تمرین سری ۱ - سوال ۴

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.

این تمرین کاملا مشابه تمرین سوم است، با این تفاوت که به جای استفاده از تابع های آماده برای بدست آوردن هموگرافی، این پروسه را خودم پیاده سازی کرده ام.
در ابتدا و تا بدست آوردن نقاط متناظر همه چیز مشابه تمرین سوم است و کدها عینا از تمرین قبلی کپی شده است(مراحل بدست آوردن interest point ها، بدست آوردن descriptor برای آنها و سپس پیداکردن نقاط متناظر).
سپس با کمک نقاط بدست آمده، تابع RANSAC را که خودم پیاده سازی کردم روی آنها با پارامترهای زیر اجرا می‌کنم. خروجی آن ماتریس H است که هموگرافی ما است.

```
# Get Homography Matrix form Implemented RANSAC function
H = RANSAC(src_points, dst_points, threshold=50.0, maxIters=10000)
print("Homography Matrix: \n", H)
```

ماتریس هموگرافی بدست آمده در یکی از اجراها چنین است:

```
Homography Matrix:
 [[ 1.38592318e-03  1.01934155e-04 -8.98954491e-01]
 [ 3.47163843e-05  8.58798448e-04 -4.38038857e-01]
 [ 6.80435050e-08 -4.41724887e-08  3.36028827e-04]]
```

ماتریس هموگرافی بدست آمده در تمرین قبل توسط تابع آماده openCV چنین بود:

```
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]]
```

طی اجراهای مختلفی که من داشتم، هربار به طور متوسط حدود 6000 ایتریشن انجام می‌شد.

سپس مشابه تمرین قبل و بعد از بدست آوردن گوشه های تصویر حاصله (min_x و min_y و ...)، تابع هموگرافی را در آن وارپ می‌کنم.

تابع RANSAC پیاده سازی شده در utils.py:

روند این تابع دقیقا مشابه اسلایید زیر است:

Algorithm:

1. $counter = 0, \quad w = 0 \implies N = \infty, \quad w_{min} = 0$
2. while $N > counter$:
   choose a sample of s points
   fit the model to this sample
   count the number of inliers
   set $w = \dfrac{\#\text{inliers}}{\#\text{all points}}$
   if $w > w_{min}$: update w_{min} *and* N with a specific p
   $counter \mathrel{+}= 1$
3. Take the inliers of the model with the largest number of inliers, and fit the model to them by least squares

$$N \geq \frac{\log(1-p)}{\log(1-w^s)}$$

```python
def RANSAC(src_points, dst_points, threshold, p=0.99, maxIters=10000):
    s = 4  # Minimum points
    number_of_points = len(src_points)

    counter = 0
    w_min = 0
    N = sys.maxsize
    best_H = None

    while N > counter and counter < maxIters:
        random_indices = random.sample(range(0, number_of_points), s)  # select s random index

        random_src_points = [src_points[i] for i in random_indices]
        random_dst_points = [dst_points[i] for i in random_indices]

        A = getA(random_src_points, random_dst_points)
        U, S, V_T = svd(A)
        H = V_T[-1].reshape(3, 3)

        supports = 0  # number of inliers by this H
        for i in range(number_of_points):
            src = np.array([[src_points[i][0]],
                            [src_points[i][1]],
                            [1]])
            dst_point_by_H = np.matmul(H, src)

            dst_point_by_H[0] /= dst_point_by_H[2]
            dst_point_by_H[1] /= dst_point_by_H[2]

            diff = math.sqrt(
                ((dst_points[i][0] - dst_point_by_H[0]) ** 2) + ((dst_points[i][1] - dst_point_by_H[1]) ** 2))
            if diff < threshold:
                supports += 1

            w = supports / number_of_points
            if w > w_min:
                w_min = w
                N = math.log(1 - p) / math.log(1 - (w ** s))
                best_H = H

        counter += 1
```

مقدار s یعنی تعداد نقاطی که هربار به صورت رندوم انتخاب می‌کنم و مدل میکنم را برابر ۴ (حداقل تعداد نقاط لازم برای هموگرافی) قرار دادم. در هر ایتریشن، ۴ نقطه به صورت رندوم انتخاب می‌شود. سپس این نقاط به تابع getA پاس داده می‌شوند که این تابع با دریافت نقاط، چنین ماتریسی می‌سازد:

$$\begin{bmatrix} -\mathbf{x}_1^t & \mathbf{0} & y'_1\mathbf{x}_1^t \\ \mathbf{0} & -\mathbf{x}_1^t & x'_1\mathbf{x}_1^t \\ \vdots & \vdots & \vdots \\ -\mathbf{x}_n^t & \mathbf{0} & y'_n\mathbf{x}_n^t \\ \mathbf{0} & -\mathbf{x}_n^t & x'_n\mathbf{x}_n^t \end{bmatrix} \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \mathbf{h}_3 \end{bmatrix} = \mathbf{0}$$

```python
def getA(src_points, dst_points):
    A = np.zeros((2 * len(src_points), 9))
    for i in range(len(src_points)):
        A[2 * i:2 * i + 2, :] = np.array([[-src_points[i][0], -src_points[i][1], -1, 0, 0, 0,
                                           src_points[i][0] * dst_points[i][0], src_points[i][1] * dst_points[i][0],
                                           dst_points[i][0]],
                                          [0, 0, 0, -src_points[i][0], -src_points[i][1], -1,
                                           src_points[i][0] * dst_points[i][1], src_points[i][1] * dst_points[i][1],
                                           dst_points[i][1]]])
    return A
```

پس از بدست آوردن A، حال به دنبال جواب AH=0 هستیم. بدین منظور تجزیه svd ماتریس A را بدست میاوریم و آخرین ستون V (آخرین سطر V^t) را به عنوان H برمیگردانیم. حال خوب بودن این H را باید بسنجیم. بدین منظور روی تمام نقاط for میزنم و حاصل ضرب Hx را بدست میاورم و آن را با مقدار متناظر x که از قبل دارم مقایسه می‌کنم. بدین منظور از فاصله اقلیدسی استفاده کرده ام و اگر این فاصله کمتر از مقدار threshold باشد آن را یک inlier محسوب می‌کنم. پس از بررسی تمام نقاط، مقدار w جدید را محاسبه و سپس با w_min مقایسه می‌کنم و درنهایت اگر شرط برقرار باشد N را آپدیت می‌کنم.

پس از اتمام ایتریشن، حال با best_H بدست آمده، میتوانم هوموگرافی نهایی را بیابم:

```
inliers_idx = []
for i in range(number_of_points):
    src = np.array([[src_points[i][0]],
                    [src_points[i][1]],
                    [1]])
    dst_point_by_H = np.matmul(best_H, src)

    dst_point_by_H[0] /= dst_point_by_H[2]
    dst_point_by_H[1] /= dst_point_by_H[2]

    diff = math.sqrt(
        ((dst_points[i][0] - dst_point_by_H[0]) ** 2) + ((dst_points[i][1] - dst_point_by_H[1]) ** 2))
    if diff < threshold:
        inliers_idx.append(i)

src_inlier = [src_points[i] for i in inliers_idx]
dst_inlier = [dst_points[i] for i in inliers_idx]

A = getA(src_inlier, dst_inlier)
U, S, V_T = svd(A)
H = V_T[-1].reshape(3, 3)

return H
```

بدین منظور ابتدا inlier ها آن را بدست می‌اورم (مشابه بالا و با همان ترشولد) و پس از آن باکمک تمام آن نقاط تابع هموگرافی را بدست می‌اورم.