Open_cv05

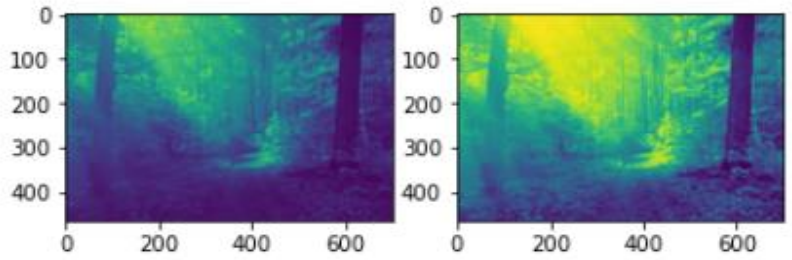
영상 특징 처리 _ feature_detection

ICT , DS&DE Y-A , KIM

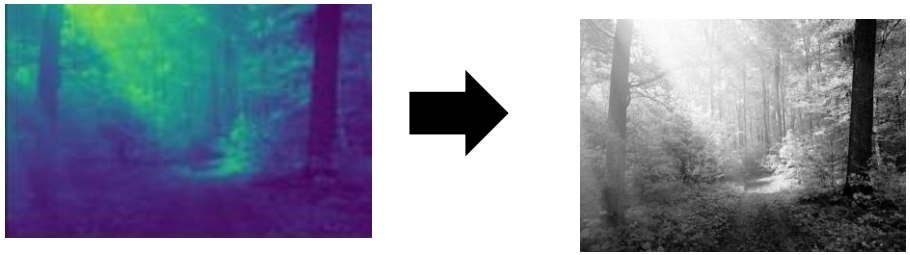
Example _ trail.jpg를 이용한 코드를 이용한 이미지 변환

Hint : cv2.equalizeHist(원본) 함수는 히스토그램의 평탄화는 편향되어 분포된 히스토그램을[0, 255]에 균등하게 되도록 하는 목적을 가진다.

1) 다음과 같이 출력하세요



2) trail.jpg 를 왼쪽 trail_res.jpg 로 변환해서 저장하세요



feature_detection

- Keypoints? : 이미지 매칭 시 사용하는 특징점
  - 이미지에서 특징이 되는 부분 이미지끼리 서로 매칭이 되는지 확인을 할 때 각 이미지에서의 특징이 되는 부분끼리 비교 하는 기능
  - 객체의 좌표뿐만 아니라 그 주변 픽셀과의 관계에 대한 정보

https://docs.opencv.org/4.x/db/d27/tutorial_py_table_of_contents_feature2d.html

feature_detection

- Keypoints 어떤 정보 ?
  - 영상이미지 size , angle , 코너(corner)점인 경우 코너의 경사도와 방향도 속성등의 대한 정보를 포함
  - Keypoints으로 **feature descriptor를 활용**

feature_detection

- 제공되는 함수는 ?

1) keypoints = detector.detect(img, mask): 특징 점 검출 함수

    img: 입력 이미지

    mask(optional): 검출 제외 마스크

    keypoints: 특징점 검출 결과 (KeyPoint의 리스트)

- Keypoints: 특징점 정보를 담는 객체

    pt: 특징점 좌표(x, y), float 타입으로 정수 변환 필요

    size: 의미 있는 특징점 이웃의 반지름

    angle: 특징점 방향 (시계방향, -1=의미 없음)

    response: 특징점 반응 강도 (추출기에 따라 다름)

    octave: 발견된 이미지 피라미드 계층

    class_id: 특징점이 속한 객체 ID

feature_detection

- 제공되는 함수는 ?

2) outImg = cv2.drawKeypoints(img, keypoints, outImg, color, flags)

- img: 입력 이미지
- keypoints: 표시할 특징점 리스트
- outImg: 특징점이 그려진 결과 이미지
- color(optional): 표시할 색상 (default: 랜덤)
- flags(optional): 표시 방법
    - cv2.DRAW_MATCHES_FLAGS_DEFAULT: 좌표 중심에 동그라미만 그림(default)
    - cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS: 동그라미의 크기를 size와 angle을 반영해서 그림)

feature_detection

- **feature descriptor_ 특징 디스크립터 ?**
  - 특징점 주변 픽셀을 일정한 크기의 블록으로 나누어 각 블록에 속한 픽셀의 그레디언트 히스토그램을 계산한 것
  - 특징점 주변의 밝기, 색상, 방향, 크기 등의 정보가 포함되어 특징 추출 알고리즘 적용

feature_detection

- 제공되는 함수는?

1) 특징점을 전달하면 특징 디스크립터를 계산해서 리턴하는 함수

keypoints, descriptors = detector.compute(image, keypoins, descriptors)

2) 특징점 검출과 특징 디스크립터 계산을 한 번에 수행하는 함수

Keypoints, descriptors = detector.detectAndCompute(image, mask, descriptors, useProvidedKeypoints)

- image: 입력 이미지
- keypoints: 디스크립터 계산을 위해 사용할 특징점
- descriptors(optional): 계산된 디스크립터
- mask(optional): 특징점 검출에 사용할 마스크
- useProvidedKeypoints(optional): True인 경우 특징점 검출을 수행하지 않음

feature_detection

- 알고리즘 종류 ?
  - SIFT : 이미지 피라미드(이미지의 크기를 피라미드처럼 단계적으로 확대하거나 축소하는 작업)를 이용해서 크기 변화에 따른 특징점 검출 문제를 해결한 알고리즘
  - SURF: 필터의 크기를 변화시키는 방식으로 성능을 개선한 알고리즘
  - ORB : 회전과 방향을 고려하도록 개선한 알고리즘

feature_detection

- SIFT (Scale-Invariant Feature Transform) ?
  - 이미지의 크기를 피라미드처럼 단계적으로 확대하거나 축소하는 작업를 이용해서 크기 변화에 따른 특징점 검출 문제를 해결한 알고리즘
  - **detector = cv2.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

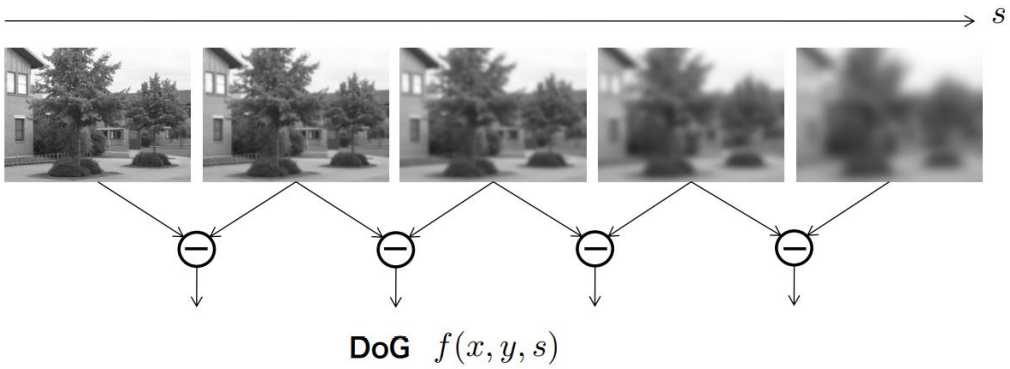
    edgeThreshold: 필터링할 엣지 문턱 값

    sigma: 이미지 피라미드 0 계층에서 사용할 가우시안 필터의 시그마 값

feature detection

- SIFT 키포인트 = 스케일 공간에서 DoG 극점
  - DoG: Difference of Gaussian
  - 스케일 자동 결정 = 스케일 불변성 획득

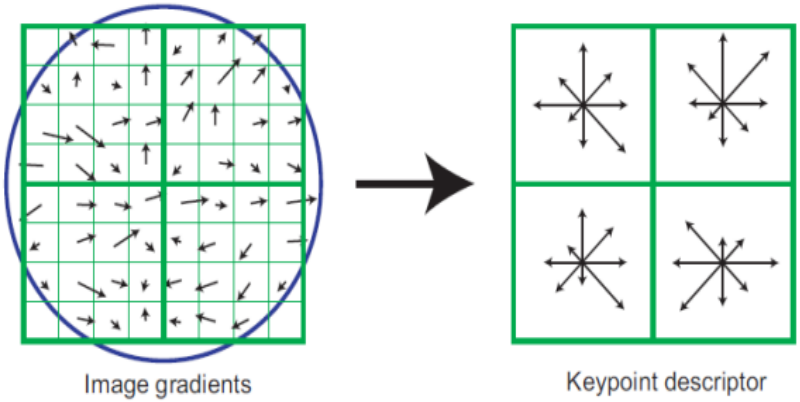
- 스케일 공간
  - 가우스 필터를 입력 이미지에 적용하여 이미지를 리턴.
  - 가우스 필터의 스케일을 변경할 때의 이미지 열



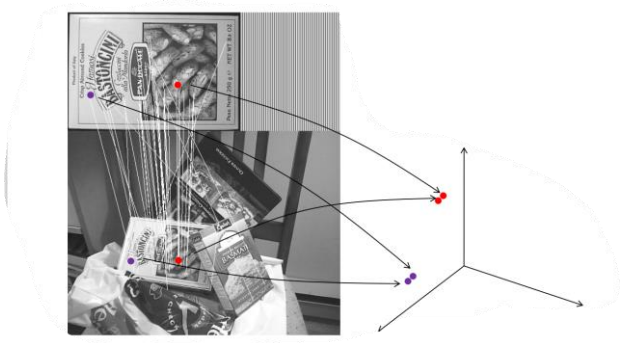
feature detection

- SIFT 주요 방향 결정
  - 360 도 방향 히스토그램의 피크 방향을 주 방향으로 선택
  - Rotation 불변성 획득

- 메인 스케일과 메인 방향으로 정규화된 직사각형 내에서 기울어진 방향의 히스 토그램
  - 위치, 방향, 크기, 128D(8방향×4×4)



Descriptor 끼리의 거리



```python
#Q2)houng.jpg를 이용한 코드를 이용한 이미지 변환SIFT
import cv2
import numpy as np

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

sift = cv2.SIFT_create()

# 키 포인트 검출 및 서술자 계산
keypoints, desc = sift.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('SIFT_draw',img_draw)
cv2.waitKey()
cv2.destroyAllWindows()
```

```
[ 12.  51. 100. ...   0.   3.  94.]
[ 89.  17.   5. ... 114. 114.  30.]
...
[  5. 120. 111. ...  39.  18.   7.]
[112.   7.   0. ...   0.   1.   5.]
[ 11.   4.   0. ...   0.   0.   1.]]
```

feature_detection

- SURF (Speeded Up Robust Features) ?
- [지원하지 않음 . OpenCV ver. 3.4.2.16]
  - 필터의 크기를 변화시키는 방식으로 성능을 개선한 알고리즘
- detector =cv2.xfeatures2d.SURF_create(hessianThreshold, nOctaves, nOctaveLayers, extended, upright)
  - hessianThreshold(optional): 특징 추출 경계 값 (default=100)
  - nOctaves(optional): 이미지 피라미드 계층 수 (default=3)
  - extended(optional): 디스크립터 생성 플래그 (default=False), True: 128개, False: 64개
  - upright(optional): 방향 계산 플래그 (default=False), True: 방향 무시, False: 방향 적용

feature_detection

- ORB (Oriented and Rotated BRIEF) ?

  : BRIEF(Binary Robust Independent Elementary Features)를 이용하여 특징점 검출을 지원하지 않는 디스크립터 추출기에 방향과 회전을 고려한 알고리즘, 회전과 방향을 고려하도록 개선했으며 속도도 빨라 SIFT와 SURF의 좋은 대안으로 사용

detector = cv2.ORB_create(nfeatures, scaleFactor, nlevels, edgeThreshold, firstLevel, WTA_K, scoreType, patchSize, fastThreshold)

nfeatures(optional): 검출할 최대 특징 수 (default=500)
scaleFactor(optional): 이미지 피라미드 비율 (default=1.2)
nlevels(optional): 이미지 피라미드 계층 수 (default=8)
edgeThreshold(optional): 검색에서 제외할 테두리 크기, patchSize와 맞출 것 (default=31)
firstLevel(optional): 최초 이미지 피라미드 계층 단계 (default=0)
WTA_K(optional): 임의 좌표 생성 수 (default=2)
scoreType(optional): 특징점 검출에 사용할 방식 (cv2.ORB_HARRIS_SCORE: 해리스 코너 검출(default), cv2.ORB_FAST_SCORE: FAST 코너 검출)
patchSize(optional): 디스크립터의 패치 크기 (default=31)
fastThreshold(optional): FAST에 사용할 임계 값 (default=20)

```python
#Q3)houng.jpg를 이용한 코드를 이용한 이미지 변환 OBR
import cv2
import numpy as np

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

obr =cv2.ORB_create()

# 키 포인트 검출 및 서술자 계산
keypoints, desc = obr.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('obr_draw',img_draw)
cv2.waitKey()
cv2.destroyAllWindows()
```

```
(500, 32) [[ 56 143  79 ... 170 116  40]
 [174 253  42 ...  61 191 222]
 [243  86 111 ... 189 116 125]
 ...
 [157 108 223 ... 243  98 169]
 [120 107  60 ... 162  70  43]
 [118 179 144 ...  14  11  22]]
```

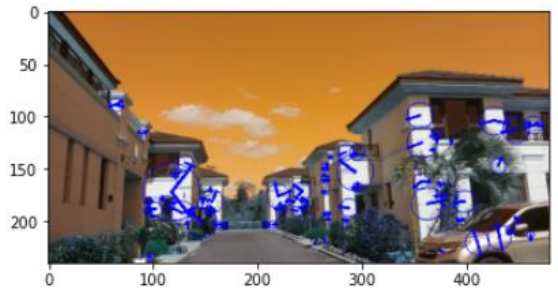
```python
#Q4) 사용자 drawKeypoints
import numpy as np
import cv2
import math
import matplotlib.pyplot as plt

def drawKeypoints(image, kp, color):
    for p in kp:
        pos = (int(p.pt[0]), int(p.pt[1]))
        pos1 = (int(p.pt[0]+p.size*math.cos(p.angle/180*math.pi)),
        int(p.pt[1]+p.size*math.sin(p.angle/180*math.pi)))
        scale = int(p.size)
        cv2.circle(image, pos, scale, color=(0,0,255), thickness=1)
        cv2.line(image, pos, pos1, color=(0,0,255), thickness=2)

sift = cv2.SIFT_create(100) # Num of keypoints
#sift = cv2.ORB_create(100)

image = cv2.imread('house.jpg')# cap.read(0)
image = cv2.resize(image,   ((int)(image.shape[1]/2),(int)(image.shape[0]/2)))
kp = sift.detect(image)
kp, des = sift.compute(image, kp)
drawKeypoints(image, kp, color=(255,0,0))
plt.imshow(image)
plt.show()

```



Example _

cv.SIFT_create(), fly.jpg를 이용한 코드를 이용한 특징 추출

① 이미지 로드

② 그레이 스케일 변환

③ cv.SIFT_create() 생성

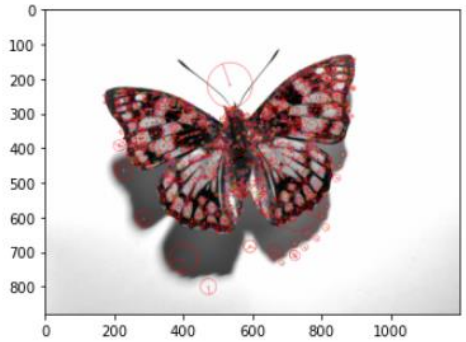
④ 키포인트 검출

⑤ 키포인트 그리기

⑥ sift_keypoints.jpg 파일저장

⑦ 출력될 키포인트 그리기 img02=cv.drawKeypoints(img,kp,None,(255,0,0),4)

⑧ 이미지 출력 plt.imshow(img2)



출력 결과

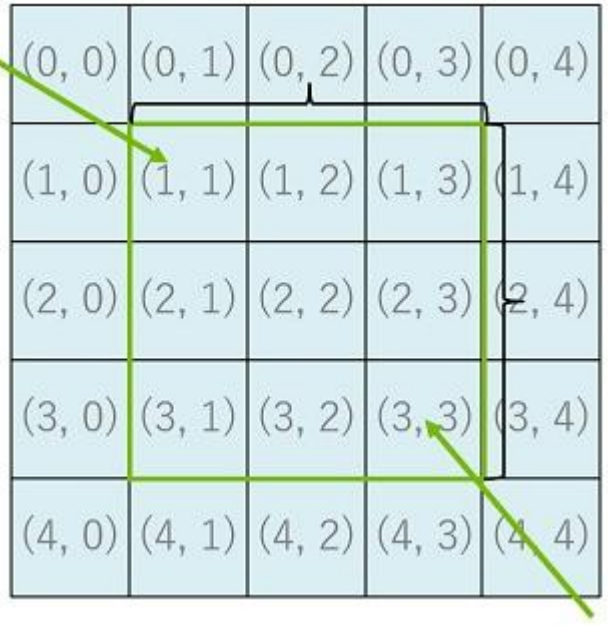
저장된 파일

Non Maximum Suppression

- 물체 감지를 수행하면 하나의 물체에 대해 여러 번 감지될 수 있다.
- 물체 검출은 중복 검출 결과를 하나로 통합하는 Non Maximum Suppression이라는 후처리가 자주 사용된다.
- ($x1$,$y1$) ,($x2$,$y2$)에서 (w, h를 얻으려면, w=$x2$– $x1$+1 ,h=$y2$– $y1$+1

왼쪽 위 좌표($x1$,$y1$) =1,1

w=3



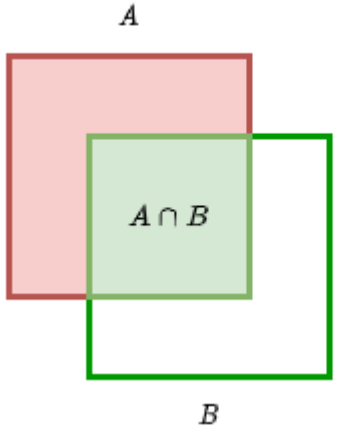
사각형 측면

h =3

오른쪽 하단 좌표($x2$,$y2$) = (3,3)

Overlap Ratio

- 두 개의 직사각형 a,b 가 있을 때에서, $\frac{area(a \cap b)}{area(a)}$ 계산할 수 있는 값을 **Overlap Ratio** 라고 한다.



$$OverlapRatio = \frac{A \cap B}{A}$$

① 중복으로 판단하여 삭제할 Overlap Ratio 임계값[0, 1][0 ,1]의 범위에서 적절한 값으로 결정.

② 값이 클수록 중복이라고 판단하는 기준이 엄격해져, 동일 물체에 복수의 구형이 남아 버릴 가능성이 있다.

③ 반대로 값이 낮을수록 중복으로 판단하는 기준이 느려지고 다른 물체를 나타내는 직사각형이 동일한 물체를 나타내는 것으로 판단되어 삭제될 수 있다

IOU(Intersection Over Union)활용예시

- Object Detector에 대한 성능 평가

  -일반적으로 두 사각형 중 하나는 GT(Ground Truth), 나머지 하나는 측정한 값으로 IOU를 구하여 성능을 평가

  -Object detector에 대한 성능 평가에서 AP50, AP75등으로 평가한 방법 중 이는 IOU가 0.5이상인 경우와 IOU가 0.75이상 일치한 경우를 평가한 것

- NMS(Non Maximum Suppression)

  -일부 Object detector는 여러 개의 검출 영역(bounding box)들을 생성하며 그 중에 클래스가 같으면서 겹치는 검출 영역 중에 가장 정확도(confidence)가 높은 영역만 남기고 나머지는 제거하는 방법

  -어느 정도 겹치는 영역들에 대해 제거 작업을 할지 선택하는 척도에 IOU를 사용하기도 한다.

Non Maximum Suppression 처리

1. 입력에서 점수가 가장 높은 직사각형을 선택하고 출력으로 이동
2. 선택한 사각형과 입력에 남아있는 각 사각형의 **IOU(Intersection Over Union)**를 계산하여 임계값 이상의 것을 입력에서 삭제한다.
3. 입력이 비워질 때까지 1, 2 반복
4. 입력이 비어 있으면 출력의 사각형을 결과 출력으로 설정한다.

0.9	img1
0.7	img2
0.5	img3
0.8	img4
0.6	img 5