OpenCV feature matching 알고리즘 + 파라미터 튜닝 Test

날짜	@2023년 10월 12일 → 2023년 10월 13일
태그	

OpenCV feature matching (SIFT + FLANN)

- 언어 : Python (예제) / C++ (iOS용 앱)
- Reference
  - https://docs.opencv.org/3.4/d5/d6f/tutorial_feature_flann_matcher.html
  - https://bkshin.tistory.com/entry/OpenCV-28-특징-매칭Feature-Matching

Feature Matching 이란?

두 개의 이미지간 유사도를 측정

1. 이미지에서 의미있는 특징들 추출하여 숫자로 변환
   - 특징을 대표하는 숫자 : Feature vector (Feature Descriptor)
2. 이 숫자들을 비교해서 얼마나 비슷한 지 판단

특징점 (Keypoints)

이미지에서 특징이 되는 부분. 주로 물체의 모서리나 코너를 특징점으로 잡는다.

특징 디스크립터 (feature descriptor)

특징점은 객체의 좌표뿐만 아니라 그 주변 픽셀과의 관계에 대한 정보를 가진다.

대표적으로 size와 angle 속성 + 코너(corner)점은 코너의 경사도와 방향까지

특징 디스크립터(feature descriptor) : 특징점 주변 픽셀을 일정한 크기의 블록으로 나누어 각 블록에 속한 픽셀의 그레디언트 히스토그램을 계산한 것. 주로 특징점 주변의 밝기, 색상, 방향, 크기 등의 정보가 포함.

추출하는 알고리즘에 따라 특징 디스크립터가 일부 달라질 수는 있음.
일반적으로 특징점 주변의 블록 크기에 8방향(상, 하, 좌, 우 및 네 방향의 대각선) 경사도를 표현

- 4 x 4 크기의 블록인 경우 한 개의 특징점당 4 x 4 x 8 = 128개의 값

Q1. 빨간색 부분 무슨 말인지 모르겠다… 그림으로 봐야 정확히 이해할 듯

OpenCV에서 제공하는 디스크립트 함수

- **keypoints, descriptors = detector.compute(image, keypoins, descriptors)**
  - 특징점을 전달하면 특징 디스크립터를 계산해서 반환
- **keypoints, descriptors = detector.detectAndCompute(image, mask, decriptors, useProvidedKeypoints)**
  - 특징점 검출과 특징 디스크립터 계산을 한 번에 수행
  - 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

이미 특징점을 검출한 경우에는 detector.compute() 함수를 사용해서 특징 디스크립터 구하기detector.compute() 함수의 keypoints 파라미터에 특징점을 전달해주면 됨.

detector.detectAndCompute() 함수는 특징점과 특징 디스크립터를 동시에 계산
특징점 검출기로 특징점을 한번 검출하고, 그다음 detector.compute() 함수를 사용하는 것보다 처음부터 detector.detectAndCompute() 함수를 사용하는 게 더 편리함

→ SIFT, SURF, ORB는 모두 특징 디스크립터를 구해주는 알고리즘

SIFT (Scale-Invariant Feature Transform)

SIFT는 이미지 피라미터를 통해 크기 변화에 따른 특징점 검출 문제를 해결
즉, 크기 변화, 회전, 조명 변화 등에 강인한 특징을 찾는 특징 추출 알고리즘

1. **스케일 스페이스에서 키포인트 탐지 (Scale Space Extrema Detection) : Scale-Invariant**
    - 이미지를 여러 다른 크기로 축소 또는 확대하여 스케일 스페이스를 생성
        - 신호처리 관점에서는 Image에 Gaussian filter를 convolution한 것과 scale을 키우는 것은 완전히 동일한 과정임
        - 한 이미지에서 여러 scale의 이미지를 얻고 싶다면 다양한 scale(sigma)의 Gaussian Filter를 Convolution하면 됨
    - 각 스케일에서 LoG (Laplacian of Gaussian) 필터를 적용하여 이미지의 가우시안 블러 버전 생성
        - LoG는 많은 연산을 요구하기 때문에, 비교적 간단하면서도 비슷한 성능을 낼 수 있는 Difference of Gaussian (DoG) 사용
        - LoG, DoG 모두 이미지에서 엣지 정보, 코너 정보를 도출할 때 널리 사용되는 방법
    - 각 픽셀에 대해 주변 픽셀들과 비교하여 극소값 또는 극대값을 찾아내고, 이를 키포인트로 선정
        - 대략적인 극값 : 내 주변 8개 내 위아래 9개 총 26개 픽셀보다 Value가 크거나 작으면 극값
        - DoG에 대한 테일러급수를 X에 대해 미분해서 0이 되는 지점이 극값

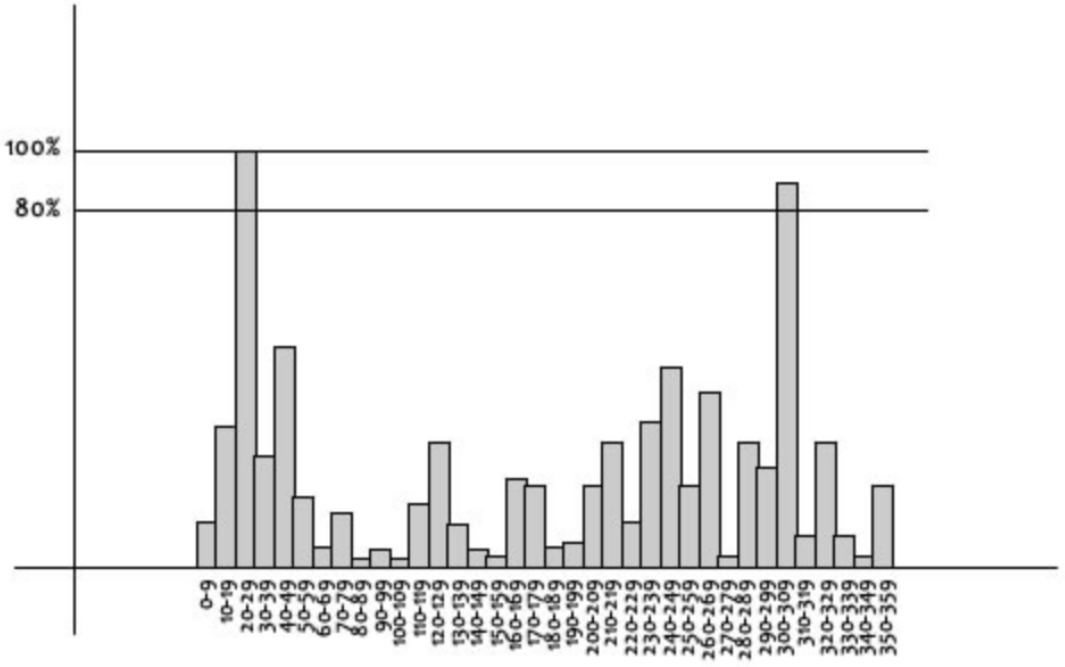
$$D(\mathbf{x}) = D + \frac{\partial D}{\partial \mathbf{x}}^T \mathbf{x} + \frac{1}{2}\mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}$$

        - D는 DoG 이미지를 나타내고, X는 (x, y, standard deviation)
    - 나쁜 keypoint를 제거하기 위해 keypoint 중에서 특정 threshold보다 낮은 keypoint를 제거하고 edge위에 위치한 noise keypoint들 또한 제거하여 **코너에 위치한 keypoint들만 남겨놓음…**
2. **키포인트 주변 영역에서 특징 벡터(Descriptor) 생성 : Rotate-Invariant**

- 각 키포인트에 대해 주변 영역을 정의하고, 이 영역에서의 그래디언트 방향과 크기 정보를 추출

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$
$$\theta(x,y) = \text{atan2}(L(x,y+1) - L(x,y-1), L(x+1,y) - L(x-1,y))$$

- 이 정보를 사용하여 키포인트를 설명하는 고유한 특징 벡터(Descriptor) 생성
  - Gradient의 크기와 방향을 구하고 가로축이 방향, 세로축이 크기인 Histogram을 그림
    아래의 Histogram에서 가장 큰 값을 가지는 방향(각도)을 keypoint의 방향으로 설정
    만약 가장 큰 keypoint 방향의 80%보다 큰 각도가 존재한다면, 그 각도도 keypoint의 orientation으로 설정



SIFT의 장점은 크기와 회전에 대한 불변성이 있어 다양한 조건에서 강력한 성능을 발휘할 수 있음. 그러나 계산 비용이 높다는 게 단점

OpenCV의 SIFT 객체 생성자

- **detector = cv2.xfeatures2d.SIFT_create(nfeatures, nOctaveLayers, contrastThreshold, edgeThreshold, sigma)**

- nfeatures: 검출 최대 특징 수n
- OctaveLayers: 이미지 피라미드에 사용할 계층 수
- contrastThreshold: 필터링할 빈약한 특징 문턱 값
- edgeThreshold: 필터링할 엣지 문턱 값sigma: 이미지 피라미드 0 계층에서 사용할 가우시안 필터의 시그마 값

```
# SIFT로 특징점 및 디스크립터 추출(desc_sift.py)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()
```

FLANN matcher object

- OpenCV Document : https://docs.opencv.org/3.4/d5/d6f/tutorial_feature_flann_matcher.html

```
// Create a FLANN matcher object
cv::FlannBasedMatcher flann;

// Use KNN matching to find the two nearest matches for each descriptor
std::vector<std::vector<cv::DMatch>> knnMatches;
flann.knnMatch(descriptors1, descriptors2, knnMatches, 2);
```

BFMatcher의 단점 : 모든 디스크립터 조사 → 이미지 사이즈가 클 경우 속도가 굉장히 느림

이를 해결하기 위해 FLANN을 사용

FLANN : **모든 디스크립터를 전수 조사하기 보다 이웃하는 디스크립터끼리 비교**
이웃하는 디스크립터를 찾기 위해 FLANN 알고리즘 함수에 **인덱스 파라미터**와 **검색 파라미터**를 전달해야함

Q1. 인덱스 파라미터랑 검색 파라미터가 뭔가? 무슨 역할을 하는가?

```
//Python

cv2.FlannBasedMatcher()
```

인덱스 파라미터로 **indexParams,** 검색 파라미터로 **searchParams**를 전달받음.
두 파라미터 모두 딕셔너리 형태.

Q2. C++에서도 딕셔너리 형태로 전달하는가? 전달한다면 key:Value pair는 뭐가 되는 것임?

indexParams: 인덱스 파라미터 (딕셔너리)

- **algorithm:** 알고리즘 선택 키, 선택할 알고리즘에 따라 종속 키를 결정하면 됨

  FLANN_INDEX_LINEAR=0: 선형 인덱싱, BFMatcher와 동일

  FLANN_INDEX_KDTREE=1: KD-트리 인덱싱 (trees=4: 트리 개수(16을 권장))

  FLANN_INDEX_KMEANS=2: K-평균 트리 인덱싱 (branching=32: 트리 분기 개수, iterations=11: 반복 횟수, centers_init=0: 초기 중심점 방식)

  FLANN_INDEX_COMPOSITE=3: KD-트리, K-평균 혼합 인덱싱 (trees=4: 트리 개수, branching=32: 트리 분기 새수, iterations=11: 반복 횟수, centers_init=0: 초기 중심점 방식)

  FLANN_INDEX_LSH=6: LSH 인덱싱 (table_number: 해시 테이블 수, key_size: 키 비트 크기, multi_probe_level: 인접 버킷 검색)

  FLANN_INDEX_AUTOTUNED=255: 자동 인덱스 (target_precision=0.9: 검색 백분율, build_weight=0.01: 속도 우선순위, memory_weight=0.0: 메모리 우선순위,

sample_fraction=0.1: 샘플 비율)

searchParams: 검색 파라미터 (딕셔너리)

- **searchParams:** 검색 파라미터 (딕셔너리)

  checks=32: 검색할 후보 수

  eps=0.0: 사용 안 함
  sorted=True: 정렬해서 반환

<u>*주로 성능이 잘 나오는 페어</u>

<SIFT나 SURF를 사용하는 경우>

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

<ORB를 사용하는 경우>

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