CV_class_2025_2_Assignment_2 과제 풀이 보고서

Assignment-2-ccchoi7171

소프트웨어학부

20201887 최창수

TODO 7, 8을 작성하기 전 TODO 1~6간단 요약

TODO 1: Harris 코너 강도 및 방향 계산

설명: sobel 필터로 x, y 방향 그래디언트(Ix, Iy)를 구함. 이를 제곱하고 곱하여 Ixx, Iyy, Ixy를 만듭니다. 여기에 가우시안 필터를 적용해(윈도우 함수) M 행렬의 구성 요소를 구하고, det와 trace를 계산하여 최종 Harris 스코어(harrisImage)를 저장. arctan2를 사용해 각 픽셀의 그래디언트 방향(orientationImage)도 저장.

TODO 2: 로컬 최대값 계산

ndimage.maximum_filter를 사용해 각 픽셀을 7x7 이웃의 최대값으로 바꾼 localMax 이미지를 생성. 원본 harrisImage와 localMax를 비교하여, 값이 같은 픽셀(즉, 자신이 이웃 내 최대값인 픽셀)만 True로 표시하는 boolean 마스크를 반환.

TODO 3: KeyPoint 객체 생성

TODO 2에서 True로 판별된 픽셀의 (x, y) 좌표, TODO 1에서 계산한 orientationImage 값과 harrisImage 값을 cv2.KeyPoint 객체 f의 각 속성(pt, angle, response)에 할당.

TODO 4: Simple Feature Descriptor 구현

각 키포인트 (x, y)를 중심으로 5x5 윈도우를 순회. 이미지 경계를 벗어나는 픽셀은 0으로 두고, 경계 내의 픽셀은 grayImage에서 값을 가져와 d에 채움. 마지막으로 5x5 d 배열을 reshape하여 25차원 벡터로 만들어 desc의 i번째 행에 저장.

TODO 5: MOPS 변환 행렬 계산

MOPS는 (1) 특징점을 원점으로 이동시키고, (2) 특징점의 방향(f.angle)에 맞춰 회전시키고, (3) 40x40 패치를 8x8로 줄이는 스케일링(0.2배)을 적용한 뒤, (4) 8x8 이미지의 중심(4, 4)으로 다시 이동시킴. 이 모든 변환을 np.dot으로 결합하여 최종 2x3 어파인 변환 행렬 transMx를 만듦.

TODO 6: MOPS 디스크립터 정규화

cv2.warpAffine로 추출한 8x8 패치 destImage의 분산을 먼저 계산. 분산이 1e-10 미만이면 (즉, 패치가 거의 단색이면) 0으로 채워진 64차원 벡터를 desc[i]에 저장. 그렇지 않으면 (패치 - 평균) / 표준편차 공식을 사용해 정규화하고, 8x8 패치를 64차원 벡터로 reshape하여 desc[i]에 저장.

TODO 7과 TODO 8을 완성

TODO 7 (SSDFeatureMatcher):

- 핵심 로직: "단순히 가장 가까운 짝 찾기"
- 이미지 1의 한 특징점(desc1[i])에 대해, 이미지 2의 *모든* 특징점(desc2)과의 거리(SSD)를 전부 계산.
- 그중 거리가 가장 작은(minimum) 특징점 하나를 유일한 매칭 상대로 결정.
- DMatch 객체에 저장되는 distance는 이 최소 거리(SSD1) 값.
- 단점: 이미지 1의 특징점이 애매한 특징점일 경우 (예: 반복되는 패턴), 이미지 2의 여러 특징점과 거리가 비슷비슷하게 나올 수 있음. 이 방식은 그런 상황에서도 억지로 "가장 가까운" 하나를 선택하므로, 잘못된 매칭(False Positive)이 많이 발생함.

```
# ------------------------------
# TODO7 - SSD matching
# ------------------------------
# Step 1. SSD matcher를 이용해 두 이미지의 MOPS 디스크립터 매칭을 수행하시오.
matches_ssd = matcher_ssd.matchFeatures(desc_mops_1, desc_mops_2)

# Step 2. 거리(distance)를 기준으로 정렬하여 상위 150개의 매칭만 선택하시오.
matches_ssd = sorted(matches_ssd, key=lambda x: x.distance)[:150]

# Step 3. 매칭 결과를 시각화하여 PNG로 저장하시오.
ssd_vis = cv2.drawMatches(
    img1, d1, img2, d2, matches_ssd, None,
    matchColor=(0,255,0), singlePointColor=(255,0,0),
    flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS
)
cv2.imwrite("results/TODO7_SSD_matches.png", ssd_vis)
print("✅ TODO7 (SSD) match result saved → results/TODO7_SSD_matches.png")
```

TODO 8 (RatioFeatureMatcher):

- 핵심 로직: "독보적으로 가장 가까운 짝 찾기" (Lowe's Ratio Test)
- 이미지 1의 한 특징점에 대해, 이미지 2에서 거리가 가장 가까운 1순위(SSD1)와 두 번째로 가까운 2순위(SSD2) 특징점을 모두 찾음.
- DMatch 객체에 저장되는 distance는 실제 거리가 아닌, 두 거리의 비율 (SSD1 / SSD2) 값.
- 장점:
  - 만약 1순위 매칭이 2순위 매칭보다 월등히 가깝다면 (예: SSD1 = 10, SSD2 = 100), 이 비율(distance)은 0.1로 매우 낮아짐. 이는 "독보적인" 매칭이므로 신뢰할 수 있게 됨.
  - 만약 1순위와 2순위가 비슷하다면 (예: SSD1 = 10, SSD2 = 11), 이 비율(distance)은 0.9 정도로 높아짐. 이는 "애매한" 매칭이므로 신뢰할 수 없음.
- 결과적으로 distance (즉, 비율)를 기준으로 정렬하면, 애매한 매칭들보다 신뢰도 높은(독보적인) 매칭들이 먼저 선택 됨.

```
# ------------------------------
# TODO8 - Ratio matching
# ------------------------------
# Step 1. Ratio matcher를 이용해 두 이미지의 MOPS 디스크립터 매칭을 수행하시오.
matches_ratio = matcher_ratio.matchFeatures(desc_mops_1, desc_mops_2)

# Step 2. distance를 기준으로 정렬하여 상위 150개의 매칭만 선택하시오.
matches_ratio = sorted(matches_ratio, key=lambda x: x.distance)[:150]

# Step 3. 매칭 결과를 시각화하여 PNG로 저장하시오.
ratio_vis = cv2.drawMatches(
    img1, d1, img2, d2, matches_ratio, None,
    matchColor=(0,255,0), singlePointColor=(255,0,0),
    flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS
)
cv2.imwrite("results/TODO8_Ratio_matches.png", ratio_vis)
print("✅ TODO8 (Ratio) match result saved → results/TODO8_Ratio_matches.png")
```

<u>해당 TODO 8의 매칭이 더 잘된 이유는 tests.py 코드의 맨 마지막에 각주로 작성했음.</u>

(features.py 기반)