تمرین سری ۱ - سوال ۱

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.

توضیح روند اصلی کد (فایل Q1.py)

پس از لود کردن عکس ها، آن‌ها را سیاه و سفید میکنم تا در ادامه به جای کار کردن روی ۳ لایه، تنها با یک لایه کار کنم. بعلاوه درمحاسبات اولیه لزومی بر رنگی بودن عکس نیست و فایده قابل توجهی نخواهد داشت. اما زمانی که می‌خواهم بردار ویژگی بدست آورم، از ۳عکس رنگی لایه استفاده می‌کنم تا بردار ویژگی های بهتری بدست آورم.

ابتدا مشتق هر دو عکس نسبت به x و y را بدست می‌آورم. برای اینکار از یک فیلتر مشتق تابع گاوس 7*7 با سیگما 1.3 استفاده کردم. سپس با استفاده از این مشتق ها I_x^2, I_y^2, I_{xy} را برای هردو عکس بدست می‌آورم.

بعد از آن گرادیان عکس ها را با کمک تابع np.hypot و مشتق های عکس‌ها نسبت به x و y بدست می‌آورم.

سپس با استفاده از یک فیلتر گوس 13*13 با سیگما 10 تابع های S_x^2, S_y^2, S_{xy} را بدست آوردم.

```
# Calculate I_x and I_y: convolve images with gaussian derivative filter in x and y axises
filter_size = 7
sigma = 1.3
im1_x = get_x_derivative(im1_gray, filter_size=filter_size, sigma=sigma)
im1_y = get_y_derivative(im1_gray, filter_size=filter_size, sigma=sigma)

im2_x = get_x_derivative(im2_gray, filter_size=filter_size, sigma=sigma)
im2_y = get_y_derivative(im2_gray, filter_size=filter_size, sigma=sigma)

# Calculate I_x2, I_y2 and I_xy
im1_x2, im1_y2, im1_xy = calculate_Ix2_Iy2_Ixy(im1_x, im1_y)
im2_x2, im2_y2, im2_xy = calculate_Ix2_Iy2_Ixy(im2_x, im2_y)

# Calculate gradient
im1_gradient = np.hypot(im1_x, im1_y)
cv2.imwrite("out/res01_grad.jpg", get_distributed_image(im1_gradient, data_type='uint8'))
im2_gradient = np.hypot(im2_x, im2_y)
cv2.imwrite("out/res02_grad.jpg", get_distributed_image(im2_gradient, data_type='uint8'))

# Calculate S_x2, S_y2 and S_xy
gaussian_filter_size = 13
gaussian_filter_sigma = 10
im1_S_x2, im1_S_y2, im1_S_xy = calculate_Sx2_Sy2_Sxy(im1_x2, im1_y2, im1_xy,
                                                     gaussian_filter_size=gaussian_filter_size,
                                                     gaussian_filter_sigma=gaussian_filter_sigma)
im2_S_x2, im2_S_y2, im2_S_xy = calculate_Sx2_Sy2_Sxy(im2_x2, im2_y2, im2_xy,
                                                     gaussian_filter_size=gaussian_filter_size,
                                                     gaussian_filter_sigma=gaussian_filter_sigma)
```

سپس مقدار R را برای هرپیکسل با کمک تابع calculate_R بدست می‌آورم. مقدار k را به صورت تجربی برای هردو تصویر مقدار 0.1 بدست آوردم. بعد از آن یک threshold با مقدار 0.5 روی آن اعمال می‌کنم.

بعد از آن باکمک تابع nms در هر component، پیکسلی که بیشترین مقدار را دارد نگه می‌دارم و بقیه را حذف می‌کنم.

```
# Build R
R1 = calculate_R(height, width, im1_S_x2, im1_S_y2, im1_S_xy, k=0.6)
cv2.imwrite("out/res03_score.jpg", get_distributed_image(R1, data_type='uint8'))
R2 = calculate_R(height, width, im2_S_x2, im2_S_y2, im2_S_xy, k=0.6)
cv2.imwrite("out/res04_score.jpg", get_distributed_image(R2, data_type='uint8'))

# Apply threshold
R1_threshold = apply_threshold(R1, threshold=0.5)
cv2.imwrite("out/res05_thresh.jpg", np.vectorize(lambda x: 255 if x > 0 else 0)(R1_threshold))

R2_threshold = apply_threshold(R2, threshold=0.5)
cv2.imwrite("out/res06_thresh.jpg", np.vectorize(lambda x: 255 if x > 0 else 0)(R2_threshold))

# Non-Maximum Suppression
im1_points = nms(R1_threshold)
im1_harris = show_points_on_image(im1, im1_points)
cv2.imwrite("out/res07_harris.jpg", im1_harris)

im2_points = nms(R2_threshold)
im2_harris = show_points_on_image(im2, im2_points)
cv2.imwrite("out/res08_harris.jpg", im2_harris)
```

حال نوبت به بدست آوردن بردارهای ویژه است. تابعی به نام get_match_points در utils.py پیاده سازی کرده ام که هردوعکس و interest_point های بدست آمده در مرحله قبل را بعلاوه چند پارامتر می‌گیرد و بردار ویژگی مربوطه برای sub window های عکس‌ها را میسازد و با مقایسه کردن آن‌ها و برای هرنقطه تصویر اول، بهترین نقطه موجود از تصویر دوم را پیشنهاد می‌دهد. من به صورت تجربی مقدار n را ۶۰ و ratio (آستانه نسبت d1/d2) را برای هردو تصویر 0.97 بدست آوردم. (باقی پارامترها را در بخش utils.py توضیح می‌دهم)

```
n = 70
im1_match_points = get_match_points(im1, im2, im1_points, im2_points, n, ratio=0.97)
im2_match_points = get_match_points(im2, im1, im2_points, im1_points, n, ratio=0.97)
```

سپس نقاطی را که هم در تصویر اول و هم درتصویر دوم متناظر یکدیگر هستند را جدا می‌کنم وبقیه را حذف می‌کنم.

```
# Check to the corresponding points be in both matches!
for point1 in list(im1_match_points):
    if point1 not in im2_match_points.values():
        im1_match_points.pop(point1)

for point2 in list(im2_match_points):
    if point2 not in im1_match_points.values():
        im2_match_points.pop(point2)
```

درنهایت هم نقاط را ابتدای به صورت جدا روی هریک از تصاویر نشان می‌دهم و سپس هردوعکس را کنار هم قرار می‌دهم و نقاط متناظرشان را با خط بهم متصل می‌کنم.

تابع‌های پیاده سازی شده در utils.py :

- **تابع gaussian:** مقدار تابع گاوس در یک نقطه مشخص با μو σمشخص را برمی‌گرداند.
- **تابع gaussian_derivative:** مقدار مشتق تابع گاوس را در یک نقطه با μو σمشخص برمی‌گرداند.
- **تابع gaussian_filter:** یک فیلتر دوبعدی گاوس با اندازه و سیگما و جهت داده شده برمی‌گرداند.
- **تابع gaussian_derivative_filter:** یک فیلتر دوبعدی مشتق گاوس با اندازه و سیگما و جهت داده شده برمی‌گرداند.
- **تابع get_x_derivative:** مشتق عکس داده شده را در جهت x به ما می‌دهد. برای این کار فیلتر بدست آمده از تابع gaussian_derivative_filter را در عکس کانوالو می‌کند.
- **تابع get_y_derivative:** مشتق عکس داده شده را در جهت y به ما می‌دهد. برای این کار فیلتر بدست آمده از تابع gaussian_derivative_filter را در عکس کانوالو می‌کند.
- **تابع get_distributed_image:** پیکسل‌های عکس داده شده را در بازه ۰تا۲۵۵ پخش می‌کند تا برای نمایش مناسب باشد.
- **تابع calculate_Ix2_Iy2_Ixy:** با کمک ضرب درایه به درایه، ماتریس‌های I_x^2, I_y^2, I_{xy}را محاسبه می‌کند.
- **تابع calculate_Sx2_Sy2_Sxy:** با کانولو کردن فیلتر گاوس با مشخصات داده شده در عکس داده شده، ماتریس‌های S_x^2, S_y^2, S_{xy} را خروجی می‌دهد.
- **تابع calculate_R:** دترمینان و تریس ماتریس M را بدست می‌آورد و باکمک آن‌ها و مقدار داده شده k،مقدار R را برای هرپیکسل بدست می‌آورد و ماتریسی را خروجی می‌دهد که هر درایه آن مقدار R برای آن نقطه است.
- **تابع apply_threshold:** یک عکس و یک مقدار threshold میگیرد و پیکسل‌هایی را که کمتر از آن مقدار هستند را حذف می‌کند.
- **تابع nms:** برای محاسبه non-maximum supression ، ابتدا مولفه های همبندی عکس داده شده را به وسیله تابع cv2.connectedComponentsWithStats بدست می‌آورد. سپس در هر مولفه همبندی، پیکسل با بیشترین مقدار را نگه می‌دارد و باقی را حذف می‌کند.
- **تابع show_points_on_image:** برای نمایش نقاط روی تصویر استفاده می‌شود. بعلاوه خود پیکسل، پیکسل های دور آن را هم قرمز می‌کند تا بهتر مشخص باشد.
- **تابع get_match_points :** ورودی های تابع عبارتند از: دو تصویر رنگی، interest point های بدست آمده از روش هریس، n (سایز پنجره) و ratio (آستانه نسبت d1/d2). برای هر نقطه از تصویر اول تک تک نقاط تصویر دوم را امتحان می‌کنیم تا مشابه ترین نقطه را بدست آوریم(دقت کنیم که یکبار تصویر im01 را به عنوان تصویر اول و یکبار تصویر im02 را به عنوان تصویر اول به این تابع می‌دهیم و نقاط مشترک این دو خروجی نقاط نهایی ما خواهند شد).
بدین صورت که یک پنجره n*n حول دو نقطه در نظر میگیرم و این دو پنجره را لایه به لایه از هم کم می‌کنیم و به توان ۲ می‌رسانیم (روش SSD). با نگه داشتن مشابه ترین و دومین مشابه ترین در نهایت نسبت d1/d1 را محاسبه می‌کنم و اگر از مقدار آستانه ratio کوچک تر بود این نقطه، یک نقطه مناسب است. همچنین اگر یک نقطه بیش از یک تناظر داشت همه را حذف می‌کنم. مراحل دوم و سوم (یعنی بدست آوردن بردار ویژگی ها و پیداکردن نقاط متناظر باهم در این تابع انجام می‌شوند. بردار ویژگی ای که من استفاده کردم ترکیبی از رنگ پنجره هاست درواقع.)
- **تابع get_sub_window :** یک عکس، یک نقطه و یک n میگیرد و یک مربع n*n حول نقطه داده شده از عکس را برمیگرداند.
- **تابع get_random_color:** یک رنگ رندوم برمی‌گرداند.