

-1

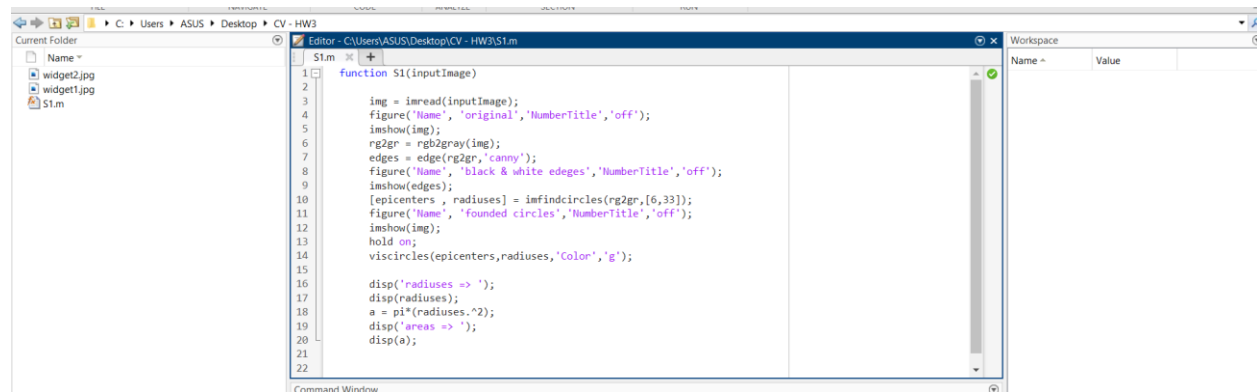
برای حل این سوال همانطور که در کلاس گفته شد ابتدا باید تصویر باینری را لبه گیری کنیم و پس از آن روش تبدیل هاف را برای یافتن دایره روی تصویر لبه گیری شده پیاده سازی کنیم. روش هاف دایره های موجود در عکس را با توجه به **range** شعاع های وارد شده پیدا می کند.

به همین منظور در کد ابتدا نام تصویر دلخواه را به تابع **S1** می دهیم (دو تصویر برای این سوال وجود دارد)، پس از نمایش آن در **figure(original)**، تصویر را باینری و لبه گیری می کنیم. در اینجا لبه گیری با متد **canny** انجام شد. سپس بعد از نمایش تصویر باینری لبه گیری شده در **figure(black & white edges)**، با دستور **imfindcircles("image name" , "radius range")**، تصویر لبه گیری شده را به عنوان آرگومان اول و محدوده شعاع دایره ها را به عنوان آرگومان دوم دادیم. با توجه به محدوده شعاع دایره های دو تصویر **widget1** و **widget2**، محدوده شعاع ها را از 6 تا 33 قرار دادیم.

در نهایت هم دایره های پیدا شده را در **figure(founded circles)**، با رنگ سبز نمایش دادیم.

در ادامه ی سوال برای محاسبه ی مساحت دایره ها، تمام شعاع های دایره های پیدا شده در قسمت قبل را با فرمول مساحت که عدد **pi** در شعاع به توان دو هست در متغیر **a** قرار می دهیم و نمایش می دهیم.

کد :

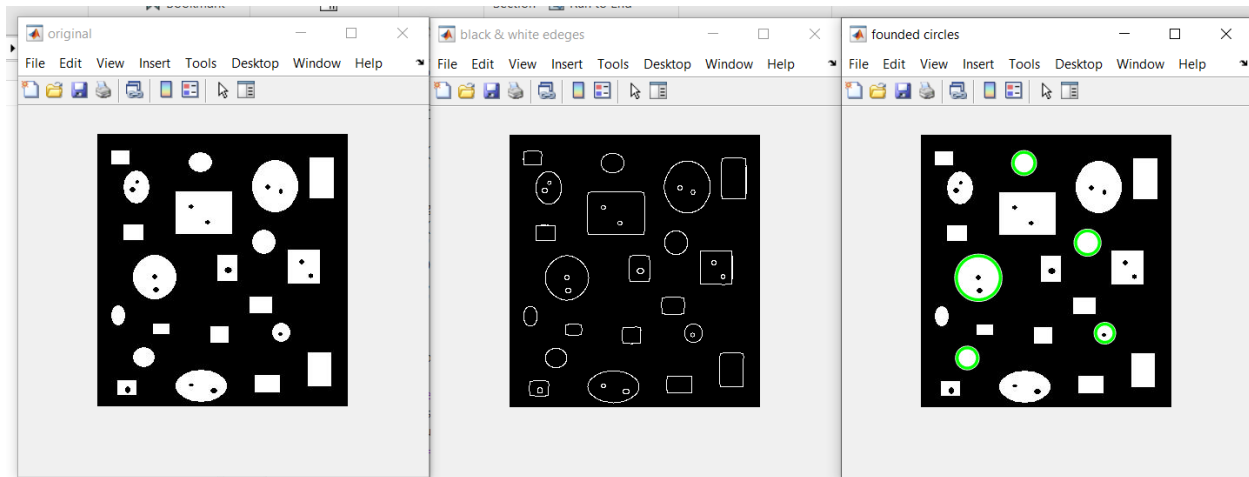


```

1 function S1(inputImage)
2
3     img = imread(inputImage);
4     figure('Name', 'original', 'NumberTitle', 'off');
5     imshow(img);
6     rg2gr = rgb2gray(img);
7     edges = edge(rg2gr, 'canny');
8     figure('Name', 'black & white edges', 'NumberTitle', 'off');
9     imshow(edges);
10    [epicenters, radiuses] = imfindcircles(rg2gr, [6, 33]);
11    figure('Name', 'founded circles', 'NumberTitle', 'off');
12    imshow(img);
13    hold on;
14    viscircles(epicenters, radiuses, 'Color', 'g');
15
16    disp('radiuses => ');
17    disp(radiuses);
18    a = pi * (radiuses.^2);
19    disp('areas => ');
20    disp(a);
21
22

```

خروجی برای تصویر widget1 :



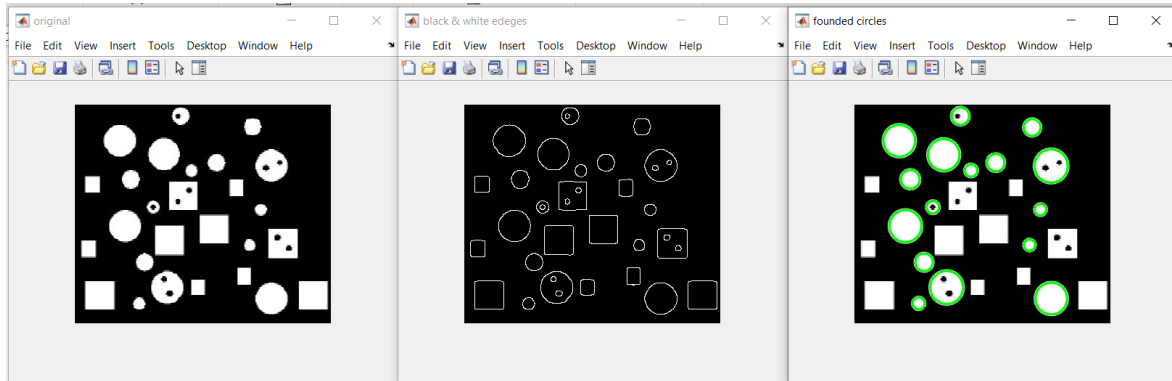
```
Command Window
New to MATLAB? See resources for Getting Started.

>> s1("widget1.jpg")
radiuses =>
    10.4625
    13.5601
    11.5920
    12.3214
    25.1789
```

```
Command Window
New to MATLAB? See resources for Getting Started.

areas =>
    1.0e+03 *
    0.3439
    0.5777
    0.4221
    0.4769
    1.9917
```

خروجی برای تصویر widget2 :



```
Command Window
New to MATLAB? See resources for Getting Started.

>> S1("widget2.jpg")
radiuses =>
    8.5261
    8.5650
    7.8964
    8.1815
    7.6845
    12.0017
    11.9937
    12.8424
    11.5242
    12.0248
    22.0641
    21.9028
    21.6928
    22.6696
    21.8011
    22.8895
```

```
Command Window
New to MATLAB? See resources for Getting Started.

areas =>
    1.0e+03 *
    0.2284
    0.2305
    0.1959
    0.2103
    0.1855
    0.4525
    0.4519
    0.5181
    0.4172
    0.4543
    1.5294
    1.5071
    1.4784
    1.6145
    1.4932
    1.6460
```

تشخيص حفره :

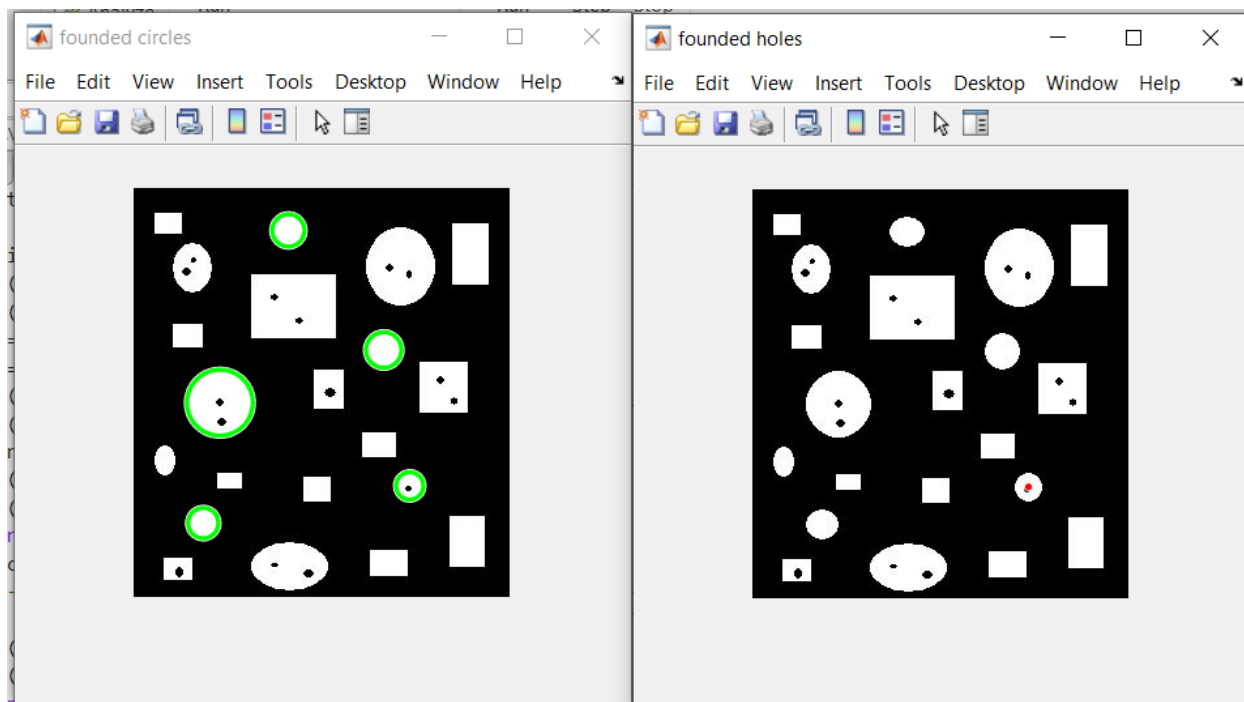
برای تشخیص حفره دوباره از تبدیل هاف استفاده کردیم ولی این بار محدوده‌ی شعاع و حساسیت آن را تغییر دادیم زیرا تمام حفره ها به صورت دایره‌ی کامل نبودند و به همین دلیل تشخیص آن مانند تشخیص دایره به راحتی با تبدیل هاف ممکن نبود.

کد تشخیص حفره :

```
%-----  
[c, r] = imfindcircles(rg2gr,[1,5],"Sensitivity",0.9,"EdgeThreshold",0.1);  
figure('Name', 'founded holes','NumberTitle','off');  
imshow(img);  
hold on;  
viscircles(c,r,'Color','r');  
%-----
```

خروجی :

در میان دایره های تشخیص داده شده، حفره ای که شباهت بیشتری به دایره داشت و توسط `Imfindcircles` قابل تشخیص بود را با رنگ قرمز نمایش داد.



فیلتر میانگین های غیرمحل (NLM) یک روش بسیار ساده و موثر برای کاهش نویز است که کمترین تاثیر را بر روی ساختار های اصلی تصویر به جا می گذارد. این روش مبتنی بر افزودن طبیعی الگوها در تصاویر است و اطلاعات edge و تصاویر و پیکسل ها را حفظ می کند. بعد از این از فیلتر بهبودیافته NLM برای حذف نویز در در حوزه ی پردازش تصویر به طور گسترده ای استفاده شد مانند استفاده در تصاویر MRI.

در کلاس هم در مورد این فیلتر گفته شد که در این روش میانگین گیری لزوما با پیکسل هایی که همسایه هستند نیست، ماسک هایی که پیکسل های شبیه به هم دارند در میانگین گیری هم شرکت می کنند. در واقع برخلاف فیلترهای میانگین محلی که مقدار میانگین گروهی از پیکسل ها را در اطراف پیکسل هدف می گیرند تا تصویر را صاف کنند، فیلتر میانگین های غیرمحل، میانگینی از تمام پیکسل های تصویر را می گیرد که بر اساس شباهت این پیکسل ها به پیکسل های هدف وزن دار شده اند. در مقایسه با الگوریتم های میانگین محلی، شفافیت پس از فیلتر کردن بسیار بیشتر است و جزئیات کمتری در تصویر از بین می رود.

پیچیدگی محاسباتی الگوریتم میانگین غیرمحل از نظر تعداد پیکسل های تصویر از نوع درجه دوم است و استفاده از آن را بسیار پرهزینه است. چندین تکنیک برای سرعت بخشیدن به اجرا پیشنهاد شد. یکی از ساده ترین آن ها شامل محدود کردن محاسبه میانگین برای هر پیکسل به یک پنجره جستجو متمرکز بر روی خود پیکسل، به جای کل تصویر است.

نحوه ی محاسبه ی الگوریتم :

فرض کنید Ω مساحت یک تصویر است و p و q دو نقطه درون تصویر هستند. الگوریتم به صورت زیر است :

$$u(p) = \frac{1}{C(p)} \sum_{q \in \Omega} v(q) f(p, q)$$

$u(p)$: مقدار فیلتر شده تصویر در نقطه p

$v(q)$: مقدار فیلتر نشده تصویر در نقطه q

$C(p)$: normalizing factor به صورت :

$$C(p) = \sum_{q \in \Omega} f(p, q)$$

$f(p,q)$: تابع وزن به صورت :

$$f(p,q) = e^{-\frac{|B(q)-B(p)|^2}{h^2}}$$

که در تابع وزن $B(p)$ به صورت :

$$B(p) = \frac{1}{|R(p)|} \sum_{i \in R(p)} v(i)$$

که در آن $R(p) \subseteq \Omega$ و یک ناحیه مربعی از پیکسل هایی است که p را احاطه کرده است و $|R(p)|$ تعداد پیکسل های منطقه R است.

بخش دوم سوال :

برای حل این سوال ابتدا تصویر را به عنوان ورودی به تابع $S2$ می دهیم با `imread` آن را می خوانیم و سپس در `figure(original pic)` آن را نمایش می دهیم. پس از آن تصویر را با استفاده از دستور `rgb2gray` از `rgb` به باینری تبدیل می کنیم و تصویر مربوطه را در `figure(binary pic)` نمایش می دهیم.

حالا باید نویز گوسی خواسته شده در صورت سوال با میانگین 0 و واریانس 0.25 را به تصویر اعمال کنیم این کار را با دستور `imnoise` انجام می دهیم و تصویر نویزی شده را در `figure(gaussian noised picture)` نمایش می دهیم.

حالا باید طبق صورت سوال با دو روش فیلتر میانگین و `NLM` رفع نویز را روی تصویر اعمال کنیم.

برای فیلتر میانگین باید اول با دستور `fspecial` و آرگومان `average` فیلتر میانگین را فراخوانی کنیم (دیفالت 3 در 3 هست) پس از آن با دستور `filter2`، فیلتر میانگین را روی تصویر نویزی شده اعمال می کنیم و نتیجه را در `figure(3*3 mean filter)` نمایش می دهیم. پس از این باید معیار های خواسته شده برای این فیلتر را به دست آوریم که معیار های `MSE` و `PSNR` هست. هر دو معیار در متلب تابع برای محاسبه دارند که به ترتیب

immse و psnr است. عکس نویزی و رفع نویز شده را به عنوان آرگومان این دو تابع می‌دهیم و خروجی معیاری های خواسته شده در سوال هست که با دستور fprintf آن را نمایش می‌دهیم.

سپس باید فیلتر NLM را روی تصویر نویزی شده اعمال کنیم. با دستور imnlmfilt این کار را انجام می‌دهیم. سپس تصویر رفع نویز شده با این فیلتر را در figure(NLM, Smoothing Degree stimation is) نمایش می‌دهیم. علاوه بر این می‌توانیم درجه رفع نویز را به صورت تخمینی نیز نشان دهیم.

مثل فیلتر قبلی برای این فیلتر هم معیار های MSE و PSNR را با همان توابع و این بار فقط با تصویر رفع نویز شده توسط این فیلتر به عنوان آرگومان، اجرا می‌کنیم.

کد :

```

1 function S2(inputImage)
2
3 img = imread(inputImage);
4 figure('Name', 'original picture','NumberTitle','off');
5 imshow(img);
6 r2g = rgb2gray(img);
7 figure('Name', 'binary picture','NumberTitle','off');
8 imshow(r2g);
9 noisedPic = imnoise(r2g,'gaussian',0,0.25);
10 figure('Name', 'gaussian noised picture (mean=0 , var=0.25)','NumberTitle','off');
11 imshow(noisedPic);
12 tic
13 % -----mean-----
14 meanFilter = fspecial('average');
15 mPic = filter2(meanFilter,noisedPic);
16 figure('Name', '3*3 mean filter','NumberTitle','off');
17 imshow(mPic/255);
18 img2uint8 = im2uint8(mPic);
19 % -----mean filter Criteria-----
20 mseMFCriteria = immse(img2uint8, r2g);
21 fprintf('MSE for mean filter 3*3 ==> %0.4f \n', mseMFCriteria);
22 PsnrMFCriteria = psnr(img2uint8, r2g);
23 fprintf('PSNR for mean filter 3*3 ==> %0.4f \n', PsnrMFCriteria);
24 toc
25 fprintf('\n');
26 fprintf('\n');
27 tic
28 % -----NLM-----
29 [NlmPic,estDegree] = imnlmfilt(noisedPic);
30 esti = num2str(estDegree);
31 figure('Name', ['NLM, Smoothing Degree stimation is >> ', esti], 'NumberTitle','off');
32

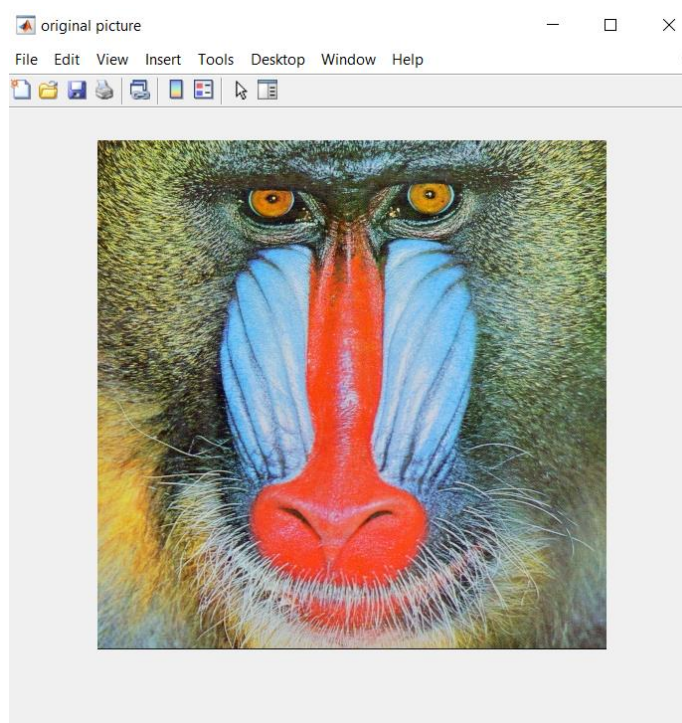
```

```

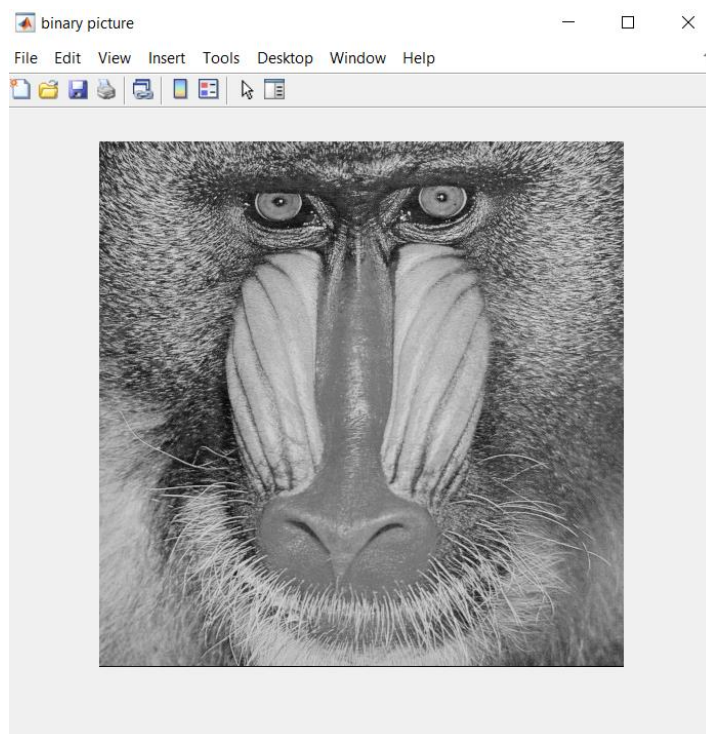
33 % -----NLM filter Criteria-----
34 mseMFCriteria = immse(NlmPic, r2g);
35 fprintf('\n');
36 fprintf('MSE for NLM ==> %0.4f \n', mseMFCriteria);
37 PsnrMFCriteria = psnr(NlmPic, r2g);
38 fprintf('PSNR for NLM ==> %0.4f \n', PsnrMFCriteria);
39 toc

```

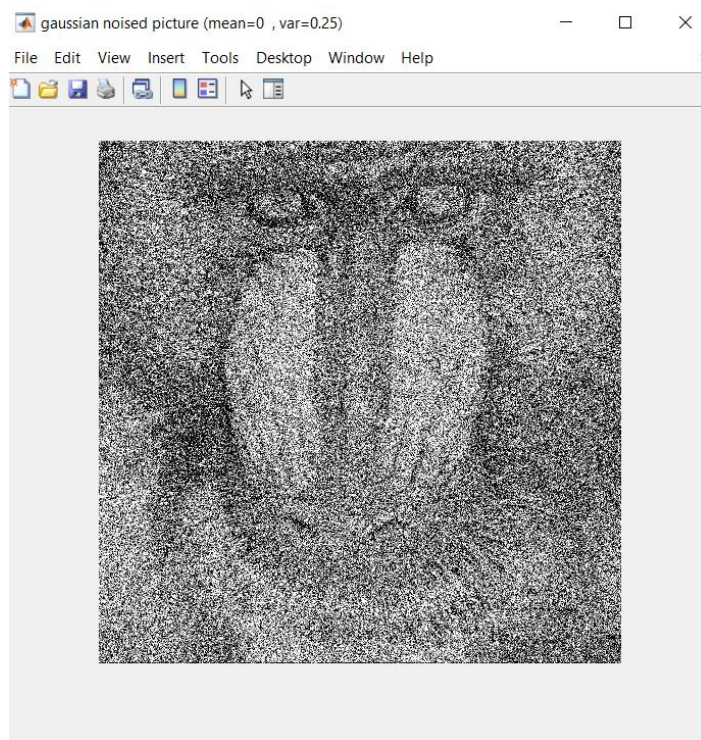
تصویر اصلی :



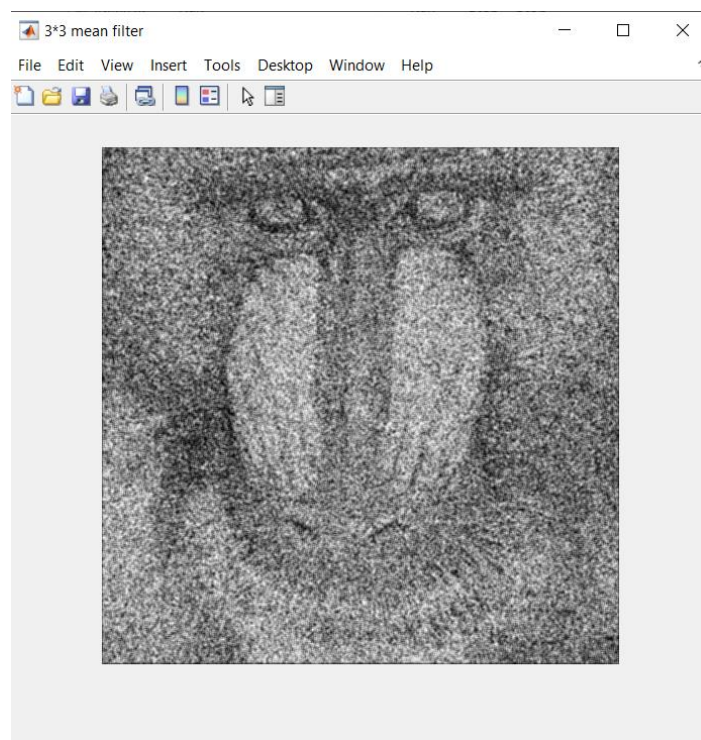
تصویر : gray scale



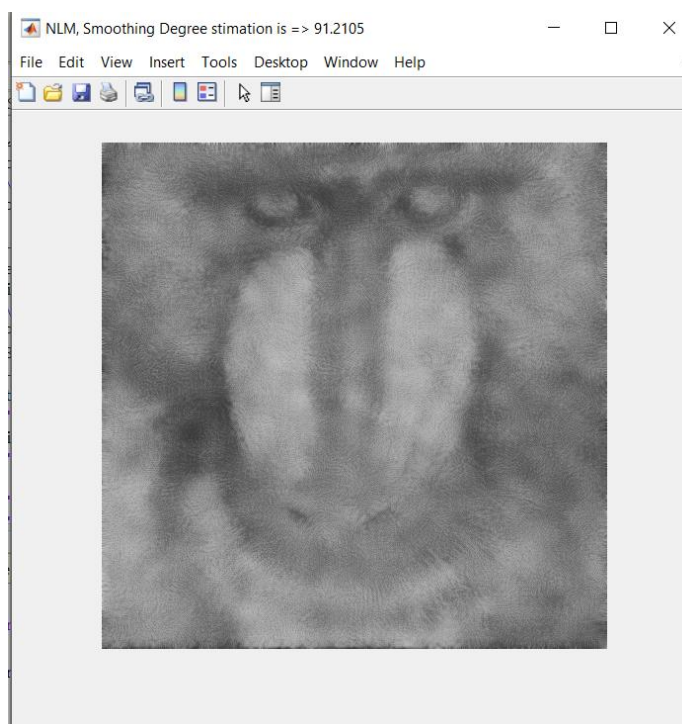
تصویر با نویز گاوسی با میانگین = صفر و واریانس = 0.25 :



تصویر با فیلتر میانگین ساده ی 3×3 :



تصویر با فیلتر NLM :



مقایسه‌ی خروجی‌ها از نظر کیفیت بصری :

در این سوال خروجی هر دو فیلتر چندان مطلوب نبود و تصویر رفع نویز شده از نظر بصری کیفیت چندان خوبی نداشت. در کل این دو فیلتر برای نویز گوسی با واریانس کمتر عملکرد بهتری نشان می‌دهند. در فیلتر میانگین ساده 3×3 ، تصویر بعد از اصلاح نویز همچنان دارای خرابی‌های زیادی هست و اختشاشات زیادی دارد و نسبت به تصویر نویزی چندان بهتر نشده است.

با فیلتر NLM بعد از اصلاح نویز تصویر دارای اختشاشات کمتری است و صاف تر شده به نسبت فیلتر میانگین ساده اما همچنان عکس ناواضح هست و جزییات از دست رفته و تار شده است.

در کل می‌توان گفت فیلتر NLM از نظر کم کردن اختشاشات در مقایسه‌ی بصری بهتر از فیلتر میانگین ساده عمل کرد.

مقایسه‌ی معیار های MSE و PSNR برای دو فیلتر:

در قسمت MSE هر چقدر کوچک تر باشد عدد فیلتر ما عملکرد بهتری داشته است که در اینجا برای فیلتر NLM عدد کمتر بوده است و بنابراین در این معیار فیلتر NLM بهتر بوده است.

در قسمت PSNR هر چقدر عدد بزرگ تر باشد یعنی فیلتر عملکرد بیشتری دارد که در اینجا عدد فیلتر NLM بیشتر بوده و از نظر کارایی هم این فیلتر بهتر عمل کرده است.

```
>> S2("Baboon.jpg")
MSE for mean filter 3*3 ==> 17513.8611
PSNR for mean filter 3*3 ==> 5.6970
Elapsed time is 0.393487 seconds.

MSE for NLM ==> 873.7296
PSNR for NLM ==> 18.7170
Elapsed time is 0.711832 seconds.
fx >>
```

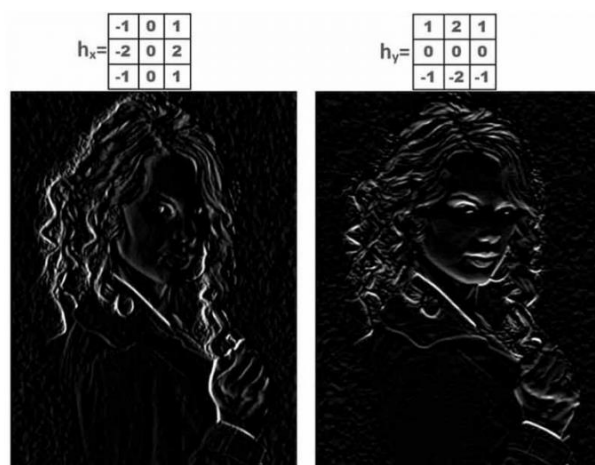
برای مقایسه‌ی زمان اجرای دو فیلتر می‌بینیم که همانطور که در ابتدای این سوال برای الگوریتم NLM توضیح دادیم زمان اجرای آن طولانی تر است نسبت به فیلتر میانگین ساده هم در اینجا قابل مشاهده است. (با استفاده از دستورات tic قبل از اجرای کد و toc در انتهای اجرا این بازه‌ی اجرایی فیلتر را به دست آوردیم).

-3

فیلتر canny یکی از معروف ترین روش های لبه گیری در پردازش تصویر است. ساختار فیلتر canny طوری طراحی شده که به نویز حساسیت کمتری دارد. فیلتر canny برای آشکارسازی لبه های تصویر، از یک الگوریتم 5 مرحله ای استفاده می‌کند. این روش شامل ویژگی هایی است مثل : لبه گیری با خطای کمی همراه است یعنی تا جای ممکن بیشتر لبه‌ی قابل تمایز در تصویر را نشان دهد، نویز لبه های اشتباه ایجاد نکند و نقطه لبه‌ی تشخیص داد شده در مرکز لبه باشد.

5 مرحله‌ی این فیلتر شامل :

- **کاهش نویز تصویر (noise reduction) :** تصویر ورودی ممکن است حاوی نویز باشد و اگر نویز تصویر کاهش نیابد، بسیاری از نقاط تصویر به اشتباه لبه شناسایی می‌شوند. همچنین ممکن است تصویر شامل یک سری **object** های خیلی کوچک باشد که مرتبط با **object** اصلی تصویر نباشد. برای همین در مرحله اول فیلتر **canny** یک فیلتر گوسی روی تصویر اعمال می‌کند تا نویز تصویر را کاهش دهد. علاوه بر کاهش نویز تصویر، **object** های کوچک و غیر ضروری نیز از تصویر محو می‌شوند.
 - **محاسبه گرادیان تصویر :** در مرحله دوم الگوریتم **canny** ، مقدار (**intensity**) و جهت (**direction**) گرادیان محاسبه می‌شود. برای اینکار لازم است در ابتدا از فیلترهای **sharpening** استفاده می‌شود تا لبه های تصویر برجسته شوند. چون تصویر ممکن است در راستای های مختلفی لبه داشته باشد الگوریتم **Canny** از 4 فیلتر برای تشخیص لبه های افقی، عمودی و قطری در تصویر بدون نویز استفاده می‌کند. عملگر هایی مثل **Roberts** و **Prewitt** و **Sobel** ، مشتق اول را در راستای افقی و عمودی به ما می‌دهند که با استفاده از آنها می‌توان گرادیان و راستای لبه را مشخص کرد.
برای مثال با استفاده از **sobel** برای تصویر زیر داریم :
- بعد از اعمال سوبل، دو تصویر ساخته می‌شود که در یکی از آنها لبه های افقی و در یکی از آنها لبه های عمودی تصویر برجسته می‌شود:



طبق روابط زیر intensity و direction گرادیان محاسبه می شود:

Gradient intensity

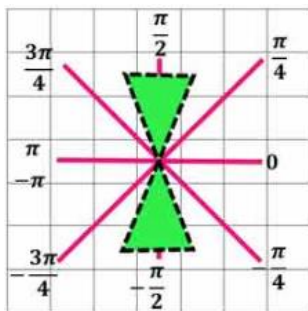
$$M = \sqrt{G_x^2 + G_y^2}$$

Gradient Direction

$$\theta = \tan^{-1} \frac{G_y}{G_x}$$

در intensity گرادیان لبه‌های تصویر بهبود پیدا کرده نسبت به direction و از این مرحله به بعد، پردازش‌ها روی تصویر intensity گرادیان انجام می شود تا لبه های تصویر استخراج شوند.

- **مرحله سوم: حذف/سرکوب نقاط غیربیشینه :** هدف الگوریتم canny پیدا کردن مرکز لبه‌های تصویر است. در تصویر intensity گرادیان جاهایی که لبه وجود دارد، ضخامت بالا هست. و اگر از همین تصویر به طور مستقیم در مرحله های بعد استفاده شود، لبه‌های بدست آمده در تصویر نهایی ضخامت بالایی خواهند داشت که مطلوب نیست. برای حل این مشکل با سرکوب نقاط غیر بیشینه که می تواند باعث صفر کردن همه مقادیر گرادیان به جز بیشینه های محلی که نشان دهنده مکان هایی با سریعترین تغییر در مقدار روشنایی هستند شود و مناسب مراحل بعدی باشند.
برای اینکه متوجه شویم کدام نقاط غیر بیشینه هستند لازم است که تک تک پیکسل های تصویر intensity را پیمایش کنیم، با کمک direction گرادیان دو پیکسل کناری را در یک جهت را پیدا کنیم، سپس مقدار آنها را با مقدار intensity پیکسل مورد نظر مقایسه کنیم. اگر مقدار یکی از دو پیکسل همسایه کمتر باشد، در اینصورت پیکسل مورد نظر یک نقطه غیربیشینه هست و باید حذف شود(به صفر تبدیل شود). در غیر اینصورت پیکسل یک نقطه بیشینه هست و باید حفظ شود(مقدار خودش باقی بماند). برای پیدا کردن دو همسایه کناری به direction هر پیکسل در تصویر direction گرادیان نگاه می کنیم.



- **مرحله چهارم: آستانه گذاری دوگانه :** در آستانه گذاری ، دو حد آستانه (حد آستانه بالا و حد آستانه

پایین) برای تصویر گرادیان (تصویر خروجی مرحله سوم) انتخاب می کنیم. سپس مقادیر پیکسل های تصویر گرادیان را با دو مقدار حد آستانه مقایسه کرده و پیکسل های تصویر را به سه گروه تقسیم میکنیم: پیکسل های قوی: پیکسل هایی که مقدار آنها بزرگتر از حد آستانه بالا است. این پیکسل ها روی لبه و قرار گرفته اند و مقدار **intensity** آنها بالا است.

پیکسل های غیرمرتبط: پیکسل هایی که مقدار آنها کمتر از حد آستانه پایین است. این پیکسل ها در یک جایی که لبه وجود ندارد قرار گرفته اند و مقدار **intensity** آنها پایین است.

پیکسل های ضعیف : پیکسل هایی که مقدار آنها کمتر از حد آستانه بالا و بیشتر از حد آستانه پایین هست. مقدار **intensity** این پیکسل ها نه به قدری بزرگ هست که به عنوان لبه شناسایی شوند و نه به قدری پایین هست که به عنوان غیرلبه شناسایی شوند.

وضعیت پیکسل های قوی و غیر مرتبط در این مرحله مشخص هست. به پیکسل های قوی مقدار 1 (لبه) اختصاص داده می شود و به پیکسل های غیر مرتبط مقدار صفر (غیرلبه) اختصاص داده می شود. ولی تکلیف پیکسل های ضعیف در مرحله بعد مشخص می شود.

مرحله پنجم: دنبال کردن لبه های تصویر با روش پسماند : در این مرحله باید برای پیکسل های ضعیف یک تصمیم نهایی گرفته شود، یا باید به لبه تبدیل شوند (پیکسل های قوی) و یا حذف شوند (پیکسل های غیرمرتبط). برای اینکه تصمیم بگیریم یک پیکسل ضعیف، باید صفر شود یا یک، به همسایگی آن نگاه می کنیم تا ببینیم که این پیکسل آیا در کنار یک لبه قوی قرار گرفته است یا نه.

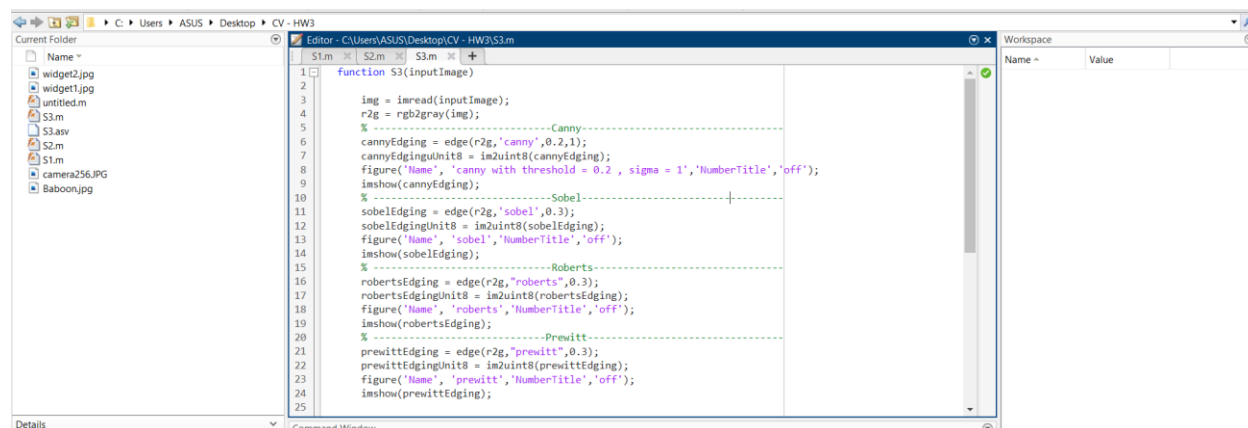
اگر در همسایگی 3×3 یک پیکسل ضعیف، حداقل یک پیکسل قوی باشد، در اینصورت این پیکسل در کنار یک لبه قوی قرار گرفته است و باید به مقدار یک (لبه) تبدیل شود و اگر در همسایگی آن هیچ پیکسل قوی وجود نداشته باشد به مقدار صفر (حذف میکنیم) تبدیل می کنیم.

بخش دوم سوال:

ابتدا تصویر camera256 را به عنوان آرگومان تابع **S3** برای اجرا می دهیم. تصویر را با **imread** دریافت می کنیم و آن را به گری اسکیل تبدیل می کنیم. حال باید لبه گیری ها را اعمال کنیم تا مقایسه انجام دهیم.

برای لبه گیری از دستور **edge** استفاده می کنیم با نام متدی که برای لبه گیری می خواهیم روی عکس اعمال شود و پس از آن پارامتر های مربوط به روش انتخابی را وارد می کنیم. برای روش **Canny** پارامتر سیگما و **threshold** را داریم که هر چقدر عدد سیگما برای این روش بزرگ تر باشد جزئیات درون تصویر لبه گیری شده کمتر خواهد بود. سیگما عددی بین 0.6 تا 2.4 هست.

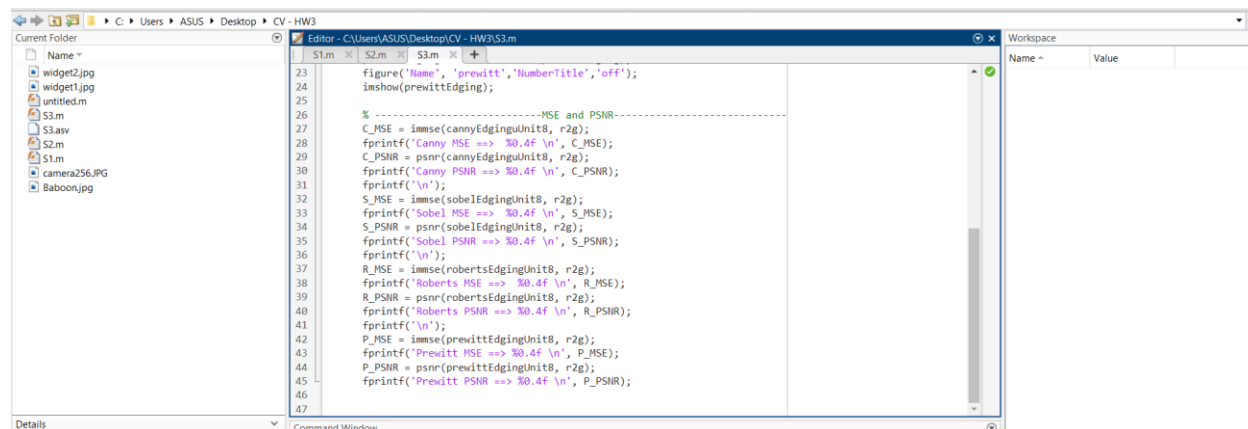
کد :



```

1 function S3(inputImage)
2
3     img = imread(inputImage);
4     r2g = rgb2gray(img);
5     % -----Canny-----
6     cannyEdging = edge(r2g,'canny',0.2,1);
7     cannyEdgingUnit8 = im2uint8(cannyEdging);
8     figure('Name', 'canny with threshold = 0.2 , sigma = 1','NumberTitle','off');
9     imshow(cannyEdging);
10    % -----Sobel-----
11    sobelEdging = edge(r2g,'sobel',0.3);
12    sobelEdgingUnit8 = im2uint8(sobelEdging);
13    figure('Name', 'sobel','NumberTitle','off');
14    imshow(sobelEdging);
15    % -----Roberts-----
16    robertsEdging = edge(r2g,'roberts',0.3);
17    robertsEdgingUnit8 = im2uint8(robertsEdging);
18    figure('Name', 'roberts','NumberTitle','off');
19    imshow(robertsEdging);
20    % -----Prewitt-----
21    prewittEdging = edge(r2g,'prewitt',0.3);
22    prewittEdgingUnit8 = im2uint8(prewittEdging);
23    figure('Name', 'prewitt','NumberTitle','off');
24    imshow(prewittEdging);
25

```



```

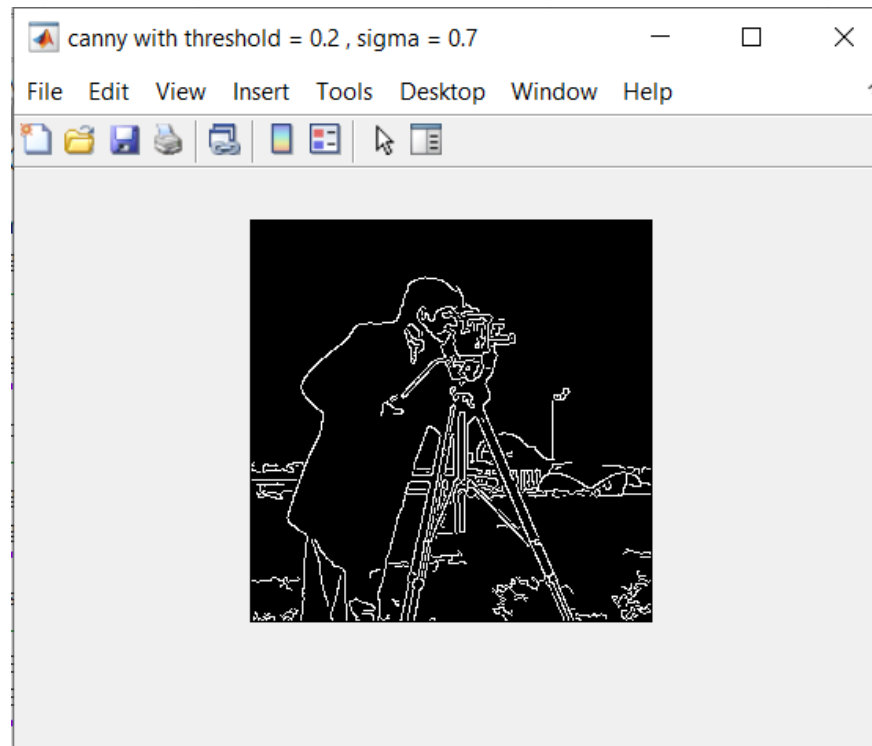
23     figure('Name', 'prewitt','NumberTitle','off');
24     imshow(prewittEdging);
25
26    % -----MSE and PSNR-----
27    C_MSE = immse(cannyEdgingUnit8, r2g);
28    fprintf('Canny MSE ==> %0.4f \n', C_MSE);
29    C_PSNR = psnr(cannyEdgingUnit8, r2g);
30    fprintf('Canny PSNR ==> %0.4f \n', C_PSNR);
31    fprintf('\n');
32    S_MSE = immse(sobelEdgingUnit8, r2g);
33    fprintf('Sobel MSE ==> %0.4f \n', S_MSE);
34    S_PSNR = psnr(sobelEdgingUnit8, r2g);
35    fprintf('Sobel PSNR ==> %0.4f \n', S_PSNR);
36    fprintf('\n');
37    R_MSE = immse(robertsEdgingUnit8, r2g);
38    fprintf('Roberts MSE ==> %0.4f \n', R_MSE);
39    R_PSNR = psnr(robertsEdgingUnit8, r2g);
40    fprintf('Roberts PSNR ==> %0.4f \n', R_PSNR);
41    fprintf('\n');
42    P_MSE = immse(prewittEdgingUnit8, r2g);
43    fprintf('Prewitt MSE ==> %0.4f \n', P_MSE);
44    P_PSNR = psnr(prewittEdgingUnit8, r2g);
45    fprintf('Prewitt PSNR ==> %0.4f \n', P_PSNR);
46
47

```

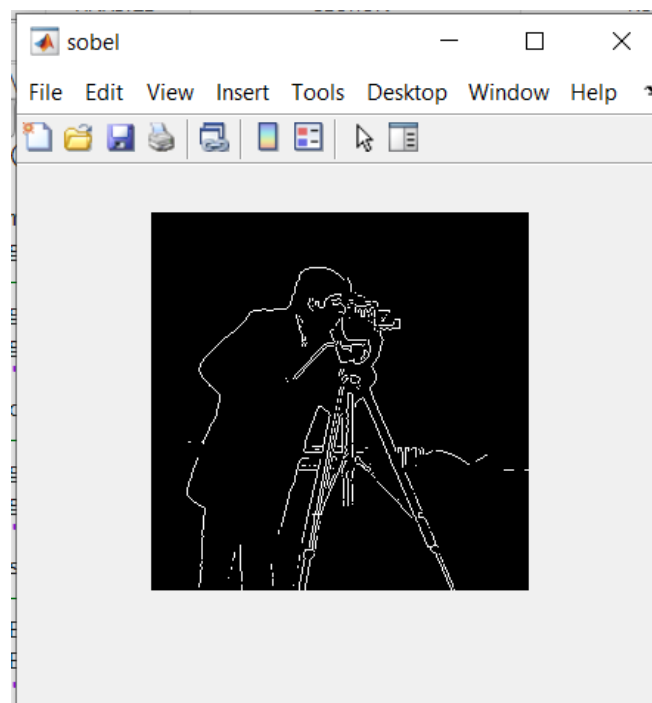
برای اینکه باید تصاویر لبه گیری شده در پارامتر های مختلف بررسی شوند این روش ها را با چند سیگمای متفاوت و چندین آستانه ی مختلف اجرا می کنیم.

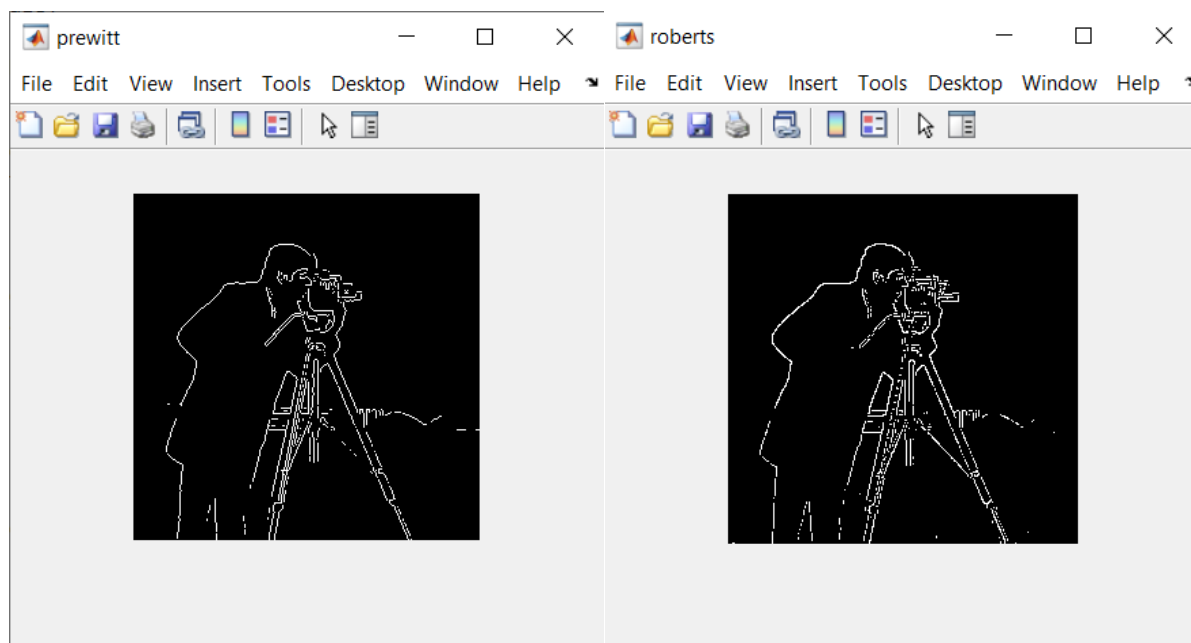
سپس خروجی هر کدام را ابتدا به صورت بصری مقایسه کرده و بعد از معیار های **MSE** و **PSNR** را مقایسه می کنیم.

Canny → Threshold = 0.2 , sigma = 0.7



Sobel, Roberts, Prewitt → Threshold = 0.2

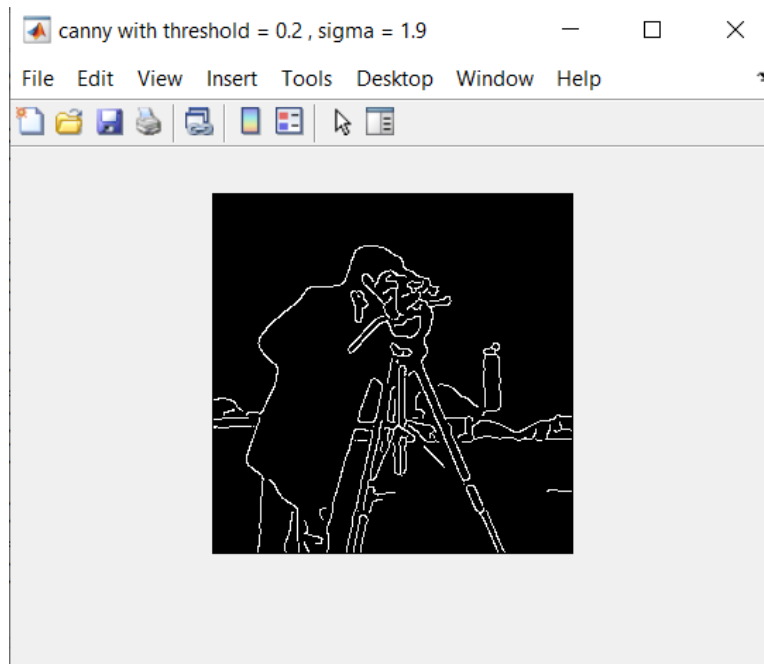




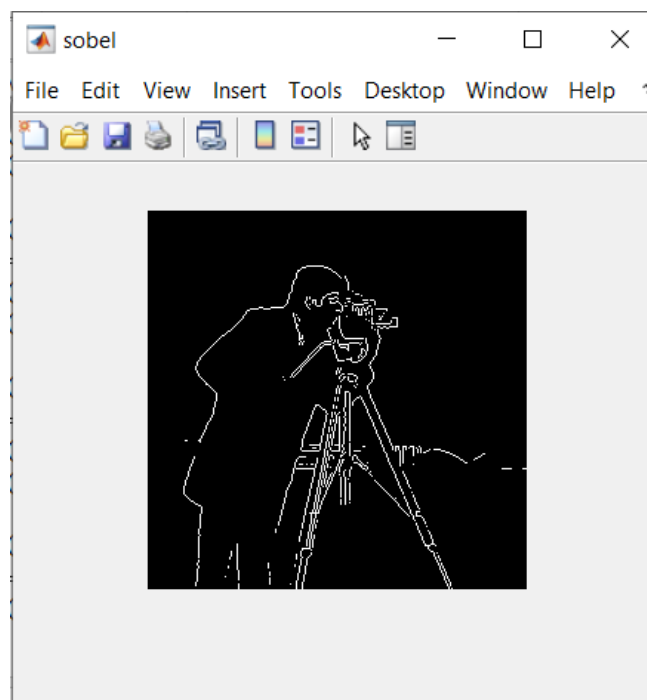
با پارامترهای ذکر شده در بالا روش Canny عملکرد بهتری داشته نسبت به 3 روش دیگر و جزئیات بیشتری را از لبه ها در اختیار ما قرار داد و حتی جزئیات زیادی از پشت تصویر را هم لبه گیری کرده و بخاطر پایین بود سیگما جزئیات داخل تصویر لبه گیری شده هم زیاد است.

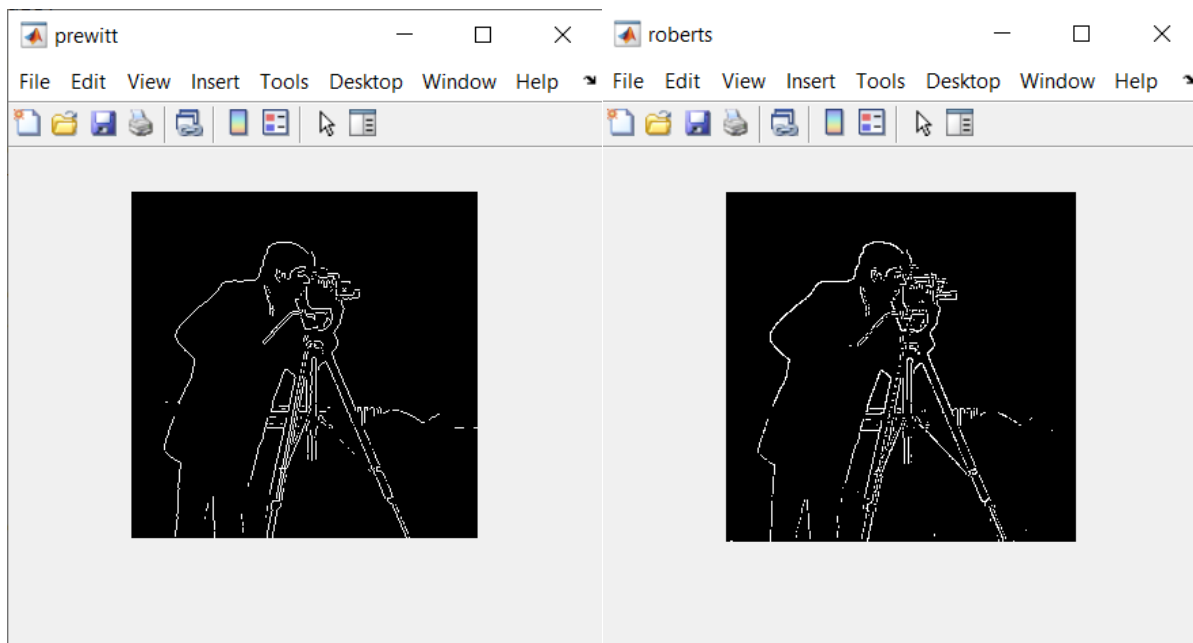
اگر فقط مقدار سیگما برای فیلتر Canny را نسبت به حالت قبل تغییر دهیم همچنان این فیلتر از بقیه عملکرد بهتری دارد فقط بخاطر بیشتر شدن مقدار سیگما جزئیات در تصویر کمتر شده است. بقیه نیز نتایج قبل را دارند فقط بخاطر سهولت بررسی در گزارش دوباره عکس ها را اینجا در کنار هم قرار می دهیم.

Canny → Threshold = 0.2 , sigma =1.9



Sobel, Roberts, Prewitt → Threshold = 0.2

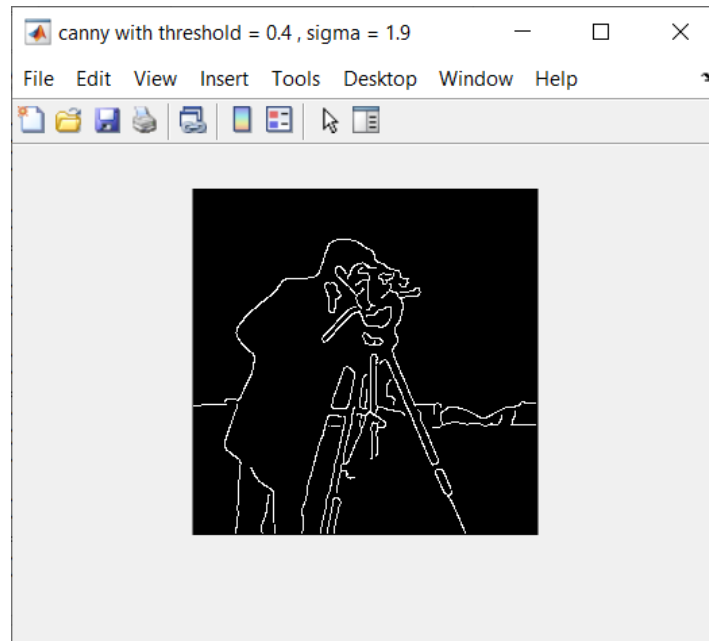




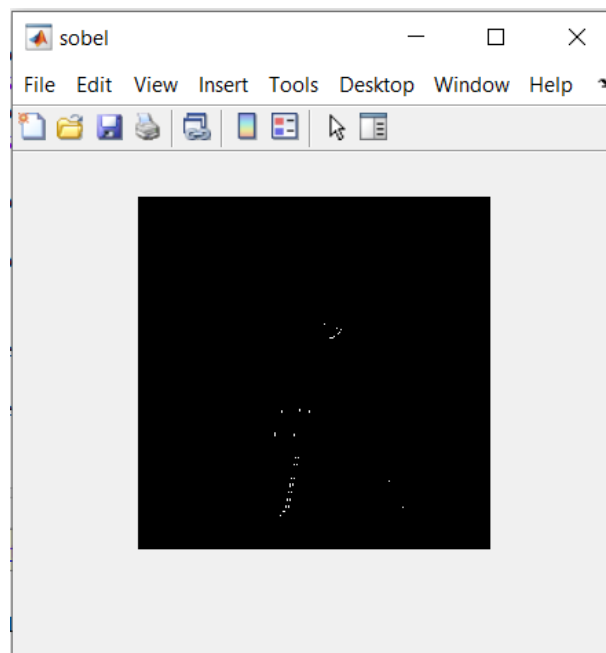
این بار پارامتر آستانه را برای همه‌ی فیلترها تغییر می‌دهیم. و نتایج را دوباره بررسی می‌کنیم.

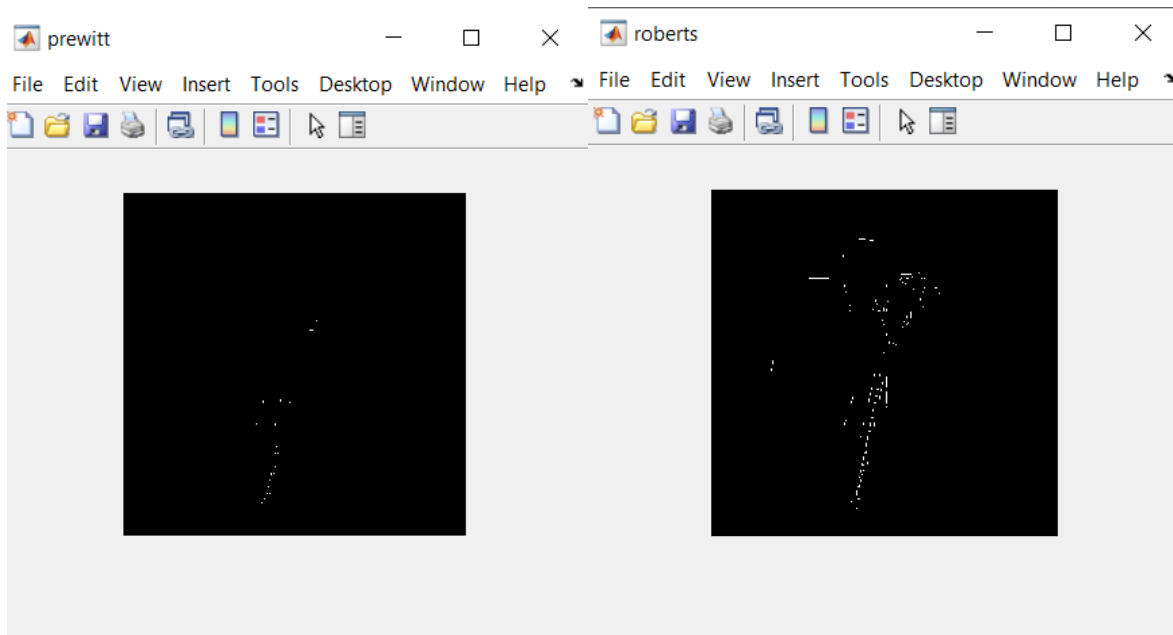
مقدار سیگما را برای فیلتر **Canny** مشابه حالت قبل نگه می‌داریم. قابل مشاهده هست که نتایج این فیلتر همچنان از نظر بصری در لبه‌گیری بسیار بهتر از بقیه است و میزان بالا رفتن پارامتر آستانه تاثیر زیادی در کیفیت فیلترهای دیگر می‌گذارد و لبه‌گیری در این حالت با این فیلترها بسیار ناکارآمدتر از فیلتر **Canny** است.

Canny → Threshold = 0.2 , sigma =1.9



Sobel, Roberts, Prewitt → Threshold = 0.3





از نظر بصری با دستکاری پارامترهای فیلترها دیدیم که فیلتر Canny از همه نتایج بهتری را در حالت‌های مختلف از خود نشان داد. حالا معیارهای PSNR و MSE را مقایسه می‌کنیم.

برای حالت آستانه 0.2 و سیگما 0.7 داشتیم :

```
Command Window
New to MATLAB? See resources for Getting Started.
Prewitt PSNR ==> 5.5291
>> S3("camera256.JPG")
Canny MSE ==> 18655.0272
Canny PSNR ==> 5.4228

Sobel MSE ==> 18204.5049
Sobel PSNR ==> 5.5290

Roberts MSE ==> 18031.7644
Roberts PSNR ==> 5.5704

Prewitt MSE ==> 18204.3415
Prewitt PSNR ==> 5.5291
fx >> |
```

برای حالت آستانه 0.2 و سیگما 1.9 داشتیم :

```
Command Window
New to MATLAB? See resources for Getting Started.
>> S3("camera256.JPG")
Canny MSE ==> 18468.3842
Canny PSNR ==> 5.4665

Sobel MSE ==> 18204.5049
Sobel PSNR ==> 5.5290

Roberts MSE ==> 18031.7644
Roberts PSNR ==> 5.5704

Prewitt MSE ==> 18204.3415
Prewitt PSNR ==> 5.5291
fx >> |
```

برای حالت آستانه 0.4 و سیگما 1.9 داشتیم :

```
New to MATLAB? See resources for Getting Started.
>> S3("camera256.JPG")
Canny MSE ==> 18404.5019
Canny PSNR ==> 5.4816

Sobel MSE ==> 17760.2860
Sobel PSNR ==> 5.6363

Roberts MSE ==> 17756.4301
Roberts PSNR ==> 5.6372

Prewitt MSE ==> 17752.8036
Prewitt PSNR ==> 5.6381
fx >>
```

برای مقایسه راحت تر داده های یک بار از اجرا را کنار هم می گذاریم :

Sigma = 07 , Threshold = 0.2	MSE	PSNR
Canny	18655.0272	5.4228
Sobel	18204.5049	5.5290
Roberts	18031.7644	5.5704
Prewitt	18204.3415	5.5291

هر چقدر مقدار PSNR بیشتر و MSE کمتر باشد بهتر است، می‌توان دید که مقدار PSNR برای فیلتر Roberts از بقیه بیشتر است و MSE آن نیز از بقیه کمتر است در نتیجه در معیار ها این فیلتر بهتر بوده و عملکرد بهینه تری دارد.