تمرین سری ۲ تمرین شماره ۲

محمد جواد شریعتی ۹۶۱۰۰۴۱۴

توضیح فایل helper.py

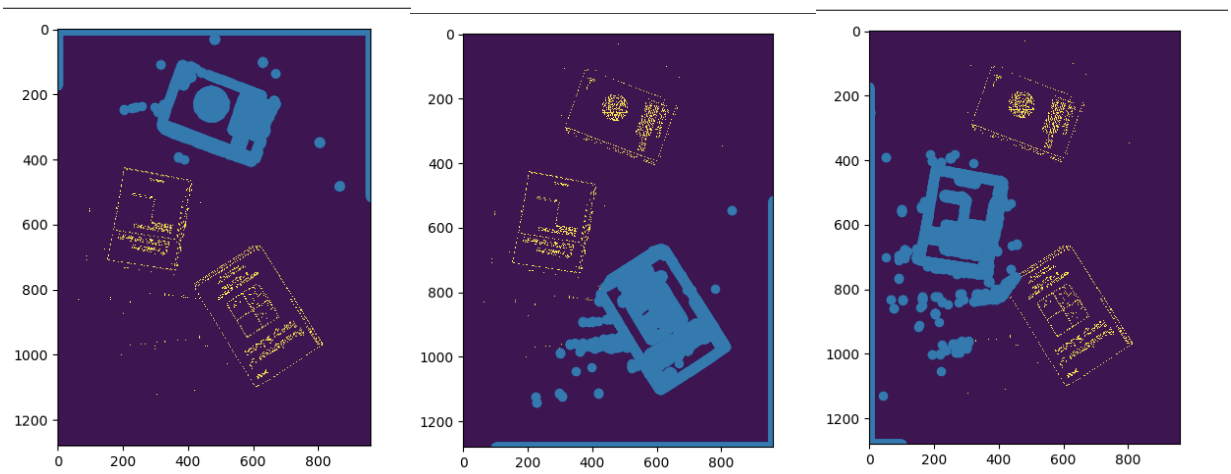
در تمامی تمرین ها،کنار فایل اصلی کد، یک فایل helper.py هم قرار داده شده است. این فایل به منظور تمیزتر شدن کد نوشته شده است. تابع هایی که استفاده میکنم را در این فایل پیاده سازی می کنم تا کد اصلی تمیز تر بشود.

ابتدا عکس اصلی و همین طور edge_detection که در سوال اول این تمرین انجام دادم را load میکنم. عکس edge رو به صورت سیاه و سفید میخونم و سپس تبدیل به عکس باینری میکنم:

```
edge_image = cv2.imread("Q1_09_edge.jpg", cv2.IMREAD_GRAYSCALE)
edge_image = np.vectorize(lambda x: 255 if x > 20 else 0)(edge_image)
```

بعد از اون با استفاده از تابع get_edge_points که در فایل helper پیاده سازی شده است نقاطی که در عکس edge ناصفر هستند را در یک لیست ذخیره میکنم. (x و y) این نقاط رو در این لیست میریزم. هدف از این کار کلاسترینگ به وسیله روش k-means است. برای k-means لازم هست که تعداد کلاستر ها را داشته باشیم. اگر تعداد کتاب ها متغیر باشد می توان از mean-shift کمک گرفت. چون اینجا تعداد کتاب ها را داریم از k-means استفاده کردم و متغیر NUMBER_OF_BOOKS که در ابتدای کد هست به همین منظور تعریف شده است.

با استفاده از k-means نقاط را به ۳ کلاستر تقسیم میکنم که ریزالت به صورت زیر است:



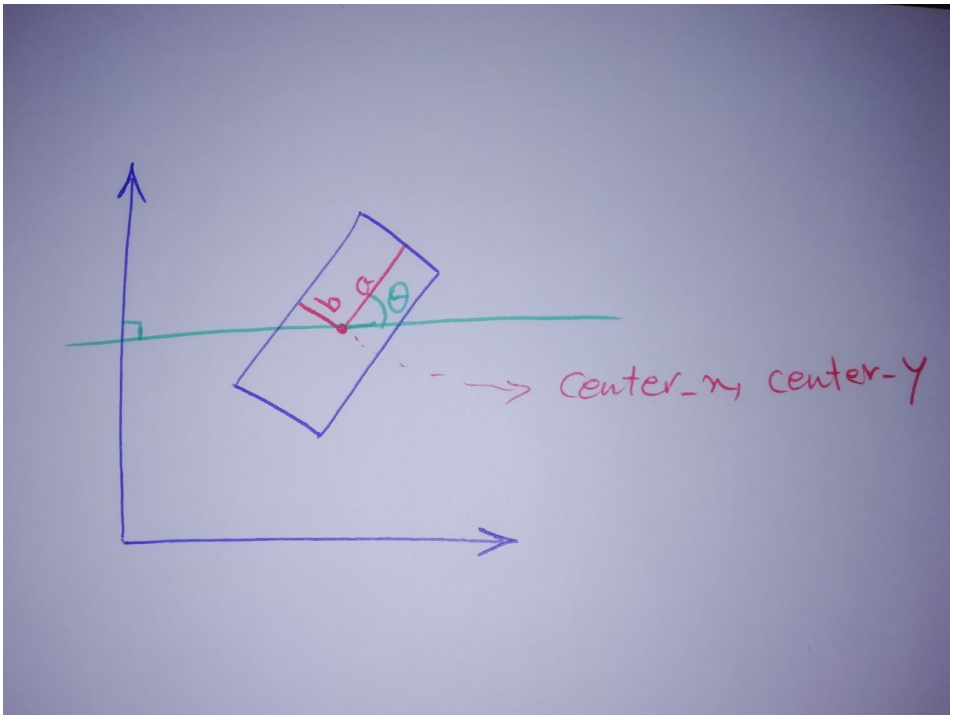
بدین ترتیب کار برای اجرای الگوریتم هاف و پیدا کردن مستطیل ها خیلی راحت تر خواهد بود. زیرا این کلاسترینگ دو مزیت مهم برای ما دارد: یکی اینکه میتوان از مرکز کلاسترها استفاده کرد و مرکز مستطیل های ما در شعاع مثلا ۱۰۰ پیکسل این مراکز کلاستر قرار دارد. بدین ترتیب دیگر لازم نیست به ازای تمام نقاط صفحه مستطیل های ممکن را بررسی کنیم و کافی است در یک شعاع معقول از مراکز کلاستری که k-means به ما می دهد تنها این کار را انجام دهیم.
مزیت دومی که به ما می دهد این است که برای بررسی مستطیل های یک نقطه، لازم نیست تمام نقاط edge را بررسی کنیم. فقط کافی است نقاطی که در یک کلاستر قرار گرفته اند را بررسی کنیم:

```
Cluster Centers:
[[532.03889304 230.10072301]
 [634.25531698 912.05138903]
 [246.85284281 647.27991857]]
Total Points:  24726
number of points in each cluster:
cluster1 points:  8021
cluster2 points:  9829
cluster3 points:  6876
```

یعنی به جای 24000 نقطه کافی است مثلا 8000 نقطه بررسی شوند.
برای سریع اجرا شدن الگورتم، من اینکه در یک دایره به شعاع مثلا ۱۰۰ حول cluster center ها دنبال مستطیل بگردم، به صورت دستی x,y مراکز مستطیل ها را بدست آوردم و فقط الگوریتم را روی آن اجرا کردم. به وضوح اگر یک for پیاده سازی شود که روی یک شعاع مشخص کارهایی که گفته می شود را انجام دهد و در نهایت بین ریزالت هایی که بدست می آید max بگیرد، به همین ریزالت خواهیم رسید. دلیل اینکه این قسمت را پیاده سازی نکردم مشکل مموری و زمان بود. زیرا در این صورت به جای یک table سه بعدی، باید یک table ۵ بعدی نگه داریم که امکان کمبود مموری وجود دارد.
در نتیجه center_x و center_y ها را دستی بدست آوردم و set کردم.

```
# Book 1
center_x = 490
center_y = 250
```

حال به ازای این مرکز مستطیل(در حالت کلی به ازای تک تک نقاط حول کلاستر سنتر) میخواهیم تمام مستطیل های ممکن را بررسی کنیم و ببینیم که به ازای یک a و b و theta چند نقطه از نقاط کلاستر روی این مستطیل می افتد. بدین ترتیب با voting مستطیلی که بیشترین نقطه را دارا باشد، کتاب ما خواهد بود. این مفهوم اصلی الگوریتم هاف است که پیاده سازی شده است.



یعنی به ازای هر نقطه حول کلاستر(در اینجا center_x و center_y) تمام مستطیل های ممکن را بررسی میکنیم. این کار را با تغییر دادن a و b و theta انجام میدهیم. یعنی با ۳ for تودرتو تمام حالت های ممکن را بررسی میکنم و به ازای هر مستطیل (با یک a و b و theta مشخص فقط یک مستطیل مشخص میشود) محاسبه میکنم که چندین نقطه روی این مستطیل قرار میگیرند. این مقدار را در table ذخیره میکنم. در نهایت بزرگترین مقدار table یعنی a و b و theta ای که بیشترین نقطه رو داشته اند را بدست میاوریم و با استفاده از اون مستطیل رو می کشم.

پر کردن table با کمک تابع زیر که در helper.py پیاده سازی شده است انجام میشود:

```
def fill_a_b_theta_table(cluster_points, table, center_x, center_y, A, AA, B, BB, ):
    for a in range(A, AA):
        for b in range(B, BB):
            for theta in range(0, 180, 10):
                for point in cluster_points:
                    point = point - np.array([center_x, center_y])
                    rotation_point = np.dot(get_rotation_matrix(-theta), point)
                    if int(np.abs(rotation_point[0])) == a or int(np.abs(rotation_point[1])) == b:
                        table[a - A, b - B, theta // 10] = table[a - A, b - B, theta // 10] + 1
    return table
```

مقدار A مشخص میکند که از چه مقداری برای a شروع کنیم و تا چه مقداری (AA) پیش برویم. من این مقادیر را نزدیک هم قرار دادم تا برنامه سریع اجرا شود. به وضوح اگر بازه را بزرگ تر کنیم باز هم همان جواب قبلی را خواهیم گرفت. برای چک کردن اینکه یک نقطه روی مستطیل می افتد یا خیر بدین صورت عمل کردم که مختصات نقطه را منهای مختصات مرکز مستطیلم کردم(به نوعی نقطه را به دستگاه مختصات مستطیل بردم یعنی مرکز مستطیل را (۰,۰) کردم به گونه ای) سپس برای اینکه اثر theta هم خنثی شود(انگار مستطیل نچرخیده است) آن را در ماتریس وارون دوران theta که همان ماتریس دوران منفی theta است ضرب کردم. در نتیجه حالا مسئله ساده سازی شده است. انگار مرکز مستطیل روی (۰,۰) قرار دارد و مستطیل نچرخیده است. کافی است چک کنیم نقطه روی اضلاع میفتد یا نه که مثل این است که قدرمطلق x یا y آن برابر a یا b شود. بدین ترتیب اگر نقطه در روی مستطیل افتاده بود، یکی به خانه مربوطه آن در table اضافه میکنیم.

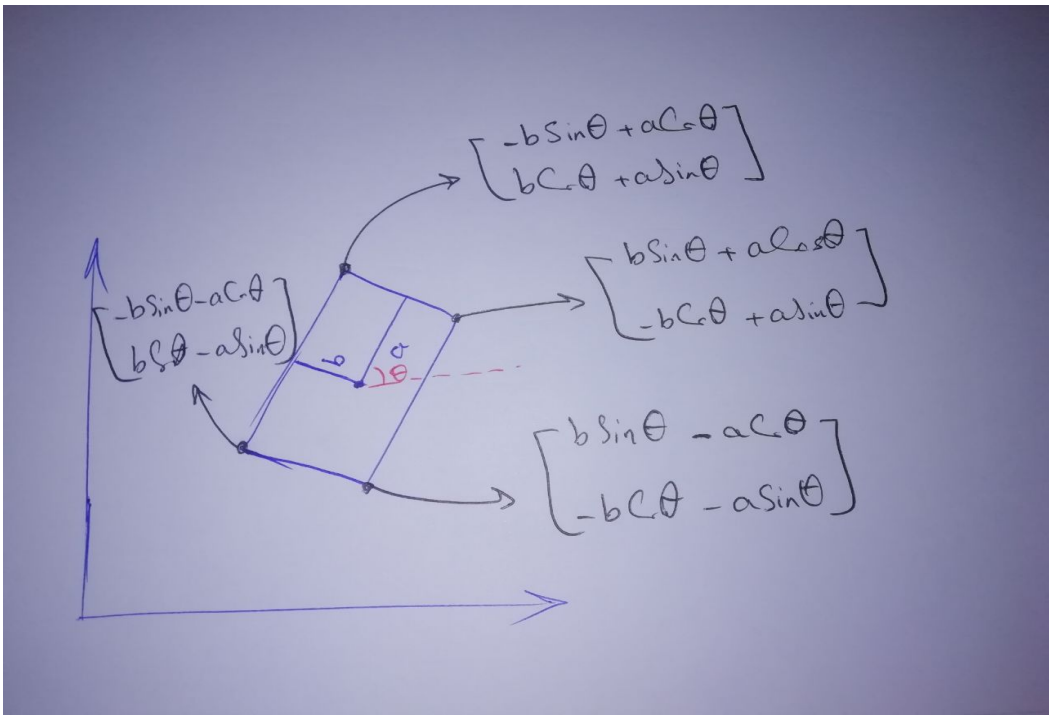
تابع get_rectangle_width_height_theta براساس max خانه های table بهترین a وb و thetaرا پیدا میکند و طول و عرض و زاویه مستطیل را پیدا میکند:

```
def get_rectangle_width_height_theta(table, A, AA, B, BB, max):
    best_a, best_b, best_theta = 0, 0, 0
    for a in range(AA - A):
        for b in range(BB - B):
            for theta in range(18):
                if table[a, b, theta] == max:
                    best_a, best_b, best_theta = a, b, theta

    rectangle_width = best_a + A
    rectangle_height = best_b + B
    rectangle_theta = best_theta * 10

    return rectangle_width, rectangle_height, rectangle_theta
```

تابع draw_rectangle با کمک نکته زیر چهارگوشه مستطیل را بدست می آورد:



بدین ترتیب با داشتن تمام مجهولات، ۴ گوشه مستطیل بدست می آید و با کمک تابع cv2.line خط ها را میکشم تا مستطیل روی شکل کشیده شود. همچنین مختصات پرینت میشود تا برای سوال بعدی استفاده شود(۲ نمره ای که در سوال ۳ برای این مورد در نظر گرفته شده)

```
rectangle found:
598 392
665 210
381 107
314 289
=================
rectangle found:
605 1103
815 982
634 668
424 789
=================
rectangle found:
156 705
354 740
403 462
205 427
=================
```