دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

دانشگاه صنعتی امیرکبیر
دانشکده مهندسی مکانیک

گزارش تشریحی **تمرین سری پنجم**

درس پردازش تصاویر دیجیتال

استاد: دکتر آذرنوش

دانشجو: محمدجواد زلقی

شماره دانشجویی: ۹۸۱۲۶۰۷۹

تاریخ: ۱۳۹۹/۱۰/۲۵

پاسخ سوال اول)

برای حل سوال، ابتدا تابعی برای اینکه نقاط ویژگی مشخص در دو تصویر توسط کاربر انتخاب شود، توسعه داده شده است. تابع `draw_circle()` ایونت دابل کلیک روی پنجره تصویر را ثبت می‌کند و بصورت لحظه‌ای مختصات را در یک آرایه ذخیره می‌کند. دستورالعمل نحوه انتخاب نقاط نیز توسط یک راهنما در ابتدای برنامه بیان شده است. پس از اینکه نقاط شاخص در دو تصویر مشخص شد، طبق مرجع برای حل مساله داریم:

$$
\begin{aligned}
x'_i &= a_{11}\,x_i + a_{12}\,y_i + a_{13} \\
y'_i &= a_{21}\,x_i + a_{22}\,y_i + a_{23}
\end{aligned}
$$

$$
\begin{bmatrix} x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix}
\begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \end{bmatrix} =
\begin{bmatrix} x'_1 \\ \vdots \\ x'_n \end{bmatrix}
$$

$$
\begin{bmatrix} x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix}
\begin{bmatrix} a_{21} \\ a_{22} \\ a_{23} \end{bmatrix} =
\begin{bmatrix} y'_1 \\ \vdots \\ y'_n \end{bmatrix}
$$

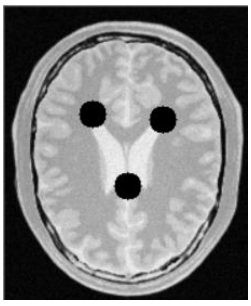
پس کافی است ماتریس A را تشکیل دهیم و از آن معکوس بگیریم. توجه داریم چون شش پارامتر مجهول می‌باشد، پس به سه نقطه لازم داریم اما اگر نقاط بیشتری توسط کاربر انتخاب شود، با مساله سودو اینورس که به نوعی اینترپولیشن بین تمام پاسخ‌ها است، دست پیدا می‌کنیم. پس برای پاسخ نهایی داریم:

```
a_1 = np.dot(np.linalg.pinv(A), xp)
a_2 = np.dot(np.linalg.pinv(A), yp)
```

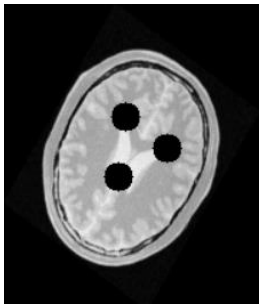
که در واقع A همان $\begin{bmatrix} x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix}$ ماتریس می‌باشد.

برای حالت سه نقطه برنامه را تست می‌کنیم. توجه داریم دایره مشکی اطراف مکان نقطه شاخص ترسیم می‌شود. همچنین در پیش پردازش ابعاد دو تصویر را یکسان می‌کنیم تا کار استاندارد باشد.

نقاط شاخص تصویر قبل از تبدیل



نقاط شاخص بعد از تبدیل

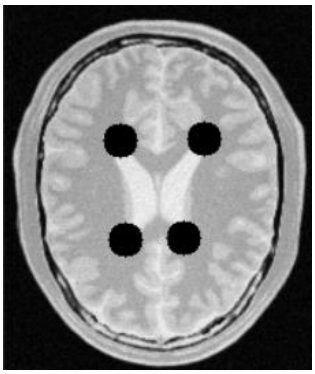


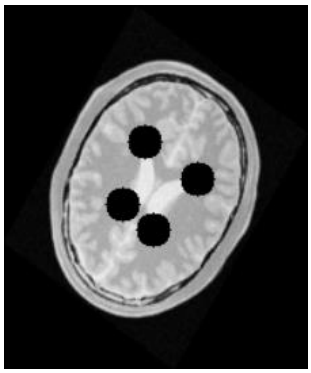
تبدیل افاین

```
afine transformation matrix is:
 [0.60037807 0.39243856 8.83931947]
 [-0.41814745  0.56294896 83.71266541]
 [0, 0, 1]
```

برای حالت ۴ نقطه داریم:

نقاط شاخص تصویر قبل از تبدیل نقاط شاخص بعد از تبدیل تبدیل افاین





```
afine transformation matrix is:
 [0.62868459 0.41189335 1.48371529]
 [-0.32570989  0.56193182 75.51800259]
 [0, 0, 1]
```

مشاهده نیز می‌شود خروجی دو حالت نزدیک می‌باشد.

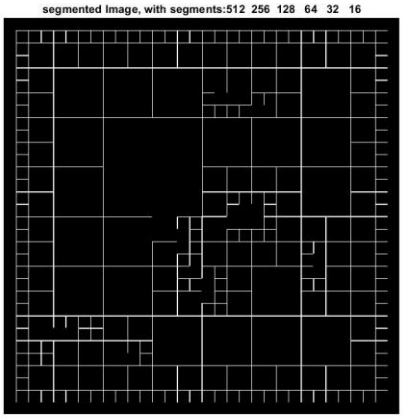
پاسخ سوال دوم)

الگوریتم اسپلیت اند مرج بر این اساس می‌باشد که ابتدا بر اساس معیار عدم شباهت تصویر به ۴ قسمت تقسیم می‌شود و هر کدام از ۴ قسمت نیز می‌تواند بهمین ترتیب تقسیم گردد. وقتی دیگر تقسیمی میسر نبود، سراغ بررسی قابلیت مرج کردن قسمت‌های مجزا با بررسی معیار شباهت قسمت‌های مرزی رفته می‌شود. پیاده‌سازی این تابع در پایتون بصورت بازگشتی با باگ‌هایی مواجه شد که متوجه ایراد در ابعاد بخاطر بازگشتی بودن نمی‌شدم . بهمین دلیل و البته وجود تابع Quadtree decomposition در متلب سراغ پیاده‌سازی در متلب رفتم. این تابع `qtdecomp(I, threshold, [mindim maxdim])` همان کار اسپلیت کردن را انجام می‌دهد. معیار عدم شباهت هم گذر اختلاف شدت بیشینه و کمینه از یک ترشولد می‌باشد. برای مرج کردن هم از تابع Qtsetblk() استفاده شده است. لازم به توضیح است که این منبع در پیاده‌سازی کمک فراوانی کرده به من کرد. ساختار را در تابع

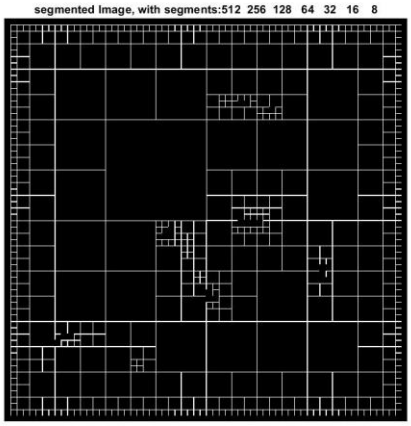
```
function blocks =  split_and_merge(I,segment_size,
Similarity_Threshold)
```

پیاده کردیم که در آن ورودی تصویر و ابعاد سگمنت‌ها و ترشولد تشابه برای اسپلیت کردن وارد می‌شود. توجه داریم کوچکترین (آخرین عدد در این آرایه همان ورودی مد نظر سوال می‌باشد). در خروجی نیز مرزبندی سگمنت‌ها پس از اسپلیت و مرج داده می‌شود. سپس اقدام به بررسی اثر سایز سگمنت کوچک کرده‌ایم:

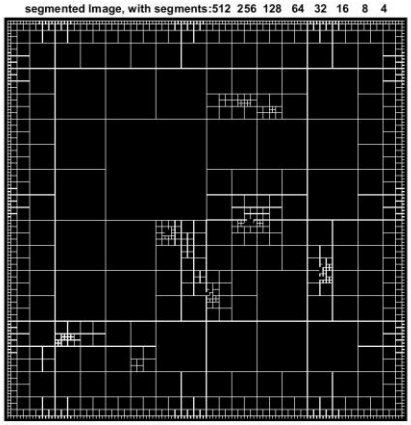
کوچکترین سگمنت: عرض ۱۶ پیکسل



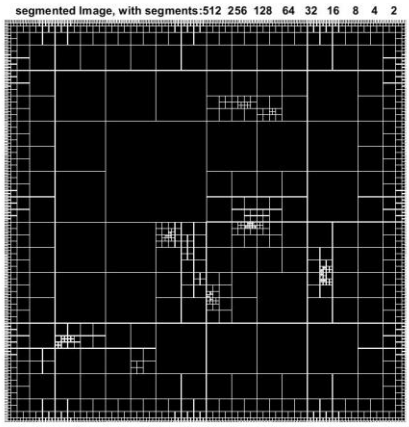
کوچکترین سگمنت: عرض ۸ پیکسل



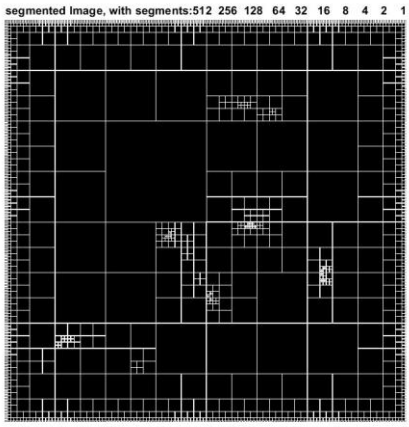
کوچکترین سگمنت: عرض ۴ پیکسل



کوچکترین سگمنت: عرض ۲ پیکسل



کوچکترین سگمنت: عرض ۱ پیکسل



تحلیل نتایج

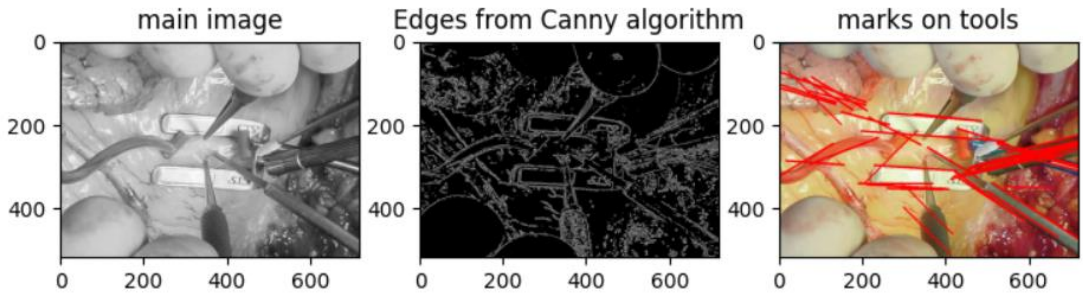
مشاهده می‌شود که هر چه قابلیت بررسی سگمنت‌های کوچک‌تر بیشتر می‌گردد، ناحیه‌های دقیقی که نسبت به زمینه پیک‌های شدتی دارند، بیشتر فراهم می‌شود و نتایج دقیق‌تری ایجاد می‌شود. همچنین در همه خروجی‌ها هرچه در یک جایی تعداد سگمنت‌ها بیشتر می‌شود، این معنی را دارد که شدت تغییرات شدت در آن نواحی بیشتر است و شامل سگمنت‌های بیشتری می‌باشد.

پاسخ سوال سوم: آ)

برای پیدا کردن اشیاء در یک تصویر می‌توان از الگوریتم هاف ترنسفورم استفاده کرد. معروف‌ترین شکلی که با این الگوریتم می‌توان یافت، خط و دایره می‌باشد. برای پیدا کردن ابزار جراحی ابتدا لبه‌های تصویر را با الگوریتم کنی که در درس نیز ارائه شده است، با تابع `cv2.Canny()` پیدا می‌کنیم. ورودی‌های این تابع شامل تصویر، بیشینه و کمینه محلی در لبه (برای ضخیم‌سازی لبه) و ابعاد کرنل سوبل می-باشد که در یکی از گام‌ها استفاده می‌کند. سپس در خروجی تصویر لبه‌ها داده می‌شود. سپس تصویر لبه‌ها را به عنوان تصویر باینری ورودی به تابع تبدیل هاف یعنی `cv2.HoughLinesP()` می‌دهیم. توجه داریم بجز این تابع، تابع `cv2.HoughLines()` نیز وجود دارد، اما تابع اول کارایی بهتری دارد. آرگومان‌های این تابع به شرح زیر هستند:

- *dst*: Output of the edge detector. It should be a grayscale image (although in fact it is a binary one)
- *lines*: A vector that will store the parameters $(x_{start}, y_{start}, x_{end}, y_{end})$ of the detected lines
- *rho* : The resolution of the parameter r in pixels. We use **1** pixel.
- *theta*: The resolution of the parameter θ in radians. We use **1 degree** (CV_PI/180)
- *threshold*: The minimum number of intersections to "*detect*" a line
- *minLineLength*: The minimum number of points that can form a line. Lines with less than this number of points are disregarded.
- *maxLineGap*: The maximum gap between two points to be considered in the same line.

سپس وقتی خطوط پیدا شد، با تابع `cv2.line()` که در ورودی مختصات دو نقطه و رنگ خط را به همراه تصویری که قرار است روی آن خط ترسیم کند را می‌گیرد و در خروجی چیزی نمی‌دهد اما روی مرزهای هدف خط ترسیم می‌کند، خطوط را روی یک کپی از تصویر ترسیم می‌کنیم. در خروجی و نمایش نتایج داریم:



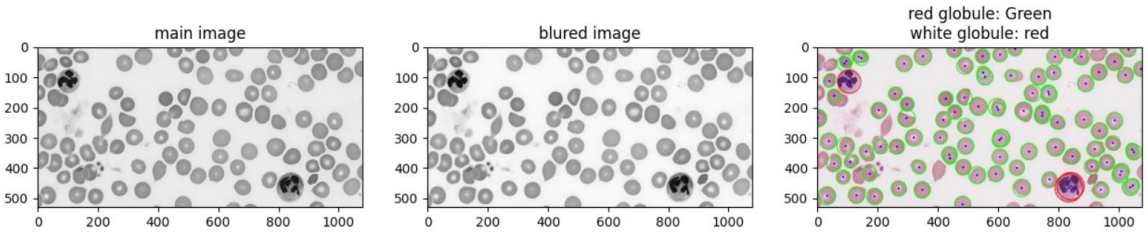
مشاهده می‌شود اکثر ابزار تشخیص داده شده‌اند البته در گوشه بالا سمت چپ یک سری خطوط درست تشخیص داده نشده‌اند. کلا این روش بسیار به پارامترها حساس می‌باشد و شاید اعمال یک سری پیش-پردازش‌ها مثل اروژن روی لبه‌ها قبل از ورودی به این روش بتواند خروجی دقیق‌تری ایجاد کند.

پاسخ سوال سوم: ب)

با توجه به اینکه گلبول‌ها دایره شکل هستند، از الگوریتم هاف‌ترنسفورم برای پیدا کردن آن‌ها استفاده می‌کنیم. بدین منظور از تابع `cv2.HoughCircles()` استفاده می‌کنیم که پارامترهای ورودی آن به شرح زیر هستند:

- image – Grayscale input image
- circles – Output vector of found circles. This vector is encoded as 3-element floating-point vector (x,y,radius). This is only needed in c++
- method – Detection method to use. CV_HOUGH_GRADIENT is currently the only available method
- dp – Inverse ratio of the accumulator resolution to the image resolution
- minDist – Minimum distance between the centers of the detected circles
- param1 – In case of CV_HOUGH_GRADIENT , it is the higher threshold of the two passed to the *Canny()* edge detector
- param2 – In case of CV_HOUGH_GRADIENT , it is the accumulator threshold for the circle centers at the detection stage
- minRadius – Minimum circle radius
- maxRadius – Maximum circle radius.

توجه داریم لبه‌یابی با الگوریتم کنی جزئی درونی در این تابع می‌باشد و فقط کافی می‌باشد پارامترهای آن را تنظیم کنیم. همچنین برای بهتر شدن نتیجه و کاهش نویز در ورودی، تصویر ورودی را بلر می‌کنیم (با استفاده از تابع `cv2.medianBlur()`). در ابتد برای رنج شعاعی دایره‌های کوچک‌تر که گلوبول‌های قرمز هستند، دایره‌ها را با مقدار مناسب پارامترها پیدا می‌کنیم و با تابع `cv2.circle()` اقدام به رسم دایره سبز بر روی آن‌ها می‌کنیم و سپس با رنج شعاعی بزرگتر برای پیدا کردن گلبول‌های سفید اقدام می‌کنیم و از رنگ قرمز برای نمایش آن‌ها استفاده می‌کنیم. سپس در انتها اقدام به رسم نتایج می‌کنیم:



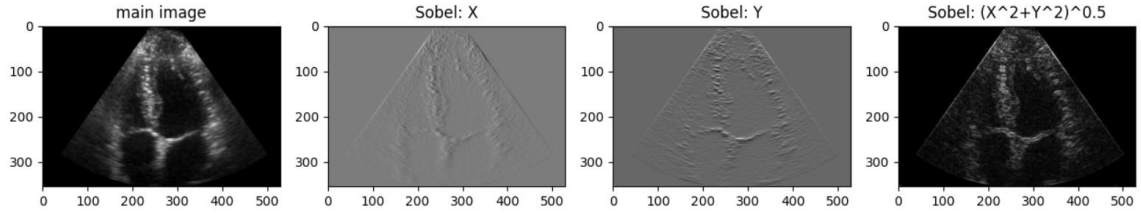
مشاهده می‌شود با دقت خوبی گلوبول‌های قرمز (سبرز) و سفید (قرمز) تشخیص داده شده‌اند. توجه داریم نتایج این روش به تنظیم دقیق پارامترها بسیار حساس و وابسته می‌باشد.

پاسخ سوال چهارم)

به ترتیب الگوریتم‌های خواسته شده جهت تشخیص لبه اجرا می‌شود.

تشخیص لبه با Sobel

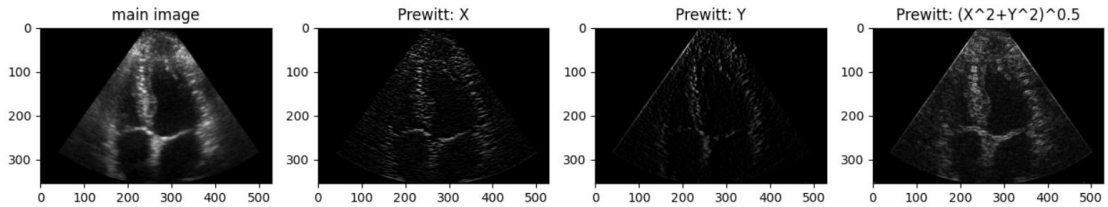
بدین منظور از تابع `cv2.Sobel()` جهت محاسبه گرادیان در هر دو راستای تصویر استفاده می‌کنیم. سپس اندازه‌های نرمال شده ۰ تا ۲۵۵ را در دو راستا بصورت فاصله اوقلیدسی مبنای اندازه گرادیان خروجی قرار می‌دهیم. در خروجی داریم:



مشاهده می‌شود که لبه‌ها پیدا شده‌اند اما هم اتصال کافی برقرار نیست و هم درون نواحی هم همچنان دقت لازم برقرار نمی‌باشد.

تشخیص لبه با Prewitt

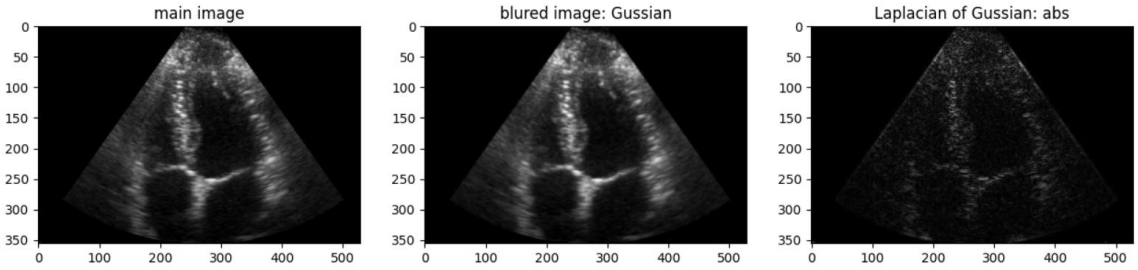
بدین منظور کرنل فضایی این فیلتر را در دو راستا با کمک تابع `cv2.filter2D()` بر روی تصویر اعمال می‌کنیم، سپس اندازه‌های نرمال شده ۰ تا ۲۵۵ را در دو راستا بصورت فاصله اوقلیدسی مبنای اندازه گرادیان خروجی قرار می‌دهیم. در خروجی داریم:



مشاهده می‌شود که لبه‌های اصلی نسبت به سوبل بهتر بولد شده‌اند. البته از داخل نیز این بولد شدن وجود دارد و کلا تصویر خروجی گرادیان نهایی محتوای درون مرز بولد شده نیز دارد که زیاد مطلوب نیست.

تشخیص لبه با LoG

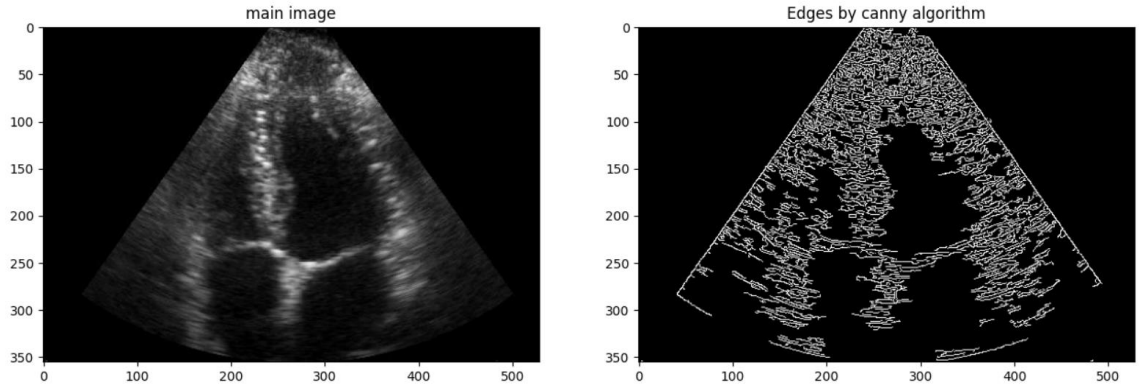
بدین منظور با تابع `cv2.GaussianBlur()` ابتدا فیلتر گوسین را جهت بلر کردن تصویر اعمال می‌کنیم. سپس با تابع `cv2.Laplacian()` از آن لاپلاسین می‌گیریم. در نهایت مقدار قدر مطلق را(نرمال شده به ۰ تا ۲۵۵) نمایش می‌دهیم. پس در نمایش داریم:



مشاهده می‌شود لبه‌ها تشخیص داده شده‌اند. در این روش نویزها به خاطر بلر شدن کم نقش‌تر در خروجی حضور دارند اما همچنان لبه‌ها بصورت نقطه‌ای و گاها ناپیوسته وجود دارند. در کل این خروجی نسبت به خروجی سوبل و پریویت تارتر می‌باشد.

تشخیص لبه با Canny

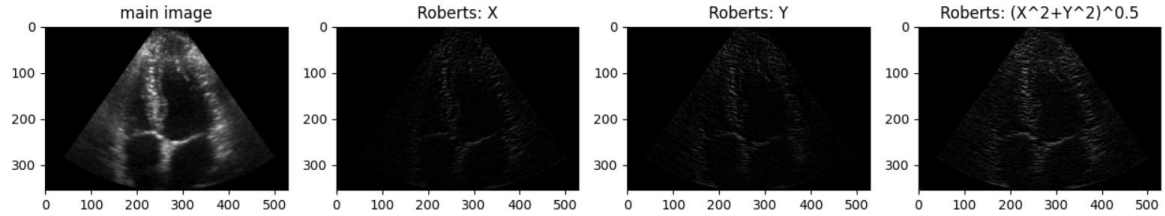
برای پیاده کردن الگوریتم کنی از تابع `cv2.Canny()` استفاده می‌کنیم که در واقع کار را بسیار ساده می‌کند. سپس تصویر اصلی و خروجی این الگوریتم را ترسیم می‌کنیم:



همانطور که مشاهده می‌شود، خروجی بر خلاف سه روش قبل لبه‌های پیوسته ایجاد می‌کند که در درس نیز بحث گردید. بعبارتی فقط نقاطی که روی مرز هستند، داده نمی‌شود بلکه خود مرز پیدا می‌شود که با تغییر پارامترهای ترشولدی می‌توان نتایج متفاوتی و بهتری پیدا کرد.

تشخیص لبه با Roberts

رابرت نیز یک کرنل در دو راستای عمودی و افقی می‌باشد که باید بر روی تصویر اعمال شود. پس کرنل را ایجاد کردیم و با تابع `cv2.filter2D()` با تصویر کانوالو کردیم. برای اینکه اثر خروجی در دو جهت مشخص گردد، اندازه اقلیدسی را نیز نمایش می‌دهیم. سپس در خروجی داریم:



ملاحظه می‌گردد که مثل ۳ لبه یاب اول این روش نیز نقطه می‌دهد و مانند کنی قادر نیست مرز ممتد بدهد که واضح است زیرا کنی یک الگوریتم پیشرفته‌تر برای چنین هدفی است. همچنین رابرتس در خروجی اگرچه نسبت به الگوریتم‌های عادی (منظور همه بجز کنی) دارای شارپ بودن کمتری است، اما نواحی درون مرز هم بولد نیستند و خبری از نویزهای شدید نیز نمی‌باشد.