به نام خدا

تمرین سری دوم درس دید کامپیوتری

دانشجو: محمدجواد قربانعلی وکیلی

شماره دانشجویی: ۹۸۲۱۰۲۹۲

استاد: دکتر نرجس‌الهدی محمدزاده

نیمسال تحصیلی دوم ۹۹-۹۸

دانشکده مهندسی مکانیک

دانشگاه صنعتی شریف

تمرین الف-۳

در این تمرین، هدف تغییر سایز شکل ۱ و اعمال یک فیلتر پایین‌گذر به آن و سپس طراحی کرنل فیلترهای لبه‌یاب عمودی و افقی و اعمال آن به شکل مورد نظر می‌باشد. اعمال این فیلترها می‌بایست بدون استفاده از توابع آماده opencv باشد. مراحل انجام این عملیات به تفصیل بیان می‌گردد.

شکل ۱- تصویر اولیه

بخش اول: در اولین بخش از این تمرین، هدف اعمال فیلتر میانگین به عنوان فیلتر پایین‌گذر به تصویر در فرمت RGB می‌باشد. گام‌های انجام این عملیات عبارتند از:

۱) Import کتابخانه‌ها و خواندن تصویر و تغییر سایز آن.

```
import cv2
import numpy as np
##################################LPF in RGB image#############################
pic=cv2.imread('2.jpg')
pic=cv2.resize(pic,(400,300))
```

۲) طراحی کرنل فیلتر میانگین ۳×۳ و سپس ایجاد یک numpy array با المان‌های سه بعدی. علت ایجاد این آرایه، همانطور که در بخش بعدی ذکر خواهد شد، zero pad کردن تصویر اولیه است. هم‌چنین با توجه به آن‌که تصویر اولیه در فرمت RGB است، المان‌های numpy array نیز می‌بایست سه بعدی باشند.

```
#kernel for mean filter
kernel = (1/9)*np.ones((3,3), np.uint8)
#define for zero padding
pic_pad = np.zeros((302,402,3), np.uint8)
```

۳) Zero pad کردن تصویر اولیه

```
#zero padded pic
pic_pad[1:301,1:401]=pic
```

۴) در این گام، هدف اعمال کرنل تعریف شده به تصویر zero pad شده می‌باشد. بدین منظور برای فیلتر کردن هر پیکسل،ابتدا در کد، مرکز کرنل بر روی پیکسل مورد نظر قرار داده شده و سپس عملیات ضرب درایه‌ای و جمع انجام می‌پذیرد. در نهایت نیز تصویر فیلتر شده به صورت شکل ۲ تولید می‌گردد.

```
for x in range(1,301):
    for y in range(1,401):
        #picks a 3*3 section
        section=pic_pad[x-1:x+2,y-1:y+2]

        #element-wise multiplication
        q=kernel*section

        #sum of elements
        pic_pad[x,y]=q[0,0]+q[0,1]+q[0,2]+q[1,0]+q[1,1]+q[1,2]+q[2,0]+q[2,1]+q[2,2]

pic_LPF=pic_pad[1:301,1:401]
```

بخش دوم: در این بخش، هدف اعمال فیلتر میانگین به عنوان فیلتر پایین‌گذر، به شکل ۱ در فرمت gray می‌باشد. لذا تمامی مراحل مشابه مراحل بخش اول می‌باشد، با این تفاوت که در این قسمت با توجه به فرمت تصویر، المان‌های numpy array بایستی یک بعدی باشند. در نهایت تصویر نهایی به صورت شکل ۳ می‌گردد.

```
#####################################LPF in gray scale#################################
pic_gray=cv2.cvtColor(pic,cv2.COLOR_BGR2GRAY)
kernel = (1/9)*np.ones((3,3), np.uint8)
pic_pad = np.zeros((302,402), np.uint8)
pic_pad[1:301,1:401]=pic_gray
for x in range(1,301):
    for y in range(1,401):
        section=pic_pad[x-1:x+2,y-1:y+2]
        q=kernel*section
        pic_pad[x,y]=q[0,0]+q[0,1]+q[0,2]+q[1,0]+q[1,1]+q[1,2]+q[2,0]+q[2,1]+q[2,2]
pic_lpf=pic_pad[1:301,1:401]
```

بخش سوم: در این بخش، هدف اعمال یک فیلتر لبه‌یاب در جهت y (افقی) به تصویر فیلتر شده بخش دوم (شکل ۳) می‌باشد. گام‌های انجام این عملیات به شرح زیر می‌باشد.

۱) Zero pad کردن تصویر اولیه و تعریف کرنل به منظور انجام لبه‌یابی در جهت y. لبه‌های یک تصویر، پیکسل‌هایی هستند که در آن‌ها intensity پیکسل به طور ناگهانی و با شیب زیاد نسبت به پیکسل مجاور تغییر می‌کند. به همین دلیل برای یافتن این لبه‌ها، کرنل مورد نظر طوری طراحی شده که اختلاف intensity یک پیکسل را با پیکسل مجاور در جهت y محاسبه می‌نماید.

```
##############################detect edge in y direction##############################
pic_pad = np.zeros((302,402), np.uint8)
pic_pad[1:301,1:401]=pic_lpf
kernel = np.array([[0,-1,1]])
```

۲) در این گام، هدف اعمال کرنل تعریف شده به تصویر zero pad شده می‌باشد. بدین منظور برای فیلتر کردن هر پیکسل،ابتدا در کد مرکز کرنل بر روی پیکسل مورد نظر قرار داده شده و سپس عملیات ضرب درایه‌ای و جمع انجام می‌پذیرد. در نهایت تصویری تولید می‌گردد که intensity هر پیکسل از آن، متناسب با آهنگ تغییر intensity همان پیکسل در تصویر اولیه است. لذا هرچه این لبه‌ها تیز تر باشند، در تصویر نهایی روشن‌تر خواهند بود. پس از اعمال threshold (که در ادامه شرح داده خواهد شد) تصویر نهایی به صورت شکل ۴ خواهد بود.

```
for x in range(1,301):
    for y in range(1,401):
        section=pic_pad[x,y-1:y+2]
        q=kernel*section
        s=q[0,0]+q[0,1]+q[0,2]
        pic_pad[x,y]=abs(s)
pic_y=pic_pad[1:301,1:401]
```

بخش چهارم: در این بخش، هدف اعمال یک فیلتر لبه‌یاب در جهت x (عمودی) به تصویر فیلتر شده بخش دوم (شکل ۳) می‌باشد. گام‌های انجام این عملیات کاملاً مشابه بخش سوم می‌باشد، با این تفاوت که این‌بار عملیات اعمال کرنل به تصویر به صورت عمودی انجام می‌پذیرد. پس از اعمال threshold (که در ادامه شرح داده خواهد شد) تصویر نهایی به صورت شکل ۵ خواهد بود.

```
##############################detect edge in x direction##############################
pic_pad = np.zeros((302,402), np.uint8)
pic_pad[1:301,1:401]=pic_lpf
kernel = np.array([[0,-1,1]])
for x in range(1,301):
    for y in range(1,401):
```

```
        section=pic_pad[x-1:x+2,y]
        q=kernel*section
        s=q[0,0]+q[0,1]+q[0,2]
        pic_pad[x,y]=abs(s)
pic_x=pic_pad[1:301,1:401]
```

بخش پنجم: فیلترهای لبه یاب بخش‌های سوم و چهارم، اثر تغییر شدید intensity را فقط در یک راستا بررسی می‌کنند و لذا اگر در راستای دیگر لبه وجود داشته باشد (تغییر ناگهانی intensity)، آن را در نظر نمی‌گیرند. لذا در این بخش، هدف در نظر گرفتن هم‌زمان اثر تغییر ناگهانی intensity در هر دو راستا به منظور شناسایی بهینه لبه‌ها می‌باشد. به منظور انجام این کار، مجذور تغییر intensity در هر دو راستا با یکدیگر جمع شده و سپس از عبارت حاصل جذر گرفته می‌شود. پس از اعمال threshold (که در ادامه شرح داده خواهد شد) تصویر نهایی به صورت شکل ۶ خواهد بود.

```
########################################edge detection#####################################
edge = np.zeros((300,400), np.uint8)
for x in range(0,300):
   for y in range(0,400):
      a=int((pic_x[x,y]**2 + pic_y[x,y]**2)**0.5)
      if a>255:
         edge[x,y]=255
      else:
         edge[x,y]=a
```

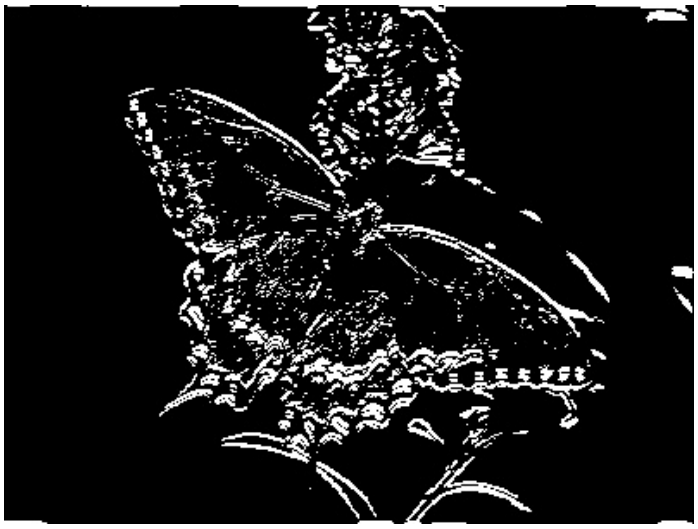
بخش ششم: در این بخش، ابتدا تصاویر حاصل از اعمال فیلترهای لبه‌یاب، به منظور بهبود وضوح تصویر، به کمک دستور threshold به تصویر binary تبدیل شده و سپس دستور نمایش تمام تصاویر نوشته می‌شود. هم‌چنین کاربر با فشردن دکمه e تمام پنجره‌ها را بسته و یا با فشردن دکمه s، ضمن بستن تمامی پنجره‌ها، تصاویر حاصل از فیلترهای لبه‌یاب عمودی و افقی را در directory با نام‌های نوشته شده ذخیره می‌نماید.

```
_,edge=cv2.threshold(edge,10,255,cv2.THRESH_BINARY)
_,pic_y=cv2.threshold(pic_y,10,255,cv2.THRESH_BINARY)
_,pic_x=cv2.threshold(pic_x,10,255,cv2.THRESH_BINARY)
cv2.imshow('INITIAL IMAGE',pic)
cv2.imshow('EDGE',edge)
cv2.imshow('INITIAL GRAY IMAGE',pic_gray)
cv2.imshow('LPF GRAY',pic_lpf)
cv2.imshow('Y EDGE',pic_y)
cv2.imshow('X EDGE',pic_x)
```

```
cv2.imshow('LPF RGB',pic_LPF)
while True:
    key=cv2.waitKey(0)
    if key==ord('e'):
        cv2.destroyAllWindows()
        break
    elif key==ord('s'):
        cv2.imwrite('X.jpg',pic_x)
        cv2.imwrite('Y.jpg',pic_y)
        cv2.destroyAllWindows()
        break
    else:
        pass
```

شکل ۳ - فیلتر میانگین در فرمت gray

شکل ۲- فیلتر میانگین در فرمت RGB



شکل ۵- فیلتر لبه‌یاب عمودی

شکل ۴- فیلتر لبه‌یاب افقی

شکل ۶- فیلتر لبه‌یاب

در شکل ۴ که فیلتر در جهت y (افقی) اعمال شده است، ملاحظه می‌گردد لبه‌هایی که در این جهت قرار دارند به خوبی نمایان شده‌اند در حالی که در شکل ۵ که فیلتر در جهت x (قائم) اعمال شده، لبه‌ها در جهت x به وضوح مشخص هستند (به عنوان مثال در شکل ۴ بر خلاف شکل ۵، ساقه گل به خوبی نمایان است در حالی که در شکل ۵ بر خلاف شکل ۴، قسمت انتهایی گلبرگ سمت چپ به خوبی نمایان می‌باشد). حال در شکل ۶ که اثر توأم این دو فیلتر لبه‌یاب اعمال شده است، ملاحظه می‌گردد که تمامی لبه‌ها با وضوح بیشتری قابل رؤیت هستند. در واقع شکل ۶، تصویر بالاگذر شده شکل ۱ در فرمت binary می‌باشد.

تمرین الف-۴

در این تمرین، هدف اعمال فیلترهای لبه‌یاب Sobel، canny و LoG به شکل‌های ۱ و ۷ و مقایسه نتایج حاصل از آن می‌باشد. همانطور که پیش‌تر گفته شد، تمامی فیلترهای لبه‌یاب، لبه‌ها را از طریق محاسبه آهنگ تغییر intensity در هر پیکسل تشخیص می‌دهند. حال هرچه آهنگ تغییر intensity بزرگتر باشد، این لبه‌ها واضح‌تر نمایان می‌شوند.

آنچه در این‌جا مسأله است این است که تغییر intensity در تمام نواحی تصویر وجود دارد و چه بسا لبه‌هایی توسط فیلترهای لبه‌یاب شناسایی گردد که مد نظر نیستند. برای کاهش این‌گونه خطاها و در نتیجه حذف نویز در تصویر نهایی و افزایش وضوح آن، معمولاً پیش از انجام عملیات لبه‌یابی، یک فیلتر پایین گذر به تصویر اعمال می‌گردد. در کدهای نوشته شده در این قسمت نیز همواره تصویر پایین گذر شده به عنوان ورودی به فیلترهای لبه‌یاب داده می‌شود.

مراحل اعمال فیلترهای مذکور به شکل‌های ۱ و ۷ و بررسی نتایج آن به تفصیل در ادامه ذکر خواهد شد. اما پیش از آن به نحوه عملکرد این فیلترهای لبه‌یاب پرداخته می‌شود.

در فیلتر Sobel، که تابع آماده آن به صورت cv.Sobel() می‌باشد، بسته به سایز فیلتر که توسط کاربر تعیین می‌گردد، آهنگ تغییر intensity در هر دو راستای x و y قابل محاسبه است (توجه شود در توابع آماده‌ای که در این تمرین مورد استفاده قرار خواهند گرفت، راستاهای x و y عکس تمرین الف-۳ می‌باشد). به عنوان مثال در صورتی که سایز کرنل ۳ باشد، کرنل‌های محسابه آهنگ تغییر (مشتق مرتبه اول) در دو راستای x و y به صورت زیر خواهد بود.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

همانطور که از کرنل‌های فوق مشخص است، فیلتر Sobel ترکیبی از مشتق‌گیری و یک فیلتر پایین‌گذر است. پس از اعمال این کرنل‌ها، آهنگ تغییر intensity در هر پیکسل از رابطه زیر محاسبه می‌گردد.

$$G = \sqrt{G_x^{\,2} + G_y^{\,2}}$$

در ورودی تابع cv.Sobel() علاوه بر سایز کرنل و تصویر ورودی، مرتبه مشتق‌ها در راستای x و y نیز به کمک دو عدد مشخص می‌شوند. همچنین data type پیکسل‌ها نیز قابل تنظیم می‌باشد.

در فیلتر Laplacian، عملیات لبه‌یابی از طریق محاسبه لاپلاسین یا همان مجموع مشتقات دوم در راستای x و y محاسبه می‌گردد. توجه شود در اینجا نیز برای محاسبه مشتقات، از کرنل‌های Sobel استفاده می‌گردد. مشابه فیلتر Sobel، در فیلتر Laplacian نیز data type پیکسل‌ها قابل تنظیم می‌باشد.

در فیلتر canny، ابتدا تصویر توسط یک فیلتر میانگین گوسی ۵×۵، پایین‌گذر می‌شود. سپس مشتقات اول با همان الگوریتم Sobel محاسبه شده و در رابطه $G = \sqrt{G_x^{\,2} + G_y^{\,2}}$ قرار داده می‌شود. از اینجا به بعد تفاوت اصلی این فیلتر با فیلترهای قبلی

مشخص می‌شود. در حقیقت کاربر قادر است با تعیین یک حد بالا و پایین برای آهنگ تغییر intensity، اثر نویز و لبه‌هایی که مطلوب نیستند را کاهش دهند.

شکل ۷- تصویر اولیه

بخش اول: در این بخش از این تمرین، هدف اعمال فیلترهای سه‌گانه فوق به تصویر ۱ می‌باشد.

۱) اعمال فیلتر LoG. در این فیلتر (که مخفف Laplacian of Gaussian می‌باشد) ابتدا بر روی تصویر، فیلتر میانگین گوسی اعمال می‌گردد. فیلتر میانگین گوسی یک فیلتر پایین‌گذر است. سپس فیلتر Laplacian به تصویر اعمال می‌گردد.

در این مثال، تغییر سایز کرنل فیلتر گوسی، اثر چندانی بر بهبود وضوح تصویر نهایی نخواهد داشت اما تغییر data type پیکسل‌ها از cv2.CV_64F (۶۴ بیتی) به cv2.CV_8U (۸ بیتی) و سپس تبدیل تصویر به binary، اثر بسیار زیادی در کاهش نویز و افزایش وضوح لبه‌ها در پی خواهد داشت. مقایسه شکل‌های ۸ و ۹ به وضوح بیانگر این امر می‌باشد.

```
#################################### LoG ###################################
pic=cv2.imread('2.jpg')
pic_gray=cv2.cvtColor(pic,cv2.COLOR_BGR2GRAY)
pic_blur=cv2.GaussianBlur(pic_gray,(5,5),0)
edge_LoG_64F = cv2.Laplacian(pic_blur,cv2.CV_64F)
edge_LoG_8U = cv2.Laplacian(pic_blur,cv2.CV_8U)
_,edge_LoG_8U=cv2.threshold(edge_LoG_8U,5,255,cv2.THRESH_BINARY)
```

۲) اعمال فیلتر Sobel. در این تصویر، تغییر سایز کرنل در هیچ‌کدام از فیلترهای Sobel نوشته شده، اثر چندانی در بهبود کیفیت خروجی نخواهد داشت. هم‌چنین تغییر data type در فیلترهای راستای x و y اثر مناسبی در پی نخواهد داشت. شکل‌های ۱۰ و ۱۱ نشاندهنده فیلتر Sobel در دو راستای x و y می‌باشد. اما در فیلتر Sobel که مشتق در

هر دو راستا را در بردارد، تغییر data type پیکسل‌ها از cv2.CV_64F (۶۴ بیتی) به cv2.CV_8U (۸ بیتی) اثر بسیار مطلوبی در نتیجه نهایی خواهد داشت. مقایسه شکل‌های ۱۲ و ۱۳ به وضوح بیانگر این امر می‌باشد.

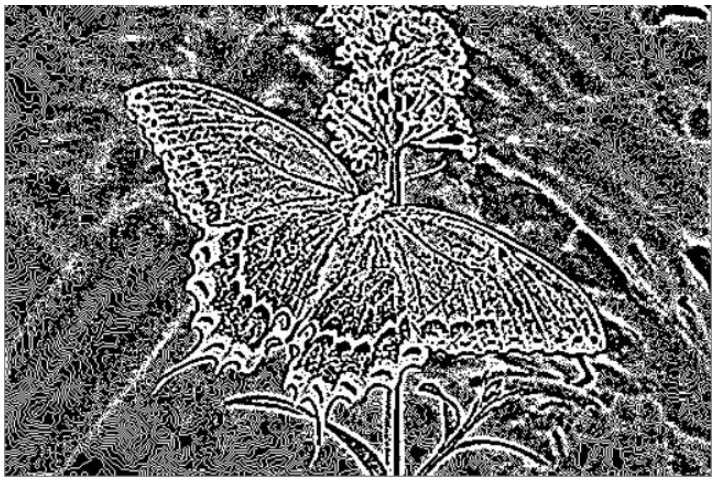
```
########################################## Sobel ###################################
edge_x=cv2.Sobel(pic_blur,cv2.CV_64F,1,0,ksize=5)
edge_y=cv2.Sobel(pic_blur,cv2.CV_64F,0,1,ksize=5)
edge_sobel_8U=cv2.Sobel(pic_blur,cv2.CV_8U,1,1,ksize=5)
edge_sobel_64F=cv2.Sobel(pic_blur,cv2.CV_64F,1,1,ksize=5)
```

۳) اعمال فیلتر canny. در این فیلتر حدود بالا و پایین با آزمون و خطا تعیین شده است. شکل ۱۴ نتیجه استفاده از این فیلتر را نشان می‌دهد.

```
####################################### Canny ###################################
edge_canny = cv2.Canny(pic_gray,100,200)
```

۴) نمایش تمامی خروجی‌ها

```
cv2.imshow('LoG_64F',edge_LoG_64F)
cv2.imshow('LoG_8U',edge_LoG_8U)
cv2.imshow('SobelX',edge_x)
cv2.imshow('SobelY',edge_y)
cv2.imshow('Sobel_64F',edge_sobel_64F)
cv2.imshow('Sobel_8U',edge_sobel_8U)
cv2.imshow('Canny',edge_canny)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



شکل ۸- LoG-64F

شکل ۹- LoG-8U

شکل ۱۰- Sobel(x)

شکل ۱۱- Sobel(y)

شکل ۱۲- Sobel-64F

شکل ۱۳- Sobel-8U

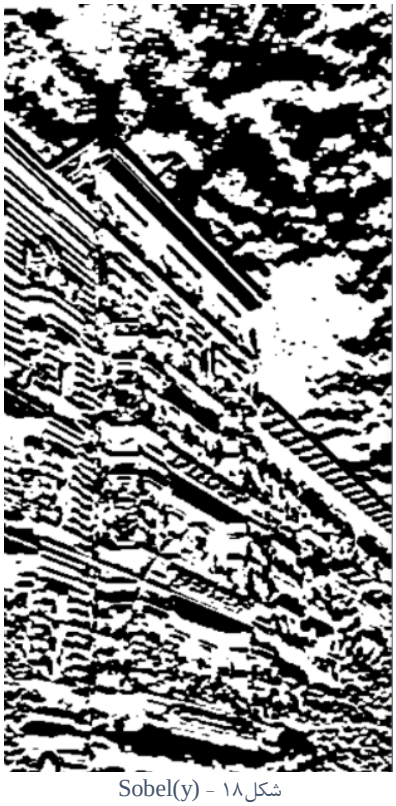
شکل ۱۴- canny

بخش دوم: در این بخش، هدف اعمال فیلترهای لبه‌یاب سه‌گانه به شکل ۸ می‌باشد. تمامی مراحل انجام این کار و کد نوشته شده برای این بخش مشابه بخش قبل می‌باشد و لذا از توضیحات و ارائه مجدد کدها صرف نظر شده و صرفاً تصاویر نهایی در قالب شکل‌های ۱۵ تا ۲۱ نمایش داده می‌شوند.

شکل ۱۵- LoG-64F

شکل ۱۶- LoG-8U

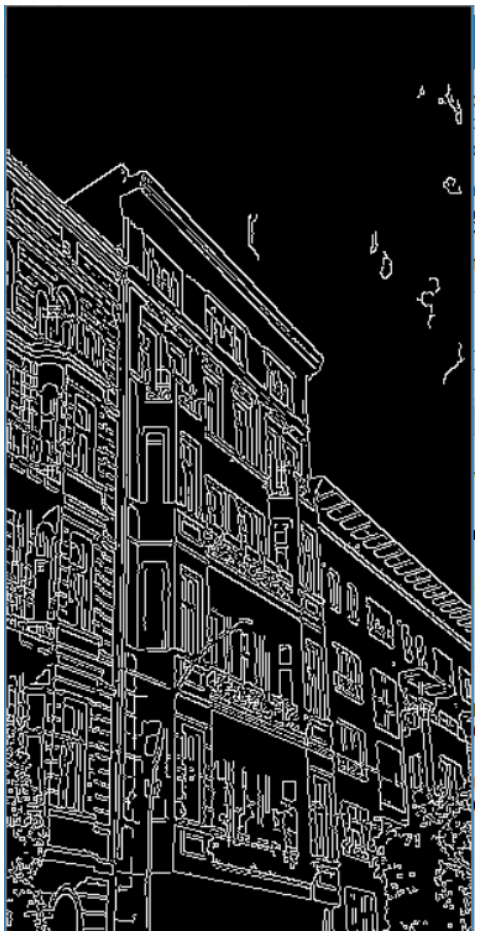
شکل ۱۷- Sobel(x)



شکل۱۸ - Sobel(y)

شکل۱۹ - Sobel-64F

شکل ۲۰ - Sobel-8U



شکل ۲۱- canny

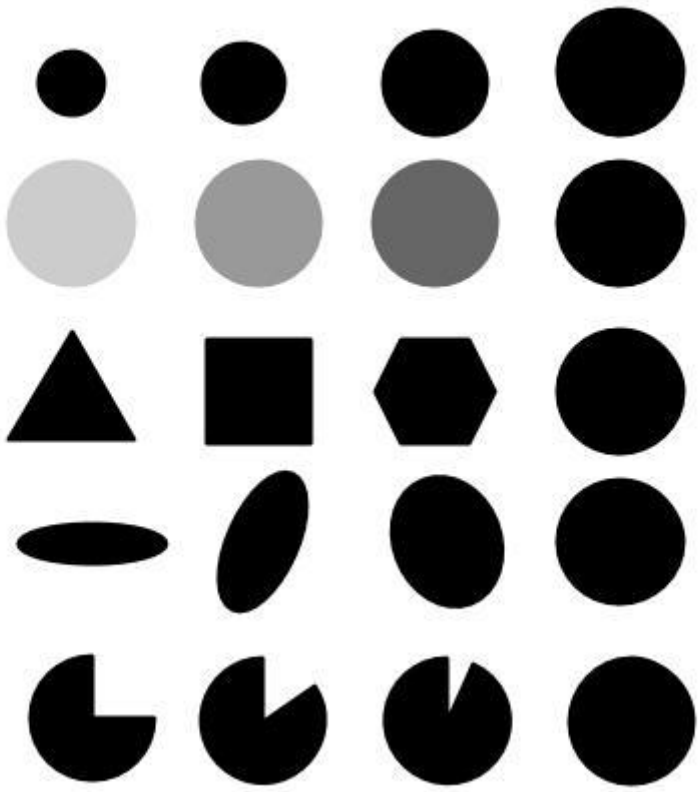
در هریک از نتایج حاصل از لبه‌یابی شکل‌های ۱ و ۷، فیلترهای Sobel در راستای x و در راستای y نتیجه مطلوبی نخواهد داشت. چرا که همانطور که ذکر شد، هر یک از این فیلترها، لبه‌ها را تنها در یک جهت شناسایی می‌کنند. این ایراد در فیلترینگ شکل ۷ که در آن لبه‌ها عمدتاً افقی هستند به وضوح قابل رؤیت است. هم‌چنین تصاویر خروجی از فیلتر Sobel به دلیل آن‌که در این تابع هیچ مکانیزمی برای مقابله با overflow شدن داده‌های intensity پیش‌بینی نشده، دارای نویزهایی با ابعاد بزرگ هستند.

هم‌چنین همانظور که پیش‌تر ذکر شد، تغییر data type پیکسل‌ها از cv2.CV_64F (۶۴ بیتی) به cv2.CV_8U (۸ بیتی) در برخی موارد موجب بهبود چشمگیر وضوح لبه‌یابی گردید.

در مجموع نتایج حاصل از فیلترهای LoG و canny به دلایلی که در ابتدای این تمرین ذکر شد، بالاترین وضوح لبه‌یابی را دارا می‌باشند. هم‌چنین در فیلتر canny، ضخامت لبه‌ها و نویز تصویر در خروجی، در مقایسه با فیلتر LoG ، کمتر است.

تمرین الف-۶

در این تمرین، هدف شناسایی لکه‌های موجود در شکل ۲۲ به روش‌های مختلف و بررسی اثر تغییر پارامترهای شناسایی در نتیجه نهایی لکه‌یابی می‌باشد. مراحل انجام عملیات به تفصیل در ادامه تشریح خواهد شد.



شکل ۲۲- تصویر اولیه

بخش اول: اولین روش لکه‌یابی استفاده شده در این تمرین، استفاده از تابع آماده SimpleBlobDetector_create() می‌باشد. نحوه عملکرد این تابع بدین صورت است که ابتدا با اعمال یک threshold به تصویر اولیه و binary کردن تصویر، پیکسل‌های مجاور با intensity مشابه را که در ابعاد معینی باشند به عنوان لکه تشخیص داده و مرکز آن‌ها را علامت‌گذاری می‌کند.

پارامترهایی که در این تابع قابل تنظیم هستند عبارتند از:

- حدود بالا و پایین threshold.
- حدود بالا و پایین ابعاد لکه بر اساس تعداد پیکسل.
- Circularity یا میزان دایره‌ای بودن لکه. این کمیت با رابطه زیر اندازه‌گیری می‌شود.

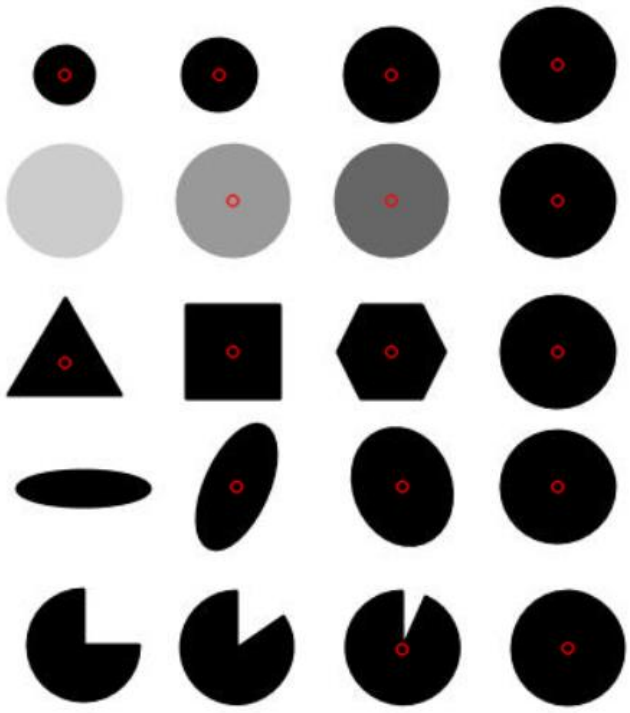
$$\text{circularity} = \frac{4\pi \times \text{Area}}{(\text{Perimeter})^2}$$

این رابطه بیانگر آن است که circularity دایره برابر ۱ می‌باشد و با فرض آن‌که دایره یک بی‌نهایت ضلعی است، هرچه تعداد اضلاع چند ضلعی محدب کاهش یابد، circularity نیز کمتر می‌شود. به عنوان مثال circularity یک مربع برابر ۰/۷۸۵ می‌باشد. Circularity همواره بین ۰ و ۱ می‌باشد که حدود بالا و پایین آن قابل تنظیم است.

- تحدب یا convexity. این کمیت برابر نسبت مساحت لکه به مساحت کوچکترین منحنی بسته محدبی است که لکه را تماماً پوشش دهد. بنابراین تحدب لکه‌های محدب برابر ۱ و تحدب لکه‌های مقعر بین ۰ و ۱ است. حدود بالا و پایین تحدب لکه‌ها نیز توسط کدنویس قابل تنظیم است.
- Inertia ratio یا نسبت اینرسی. این کمیت بیانگر میزان کشیدگی یک لکه است. به عنوان مثال نسبت اینرسی دایره برابر ۱ و نسبت اینرسی خط برابر ۰ است و نسبت اینرسی بیضی‌ها بین این دو مقدار می‌باشد. حدود بالا و پایین نسبت اینرسی نیز قابل تنظیم است.

شکل ۲۳ دارای ۵ ردیف لکه است که در هر یک از این ردیف‌ها، یکی از کمیت‌های فوق درحال تغییر است. همانطور که ملاحظه می‌شود، در ردیف اول ابعاد، در ردیف دوم threshold، در ردیف سوم circularity، در ردیف چهارم نسبت اینرسی و در ردیف پنجم تحدب لکه در حال تغییر است.

بدون تنظیم پارامترهای پنج‌گانه فوق و با همان پارامترهای پیش فرض تابع SimpleBlobDetector_create()، نتیجه نهایی لکه‌یابی به صورت شکل ۲۳ می‌باشد (مراحل نوشتن کد و نشان‌گذاری بر روی لکه‌ها در ادمه ذکر خواهد شد).



شکل ۲۳ - تصویر نهایی با پارامترهای پیش فرض

همانطور که در شکل ۲۳ ملاحظه می‌گردد، برخی از لکه‌ها شناسایی نشده‌اند و لذا می‌بایست حدود بالا و پایین threshold، نسبت اینرسی و تحدب تنظیم گردد.

مراحل کدنویسی عبارتند از:

۱) Import کتابخانه‌ها و خواندن تصویر در فرمت gray.

```
import cv2
import numpy as np
pic=cv2.imread('3.jpg',cv2.IMREAD_GRAYSCALE)
```

۲) راه‌اندازی و فراخوانی تابع SimpleBlobDetector_Params() به منظور تغییر پارامترها.

```
# Setup SimpleBlobDetector parameters.
params = cv2.SimpleBlobDetector_Params()
```

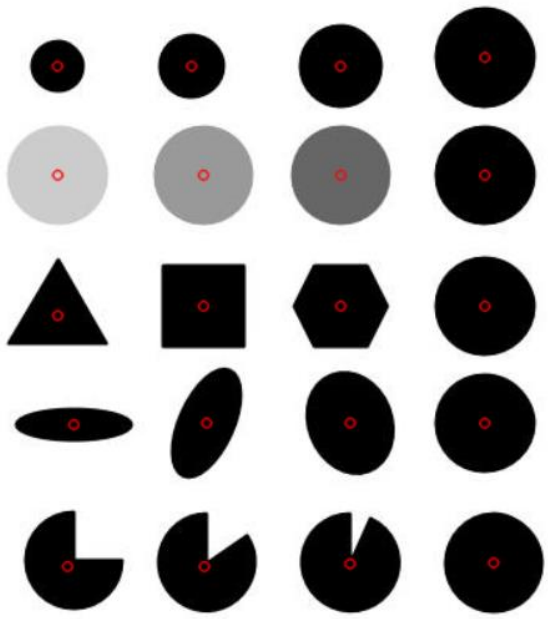
۳) تنظیم حدود threshold، نسبت اینرسی و تحدب. توجه شود برای تنظیم نسبت اینرسی و تحدب، یک flag می‌بایست برابر ۱ قرار داده شود. هم‌چنین این حدود به کمک آزمون و خطا بدست آمده‌اند.

```
# Change thresholds
params.minThreshold = 220;
params.maxThreshold = 255;
# Filter by Inertia
params.filterByInertia = 1
params.minInertiaRatio = 0.01
# Filter by Convexity
params.filterByConvexity = 1
params.minConvexity = 0.7
```

۴) فراخوانی تابع SimpleBlobDetector_create() به منظور شناسایی لکه‌ها با پارامترهای تنظیم شده و سپس علامت‌گذاری مرکز لکه‌ها و رسم نتیجه نهایی. همانطور که ملاظحه می‌شود ورودی این تابع، تصویر اولیه در فرمت gray و ورودی تابع cv2.drawKeypoints() که وظیفه نشانه‌گذاری لکه‌ها را بر عهده دارد، تصویر اولیه، مرکز لکه‌ها که از تابع SimpleBlobDetector_create() بدست آمده و نیز رنگ دایره نشانگر می‌باشد.

```
det=cv2.SimpleBlobDetector_create(params)
points=det.detect(pic)
draw_points=cv2.drawKeypoints(pic,points,np.array([]),(0,0,255))
cv2.imshow("POINTS",draw_points)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

نتیجه نهایی به صورت شکل ۲۴ خواهد بود. همانطور که ملاحظه می‌شود، تمامی لکه‌ها به خوبی شناسایی شده‌اند.



شکل ۲۴- تصویر نهایی با تنظیم پارامترها

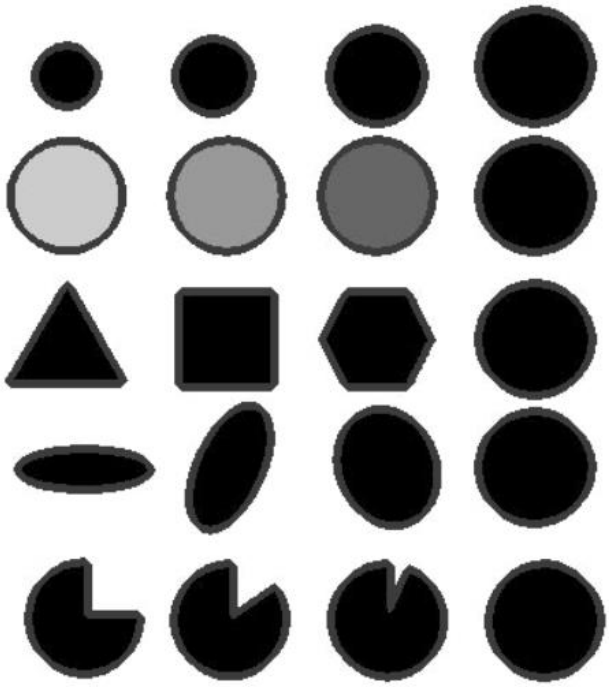
بخش دوم: روش دوم پیدا کردن این لکه‌ها، استفاده از تابع findContours() می‌باشد که مراحل انجام این عملیات به این شرح می‌باشد.

۱) خواندن تصویر و اعمال threshold به آن به منظور تولید تصویر binary. اعداد مربوط به حدود بالا و پایین threshold به روش آزمون و خطا بدست آمده‌اند. علت تبدیل تصویر به binary این است که تابع findContours() پیکسل‌هایی را که در آن‌ها intensity از ۱ به ۰ یا بالعکس تغییر نماید، شناسایی و ذخیره می‌کند.

```
pic=cv2.imread('3.jpg',cv2.IMREAD_GRAYSCALE)
_,threshold=cv2.threshold(pic,220,255,cv2.THRESH_BINARY)
```

۲) شناسایی و ذخیره‌کردن موقعیت پیکسل‌های کانتور و علامت‌گذاری آن‌ها در تصویر اولیه و نهایتاً رسم تصویر نهایی. توجه شود در صورت استفاده از عبارت cv2.CHAIN_APPROX_SIMPLE تابع پس از شناسایی یک خط مرزی، مختصات تمام پیکسل‌های مرزی را ذخیره نکرده و صرفاً مختصات نقاط ابتدایی و انتهایی مرز را ذخیره می‌نماید که این امر باعث صرفه‌جویی حافظه می‌گردد. هم‌چنین در تابع drawContours() که به منظور رسم کانتورها استفاده می‌گردد، ضخامت کانتورهای رسم‌شده و رنگ آن در فرمت gray قابل تنظیم است. نتیجه نهایی این روش لکه‌یابی به صورت شکل ۲۵ می‌باشد. همانطور که ملاحظه می‌شود، دور لکه‌ها کانتورهای خاکستری رنگ رسم شده است.

```
contours,_ =cv2.findContours(threshold, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
for cnt in contours:
    cv2.drawContours(pic,[cnt],0,(60),3)
cv2.imshow("POINTS",pic)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



شکل ۲۵- تصویر نهایی

تمرین ب-۳

در این تمرین، هدف آن است که ابتدا با کد نوشته شده در تمرین ب-۱ تکلیف سری اول، ویدویی توسط دوربین کامپیوتر ذخیره شده و سپس فیلترهای لبه‌یاب یک بار به همراه فیلتر گوسی و بار دیگر بدون فیلتر گوسی بر روی این فایل ضبط شده اعمال گردند و درنهایت نتایج این دو حالت با یکدیگر مقایسه شوند. گام‌های انجام این عملیات ب شرح زیر می‌باشد.

۱) ذخیره یک فایل ویدیویی با نام 'hw.avi' بوسیله کد نوشته شده در تمرین ب-۱ تکلیف سری اول.

۲) Import کتابخانه‌ها و راه‌اندازی فایل ویدویی.

```
import numpy as np
import cv2


file=cv2.VideoCapture('hw.avi')
```

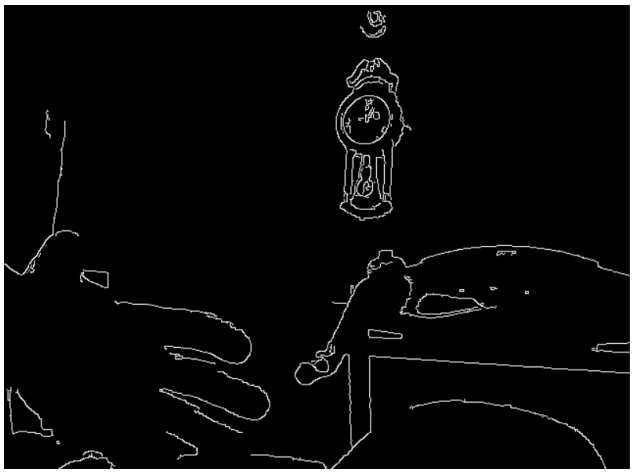
۳) دریافت فریم‌های فایل ویدیویی در یک حلقه while و سپس تبدیل فرمت آن به gray و اعمال یک فیلتر گوسی ۵×۵ به تصویر gray. پس از انجام این عملیات، فیلتر canny به فریم اعمال می‌گردد و سپس فیلترهای Sobel و فیلترهای prewitt با کرنل‌های تعریف شده توسط کدنویس، یک بار به تصویر gray عادی و بار دیگر به تصویر gray فیلتر شده با فیلتر گوسی مذکور اعمال شده و نتایج آن نمایش داده می‌شود. فیلتر canny در داخل خود دارای یک فیلتر گوسی می‌باشد و لذا مقایسه این دوحالت برای این فیلتر بی معنا است.

```
while True:
    #start getting frames
    _,frame = file.read()
    frame_gray=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    frame_blur=cv2.GaussianBlur(frame_gray,(5,5),0)
    #canny
    edge_canny = cv2.Canny(frame_gray,50,200)
    #sobel
    #blurred
    edge_x_blur=cv2.Sobel(frame_blur,cv2.CV_64F,1,0,ksize=5)
    edge_y_blur=cv2.Sobel(frame_blur,cv2.CV_64F,0,1,ksize=5)
    edge_sobel_blur=cv2.Sobel(frame_blur,cv2.CV_8U,1,1,ksize=5)
    #sobel
    #org
    edge_x=cv2.Sobel(frame_gray,cv2.CV_64F,1,0,ksize=5)
    edge_y=cv2.Sobel(frame_gray,cv2.CV_64F,0,1,ksize=5)
    edge_sobel=cv2.Sobel(frame_gray,cv2.CV_8U,1,1,ksize=5)
```

```
        #prewitt
        #blur
        xkernel=np.array([[1,1,1],[0,0,0],[-1,-1,-1]])
        ykernel=np.array([[-1,0,1],[-1,0,1],[-1,0,1]])
        xedge_prewitt_blur=cv2.filter2D(frame_blur,-1,xkernel)
        yedge_prewitt_blur=cv2.filter2D(frame_blur,-1,ykernel)
        #prewitt
        #org
        xedge_prewitt=cv2.filter2D(frame_gray,-1,xkernel)
        yedge_prewitt=cv2.filter2D(frame_blur,-1,ykernel)

        #show frames to produce video
        cv2.imshow("CANNY",edge_canny)
        cv2.imshow('Sobel_ORG',edge_sobel)
        cv2.imshow('Sobel_BLUR',edge_sobel_blur)
        cv2.imshow('PrewittX_BLUR',xedge_prewitt_blur)
        cv2.imshow('PrewittY_BLUR',yedge_prewitt_blur)
        cv2.imshow('PrewittX_ORG',xedge_prewitt)
        cv2.imshow('PrewittY_ORG',yedge_prewitt)
        key=cv2.waitKey(20)
        if key == ord('e'):
            break
        else:
            pass
    cam.release()
    cv2.destroyAllWindows()
```

نتایج حاصل بر روی یک فریم در شکل‌های ۲۶ تا ۳۲ نمایش داده شده است. همانطور که ملاحظه می‌شود، تصاویری که پس از اعمال فیلتر گوسی لبه‌یابی شده اند، دارای لبه‌هایی با وضوح پایین‌تر هستند اما در عوض نویز کمتری دارند. همچنین در فیلتر prewitt در جهت x، لبه‌های افقی نمایان شده و در فیلتر prewitt در جهت y، لبه‌های قائم نمایان شده‌اند.



شکل ۲۶- canny

شکل ۲۷- Sobel بدون فیلتر گوسی

شکل۲۸- Sobel با فیلتر گوسی

شکل ۲۹- prewitt-x با فیلتر گوسی

شکل ۳۰- prewitt-y با فیلتر گوسی

شکل ۳۱- prewitt-x بدون فیلتر گوسی

شکل ۳۲- prewitt-y بدون فیلتر گوسی