Project 2: Local Feature Matching

Local Feature Detection + Description + Matching

요약 Summary

- 마감: 2022.04.18
- Part 1: Local feature matching 을 위한 함수 programing
  - Local feature 를 detect 하는 Harris corner detector 기능을 수행하도록 파일 `student.py` 내 함수 get_interest_points() 본문을 구현하세요.
  - Local feature descriptor 를 생성하도록 파일 `student.py` 내 함수 get_features(image, x, y, feature_width)을 구현하세요.
  - 생성된 2 개의 local feature descriptor 집합 im1_features, im2_features 가 입력될 경우 에 대응 관계를 도출하도록 파일 `student.py` 내 함수 match_features(im1_features, im2_features)을 구현하세요.
  - 필요에 따라 `Proj2.ipynb, student.py, helper.py` 등 파일들의 주석 설명문을 자세히 읽고 작업하세요.
  - 희망에 따라 아래 채점 기준 중 추가 점수 +α 획득을 위한 요소들을 구현해 보세요.
- Part 2: 보고서
  - 각 세부 과정에 대한 관련 이론을 요약하여 구현된 코드와 관련지어 설명하세요.
  - 기본 제공된 `Data` 폴더의 예시 영상들에 대해 도출된 결과를 분석하세요.
  - 제공된 보고서 양식 문서를 기반으로 작성하세요.
- 제출방법:
  - `student.py` 파일과 보고서를 하나의 파일로 압축하여 가상대학에 업로드 하세요.
  - 압축파일 파일 이름은 `hw2_본인학번.zip` 으로 하세요. (ex: `hw2_20181234.py`)
  - `student.py` 파일은 파일 이름을 `student_본인학번.py` 으로 변경하세요. (ex: `student_20181234.py`)
  - 보고서 파일는 report_본인학번.pdf 로 지정하세요. (ex: `report_20181234.pdf`)

개요 Overview

강의 6 및 강의 7 에서 설명된 지역적 특징점 local feature 정합 matching 알고리즘을 구현하여 실제 장면에 대한 다시점 영상 multiple views of real-world scenes 을 정합하고자 합니다. 해당 문제들은 지금까지 각 세부 과정 별로 각각 수백 편의 컴퓨터비전 computer vision 논문이 발표되었을 만큼 근본적이고 중요한 문제들입니다. 우리는 이번 과제에서 걸쳐 Harris corner detector 와 단순화된 버전의 SIFT simplified version of SIFT 알고리즘을 구현하고자 합니다. 희망하는 학생들은 기본적인 단순화된 알고리즘을 더 심화시키는 세부 사항들을 추가하여 알고리즘을 개선하는 다양한 노력을 해 볼 것을 권장합니다.

주어진 과제 Task: 지역적 특징점 정합 local feature matching 의 세 가지 주요 단계를 구현할 것:

- **검출 Detection** in the `get_interest_points` function in `student.py`. Please implement the Harris corner detector (Szeliski 7.1.1, Interest Point 강의자료 Slide 46).

- **기술자 생성 Description** in the `get_features function`, also in `student.py`. *유사-SIFT (SIFT-like)* 특징점 기술자 local feature descriptor (Szeliski 4.1.2)를 구현하시오. *완전-SIFT*까지 구현하지는 않아도 됨! 요구 기준을 충족시킬 때까지 디테일을 추가하여 보시오. 정합 파이프라인 matching pipeline 을 빠르게 테스트하기 위해서는 정규화된 패치 normalized patch 를 기술자로 우선 사용해 보세요.

- **정합 Matching** in the `match_features` function of `student.py`. 특징점 정합 matching local features 방법론 중 "비율 테스트 ratio test" 또는 "최소거리 비율테스트 nearest neighbor distance ratio test"을 구현해 보세요 (Szeliski 4.1.3; equation 4.18 in particular).

유용할 수 있는 함수들 Potentially useful functions: 기존의 필터링 함수들 및, `zip()`, `skimage.measure.regionprops()`, `skimage.feature.peak_local_max()`, `numpy.arctan2()` 등

유용할 수 있는 함수들 라이브러리들 Potentially useful libraries: 다양한 필터링 함수를 제공하는 `skimage.filters.x()` 또는 `scipy.ndimage.filters.x()`, 영상의 미분을 연산하는 `np.gradient()`, 원소별 element-wise binning 기능을 제공하는 `np.digitize()` 등. 그리고 이전 과제에서 각자 구현한 함수 등

사용 불가 함수들 Forbidden functions: `skimage.feature.daisy()`, `skimage.feature.ORB()` 등 특징점 기술자를 바로 도출해주는 함수들, `skimage.feature.corner_harris()` 등 특징점을 바로 검출해 주는 함수들, 그리고 히스토그램을 연산 *computes histograms* 해 주는 함수들, `sklearn.neighbors.NearestNeighbors()` 등 가장 가까운 이웃 비율 nearest neighbor ratios 을 계산해 주는 함수들, 그리고 `scipy.spatial.distance.cdist()` 등 벡터의 배열 간 거리를 계산해 주는 함수들 등 (벡터 간 거리를 계산하는 함수는 제공된 numpy tutorial 등 자료를 참조하여 직접 구현해 보세요). 특정 함수의 사용 가능성 여부가 불분명할 경우 반드시 질의응답 게시판에 질문을 해 주세요.

실험하기 Running the code

`Proj2.ipynb` 의 각 셀의 세부 내용을 자세히 읽어본 후, 순서대로 실행하면서 각자 구현한 함수의 결과를 도출하세요. 필요에 따라 `student.py` 에 직접 구현한 함수 내부 또는 notebook 파일의 셀 내부에 중간 결과를 확인할 수 있는 코드를 삽입하여 코드 각 줄마다 결과가 의도와 같게 도출되는지 확인하세요.
아울러 아래의 평가기준도 자세히 확인하고, 함수 내의 주석으로 표기된 설명문도 자세히 읽어서 확인하세요. 영문으로 된 설명은 직접 해석을 하기 어려우면 Papago 등 번역 사이트를 활용하세요. 아직 Python 프로그래밍이 익숙하지 않은 학생은 제공되는 Python 프로그래밍 tutorial 문서들을 학습하고, 특히 Numpy tutorial 을 확인하세요. 그리고도 모르는 부분은 가상대학 질의응답 게시판에 편하게 질문을 올려주세요.

요구사항 Requirements / 평가기준 Rubric

만점: 만약 여러분의 구현 결과에서 'matches' 내 most confident 50 개의 정합 쌍에 대해 60%의 정확도를 Notre Dame 데이터 및 Mt. Rushmore 데이터에 대해 모두 달성할 경우 85 점(코딩 점수 만점)을 부여합니다. 이때 정확도는 helpers.py 파일 내 evaluate_correspondence() 함수를 이용하여 측정합니다.

소요시간: 총 소요시간은 하나의 실험 데이터에 대해 20 분을 넘기지 않아야 하며 20 분을 넘기게 될 경우 코딩 점수는 최대 50 점 이내로만 부여합니다.
메모리제한: 가용 메모리가 부족할 경우 코드가 실행 도중 멈출 수 있으므로 사용하는 메모리를 주의하세요. 코드가 들어있는 폴더로 경로를 잡고 터미널에서 python memusecheck 를 실행하면 사용 메모리를 체크할 수 있습니다.

- 0 점: 구현한 함수가 제대로 동작하지 않거나, 결과를 아예 도출하지 못하거나 수행 중에 프로그램이 멈추는 경우
- +30 pts: `get_interest_points` 함수에 Harris corner detector 를 적절하게 구현함
- +40 pts: `get_features` 함수에 SIFT-like 특징점 기술자를 적절하게 구현함
- +15 pts: `student.py` 파일 내 `match_features` 함수에 matching local features 방법론 중 "최소거리 비율테스트 nearest neighbor distance ratio test"를 적절하게 구현함
- +15 : 구현한 프로그램에 대해 적절하게 보고서를 작성함
    - 최소 결과 영상을 3 개씩 제시할 것. 도출된 결과의 결과 영상 및 정량적인 평가 결과를 구체적으로 제시할 것. 또한 필요에 따라 관련되어 연관성이 있는 정보나 설명을 적절하게 제시할 것
    - 추가 점수 항목에 대한 구현 항목의 경우 관련 이론 및 코드 설명을 상세하게 제시할 것. 아울러 각 항목을 추가하지 않았을 때와 추가했을 때의 결과를 비교하여 구현한 추가 항목의 효과에 대한 검증 결과를 제시할 것
    - MS word, 아래아 한글 등의 문서 편집 프로그램을 사용하여 보고서를 작성하되, 제출 파일은 pdf 형식으로 제출할 것
- -5 점 : 위 설명 사항을 제대로 따르지 않은 건 별로 감점함

+α (추가 점수, 최대 15 점까지 부여):

- Detection:
    - 최대 +5 pts: Szeliski 교재에 제시된 adaptive non-maximum suppression 을 구현하여 적용함
    - 최대 +5 pts: feature point 를 다중 스케일에서 검출하거나 또는 스케일 선별적 기법을 활용하여 특징점의 스케일 정보를 도출함
- Description:
    - 최대 +3 pts: window size, number of local cells, orientation bins, normalization 방법 등 SIFT 기술자의 각종 인자 parameter 값을 다양하게 바꿔가면서 결과 확인함
    - 최대 +5 pts: 특징점의 지배적 방향성 orientation 을 추정하여 적용함
    - 최대 +5 pts: 특징점을 검출할 때 다중 스케일을 고려하여 검출하였으면, 기술자를 생성할 때 이 정보를 반영하여 해당 스케일에서 적절하게 기술자를 생성함
- Matching:
    - 최대 +10 pts: 연산 속도는 향상시키면서도 정합 정확도는 감소되지 않도록 하는 더 낮은 차원의 기술자를 생성함. 예를 들어 PCA 등의 방법을 활용하여 기술자의 차원을 축소하는 등의 방법을 적용함
    - 최대 +5 pts: k-d tree 등 벡터 기술자의 벡터 공간을 분할하여 연산을 가속화시키는 근사적 알고리즘을 적용하여 연산 속도를 개선함

추천 구현 전략 Implementation Strategy

1. `get_interest_points()`를 구현하기 전에 우선 `cheat_interest_points()` 함수를 활용하고 우선 Notre Dame 영상 쌍에 대해서만 실험하면서 구현함. 이 함수는 바로 참값

데이터의 특징점 좌표들을 로딩해와서 적용하는데, 이 데이터를 적용해도 초반에는 기술자가 제대로 구현되지 않았기 때문에 정합 정확도가 매우 낮을 것임.

2. `match_features()` 함수를 다음으로 구현함. 이때 for loop 보다는 numpy 의 vectorization 기능을 잘 활용하도록 하여 수행속도를 최대한 효율적으로 함. 특히 아래의 가이드 문서를 참조하여 구현하면 vectorization 효과를 극대화할 수 있음. https://browncsci1430.github.io/webpage/proj2_featurematching/matching/ 그리고 각 정합 대응쌍마다 기술자의 유사성 등을 반영하는 confidence 값도 적절하게 반환되도록 함.
3. `get_features()` 함수를 단순히 잘라낸 영상 패치를 활용하도록 수정해 봄. 픽셀 값들을 그대로 담고 있는 영상 패치 자체도 기술자라고 볼 수는 있지만, 조명 변화, 시점 변화 등에 민감하게 달라지기 때문에 최소한의 baseline 으로만 사용을 권장함.
4. `get_features()` 함수에 SIFT-like 특징 기술자 생성 기법을 구현함.
5. `get_interest_points()` 함수를 구현하고 이외의 Mt. Rushmore, Episcopal Gaudi 등 나머지 data 에 대해서도 실험 함.
6. 성능이 개선되는지 확인하면서 추가 점수 사항들을 구현 및 실험 함. 필요에 따라 아래의 추가 데이터도 활용함. https://drive.google.com/file/d/1x7qgcvDSmH57g0nuLcSg9wwnyRyTOCoI/view?usp=sharing

참고자료 References

1. James Tompkin, Project 2, CS1430 course description https://browncsci1430.github.io/webpage/hw2_featurematching/
2. Richard Szeliski, **Computer Vision: Algorithms and Applications, 2nd ed., Chapter 7. Feature** https://szeliski.org/Book/