

به نام خدا

درس دید کامپیوتری

گزارش تمرین شماره یک بخش دوم

علیرضا بانشی

95101185

سوال اول

(3)

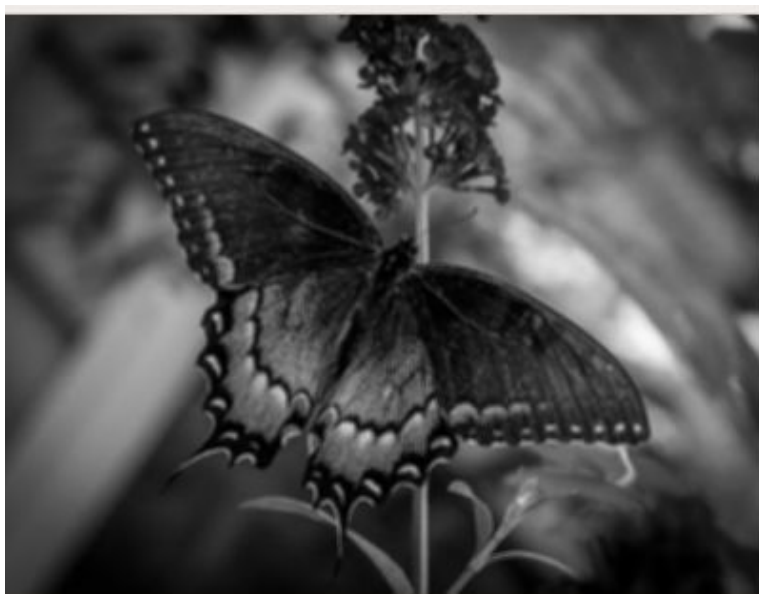
نکته: من منظور سوال را اینطور دریافتم که اول فیلتر پایین گذر زده و روی آن لبه یاب هم اعمال کنیم حالا سعی کنیم فرکانس بالای تصویر اولیه را بدست بیاوریم ... ابتدا تصویر را می خوانیم و به کمک دستور زیر تصویر را `resize` می کنیم:

```
img = cv2.resize(img, (400, 300), interpolation=cv2.INTER_AREA)
```

حالا برای این که تصویر پایین گذر شده را ایجاد کنیم باید کرنل مناسب را تعیین کنیم و سپس تصویر را با آن فیلتر کنیم.

```
kernel = np.ones((3, 3), np.float32) / (3 ** 2)  
low_pass = cv2.filter2D(img, -1, kernel)
```

برای این کار از یک کرنل سه در سه استفاده کرده ایم که همه درایه ها آن $1/9$ است. عملاً یک فیلتر میانگین است که هر درایه را میانگین پیکسل های اطراف قرار می دهد و تصویر پایین گذر می سازد پس از فیلتر کردن تصویر پایین گذر شده به قرار زیر است:



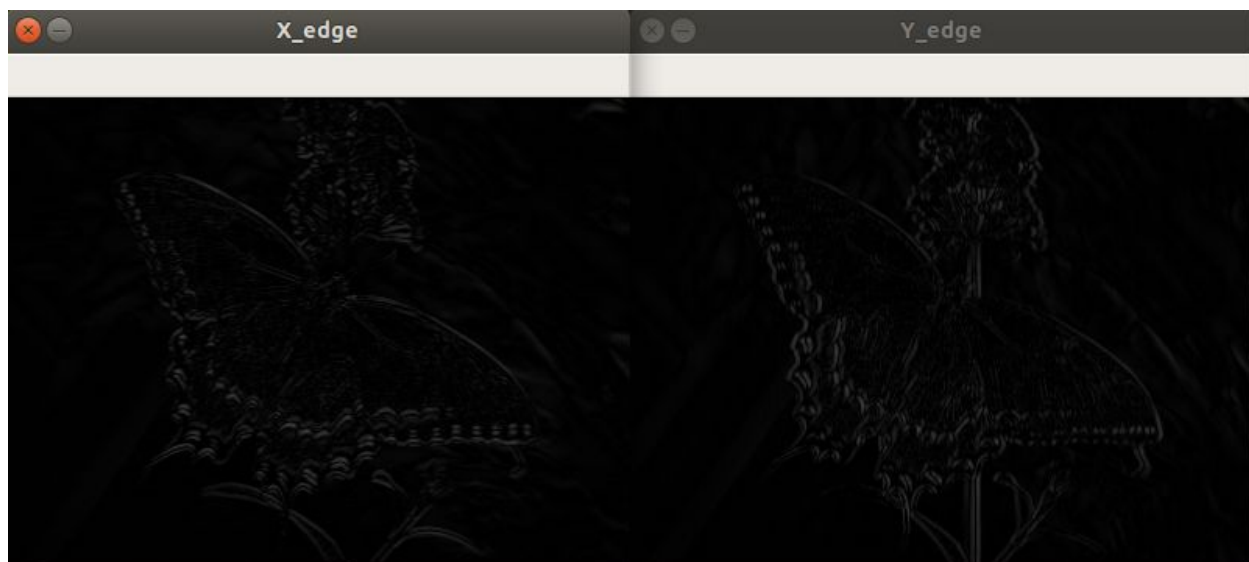
حالا می خواهیم لبه های افقی و عمودی را از روی این تصویر بدیت بیاوریم. برای این کار از فیلتر ساده مشتق گیر که در درس آموختیم استفاده می کنیم.

برای این کار تصویر را پرمایش می کنیم مشتق در جهت y و در جهت x به ترتیب برابر است با:

```
y_edge[i, j] = img[i, j + 1] - img[i, j]
```

```
x_edge[i, j] = img[i + 1, j] - img[i, j]
```

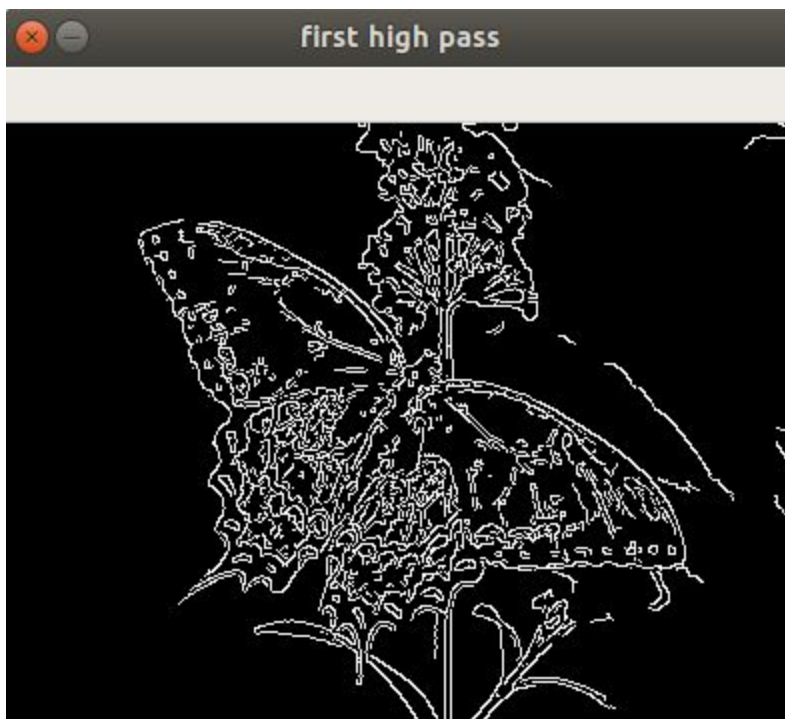
که حاصل به قرار زیر است:



دلیل خوب پیدا نبودن لبه ها این است که ابتدا تصویر را از فیلتر پایین گذر رد کرده ایم و بدیتهای لبه ها که فرکانس بالاهای عکس هستند مقداری از اطلاعات خود را از دست می دهند و تصویر به شکل بالا در میآید. دو عکس با نام های X_edge و Y_edge ضمیمه شده اند.

حال سعی میکنیم فرکانس بالاهای عکس اولیه را به کمک این دو تصویر بدست بیاوریم.

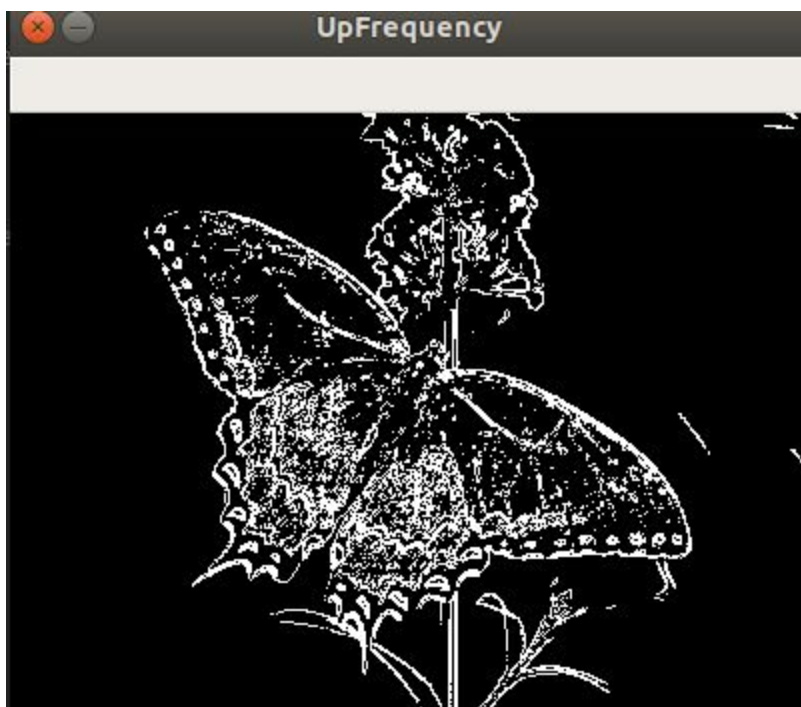
ابتدا به فرکانس بالاهاى تصویر اولیه دقت کنیم که به کمک یک فیلتر Canny که بالاگذر است بدست آمده است و هدف ما نزدیک شدن به این تصویر است.



برای این که به این تصویر برسیم از یک فیلتر بالاگذر که با ترکیب این دو عکس ساخته می شود استفاده میکنیم
فیلتر ما به قرار زیر است:

```
# high frequency Filter.  
# edge = sqrt(X_edge^2 + Y_edge^2)
```

که عملاً فرکانس بالاهاى دو تصویر را با هم ترکیب می کند و کمی نیز به خاطر توان دو تصویر ها را شارپ می کند.
پس از این کار و اعمال یک ترشهولدینگ ساده به تصویر زیر می رسم:



با دقت مشخص می شود که تا حد خوبی تصویر را بازسازی کرده ایم. این تصویر نیز با نام UpFreq_output.jpg ذخیره و پیوست شده است.

(4

ابتدا یک معرفی اجمالی از متدها انجام می دهیم:

:sobel

Sobel یکی از معروف ترین روش های تشخیص لبه می باشد.

این روش بر پایه کانوالو کردن تصویر با دو فیلتر کوچک مجزا و اینتیجر می باشد که باعث می شود محاسبات آن بسیار آسان باشد.

در واقع خوبی این روش آسان بودن محاسبه آن است.

فیلتر ها به قرار زیر است:

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

یک بار تصویر را در جهت افقی فیلتر و یک بار در جهت عمودی فیلتر و نتیجه را باهم جمع می کنیم.

دو خوبی دیگر نیز در این روش وجود دارد.

اولی هموار سازی که باعث کاهش نویز می شود و دیگر اینکه لبه ها را با خوبی و تمایز زیاد مشخص می کند.

Laplasian

این عملگر تنها از یک کرنل برای فیلتر کردن تصویر استفاده می کند و از این حیث با sobel و prewitt متفاوت است.

دو کرنل رایج عبارتند از:

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

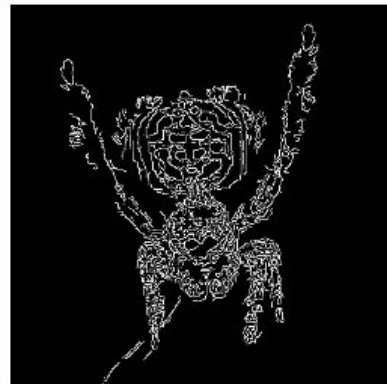
این عملگر گرادین مرتبه دوم را در یک جهت حساب می کند و به این دلیل به شدت به نویز حساس است به همین خاطر ابتدا معمولا با یک فیلتر گوسی تصویر را فیلتر می کنیم تا نویز کم شود.

Canny

این اپراتور معروف ترین راهکار شناسایی لبه می باشد و مکانیزم نسبتا متفاوتی با روش های قبل دارد که به اختصار به شرح زیر است:

1. Smooth the image with a Gaussian filter to reduce noise.
2. Compute gradient of using any of the gradient operators Sobel or Prewitt.
3. Extract edge points: Non-maximum suppression.
4. Linking and thresholding: Hysteresis

حالا به عنوان مقایسه یک عکس را در همه حالات قرار می دهیم:



Top left: (Original). Top middle: (Sobel). Top right: (Sobel RGB). Bottom right: (Prewitt). Bottom middle: (Laplacian).
Bottom left: (Canny)

فیلتر های sobbel و canny و لاپلاسیان به تصاویر اعمال شده است و نتایج در زیر آمده است.

برای این کار یعنی بدست آوردن تصویر فیلتر شده با این فیلتر ها از توابع آماده زیر استفاده کرده ایم.

همینطور بعد از فیلتر کردن از عملگر قدر مطلق استفاده می کنیم تا بتوانیم لبه ها را به خوبی تشخیص دهیم

```
canny_edge = cv2.Canny(img,200,400)
laplacian_edge = cv2.Laplacian(img,cv2.CV_64FC1)
sobelx_edge = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=1)
sobely_edge = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=1)
laplacian = np.uint8(np.absolute(laplacian_edge))
sobelx = np.uint8(np.absolute(sobelx_edge))
sobely = np.uint8(np.absolute(sobely_edge))
```

نتایج این فیلتر ها برای تصاویر یک و دو به شرح زیر است:

Original



Laplacian



canny



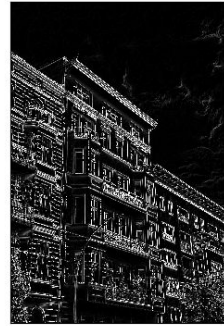
Sobel X



Sobel Y



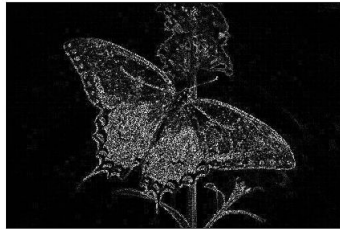
Sobel



Original



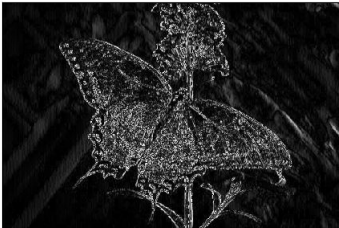
Laplacian



canny



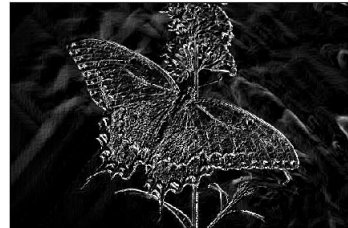
Sobel X



Sobel Y



Sobel



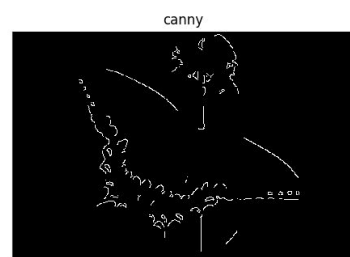
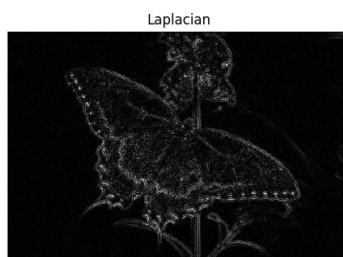
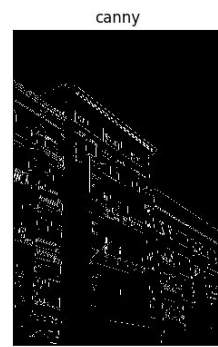
برای بهتر کردن نتیجه و حذف نویز راه حل پیشنهادی آن است که ابتدا یک فیلتر گاوسین روی تصویر اعمال کنیم و سپس فیلتر کنیم.

البته باید این نکته را در نظر داشت که ما در بالا صرفاً لبه را به کمک لاپلاسین تنها بدست آورده ایم و اگر بخواهیم فیلتر ما LoG باشد باید حتماً ابتدا یک فیلتر گاوسی اعمال کنیم.

برای این بخش ابتدا تصویر را به کمک یک فیلتر گاوسی 5 در 5 فیلتر یا به اصطلاح blur می کنیم .

حالا بار دیگر عملگر های یافت لبه را اعمال می کنیم.

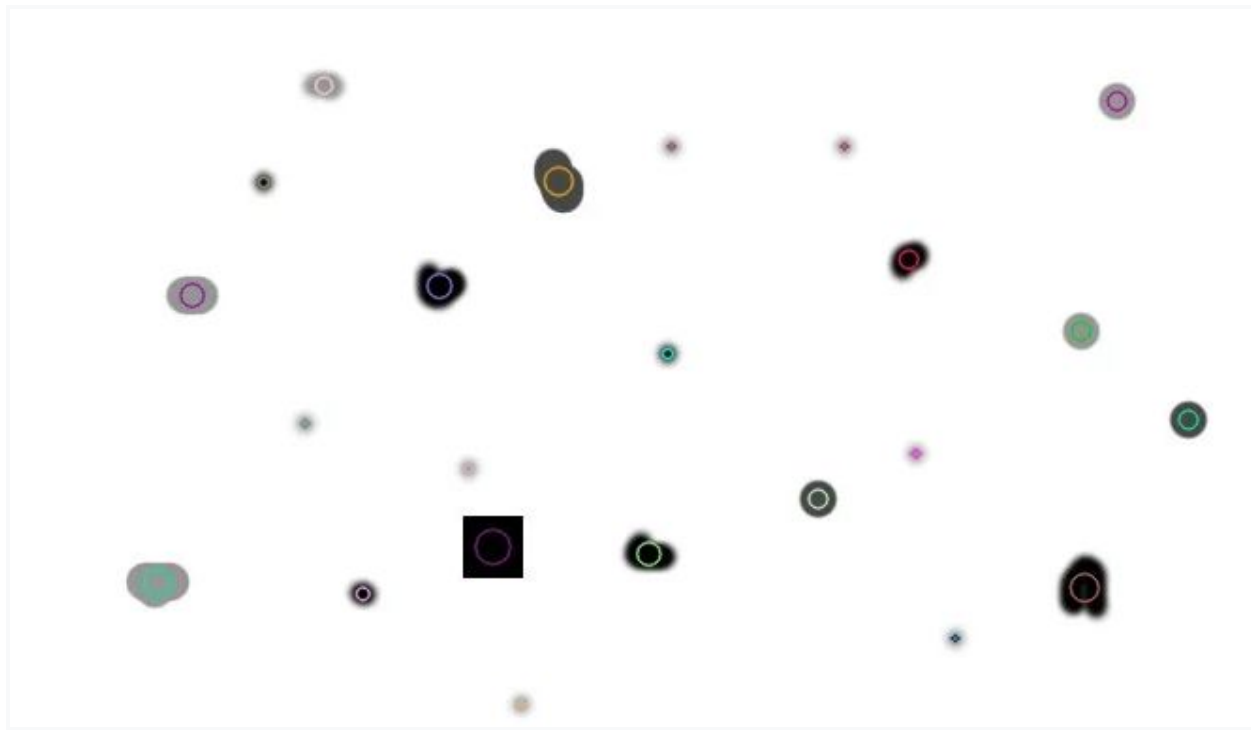
نتایج به قرار زیر است:



این تصاویر با نام های figure_1 تا 4 ذخیره و پیوست شده است.

(6)

Blob یا اصطلاحاً لکه یا حباب گروهی از پیکسل های متصل در یک تصویر است که دارای برخی از ویژگی های مشترک است (به عنوان مثال مقدار مقیاس خاکستری). در تصویر زیر ، مناطق متصل به تاریک حباب هستند و هدف از شناسایی لکه ها شناسایی و علامت گذاری این مناطق است.



اما یک simple blob detector چگونه کار می کند؟

همانطور که از نام آن پیداست ، **SimpleBlobDetector** براساس یک الگوریتم نسبتاً ساده است که در زیر شرح داده شده است. این الگوریتم توسط پارامترها کنترل می شود و مراحل زیر را دارد.

Thresholding: با آستانه گرفتن تصویر منبع با آستانه شروع از **min** **Threshold** ، تصاویر مبدأ را به چندین تصویر باینری تبدیل کنید. این آستانه ها توسط

آستانه step تا حداکثر آستانه افزایش می یابد. به این صورت که در هر مرحله یک step به min اضافه می شود و این روند تا رسیدن به ترشهولد ماکزیمم ادامه دارد.

Grouping: در هر تصویر دودویی ، پیکسل های سفید متصل به هم گروه بندی می شوند. به این ها حباب باینری می گوئیم.

Merging: مراکز حباب های باینری در تصاویر باینری محاسبه می شوند و حباب هایی که نزدیکتر از minDistBetweenBlobs هستند با هم ترکیب می شوند.

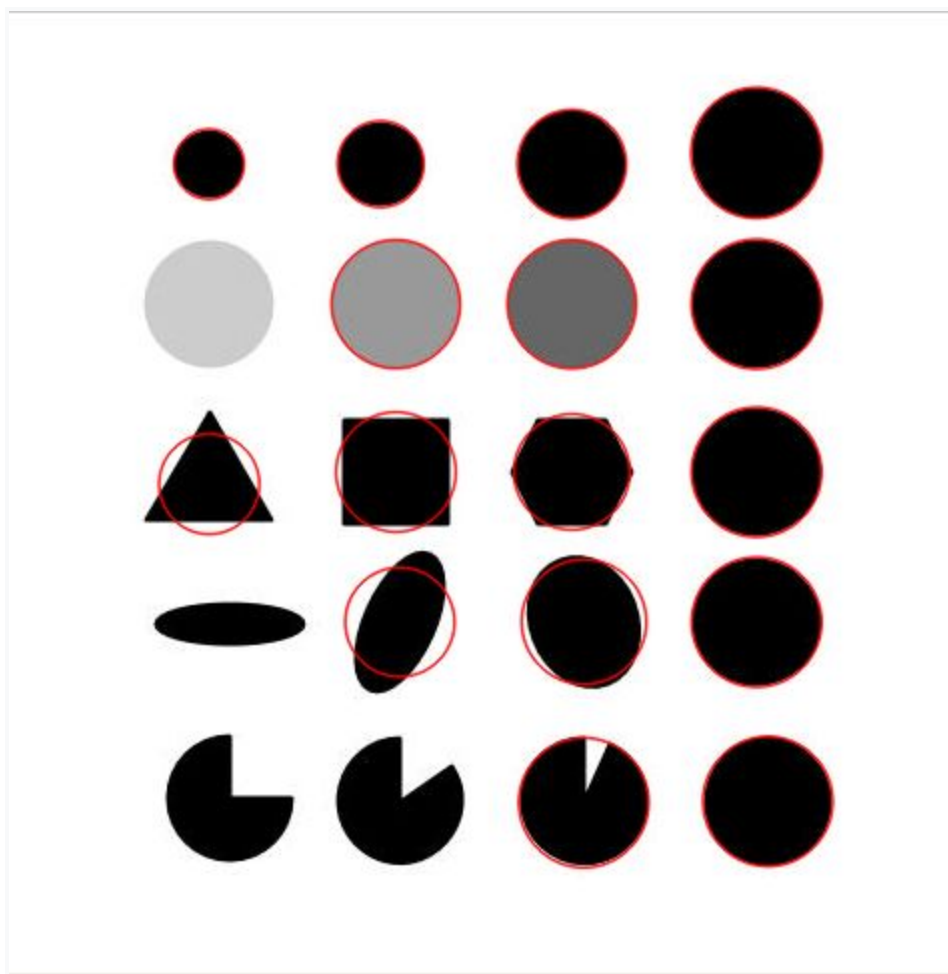
Center & Radius Calculation: مراکز و شعاع حباب های ادغام شده جدید محاسبه و بازگردانده می شوند.

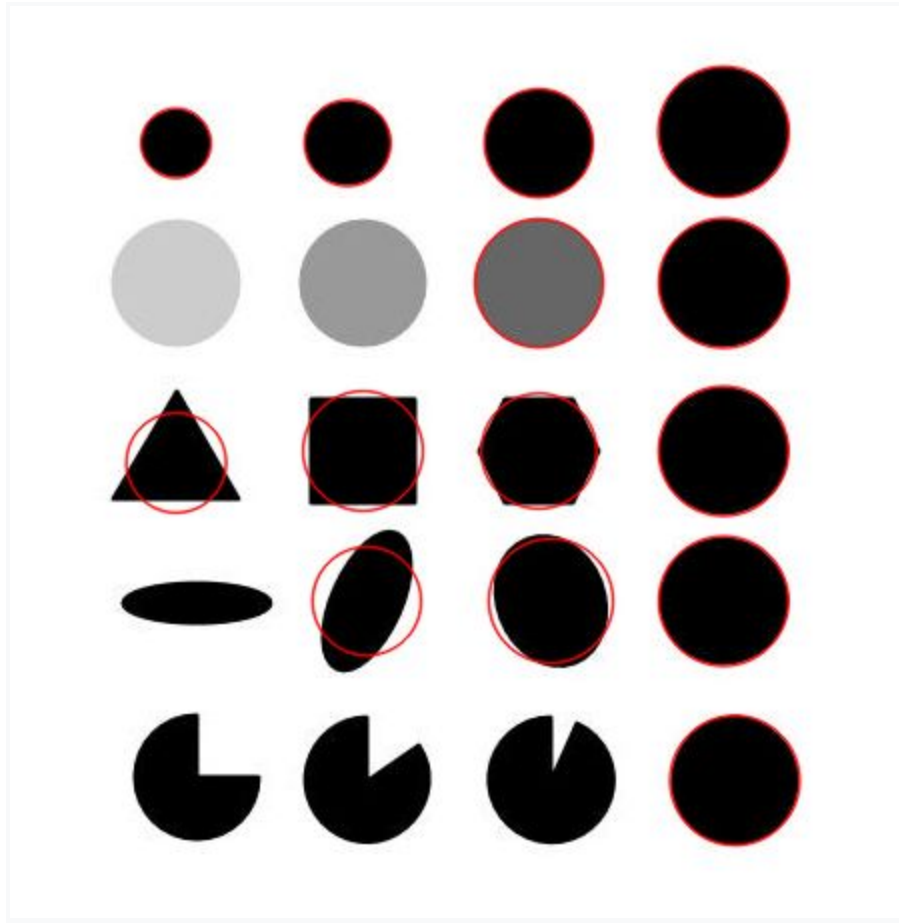
می توان از یک سری پارامترها برای فیلتر کردن حباب ها استفاده کرد تا حباب های مورد نظر را تشخیص دهیم.
این پارامتر ها عبارتند از:

```
# Change thresholds
params.minThreshold = 10
params.maxThreshold = 200
# Filter by Area.
params.filterByArea = True
params.minArea = 1500
# Filter by Circularity
params.filterByCircularity = True
params.minCircularity = 0.1
# Filter by Convexity
params.filterByConvexity = True
params.minConvexity = 0.87
# Filter by Inertia
params.filterByInertia = True
params.minInertiaRatio = 0.01
```

اولین پارامتر نشان داده شده در شکل بالا که حد مینیمم و ماکسیمم ترشهولد برای تشخیص حباب را نشان می دهد.

برای مثال یکبار عکس خواسته شده را با اعداد 0 و 150 تشخیص حباب می دهیم و یکبار با 150 و 255 و به ترتیب خروجی آن ها را می بینیم.
بقیه پارامترها را روی دیفالت تنظیم می کنیم



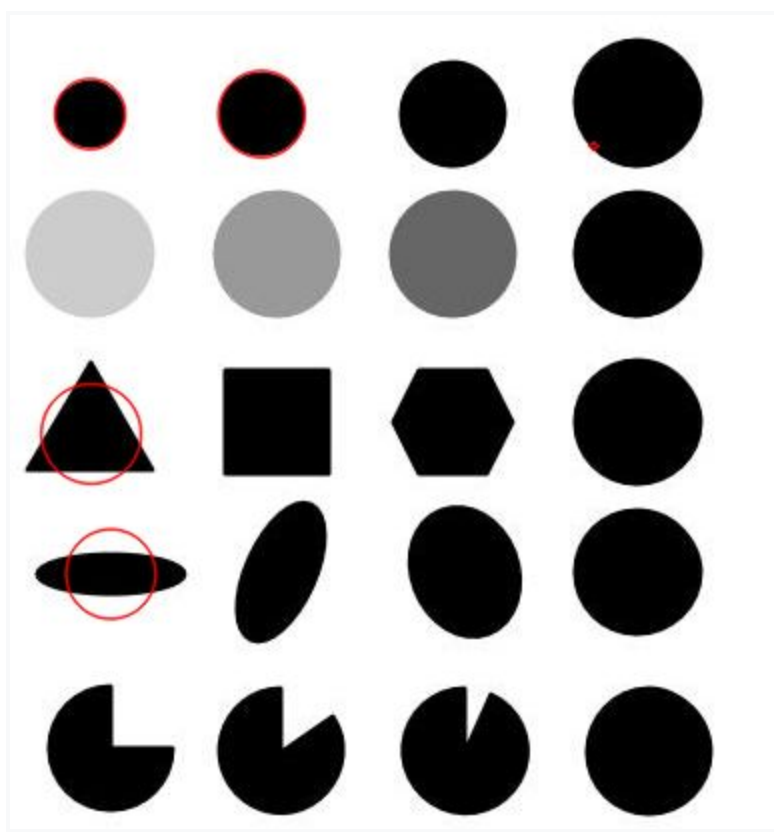
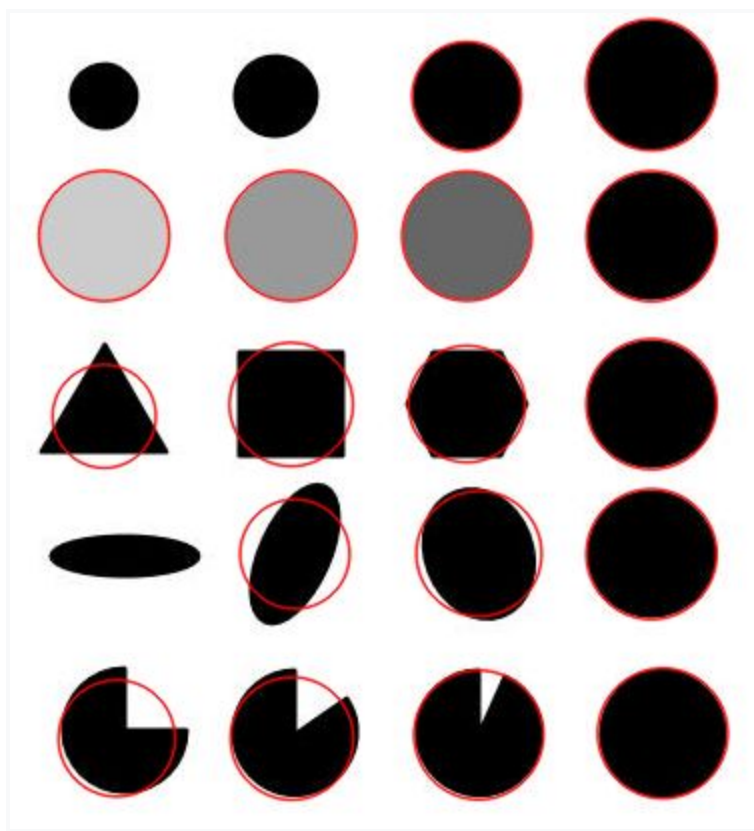


از دومین پارامتر برای فیلتر کردن به کمک مساحت استفاده می کنیم.

برای آن نیز می توانیم مینیمم و ماکسیمم را تعیین کنیم و فقط حباب هایی که مساحتشان بین این بازه است را انتخاب کنیم.

از این به بعد هنگام مشاهده تاثیر پارامتر بقیه پارامتر ها را بین ماکس و مین می گذاریم به عبارت دیگر آن ها را بی اثر می کنیم تا تنها بتوانیم تاثیر پارامتر مورد نظر را ببینیم

برای مثال یک بار بین 2000 و 5000 و یکبار بین 0 و 2000 تنظیم می کنیم نتایج به قرار زیر است



پارامتر دیگر circularity است.

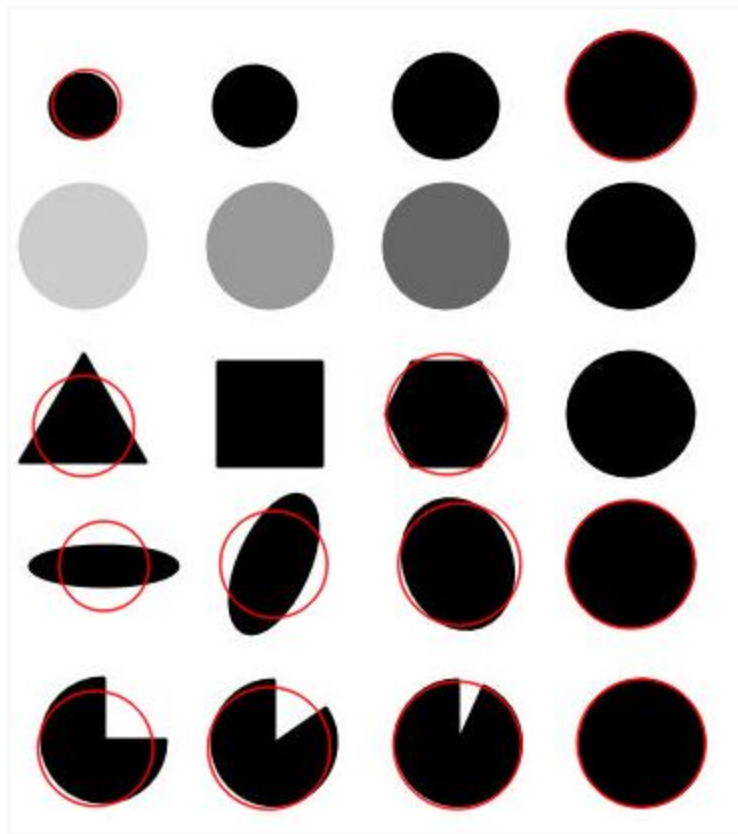
این پارامتر میزان نزدیکی بودن شکل به دایره را به کمک پارامتری به نام circularity تعیین می کند که به شکل زیر بدست می آید

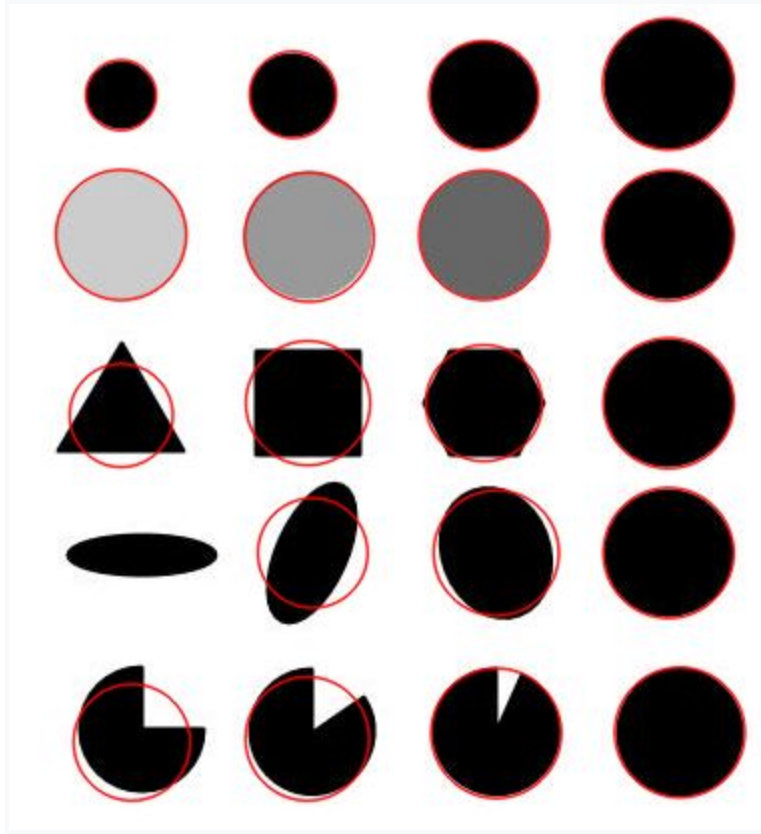
$$\frac{4\pi Area}{(perimeter)^2}$$

برای مثال این مقدار برای دایره برابر با 1 برای مربع 0.785 و ... است.

دوباره برای این مقدار نیز می توانیم یک مینیمم و ماکسیمم تعریف کرده و خروجی بین آن دو را فیلتر کنیم .

برای مثال یک بار بین یک دهم و شش دهم و یکبار بین شش دهم و 1 تنظیم می کنیم نتایج به قرار زیر است.

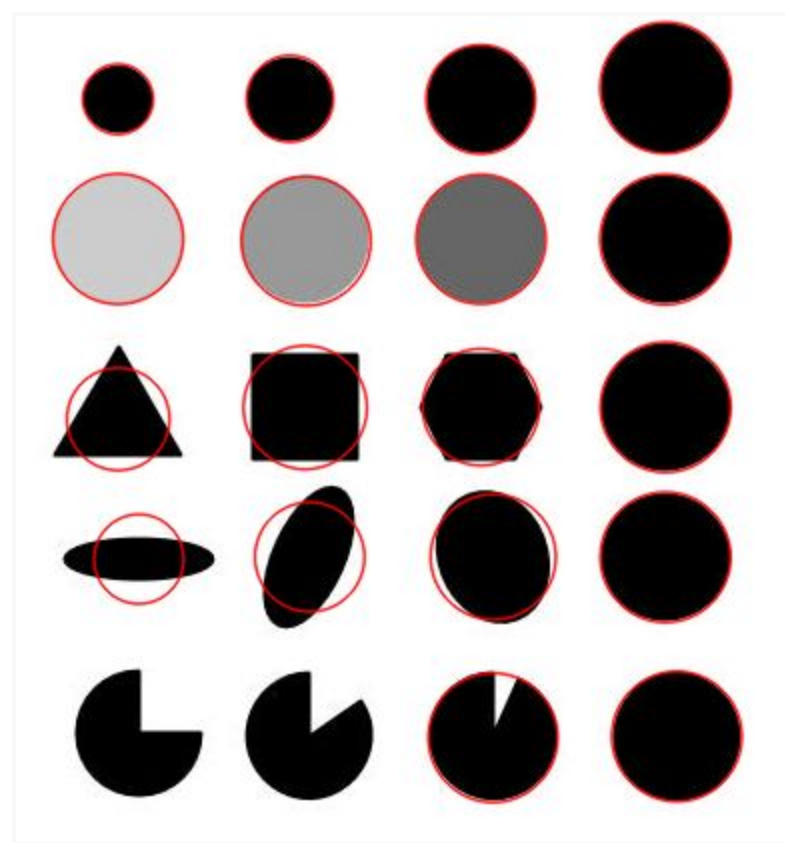
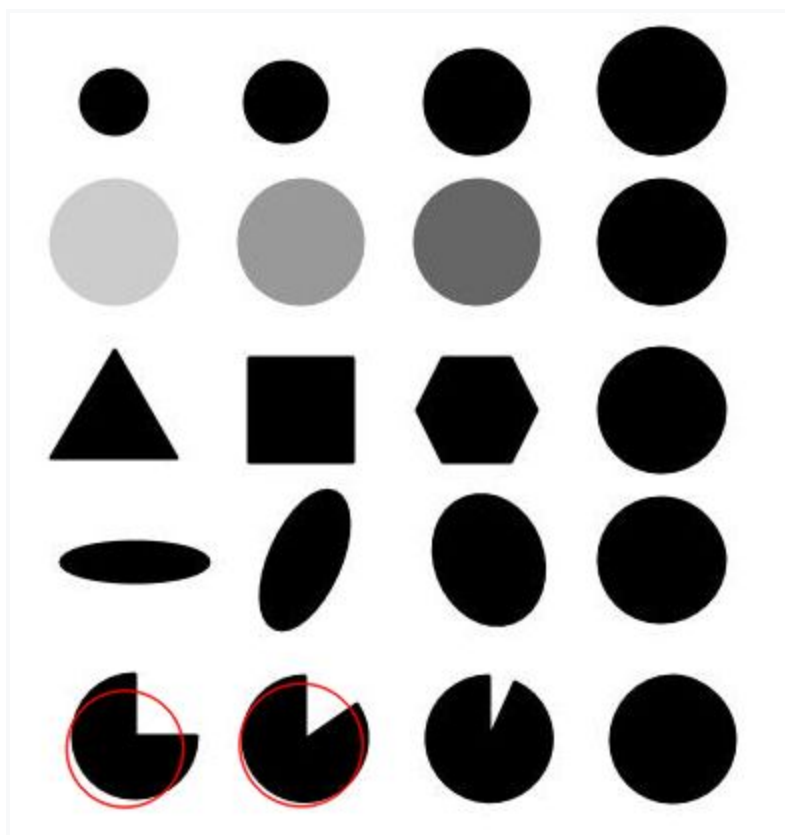




پارامتر دیگر مورد استفاده **convexity** است که میزان محدب بودن شکل را تعیین می کند که به شکل زیر تعریف می شود.



$$\text{Convexity} = (\text{Area of the Blob} / \text{Area of it's convex hull})$$

دوباره از دو پارامتر مینیمم و ماکسیمم برای تعیین حد آن استفاده می شود.
برای مثال یک بار بین یک دهم و نه دهم و یکبار بین نه دهم و 1 تنظیم می کنیم نتایج به قرار زیر است.

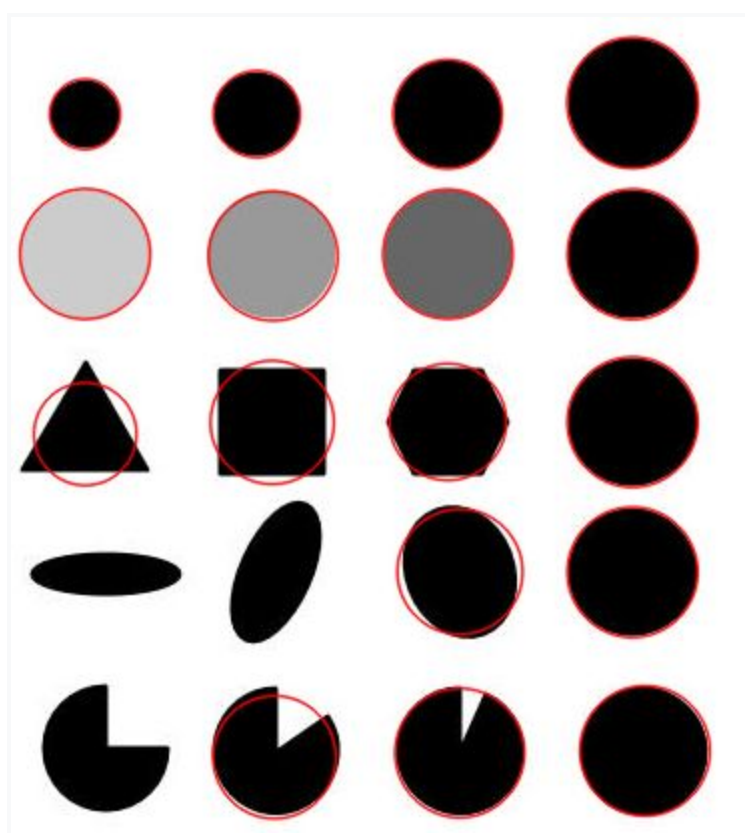
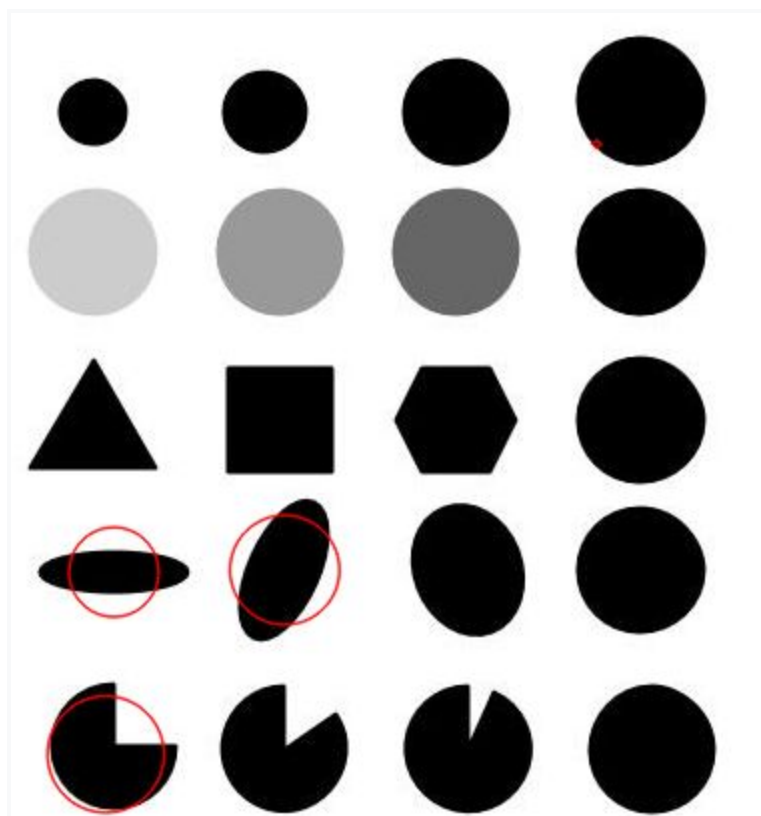


اما آخرین پارامتر Interia می باشد

این پارامتر معیاری از دراز بودن تصویر است و مقادیری بین 0 و 1 دارد
به شکل زیر نیز دقت شود.

Low Inertia Ratio	High Inertia Ratio
	

این بار نیز ما از دو مقدار ماکسیمم و مینیمم برای تعیین آن استفاده می کنیم.
برای مثال یک بار بین 0 و شش دهم و یکبار بین شش دهم و 1 تنظیم می کنیم نتایج به
قرار زیر است.



سوال دوم

(3)

به کمک قسمت اول این سوال یعنی قسمت ب 1 که در تمرین قبل تحویل دادیم ویدیو cam.mp4 را ذخیره می کنیم و در این تمرین آن را می خوانیم به صورت فریم به فریم و روی آن فیلترهای لبه مختلف را اعمال می کنیم و نتایج را ذخیره می کنیم برای این کار از فیلتر های زیر استفاده می کنیم

```
#Gaussian Filter
frame = cv2.GaussianBlur(frame, (5, 5), 0)
#canny edge detector
c_img = cv2.Canny(frame, 50, 100)
#sobel edge detector
sobelx_edge = cv2.Sobel(frame, cv2.CV_8U, 1, 0, ksize=5)
sobely_edge = cv2.Sobel(frame, cv2.CV_8U, 0, 1, ksize=5)
sobel_img = sobelx_edge+sobely_edge
#prewitt edge detector
kernelx = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]])
kernely = np.array([[1, 0, 1], [-1, 0, 1], [-1, 0, 1]])
img_prewittx = cv2.filter2D(frame, -1, kernelx)
img_prewitty = cv2.filter2D(frame, -1, kernely)
prewitt_img = img_prewittx + img_prewitty
```

دقت شود که همانطور که می دانیم فیلتر های sobel و prewitt جهت دار هستند یعنی ابتدا آنها را در راستای x سپس در راستای y فیلتر می کنیم و حاصل را با هم جمع می کنیم.

حال به مقایسه این فیلتر ها می پردازیم.

البته ابتدا یک معرفی کوچک از prewitt می کنیم

فیلتر های canny و sobel در قسمت 4 سوال یک توضیح داده شده اند.

Prewitt

این اپراتور نیز مشابه sobel می باشد و از دو فیلتر استفاده می کند و لبه ها را در راستای افقی و عمودی بدست می آورد.

این فیلتر ها عبارتند از:

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

خوب اما اگر در ویدیو ها دقت کنیم در ویدیو ذخیره شده با canny لبه ها مشخص تر است و به واسطه ترشهولد انتخابی کوچکترین لبه ها نیز شناسایی شده اند. اما وضع نویز در ویدیوی ذخیره شده با prewitt به وضوح بهتر است. اما اگر قبل از استفاده از عملگر های بالا ابتدا یک فیلتر گوسی به همه فریم ها اعمال کنیم باعث می شود نویز به شدت کاهش بیابد و تصویر smoth تر می شود و کیفیت تشخیص لبه ها نیز بیشتر می شود.

همه خروجی ها ذخیره شده (به ازای همه حالات یک بار با فیلترینگ گوسی و یک بار بدون آن) و همه نتایج ذخیره شده و در google drive با لینک زیر به اشتراک گذاشته شده است.

<https://drive.google.com/drive/folders/1zjmYBZH9yEOw3EmelLNQJjsLcd1P545o?usp=sharing>