

تمرین دوم بینایی

98522382

نیکی مجیدی فرد

سوال 1)

الف) برای پیاده سازی عملگر سوبل ، کرنل های لبه ی افقی و لبه های عمودی را در عکس اصلی convolve می کنیم.

G_y				G_x			
-1	-2	-1		-1	0	+1	
0	0	0		-2	0	+2	
+1	+2	+1		-1	0	+1	

$$\text{mag} = \sqrt{g_x^2 + g_y^2}$$

$$\text{dir} = \text{atan2}(g_y, g_x)$$

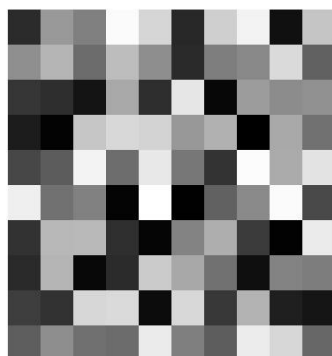
*برای پیاده سازی convolution، ابتدا کرنل را برگردانده و عکس اصلی را با کمک پدینگ (np.pad) کمی از هر دو طرف به اندازه لازم برای کرنلمان گسترش می دهیم. سپس هر عنصر از عکس را در کرنل آن ناحیه ضرب کرده و تمام مقادیر ضرب شده در آن ناحیه را جمع می کنیم.

ماتریس اندازه و ماتریس جهت گرادیان با کمک فرمول های بالا به راحتی قابل محاسبه است.

نتیجه:

ماتریس نمونه:

```
[ [ 45 154 129 251 214 43 207 244 18 197]
  [144 181 109 189 143 44 127 138 218 99]
  [ 56 48 22 170 48 230 9 157 141 146]
  [ 30 5 198 216 211 153 177 2 170 115]
  [ 73 93 243 110 232 120 53 252 171 228]
  [239 115 130 6 255 3 98 139 251 75]
  [ 52 184 185 49 7 133 174 61 4 235]
  [ 44 181 10 45 203 169 115 17 133 126]
  [ 64 52 215 218 14 216 58 180 33 23]
  [ 95 143 113 109 235 128 94 236 215 104]]
```



ماتریس بالا با کرنل لبه های افقی :

```
[[-364. -217. -299. -289. 769. 37. -697. 476. 180. -418.]
[-175. 20. -235. -179. 438. 78. -316. -125. 136. 54.]
[ 4. -65. -463. -99. 88. 128. 203. -348. -52. 164.]
[ 38. -472. -561. -41. 56. 286. 243. -236. -191. 48.]
[ 109. -399. -136. -116. 46. 549. -249. -382. -1. 117.]
[ 96. -85. 336. -61. -88. 326. -332. -254. -22. 64.]
[-277. -123. 515. 38. -289. -89. 160. 169. -393. -279.]
[-394. -216. 241. -7. -330. -35. 412. 159. -235. -207.]
[-161. -286. -162. 87. -139. 141. 116. -89. 337. 138.]
[-132. -205. -64. -165. -55. 379. -288. -338. 553. 343.]]
```

لبه های عمودی :

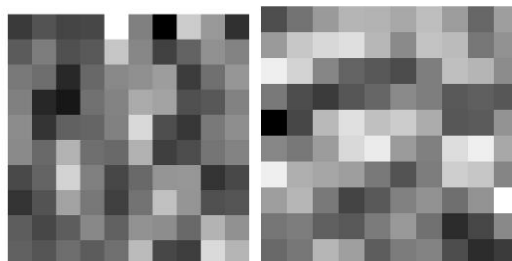
```
[[-324. -133. 75. 215. 203. 149. 265. 92. -196. 94.]
[ 73. 308. 401. 435. 226. -10. 296. 249. -108. 30.]
[ 518. 377. -29. -211. -272. -336. -73. 270. 216. 0.]
[ -96. -328. -427. -285. -198. -8. -73. -264. -237. -276.]
[ -737. -361. 236. 444. 272. 335. 171. -276. -259. 39.]
[ -28. -103. 86. 405. 498. 78. -64. 428. 518. 146.]
[ 519. 183. 135. 94. -101. -297. -78. 345. 307. -35.]
[ 96. 222. -97. -375. -266. -57. 30. -151. 35. 607.]
[ -115. -78. -232. -263. -87. 71. -136. -499. -361. -16.]
[-184. -111. 222. 99. -245. -81. -40. -330. -501. -425.]]
```

اندازه و جهت گرادیان :

```
direction in degree
[[-131.67259576 -121.50426672 -75.91867725 -53.35279443 75.21243683
 13.94572888 -69.18315332 79.06090882 137.43664825 -77.32613999]
[ -67.35689133 3.71528911 -30.37179168 -22.36688441 62.70709241
 97.30575953 -46.87174497 -26.65701864 128.45370922 60.9453959 ]
[ 0.44242966 -9.78240703 -93.58403853 -154.8642974 162.07208024
 159.14554196 109.77888481 -52.19347006 -13.53585637 90. ]
[ 158.40468955 -124.79602628 -127.27624388 -171.81361497 164.20759647
 91.60226135 106.72087344 -138.20521593 -141.13440994 170.13419306]
[ 171.58711895 -132.13759477 -29.95360817 -14.64188454 9.59890307
 58.60844293 -55.52078431 -125.84860398 -179.77878187 71.56505118]
[ 106.26020471 -140.46909497 75.64322405 -8.56535198 -10.02110598
 76.54415702 -100.91112838 -30.68734899 -2.43194995 23.67053848]
[ -28.08967395 -33.90627699 75.31121323 22.0112832 -109.26354902
 -163.31845604 115.98923358 26.0981576 -52.00409684 -97.15029018]
[ -76.30643055 -44.2151754 111.92427373 -178.93060298 -128.87091924
 -148.44861505 85.83532735 133.52172734 -81.52885537 -18.83047553]
[ -125.53767779 -105.2551187 -145.07433273 161.69586323 -122.04241991
 63.27263827 139.53777251 -169.88725089 136.96927964 96.61346048]
[ -144.3446719 -118.4339241 -16.08165196 -59.03624347 -167.3474435
 102.06378081 -97.9071627 -134.3138558 132.17555336 141.09442926]]
```

اندازه:

```
[[487.310989 254.51522548 308.26287483 360.2027207 795.34269343
153.52524222 745.67687372 484.80924084 266.11275806 428.43902717]
[189.61540022 308.64866758 464.78597225 470.38920056 492.86915099
78.63841301 432.98036907 278.61442892 173.66634677 61.77378085]
[518.01544379 382.56241321 463.90731833 233.07080469 285.88109416
359.55528087 215.72667892 440.45885165 222.17110523 164.
]
[103.24727599 574.77647829 705.01773027 287.93402022 205.76685836
286.11186623 253.72820103 354.10732836 304.3846251 280.14282072]
[745.01677833 538.07248582 272.38208458 458.90303987 275.86228448
643.13762135 302.06290736 471.27486672 259.00193049 123.32882875]
[100. 133.54400024 346.83137113 409.56806516 505.71533495
335.20143198 338.11240734 497.69468553 518.46697098 159.41141741]
[588.29414411 220.4948979 532.40022539 101.39033485 306.14049062
310.04838332 178. 384.16923354 498.69630037 281.18677067]
[405.52681786 309.74182798 259.78837541 375.06532764 423.85846694
66.88796603 413.09078905 219.27608169 237.59208741 641.32519052]
[197.85348114 296.44561053 282.96289509 277.01624501 163.9817063
157.86703266 178.75122377 506.87473798 493.85220461 138.92443989]
[226.45087768 233.12228551 231.04112188 192.42141253 251.09759059
387.55902776 290.76450953 472.381202 746.19702492 546.14466948]]
```



Sobelx

sobely

ب) با کمک تعریفی که از فیلتر گاوسی ارایه شده، یک فیلتر گاوسی میسازیم. با کمک این فیلتر تصویر به اصطلاح نرم و smooth تر می شود و در نتیجه کمک بهتری می کند که لبه های مهم تر آشکار و مشخص تر باشند.

از کرنل گاوسی با اندازه 7*7 انتخاب کردیم.

sobel without smoothing



sobel with smoothing



```
def opencv_sobel(image, title):
    sobely = cv.Sobel(src=image, ddepth=-1, dx=0, dy=1, ksize=3, borderType=cv.BORDER_CONSTANT)
    sobelx = cv.Sobel(src=image, ddepth=-1, dx=1, dy=0, ksize=3)
    sobel = cv.addWeighted(sobelx, 0.5, sobely, 0.5, 0)
    show_img(sobel, title)
```

opencv implementation



opencv implementation with smoothing filter



```
cv.Sobel(src, ddepth, dx, dy, dst[, ksize[, scale[, delta[, borderType]]]])
```

Src = main picture , dx = derivative x , dy = derivative y , ksize = kernel size , bordertype = padding border ,

ddepth :

عمق تصویر . در صورت - عمق تصویر 1 دقیقاً شبیه سورس می شود

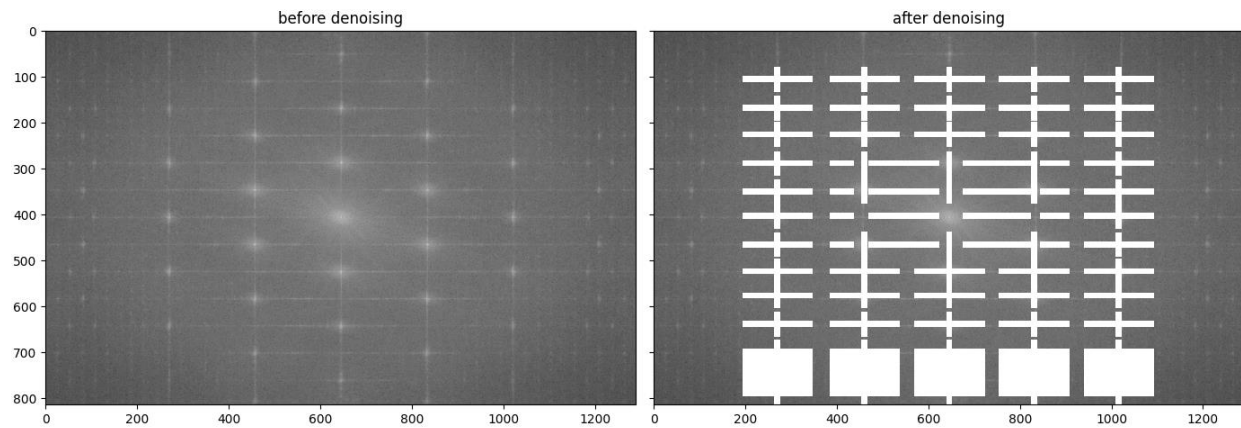
سوال دوم)

(الف)

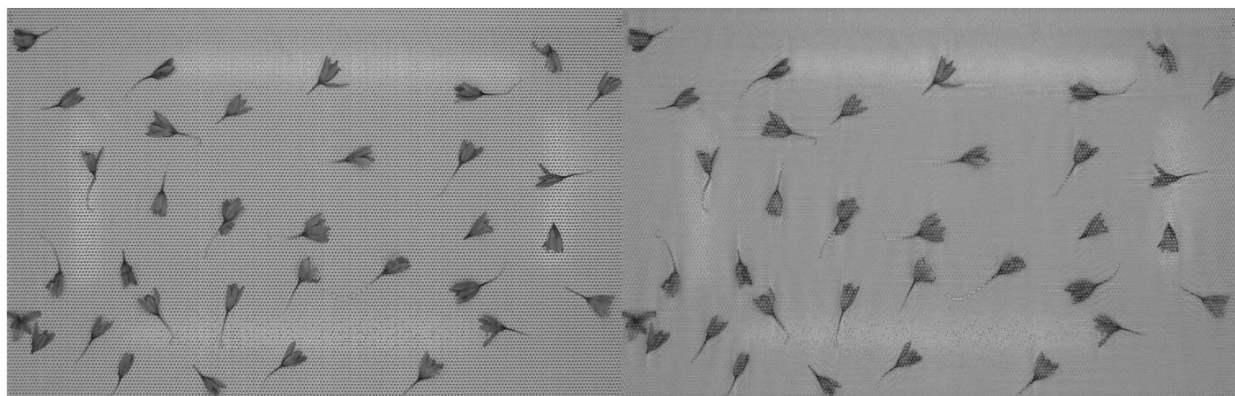
به صورت کلی ، اول عکس را با کمک تابع fft به فرمت فرکانسی آن درمی آوریم. سپس شکل آن را رسم می کنیم (لگاریتم مقدار abs آن) و آن را برای خوانایی بهتر به وسط تصویر شیفت می دهیم. حالا لازم است که آن نقاطی که نویز را به وجود آورده اند از بین ببریم.

همان طور که مشخص است روی تصویر نقاطی حاصل از برخورد خطوط فرکانسی سفید در راستای ایکس و وای هستند. نویز ما متناوب ، در تمام صفحه و در هر دو جهت x و y است، پس حدس زده می شود که در نزدیکی این نقاط خیلی روشن فرکانس نقاط متناوب باشد.

در راستای ایکس و ایگرگ هر کدام از خطوط را در یک ارایه میریزیم و یک فور تو در تو می زنیم، در نقاط تقاطع نزدیک به مرکز عکس را سعی کرده که تا جای ممکن دینویز نکرده ، چرا که به علت شیفت کردن آن به وسط ، بیشتر اجزای شکل مربوط به وسط تصویر است. اما نقاط روشنی که به صورت افقی و عمودی تکرار شده اند و اطراف آن ها (در راستای خطوط) فرکانس های بالای غیر عادی دیده میشود



نتیجه تصویر (سمت چپ تصویر اصلی و سمت راست بعد از عملیات نویز گیری):



(ب) همان طور که میدانیم، لبه یاب کنی یک لبه یاب باینری است که از چهار مرحله تشکیل شده است.

فیلتر گاوسی، گرادیان، حذف مقادیر غیر بیشینه و آستانه گذاری چند مرحله ای.

در مرحله آخر کاملاً مشخص است که به دو حد آستانه احتیاج داریم.

در کد زده شده ابتدا یک فیلتر گاوسی با کرنل 3 بر روی تصویر اعمال میکنیم تا طبق مراحل پیش برود و از طرفی نویزها را نیز کم کند.

از لبه یاب خود OpenCV استفاده می کنیم ک فرمت آن:

```

• Canny() [2/2]
void cv::Canny ( InputArray  dx,
                 InputArray  dy,
                 OutputArray edges,
                 double       threshold1,
                 double       threshold2,
                 bool         L2gradient = false
                 )

```

```
denoised_image = cv.GaussianBlur(denoised_image, (5 ,5), 0)
```

```
pic = np.uint8(denoised_image)
```

```
edges = cv.Canny(pic,100,240)
```

در نمونه ی بالا ، pic تصویر نرم شده با فیلتر گاوسی است. pic تصویر ورودی و دو عدد دیگر به ترتیب ترشولد های لبه یاب (طبق تعریف) هستند.

canny



برای به دست آوردن گرادیان طبق شکل سوال 1 از سوبل در جهت افقی و عمودی استفاده می کنیم.

```

sx = cv.Sobel(src=denoised_image, ddepth=-1, dx=1, dy=0, ksize=3)
sy = cv.Sobel(src=denoised_image, ddepth=-1, dx=0, dy=1, ksize=3)
sxy = cv.Sobel(src=denoised_image, ddepth=-1, dx=1, dy=1, ksize=7)

```

SXY، سوبل در جهت X, Y است.

تابع $\arctan2(x, y)$ مشتق د راستای ایکس و ایگرگ را می گیرد و زاویه آن را محاسبه می کند. اندازه گرادیان را به صورت یک ارایه بر می گرداند. و هر مقدار اندازه گرادیان پیکسل تصویر SXY است.

```

gr_direction = np.arctan2(sx , sy) *180 /np.pi

```

```

gradient value [[ 0. 90. 90. 90. ... 90.
 90. 0. ]
 [ 180. 97.20863686 91.47174657 ... 103.72966104
 101.91598423 180. ]
 [ 180. 111.01265972 113.39183999 ... 103.38872056
 115.4189438 180. ]
 ...
 [ 0. -18.33976865 -38.1704262 ... 174.77105245
 -170.90375844 180. ]
 [ 0. 17.43039839 -36.78234575 ... 145.72701812
 165.96911186 180. ]
 [ 0. 90. -90. ... 90.
 90. 0. ]]
```

(د) می توان از جهت گرادیان های مشابه در نقاط نزدیک به هم تصویر و لبه ها و مقایسه عدد گرادیان آن ها دم ها را شناسایی کرده چرا که آن ها کنار هم یک خط را تشکیل میدهند. در نقاط نزدیک به هم و لبه و با کمک الگوریتم نقطه یابی ، جهت گرادیان های نزدیک به هم و در یک راستا هر یک مال یک دم زعفران است.

سوال (3)

الف

فیلتر پایین گذر: تغییرات را بر روی پیکسل هایی با فرکانس بالا اعمال می کند و از پیکسل هایی با فرکانس پایین عبور می کند. در نتیجه تصویری smooth خواهیم داشت.

فیلتر بالاگذر: تغییرات را بر روی پیکسل هایی با فرکانس پایین اعمال می کند و از پیکسل هایی با فرکانس بالا عبور می کند. در نتیجه جزئیات تصویر مشخص تر می شود. ب) فیلتر بالاگذر - چون جزئیات تصویر مشخص تر شده است و مات شده نیست.

ج) نویز های جمع شونده نویز های هستند که با تابع عکس جمع میشوند. عموماً نویز های هستند که تشخیص و رفع آن ها بهتر است. همچنین تأثیری روی خود عکس نمی گذارد. نویز های جمعی با کمک فیلتر های پایین گذر و یا روش های geometric enhancement می توانند تا حد خوبی از بین بروند. نویز های گوسی نمونه ای از نویز جمع شونده است.

نویز ضربی نویز های هستند که ضرب می شوند، پیچیده تر هستند و رفع آن ها سخت تر است و تأثیر زیادی روی عکس می توانند بگذرانند. نویز اسپیکل نوعی از نوع نویز ضربی است. این دسته از نویزها بیشتر در تصاویر راداری اتفاق می افتند.

د) این نویز فقط در دو مقدار دارد، یا صفر یا 255 است. یا یک سیگنال را (صفر می کند) و یا یک می کند و چیزی بین آن نیست. (مقدار نویز را با مقدار سیگنال جایگزین می کند. از راه حل هایی که می توان استفاده کرد، استفاده از فیلتر میانه گیر median filter است تا نویز های 0-1 انتخاب نشوند.

سوال (4)

(4)

FARABOOM
Open Innovation Platform

$$F(u, v) = \sum_{n=0}^{M-1} \sum_{r=0}^{N-1} f(n, r) e^{-j2\pi \left(\frac{un}{M} + \frac{vr}{N} \right)} \quad (ب)$$

4	0
3	2

$N=2 \quad M=2$

$$F(0,0) = \cancel{f(0,0)} + \cancel{f(0,1)} + \cancel{f(1,0)} + \cancel{f(1,1)} = 9$$

$$F(0,1) = \cancel{f(0,0)} + \cancel{f(0,1)} \times e^{-j2\pi \left(\frac{0}{2} \right)} + \cancel{f(1,0)} \times e^{-j2\pi \left(\frac{1}{2} \right)} + \cancel{f(1,1)} \times e^{-j2\pi \left(\frac{0}{2} + \frac{1}{2} \right)} = 4 + 0 + 3 - 2 = 5$$

$$F(1,0) = \cancel{f(0,0)} + \cancel{f(0,1)} \times e^{-j2\pi \left(\frac{1}{2} \right)} + \cancel{f(1,0)} \times e^{-j2\pi \left(\frac{1}{2} \right)} + \cancel{f(1,1)} \times e^{-j2\pi \left(\frac{1}{2} \right)} = 4 + 0 + (-3) + (-2) = -1$$

$$F(1,1) = \cancel{f(0,0)} + \cancel{f(0,1)} \times e^{-j2\pi \left(\frac{1}{2} \right)} + \cancel{f(1,0)} \times e^{-j2\pi \left(\frac{1}{2} \right)} + \cancel{f(1,1)} \times e^{-j2\pi \left(\frac{1}{2} + \frac{1}{2} \right)} = 4 + (-3) + 2 = 3$$

الف) نقطه میانه $\sqrt{0,0}=0,0=0$ است پس مقدار $F(u,v)$ مجموع

مقادیر $P(n,r)$ است

$$\Rightarrow F(0,0) = \underset{(1)}{f(0,0)} + f(0,1) + f(1,0) + f(1,1) = 9$$

سوال 5)

کانتور ها را می توان به سادگی به عنوان یک منحنی توضیح داد که تمام نقاط پیوسته (در امتداد مرز) را با رنگ یا شدت یکسان به هم می پیوندد.

با کمک این تابع همه ی این نقاط در یک تصویر را می توانیم پیدا کنیم.

```
def find_contour(img):
    contours, hierarchy = cv.findContours(img, cv.RETR_TREE,
cv.CHAIN_APPROX_SIMPLE)
    return contours
```

inputتابع یک تصویر است که حتما لازم است سیاه و سفید باشد و برای نتیجه گیری از یک تصویر باینری استفاده کنیم.

برای به دست آوردن این تصویر می توان از لبه یاب canny استفاده کرد.

cv.retr_tree تمام کانتور ها و کانتور های تو در تو را به دست می آورد.

cv.chain_approx_simple : بخش های افقی، عمودی و مورب را فشرده می کند و تنها نقاط انتهایی آنها را باقی می گذارد. به عنوان مثال، یک کانتور مستطیلی به سمت راست با 4 نقطه کدگذاری می شود.

```
def canny_edge(image):
    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    return cv.Canny(gray, 100, 150)
```

*** استفاده از لبه یاب کنی برای استفاده از کانتورینگ.

(ب)

1- اجرای canny بر روی تصویر

2- اجرای find_contour بر روی تصویر کنی.

3- استفاده از تابع cv.approxpolydp() برای پیدا کردن شکل و محیط تصویر.

فرآیند تقریب یک شکل کانتور به شکل دیگری متشکل از تعداد کمتری از رئوس به گونه ای که فاصله خطوط خطوط برابر با دقت مشخص شده یا کمتر از دقت مشخص شده باشد، تقریب یک شکل نامیده می شود. کانتور و یک تابع داخلی در OpenCV برای تقریب شکل منحنی های چند ضلعی به دقت مشخص به نام تابع approxPolyDP (استفاده می شود و تابع approxPolyDP) کانتور تقریبی را که شکل آن مشابه منحنی ورودی و تقریبی یک شکل است برمی گرداند.

approxPolyDP(input_curve, epsilon, closed)

که در آن input_curve چند ضلعی ورودی را نشان می دهد که کانتور آن باید با دقت مشخص تقریبی شود.

```
perimeter = cv.arcLength(c, True)

approx = cv.approxPolyDP(c, 0.01 * perimeter, True)
```

اپسیلون شان دهنده حداکثر فاصله بین تقریب یک کانتور شکل از چند ضلعی ورودی و چند ضلعی ورودی اصلی است.

- ابتدا بر روی کانتور های تصویر، یک فور میزنیم.
- 2- برای هر کانتور یک تابع approx. می نویسیم تا نقاط تقریبی شکل را برای ما تقریب دهد. ضرب اپسیلون $0.01 * perimeter$ خواهد بود.

- حال برای نقاط به دست آمده از **approx**، سعی می کنیم اشکال را تشخیص دهیم. در صورتی که شکل سه گوشه ای باشد، در کلاس مثلث است. در صورتی که 4 نقطه ای باشد دو حالت وجود دارد. با کمک **cv2.boundingrect** اندازه طول اضلاع را پیدا میکنیم. در صورتی که نسبت اضلاع 1 باشد مربع و در غیر این صورت مستطیل است. اگر هم تعداد نقطه ها زیاد باشد دایره است. برای تمام این حالات **if else** می نویسیم و از یک دیکشنری برای ذخیره سازی نوع شکل و نوع شکل به عنوان کلید استفاده میکنیم.

```
if len(approx) == 3:
    if 'triangle' not in final:
        final['triangle'] = (approx , (m,n))
elif len(approx) == 4:
    x, y, w, h = cv.boundingRect(c)
    r = w/h
    if r>= 0.95 and r < 1.05:
        if 'square' not in final:
            final['square'] = (approx , (m,n))
        else:
            if 'square' not in final:
                final['rectangle'] = (approx , (m,n))
    else:
        final['circle'] = (approx , (m,n))
```

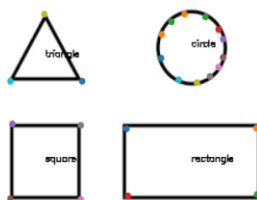
- برای نوشتن اسم هر شکل داخل کلاس، ابتدا نقطه ی وسط شکل را میابیم و آن را به صورت تاپل با نقاط کلیدی هر شکل، به عنوان مقدار به دیکشنری میدهیم.

```
M = cv.moments(c)
if M['m00'] != 0.0:
    m = int(M['m10']/M['m00'])
    n = int(M['m01']/M['m00'])
```

- در مرحله آخر با کمک **plt.scatter()** و **cv.putText()** نقاط را مشخص کرده و اسم هر یک در وسط شکل نوشته می شود.

```
edges = shape_founder(image)
for e in edges.values():
    for p in e[0]:
        plt.scatter([x for x, y in p], [y for x, y in p])
for shape in edges :
    cv.putText(image, shape, edges[shape][1],cv.FONT_HERSHEY_SIMPLEX, 0.25, (0, 0, 0), 1)
```

در نهایت نتیجه به صورت زیر است:



د) یکی از مهم ترین ویژگی ها میتواند تعداد ریوس هر شکل باشد که هر شکل اصلی را از دیگری متمایز می کند. در بین تمام اشکال چند ضلعی، در زاویه بین خطوط و اندازه اضلاع تفاوت دارند و باعث تفاوت شده است. ما با کمک این روش به راحتی می توانیم اشکال چند ضلعی را تشخیص دهیم و در حالت خاص برخی از اشکال مانند اضلاع برابر یا زوایای یکسان می توان با محاسبه آن ها ، متمایز و تشخیص داد.

سوال (6)

الف (تابع reflect101 :

BORDER REFLECT 101: afedcblabcdefahlafedcba

```
def Reflect101(img,filter_size):  
    indx = filter_size//2  
    return np.pad(img, (indx, indx), 'reflect')
```

این نوع پدینگ عینا به کمک کتابخانه numpy به صورت بالا پیاده سازی می شود.

پیاده سازی avrg : برای پیاده سازی این فیلتر باید کرنل میانگین گیری را با تصویر کانوالو کنیم. کرنل میانگیر تمام اجرای آن یک و به اندازه کرنل تقسیم می شوند.

```
def Averaging_Blurring(img, filter_size):  
  
    image = Reflect101(img, filter_size)  
    kernel =1/filter_size* np.ones((filter_size , filter_size))  
    h,w = img.shape  
    m,n = image.shape  
    result = np.zeros(img.shape)  
    print(m,n,h,w)  
    for i in range( w):  
        for j in range(h):  
            if(i+filter_size <= m and j+filter_size <= n):  
                window = image[i:i+filter_size, j:j+filter_size]  
                sum_product = np.sum(window * kernel)  
                result[i,j] = sum_product  
    return result
```

پیاده سازی مدین : برای این کار، برای هر پنجره به اندازه فیلتر خواهیم. در هر پنجره میانه گرفته و به جای پیکسل اصلی قرار میدهم.

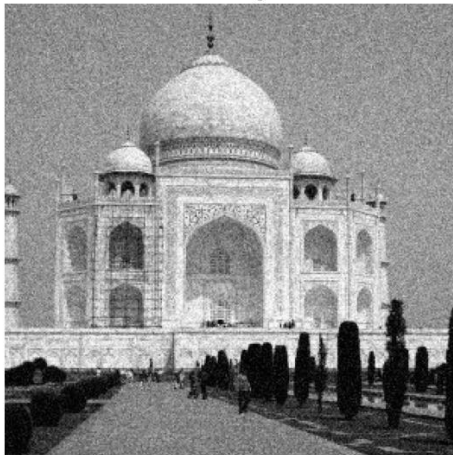
```
def Median_Blurring(img, filter_size):  
    image = Reflect101(img, filter_size)  
    h,w = img.shape  
    m,n = image.shape  
    result = np.zeros((img.shape))  
    for i in range( w):  
        for j in range(h):  
            if(i+filter_size <= m and j+filter_size <= n):  
                window = image[i:i+filter_size, j:j+filter_size]  
                val = np.median(window)  
                result[i,j]= val  
    return result
```

کرنل گوسی: با کمک فرمول زیر آن را تشکیل داده و با کمک filter2d کانوالو می کنیم.

$$G(s,t) = Ke^{-\frac{s^2+t^2}{2\sigma^2}} = Ke^{-\frac{r^2}{2\sigma^2}}$$

```
def Gaussian_Blurring(img, filter_size, std):  
    dst = np.linspace(-1, 1 ,filter_size)  
    x, y = np.meshgrid(dst, dst)  
    d = np.hypot(x,y)  
    gaussian_kernel= np.exp(-( d)**2 / ( 2.0 * (std**2) ) ) )  
    gaussian_kernel= 1/np.sum(gaussian_kernel) *gaussian_kernel  
    output = img.copy()  
    result = cv2.filter2D(src = output, ddepth = -1, kernel = gaussian_kernel)  
    return result
```

main image



Averaging Blurring



Median Blurring



Gaussian Blurring



نتیجه :


در هر یک از این فیلتر ها، با توجه به اعمال کانولوشن بر روی هر یک از آن ها، با بزرگ شدن کرنل پیکسل های بیشتری از اطراف تحت پوشش قرار میگیرند در نتیجه تاثیر کرنل بیشتر شده و با بزرگ شدن آن تصویر تار تر و smooth تر خواهد شد و لبه ها محو تر خواهند بود.

(ب) این فیلتر برای صاف کردن تصاویر و کاهش نویز و در عین حال حفظ لبه ها استفاده می شود. با استفاده از فیلتر های بالا منجر به از دست رفتن اطلاعات مهم لبه می شوند، زیرا آنها همه چیز را محو می کنند، صرف نظر از اینکه نویز یا لبه باشد.

فرمول:

$$GB[I]_p = \sum_{q \in S} G_{\sigma}(\|p - q\|) I_q$$

↓
Normalized Gaussian Function



یا می تواند به حالت زیر بیان شود.


$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(\|I_p - I_q\|) I_q$$

↓

Normalization Factor


↓

Space Weight



↓

Range Weight



Sigma_s اندازه همسایگی، و **sigma_r** نشان دهنده حداقل دامنه یک پال است. این تضمین می کند که تنها پیکسل هایی با مقادیر شدت مشابه پیکسل مرکزی برای تار شدن در نظر گرفته می شوند، در حالی

که تغییرات شدت واضح حفظ می شود. هر چه مقدار **sigma_r** کوچکتر باشد، لبه تیزتر است. همانطور که **sigma_r** به بی نهایت میل می کند، معادله به شکل گوسی متمایل میشود.

P پیکسل همسایه در یک محدوده ی خاص است. **s** پیکسل فعلی است و **lp lq** اندازه ی پیکسل در تصویر هستند که در دو معادله ی گوسی با ضرایب متفاوت جا گذاری می شوند.

برای پیاده سازی نیز از فرمول بالا استفاده کردیم. به طوری که روی تصویر یک پدینگ زده. هر بار تصویر را به اندازه **filter_size** مشخص شده جدا خواهیم کرد و با کمک فرمول گوسی ، هر کدام از ضرایب را

محاسبه و جاگذاری میکنیم.

<https://www.geeksforgeeks.org/python-bilateral-filtering>

```

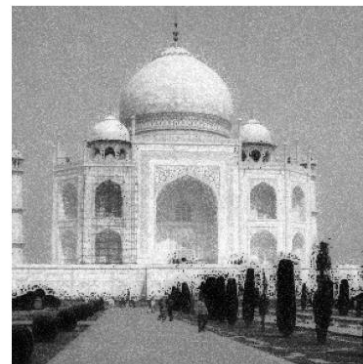
def bilateral_filter(d,im,strd , g):
    sum_num=0
    ks = 0
    for k in range(d):
        for l in range(d):
            gvalue = np.exp(-( (abs(im[k,l] - im[ d//2, d//2] ))**2 / ( 2.0 * (strd**2) ) ) )
            t1= gvalue
            t2 = g[k,l]
            w = t1*t2
            ks +=w
            sum_num += im[k,l] * w
    return round(sum_num/ks)

def Bilateral_Filtering(img, filter_size, std, rstd):
    w,h = img.shape
    result = img.copy()
    img = Reflect101(img , filter_size)
    m,n = img.shape
    dst = np.linspace(-1, 1 ,filter_size)
    x, y = np.meshgrid(dst, dst)
    d = np.hypot(x,y)
    gaussian_kernel= np.exp(-( d)**2 / ( 2.0 * (std**2) ) ) )

    for i in range(w):
        for j in range(h):
            if(i+filter_size <= m and j+filter_size <= n):
                window = img[i:i+filter_size, j:j+filter_size]
                result[i,j]=bilateral_filter(filter_size,window,rstd , gaussian_kernel)

    return result

```



ج) مقایسه با opencv

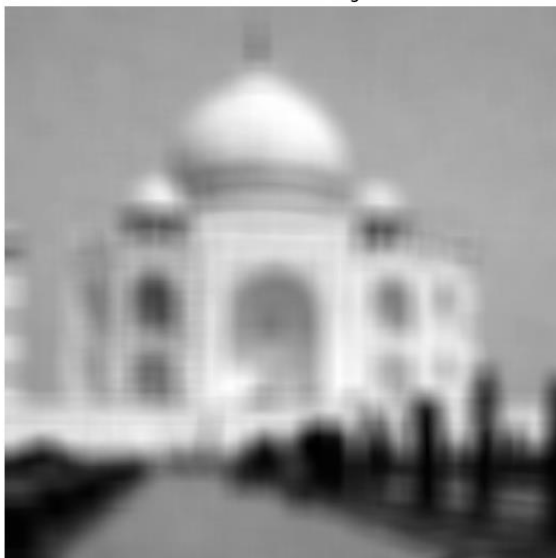
Averaging Blurring



Median Blurring



Gaussian Blurring



Bilateral

