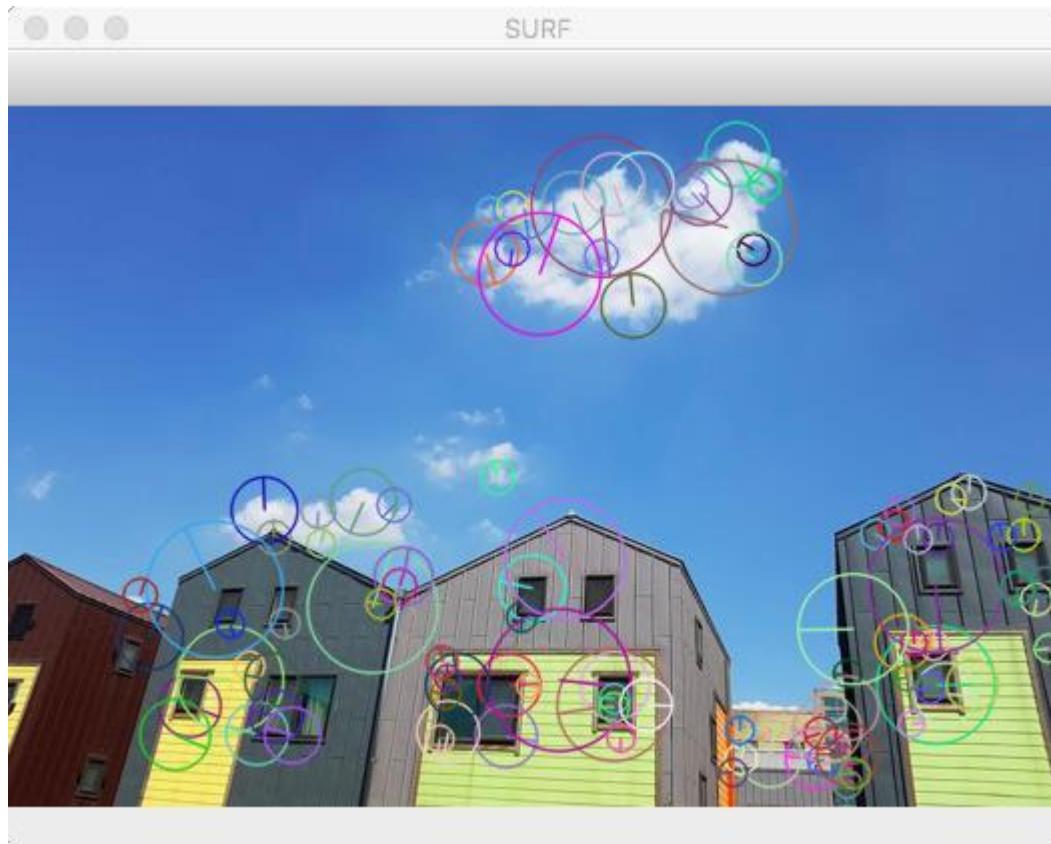
```
import cv2
import numpy as np
img = cv2.imread('../img/house.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# SURF 추출기 생성 ( 경계:1000, 피라미드:3, 서술자확장:True, 방향적용:True)
surf = cv2.xfeatures2d.SURF_create(1000, 3, True, True)
# 키 포인트 검출 및 서술자 계산
keypoints, desc = surf.detectAndCompute(gray, None)
print(desc.shape, desc)
# 키포인트 이미지에 그리기
img_draw = cv2.drawKeypoints(img, keypoints, None, \
                             flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv2.imshow('SURF', img_draw)
cv2.waitKey()
cv2.destroyAllWindows()
```

[예제 8-11] SURF로 키 포인트 및 디스크립터 추출(desc_surf.py)

Descriptor Extractor

❖ SURF (Speeded-Up Robust Feature)

- Example <결과>



[그림 8-16] [예제 8-11]의 실행 결과

BRIEF, ORB

❖ **ORB(Oriented FAST and Rotated BRIEF)**

- Rublee, Rabaud, Konolige, Bradski, 2011
- FAST로 특징점 검출
- BRIEF에 방향과 회전을 고려하도록 개선
 - SIFT/SURF/BRIEF 대안
- detector = cv.ORB_create([nfeatures, scaleFactor, nlevels, edgeThreshold, firstLevel, WTA_K, scoreType, patchSize, fastThreshold])
 - nfeatures=500 : 검출 할 최대 특징 수
 - scaleFactor = 1.2 : 이미지 피라미드 비율
 - nlevels = 8 : 이미지 피라미드 계층 수
 - edgeThreshold = 31 : 검색 제외 테두리 크기, patchSize와 마찬가지
 - firstLevel = 0 : 최초 이미지 피라미드 계층 단계
 - WTA_K = 2 : 임의 좌표 생성 수
 - scoreType : 키 포인트 검출에 사용 할 방식
 - cv2.ORB_HARRIS_SCORE : 해리스 코너 검출(기본 값)
 - cv2.ORB_FAST_SCORE : FAST 코너 검출
 - patchSize = 31 : 서술자의 패치 크기
 - fastThreshold = 20 : FAST에 사용할 임계 값

BRIEF, ORB

❖ **ORB(Oriented FAST and Rotated BRIEF)**

```
import cv2
import numpy as np
img = cv2.imread('../img/house.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# ORB 추출기 생성
orb = cv2.ORB_create()
# 키 포인트 검출과 서술자 계산
keypoints, descriptor = orb.detectAndCompute(img, None)
# 키 포인트 그리기
img_draw = cv2.drawKeypoints(img, keypoints, None, \
                             flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
# 결과 출력
cv2.imshow('ORB', img_draw)
cv2.waitKey()
cv2.destroyAllWindows()
```

[예제 8-12] ORB로 키 포인트 및 디스크립터 추출(desc_orb.py)

BRIEF, ORB

❖ ORB(Oriented FAST and Rotated BRIEF)

- Example <결과>



[그림 8-17] [예제 8-12]의 실행 결과

세부목차

1. Matching with a similar image
2. Feature and Key Point
3. Descriptor Extractor
- 4. Feature Matching**
5. Tracking
6. Workshop

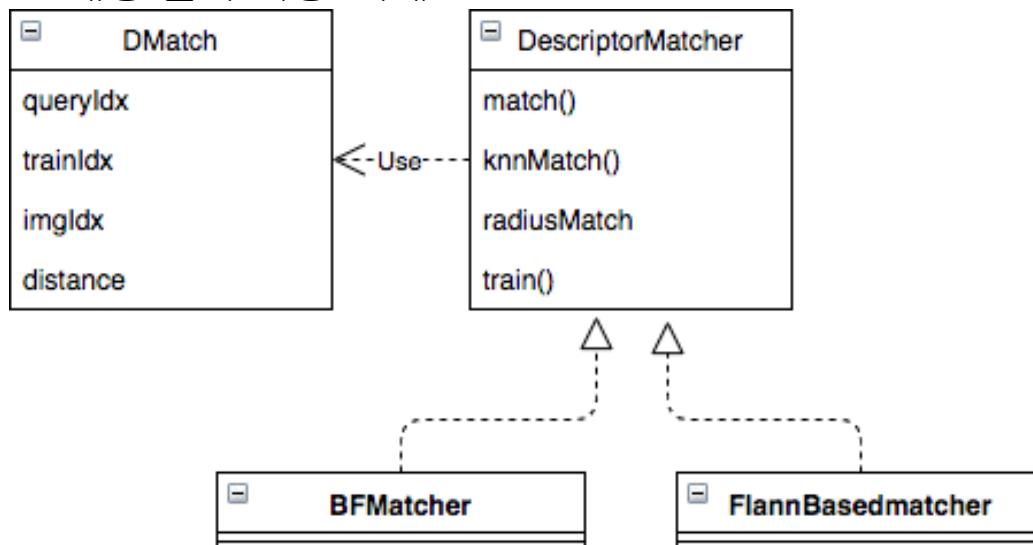
Feature Matching

❖ Matching

- 서로 다른 이미지의 KeyPoint와 Descriptor를 비교하여 비슷한 것끼리 짹짓는 것
- 매칭점의 갯수에 따라 영상의 유사도 측정, 영역 검출
- 이미지 검색, 파노라마 사진생성, 객체 인식 등에 활용

❖ 공통 인터페이스

- cv2.DescriptorMatcher : 매칭 추상 클래스 상속
- cv2.DMatch : 매칭 결과 저장 객체



[그림 8-18] 특징 매칭 인터페이스 클래스 다이어그램

Feature Matching

❖ 공통 인터페이스

- matcher = cv2.DescriptorMatcher_create(matcherType) : 매칭기 생성자
 - matcherType : 생성할 구현 클래스의 알고리즘, 문자열
 - "BruteForce" : NORM_L2를 사용하는 BFMatcher
 - "BruteForce-L1" : NORM_L1을 사용하는 BFMatcher
 - "BruteForce-Hamming" : NORM_HAMMING을 사용하는 BFMatcher
 - "BruteForce-Hamming(2)" : NORM_HAMMING2 를 사용하는 BFMatcher
 - "FlannBased" : NORM_L2를 사용하느 FlannBasedMatcher
- matches = matcher.match(queryDescriptors, trainDescriptors[, mask]): 1개의 최적 매칭
 - queryDescriptors : 특징 서술자 배열, 매칭의 기준이 될 서술자
 - trainDescriptors : 특징 서술자 배열, 매칭의 대상이 될 서술자
 - mask : 매칭 진행 여부 마스크
 - matches : 매칭 결과, DMatch 객체의 리스트

Feature Matching

❖ 공통 인터페이스

- matches = matcher.knnMatch(queryDescriptors, trainDescriptors, k[, mask[, compactResult]]] : k개의 가장 근접한 매칭
 - k : 매칭 할 근접 이웃 갯수
 - compactResult=False : True : 매칭이 없는 경우 매칭 결과에 불 포함
- matches = matcher.radiusMatch(queryDescriptors, trainDescriptors, maxDistance[, mask, compactResult]) : maxDistance 이내의 거리 매칭
 - maxDistance : 매칭 대상 거리
- DMatch
 - queryIdx : queryDescriptor의 인덱스
 - trainIdx : trainDescriptor의 인덱스
 - imgIdx : trainDescriptor의 이미지 인덱스
 - distance : 유사도 거리

Feature Matching

❖ 공통 인터페이스

- cv2.drawMatches(img1, kp1, img2, kp2, matches, flags): 매칭점을 영상에 표시
img1, kp1 : queryDescriptor의 영상과 키 포인트
- img1, kp1 : trainDescriptor의 영상과 키 포인트
- matches : 매칭 결과
- flags : 매칭점 그리기 옵션
 - cv2.DRAW_MATCHES_FLAGS_DEFAULT : 결과 이미지 새로 생성(기본 값)
 - cv2.DRAW_MATCHES_FLAGS_DRAW_OVER_OUTIMG : 결과 이미지 새로 생성 안함
 - cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS : 키 포인트 크기와 방향도 그리기
 - cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS : 한쪽만 있는 매칭 결과 그리기 제외

Feature Matching

❖ BF(Brute-Force) Matcher

- 두 이미지의 Feature Descriptor를 전수 조사
- matcher = cv.BFMatcher_create([normType[, crossCheck]])
 - normType : 거리 측정 알고리즘
 - cv2.NORM_L1
 - cv2.NORM_L2 : 기본 값
 - cv2.NORM_L2SQR
 - cv2.NORM_HAMMING
 - cv2.NORM_HAMMING2
 - crossCheck=False : 상호 매칭이 있는 것만 반영
- 특정 기술자에 따른 normType 선택 방법
 - SIFT SURF : NORM_L1, NORM_L2
 - ORB : NORM_HAMMING
 - ORB , WTA_K : NORM_HAMMING2

Feature Matching

❖ BF(Brute-Force) Matching

- SIFT-BF Example

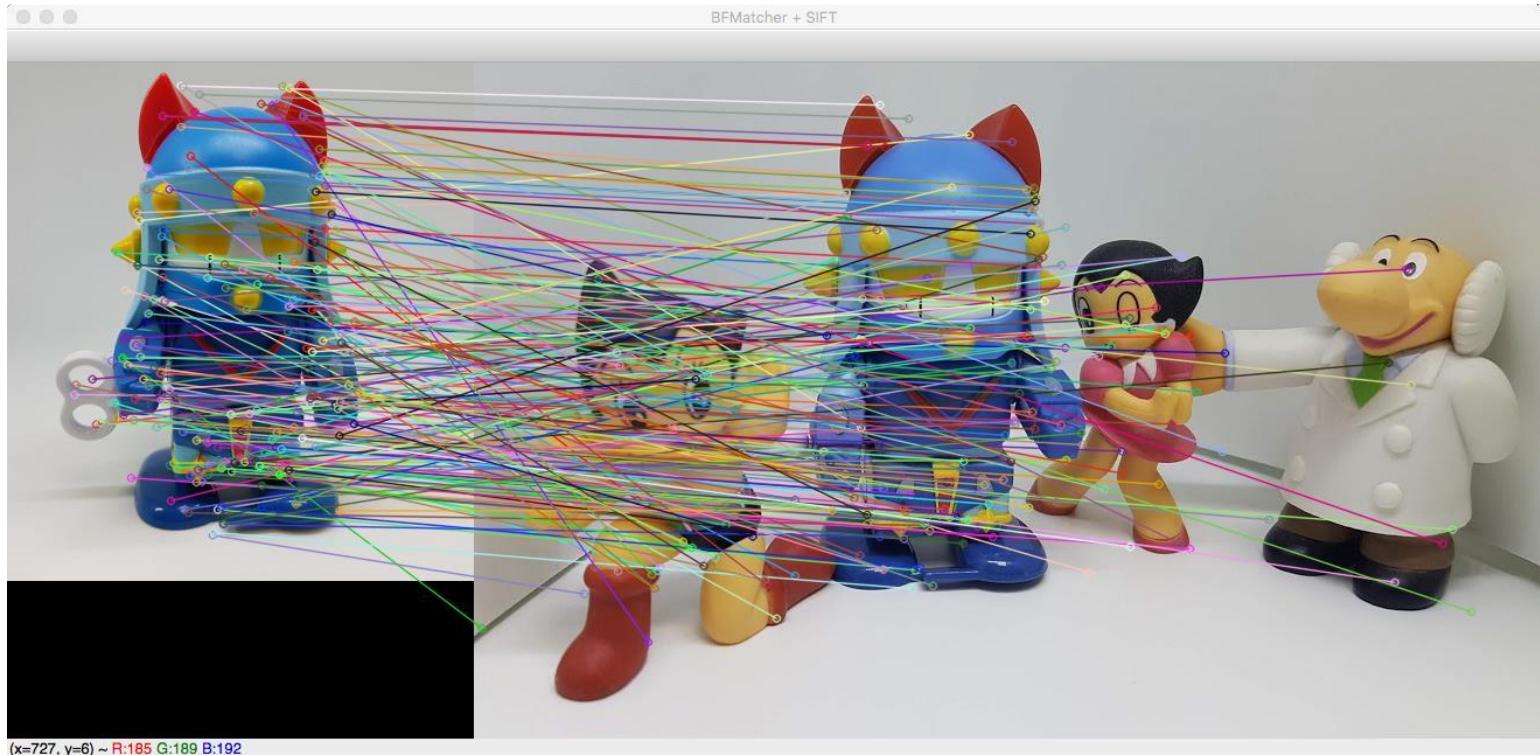
```
import cv2, numpy as np
img1 = cv2.imread('../img/taekwonv1.jpg')
img2 = cv2.imread('../img/figures.jpg')
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
# SIFT 서술자 추출기 생성 ---①
detector = cv2.xfeatures2d.SIFT_create()
# 각 영상에 대해 키 포인트와 서술자 추출 ---②
kp1, desc1 = detector.detectAndCompute(gray1, None)
kp2, desc2 = detector.detectAndCompute(gray2, None)
# BFMatcher 생성, L1 거리, 상호 체크 ---③
matcher = cv2.BFMatcher(cv2.NORM_L1, crossCheck=True)
# 매칭 계산 ---④
matches = matcher.match(desc1, desc2)
# 매칭 결과 그리기 ---⑤
res = cv2.drawMatches(img1, kp1, img2, kp2, matches, None, \
                      flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)
# 결과 출력
cv2.imshow('BFMatcher + SIFT', res)
cv2.waitKey()
cv2.destroyAllWindows()
```

[예제 8-13] BFMatcher와 SIFT로 매칭(match_bf_sift.py)

Feature Matching

❖ BF(Brute-Force) Matching

- SIFT-BF Example <결과>



[그림 8-19] [예제 8-13]의 실행 결과

Feature Matching

❖ BF(Brute-Force) Matching

- SURF– BF Example

```
import cv2
import numpy as np
img1 = cv2.imread('../img/taekwonv1.jpg')
img2 = cv2.imread('../img/figures.jpg')
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
# SURF 서술자 추출기 생성 ---①
detector = cv2.xfeatures2d.SURF_create()
kp1, desc1 = detector.detectAndCompute(gray1, None)
kp2, desc2 = detector.detectAndCompute(gray2, None)
# BFMatcher 생성, L2 거리, 상호 체크 ---③
matcher = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
# 매칭 계산 ---④
matches = matcher.match(desc1, desc2)
# 매칭 결과 그리기 ---⑤
res = cv2.drawMatches(img1, kp1, img2, kp2, matches, None, \
                      flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)
cv2.imshow('BF + SURF', res)
cv2.waitKey()
cv2.destroyAllWindows()
```

[예제 8-14] BFMatcher와 SURF로 매칭(match_bf_surf.py)

Feature Matching

❖ BF(Brute-Force) Matching

- SURF- BF Example <결과>



[그림 8-2] [예제 8-14]의 실행 결과

Feature Matching

❖ BF(Brute-Force) Matching

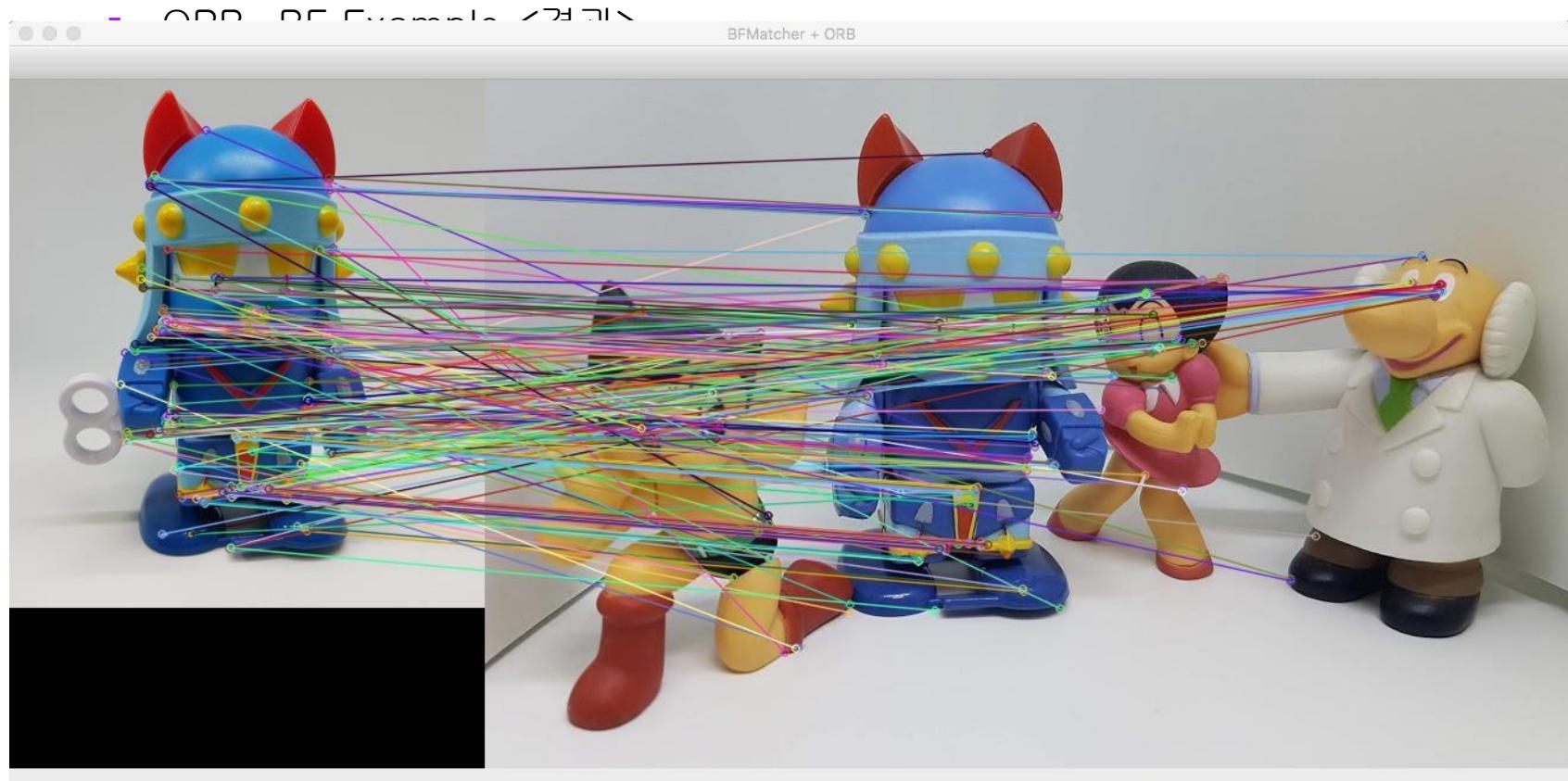
- ORB– BF Example

```
import cv2, numpy as np
img1 = cv2.imread('../img/taekwonv1.jpg')
img2 = cv2.imread('../img/figures.jpg')
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
# SIFT 서술자 추출기 생성 ---①
detector = cv2.ORB_create()
# 각 영상에 대해 키 포인트와 서술자 추출 ---②
kp1, desc1 = detector.detectAndCompute(gray1, None)
kp2, desc2 = detector.detectAndCompute(gray2, None)
# BFMatcher 생성, Hamming 거리, 상호 체크 ---③
matcher = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
# 매칭 계산 ---④
matches = matcher.match(desc1, desc2)
# 매칭 결과 그리기 ---⑤
res = cv2.drawMatches(img1, kp1, img2, kp2, matches, None,
                      flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)
cv2.imshow('BFMatcher + ORB', res)
cv2.waitKey()
cv2.destroyAllWindows()
```

[예제 8-15] BFMatcher와 ORB로 매칭(match_bf_orb.py)

Feature Matching

❖ BF(Brute-Force) Matching



[그림 8-21] [예제 8-15]의 실행 결과

Feature Matching

❖ FLANN Matching

- Fast Library for Approximate Nearest Neighbors Matching
- BF 매칭의 단점
 - 전수 조사 : 영상이 큰 경우 성능 저하
- 모든 descriptor를 비교하지 않고 가장 가까운 이웃의 근사값으로 매칭
 - 알고리즘 선택을 위한 추가 파라미터 필요
- 인덱스 파라미터와 검색 파라미터 지정
 - C++ 클래스 구현이 파이썬 바이딩에 적용 누락
 - 딕셔너리 객체에 키-값 쌍으로 작성

Feature Matching

❖ FLANN Matching

- matcher = cv2.FlannBasedMatcher([indexParams[, searchParams]])
 - indexParams : 인덱스 파라미터, 딕셔너리
 - algorithm : 알고리즘 선택 키, 선택한 키에 따라 종속 키 결정
 - FLANN_INDEX_LINEAR = 0 : 선형 인덱싱, BFMatcher와 동일
 - FLANN_INDEX_KDTREE = 1 : KD-트리 인덱싱
 - FLANN_INDEX_KMEANS = 2 : K-평균 트리 인덱싱
 - FLANN_INDEX_COMPOSITE = 3 : KD트리, K평균 혼합 인덱싱
 - FLANN_INDEX_LSH = 6 : LSH 인덱싱
 - FLANN_INDEX_AUTOTUNED = 255 : 자동 인덱스
 - searchParams : 검색 파라미터, 딕셔너리 객체
 - checks=32 : 검색할 후보 수
 - eps =0.0 : 사용 안함
 - sorted=True : 정렬해서 반환

Feature Matching

❖ FLANN Matching

- 인덱스 파라미터 작성 어렵고 까다로움
- 권장 인덱스 파라미터 값
 - SIFT, SURF

```
FLANN_INDEX_KDTREE=1  
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
```

- ORB

```
FLANN_INDEX_LSH=6  
index_params= dict(algorithm = FLANN_INDEX_LSH,  
                  table_number = 6,  
                  key_size = 12,  
                  multi_probe_level = 1)
```

Feature Matching

❖ FLANN Matching

- SIFT-FLANN Example

```
import cv2, numpy as np

img1 = cv2.imread('../img/taekwonv1.jpg')
img2 = cv2.imread('../img/figures.jpg')
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

# SIFT 생성
detector = cv2.xfeatures2d.SIFT_create()
# 키 포인트와 서술자 추출
kp1, desc1 = detector.detectAndCompute(gray1, None)
kp2, desc2 = detector.detectAndCompute(gray2, None)
```

[예제 8-16] FLANNMatcher 와 SIFT 로 매칭(match_flann_sift .py)

Feature Matching

❖ FLANN Matching

- SIFT-FLANN Example

```
# 인덱스 파라미터와 검색 파라미터 설정 ---①
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
search_params = dict(checks=50)

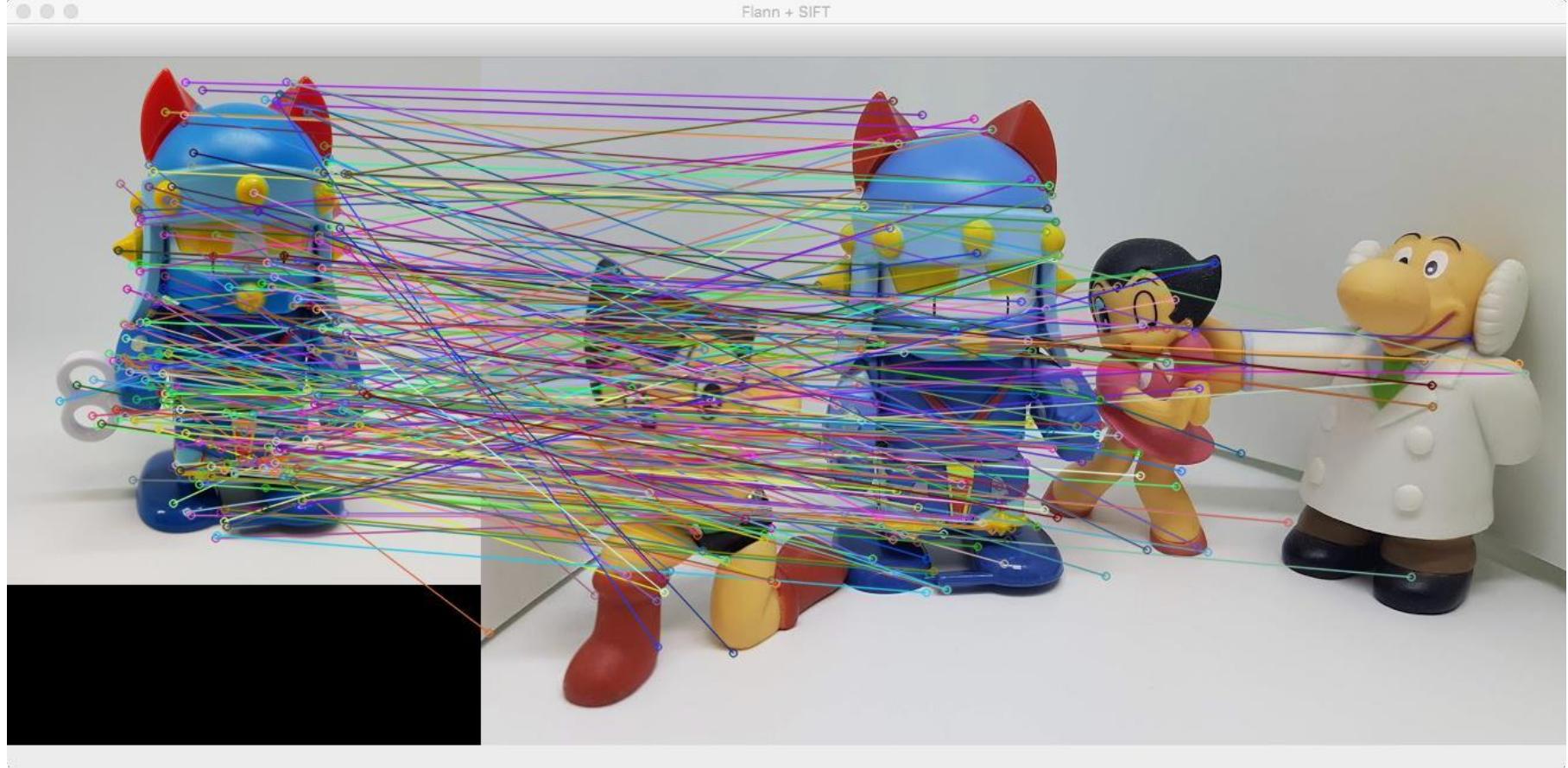
# Flann 매처 생성 ---③
matcher = cv2.FlannBasedMatcher(index_params, search_params)
# 매칭 계산 ---④
matches = matcher.match(desc1, desc2)
# 매칭 그리기
res = cv2.drawMatches(img1, kp1, img2, kp2, matches, None, \
                      flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)

cv2.imshow('Flann + SIFT', res)
cv2.waitKey()
cv2.destroyAllWindows()
```

Feature Matching

❖ FLANN Matching

- SIFT-FLANN Example



[그림 8-22] [예제 8-16]의 실행 결과

Feature Matching

❖ FLANN Matching

- SURF-FLANN Example

```
import cv2, numpy as np

img1 = cv2.imread('../img/taekwonv1.jpg')
img2 = cv2.imread('../img/figures.jpg')
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

# SURF 생성
detector = cv2.xfeatures2d.SURF_create()
# 키 포인트와 서술자 추출
kp1, desc1 = detector.detectAndCompute(gray1, None)
kp2, desc2 = detector.detectAndCompute(gray2, None)
```

[예제 8-17] FLANNMatcher 와 SURF 로 매칭(match_flann_surf.py)

Feature Matching

❖ FLANN Matching

- SURF-FLANN Example

```
# 인덱스 파라미터와 검색 파라미터 설정 ---①
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
search_params = dict(checks=50)

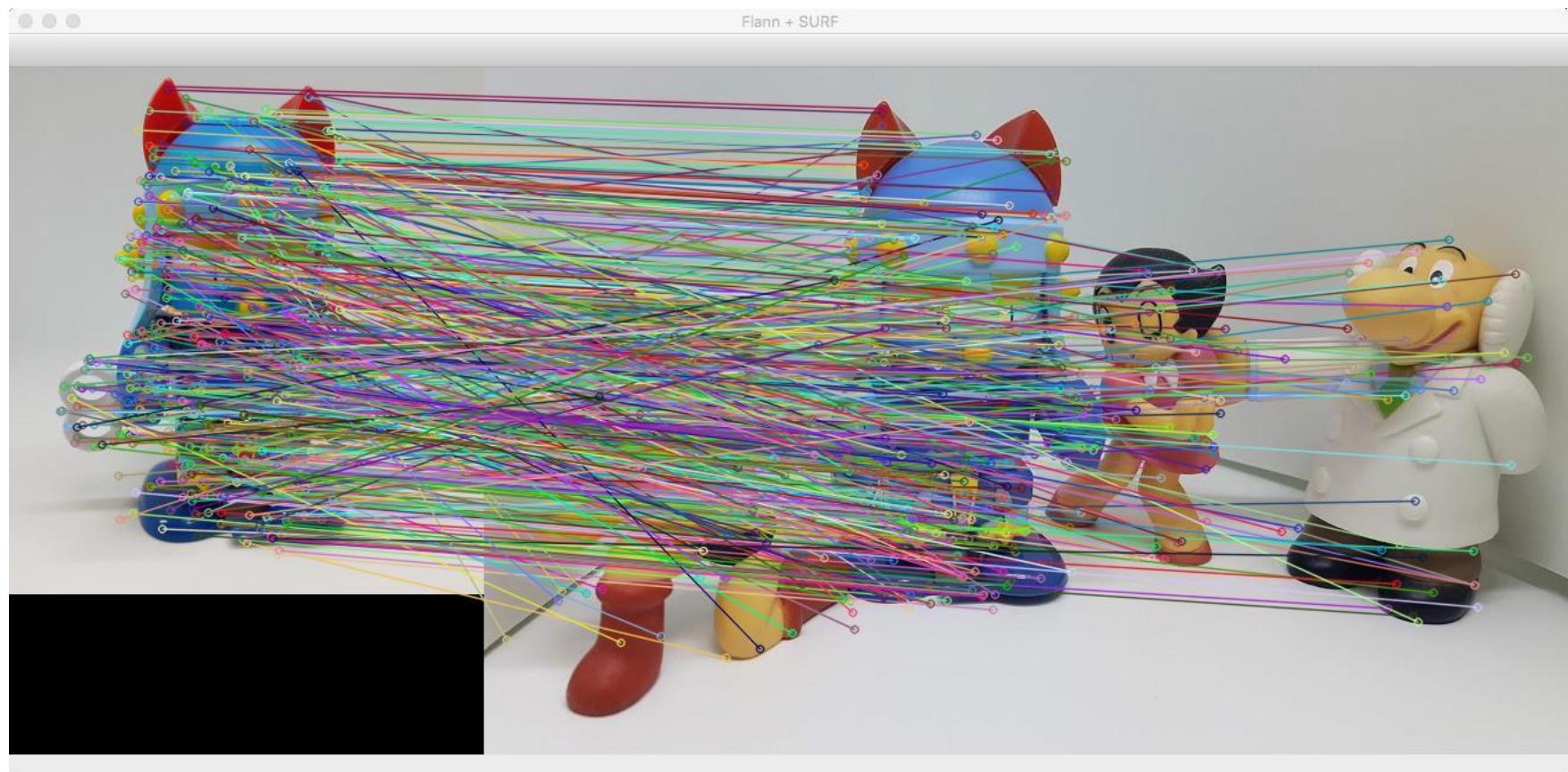
# Flann 매처 생성 ---③
matcher = cv2.FlannBasedMatcher(index_params, search_params)
# 매칭 계산 ---④
matches = matcher.match(desc1, desc2)
# 매칭 그리기
res = cv2.drawMatches(img1, kp1, img2, kp2, matches, None, \
                      flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)

cv2.imshow('Flann + SURF', res)
cv2.waitKey()
cv2.destroyAllWindows()
```

Feature Matching

❖ FLANN Matching

- SURF-FLANN Example



[그림 8-23] [예제 8-17]의 실행 결과

Feature Matching

❖ FLANN Matching

- ORB-FLANN Example

```
import cv2, numpy as np

img1 = cv2.imread('../img/taekwonv1.jpg')
img2 = cv2.imread('../img/figures.jpg')
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

# ORB 추출기 생성
detector = cv2.ORB_create()
# 키 포인트와 서술자 추출
kp1, desc1 = detector.detectAndCompute(gray1, None)
kp2, desc2 = detector.detectAndCompute(gray2, None)
```

[예제 8-18] FLANNMatcher 와 ORB 로 매칭(match_flann_orb.py)

Feature Matching

❖ FLANN Matching

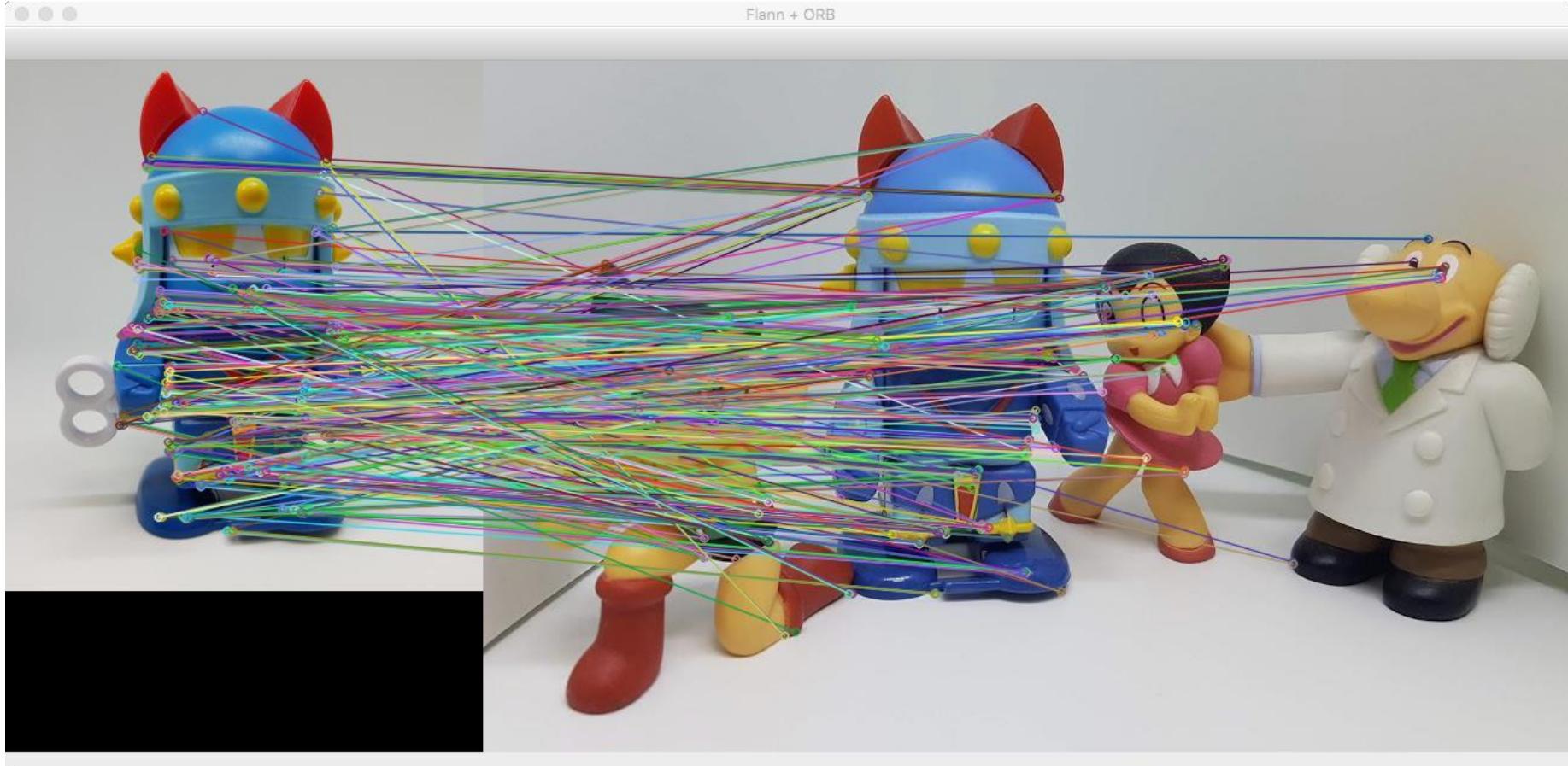
- ORB-FLANN Example

```
# 인덱스 파라미터 설정 ---①
FLANN_INDEX_LSH = 6
index_params= dict(algorithm = FLANN_INDEX_LSH,
                    table_number = 6,
                    key_size = 12,
                    multi_probe_level = 1)
# 검색 파라미터 설정 ---②
search_params=dict(checks=32)
# Flann 매처 생성 ---③
matcher = cv2.FlannBasedMatcher(index_params, search_params)
# 매칭 계산 ---④
matches = matcher.match(desc1, desc2)
# 매칭 그리기
res = cv2.drawMatches(img1, kp1, img2, kp2, matches, None, \
                      flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)
# 결과 출력
cv2.imshow('Flann + ORB', res)
cv2.waitKey()
cv2.destroyAllWindows()
```

Feature Matching

❖ FLANN Matching

- SURF-FLANN Example



[그림 8-24] [예제 8-18]의 실행 결과

Feature Matching

❖ Finding Good Match Points

- match()
 - 서술자 하나 당 한개의 매칭점 반환
 - 거리가 작은 상위 퍼센트 사용
- knnMatch()
 - 서술자 당 k개의 최근접 이웃 매칭점
 - 이웃 매칭점들 간 거리가 가까운 것
- radiusMatch()
 - 특정 거리 이내
 - 좋은 매칭점 찾기 의미 없음

Feature Matching

❖ Finding Good Matching

- match() Example <1/2>

```
import cv2, numpy as np

img1 = cv2.imread('../img/taekwonv1.jpg')
img2 = cv2.imread('../img/figures.jpg')
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

# ORB로 서술자 추출 ---①
detector = cv2.ORB_create()
kp1, desc1 = detector.detectAndCompute(gray1, None)
kp2, desc2 = detector.detectAndCompute(gray2, None)
# BF-Hamming으로 매칭 ---②
matcher = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
matches = matcher.match(desc1, desc2)
```

[예제 8-19] match 함수로부터 좋은 매칭점 찾기(match_good.py)

Feature Matching

❖ Finding Good Matching

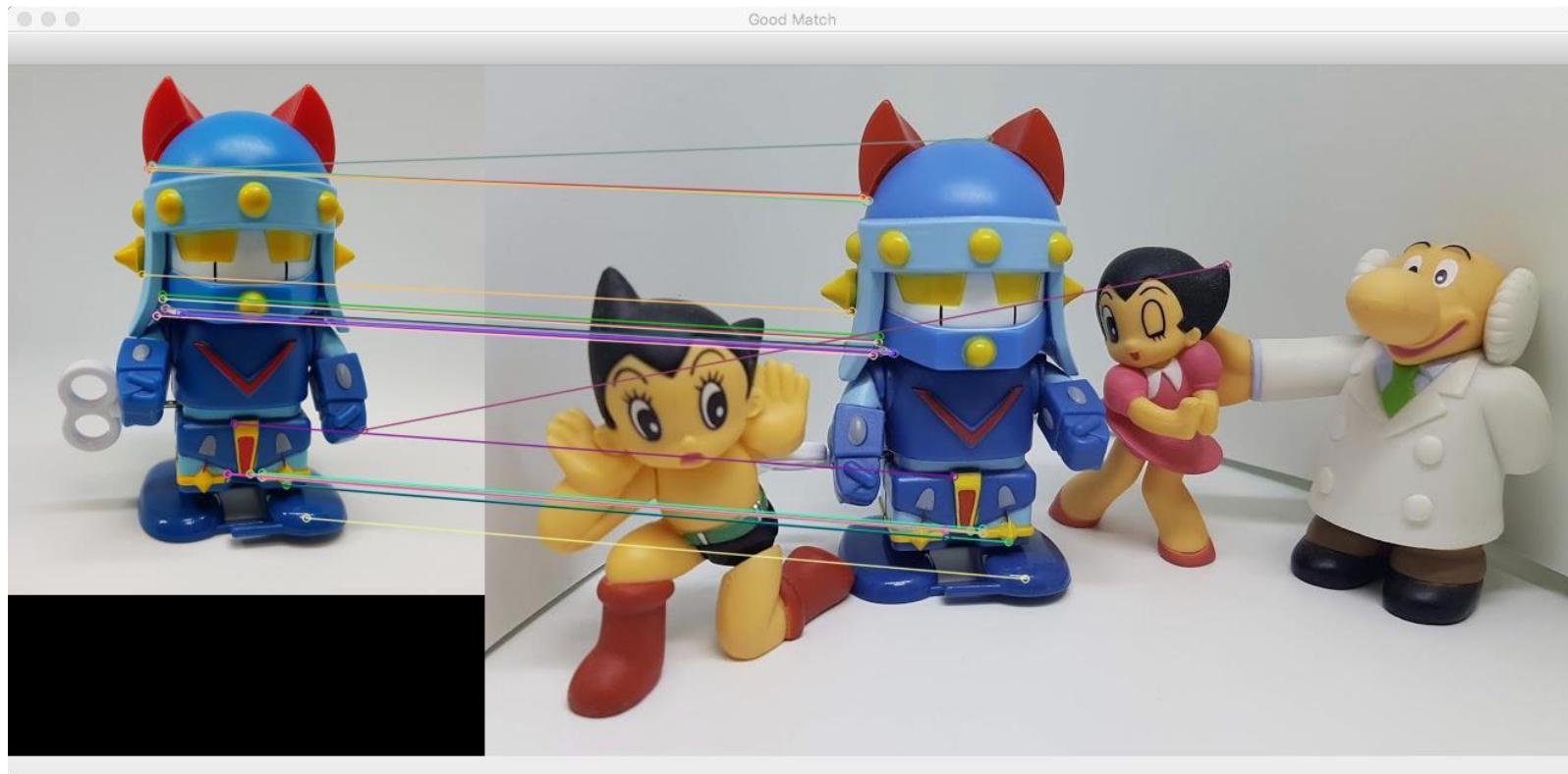
- match() Example <2/2>

```
# 매칭 결과를 거리기준 오름차순으로 정렬 ---③
matches = sorted(matches, key=lambda x:x.distance)
# 최소 거리 값과 최대 거리 값 확보 ---④
min_dist, max_dist = matches[0].distance, matches[-1].distance
# 최소 거리의 15% 지점을 임계점으로 설정 ---⑤
ratio = 0.2
good_thresh = (max_dist - min_dist) * ratio + min_dist
# 임계점 보다 작은 매칭점만 좋은 매칭점으로 분류 ---⑥
good_matches = [m for m in matches if m.distance < good_thresh]
print('matches:%d/%d, min:%.2f, max:%.2f, thresh:%.2f' \
      %(len(good_matches),len(matches), min_dist, max_dist, good_thresh))
# 좋은 매칭점만 그리기 ---⑦
res = cv2.drawMatches(img1, kp1, img2, kp2, good_matches, None, \
                      flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)
# 결과 출력
cv2.imshow('Good Match', res)
cv2.waitKey()
cv2.destroyAllWindows()
```

Feature Matching

❖ FLANN Matching

- match() Example



[그림 8-25] [예제 8-19]의 실행 결과

Feature Matching

❖ Finding Good Matching

- knnMatch() Example

```
import cv2, numpy as np

img1 = cv2.imread('../img/taekwonv1.jpg')
img2 = cv2.imread('../img/figures.jpg')
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
# ORB로 서술자 추출 ---①
detector = cv2.ORB_create()
kp1, desc1 = detector.detectAndCompute(gray1, None)
kp2, desc2 = detector.detectAndCompute(gray2, None)
# BF-Hamming 생성 ---②
matcher = cv2.BFMatcher(cv2.NORM_HAMMING2)
# knnMatch, k=2 ---③
matches = matcher.knnMatch(desc1, desc2, 2)
```

[예제 8-20] knnmatch 함수로부터 좋은 매칭점 찾기(match_good_knn.py)

Feature Matching

❖ Finding Good Matching

- knnMatch() Example

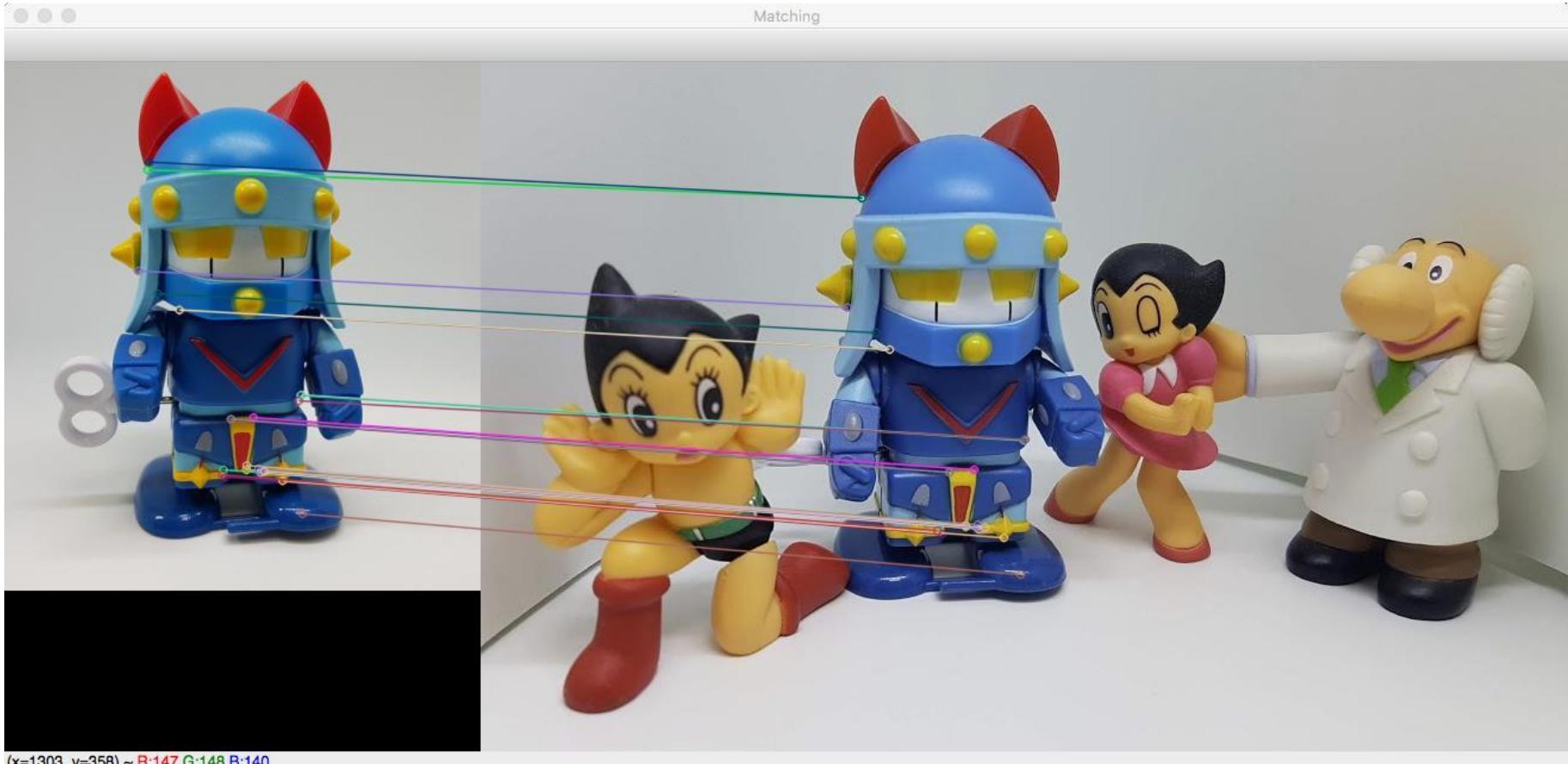
```
# 첫번째 이웃의 거리가 두 번째 이웃 거리의 75% 이내인 것만 추출---⑤
ratio = 0.75
good_matches = [first for first,second in matches \
                 if first.distance < second.distance * ratio]
print('matches:%d/%d' %(len(good_matches),len(matches)))

# 좋은 매칭만 그리기
res = cv2.drawMatches(img1, kp1, img2, kp2, good_matches, None, \
                      flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)
# 결과 출력
cv2.imshow('Matching', res)
cv2.waitKey()
cv2.destroyAllWindows()
```

Feature Matching

❖ Finding Good Matching

- kNNmatch() Example



[그림 8-26] [예제 8-20]의 실행 결과

Feature Matching

❖ **Matching Homography to find Object**

- 좋은 매칭점들로 두 영상간의 원근 변환 행렬 계산
- 객체 위치 파악
- 변환 행렬 이상치로 나쁜 매칭점 제거
- cv2.findHomography()
 - 여러개의 매칭쌍으로 원근 변환 행렬 근사 계산
- cv2.perspectiveTransform()
 - 원래의 좌표를 변환 행렬에 맞게 원근 변환

Feature Matching

❖ Matching Homography to find Object

- mtrx, mask = cv.findHomography(srcPoints, dstPoints[, method[, ransacReprojThreshold[, mask[, maxIters[, confidence]]]]])
 - srcPoints: 원본 좌표 배열
 - dstPoints: 결과 좌표 배열
 - method=0 : 근사 계산 알고리즘 선택
 - 0 : 모든 점으로 최소 제곱 오차 계산
 - cv2.RANSAC
 - cv2.LMEDS
 - cv2.RHO
 - ransacReprojThreshold=3 : 정상치 거리 임계 값, RANSAC,RHO 인 경우
 - maxIters=2000 : 근사계산 반복 횟수
 - confidence=0.995 : 신뢰도, 0 ~ 1
 - mtrx : 결과 변환 행렬
 - mask : 정상치 판별 결과, N x 1행 배열 (0:비정상치, 1:정상치)
- dst = cv.perspectiveTransform(src, m[, dst])
- src : 입력 좌표 배열
- m : 변환 행렬
- dst : 출력 좌표 배열

Feature Matching

❖ Matching Homography to find Object

- 근사 계산 알고리즘
 - RANSAC(Random Sample Consensus)
 - 임의의 점들 선정, 만족도가 가장 큰 점들만 근사 계산
 - 정상치(Inlier), 이상치(outlier)
 - LMEDS(Least Median of Squares)
 - 제곱의 최소 중간 값
 - 추가 파라미터 불 필요
 - 정상치가 50% 이상인 경우만 동작
 - RHO
 - RANSAC을 개선한 PROSAC(Progressive Sample Consensus) 사용
 - 이상치가 많은 경우 더 빠르다

Feature Matching

❖ Matching Homography to find Object

```
import cv2, numpy as np
img1 = cv2.imread('../img/taekwonv1.jpg')
img2 = cv2.imread('../img/figures.jpg')
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
# ORB, BF-Hamming 로 knnMatch ---①
detector = cv2.ORB_create()
kp1, desc1 = detector.detectAndCompute(gray1, None)
kp2, desc2 = detector.detectAndCompute(gray2, None)
matcher = cv2.BFMatcher(cv2.NORM_HAMMING2)
matches = matcher.knnMatch(desc1, desc2, 2)
# 이웃 거리의 75%로 좋은 매칭점 추출---②
ratio = 0.75
good_matches = [first for first,second in matches \
                if first.distance < second.distance * ratio]
print('good matches:%d/%d' %(len(good_matches),len(matches)))
```

[예제 8-21] 매칭점 원근 변환으로 영역 찾기(match_homography.py)

Feature Matching

❖ Matching Homography to find Object

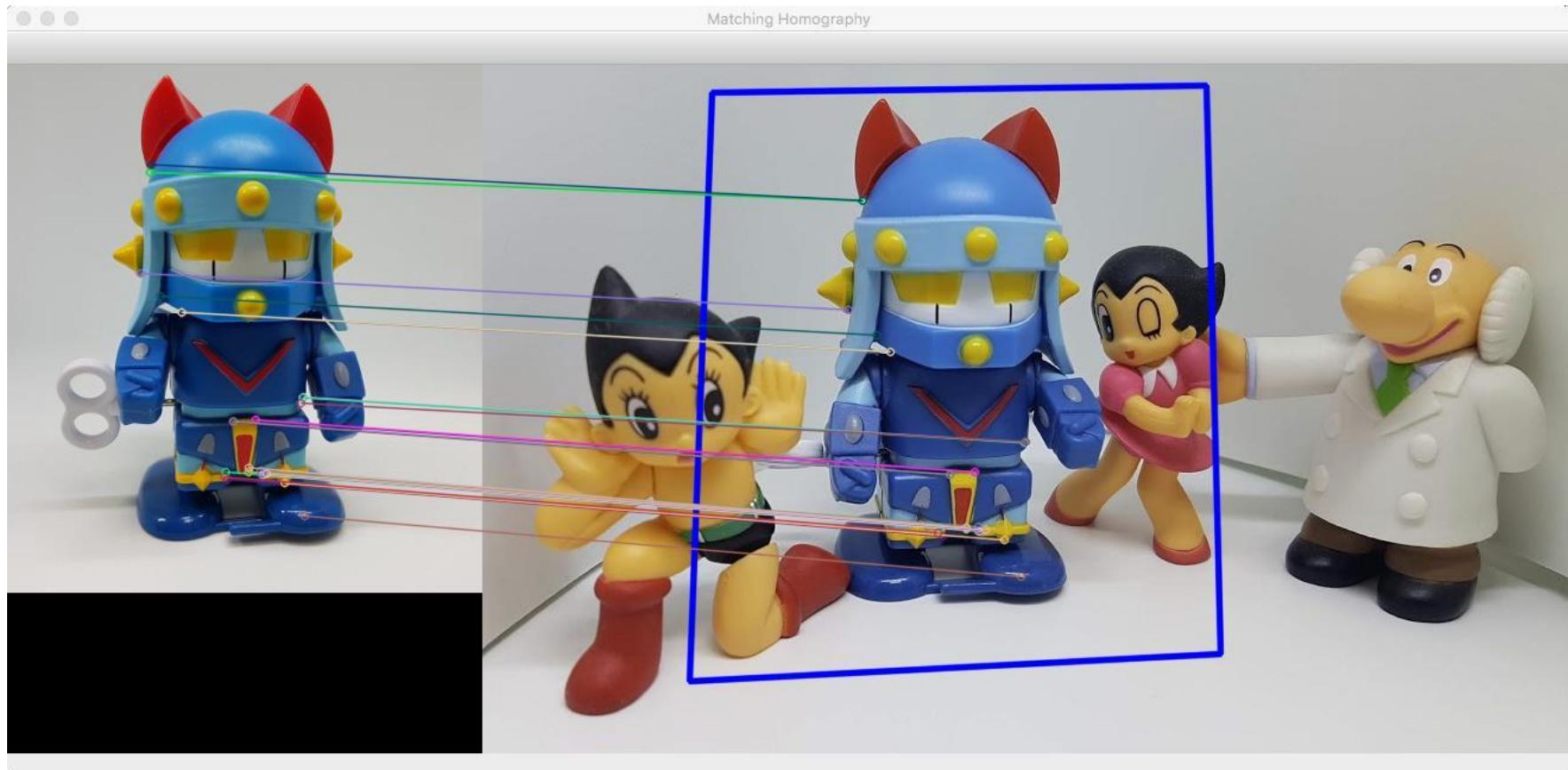
- Example <2/2>

```
# 좋은 매칭점의 queryIdx로 원본 영상의 좌표 구하기 ---③
src_pts = np.float32([ kp1[m.queryIdx].pt for m in good_matches ])
# 좋은 매칭점의 trainIdx로 대상 영상의 좌표 구하기 ---④
dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good_matches ])
# 원근 변환 행렬 구하기 ---⑤
mtrx, mask = cv2.findHomography(src_pts, dst_pts)
# 원본 영상 크기로 변환 영역 좌표 생성 ---⑥
h,w, = img1.shape[:2]
pts = np.float32([ [[0,0]],[[0,h-1]],[[w-1,h-1]],[[w-1,0]] ])
# 원본 영상 좌표를 원근 변환 ---⑦
dst = cv2.perspectiveTransform(pts,mtrx)
# 변환 좌표 영역을 대상 영상에 그리기 ---⑧
img2 = cv2.polylines(img2,[np.int32(dst)],True,255,3, cv2.LINE_AA)
# 좋은 매칭 그려서 출력 ---⑨
res = cv2.drawMatches(img1, kp1, img2, kp2, good_matches, None, \
                      flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)
cv2.imshow('Matching Homography', res)
cv2.waitKey()
cv2.destroyAllWindows()
```

Feature Matching

❖ Matching Homography to find Object

- Example <2/2>



[그림 8-27] [예제 8-21]의 실행 결과

Feature Matching

❖ Remove Bad Matching with RANSAC

- Example <1/2>

```
import cv2, numpy as np
img1 = cv2.imread('../img/taekwonv1.jpg')
img2 = cv2.imread('../img/figures2.jpg')
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
# ORB, BF-Hamming 로 knnMatch ---①
detector = cv2.ORB_create()
kp1, desc1 = detector.detectAndCompute(gray1, None)
kp2, desc2 = detector.detectAndCompute(gray2, None)
matcher = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
matches = matcher.match(desc1, desc2)
# 매칭 결과를 거리기준 오름차순으로 정렬 --- ②
matches = sorted(matches, key=lambda x:x.distance)
# 모든 매칭점 그리기 --- ③
res1 = cv2.drawMatches(img1, kp1, img2, kp2, matches, None, \
                      flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)
```

[예제 8-22] RANSAC 원근 변환 근사 계산으로 나쁜 매칭 제거(match_homography_accuracy.py)

Feature Matching

❖ Remove Bad Matching with RANSAC

- Example <2/2>

```
# 매칭점으로 원근 변환 및 영역 표시 --- ④
src_pts = np.float32([ kp1[m.queryIdx].pt for m in matches ])
dst_pts = np.float32([ kp2[m.trainIdx].pt for m in matches ])
# RANSAC으로 변환 행렬 근사 계산 --- ⑤
mtrx, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
h,w = img1.shape[:2]
pts = np.float32([[ [0,0]],[[0,h-1]],[[w-1,h-1]],[[w-1,0]] ])
dst = cv2.perspectiveTransform(pts,mtrx)
img2 = cv2.polylines(img2,[np.int32(dst)],True,255,3, cv2.LINE_AA)
# 정상치 매칭만 그리기 --- ⑥
matchesMask = mask.ravel().tolist()
res2 = cv2.drawMatches(img1, kp1, img2, kp2, matches, None, \
                      matchesMask = matchesMask,
                      flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)
# 모든 매칭점과 정상치 비율 --- ⑦
accuracy=float(mask.sum()) / mask.size
print("accuracy: %d/%d(% .2f%%)"% (mask.sum(), mask.size, accuracy))
# 결과 출력
cv2.imshow('Matching-All', res1)
cv2.imshow('Matching-Inlier ', res2)
cv2.waitKey()
cv2.destroyAllWindows()
```

Feature Matching

❖ Remove Bad Matching with RANSAC

- Example <2/2>



[그림 8-28] [예제 8-22]의 실행 결과

Feature Matching

❖ Finding Object with Camera

- Example <1/5>

```
import cv2, numpy as np
img1 = None
win_name = 'Camera Matching'
MIN_MATCH = 10
# ORB 검출기 생성
detector = cv2.ORB_create(1000)
# Flann 추출기 생성
FLANN_INDEX_LSH = 6
index_params= dict(algorithm = FLANN_INDEX_LSH,
                    table_number = 6,
                    key_size = 12,
                    multi_probe_level = 1)
search_params=dict(checks=32)
matcher = cv2.FlannBasedMatcher(index_params, search_params)
```

[예제 8-23] 카메라로 객체 매칭(match_camera.py)

❖ Finding Object with Camera

- Example <2/5>

```
# 카메라 캡쳐 연결 및 프레임 크기 축소
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
while cap.isOpened():
    ret, frame = cap.read()
    if img1 is None: # 등록된 이미지 없음, 카메라 바이패스
        res = frame
    else: # 등록된 이미지 있는 경우, 매칭 시작
        img2 = frame
        gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
        gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
        # 키포인트와 디스크립터 추출
        kp1, desc1 = detector.detectAndCompute(gray1, None)
        kp2, desc2 = detector.detectAndCompute(gray2, None)
        # k=2로 knnMatch
        matches = matcher.knnMatch(desc1, desc2, 2)
```

❖ Finding Object with Camera

- Example <3/5>

```
# 이웃 거리의 75%로 좋은 매칭점 추출
ratio = 0.75
good_matches = [m[0] for m in matches \
                 if len(m) == 2 and m[0].distance < m[1].distance * ratio]
print('good matches:%d/%d' %(len(good_matches),len(matches)))
# 모든 매칭점 그리지 못하게 마스크를 0으로 채움
matchesMask = np.zeros(len(good_matches)).tolist()
# 좋은 매칭점 최소 갯수 이상인 경우
if len(good_matches) > MIN_MATCH:
    # 좋은 매칭점으로 원본과 대상 영상의 좌표 구하기
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good_matches ])
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good_matches ])
    # 원근 변환 행렬 구하기
    mtrx, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
    accuracy=float(mask.sum()) / mask.size
    print("accuracy: %d/%d(% .2f%%)"% (mask.sum(), mask.size, accuracy))
```

❖ Finding Object with Camera

- Example <4/5>

```
if mask.sum() > MIN_MATCH: # 정상치 매칭점 최소 갯수 이상 인 경우
    # 이상점 매칭점만 그리게 마스크 설정
    matchesMask = mask.ravel().tolist()
    # 원본 영상 좌표로 원근 변환 후 영역 표시
    h,w, = img1.shape[:2]
    pts = np.float32([ [0,0],[[0,h-1],[[w-1,h-1],[[w-1,0]] ]])
    dst = cv2.perspectiveTransform(pts,mtrx)
    img2 = cv2.polylines(img2,[np.int32(dst)],True,255,3, cv2.LINE_AA)
# 마스크로 매칭점 그리기
res = cv2.drawMatches(img1, kp1, img2, kp2, good_matches, None, \
                      matchesMask=matchesMask,
                      flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)
```

❖ Finding Object with Camera

- Example <5/5>

```
# 결과 출력
cv2.imshow(win_name, res)
key = cv2.waitKey(1)
if key == 27: # Esc, 종료
    break
elif key == ord(' '): # 스페이스바를 누르면 ROI로 img1 설정
    x,y,w,h = cv2.selectROI(win_name, frame, False)
    if w and h:
        img1 = frame[y:y+h, x:x+w]
else:
    print("can't open camera.")
cap.release()
cv2.destroyAllWindows()
```

Feature Matching

❖ Finding Object with Camera

- Example <결과>



[그림 8-29] [예제 8-23]의 실행 결과

세부목차

1. Matching with a similar image
2. Feature and Key Point
3. Descriptor Extractor
4. Feature Matching
- 5. Tracking**
6. Workshop

Tracking

❖ Background Subtraction

- 방문객수나 교통량 측정에 필요
- 준비된 배경 사진과 연산을 통해 쉽게 계산
- 준비된 배경 사진이 없는 경우가 더 많다
- cv2.BackgroundSubtractor 추상 클래스
 - fgmask = bgs subtractor.apply(image[, fgmask[, learningRate]])
 - image : 입력 영상
 - fgmask : 전경 마스크
 - learningRate=-1 : 배경 훈련 속도, 0 ~ 1, -1:자동
 - backgroundImage =
bgs subtractor.getBackgroundImage([,backgroundImage])
 - backgroundImage : 훈련용 배경 이미지
- 배경 제거 알고리즘
 - BackgroundSubtractorMOG
 - BackgroundSubtractorMOG2
 - BackgroundSubtractorGMG

Tracking

❖ **BackgroundSubtractorMOG**

- Gausisian Mixture-based Backgorund/Fofeground Segmentation Algorithm
- K Gaussian 분포(K=3에서 5)
- 혼합을 배경 픽셀 각각에 적용
- 가중치는 특정 픽셀이 같은 곳에 오래 머물러 있는가에 대한 비율
 - 배경이 오래 머물렀거나 더 고정적일 것
- cv2.bgsegm.createBackgroundSubtractorMOG()
 - history 길이, 가우시안 믹스쳐의 숫자, threshold 값을 지정해야 한다.
 - 기본값이 설정되어 있어서 apply(frame)만 호출
 - contrib 모듈

Tracking

❖ BackgroundSubtractorMOG

- Example

```
import numpy as np, cv2
cap = cv2.VideoCapture('../img/walking.avi')
fps = cap.get(cv2.CAP_PROP_FPS) # 프레임 수 구하기
delay = int(1000/fps)
# 배경 제거 객체 생성 --- ①
fgbg = cv2.bgsegm.createBackgroundSubtractorMOG()
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
    # 배경 제거 마스크 계산 --- ②
    fgmask = fgbg.apply(frame)
    cv2.imshow('frame',frame)
    cv2.imshow('bgsu',fgmask)
    if cv2.waitKey(1) & 0xff == 27:
        break
cap.release()
cv2.destroyAllWindows()
```

[예제 8-24] BackgroundSubtractorMOG로 배경 제거(track_bgsu_mog.py)

Tracking

❖ BackgroundSubtractorMOG

- Example <결과>



[그림 8-30] [예제 8-24]의 실행 결과

Tracking

❖ **BackgroundSubtractorMOG2**

- 2개의 논문을 기반.
- Z.Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction" in 2004
- "Efficient Adaptive Density Estimation per Image Pixel for the Task of Background Subtraction" in 2006.
- 각 픽셀의 적절한 가우시안 분포 값을 선택
- 빛에 대한 변환에 따른 다양한 장면에 더 적합
- `retval = cv.createBackgroundSubtractorMOG2([, history[, varThreshold [, detectShadows]]])`
 - `history=500` : 히스토리 갯수
 - `varThreshold=16` : 분산 임계 값
 - `detectShadows=True` : 그림자 표시
- `fgbg.apply(frame)`
 - `detecShadows=True`(기본값)를 지정
 - 그림자를 찾아서 gray로 표시, 느려진다.

Tracking

❖ BackgroundSubtractorMOG2

- Example

```
import numpy as np, cv2
cap = cv2.VideoCapture('../img/walking.avi')
fps = cap.get(cv2.CAP_PROP_FPS) # 프레임 수 구하기
delay = int(1000/fps)
# 배경 제거 객체 생성 --- ①
fgbg = cv2.createBackgroundSubtractorMOG2()
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
    # 배경 제거 마스크 계산 --- ②
    fgmask = fgbg.apply(frame)
    cv2.imshow('frame',frame)
    cv2.imshow('bgsu',fgmask)
    if cv2.waitKey(delay) & 0xff == 27:
        break
    cap.release()
cv2.destroyAllWindows()
```

[예제 8-24] BackgroundSubtractorMOG2로 배경 제거(track_bgsu_mog2.py)

Tracking

❖ BackgroundSubtractorMOG2

- Example <결과>

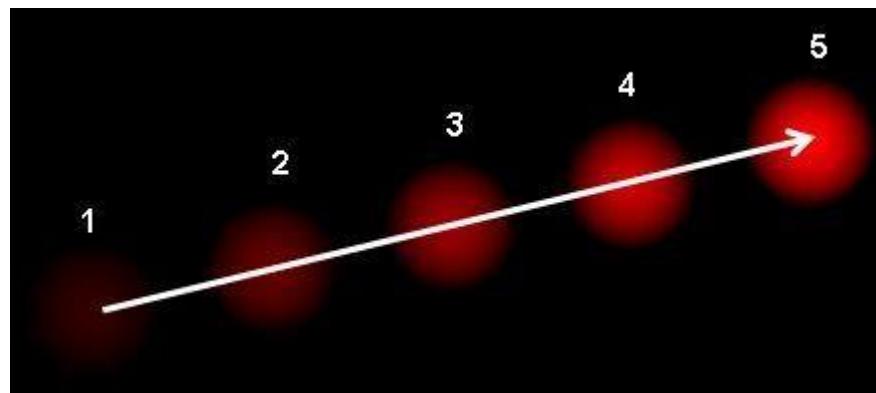


[그림 8-31] [예제 8-25]의 실행 결과

Tracking

❖ Optical Flow

- 이전 프레임과 현재 프레임 사이의 픽셀 이동 방향과 거리에 대한 분포
- 프레임에 사전 정보 없이 움직임을 추정하는 기법
- Sparse Optical Flow (희소)
 - 추적할 점들의 일부를 미리 지정
 - Lucas-Kanade(루카스 카나데) : 대표적 Sparse 방법
- Dense Optical Flow(밀도)
 - 모든 픽셀에서 속도를 구하는 것
 - 연산양이 많음



Tracking

❖ Lucas—Kaneda

- LK 알고리즘의 3가지 가정
 - Brightness Constancy(밝기 항상성)
 - 주적하는 객체 픽셀의 밝기는 변하지 않는다
 - Temporal Persistance(시간 지속성)
 - 영상에서 객체의 움직임에 비해 시간이 더 빠르다
 - 프레임간의 차이는 크지 않다
 - Spatial Consistency(공간 일관성)
 - 공간적으로 인접한 점들(이웃 픽셀)은 동일한 객체에 속할 가능성이 높고
 - 같은 움직임을 같는다
- 작은 윈도우를 사용하기 때문에 큰 움직임을 계산하지 못한다
- 이것을 개선하기 위해 이미지 피라미드 사용
- OpenCV 함수
 - cv2.calcOpticalFlowLK : 이미지 피라미드 사용 안함
 - cv2.calcOpticalFlowPyrLK : 이미지 피라미드 사용
 - goodToTrack에 의해 선정된 점을 입력으로 사용

Tracking

❖ Lucas—Kaneda

- calcOpticalFlowPyrLK(prevImg, nextImg, prevPts, nextPts)
 - prevImg : 이전 프레임 , $n \times 1 \times 2$
 - nextImg : 다음 프레임, $n \times 1 \times 2$
 - prevPts : 움직임 추적 대상 점 , $n \times 1 \times 2$
 - nextPts : 추적 대상점들이 이동한 점들의 위치, None, return 값 사용
 - winSize : 각 피라미드 레벨의 윈도우 크기,
 - maxLevel : 이미지 피라미드 계층 수, 0이면 사용안함
 - criteria : 검색 중지 판단기준
- return:
 - nextPts : 추적 대상점들이 이동한 점들의 위치
 - status : nextPts와 동일한 갯수
 - 각 인덱스에 1 또는 0, 0: 대응점 발견 못함($n \times 1$)
 - err : 이전 프레임에서의 특징점과 현재 프레임에서 찾아진 이동된 점 사이의 거리, 제거 용도

Tracking

❖ Lucas—Kaneda

- Flow Example <1/3>

```
import numpy as np, cv2
cap = cv2.VideoCapture('../img/walking.avi')
fps = cap.get(cv2.CAP_PROP_FPS) # 프레임 수 구하기
delay = int(1000/fps)
# 추적 경로를 그리기 위한 랜덤 색상
color = np.random.randint(0,255,(200,3))
lines = None #추적 선을 그릴 이미지 저장 변수
prevImg = None # 이전 프레임 저장 변수
# calcOpticalFlowPyrLK 중지 요건 설정
termcriteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03)
while cap.isOpened():
    ret,frame = cap.read()
    if not ret:
        break
    img_draw = frame.copy()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

[예제 8-26] calcOpticalFlowPyrLK 추적 (track_opticalLK.py)

Tracking

❖ Lucas—Kaneda

- Flow Example <2/3>

```
# 최초 프레임 경우
if prevImg is None:
    prevImg = gray
    # 추적선 그릴 이미지를 프레임 크기에 맞게 생성
    lines = np.zeros_like(frame)
    # 추적 시작을 위한 코너 검출 ---①
    prevPt = cv2.goodFeaturesToTrack(prevImg, 200, 0.01, 10)
else:
    nextImg = gray
    # 옵티컬 플로우로 다음 프레임의 코너점 찾기 ---②
    nextPt, status, err = cv2.calcOpticalFlowPyrLK(prevImg, nextImg,
                                                   prevPt, None, criteria=termcriteria)
    # 대응점이 있는 코너, 움직인 코너 선별 ---③
    prevMv = prevPt[status==1]
    nextMv = nextPt[status==1]
    for i,(p, n) in enumerate(zip(prevMv, nextMv)):
        px,py = p.ravel()
        nx,ny = n.ravel()
```

Tracking

❖ Lucas—Kaneda

▪ Flow Example <3/3>

```
# 이전 코너와 새로운 코너에 선 그리기 ---④
cv2.line(lines, (px, py), (nx,ny), color[i].tolist(), 2)
# 새로운 코너에 점 그리기
cv2.circle(img_draw, (nx,ny), 2, color[i].tolist(), -1)
# 누적된 추적 선을 출력 이미지에 합성 ---⑤
img_draw = cv2.add(img_draw, lines)
# 다음 프레임을 위한 프레임과 코너점 이월
prevImg = nextImg
prevPt = nextMv.reshape(-1,1,2)
cv2.imshow('OpticalFlow-LK', img_draw)
key = cv2.waitKey(delay)
if key == 27 : # Esc:종료
    break
elif key == 8: # Backspace:추적 이력 지우기
    prevImg = None
cv2.destroyAllWindows()
cap.release()
```

Tracking

❖ Luca—Kaneda

- Flow Example <결과>



[그림 8-32] [예제 8-26]의 실행 결과

Tracking

❖ Dense Optical Flow

- cv2.calcOpticalFlowFarneback(prev, next, flow, pyr_scale, levels, winsize, iterations, poly_n, poly_sigma, flags)
 - prev : 이전 이미지
 - next : 다음 프레임
 - flow : 32F 타입으로 이전프레임과 같은 크기를 갖는 계산 결과 flow, None
 - pyr_scale : 피라미트 스케일, 0.5
 - levels : 피라미트 레벨 갯수
 - winsize : 평균 윈도우 사이즈, 큰 값은 강한 알고리즘을 만든다.
 - iteration : 각 피라미드에서 반복할 횟수
 - poly_n : 각픽셀에 polynomial 식을 찾기위해 사용할 이웃픽셀 수, 5 or 7
 - poly_sigma: 가우시안 편차에 사용할 시그마값 poly_n=5:1.1,poly_n=7:1.5
 - flags : 연산 모드
 - cv2.OPTFLOW_USE_INITIAL_FLOW : flow 값을 초기 값으로 사용
 - cv2.OPTFLOW_FARNEBACK_GAUSSIAN : 박스 필터 대신 가우시안 필터 사용
 - 반환 : 입력 이미지와 같은 크기, 각 픽셀이 이동한 거리 값

Tracking

❖ Dense Optical Flow

- Example <1/2>

```
import cv2, numpy as np
# 플로우 결과 그리기 ---①
def drawFlow(img,flow,step=16):
    h,w = img.shape[:2]
    # 16픽셀 간격의 그리드 인덱스 구하기 ---②
    idx_y,idx_x = np.mgrid[step/2:h:step,step/2:w:step].astype(np.int)
    indices = np.stack( (idx_x,idx_y), axis =-1).reshape(-1,2)
    for x,y in indices: # 인덱스 순회
        # 각 그리드 인덱스 위치에 점 그리기 ---③
        cv2.circle(img, (x,y), 1, (0,255,0), -1)
        # 각 그리드 인덱스에 해당하는 플로우 값 (이동 거리) ---④
        dx,dy = flow[y, x].astype(np.int)
        # 각 그리드 인덱스 위치에서 이동한 거리 만큼 선 그리기 ---⑤
        cv2.line(img, (x,y), (x+dx, y+dy), (0,255, 0),2, cv2.LINE_AA )
    prev = None # 이전 프레임 저장 변수
    cap = cv2.VideoCapture('../img/walking.avi')
    fps = cap.get(cv2.CAP_PROP_FPS) # 프레임 수 구하기
    delay = int(1000/fps)
```

[예제 8-27] calcOpticalFlowFarneback 추적 (track_optical_farneback.py)

Tracking

❖ Dense Optical Flow

- Example <2/2>

```
while cap.isOpened():
    ret, frame = cap.read()
    If not ret: break
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # 최초 프레임 경우
    if prev is None:
        prev = gray # 첫 이전 프레임 --- ⑥
    else:
        # 이전, 이후 프레임으로 옵티컬 플로우 계산 --- ⑦
        flow = cv2.calcOpticalFlowFarneback(prev, gray, None, \
            0.5, 3, 15, 3, 5, 1.1, cv2.OPTFLOW_FARNEBACK_GAUSSIAN)
        # 계산 결과 그리기, 선언한 함수 호출 --- ⑧
        drawFlow(frame, flow)
        # 다음 프레임을 위해 이월 --- ⑨
        prev = gray
    cv2.imshow('OpticalFlow-Farneback', frame)
    if cv2.waitKey(delay) == 27:
        break
    cap.release()
    cv2.destroyAllWindows()
```

Tracking

❖ Dense Optical Flow

- Example <2/2>

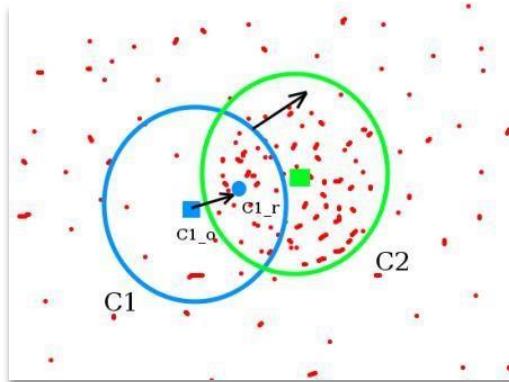


[그림 8-33] [예제 8-27]의 실행 결과

Tracking

❖ Meanshift

- 평균 이동
- 밀도 분포(특징점 또는 색상)를 기반으로 ROI 객체 추적
 - 초기 윈도우 결정
 - 윈도우 내에서 데이터의 무게 중심(center of mass) 계산
 - 윈도우의 중심을 무게 중심 위치로 이동
 - 윈도우 움직임이 멈출 때 까지 반복
- 적용 절차
 - 추적 대상 선정해서 HSV 컬러의 H 히스토그램 계산
 - 전체 영상의 히스토그램으로 Back Projection
 - Back Projection에서 이동한 객체 MeanShift로 추적



Tracking

❖ **Meanshift**

- cv2.meanShift(img, window, criteria)
 - img : 입력영상
 - window : 초기 원도우 좌표
 - criteria : 반복 검색 중지 요건
 - type :cv2.TERM_CRITERIA_*
 - EPS : 정확도가 epsilon 보다 작아지면
 - MAX_ITER : max_iter 회수 반복
 - COUNT : MAX_ITER와 동일
 - max_iter : 최대 반복 횟수
 - epsilon : 최소 정확도

Tracking

❖ Meanshift

- Example <1/3>

```
import numpy as np, cv2
roi_hist = None # 추적 객체 히스토그램 저장 변수
win_name = 'MeanShift Tracking'
termination = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1)
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
while cap.isOpened():
    ret, frame = cap.read()
    img_draw = frame.copy()
    if roi_hist is not None: # 추적 대상 객체 히스토그램 등록 됨
        # 전체 영상 hsv 컬로 변환 ---①
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        # 전체 영상 히스토그램과 roi 히스트그램 역투영 ---②
        dst = cv2.calcBackProject([hsv], [0], roi_hist, [0,180], 1)
        # 역 투영 결과와 초기 추적 위치로 평균 이동 추적 ---③
        ret, (x,y,w,h) = cv2.meanShift(dst, (x,y,w,h), termination)
```

[예제 8-28] MeanShift 객체 추적 (track_meanshift_cam.py)

Tracking

❖ Meanshift

- Example <2/3>

```
# 새로운 위치에 사각형 표시 ---④
cv2.rectangle(img_draw, (x,y), (x+w, y+h), (0,255,0), 2)
# 컬러 영상과 역투영 영상을 통합해서 출력
result = np.hstack((img_draw, cv2.cvtColor(dst, cv2.COLOR_GRAY2BGR)))
else: # 추적 대상 객체 히스토그램 등록 안됨
    cv2.putText(img_draw, "Hit the Space to set target to track", \
                (10,30),cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 1, cv2.LINE_AA)
    result = img_draw
cv2.imshow(win_name, result)
key = cv2.waitKey(1) & 0xff
if key == 27: # Esc
    break
elif key == ord(' '): # 스페이스-바, ROI 설정
    x,y,w,h = cv2.selectROI(win_name, frame, False)
    if w and h : # ROI가 제대로 설정됨
        # 초기 추적 대상 위치로 roi 설정 --- ⑤
        roi = frame[y:y+h, x:x+w]
```

Tracking

❖ Meanshift

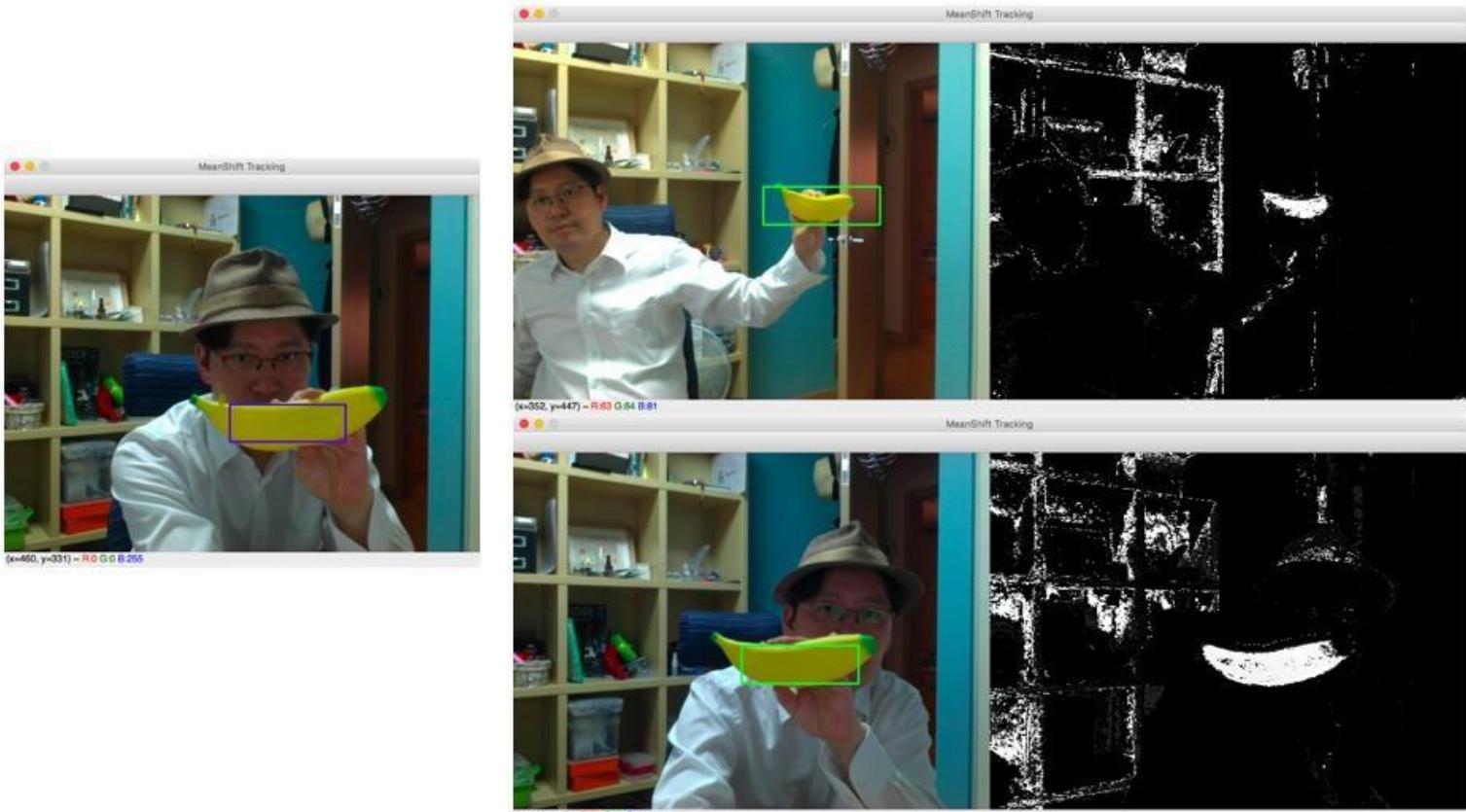
- Example <3/3>

```
# roi를 HSV 컬러로 변경 ---⑥
roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
mask = None
# roi에 대한 히스토그램 계산 ---⑦
roi_hist = cv2.calcHist([roi], [0], mask, [180], [0,180])
cv2.normalize(roi_hist, roi_hist, 0, 255, cv2.NORM_MINMAX)
else: # ROI 설정 안됨
    roi_hist = None
else:
    print('no camera!')
cap.release()
cv2.destroyAllWindows()
```

Tracking

❖ Meanshift

- Example <결과>



[그림 8-34] [예제 8-28]의 실행 결과

Tracking

❖ Camshift

- Continuously Adaptive Meanshift
- Meanshift의 윈도우 고정 문제를 개선
- meanshift를 이용해서 중심점 계산
- 윈도우 크기와 방향 재설정

❖ cv2.CamShift(img, window, criteria)

- img : 입력 영상
- window : 초기 검색 윈도우
- criteria : 검색 중지 요건

Tracking

❖ Camshift

- Example <1/3>

```
import numpy as np, cv2
roi_hist = None # 추적 객체 히스토그램 저장 변수
win_name = 'Camshift Tracking'
termination = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1)
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
while cap.isOpened():
    ret, frame = cap.read()
    img_draw = frame.copy()
    if roi_hist is not None: # 추적 대상 객체 히스토그램 등록 됨
        # 전체 영상 hsv 컬로 변환 ---①
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        # 전체 영상 히스토그램과 roi 히스트그램 역투영 ---②
        dst = cv2.calcBackProject([hsv], [0], roi_hist, [0,180], 1)
```

Tracking

❖ Camshift

- Example <2/3>

```
# 역 투영 결과와 초기 추적 위치로 평균 이동 추적 ---③
ret, (x,y,w,h) = cv2.CamShift(dst, (x,y,w,h), termination)
# 새로운 위치에 사각형 표시 ---④
cv2.rectangle(img_draw, (x,y), (x+w, y+h), (0,255,0), 2)
# 컬러 영상과 역투영 영상을 통합해서 출력
result = np.hstack((img_draw, cv2.cvtColor(dst, cv2.COLOR_GRAY2BGR)))
else: # 추적 대상 객체 히스토그램 등록 안됨
    cv2.putText(img_draw, "Hit the Space to set target to track", \
                (10,30),cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 1, cv2.LINE_AA)
    result = img_draw
cv2.imshow(win_name, result)
key = cv2.waitKey(1) & 0xff
if key == 27: # Esc
    break
```

Tracking

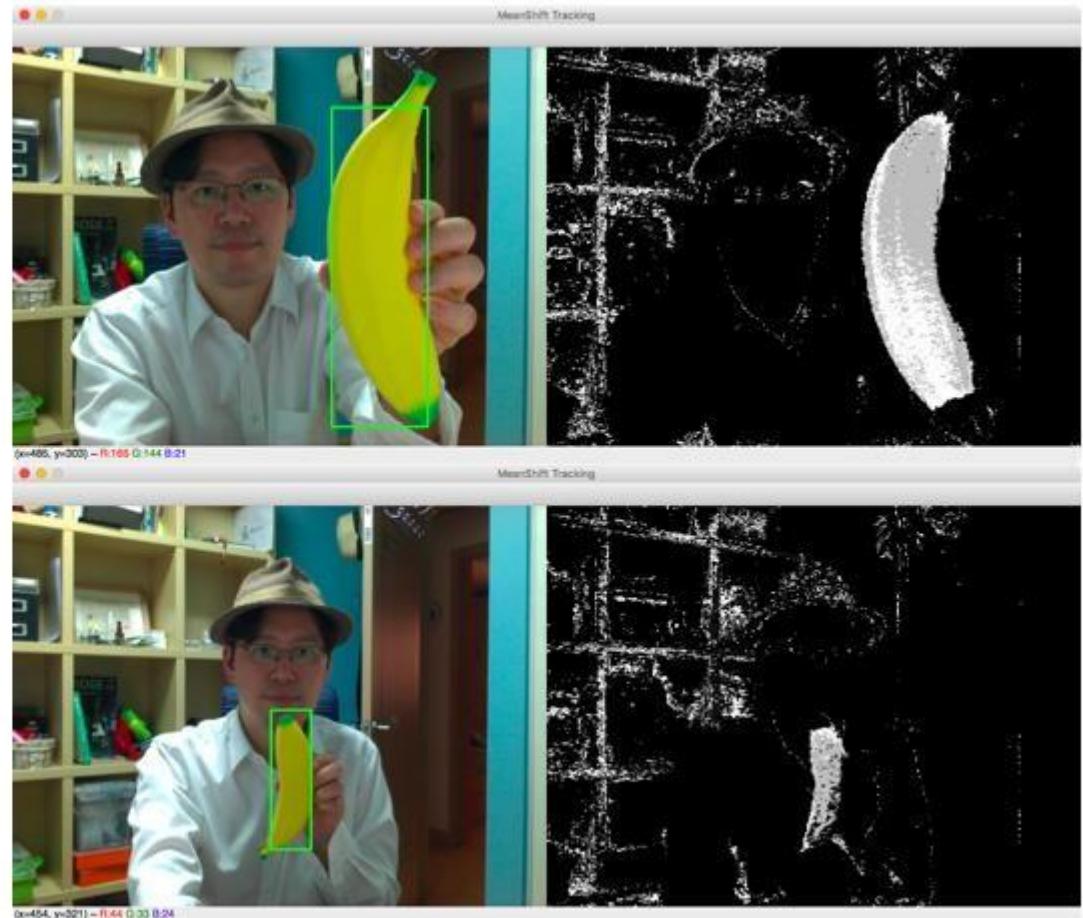
❖ Camshift

- Example <3/3>

```
elif key == ord(' '): # 스페이스-바, ROI 설정
    x,y,w,h = cv2.selectROI(win_name, frame, False)
    if w and h : # ROI가 제대로 설정됨
        # 초기 추적 대상 위치로 roi 설정 --- ⑤
        roi = frame[y:y+h, x:x+w]
        # roi를 HSV 컬러로 변경 ---⑥
        roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
        mask = None
        # roi에 대한 히스토그램 계산 ---⑦
        roi_hist = cv2.calcHist([roi], [0], mask, [180], [0,180])
        cv2.normalize(roi_hist, roi_hist, 0, 255, cv2.NORM_MINMAX)
    else: # ROI 설정 안됨
        roi_hist = None
else:
    print('no camera!')
cap.release()
cv2.destroyAllWindows()
```

Tracking

❖ Camshift



[그림 8-35] [예제 8-29]의 실행 결과

Tracking

❖ Tracking API

- OpenCV3 contrib에 추가
- 머신러닝 알고리즘 구현한 트雷킹 API
- 알고리즘 몰라도 동일한 인터페이스로 다양한 알고리즘 트랙커 선택 사용
- `retval = cv.Tracker.init(image, boundingBox)` : 트랙커 초기화
 - `image` : 입력 영상
 - `boundingBox` : 추적 대상 객체가 있는 좌표(x,y,w,h)
- `retval, boundingBox = cv.Tracker.update(image)` : 새로운 프레임에서 추적 객체 위치 찾기
 - `image` : 새로운 프레임 영상
 - `retval` : 추적 성공 여부
 - `boundingBox` : 새로운 프레임에서의 추적 대상 객체의 새로운 위치(x,y,w,h)

Tracking

❖ Tracking API

- 트래커 생성자
 - tracker = cv2.TrackerBoosting_create()
AdaBoot 알고리즘 기반
 - tracker = cv2.TrackerMIL_create() MIL(Multiple Instance Learning) 알고리즘 기반
 - tracker = cv2.TrackerKCF_create() KCF(Kernelized Correlation Filters) 알고리즘 기반
 - tracker = cv2.TrackerTLD_create()
TLD(Tracking, learning and detection) 알고리즘 기반
 - tracker = cv2.TrackerMedianFlow_create()
객체의 전방향/역방향 추적해서 불일치성 측정
 - tracker = cv2.TrackerGOTURN_create()
CNN(Convolutional Neural Networks) 기반, OpenCV 3.4 버전에 버그로 동작 안됨
 - tracker = cv.TrackerCSRT_create()
CSRT(Channel and Spatial Reliability)
 - tracker = cv.TrackerMOSSE_create()
내부적으로 그레이 스케일 사용

Tracking

❖ Tracking API

- Example <1/3>

```
import cv2
# 트랙커 객체 생성자 함수 리스트 ---①
trackers = [cv2.TrackerBoosting_create,
            cv2.TrackerMIL_create,
            cv2.TrackerKCF_create,
            cv2.TrackerTLD_create,
            cv2.TrackerMedianFlow_create,
            cv2.TrackerGOTURN_create, #버그로 오류 발생
            cv2.TrackerCSRT_create,
            cv2.TrackerMOSSE_create]
trackerIdx = 0 # 트랙커 생성자 함수 선택 인덱스
tracker = None
isFirst = True
video_src = 0 # 비디오 파일과 카메라 선택 ---②
#video_src = "../img/highway.mp4"
cap = cv2.VideoCapture(video_src)
fps = cap.get(cv2.CAP_PROP_FPS) # 프레임 수 구하기
delay = int(1000/fps)
win_name = 'Tracking APIs'
```

[예제 8-30] Tracker APIs(track_trackingAPI.py)

Tracking

❖ Tracking API

▪ Example <2/3>

```
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        print('Cannot read video file')
        break
    img_draw = frame.copy()
    if tracker is None: # 트랙커 생성 안된 경우
        cv2.putText(img_draw, "Press the Space to set ROI!!", \
                    (100,80), cv2.FONT_HERSHEY_SIMPLEX, 0.75,(0,0,255),2,cv2.LINE_AA)
    else:
        ok, bbox = tracker.update(frame) # 새로운 프레임에서 추적 위치 찾기 ---③
        (x,y,w,h) = bbox
        if ok: # 추적 성공
            cv2.rectangle(img_draw, (int(x), int(y)), (int(x + w), int(y + h)), \
                          (0,255,0), 2, 1)
        else : # 추적 실패
            cv2.putText(img_draw, "Tracking fail.", (100,80), \
                        cv2.FONT_HERSHEY_SIMPLEX, 0.75,(0,0,255),2,cv2.LINE_AA)
    trackerName = tracker.__class__.__name__
    cv2.putText(img_draw, str(trackerIdx) + ":" +trackerName , (100,20), \
                cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0,255,0),2,cv2.LINE_AA)
    cv2.imshow(win_name, img_draw)
```

Tracking

❖ Tracking API

- Example <3/3>

```
key = cv2.waitKey(delay) & 0xff
    # 스페이스 바 또는 비디오 파일 최초 실행 ---④
if key == ord(' ') or (video_src != 0 and isFirst):
    isFirst = False
    roi = cv2.selectROI(win_name, frame, False) # 초기 객체 위치 설정
    if roi[2] and roi[3]: # 위치 설정 값 있는 경우
        tracker = trackers[trackerIdx]() #트랙커 객체 생성 ---⑤
        isInit = tracker.init(frame, roi)
elif key in range(48, 56): # 0~7 숫자 입력 ---⑥
    trackerIdx = key-48 # 선택한 숫자로 트랙커 인덱스 설정
    if bbox is not None:
        tracker = trackers[trackerIdx]() # 선택한 숫자의 트랙커 객체 생성 ---⑦
        isInit = tracker.init(frame, bbox) # 이전 추적 위치로 추적 위치 초기화
elif key == 27 :
    break
else:
    print( "Could not open video")
cap.release()
cv2.destroyAllWindows()
```

Tracking

❖ Tracking API

- Example <결과>



[그림 8-36] [예제 8-30]의 실행 결과

세부목차

1. Matching with a similar image
2. Feature and Key Point
3. Descriptor Extractor
4. Feature Matching
5. Tracking
- 6. Workshop**

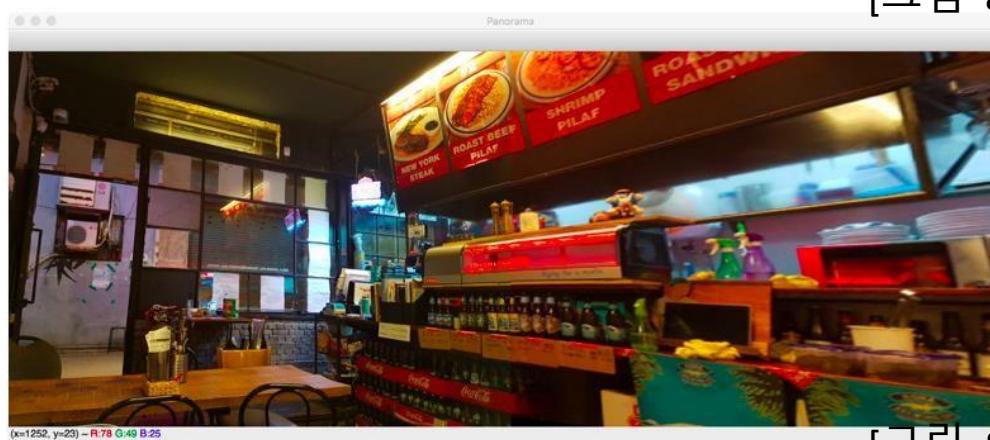
Workshop

❖ Panorama Picture

- 연속 촬영한 2개의 사진을 하나의 파노라마 사진으로 만드는 프로그램을 작성하세요.
- 결과 예시



[그림 8-37] 따로 찍은 2장의 사진



[그림 8-38] 생생한 파노라마 사진

Workshop

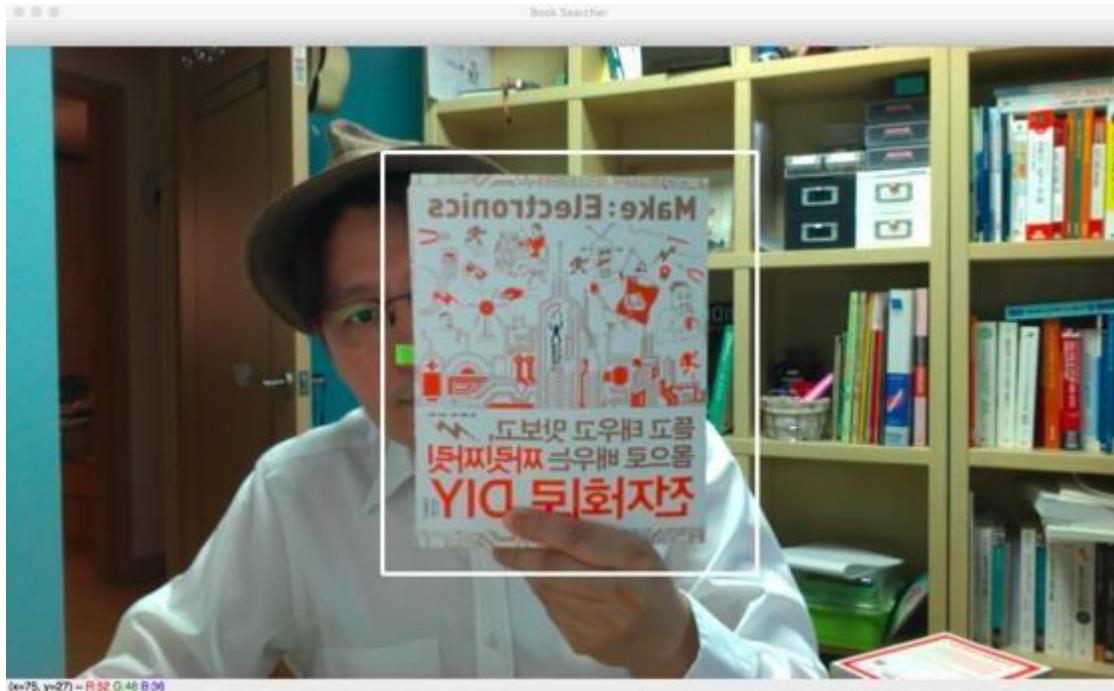
❖ Panorama Picture

- 힌트
 - 좌측 우측 연결 순서 정하고 특징점과 서술자 추출
 - 좌측 사진 기준으로 우측 사진 매칭후 원근 변환 행렬 계산
 - cv2.warpPerspective() 원근 변환
 - 2개의 영상 더한 크기 생성해서 각 자리에 합성

Workshop

❖ Book cover Searcher

- 카메라에 검색할 책 표지를 위치하고 스페이스 키를 누르면 책 표지로 검색해서 등록한 원본 이미지를 출력하세요.
- 결과 예시



[그림 8-40] 책 표지 검색기를 실행한 모습

Workshop

❖ Book cover Searcher

- 결과 예시



[그림 8-41] 검색할 책 표지와 검색 결과

Workshop

❖ Book cover Searcher

- 힌트
 - 카메라 사각 영역 표시
 - 책 표지 위치하고 스페이스 키 누르면 사각 영역만 ROI 설정 후 특징, 서술자 추출
 - 검색 대상 디렉토리 파일을 하나씩 읽어서 각각 특징점, 서술자 추출
 - 검색 이미지와 매칭 및 좋은 매칭 추출
 - 원근 변환 수행해서 정상치 비율 저장
 - 모든 매칭 끝나고 정상치 비율이 높은 순 정렬해서 가장 높은 결과 출력