1.텍스트, 폰트, 스크린샷, 문서이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

요구사항에 나온 코드를 그대로 사용하면 구현할 수 있다.  
수업 자료에 나온 예제 코드를 참조할 수도 있다.  
텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

예제코드를 확인하면 SIFT\_create() 이외의 사용하는 매개변수를 확인할 수 있고  
SIFT\_create()의 매개변수는 다음과 같다

nfeatures=1000, nOctaveLayers=3, contrastThreshold=0.09, edgeThreshold=10, sigma=1.5

nfeatures : 최대 검출할 특징점 수 (0이면 무제한)

nOctaveLayers : 옥타브 당 레이어 수 (기본값: 3)

contrastThreshold : 대비 임계값 (작을수록 더 많은 특징점, 하지만 노이즈 가능성 증가, 공식 사이트 document에선 특정 논문이 0.09를 사용하고 있다고 첨언함.)

edgeThreshold : 엣지 필터링 임계값 (작을수록 엣지 근처 특징 제거 강화)

sigma : 초기 이미지 블러의 Gaussian sigma 값 (기본값: 1.6)

이후 matplotlib으로 시각화를 해주면 1번 과제가 해결된다.육상 차량, 야외, 차량, 건물이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

2.

텍스트, 폰트, 스크린샷, 문서이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

Flann 라이브러리 사용 코드텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.  
  
수업 자료에 FLANN 라이브러리 사용 코드가 있으나 사용 함수가 다르므로 matcher부분은 따로 공부 해야한다.  
<자료 코드>  
텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.  
텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

cv.FlannBasedMatcher(indexParams, searchParams)

각각의 파라미터는   
index\_params :

어떤 데이터 구조(트리)를 사용할지 정의 알고리즘 번호와 관련 매개변수들을 딕셔너리 형태로 지정 하여 사용한다. (dict(algorithm=1, trees=5) 식으로 선언 가능.)

searchParams: 검색 시 설정할 조건들

트리 탐색 시 고려할 노드 수로 마찬가지로 딕셔너리 형태로 지정.  
dict(checks=50) 식으로 선언 가능합니다.

힌트에 나온 최근접 이웃 거리 비율을 사용하기 위해 knnMatch()와 DMatch 객체의 설명을 추가로 적어보자면,

matches = matcher.knnMatch(des1, des2, k=2) 형태로   
des1, des2 : 두 이미지의 디스크립터 (예: SIFT의 des)

k : 각 디스크립터마다 찾을 최근접 이웃 개수로 2를 사용하여 최근접 이웃 거리 비율 사용.

knnMatch()의 결과물이 DMatch가 되며 속성에는

|  |
| --- |
| queryIdx:첫 번째 이미지에서의 디스크립터 인덱스  trainIdx :두 번째 이미지에서의 디스크립터 인덱스  distance:두 디스크립터 간 거리 (작을수록 유사)  가 있다.  최근접 이웃 거리 비율 구현 및 cv.drawMatches()는 예시 자료에서 참고할 수 있다. |

이후 matplotlib로 시각화 한다.

BFMatcher의 경우

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

스크린샷, 운동장, 예술, 야외이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

BFMatcher의 경우 모든 디스크립터 쌍을 하나하나 비교하여 가장 가까운 매칭을 직접 찾는 방식으로 작동한다고 하며 힌트에서 나온 코드의 경우 방식을 설정하는 초기화에 해당한다.

이후 다음 함수들을 사용하여 매칭을 수행한다.  
matches = bf.match(des1, des2)

matches = sorted(matches, key=lambda x: x.distance)

스크린샷, 건물, 예술, 야외이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다. 텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

텍스트, 스크린샷, 문서, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

Img1, img2를 선택했다.

요구사항 SIFT\_create()  
cv.BFMatcher는 직전까지 했으므로 생략하고, 이후 findHomography()는 자료에 사용법이 나와있으므로 유용하게 사용할 수 있다.

텍스트, 스크린샷, 폰트, 웹 페이지이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다. 텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

이후 .warpPerspective() 함수는 cv.warpPerspective(src, M, dsize) 형태로 각 파라미터는

src: 입력 이미지 (img2)

M: 3x3 변환 행렬 (아까 계산한 Homography 사용)

dsize: 출력 이미지의 크기 (요구사항에서 원본 이미지와 같을것을 주문함)

이후 cv.threshold(src, thresh, maxval, type) 함수로 마스크를 생성한 후   
(파라미터 설명)  
src: 입력 그레이스케일 이미지

thresh: 임계값 (1보다 크면 255로 설정)

maxval: 임계값을 초과할 경우 적용할 값

type: 이진 임계값 처리 (cv.THRESH\_BINARY)  
  
np.where(condition, x, y) 함수로 두 이미지를 합성한다.

condition: 마스크 위치가 255인 곳 (정렬된 이미지가 존재하는 위치)

x: 그 위치에 대해 aligned\_img2 값 사용

y: 그렇지 않은 곳은 img1 값 사용  
그 이후 matplotlib를 사용하여 시각화 해주어 마친다.