تمرین سری ۱ - سوال ۳

محمدجواد شریعتی 96100414

به منظور خوانایی بیشتر و تمیزی کد، گام های اصلی برنامه (تابع main) در فایل Q*.py قرار دارد و تابع هایی که پیاده سازی کردم در فایل utils.py قرار دارد. ابتدا تابع main و گام های اصلی برنامه را توضیح میدهم و درانتها در مورد هریک از تابع های پیاده سازی شده در فایل utils.py توضیح خواهم داد.

هدف از این تمرین بدست آوردن هموگرافی بین دو تصویر با کمک تابع های آماده SIFT و RANSAC است. من در این تمرین از داکیومنت‌های OpenCv زیر کمک گرفتم:

[Introduction to SIFT]
[Feature Matching]
[Feature Matching + Homography to find Objects]

در ابتدا پس از لود کردن عکس‌ها، با کمک SIFT detector لایبرری opencv ابتدا interest point های هرکدام از تصاویر را به صورت جداگانه بدست آوردم و سپس descriptor های آنها را هم بدست آوردم:

```
# Create SIFT detector
sift = cv2.SIFT_create()

# Find interest points
interest_points_1 = sift.detect(im1_gray, None)
interest_points_2 = sift.detect(im2_gray, None)

# Show points on image
im1_with_interest_points = cv2.drawKeypoints(im1, interest_points_1, copyof(im1), color=(0, 255, 0))
im2_with_interest_points = cv2.drawKeypoints(im2, interest_points_2, copyof(im2), color=(0, 255, 0))

corners = show_together(im1_with_interest_points, im2_with_interest_points)
cv2.imwrite('out/res13_corners.jpg', corners)

# Compute Descriptors
_, descriptor1 = sift.compute(im1_gray, interest_points_1)
_, descriptor2 = sift.compute(im2_gray, interest_points_2)
```

پس از آن با کمک تابع آماده BFMatcher نقاط متناظر هر دو تصویر را بدست آوردم. همچون تمرین ۱، دوتا از مشابه ترین ها را بدست آوردم و سپس یک ratio test با نسبت 0.9 روی آن انجام دادم. سپس عکس‌های لازم را خروجی گرفتم.

```
# Use BFMatcher to match points between two images
bf_matcher = cv2.BFMatcher()
match_points = bf_matcher.knnMatch(descriptor1, descriptor2, k=2)

# Ratio between most similar and second most similar should less than 0.9
final_match_points = []
for m, n in match_points:
    if m.distance / n.distance < 0.90:
        final_match_points.append(m)
```

حال که نقاط متناظر دو تصویر را بدست آوردیم، نوبت به پیدا کردن هموگرافی بین آنهاست. بدین منظور هم از تابع آماده findHomography با متد RANSAC و ترشولد 150 استفاده کردم. خروجی این تابع ماتریس H (که هموگرافی ماست) و یک mask است که درواقع مشخص می‌کند کدام یک از نقاط متناظر inlier هستند و کدام ها outlier.

```
# Find Homography
# Get Src(im2) and Dst(im1) match points for cv2.findHomography function
src_points = []
dst_points = []
for match in final_match_points:
    src_points.append(interest_points_2[match.trainIdx].pt)
    dst_points.append(interest_points_1[match.queryIdx].pt)
src_points = np.float32(src_points).reshape(-1, 1, 2)
dst_points = np.float32(dst_points).reshape(-1, 1, 2)

# Run RANSAC algorithm with threshold = 5.0
# mask specifies that which points are inlier and which are outlier
H, mask = cv2.findHomography(src_points, dst_points, cv2.RANSAC, 150.0)
print("Homography Matrix: \n", H)
```

ماتریس هموگرافی بدست آمده چنین است:

```
Homography Matrix:
 [[ 5.75592530e+00  6.47390761e-01 -3.88827442e+03]
 [ 3.78833226e-01  3.81838338e+00 -2.15784451e+03]
 [ 5.38800290e-04  8.89290992e-05  1.00000000e+00]]
```

الگوریتم حدود ۱۵۰۰ اینتریشن لازم دارد تا به جواب درست برسد. (با ست کردن max_iters و سرچ باینری!)
سپس عکس های خواسته شده را خروجی دادم و پس از آن برای وارپ کردن، ابتدا نقاط دور آن را بدست آوردم (min_x و min_y و max_x و max_y) (کدهای کامنت شده) و درنهایت ماتریس هموگرافی بدست آمده را در تصویر دوم وارپ کردم:

```
im2_warp = cv2.warpPerspective(im2, H, (6000, 4500))
cv2.imwrite('out/res19.jpg', im2_warp)
```

تابع های پیاده سازی شده در utils.py هم تابع های ساده ای برای نشان دادن عکس ها کنار هم (تابع show_together) ، گرفتن یک کپی از یک عکس (copyof) مشابه کاری که deepcopy انجام می‌دهد و کشیدن خط روی تصویر به هم‌چسبیده تصویر اول و دوم است(برای نشان دادن mismatch) (تابع draw_lines_on_together_image).