COMPUTER VISION Lab 08

-SIFT-

과목명 : [CSI4116-01] COMPUTER VISION
담당교수 : 연세대학교 컴퓨터공학과 변혜란
학번 : S20131598
이름 : 전상현

REPORT

I. 개발 목표

- SIFT(Scale-Invariant Feature Transform)은 이미지의 크기와 회전에 불편하는 특징을 추출하는 알고리즘 중 하나이다. SIFT 알고리즘을 이용하여 서로 다른 두 이미지의 SIFT 특징점을 각각 추출한 다음에 서로 가장 비슷한 특징끼리 매칭해주면 두 이미지에서 대응되는 부분을 찾을 수 있다는 것이 알려져 있다. 본 과제에서는 Python의 OpenCV에서 기본적으로 제공하는 SIFT 알고리즘을 통하여 서로 다른 두 이미지에서 동일한 물체를 찾아서 매칭해 보도록 한다.

II. 개발 범위 및 내용

가. 개발 범위

1. SIFT

나. 개발 내용

1. SIFT

- OpenCV에서 제공하는 SIFT 함수를 통하여 서로 다른 두 이미지의 특징점을 추출한다.
- 추출한 특징점을 각각 두 가지 방법을 이용하여 매칭해본다
    a. Brute Force
    b. FLANN
- 두 가지 방법을 통하여 비교한 결과값을 서로 비교한다.

III. 결과

1. 제작 내용

__main__

```
if __name__ == "__main__":
    #img1 = cv2.imread('box.png', 0)  # queryImage
    #img2 = cv2.imread('box_in_scene.png', 0)  # trainImagepng

    img1 = cv2.imread('monalisa.png', 0)  # queryimage
    img2 = cv2.imread('monalisa_scene.jpg', 0)  # trainimage

    SIFT_with_BF(img1, img2)
    SIFT_with_FLANN(img1, img2)
```

- 두 개의 다른 이미지를 입력받아 각각 다른 방법(Brute Force, FLANN based)을 이용하여 이미지를 비교한다.

SIFT_with_BF(Brute Force)

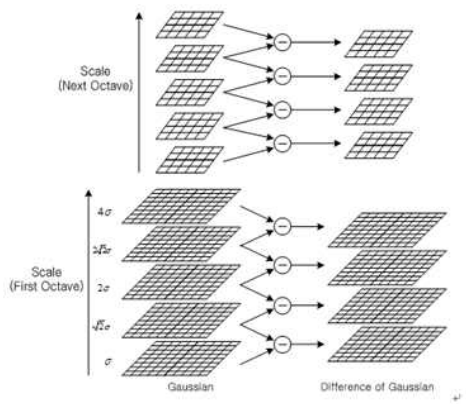
```
def SIFT_with_BF(queryimg, trainimg):
    start = time.time()

    # Initiate SIFT detector
    sift = cv2.xfeatures2d.SIFT_create()

    # find the keypoints and descriptors with SIFT
    kp1, des1 = sift.detectAndCompute(queryimg, None)
    kp2, des2 = sift.detectAndCompute(trainimg, None)

    # BFMatcher with default params
    bf = cv2.BFMatcher()

    matches = bf.knnMatch(des1, des2, k=2)

    matchesMask = [[0, 0] for i in range(len(matches))]

    for i, (m,n) in enumerate(matches):
        if m.distance < 0.75 * n.distance:
            matchesMask[i] = [1,0]

    draw_params = dict(singlePointColor=(255, 0, 0),
                       matchesMask = matchesMask,
                       flags=2)

    # cv2.drawMatchesKnn expects list of lists as matches
    img = cv2.drawMatchesKnn(queryimg, kp1, trainimg, kp2, matches, None, **draw_params)
    print("BF time : ", time.time() - start)

    plt.imshow(img),plt.show()
```

SIFT_with_FLANN

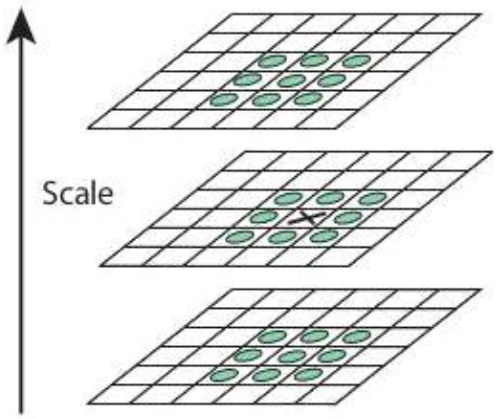
```
def SIFT_with_FLANN(queryimg, trainimg):
    start = time.time()

    # Initiate SIFT detector
    sift = cv2.xfeatures2d.SIFT_create()

    # find the keypoints and descriptors with SIFT
    kp1, des1 = sift.detectAndCompute(queryimg, None)
    kp2, des2 = sift.detectAndCompute(trainimg, None)

    #FLANN parameters
    FLANN_INDEX_KDTREE = 0
    index_params = dict(algorithm = FLANN_INDEX_KDTREE, tress = 5)
    search_params = dict(checks=50) # or pass empty dictionary

    flann = cv2.FlannBasedMatcher(index_params, search_params)
    matches = flann.knnMatch(des1, des2, k=2)

    # Need to draw only good matches, so create a mask
    matchesMask = [[0, 0] for i in range(len(matches))]

    # ratio test as per Lowe's paper
    for i, (m,n) in enumerate(matches):
        if m.distance < 0.75 * n.distance:
            matchesMask[i] = [1,0]

    draw_params = dict(singlePointColor = (255, 0, 0),
                       matchesMask = matchesMask,
                       flags = 2)

    img = cv2.drawMatchesKnn(queryimg, kp1, trainimg, kp2, matches, None, **draw_params)
    print("FLANN time : ", time.time() - start)

    plt.imshow(img),plt.show()
```

기본적으로 OpenCV에서 제공하는 SIFT를 이용하여 특징점들을 검출하였다. SIFT 알고리즘의 기본적인 진행 **순서**는 다음과 같다.

1. Scale-space extrema detection



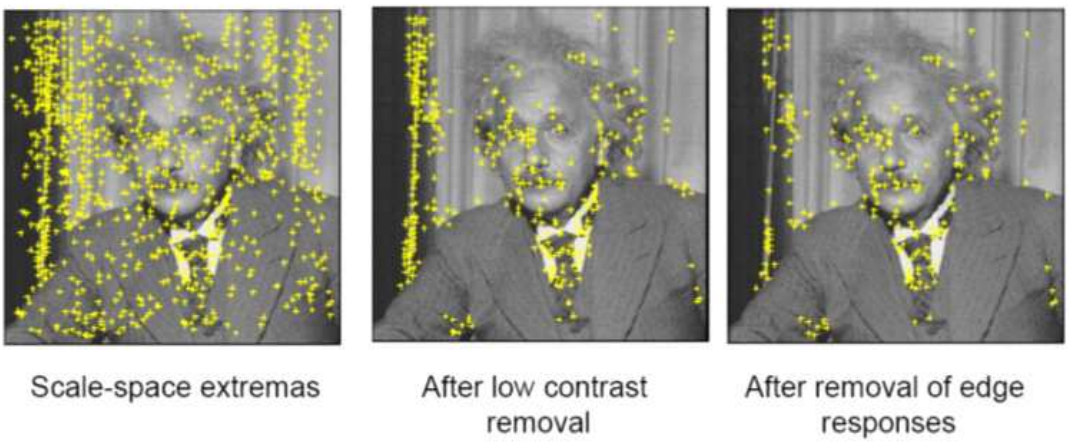
[그림 1] SIFT의 Scale별 DoG 계산 과정 예시



[그림2] SIFT의 특징점 후보지 검출 과정 예시

Scale-space extrema detection 단계에서는 [그림 1]과 같이 Gaussian Pyramid*를 생성한 후 DoG(Difference of Gaussian)을 구해서 극점인 부분을 특징점의 후보지로 잡아준다. 그 후, [그림 2]와 같이 각 DoG의 상위와 하위 층을 포함하여 주변 26개의 값보다 모두 크거나 모두 작을 때만 특징점의 후보지로 검출한다. 위와 같은 방법을 통함으로서 SIFT는 실제 Target 물체가 작아져도 인식할 수 있는 장점을 가지고 있다.

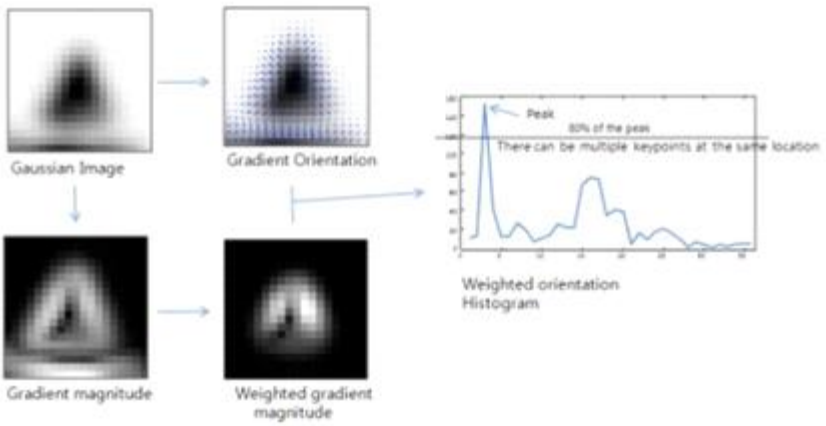
2. Keypoint Localization



[그림 3] Keypoint Localization 예시

Keypoint Localization 단계에서는 테일러 급수를 사용하여 더 정확한 극점을 찾은뒤 테일러 급수로 Interpolation을 해서 그 점의 값을 구해 특징점 후보지 중에서 정확하지 않은 특징점을 제거한다.

3. Orientation assignment

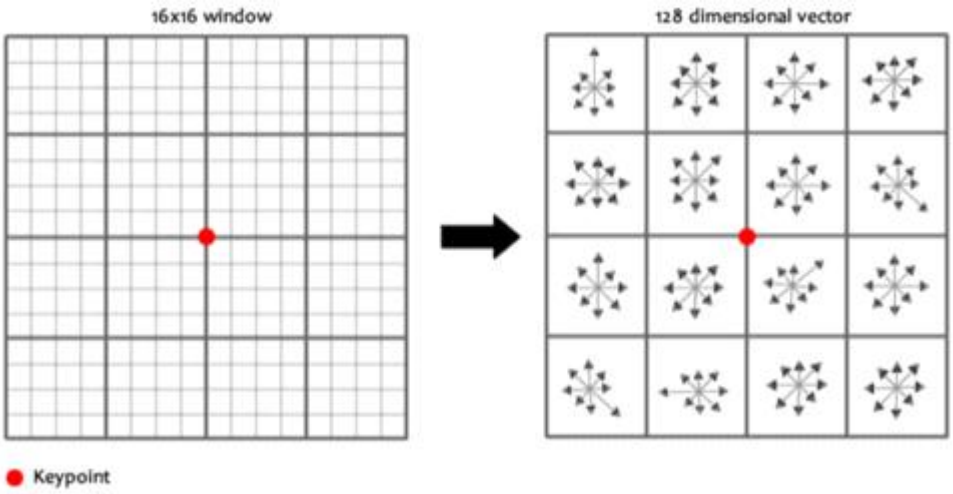


[그림 4] Gradient 방향과 크기를 추출하여 Orientation을 구하는 과정 예시

Orientation assignment 단계에서는 추출된 특징점에 대하여 주 방향을 할당해 준다. 특징점 주변으로 16x16 영역을 할당한 후 그 해당 영역에 Gaussian Blurring을 적용함으로서 특징점 주변의 영역에 대한 Gradient의 방향과 크기를 결정할 수 있다. 해당

Gradient의 방향과 크기를 통하여 Orientation Histogram을 형성한 후, 가장 값이 큰 것을 해당 Orientation으로 결정한다. 이 과정을 통하여 특징점이 Orientation에 대한 정보를 가지게 됨으로서 SIFT는 물체가 회전해도 인식할 수 있는 장점을 가지고 있다.

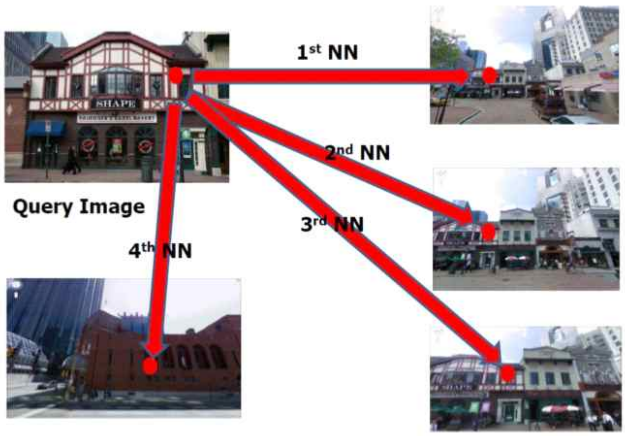
4. Keypoint descriptor



[그림 5] Keypoint descriptor 계산 예시

Keypoint Descriptor 단계에서는 각 특징점들마다 Descriptor를 생성한다. 추출된 특징점 주변으로 16x16 의 윈도우를 세팅하는데, 이 윈도우는 각각 4x4의 작은 윈도우 16개로 구성이 된다. 작은 윈도우는 각각 8개의 bin histogram을 가지게 된다. 즉, 하나의 특징점에 대해 128개의 차원을 가지게 된다. 이때 회전에 불변인 Descriptor를 가지기 위하여 특징점의 Orientation을 각각의 Descriptor에서 빼준다. 즉 Descriptor가 특징점의 Orientation에 상대적이게 되므로 물체가 회전하는 경우에도 항상 같은 Descriptor를 가질 수 있다.

5. Keypoint Matching



[그림 6] Keypoint Matching 수행 예시

Keypoint Matching 단계에서는 서로 다른 두 이미지가 존재할 때 추출한 특징점에 대하여 거리(ex. 유클리드 거리)를 구한다. 그 중 가장 가까운 점이 Matching된 점이라고 간주하는 방법이다. 단 가장 가까운 거리와 두 번째로 가까운 거리의 비율이 일정 threshold 미만일 경우에 대해서만 matching 했다고 간주한다(Ratio test)

a. Brute Force

Brute Force는 추출된 모든 특징점에 대하여 모든 점들을 모두 검사한다.

b. FLANN based match

FLANN(Fast Library for Approximate Nearest Neighbors)은 고차원 공간에 대하여 가장 가까운 neighbor를 빠르게 찾을 수 있는 library이다. 여기서는 kd-tree를 이용하는 방법을 사용하였다.

2. 시험 및 평가 내용

A. Accuarcy

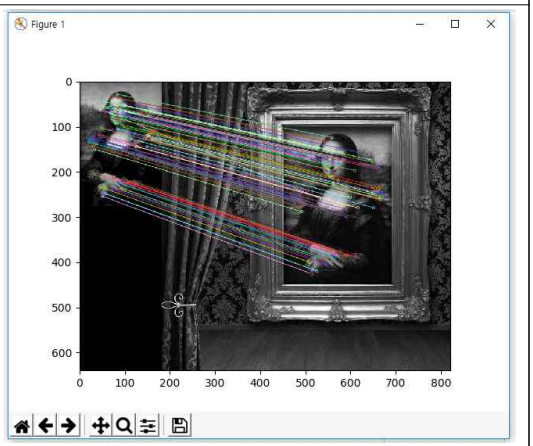
Brute Force	FLANN based
	

표 1. 수행 결과 예시 비교표

B. Time Consumption

Algorithm	1회	2회	3회	4회	5회	평균(sec)
Brute Force	0.344729	0.339881	0.276768	0.424078	0.281228	0.333337
FLANN based	0.469213	0.377447	0.485585	0.485086	0.394813	0.442429

표 2. 수행 시간 측정 예시 결과표

두 가지 방법(Brute Force, FLANN based)을 이용하여 서로 다른 두 이미지를 매칭해 보았다. 실제 출력된 결과값을 통해 두 방법이 차이 없이 같은 특징점을 같은 특징점에 **정확히** 매칭함을 알 수 있었다.

또한, 두 가지 방법을 이용하여 결과값을 출력하는데 걸린 시간을 측정해 보았다. 기본적으로는 두 가지 방법 모두 0.5초 미만의 짧은 시간안에 수행이 완료되었다. 실제로

FLANN based Matching을 이용한 경우가 Brute Force를 이용한 경우보다 약간 더 오래 걸림을 확인 할 수 있었는데, 실제로 특징점의 개수가 1K가 넘는 큰 이미지의 경우에는 FLANN 방법을 사용하는 것이 Brute Force를 사용하는 것보다 빠르다는 사실을 검색을 통하여 확인하였다. 단 여기서는 SIFT를 적용가능한 큰 image data set을 구하지 못해서 실제로 확인해 보지는 못하였다.

Ⅳ. 기타

1. 느낀 점

수업시간에 다루었던 SIFT에 대해서 직접 구현해 보았다. SIFT 알고리즘 자체는 매우 어렵고 이해하기 어려웠으나 Python의 OpenCV에서 제공하는 SIFT 함수를 이용하여 간단히 구현할 수 있었다. 실제 찾은 특징점을 매칭하는 과정에서 Brute Force와 FLANN based의 두 가지 방법을 이용하여 특징점을 매칭해 보았다. 실제 알아본 결과로는 이미지의 크기가 커지면, 즉 특징점의 개수가 약 1K가 넘어가면 FLANN 방법을 사용하였을 때 더 효율적임을 알게 되었다. 하지만 실제 해당 조건을 만족하는 이미지를 찾지 못해서 아쉬웠다. 추후에는 SIFT의 내부 구조에 대해 공부하고 알아볼 수 있는 기회를 가지기 위해서 노력해 보아야겠다.