



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

تمرین سری دو

درس بینایی ماشین

فرهاد دلیرانی

۹۶۱۳۱۱۲۵

dalirani@aut.ac.ir

dalirani.1373@gmail.com

فهرست

۱	ابزارهای استفاده شده
۲	تمرین ۱
۵	تمرین ۲
۱۰	تمرین ۳
۱۹	تمرین ۴
۲۲	تمرین ۵
۲۵	تمرین ۶

ابزارهای استفاده شده

زبان برنامه نویسی: پایتون ۳.۶

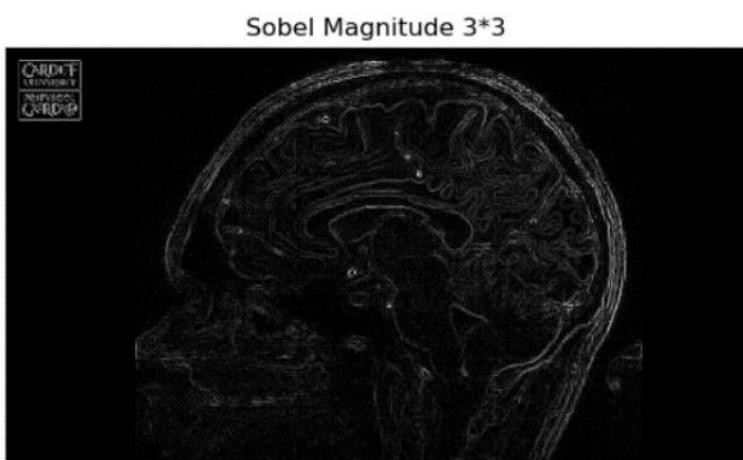
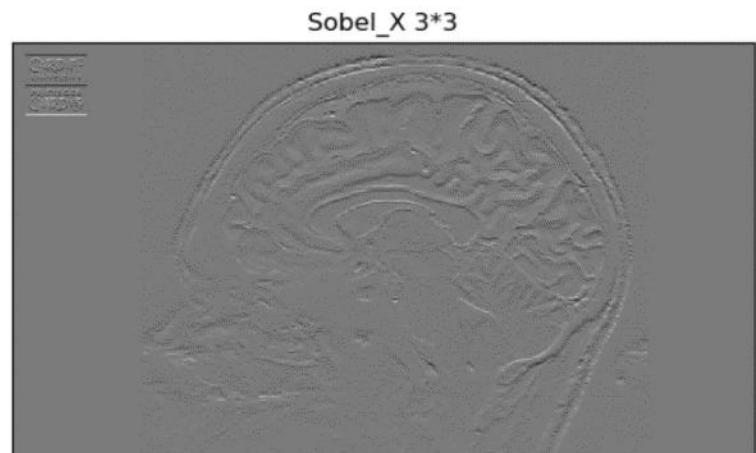
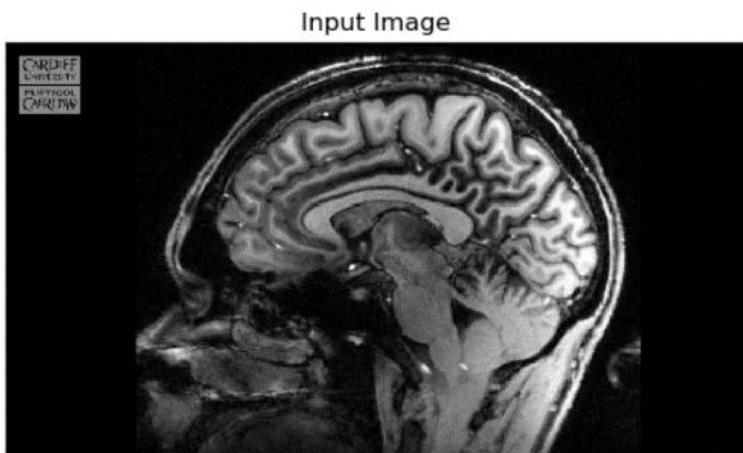
محیط توسعه: PyCharm

سیستم عامل: Windows 10

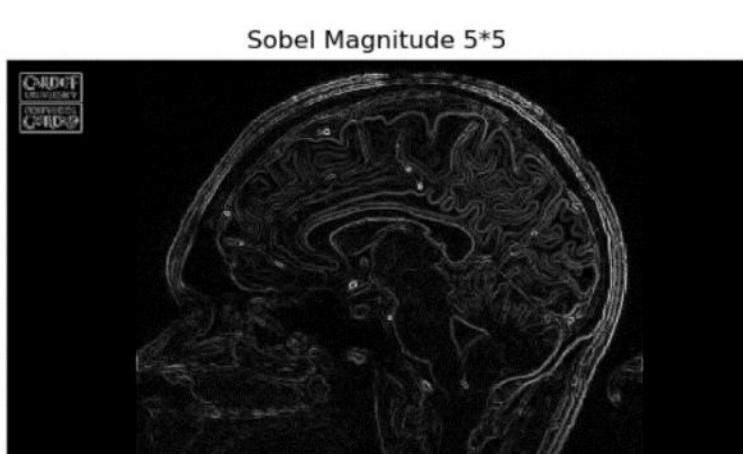
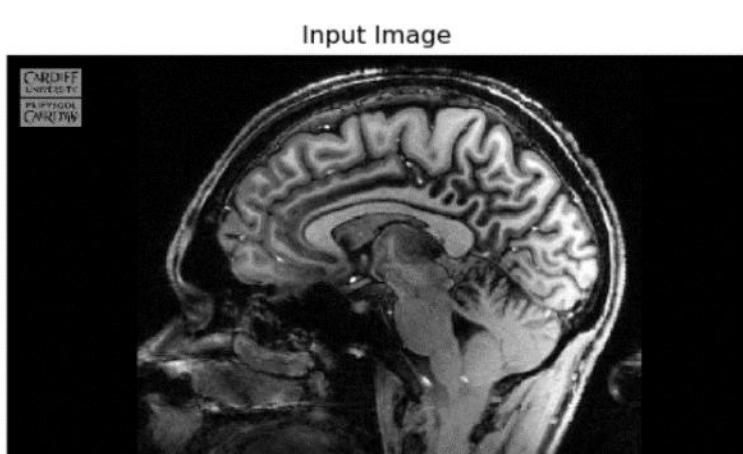
تمرین ۱

کد این سوال در `problem-1.py` قرار دارد. در این کد از ۴ فیلتر Sobel با سایزهای 3×3 , 5×5 , 7×7 و 9×9 استفاده کرده‌ایم. در زیر خروجی حاصل از اجرای کد را مشاهده می‌کنید.

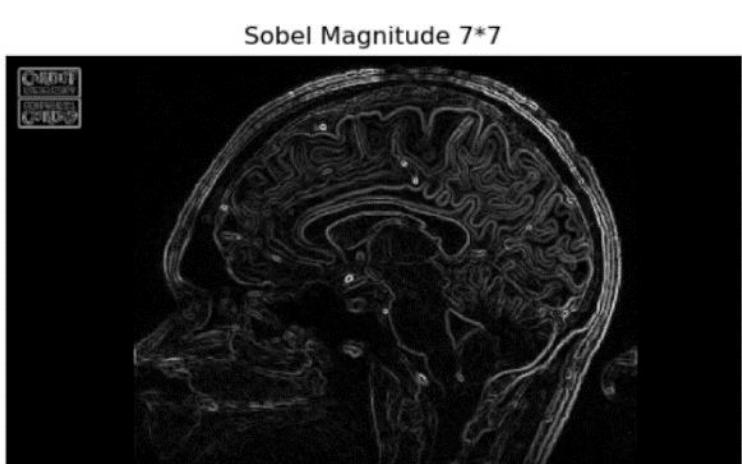
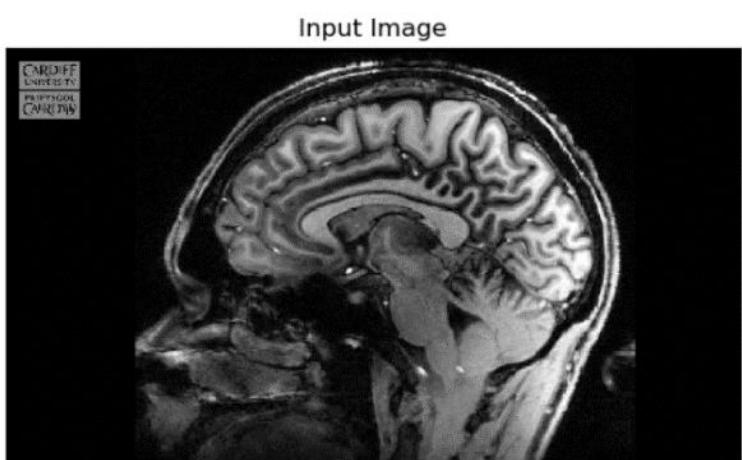
عملگر سوبل با سایز 3×3 :



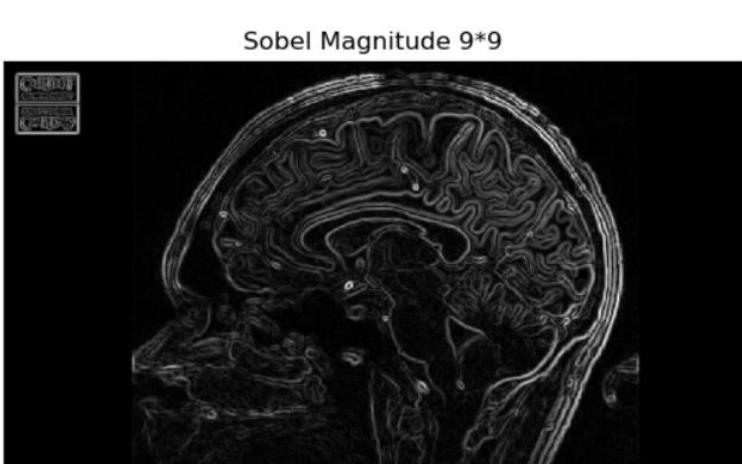
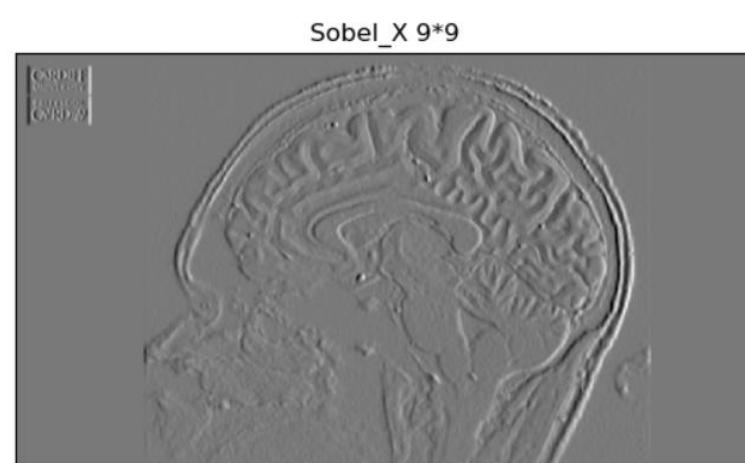
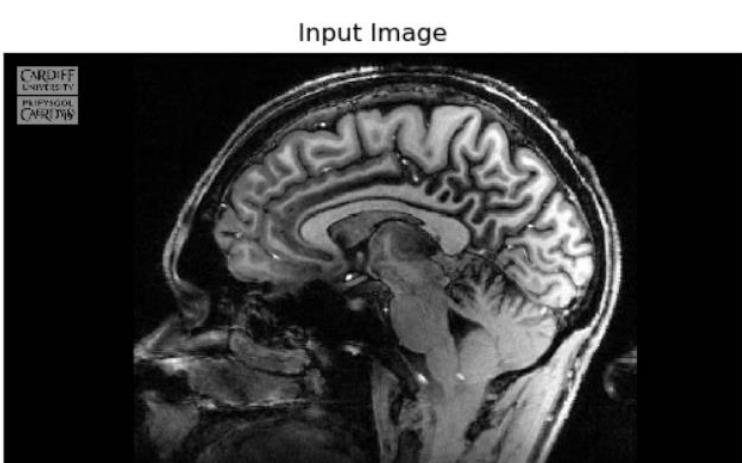
عملگر سوبل با سایز 5×5 :



عملگر سوبل با سایز 7×7 :

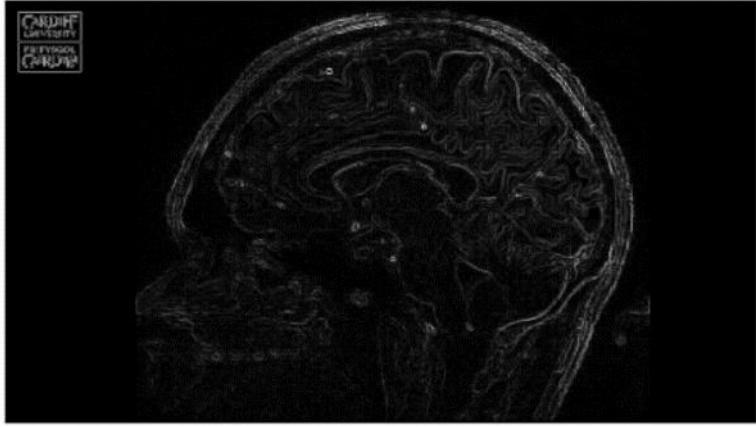


عملگر سوبل با سایز ۹*۹:

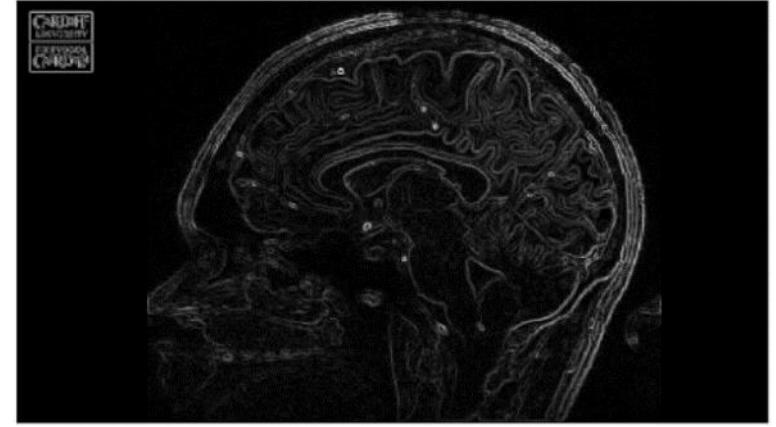


برای مقایسه بهتر، **Magnitude** فلیترهای سوبل با سایزهای متفاوت را در زیر مشاهده می‌کنید:

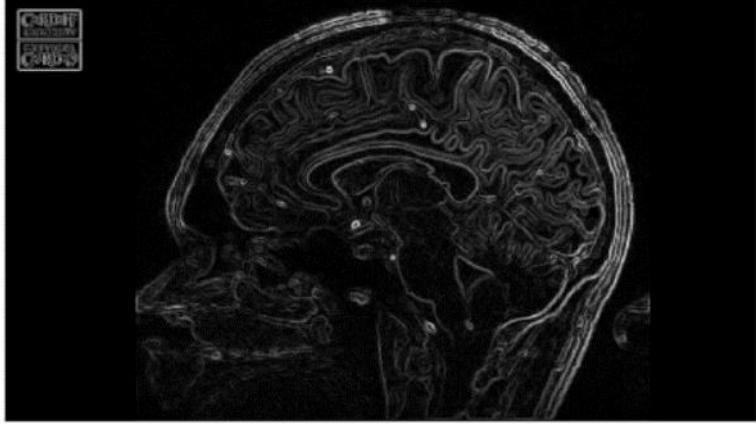
Sobel Magnitude 3*3



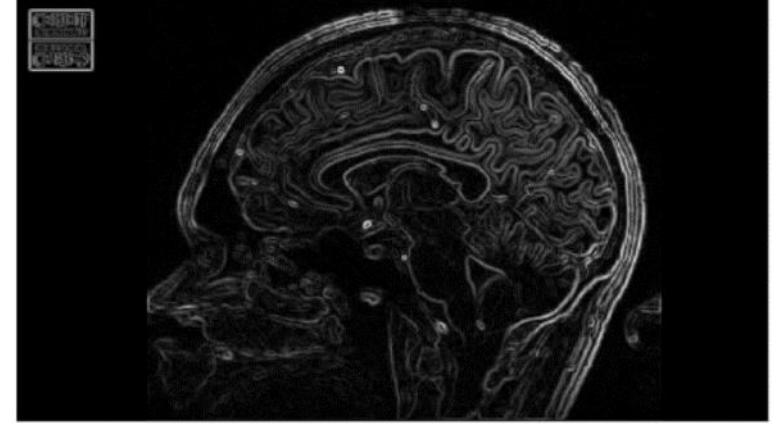
Sobel Magnitude 5*5



Sobel Magnitude 7*7



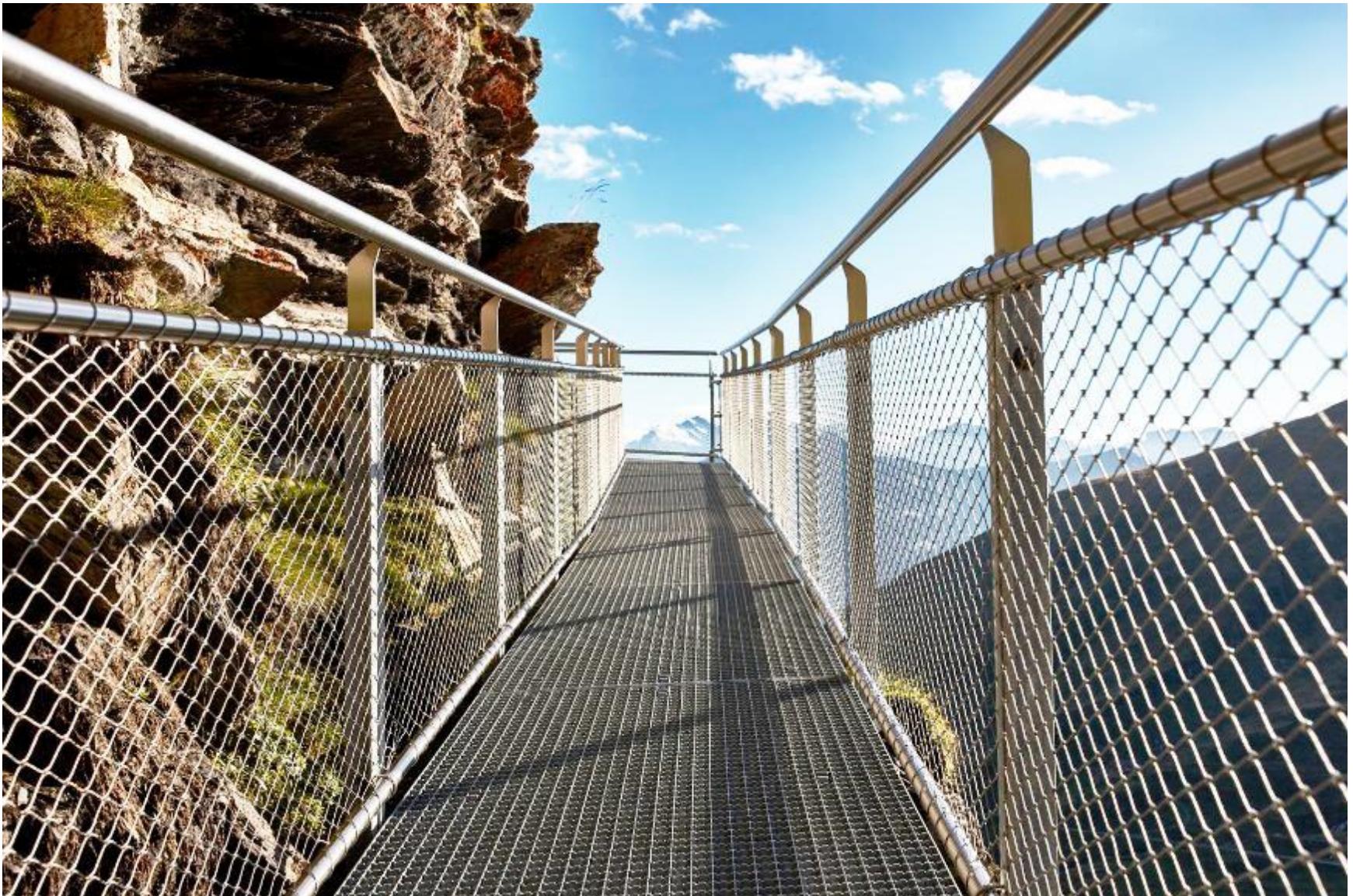
Sobel Magnitude 9*9



همین طور که در نتایج مشاهده می‌شود، هر چه ابعاد فیلتر بزرگ‌تر شده است، لبه‌ها نسبت به لبه‌های حاصل از فیلترهای کوچک‌تر، پهن‌تر و بلورتر شده‌اند. البته این اثر قابل پیش‌بینی بود، زیرا هر چه سایز فیلتر بزرگ‌تر می‌شود پیکسل‌های بیشتری در مقدار محاسبه‌ی لبه در یک نقطه شرکت می‌کنند. همچنین حساسیت به نویز و تغییرات کم شدت روشنایی کمتر می‌شود. با افزایش سایز بزرگی سوبل، بار محاسباتی نیز افزایش می‌یابد.

کد این قسمت در `problem-2.py` قرار دارد. ابتدا تاثیر اندازه‌ی فیلتر را بررسی می‌کنیم سپس تاثیر سیگما را بررسی می‌کنیم.

تصویر ورودی سوال:



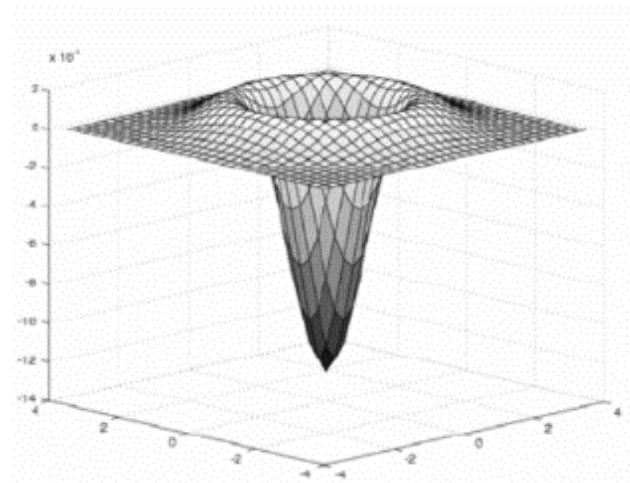
برای بررسی تاثیر اندازه‌ی فیلتر بر عملکرد LoG چهار فیلتر با مشخصات زیر را مورد آزمایش قرار می‌دهیم.
 $(\text{sigma}=0.8, \text{size}=3)$, $(\text{sigma}=0.8, \text{size}=5)$, $(\text{sigma}=0.8, \text{size}=7)$, $(\text{sigma}=0.8, \text{size}=17)$

در زیر نحوه عملکرد این فیلترها را مشاهده می‌کنید:

فیلتر با سایز 3×3 و سیگما 0.8 :

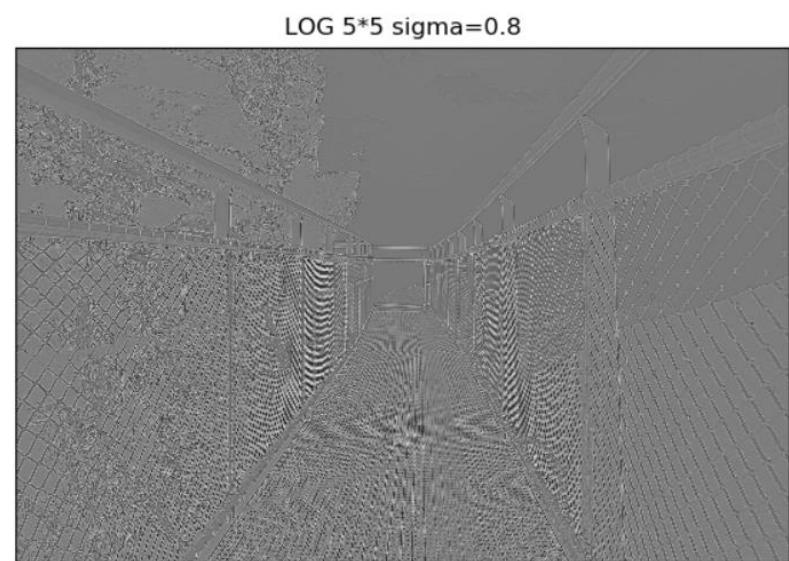


در فیلتر با سایز 3×3 و سیگما 0.8 ، اتفاق جالبی رخ داده است. فیلتر LoG به گونه‌ای است که مرکز آن مقدار بیشتری نسبت به سایر نقاط دارد، و هرچه از مرکز دور می‌شویم بزرگی مقدار پیکسل‌ها در فیلتر کاهش پیدا می‌کنند. در صورتی که اندازه‌ی فیلتر نسبت به سیگما کوچک باشد و المنت‌ها با علامت مثبت درون فیلتر جا نشوند، فیلتر LoG درست عمل نمی‌کند و عملیات استخراج لبه از تصویر را انجام نمی‌دهد. برای درک بهتر موضوع از تصویر زیر استفاده می‌کنیم:



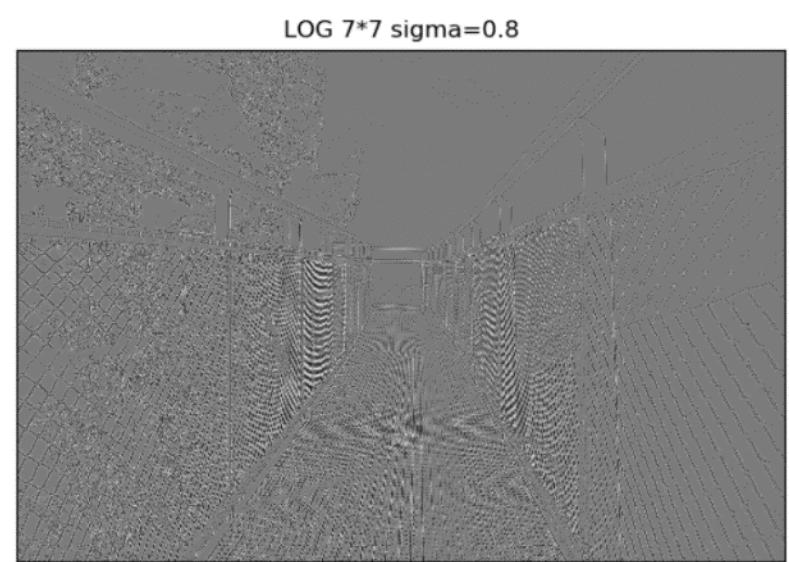
در صورتی که اندازه‌ی فیلتر کوچک باشد آن Side lobe مثبت درون فیلتر قرار نمی‌گیرد و فیلتر حالت استخراج لبه را از دست می‌دهد و مانند این می‌شود که یک مقدار از کل تصویر کم کرده باشیم.

فیلتر با سایز 5×5 , سیگما 0.8 :

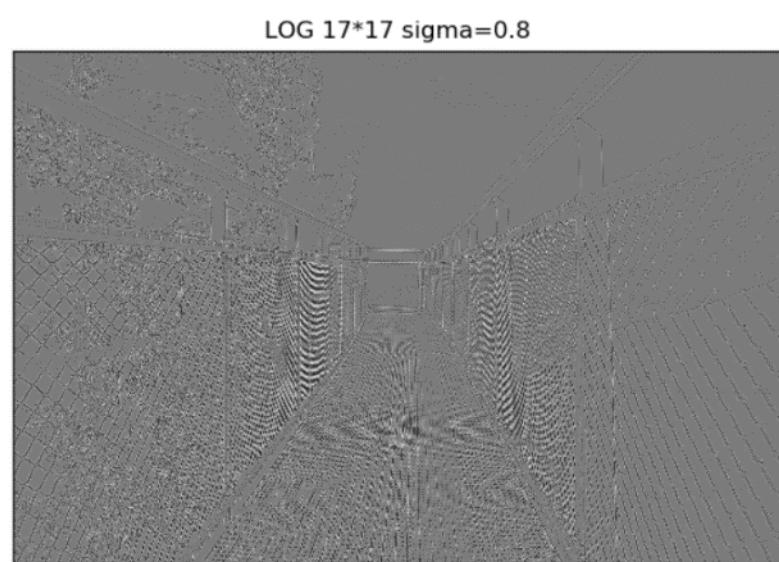


اکنون با بزرگ‌کردن فیلتر، side lobe های مثبت در فیلتر قرار می‌گیرند و فیلتر عملیات پیدا کردن لبه را به خوبی انجام می‌دهد.

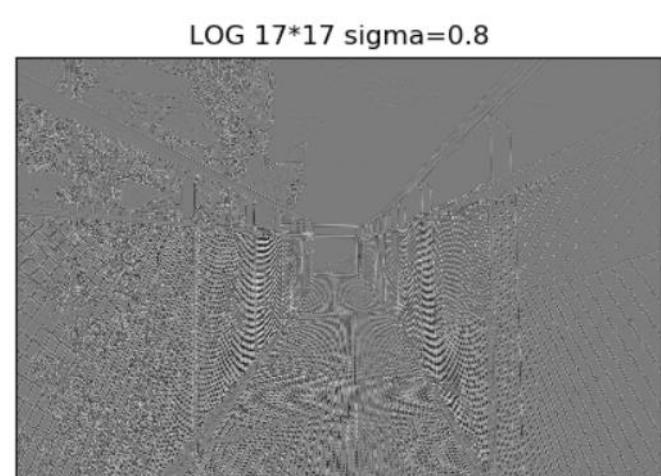
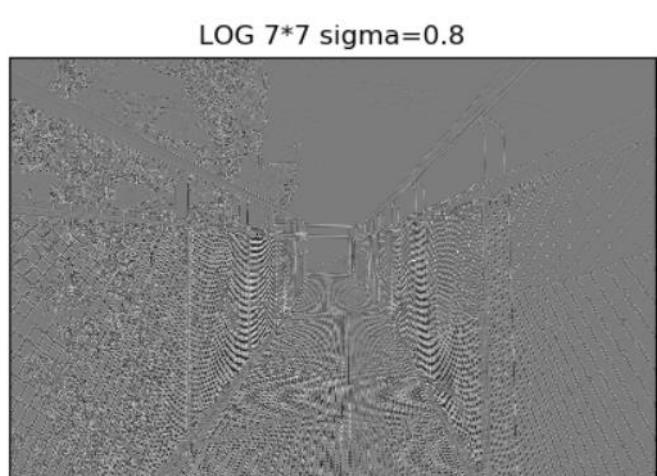
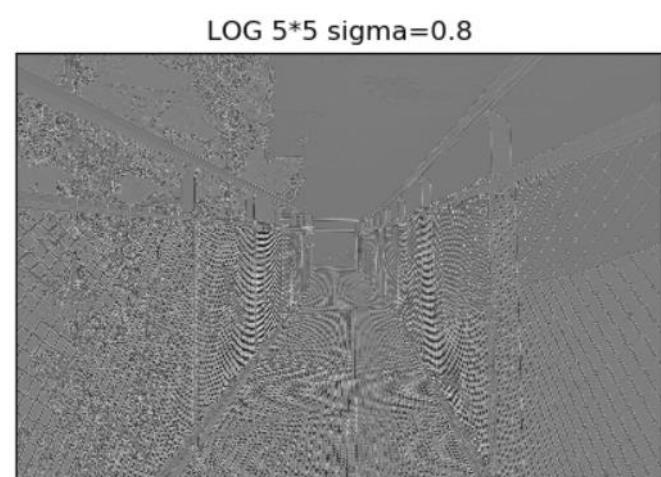
فیلتر با سایز 7×7 , سیگما 0.8 :



فیلتر با سایز 17×17 , سیگما 0.8 :

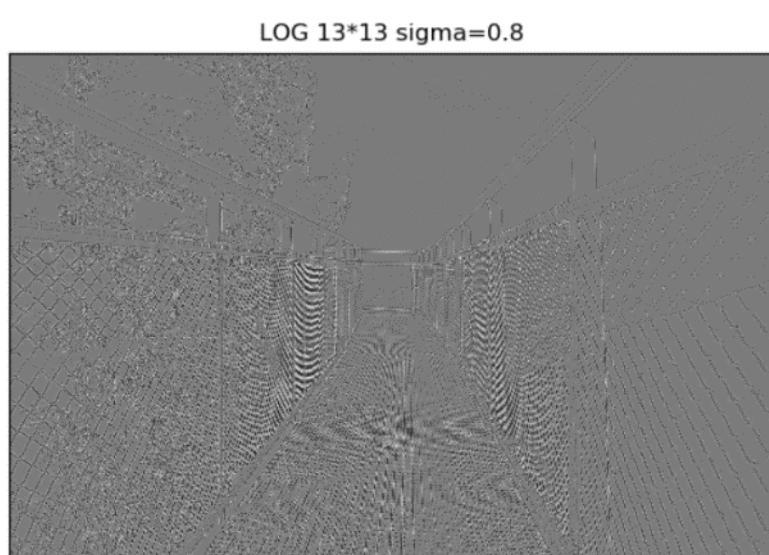


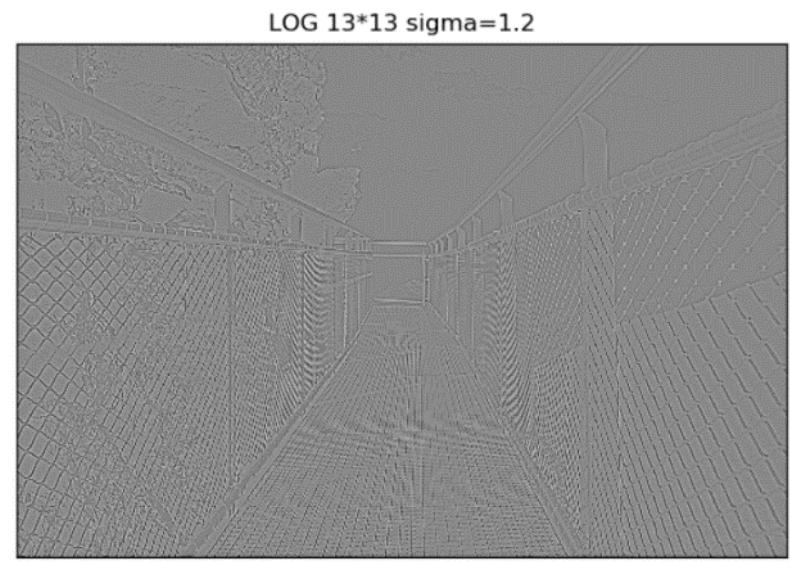
برای مقایسه بهتر، فیلترها با اندازه‌های مختلف و سیگماهای یکسان را در کنار هم قرار می‌دهیم:



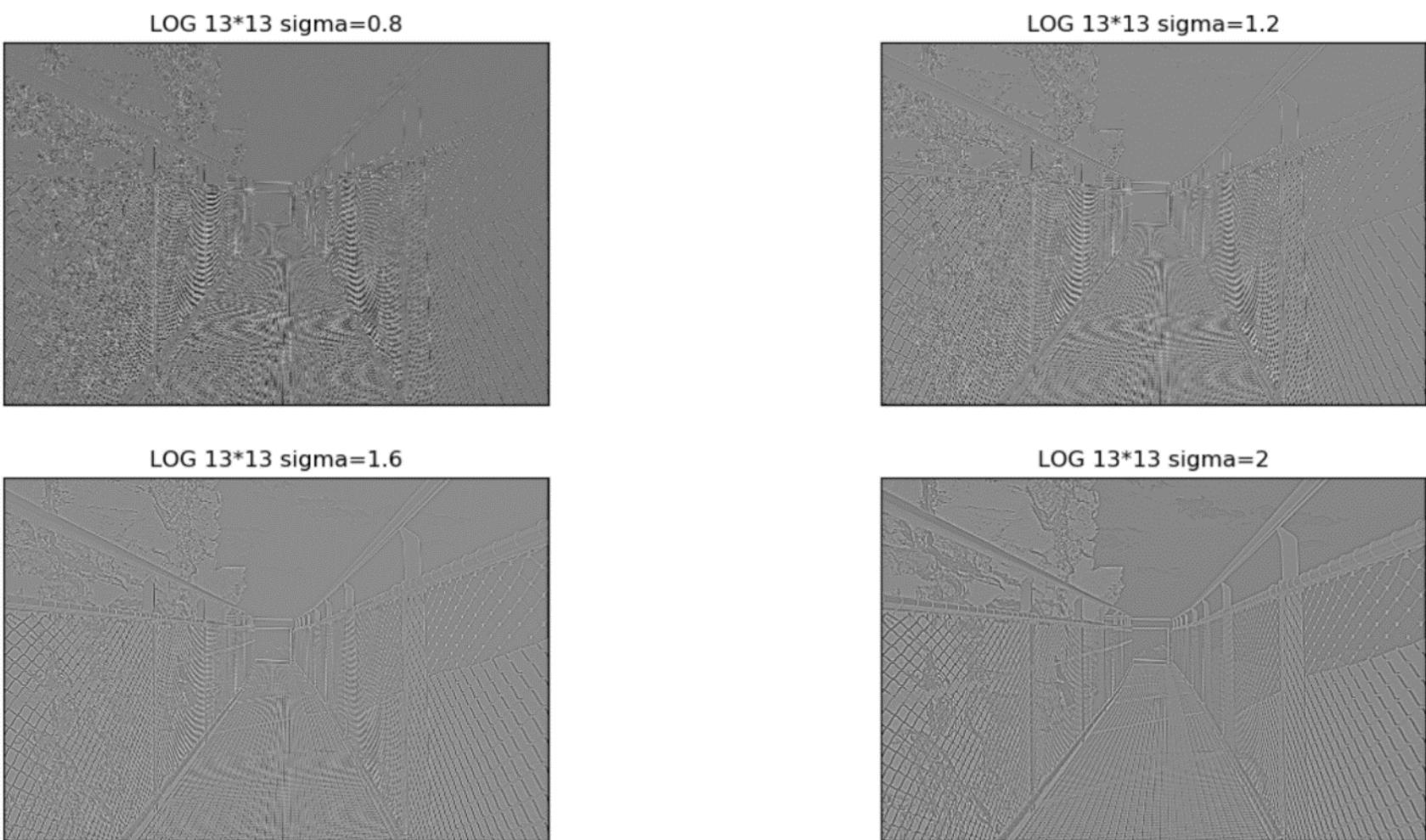
اکنون اپراتور LoG را با سیگماهای مختلف زیر
(sigma=0.8, size=13), (sigma=1.2, size=13), (sigma=1.6, size=13), (sigma=2, size=13)

مورد آزمایش قرار می‌دهیم.



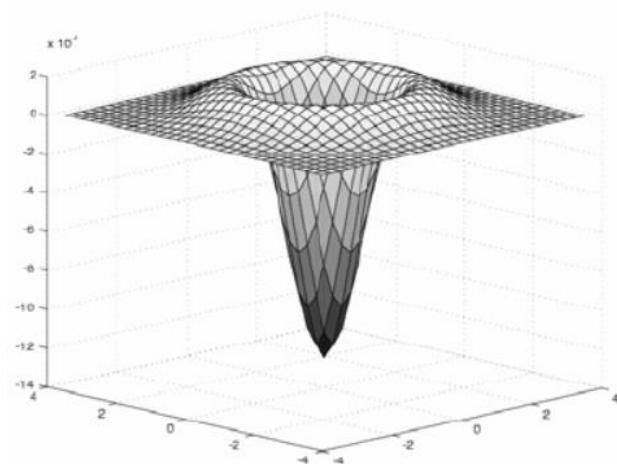


برای مقایسه بهتر ۴ خروجی حاصل از اعمال فیلترها با سیگماهای مختلف را کنار هم قرار می‌دهیم:



نتیجه‌گیری:

فیلتر LoG شکلی به صورت زیر دارد



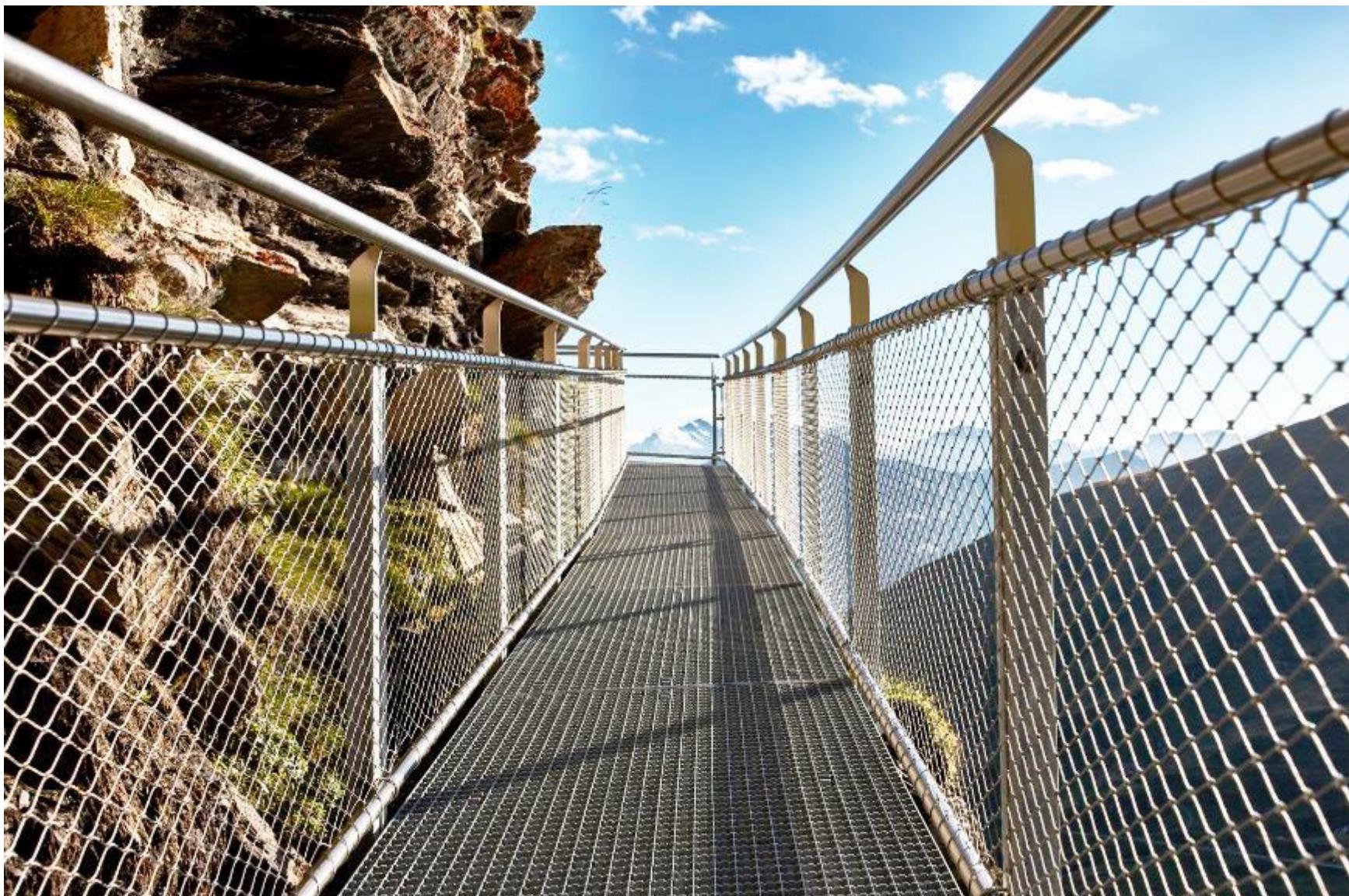
این فیلتر دو پارامتر n و σ را دارد که تغییرات آنها بر لبه‌ها را با آزمایش‌های مختلف که انجام دادیم، بررسی کردیم. برای بررسی تاثیر تغییر n بر لبه‌ها چهار فیلتر LoG با چهار n متفاوت را آزمودیم. در صورتی که n نسبت به Sigma کوچک باشد، در فیلتر فقط شاهد اعداد منفی خواهیم بود و Side Lobe مثبت در فیلتر قرار نخواهد گرفت، همین باعث می‌شود که فیلتر عمل لبه‌یابی را نتواند انجام دهد و مجموعه اعداد روی فیلتر یک عدد منفی باشد.

فیلتر با سایز بسیار کوچک مانند کم کردن یک عدد از تمام پیکسل‌ها عمل می‌کند. با زیاد شدن اندازه‌ی فیلتر از آن حد کم، Side Lobe‌های مثبت درون تصویر قرار گرفت و فیلتر به خوبی عمل لبه‌یابی را انجام داد. با دوباره زیاد کردن اندازه تصویر مقداری smooth شد زیرا پیکسل‌های بیشتری در همسایگی برای لبه‌یابی مورد استفاده قرار می‌گیرند. با زیاد کردن اندازه تصویر بیش از یک اندازه‌ای، تغییر خاصی در لبه‌ها ایجاد نمی‌شود زیرا بزرگی مقدار نقاط هر چه از مرکز دور می‌شوند به صورت غیر خطی کاهش می‌یابند و با زیاد کردن اندازه‌ی فیلتر، بزرگی مقدار نقطه‌های دور از فیلتر بسیار کوچک می‌شود.

پارامتر سیگما مقیاس لبه‌هایی که پیدا می‌شوند را نشان می‌دهد. زیرا لبه‌ها مستقل از مقیاس نیستند. هر چه سیگما زیادتر می‌شود، لبه‌ها و خط‌های پهن‌تری پیدا شده است. این موضوع به خوبی در نتایجی که آزمایش کردیم مشخص است. در سیگماهای کوچک، لبه‌های باریکی پیدا شده است هر چه سیگما بیشتر شده است، لبه‌های بزرگ‌تری پیدا شده است. با زیاد شدن سیگما حساسیت به نویز نیز کم شده است.

کد این قسمت در `problem-3.py` قرار دارد. در این سوال ابتدا تاثیر Low Threshold را بررسی می‌کنیم سپس تاثیر High Threshold را بررسی می‌کنیم و در ادامه تاثیر اندازه را بررسی می‌کنیم و بعد از آن مقایسه‌ای با سوبول و LOG خواهیم داشت و در آخر نتیجه‌گیری را مشاهده می‌کنیم.

تصویر ورودی سوال:



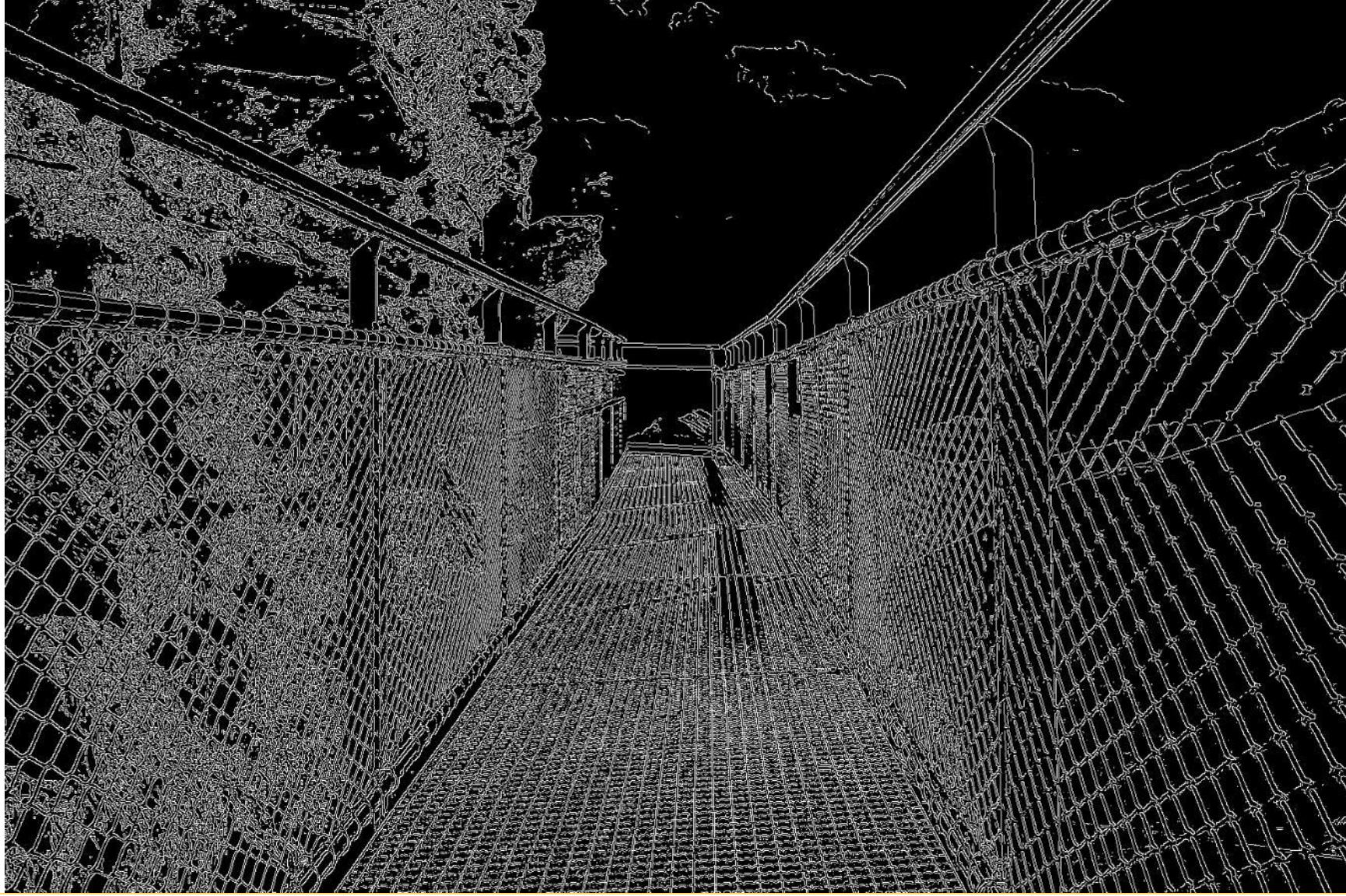
آزمایش‌هایی برای بررسی اثر تغییر Low Threshold:
برای بررسی تاثیر تغییر Low Threshold از چهار پارامتر مختلف 5,45,75,130 برای Low Threshold، به ازای High Threshold برابر با 150 و سایز 3 استفاده کرده‌ایم. در زیر نتایج را مشاهده می‌کنید:

Low Threshold





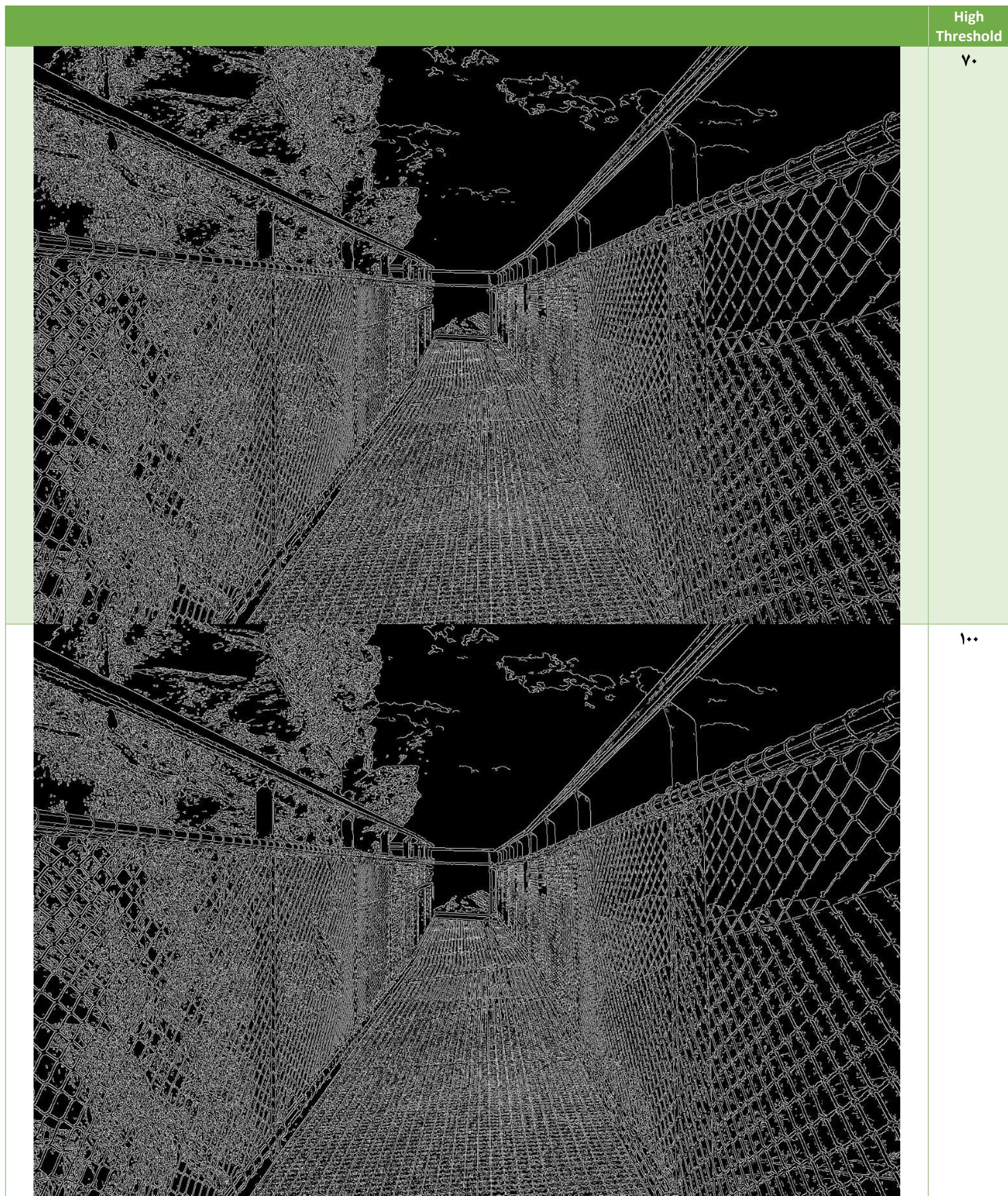
130

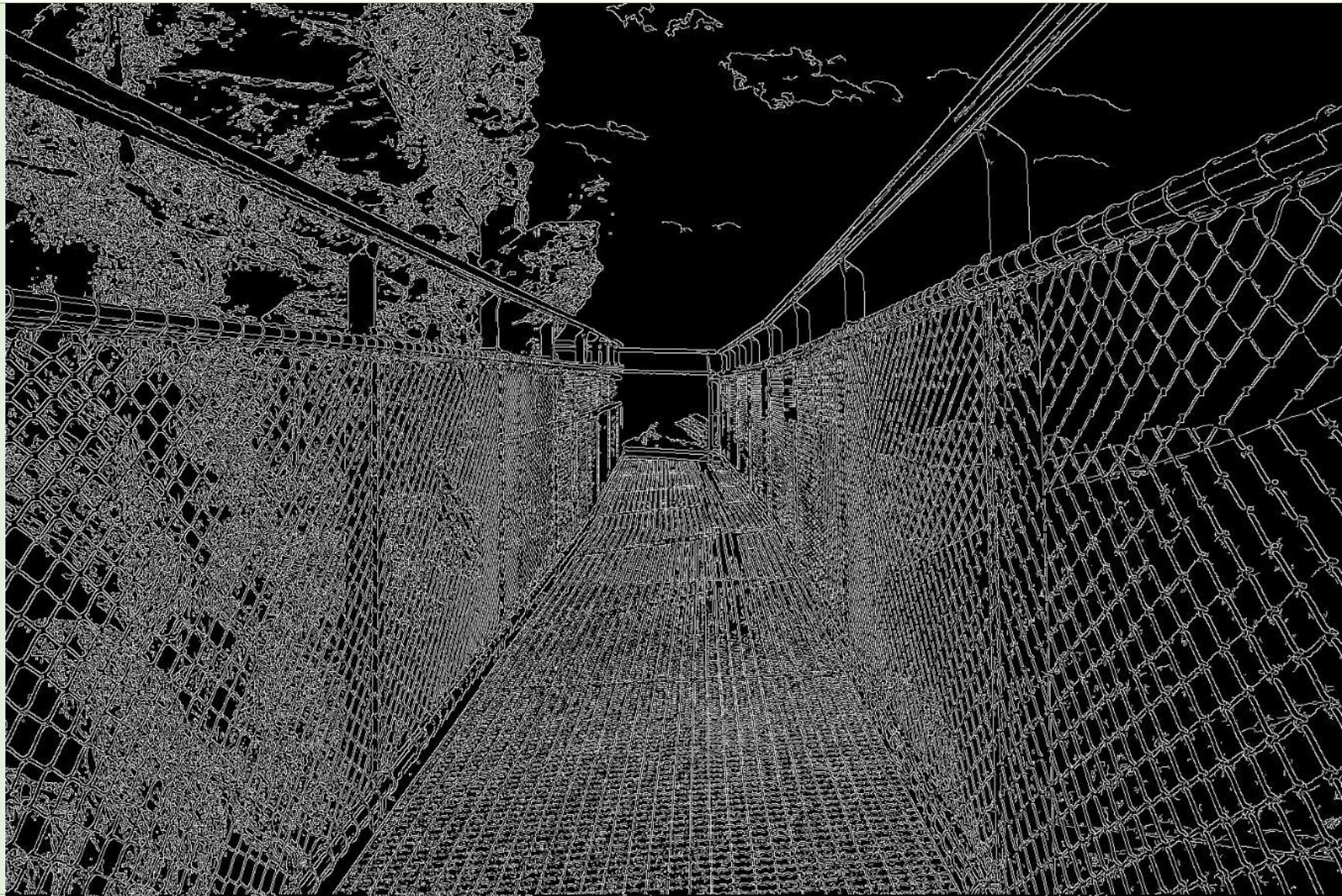


هرچه Low Threshold بیشتر شده است، جزئیات بیشتر از لبه‌ها حذف شده است این جزئیات حذف شده بیشتر مربوط به لبه‌ای کوچک و غیر اصلی هستند و خروجی کمتر شده است. این موضوع در تصویرهای بالا به خوبی ناحیه‌های ابر و لبه‌ای سمت راست و چپ قابل مشاهده است. البته این موضوع قابل پیش‌بینی بود زیرا Low Threshold سطح آستانه‌ی نقطه‌های کاندید ضعیف را نشان می‌دهد نقطه‌های بین آستانه‌ی Low و High ممکن است در خروجی نهایی باشد و نقطه‌های زیر آستانه Low حذف می‌شوند، هر چه این پارامتر بیشتر باشد باعث می‌شود نقطه‌های کمتری به عنوان کاندید نقاط ضعیف لبه داشته باشیم و نقاط بیشتری حذف شوند زیرا نقطه‌هایی که میزان Magnitude آن‌ها کمتر از Low Threshold باشند حذف می‌شوند.

آزمایش‌هایی برای بررسی اثر تغییر :High Threshold

برای بررسی تاثیر تغییر High Threshold از چهار پارامتر مختلف 70, 100, 150, 200 برای Low Threshold، به ازای High Threshold برای 50 و سایز 3 استفاده کردہ‌ایم. در زیر نتایج را مشاهد می‌کنید:

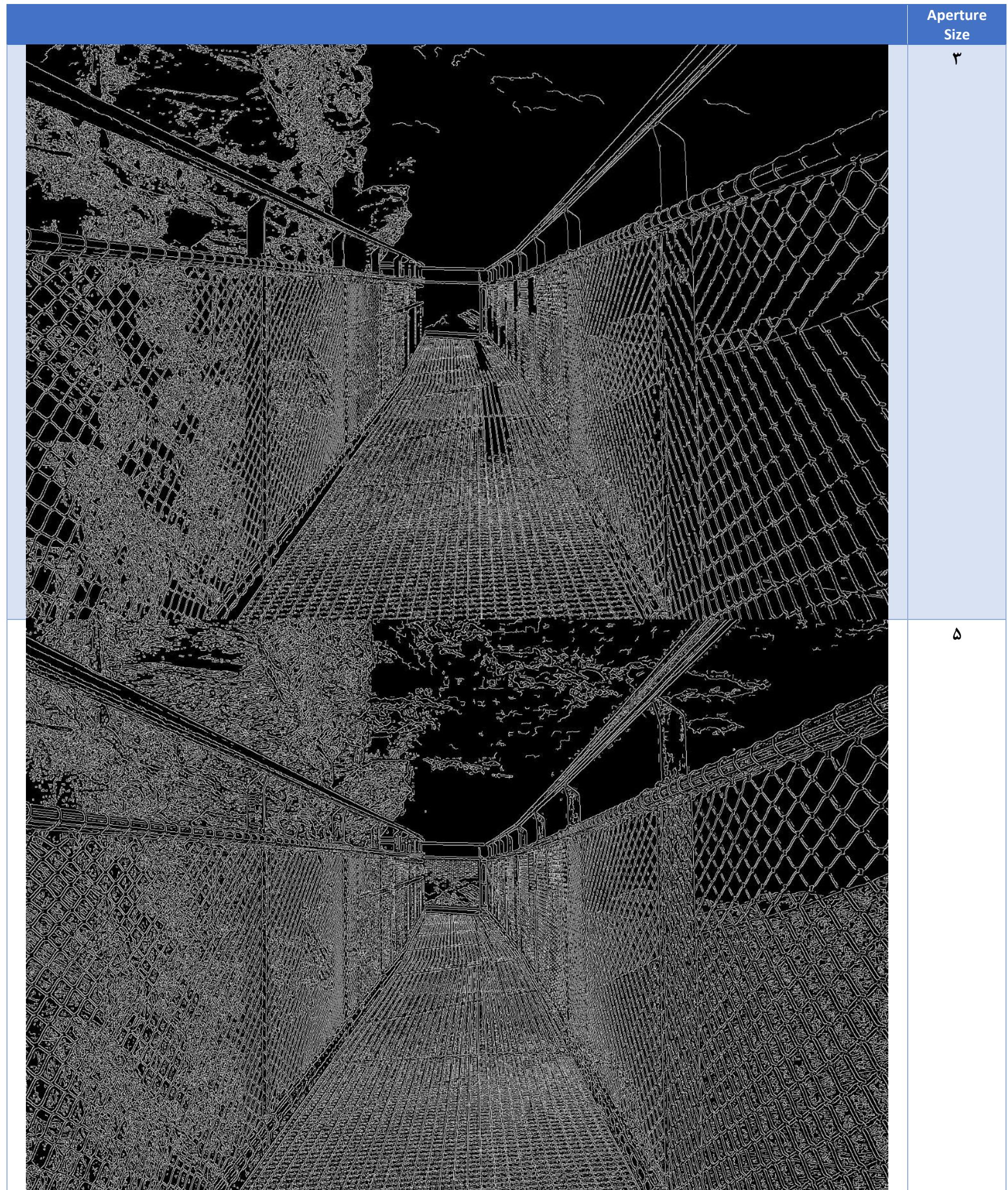




هرچه High Threshold بیشتر شده است، لبه‌های بیشتری از تصویر حذف شده است، لبه‌های حذف شده هم شامل لبه‌های اصلی و بزرگ می‌شود و هم لبه‌های کوچک و جزئی. البته این موضوع قابل پیش بینی است، زیرا نقطه‌ها با مقدار بیش از High Threshold به عنوان لبه در نظر گرفته می‌شوند و هر چه High Threshold بیشتر شود، نقطه‌های کمتری به عنوان لبه در نظر گرفته می‌شوند. همچنین نقطه‌هایی که مقدار آنها بین دو آستانه است در صورت قرار داشتن در همسایگی نقاطی که سطح آستانه آنها بیش از High Threshold است، لبه در نظر گرفته می‌شوند. با زیاد شدن آستانه بالا باعث می‌شود نقاط کمتری از بین نقاط بین دو آستانه به عنوان لبه در نظر گرفته شوند.

آزمایش‌هایی برای بررسی اثر تغییر aperture Size

برای بررسی تاثیر تغییر aperture Size از سه پارامتر مختلف ۳،۵ و ۷ برای apertureSize، به ازای Low Threshold برابر با ۱۰۰ و High Threshold برابر با ۲۰۰ استفاده کردہ‌ایم. در زیر نتایج را مشاهده می‌کنید:





هرچه Size Aperture افزایش پیدا کرده است، لبه‌های بیشتری به تصویر افزوده شده است که بیشتر این لبه‌ها جز لبه‌های جزئی هستند بسیاری از نقطه‌ها به عنوان لبه‌ی اشتباه تشخیص داده شده‌اند. البته این موضوع قابل پیش‌بینی بود زیرا روش Canny از مشتق اول در مراحل اولیه‌ی خوب استفاده می‌کند و همانطور که در سوال‌های قبل نشان دادیم هر چه اندازه‌ی پنجره بزرگ‌تر شود لبه‌ها پهن‌تر و بلورتر می‌شوند. همین طور در روش Canny تصویر در ابتدا بلور می‌شود، سایز بیش از اندازه‌ی پنجره باعث می‌شود که تشخیص لبه‌ها سخت شود.

مقایسه‌ی بین Canny و Sobel, Laplacian

برای مقایسه‌ی ۳ روش مختلف، لبه‌های به دست آمده از سه روش را با یکدیگر مقایسه می‌کنیم.

Type

Sobel
Kernel size
3



Laplacian
Kernel
Size 3



Canny
Low
Threshold
100,
High
Threshold
200, size 3



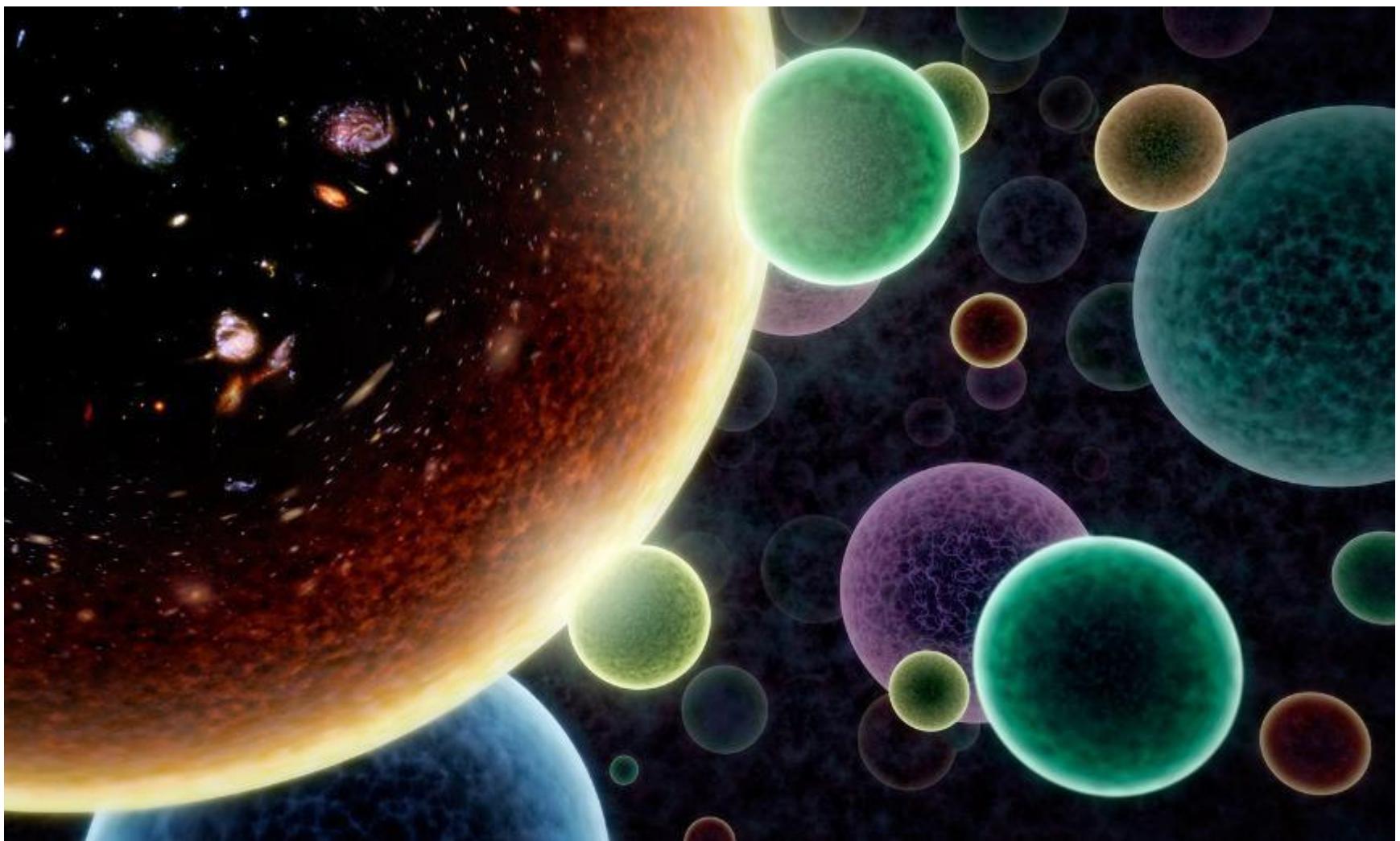
در مقایسه Canny با دو روش Sobel و Laplacian، یک بهبود چشم‌گیر در تشخیص جزئیات لبه‌های اصلی را مشاهده می‌کنیم. همچنین شاهد نادیده شدن بسیاری از پیکسل‌ها به عنوان لبه هستیم که در دو روش دیگر به اشتباه لبه تشخیص داده شده‌اند. کیفیت خط‌ها از جنبه‌های مختلفی همچون پیوستگی، باریکی و صاف بودن در Canny نسبت به دو روش دیگر برتر است.

در پایان هر بخش در این سوال در بالاتر، نتیجه‌گیری برای پارامتر مورد بحث ارائه شد.

تمرین ۴

کدهای این قسمت در `problem-4.py` قرار دارد.

تصویر ورودی:



ابتدا تصویر ورودی را می‌خوانیم، سپس با استفاده از روش Canny لبه‌ها را استخراج می‌کنیم و بعد در ادامه با استفاده از Thresholding مقدار شدت روش‌نایی لبه‌ها را برابر 0 و 255 می‌گذاریم. سپس یک accumulator cells با ابعاد $c_1 * c_2 * c_3$ می‌سازیم که همه‌ی مقادیر آن صفر است.

$$(x - c_1)^2 + (y - c_2)^2 = c_3^2$$

$C1$ و $C2$ مرکز دایره هستند و $C3$ شعاع دایره است. مقدار $C1$ بین صفر تا تعداد سطرهای تصویر است و مقدار مجاز $C2$ بین صفر تا تعداد ستون‌های تصویر است. $C3$ مقادیر بین حداقل شعاع دایره و حداقل شعاع دایره را داشته باشد.

سپس به ازای نقاط مختلف از پیکسل‌های تصویر لبه‌ها که روی لبه‌ها هستند و جز پس‌زمینه نیستند و مقادیر مختلف $C1$ و $C2$ از فرمول بالا استفاده می‌کنیم و مقدار $C3$ را پیدا می‌کنیم که عددی حقیقی است. سپس این عدد را گرد می‌کنیم و مقدار accumulator cell متاظر با c_1 و c_2 و c_3 را به علاوه‌ی یک می‌کنیم.

در انتها در ماتریس accumulator cell پیکسل‌هایی که بیشترین مقدارها را دارند را انتخاب می‌کنیم که دایره‌ها را به ما می‌دهند.

این روش از کتاب گنزالس برداشته شده است، ویرایش ۳ صفحه‌ی ۷۳۶:

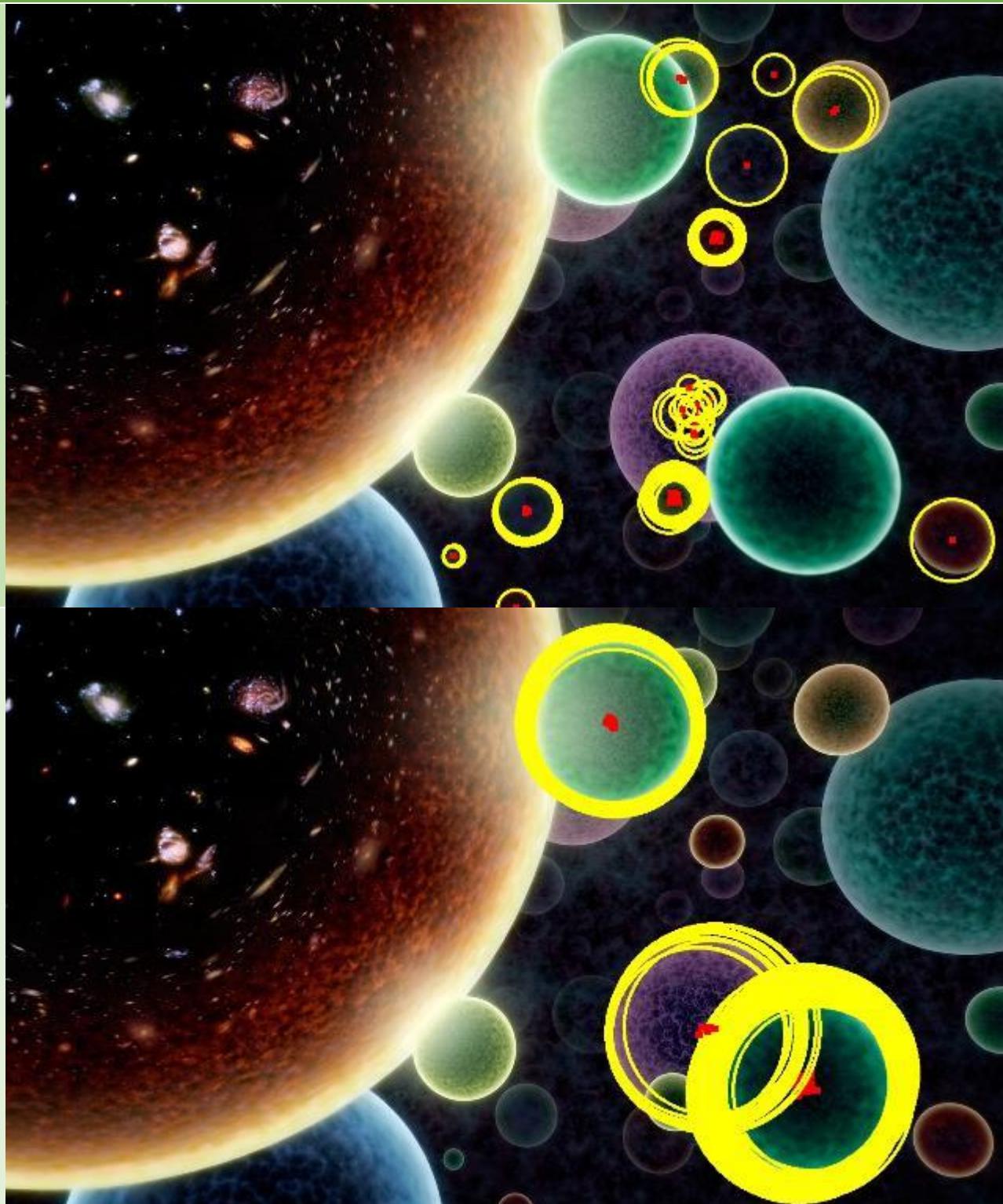
Although the focus thus far has been on straight lines, the Hough transform is applicable to any function of the form $g(\mathbf{v}, \mathbf{c}) = 0$, where \mathbf{v} is a vector of coordinates and \mathbf{c} is a vector of coefficients. For example, points lying on the circle

$$(x - c_1)^2 + (y - c_2)^2 = c_3^2 \quad (10.2-39)$$

can be detected by using the basic approach just discussed. The difference is the presence of three parameters (c_1 , c_2 , and c_3), which results in a 3-D parameter space with cube-like cells and accumulators of the form $A(i, j, k)$. The procedure is to increment c_1 and c_2 , solve for the c_3 that satisfies Eq. (10.2-39), and update the accumulator cell associated with the triplet (c_1, c_2, c_3) . Clearly, the complexity of the Hough transform depends on the number of coordinates and coefficients in a given functional representation. Further generalizations of the Hough transform to detect curves with no simple analytic representations are possible, as is the application of the transform to gray-scale images. Several references dealing with these extensions are included at the end of this chapter.

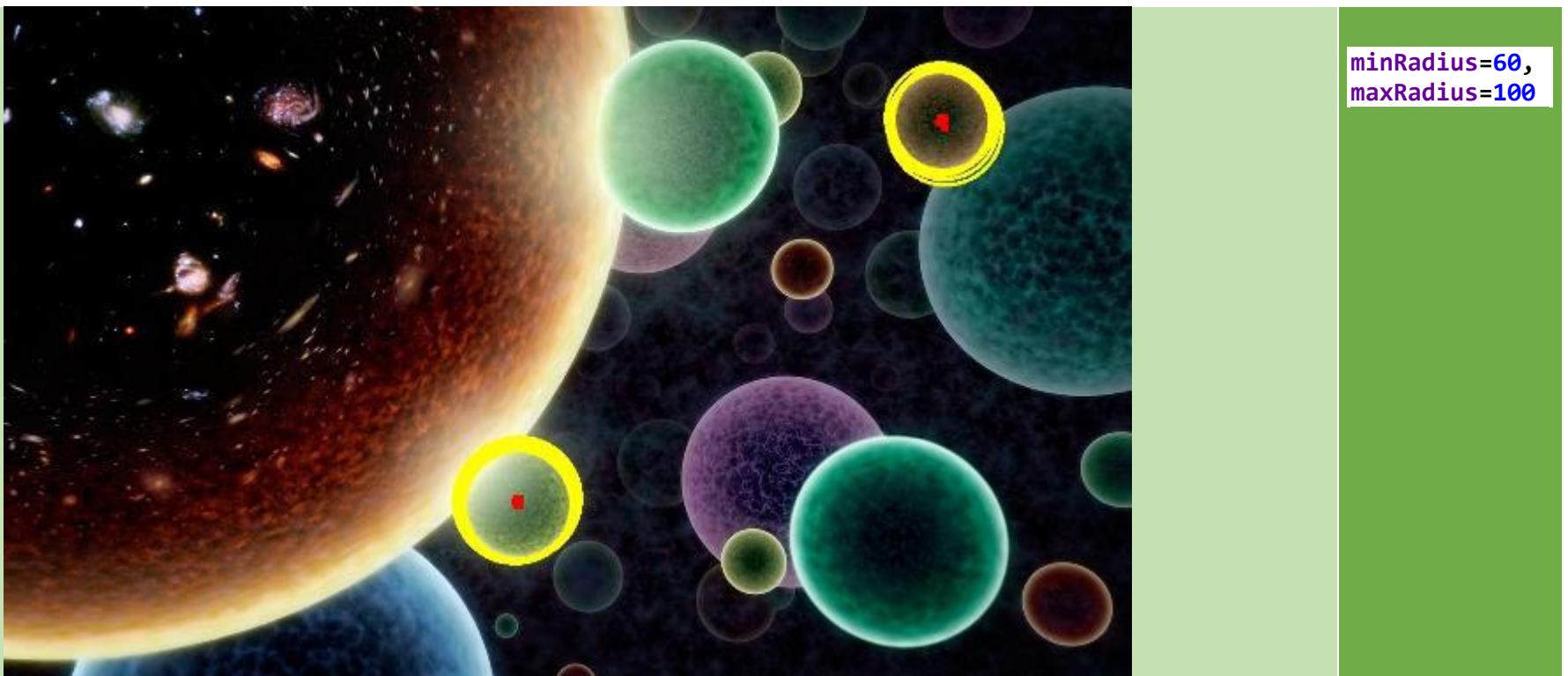
We return now to the edge-linking problem. An approach based on the Hough transform is as follows:

1. Obtain a *binary* edge image using any of the techniques discussed earlier in this section.
2. Specify subdivisions in the $\rho\theta$ -plane.
3. Examine the counts of the accumulator cells for high pixel concentrations.
4. Examine the relationship (principally for continuity) between pixels in a chosen cell.



`minRadius=10,
maxRadius=60`

`minRadius=100,
maxRadius=200`



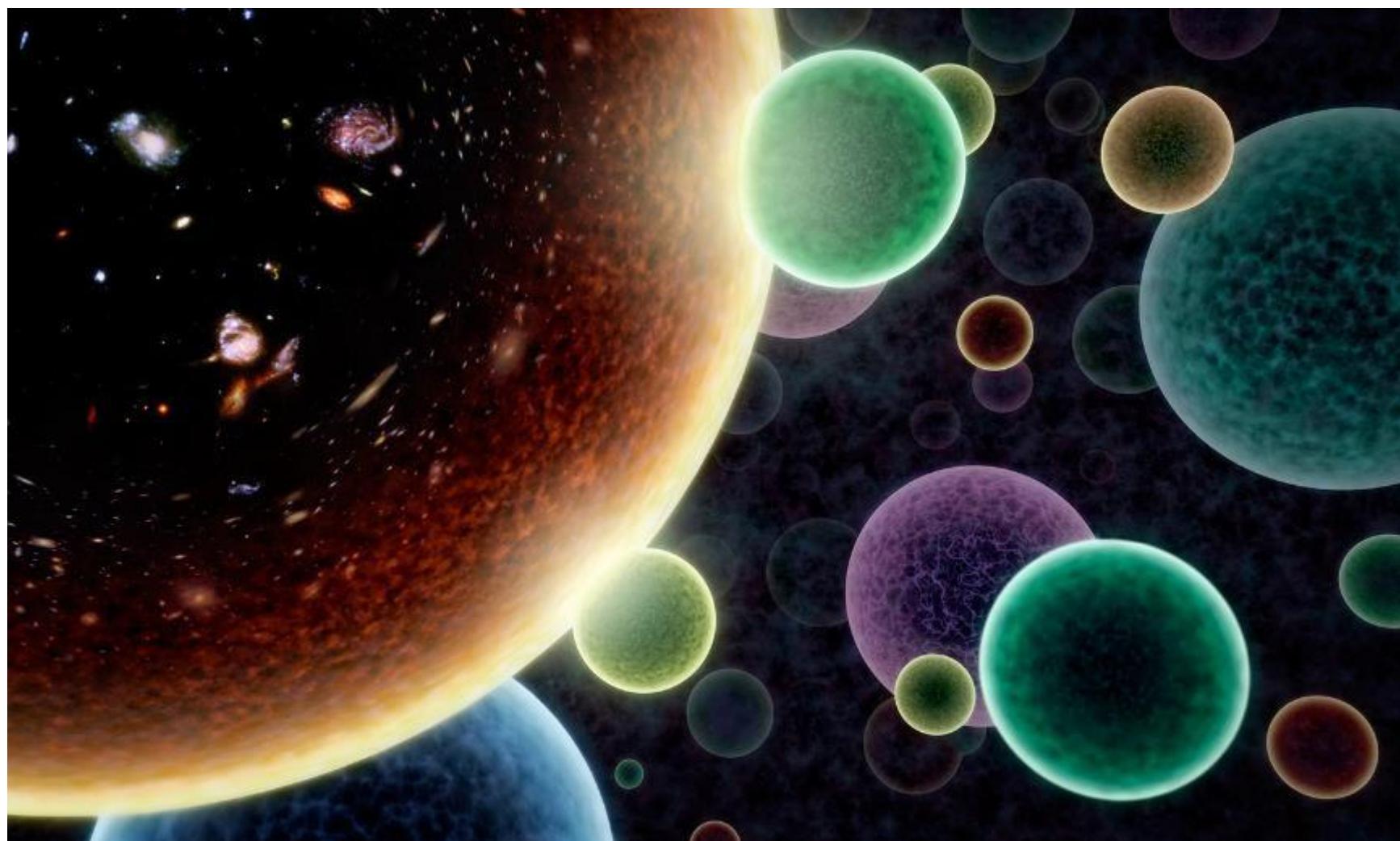
minRadius=60,
maxRadius=100

میزان `minRadius` و `maxRadius` بزرگی شعاع دایره را تعیین می‌کنند. هر چه `minRadius` افزایش یافته است دایره‌ها کوچکتر حذف شده‌اند. هر چه `maxRadius` افزایش یافته است، دایره‌های بزرگتری کشف شده است.

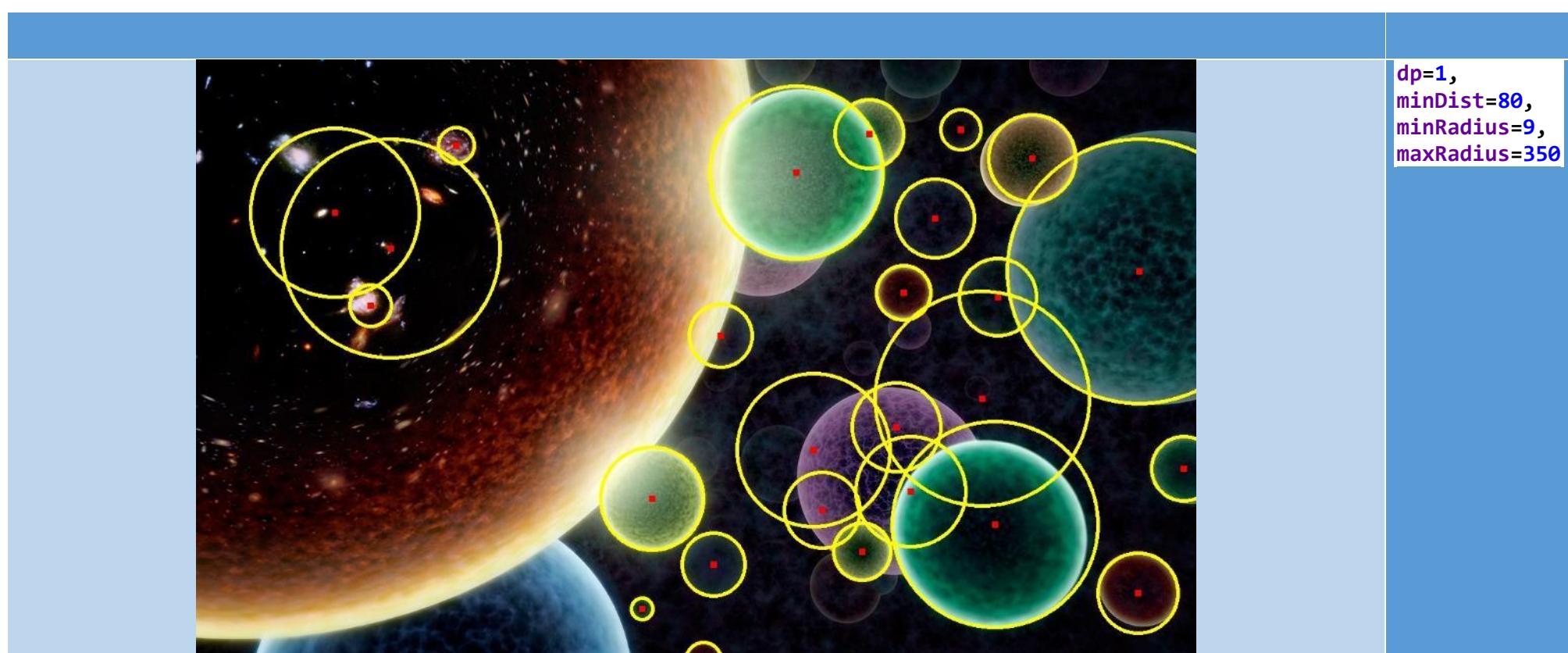
تمرین ۵

کد این قسمت در `problem-5.py` قرار دارد. در این قسمت با استفاده از توابع `openCV`, دایره‌های موجود در تصویر را پیدا کردہ‌ایم. در زیر خروجی‌های تصویر را مشاهده می‌کنید:

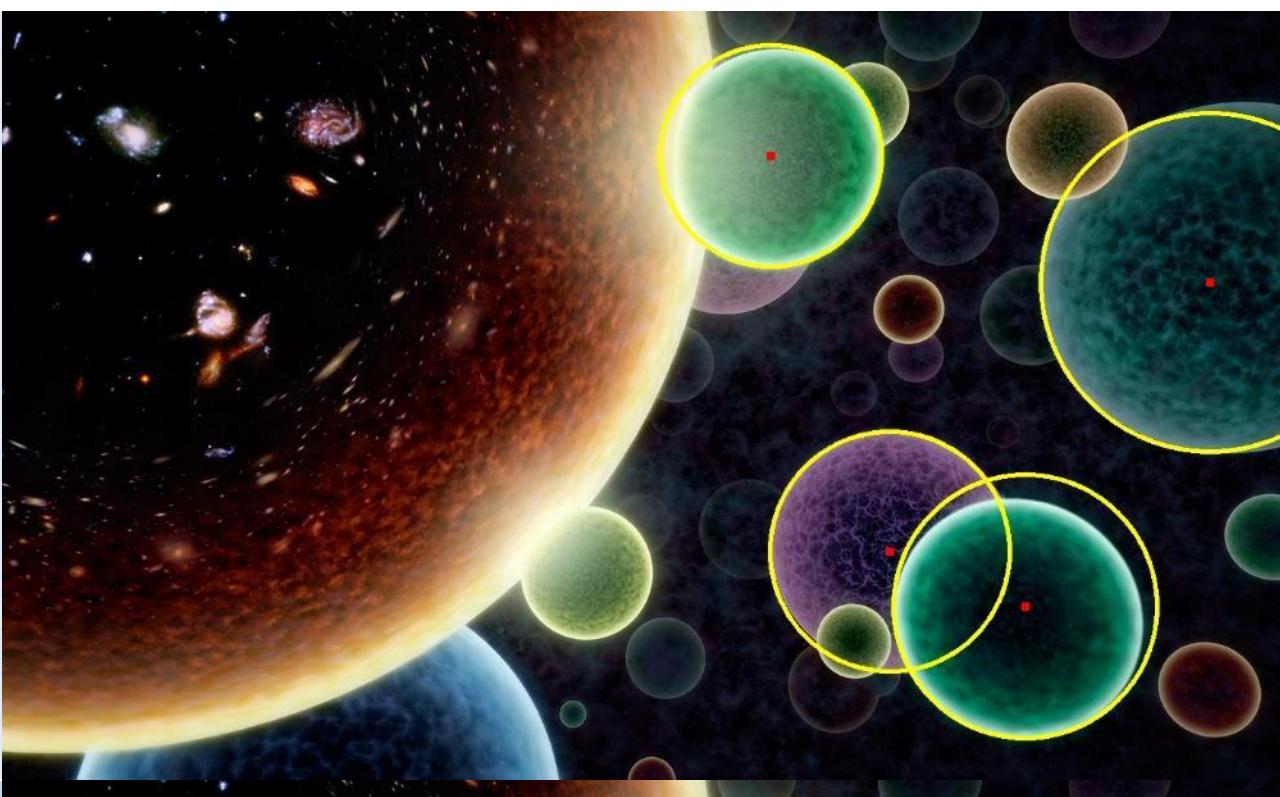
تصویر ورودی:



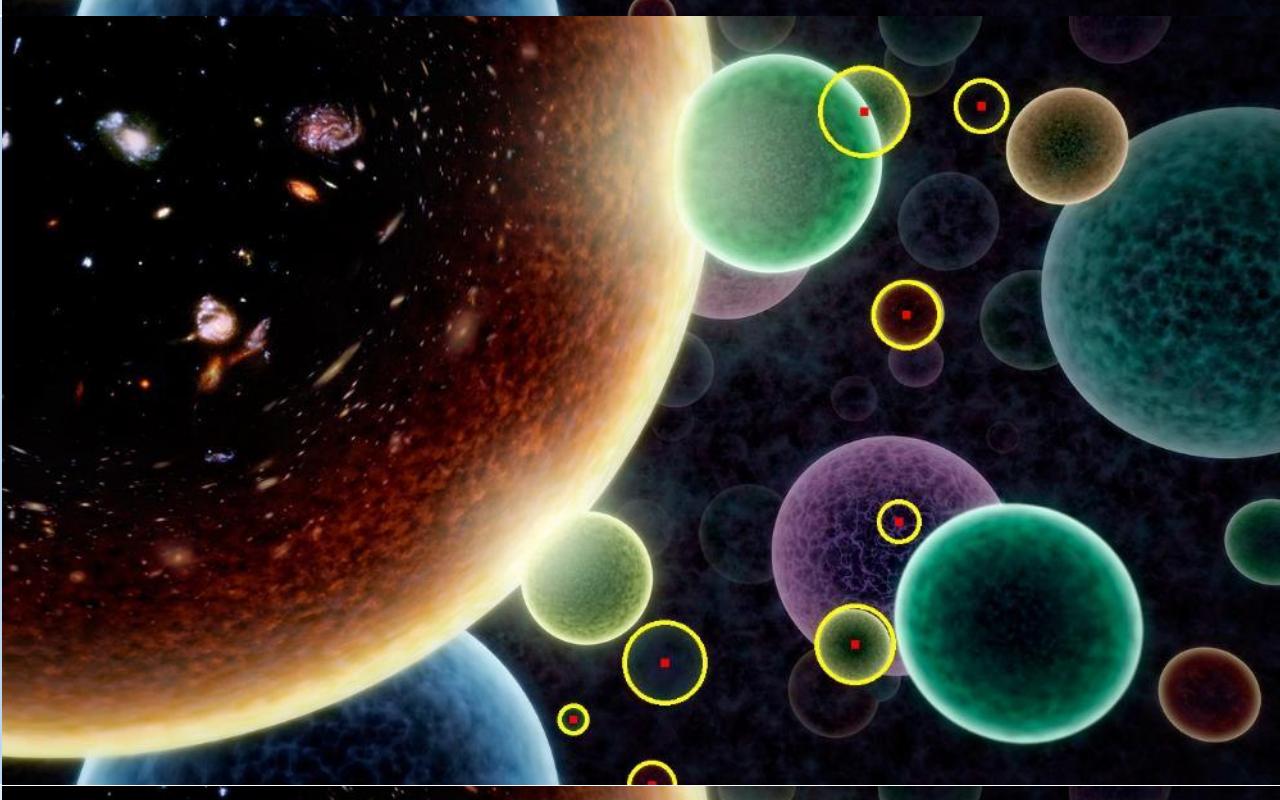
با پارامترهای مختلفی اقدام به استخراج دایره‌ها با سایزهای مختلف کردیم:



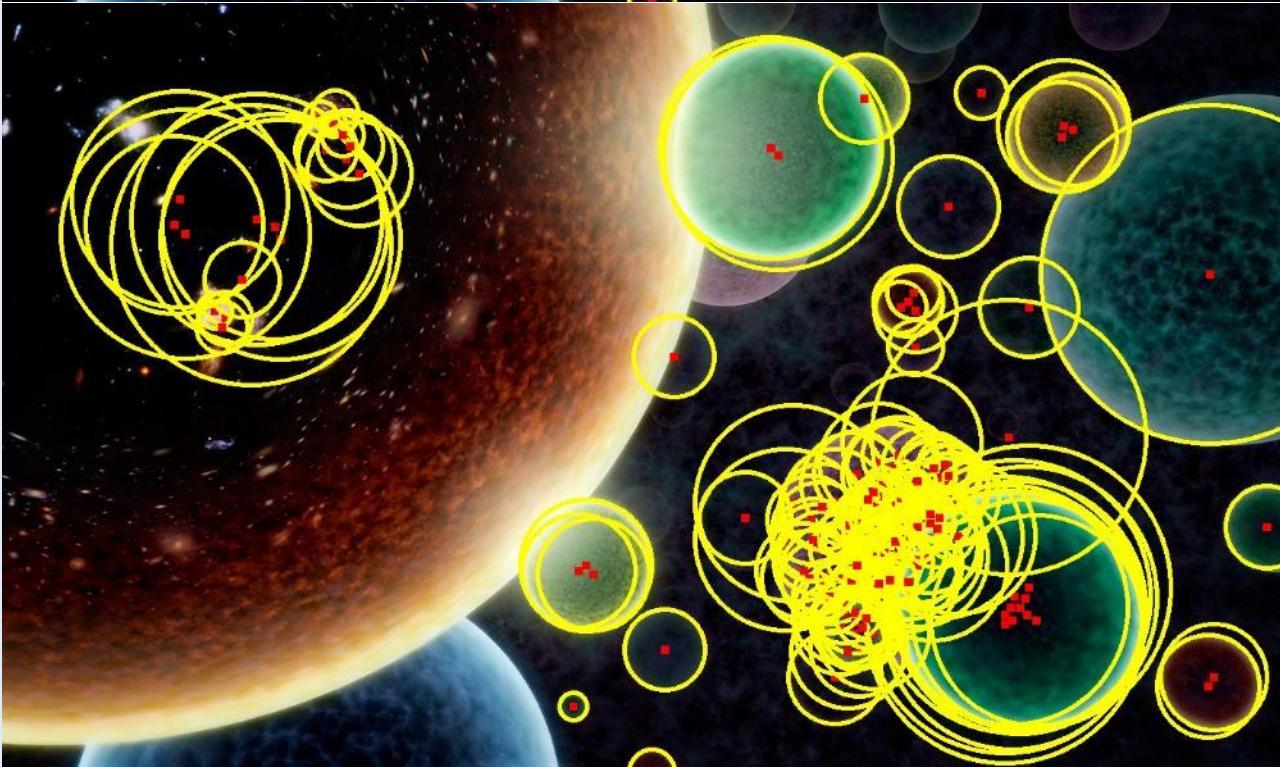
1, 80, 100, 350

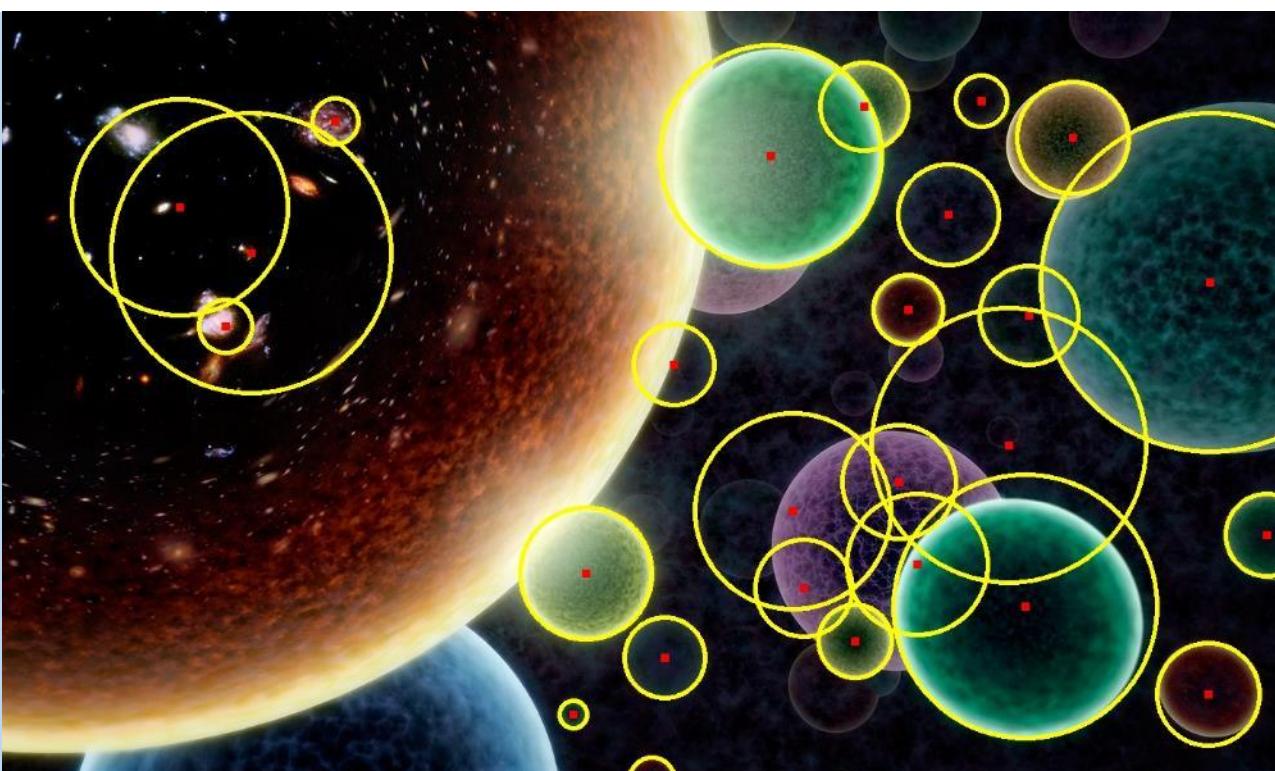


1, 80, 9, 50



1, 10, 9, 350





پارامتر `dp`, تعداد سلول‌های ماریس `accumulator cells` را مشخص می‌کند که مرکز دایره‌های مختلف حداقل باید به چه میزان از هم دور باشند. `minRadius` مشخص می‌کند که حداقل شعاع دایره‌ها چه میزان است و `maxRadius` تعیین می‌کند حداقل شعاع دایره‌ها چه میزان است.

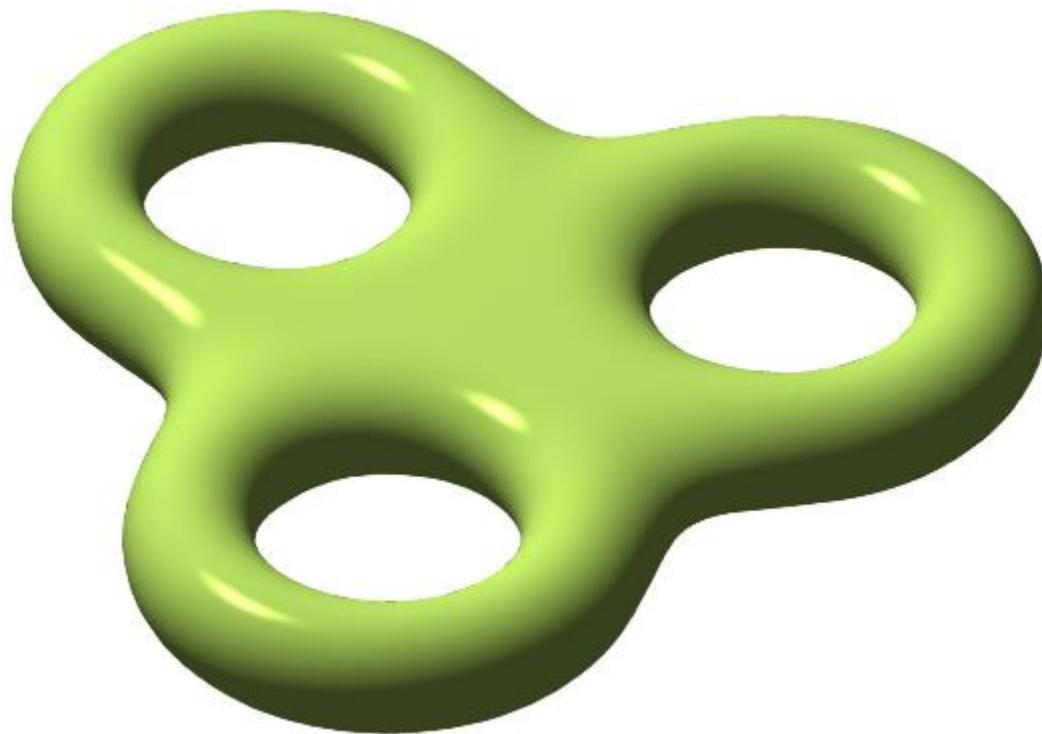
با زیاد شدن `minDist` دایره‌ها زیاد شده اند زیرا دایره‌ها اجازه دارند نزدیک‌تر نسبت به هم قرار بگیرند. با زیاد شدن `minRadius` شاهد وجود دایره‌های بزرگ‌تری در تصویر هستیم زیرا دایره‌های کوچک‌تر حذف می‌شوند. با کم شدن `maxRadius` شاهد از بین رفتن دایره‌ها با شعاع بزرگ هستیم و دایره‌ها موجود کوچک می‌شوند.

در مقایسه با الگوریتم پیاده سازی شده تابع `openCV` بسیار سریع‌تر عمل می‌کند و امکان `minDist` کنترل خوبی برای کنترل کیفیت دایره‌های پیدا شده می‌دهد به همین دلیل شاهد دایره‌های متعدد مرکز و شبیه بهم نیستیم. همچین تابع `openCV` از دقت بیشتری برخوردار است. همچنین این تابع آماده نیازی به تعیین حد آستانه برای زیادتر بودن مقدار `accumulator cells` از حد مشخصی که در نتایج نهایی باشد ندارد.

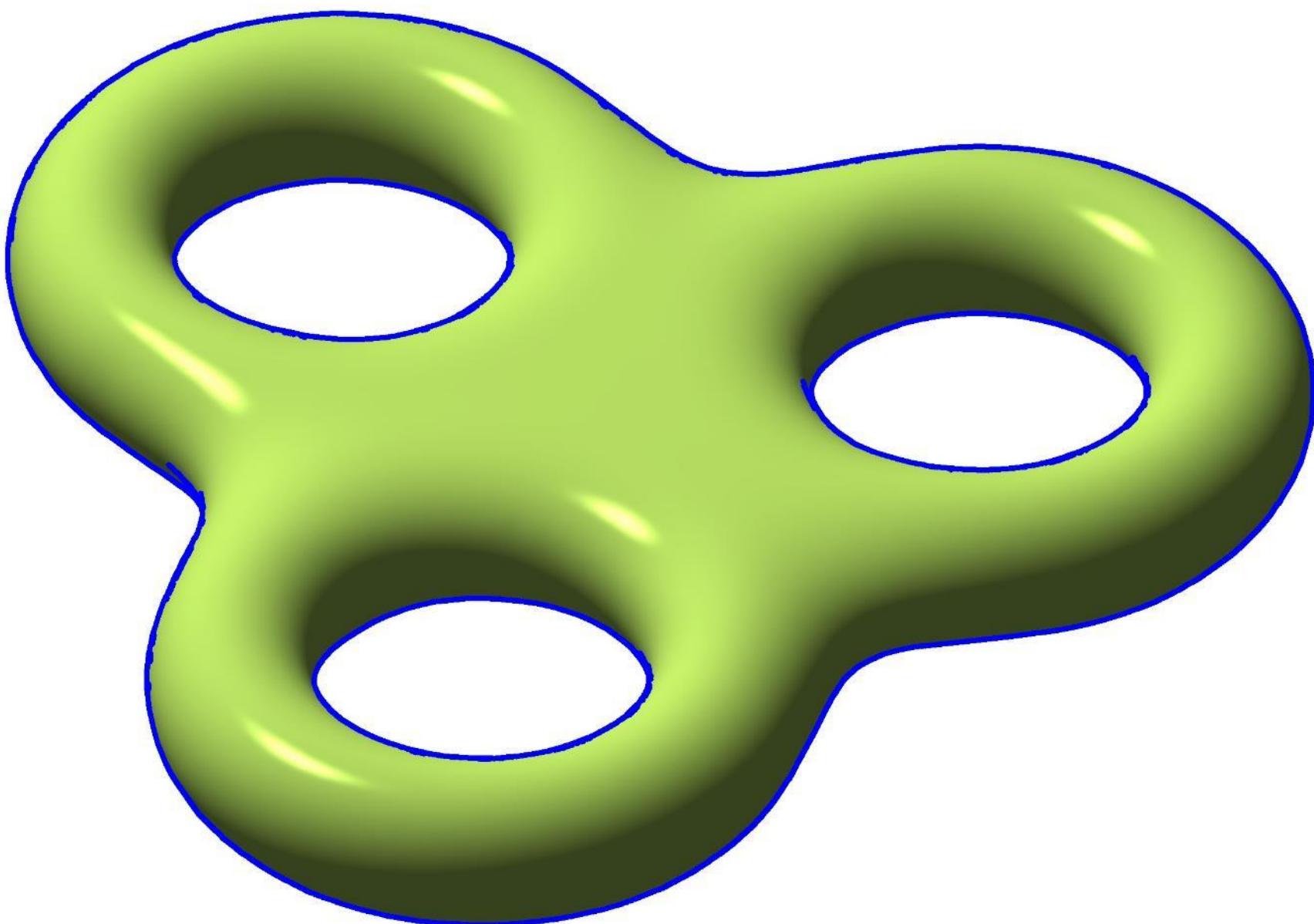
تمرین ۶

کد این سوال در `problem-6.py` قرار دارد. در ابتدا تصویر ورودی خوانده شده است سپس به صورت سیاه و سفید در آمده است و در ادامه با استفاده از روش Canny لبه‌ها استخراج شده‌اند و سپس تمام Active Contour استخراج و سپس رسم شده‌اند. در زیر خروجی این کد را مشاهد می‌کنید:

تصویر ورودی:

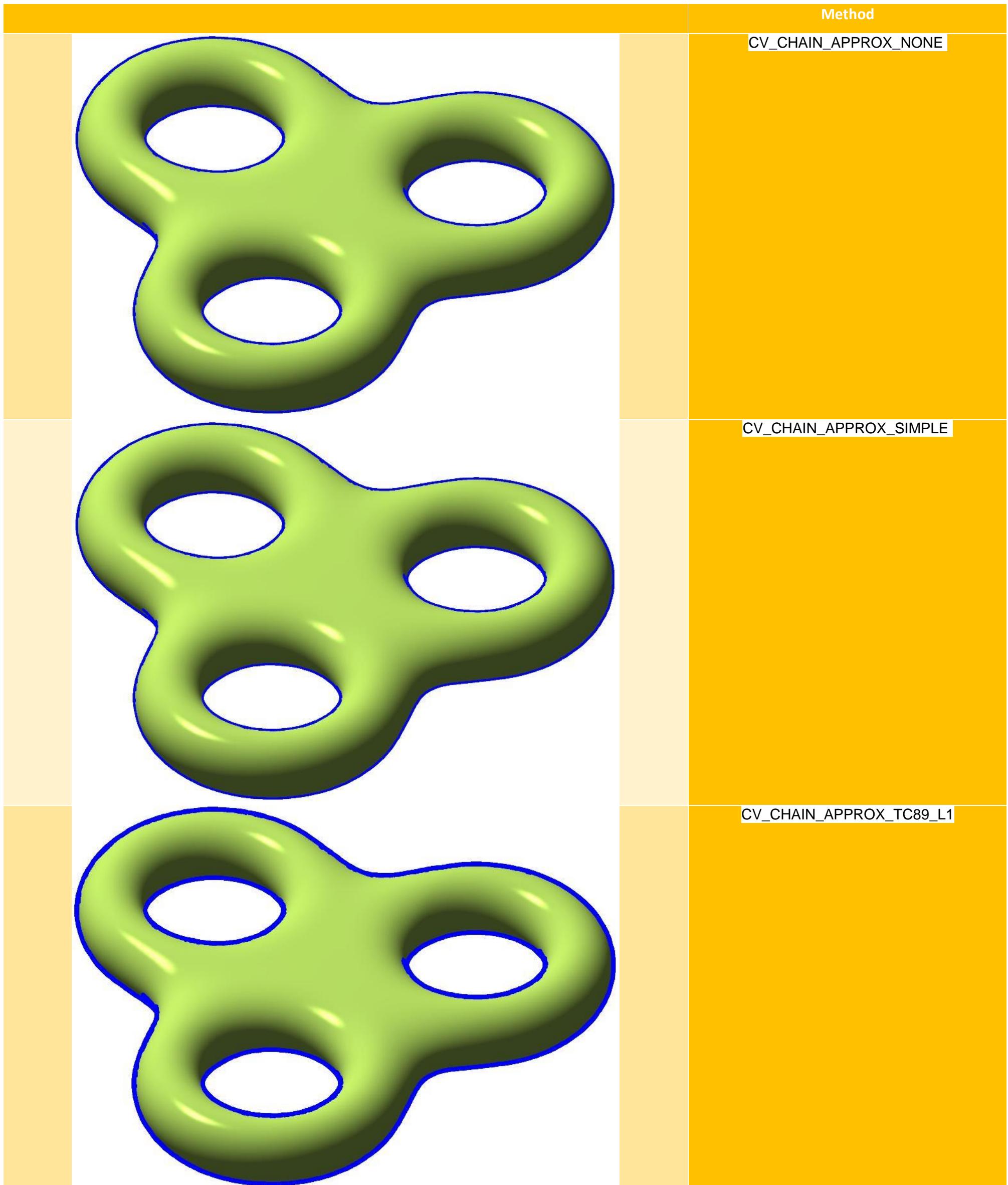


تمام های موجود در تصویر با رنگ آبی مشخص شده است:



همین طور که مشاهده می‌شود تابع OpenCV به خوبی توانسته است تمام کانتورهای شی را پیدا کند.

بررسی متدهای مختلف بر کانتورهایی که پیدا می‌شود:



به دلیل سادگی شکل، هر سه متد بسیار خوب و شبیه به هم عمل کردند، **CV_CHAIN_APPROX_NONE** نسبت به **CV_CHAIN_APPROX_SIMPLE** نقاط کمتری دارد زیرا در این روش نقاط در امتداد هم با یک دیگر ترکیب می‌شوند، منجر به کانتورهای ضخیم‌تری شده است و همچنین باز تعداد نقاط کانتور کاهش یافته است. در زیر تعداد نقاط روی کانتور روش‌های مختلف را مشاهده می‌کنید:

```

C:\Users\Home\AppData\Local\Programs\
segments= CHAIN_APPROX_SIMPLE 10121
segments CHAIN_APPROX_NONE= 19966
segments CHAIN_APPROX_TC89_L1= 7750

```