

تمرین سری پنج

درس تصویرپردازی رقمی

فرهاد دلیرانی

۹۶۱۳۱۱۲۵

dalirani@aut.ac.ir

dalirani.1373@gmail.com

فهرست

| | |
|----|----------------------|
| ۱ | ابزارهای استفاده شده |
| ۲ | تمرین ۱ |
| ۲ | ۲ قسمت a |
| ۳ | ۳ قسمت b |
| ۵ | ۵ قسمت c |
| ۹ | ۹ قسمت d |
| ۱۰ | ۱۰ قسمت e |
| ۱۱ | ۱۱ قسمت f |
| ۱۶ | ۱۶ تمرین ۲ |
| ۱۶ | ۱۶ قسمت a |
| ۱۶ | ۱۶ A1 – EZW |
| ۱۶ | ۱۶ A2 – SPIHT |
| ۱۶ | ۱۶ A3 – STW |
| ۱۷ | ۱۷ A4 – WDR |
| ۱۷ | ۱۷ A5 – ASWDR |
| ۱۷ | ۱۷ b قسمت |
| ۱۸ | ۱۸ تمرین ۳ |
| ۱۸ | ۱۸ a قسمت |
| ۲۰ | ۲۰ b قسمت |
| ۲۲ | ۲۲ c قسمت |
| ۲۴ | ۲۴ d قسمت |
| ۲۶ | ۲۶ e قسمت |
| ۲۹ | ۲۹ f قسمت |
| ۳۳ | ۳۳ g قسمت |
| ۳۶ | ۳۶ h قسمت |
| ۴۰ | ۴۰ تمرین ۴ |

| | |
|----------|---------|
| ٤٠..... | قسمت a |
| ٤١..... | قسمت b |
| ٤٢..... | قسمت c |
| ٤٣..... | قسمت d |
| ٤٤..... | قسمت e |
| ٤٥..... | قسمت f |
| ٤٦..... | تمرين ٥ |
| ٤٧..... | قسمت a |
| ٤٩..... | قسمت b |
| ٥٥..... | قسمت c |
| ٥٧..... | قسمت d |
| ٦٠..... | قسمت e |
| ٦٣..... | قسمت f |
| ٦٦..... | قسمت g |
| ٧٠..... | قسمت h |
| ٧٤..... | تمرين ٦ |
| ٧٤..... | قسمت a |
| ٧٧..... | قسمت b |
| ٧٩..... | قسمت c |
| ٨٣..... | تمرين ٧ |
| ٨٣..... | قسمت a |
| ٨٦..... | قسمت b |
| ٩٠..... | قسمت c |
| ٩٢..... | قسمت d |
| ٩٤..... | قسمت e |
| ٩٨..... | قسمت f |
| ١٠٠..... | قسمت g |
| ١٠٢..... | قسمت h |

| | |
|----------|---------|
| ١٠٨..... | تمرين ٨ |
| ١٠٨..... | قسمت a |
| ١١٢..... | قسمت b |
| ١١٣..... | قسمت c |
| ١٢٩..... | قسمت d |
| ١٣٠..... | قسمت e |
| ١٣٢..... | تمرين ٩ |
| ١٣٢..... | قسمت a |
| ١٣٢..... | قسمت b |
| ١٣٤..... | قسمت c |
| ١٣٤..... | قسمت d |
| ١٣٥..... | قسمت e |

ابزارهای استفاده شده

زبان برنامه نویسی: متلب

محیط توسعه: Matlab R2016a

سیستم عامل: ویندوز ۱۰

تمرین ۱

کدها و خروجی‌های این قسمت در p1 قرار دارد.

قسمت a

کد این قسمت در p1a.m قرار دارد. در ابتدا تصویر را می‌خوانم و از رابطه‌ی زیر استفاده می‌کنم:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 0.000 & 1.403 \\ 1.000 & -0.344 & -0.714 \\ 1.000 & 1.773 & 0.000 \end{bmatrix} \cdot \begin{bmatrix} Y \\ C_b - 128 \\ C_r - 128 \end{bmatrix}$$

این رابطه، پیکسل را از فضای YC_bC_R به RGB می‌برد، اگر دو طرف رابطه را ضرب در معکوس ماتریس تبدیل کنیم، می‌توان از رابطه‌ی جدید به دست آمده برای تبدیل RGB به YC_bC_R استفاده کرد.

ورودی:



: (p1a.png



قسمت b

کد این قسمت در p1b.m قرار دارد. در کد، در ابتدا تصویر را می‌خوانم، سپس آن را از uint8 به double تبدیل می‌کنم. از آنجایی که تصویر را برای اعمال DCT باید به بلاک‌هایی 4×4 تبدیل کنیم، یک ستون به تصویر اضافه می‌کنم زیرا تعداد سطرها بر چهار بخش‌پذیر است و لی در مورد تعداد ستون‌ها اینطور نیست. از آنجایی که تصویر رنگی است، بر روی سه کانال مختلف به صورت مجزا، همانند توضیحات لینک درون متن تمرین، تصویر را حساب می‌کنم و ۲۵ درصد از ضرایب را نگه می‌دارم. در زیر خروجی‌های کد را مشاهده می‌کنید:

وروودی:



:خروجی(p1b.png)



:PSNR مقدار

مقدار PSNR در شرایطی که تصویرها به صورت double هستند محاسبه شده‌اند.

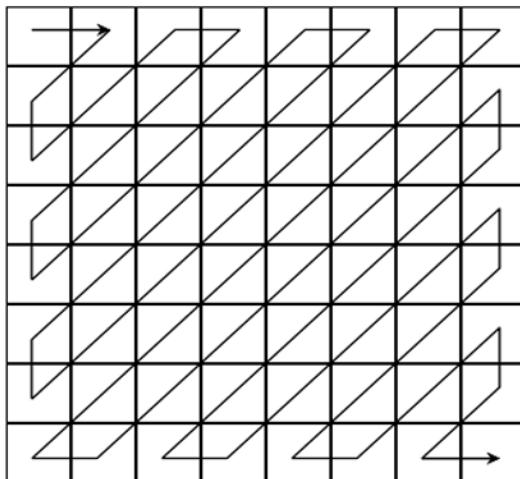
PSNR:

32.5836

دو تصویر بسیار شبیه هم هستند ولی تفاوت‌هایی دیده می‌شود. در ناحیه‌های یکنواخت به سختی تغییر قابل مشاهده است ولی در ناحیه‌های غیر یک نواخت و لبه‌ها، مانند مزرعه‌ی پشت سر کیم جونگ اون به راحتی می‌توان مشاهده کرد که تصویر شترنجی مانند شده است و هر گروه از پیکسل‌ها یک مقدار را دارند.

قسمت C

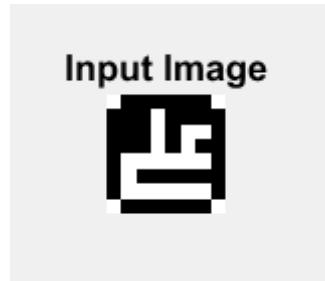
کد این قسمت در p1c.m قرار دارد. در این کد بر اساس ترتیب موجود در شکل زیر پیکسل‌ها را پیمایش می‌کنم.



| | | | | | | | |
|----|----|----|----|----|----|----|----|
| 0 | 1 | 5 | 6 | 14 | 15 | 27 | 28 |
| 2 | 4 | 7 | 13 | 16 | 26 | 29 | 42 |
| 3 | 8 | 12 | 17 | 25 | 30 | 41 | 43 |
| 9 | 11 | 18 | 24 | 31 | 40 | 44 | 53 |
| 10 | 19 | 23 | 32 | 39 | 45 | 52 | 54 |
| 20 | 22 | 33 | 38 | 46 | 51 | 55 | 60 |
| 21 | 34 | 37 | 47 | 50 | 56 | 59 | 61 |
| 35 | 36 | 48 | 49 | 57 | 58 | 62 | 63 |

خروجی‌های کد را در زیر مشاهده می‌کنید:

وروودی:



وکتور بر اساس ترتیب زیک زاک (p1c.png)



مقدار پیکسل هایی که در ترتیب زیک زاک مشاهده می شوند:

Zig Zag Order of Image(values):

Columns 1 through 19

1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0

Columns 20 through 38

1 0 0 1 1 1 0 0 0 1 0 1 0 1 0 1 1 0 1

Columns 39 through 57

0 1 1 1 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1

Columns 58 through 64

0 0 1 0 0 0 1

اندیس هایی که به ترتیب باید مشاهده کنیم:

Zic Zac Order of Image (Index) :

| | |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 2 | 1 |
| 3 | 1 |
| 2 | 2 |
| 1 | 3 |
| 1 | 4 |
| 2 | 3 |
| 3 | 2 |
| 4 | 1 |
| 5 | 1 |
| 4 | 2 |
| 3 | 3 |
| 2 | 4 |
| 1 | 5 |
| 1 | 6 |
| 2 | 5 |
| 3 | 4 |
| 4 | 3 |
| 5 | 2 |
| 6 | 1 |
| 7 | 1 |
| 6 | 2 |
| 5 | 3 |
| 4 | 4 |
| 3 | 5 |
| 2 | 6 |
| 1 | 7 |

| | |
|---|---|
| 1 | 8 |
| 2 | 7 |
| 3 | 6 |
| 4 | 5 |
| 5 | 4 |
| 6 | 3 |
| 7 | 2 |
| 8 | 1 |
| 8 | 2 |
| 7 | 3 |
| 6 | 4 |
| 5 | 5 |
| 4 | 6 |
| 3 | 7 |
| 2 | 8 |
| 3 | 8 |
| 4 | 7 |
| 5 | 6 |
| 6 | 5 |
| 7 | 4 |
| 8 | 3 |
| 8 | 4 |
| 7 | 5 |
| 6 | 6 |
| 5 | 7 |
| 4 | 8 |
| 5 | 8 |
| 6 | 7 |
| 7 | 6 |
| 8 | 5 |
| 8 | 6 |
| 7 | 7 |
| 6 | 8 |
| 7 | 8 |
| 8 | 7 |
| 8 | 8 |

کد این قسمت در p1d.m قرار دارد. بر اساس توضیحات درون تصویر زیر عمل کرده‌ام:



Lossless Compression

- ❖ **Run-length encoding**
 - Replaces runs of 0s with a count of how many 0s.

```
00000000000000100000000011000000000000000000000010...011000000000000
          ^                                (30 0s)
          14           9           0           20           30   0           11
```

- ❖ **Replace each decimal value with a 4-bit binary value (nibble)**
 - Note: If you need to code a value larger than 15, you need to use two consecutive 4-bit nibbles
 - ✓ The first is decimal 15, or binary 1111, and the second nibble is the remainder
 - ✓ For example, if the decimal value is 20, you would code 1111 0101 which is equivalent to $15 + 5$
 - If you want to code the value 15, you still need two nibbles: 1111 0000
 - ✓ The rule is that if you ever have a nibble of 1111, you must follow it with another nibble

وروودی:

```
11111100001111000000001110000011111111111110111111110101110111100
```

در زیر کد فشرده شده را مشاهده می‌کنید:

| Run Length Code | | | | | | | | | | | | | | | | | | |
|-----------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Columns 1 through 19 | | | | | | | | | | | | | | | | | | |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| Columns 20 through 38 | | | | | | | | | | | | | | | | | | |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| Columns 39 through 57 | | | | | | | | | | | | | | | | | | |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| Columns 58 through 68 | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | | | | | | | | |

و به صورت مرتب تر:

Run Length Code

```
[01100100010010000011010111101000001100100010
00100010011000101000010]
```

قسمت e

کد این قسمت در p1e.m قرار دارد. ابتدا تصویر را از ورودی می‌خوانیم. سپس تعداد تکرار هر شدت روشنایی را می‌شماریم و در ادامه تعداد تکرار هر شدت روشنایی را تقسیم بر تعداد کل پیکسل‌ها می‌کنیم تا احتمال رخداد هر شدت روشنایی به دست آید سپس بر اساس تابع زیر آنتروپی را محاسبه می‌کنیم:

$$E(S) = \sum_{i=1}^c -p_i \log_2 p_i$$

خروجی کد را در زیر مشاهده می‌کنید:

```
>> p1e
Entropy:
7.031236675705144
```

قسمت f

کد این قسمت در p1f.m قرار دارد، ابتدا تصویر را از ورودی می‌خوانم سپس با استفاده از دستور `imwrite` و پارامتر `Quality` ، تصویر را با فرمت `jpeg` ذخیره می‌کنم. در زیر خروجی‌های کد را مشاهده می‌کنید:

تصویر ورودی:



تصویر با پارامتر `quality` برابر با ۱۰ (p1f-q-10.jpeg)



تصویر با پارامتر quality برابر با ۲۵ (: p1f-q-25.jpeg)



تصویر با پارامتر quality برابر با ۵۰ (p1f-q-50.jpeg) :



تصویر با پارامتر quality برابر با ۷۵ (p1f-q-75.jpeg) :



تصویر با پارامتر quality برابر با ۱۰۰ (p1f-q-100.jpeg)



حجم و PSNR تصویرهای مختلف:

| تصویر | PSNR | حجم |
|-------------|-------|--------|
| تصویر اصلی | - | 1.55MB |
| Quality 10 | ۲۹,۵۴ | 36.1KB |
| Quality 25 | ۳۴,۰۰ | 55.0KB |
| Quality 50 | ۳۷,۱۴ | 77.8KB |
| Quality 75 | ۴۰,۰۷ | 108KB |
| Quality 100 | ۴۸,۲۹ | 525KB |

میزان پارامتر Quality می‌تواند عددی از ۰ تا ۱۰۰ باشد، هر چه این عدد بیشتر باشد تصویر در فرمت JPEG کیفیت بیشتری خواهد داشت و فشرده سازی کمتری صورت می‌پذیرد. هر چه Quality بیشتر شده است

افزایش پیدا کرده و این نشان می‌دهد که تصویر با پارامتر Quality بیشتر به تصویر اصلی نزدیک‌تر است. همین‌طور با افزایش پارامتر Quality حجم تصویر زیادتر شده است.

در تصویر با Quality برابر با ۱۰ همین که مشاهده می‌شود یک نوع حالت آبرنگ مانند در بعضی ناحیه‌ها ماند دست راست کیم جونگ اون ایجاد شده است و تغییر آهسته‌ی رنگ در تصویر اصلی در تصویر فشرده شده رخ نداده است و ناحیه‌هایی که رنگ به آرامی در آن تغییر پیدا کرده است به صورت یک پارچه در آمده‌اند، هر چه میزان پارامتر Quality افزایش یافته است اثر این موضوع کمتر شده است.

تمرين ۲

کدهای این قسمت در پوشه‌ی p2 قرار دارد.

قسمت a

A1 – EZW

مخف EZW مخف Embedded ZeroTree Wavelet است. این الگوریتم از اولین روش‌هایی بود که از ویژگی‌های wavelet استفاده کرد. همچنین این الگوریتم از نوع Lossy است. این الگوریتم بر اساس ۴ ایده‌ی اصلی زیر است:

- تبدیل Discrete Wavelet

- پیش‌بینی غیبت اطلاعات ارزشمند با بررسی شباهت اجزای تصویر به صورت سلسله مراتبی
Entropy-coded successive-approximation quantization-
- فشرده سازی Lossless اطلاعات که با استفاده از adaptive arithmetic coding حاصل می‌شود.
برای بررسی ساختاری و سلسله مراتبی تصویر از Quad Tree استفاده می‌شود.

A2 – SPIHT

این الگوریتم نمونه‌ی بسیار بهبود یافته‌ی EZW است. تقریباً این الگوریتم در میان روش‌هایی بر اساس wavelet Set Partitioning in Hierarchical Trees SPIHT مخف است. Set Partitioning به درخت Quad Tree اشاره دارد که در EZW هم از آن استفاده می‌شود. Hierarchical Trees به نحوی تقسیم کردن Quad Tree اشاره دارد. با آنالیز دقیق ضرایت حاصل از تبدیل Partitioning wavelet و با تغییر نحوی تقسیم کردن درخت در هر مرحله و تعیین Threshold، بهبود فشرده سازی چشمگیری در این الگوریتم رخ می‌دهد.

A3 – STW

همان SPIHT STW است فقط در سازماندهی کدهای خروجی با دقت بیشتری عمل می‌کند. تنها تفاوا STW و EZW است که STW از روش متفاوتی برای کد کردن اطلاعات zerotree استفاده می‌کند.

A4 – WDR

یکی از مشکلات SPIHT این است که به صورت ضمنی مکان ضرایب با اهمیت را تعیین می‌کند. این موضوع باعث ایجاد مشکلاتی در عملگرهایی مانند انتخاب ناحیه در داده‌های فشرده شده می‌شود که به مکان دقیق ضرایب با اهمیت نیاز دارد. ROI، رزولوشن ناحیه‌های خاصی را افزایش می‌دهد.

A5 – ASWDR

این الگوریتم Scanning Order که توسط WDR استفاده می‌شود را جهت بهبود عوامل فشرده سازی تغییر می‌دهد. این الگوریتم از WDR و SPIHT بهتر عمل می‌کند.

قسمت b

کد این قسمت در p2b.m قرار دارد. در این قسمت باید ۳۰ عکس تراپ را با الگوریتم‌های مختلف فشرده کنیم، در هنگام فشرده‌سازی با استفاده از بعضی از الگوریتم‌ها متوجه یک خطأ شدم، که یکی از راه‌های رفع آن این بود که طول و عرض تصویر توانی از عدد دو باشد. به همین دلیل برای اینکه به تابع فشرده‌سازی آرگمان‌های اضافه‌ای ندهم، تصویرها را با صفر Pad می‌کنم به گونه‌ای که عرض و طول تصویر به تزدیک‌ترین توان دو برسد. در همان جایی که تصویر را فشرده می‌کنیم، تصویر را از حالت فشرده خارج می‌کنیم تا PSNR را حساب کنیم. در زیر نتیجه‌ی فشرده‌سازی با الگوریتم‌های مختلف را مشاهده می‌کنید:

| نام روش | Average PSNR | حجم پوشش |
|------------------------|--------------|-----------------------|
| | | نسبت به تصویر Pad شده |
| تصویرهای ورودی | - | 4.05 MB |
| تصویرهای ورودی Pad شده | - | 4.08 MB |
| EZW | 37.9973 | 4.64 MB |
| SPHIT | 35.2082 | 2.26 MB |
| STW | 36.3420 | 3.13 MB |
| WDR | 37.9973 | 5.22 MB |
| ASWDR | 37.9973 | 5.14 MB |

تمرین ۳

کدهای این قسمت در p3 قرار دارد.

قسمت a

کدهای این قسمت در p3a.m و binary_dilation.m قرار دارد. در binary_dilation.m تابع زیر را نوشته‌ام که عمل Dilation را انجام می‌دهد:

```
function dialated_img = binary_dilation(img, st)
    % This function dialates input image with given structuring element
    % img = input image, binary image
    % st = structuring element, size of st should be odd
```

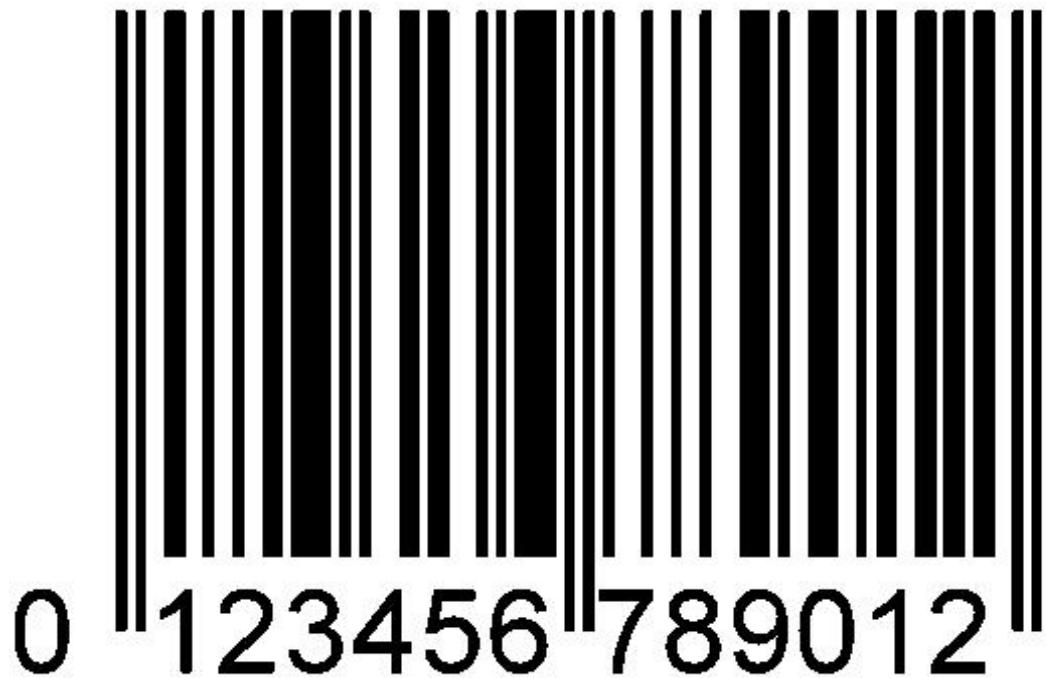
این تابع یک تصویر باینری و یک structuring element را دریافت می‌کند، سپس تصویر را به اندازه‌ی کافی با صفر Pad می‌کند، در ادامه Structuring Element را ۱۸۰ درجه چرخش می‌دهد. سپس ساختار را بر روی تصویر حرکت می‌دهد، اگر حداقل یک پیکسل یک از ساختار با یک پیکسل یک از تصویر همپوشانی داشته باشد، پیکسلی را که مرکز ساختار بر روی آن قرار گرفته است در خروجی برابر یک می‌شود.

در p3a.m تصویر را می‌خوانم سپس با فراخوانی تابع بالا و استفاده از ساختار زیر

```
% structuring element
st = [0 0 0 1 0 0 0;
      0 0 1 1 1 0 0;
      0 1 1 1 1 1 0;
      1 1 1 1 1 1 1;
      0 1 1 1 1 1 0;
      0 0 1 1 1 0 0;
      0 0 0 1 0 0 0];
```

عمل dilation را انجام می‌دهم. در زیر خروجی کد را مشاهده می‌کنید:

ورودي:



تصویر Dilate شده:



همین طور که مشاهده می‌شود تصویر ورودی بر اساس Structuring element گسترش یافته است و ناحیه‌های سفید گسترش یافته‌اند و قسمت‌هایی از ناحیه‌های سیاه را از بین برده‌اند.

قسمت b

کدهای این قسمت در binary_erosion.m و p3b.m قرار دارد. در binary_erosion.m تابع زیر را نوشته‌ام که عمل Erosion را انجام می‌دهد:

```
|function erosion_img = binary_erosion(img, st)
|    % This function erosions input image with given structuring element
|    % img = input image, binary image
|    % st = structuring element, size of st should be odd
```

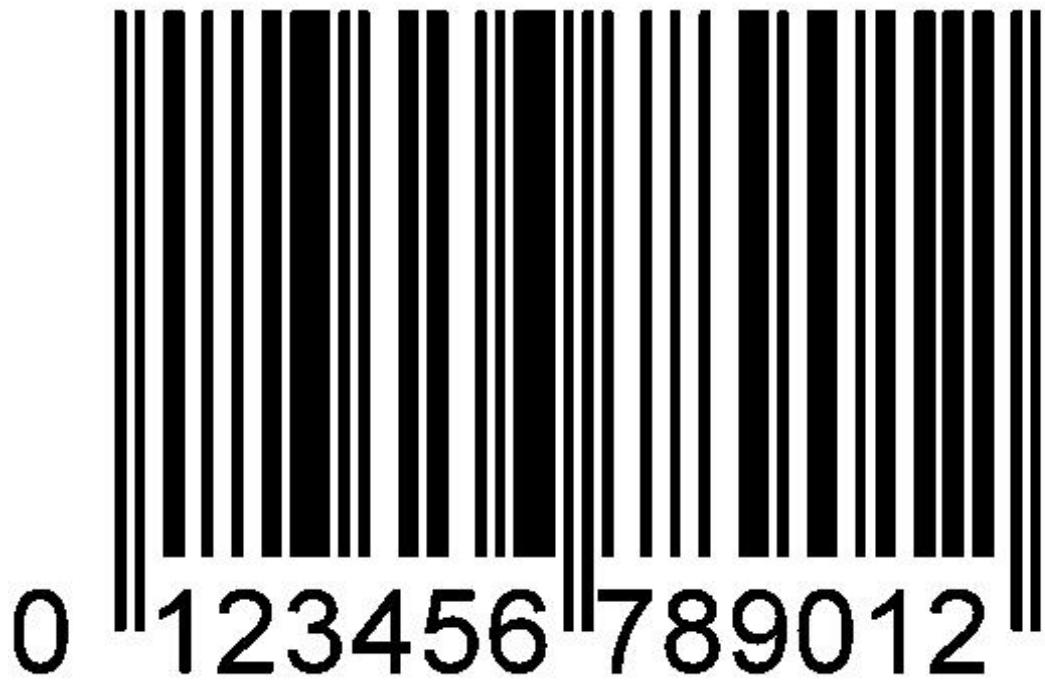
این تابع یک تصویر باینری و یک structuring element را دریافت می‌کند، سپس تصویر را به اندازه‌ی کافی با عدد یک Pad می‌کند. سپس ساختار را بر روی تصویر حرکت می‌دهد، اگر تمام پیکسل‌ها با مقدار برابر یک ساختار بر روی عده‌های یک در تصویر قرار گرفتند، پیکسلی را که مرکز ساختار بر روی آن قرار گرفته است در خروجی برابر یک می‌شود.

در p3b.m تصویر را می‌خوانم سپس با فراخوانی تابع بالا و استفاده از ساختار زیر

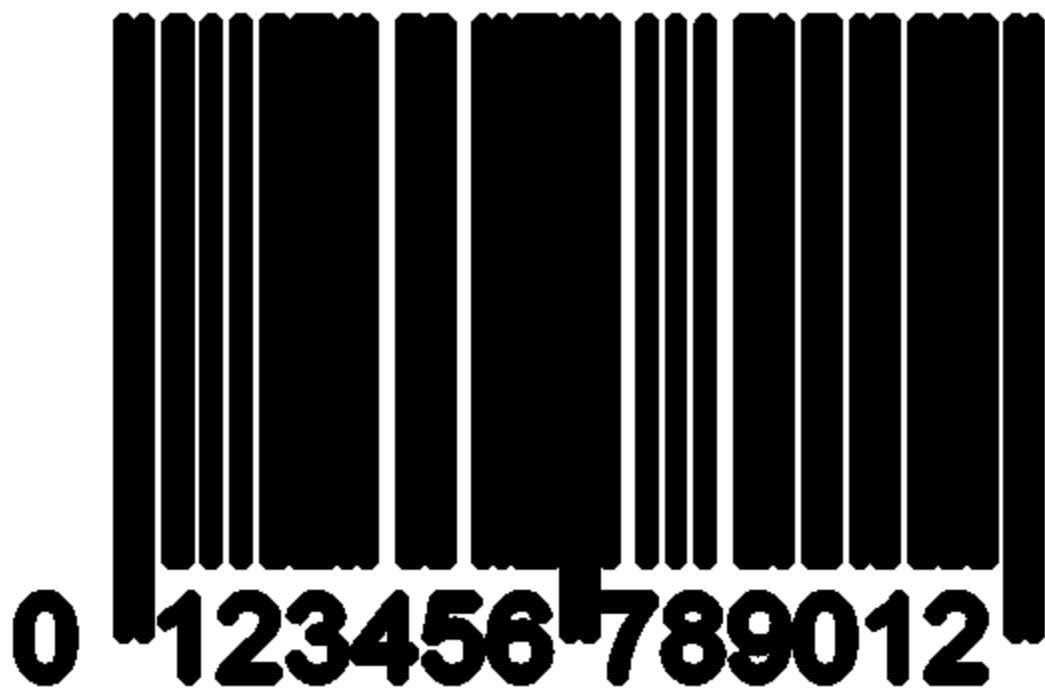
```
% structuring element
st = [0 0 0 1 0 0 0;
      0 0 1 1 1 0 0;
      0 1 1 1 1 1 0;
      1 1 1 1 1 1 1;
      0 1 1 1 1 1 0;
      0 0 1 1 1 0 0;
      0 0 0 1 0 0 0];
```

عمل Erosion را انجام می‌دهم. در زیر خروجی کد را مشاهده می‌کنید:

: ورودی



تصویر Erosion شده:



همین طور که در تصویر بالا مشاهده می شود، ناحیه های سفید دچار فرسایش شده اند.

قسمت C

کدهای این قسمت در p3c.m و binary_opening.m قرار دارد. در binary_opening.m تابع زیر را نوشته‌ام که عمل Opening را انجام می‌دهد:

```
function opened_img = binary_opening(img, st)
    % this function calculated opening of an image
    % structuring elemnt
```

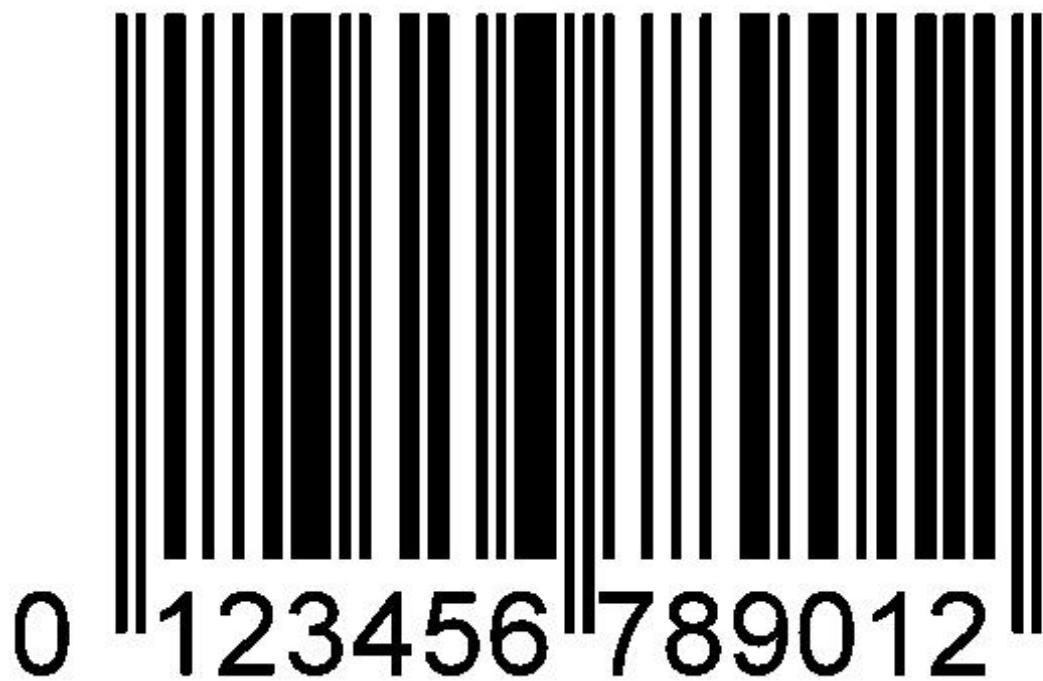
این تابع یک تصویر باینری و یک structuring element را دریافت می‌کند، سپس عمل opening را بر تصویر، با فراخوانی تابع Erosion و Dilation قسمتهای قبل اعمال می‌کند.

در p3c.m تصویر را می‌خوانم سپس با فراخوانی تابع بالا و استفاده از ساختار زیر

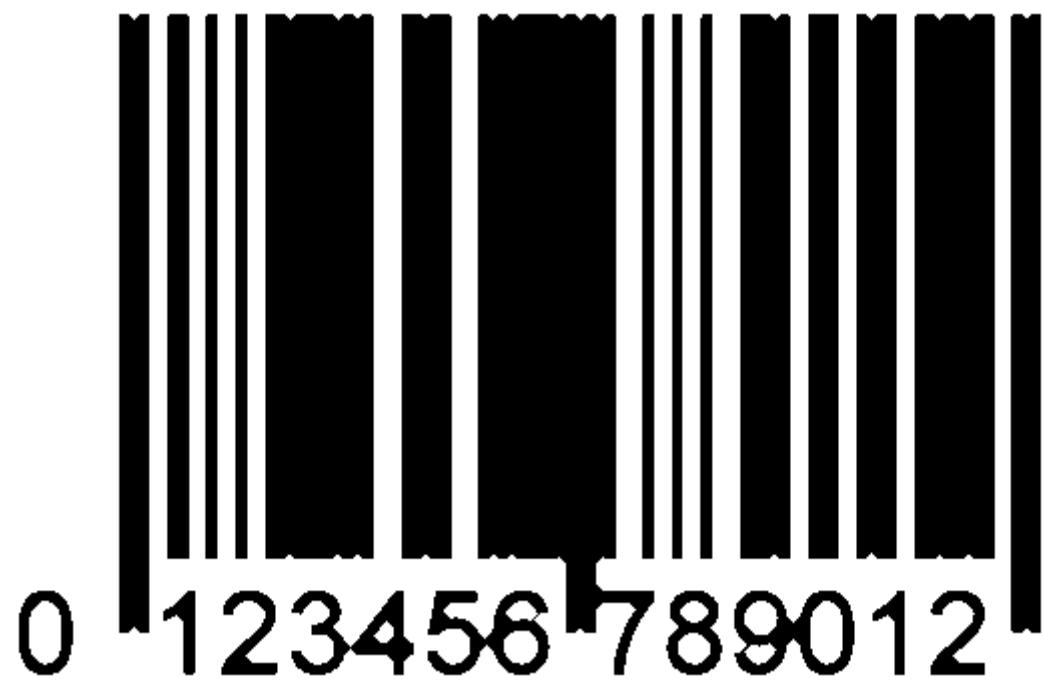
```
% structuring element
st = [0 0 0 1 0 0 0;
      0 0 1 1 1 0 0;
      0 1 1 1 1 1 0;
      1 1 1 1 1 1 1;
      0 1 1 1 1 1 0;
      0 0 1 1 1 0 0;
      0 0 0 1 0 0 0];
```

عمل opening را انجام می‌دهیم. در زیر خروجی کد را مشاهده می‌کنید:

وروودی:



تصویر حاصل از opening شده:



قسمت d

کدهای این قسمت در p3d.m و binary_closing.m قرار دارد. در binary_closing.m تابع زیر را نوشته‌ام که عمل closing را انجام می‌دهد:

```
function opened_img = binary_opening(img, st)
    % this function calculated opening of an image
    % structuring elemnt
```

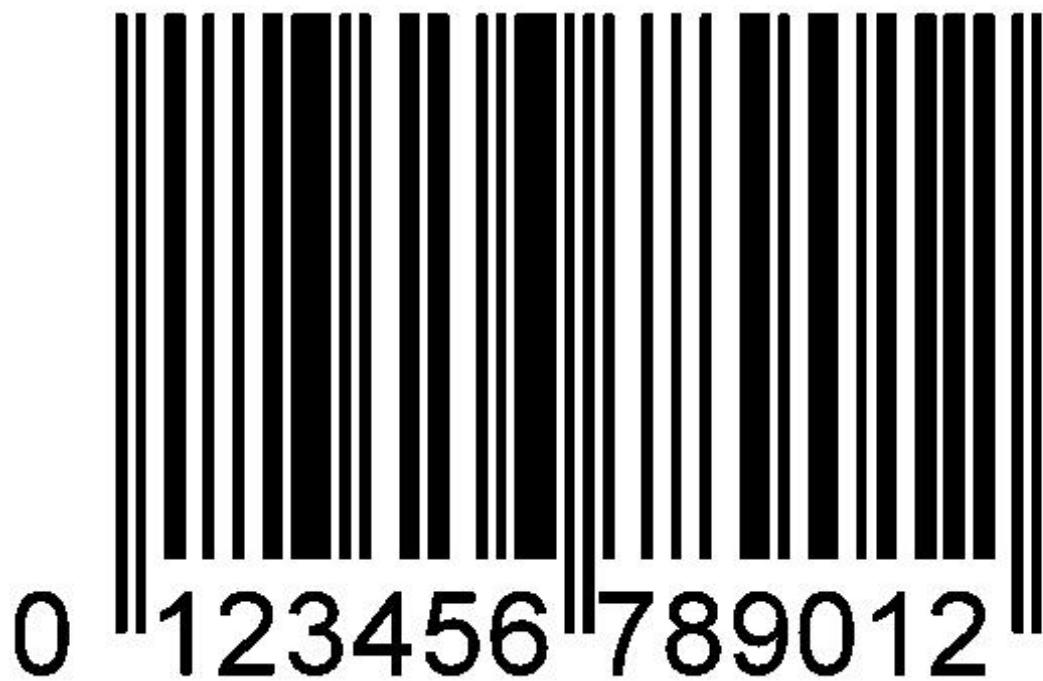
این تابع یک تصویر باینری و یک structuring element را دریافت می‌کند، سپس عمل closing را بر تصویر، با فراخوانی تابع Erosion و Dilation قسمت‌های قبل اعمال می‌کند.

در p3d.m تصویر را می‌خوانم سپس با فراخوانی تابع بالا و استفاده از ساختار زیر

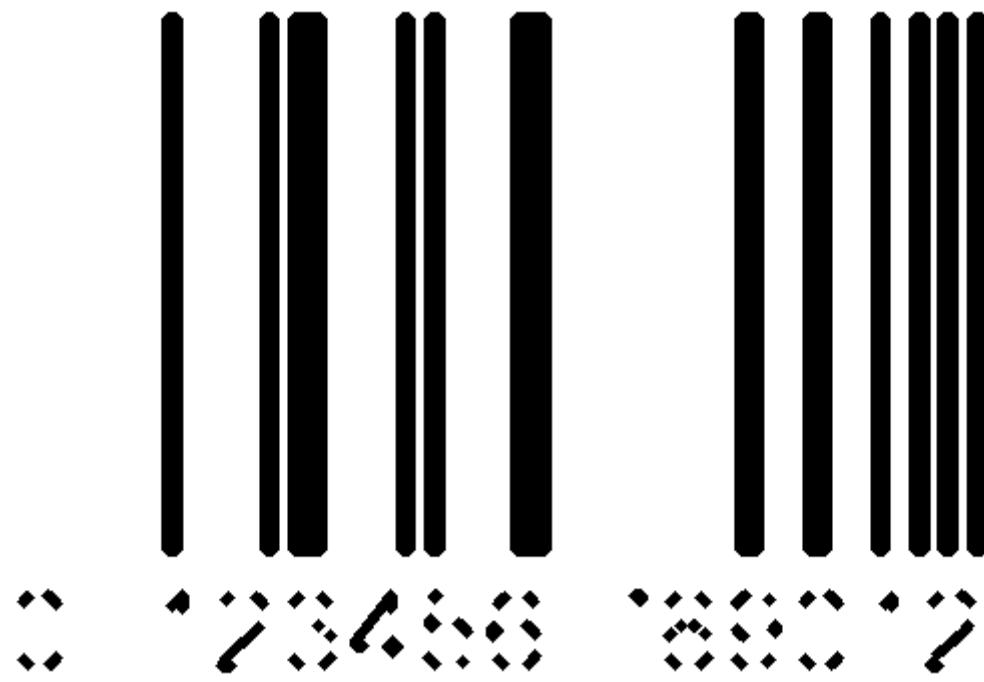
```
% structuring element
st = [0 0 0 1 0 0 0;
      0 0 1 1 1 0 0;
      0 1 1 1 1 1 0;
      1 1 1 1 1 1 1;
      0 1 1 1 1 1 0;
      0 0 1 1 1 0 0;
      0 0 0 1 0 0 0];
```

عمل closing را انجام می‌دهیم. در زیر خروجی کد را مشاهده می‌کنید:

وروودی:



تصویر حاصل از closing شده:



قسمت e

کدهای این قسمت در gray_dilation.m و p3e.m قرار دارد. در gray_dilation.m تابع زیر را نوشته‌ام که عمل Dilation را انجام می‌دهد:

```
|function dialated_img = gray_dilation(img, st)
% This function dialates input image with given structuring element
% img = input image
% st = structuring element, size of st should be odd
```

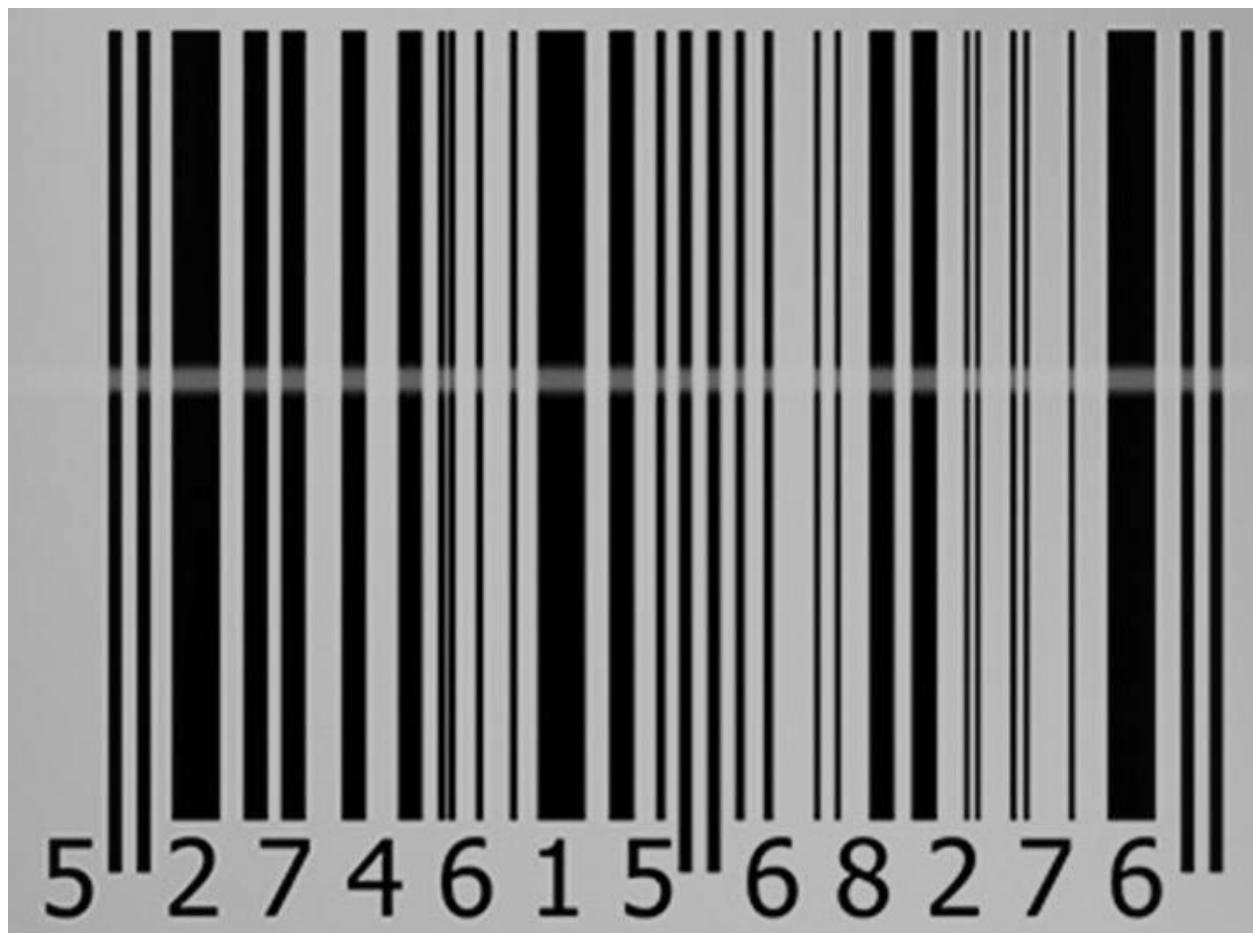
این تابع یک تصویر Gray Level و یک structuring element را دریافت می‌کند، سپس تصویر را به اندازه‌ی کافی با صفر Pad می‌کند، در ادامه Structuring Element را ۱۸۰ درجه چرخش می‌دهد. سپس ساختار را بر روی تصویر حرکت می‌دهد، در هر مرحله ساختار را با همسایگی از تصویر که ساختار بر روی آن قرار دارد جمع می‌کند و ماکریزیم شدت روشنایی موجود در تصویر را انتخاب می‌کند و در خروجی مقدار پیکسلی که مرکز همسایگی بر روی آن قرار دارد را برابر با آن قرار می‌دهد.

در p3e.m تصویر را می‌خوانم سپس با فراخوانی تابع بالا و استفاده از ساختار زیر

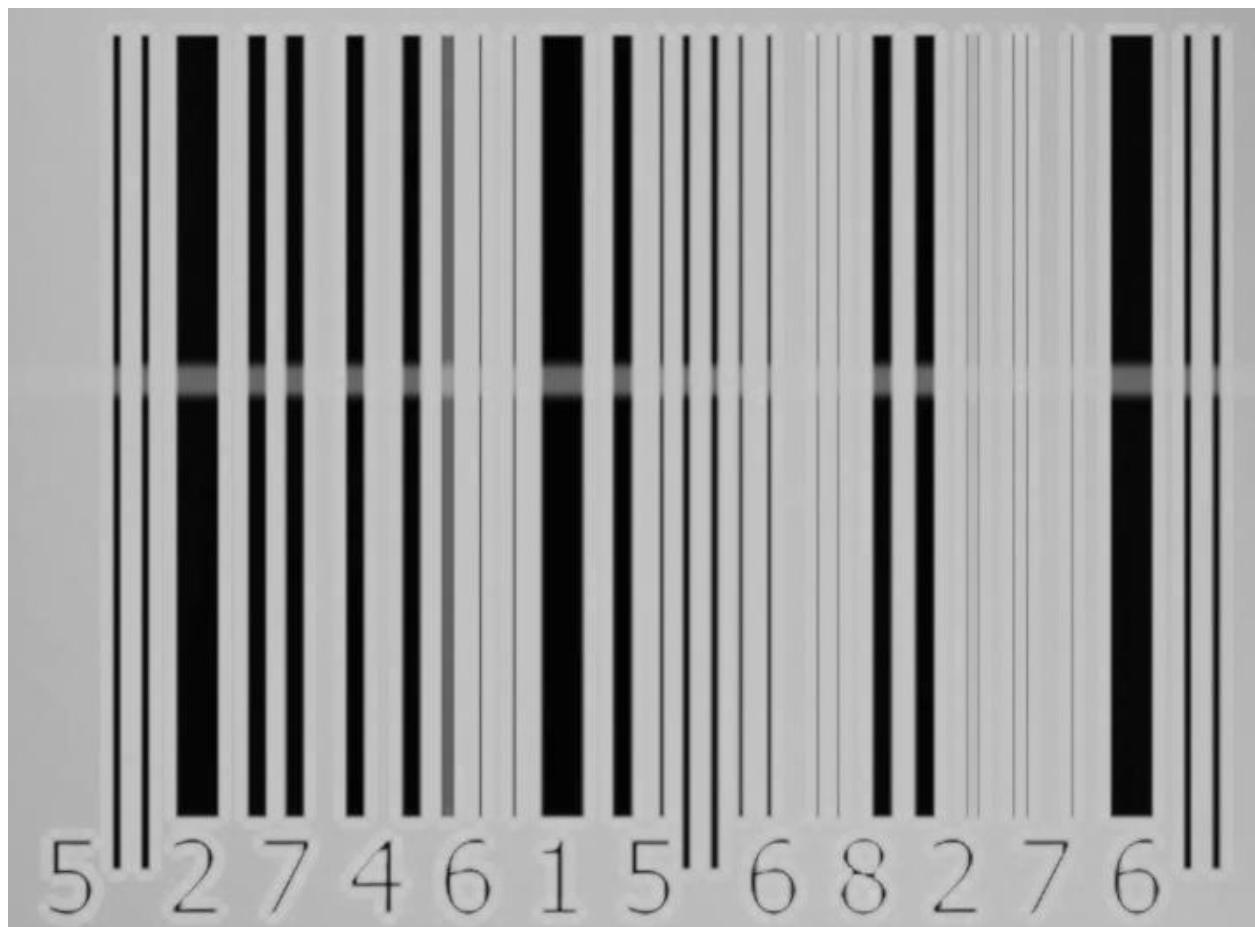
```
% structuring element
st = [0 0 1 0 0;
       0 1 1 1 0;
       1 1 1 1 1;
       0 1 1 1 0;
       0 0 1 0 0;];
st = st .* 0.01;
```

عمل dilation را انجام می‌دهم. در زیر خروجی کد را مشاهده می‌کنید:

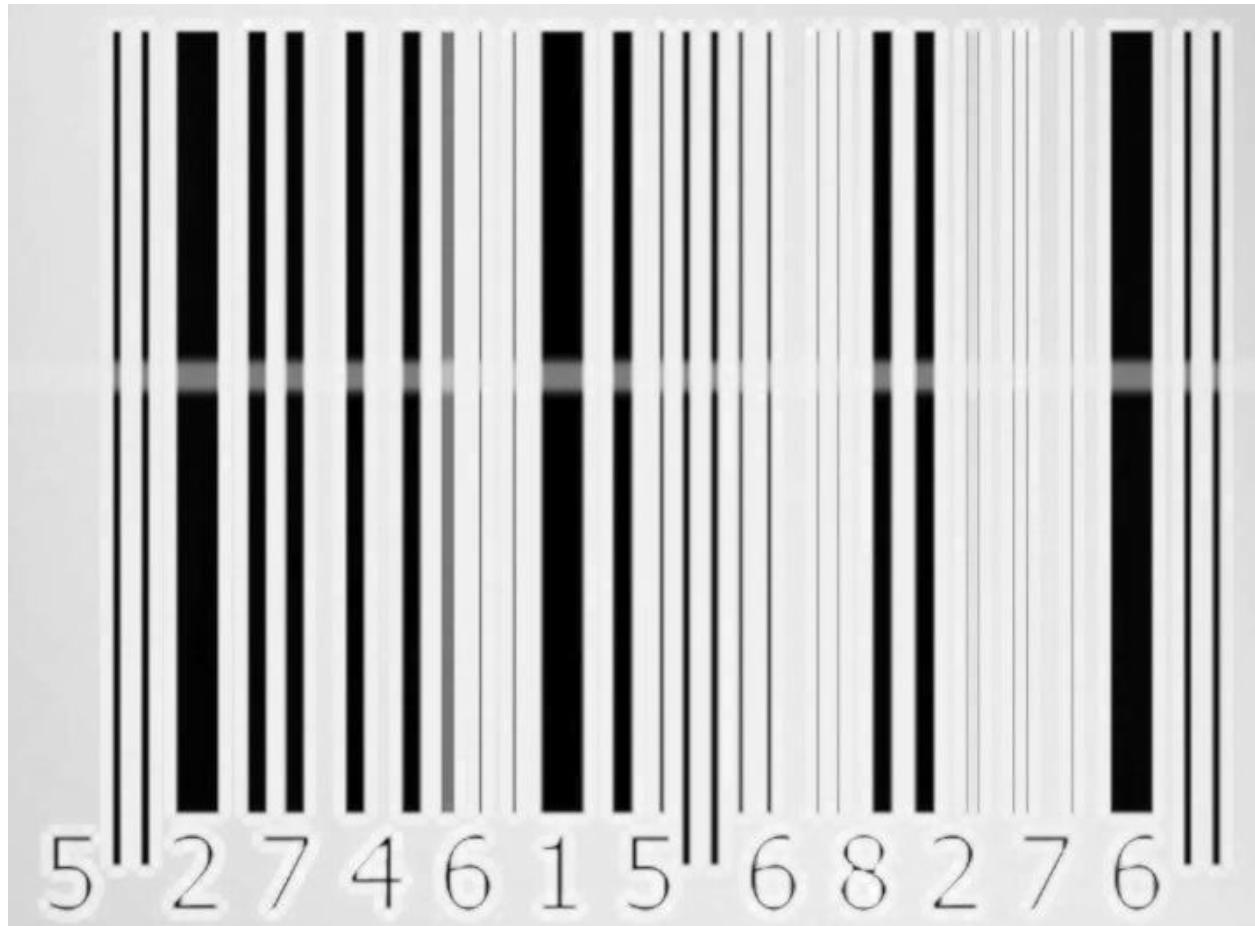
وروودی:



تصویر Dilate شده:



تصویر Dilate شده بعد از gray level adjustment



همین طور که مشاهده می شود تصویر ورودی بر اساس Structuring element گسترش یافته است و ناحیه های سفید گسترش یافته اند و قسمت هایی از ناحیه های سیاه را از بین برده اند.

قسمت f

کدهای این قسمت در gray_erosion.m و p3f.m قرار دارد. در gray_erosion.m تابع زیر را نوشته ام که عمل Erosion را انجام می دهد:

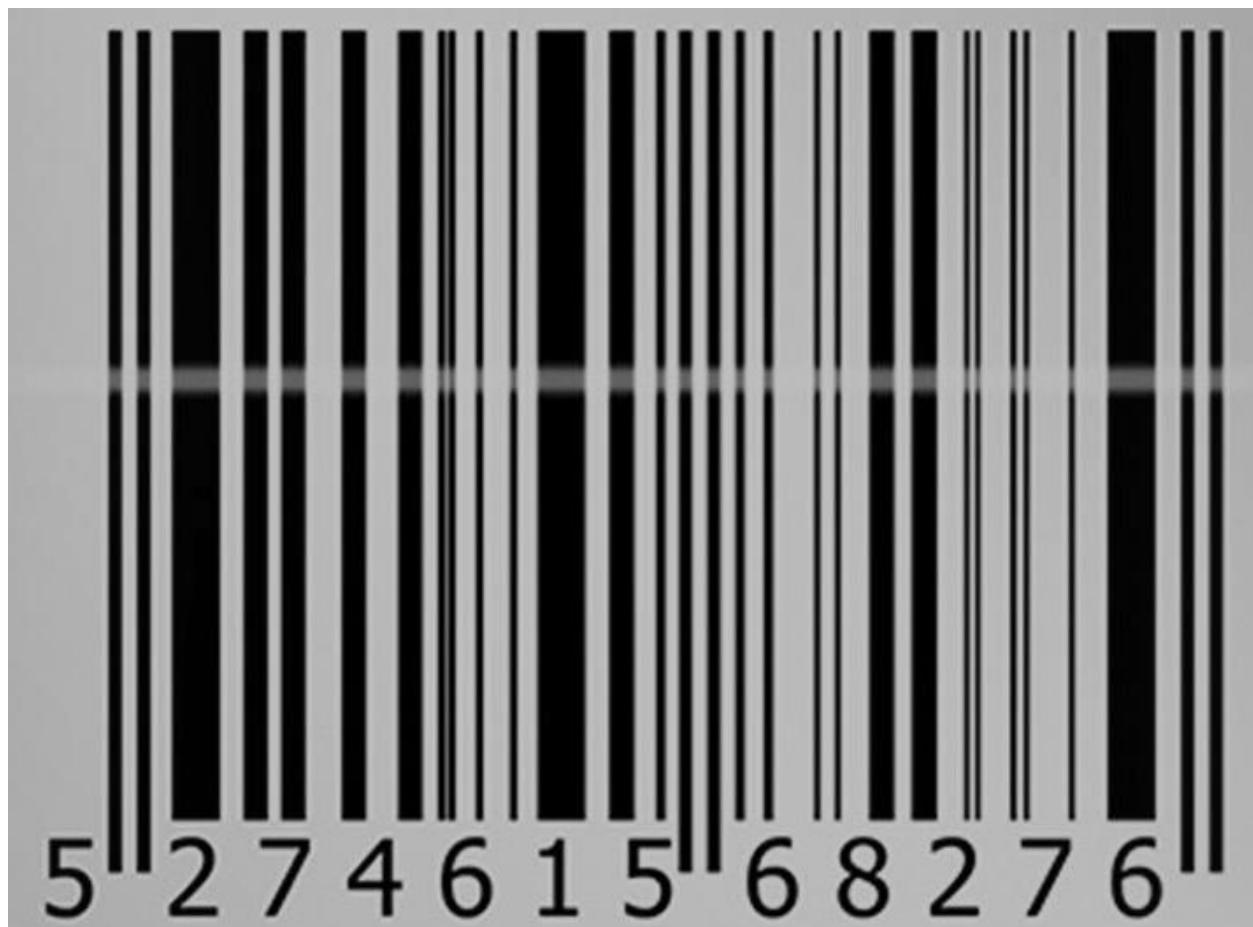
این تابع یک تصویر Gray Level و یک structuring element را دریافت می‌کند، سپس تصویر را به اندازه‌ی کافی با صفر Pad می‌کند. سپس ساختار را بر روی تصویر حرکت می‌دهد، در هر مرحله ساختار را با همسایگی از تصویر که ساختار بر روی آن قرار دارد منها می‌کند و منیم شدت روشنایی موجود در تصویر را انتخاب می‌کند و در خروجی مقدار پیکسلی که مرکز همسایگی بر روی آن قرار دارد را برابر با آن قرار می‌دهد.

در p3f.m تصویر را می‌خوانم سپس با فراخوانی تابع بالا و استفاده از ساختار زیر

```
% structuring element  
st = ones(15, 5);  
st = st .* 0.01;
```

عمل erosion را انجام می‌دهم. در زیر خروجی کد را مشاهده می‌کنید:

وروودی:



تصویر erosion شده:



همین طور که مشاهده می شود تصویر ورودی بر اساس Structuring element فرسایش یافته است و ناحیه های سفید دچار فرسایش شده اند و با استفاده از ساختار مناسب نوار سفید را حذف کرده ایم و رقم ها را در راستای عمودی پهن کرده ایم.

قسمت g

کدهای این قسمت در gray_opening.m و p3g.m قرار دارد. در gray_opening.m تابع زیر را نوشته‌ام که عمل Opening را انجام می‌دهد:

```
function opened_img = gray_opening(img, st)
    % this function calculated opening of an image
    % structuring elemnt
```

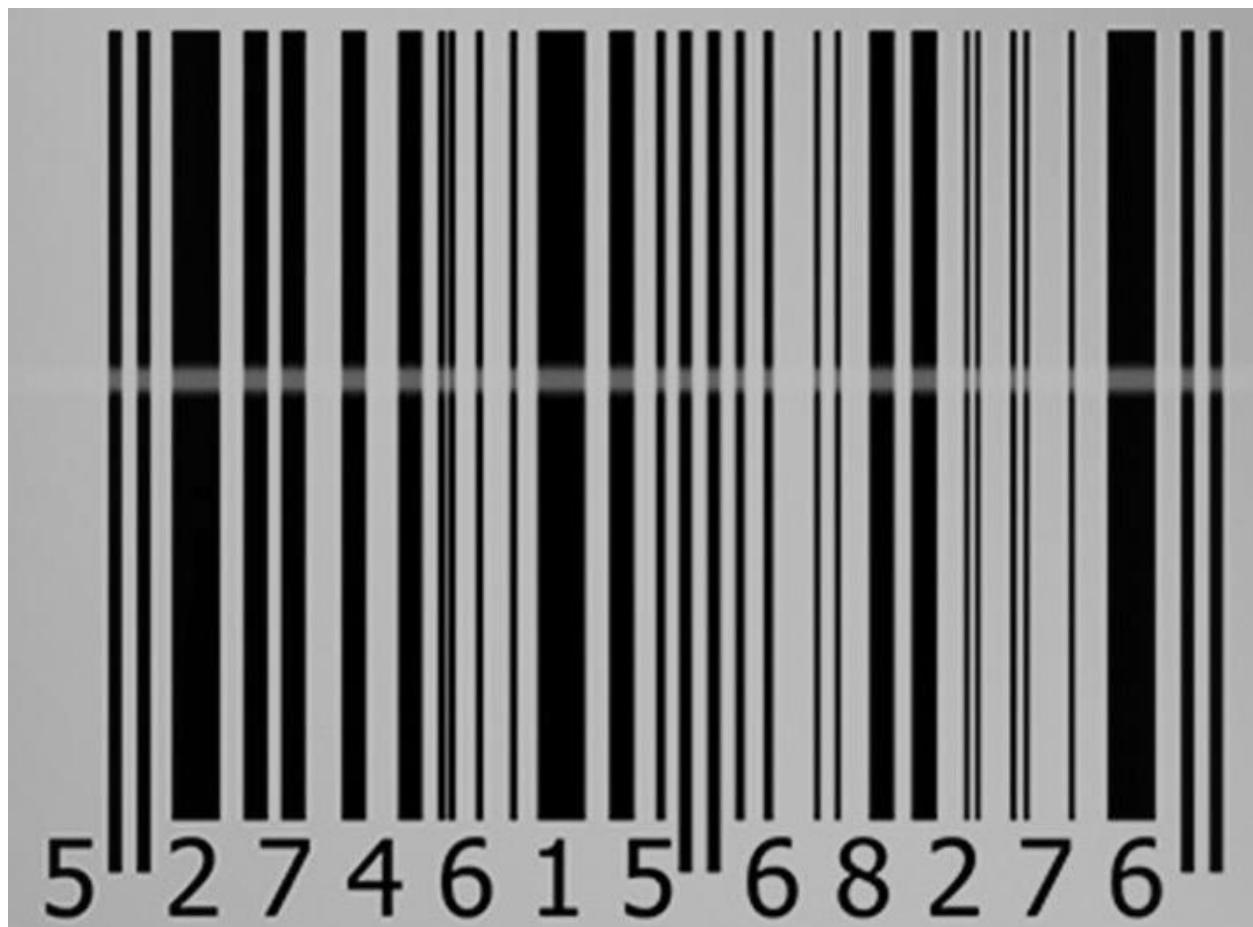
این تابع یک تصویر Gray Level و یک structuring element را دریافت می‌کند، سپس opening را با استفاده از erosion و dilation دو قسمت قبل بر تصویر اعمال می‌کند.

در p3g.m تصویر را می‌خوانم سپس با فرآخوانی تابع بالا و استفاده از ساختار زیر

```
% structuring element
st = ones(19,5);
st = st .* 0.01;
```

عمل opening را انجام می‌دهم. در زیر خروجی کد را مشاهده می‌کنید:

وروودی:



تصویر Opening شده:



تصویر gray level adjustment شده بعد از opening



همین طور که مشاهده می شود جزئیات سفید (نوار سفید عمود بر خط های سیاه) را با بهره گیری از ساختار مناسب حذف کرده ایم در حالی که عرض خط های سیاه را حفظ کرده ایم.

قسمت h

کدهای این قسمت در gray_closing.m و p3g.m قرار دارد. در gray_closing.m تابع زیر را نوشتہ ام که عمل Closing را انجام می دهد:

این تابع یک تصویر Gray Level و یک structuring element را دریافت می‌کند، سپس Closing را با استفاده از erosion و dilation دو قسمت قبل بر تصویر اعمال می‌کند.

در p3h.m تصویر را می‌خوانم سپس با فرآخوانی تابع بالا و استفاده از ساختار زیر

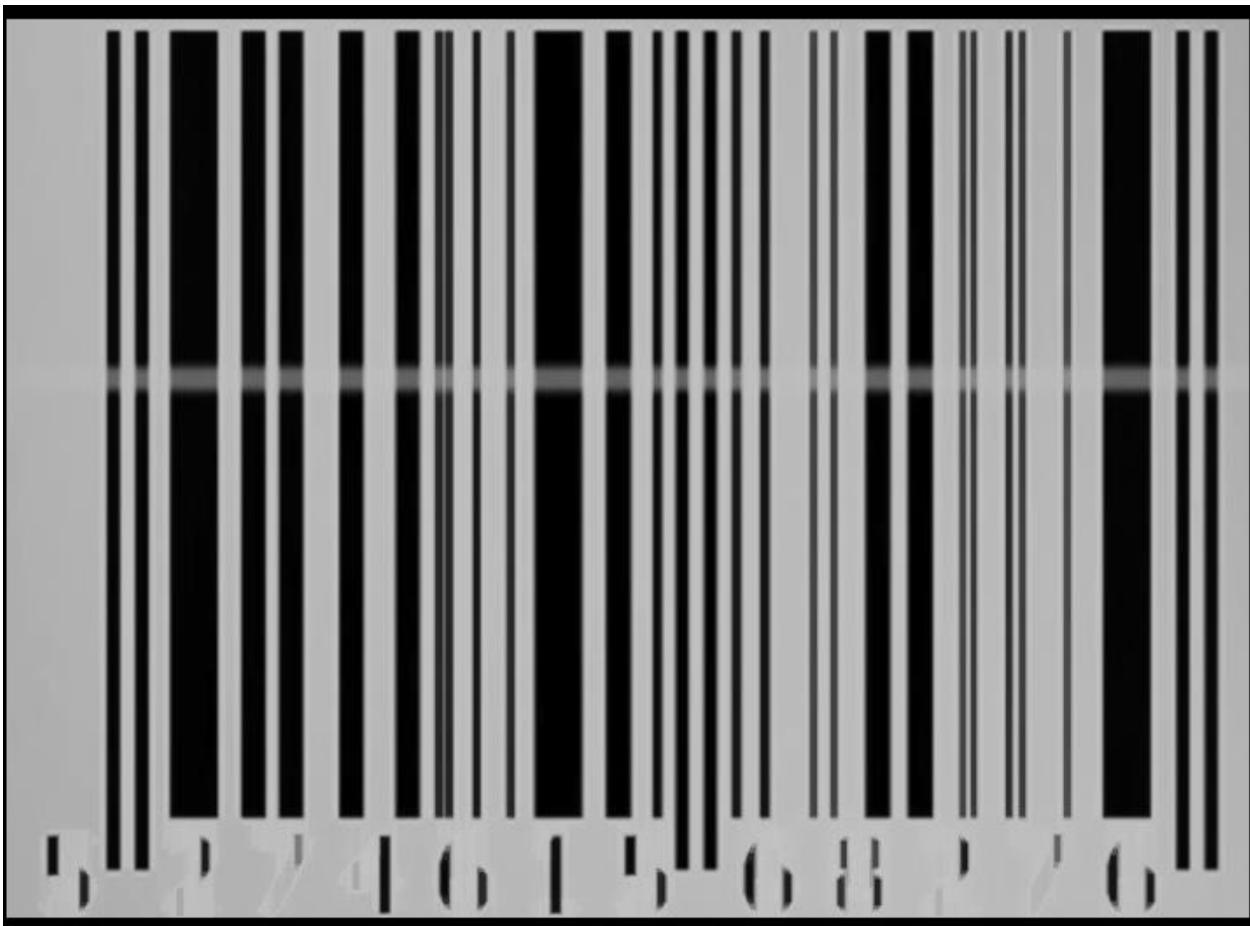
```
% structuring element  
st = ones(15, 3);
```

عمل closing را انجام می‌دهم. در زیر خروجی کد را مشاهده می‌کنید:

وروودی:



تصویر closing شده:



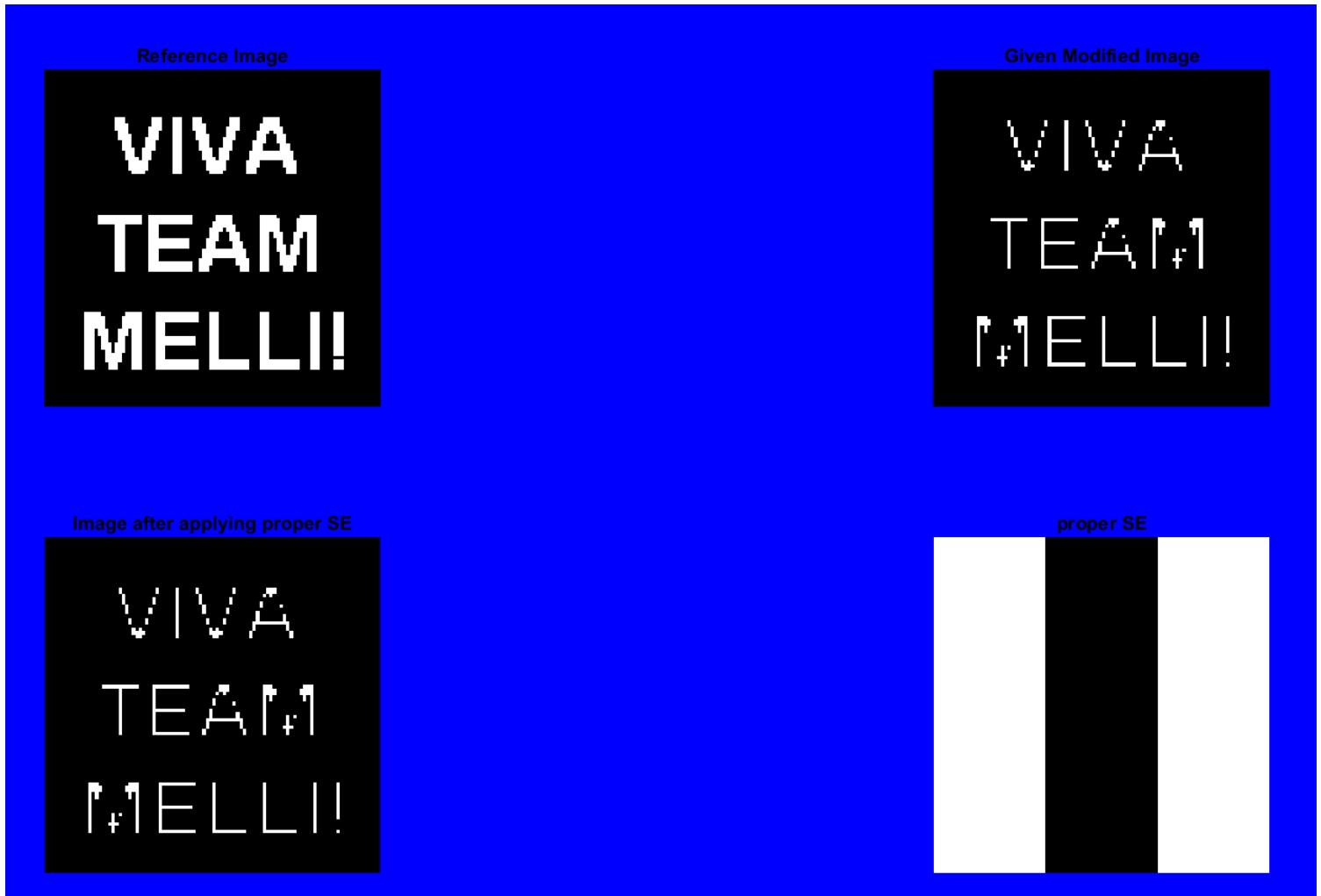
با استفاده از ساختار استفاده شده و closing، قسمت‌هایی از اعداد که عمودی نیستند را حذف کردایم.

تمرین ۴

کدهای این قسمت از سوال در پوشه‌ی p4 قرار دارد.

قسمت a

در این قسمت بر اساس سوال، باید Structuring Element را پیدا کنیم که تصویر a را تولید کند، در سوال بیان شده است که ساختار 3×3 است، از آنجایی که قسمت‌های سفید فرسایش یافته‌اند باید از Erosion رخ داده باشد. برای پیدا کردن فیلتر استفاده شده تمام 256×256 فیلتر ممکن را تست کردم (دو به توان نه). در نهایت ساختاری را انتخاب می‌کنیم که خروجی‌اش نزدیک‌تر به تصویر داده شده باشد. کد این قسمت در p4a.m قرار دارد. در زیر خروجی کد را مشاهده می‌کنید:



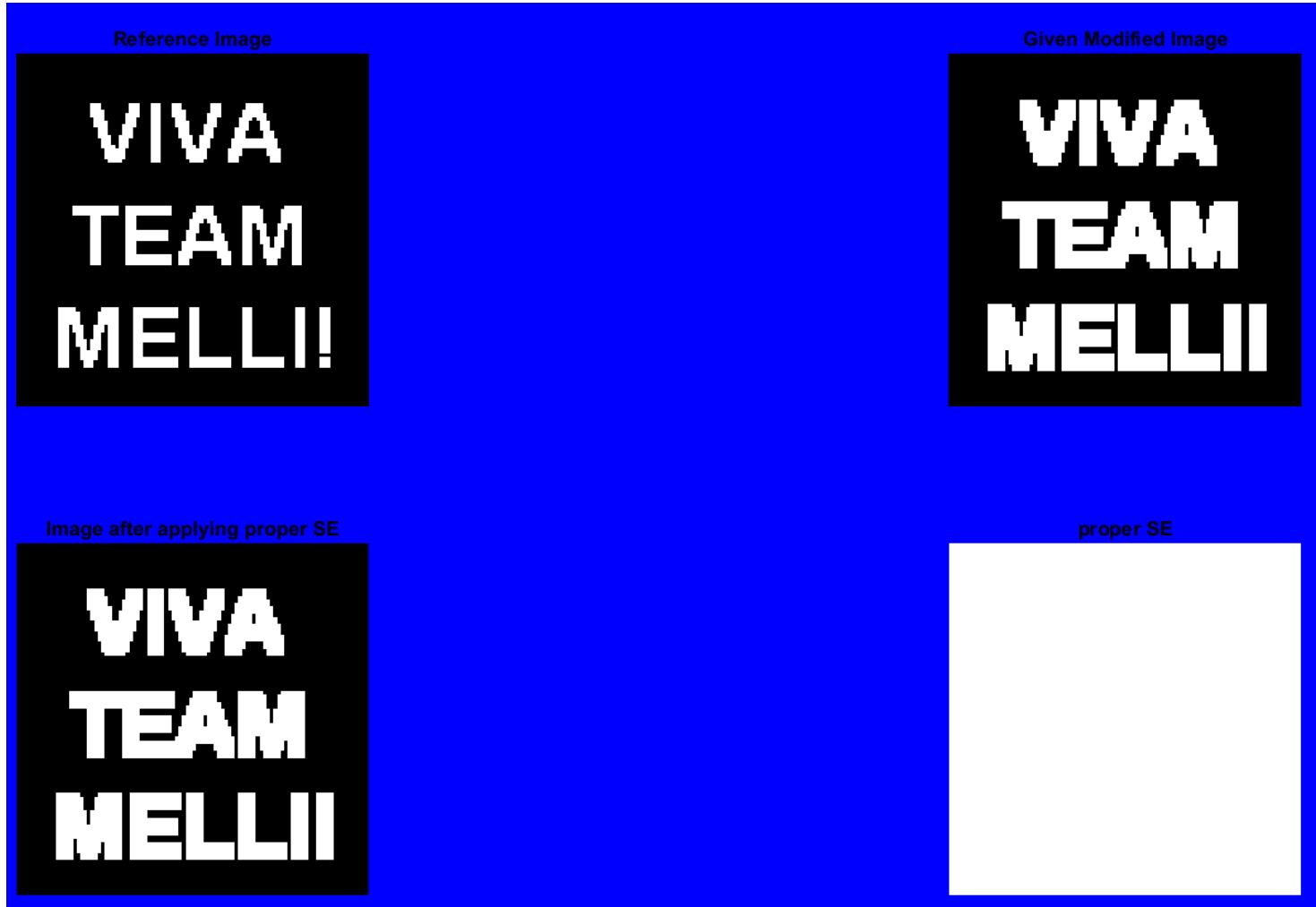
قسمت b

در این قسمت اگر به حرف T دقت کنیم مشاهده می‌کنیم که در قسمت افقی حرف T، یک خط سیاه در میانش به وجود آمده است، از آنجایی که آن قسمت از حرف T سه پیکسل است، باید از Opening با ساختاری استفاده شده باشد که وسطش صفر باشد به همین دلیل از ساختار زیر استفاده کردیم که نتایج زیر را حاصل کرد(p4b.png)



قسمت ۵

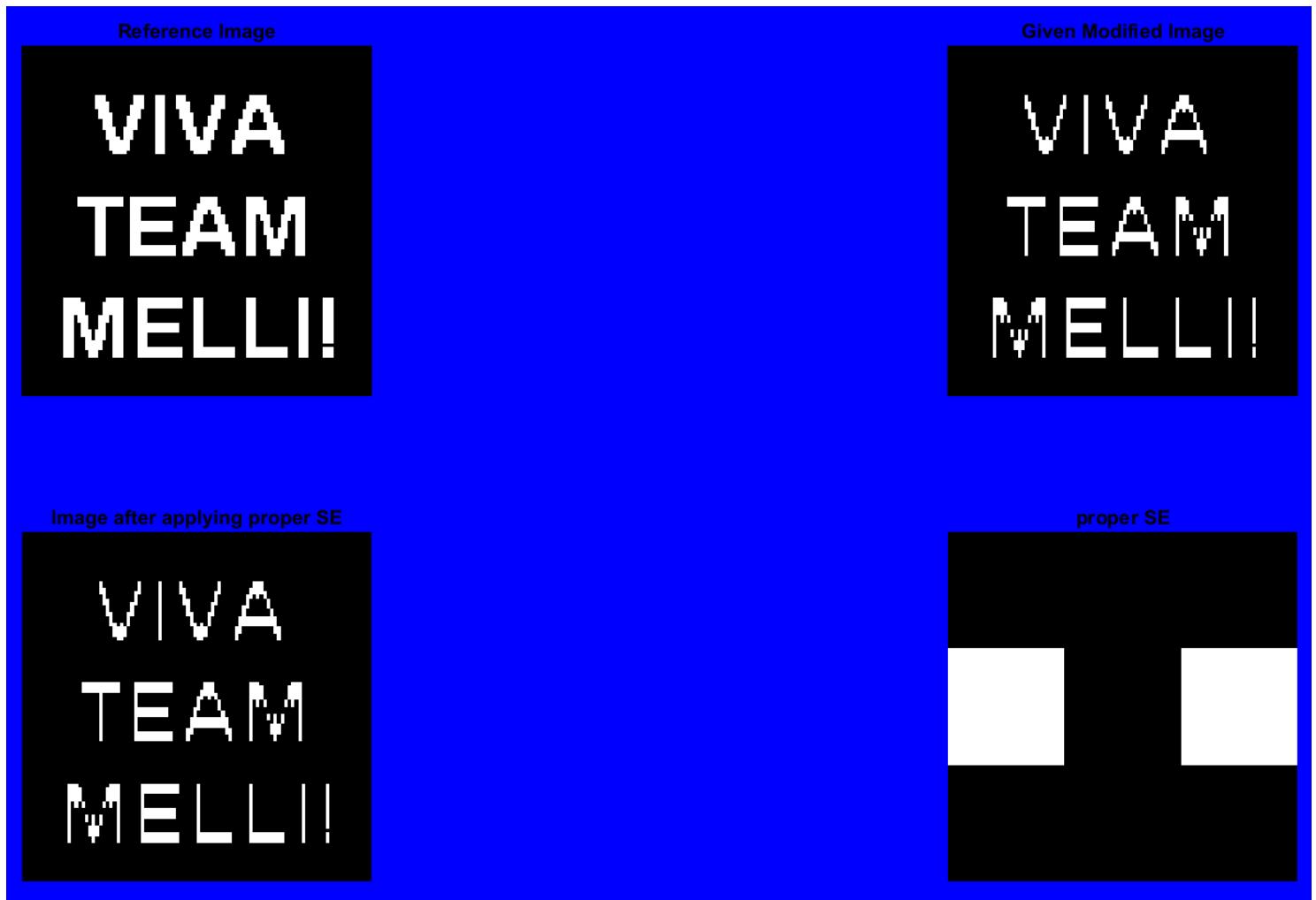
در این قسمت، در تصویر داده شده، مشاهده می‌شود که ناحیه‌ی سفید در تمام جهات گسترش یافته است، پس باید از Dilation با سختاری که تمام درایه‌های آن یک است استفاده شده باشد. برای اطمینان بیشتر در کد، تمام ۵۱۲ ساختار ممکن را تست می‌کنم. کد این قسمت در p4c.m قرار دارد. در زیر خروجی کد را مشاهده می‌کنید. (p4c.png)



قسمت ۶

کد این قسمت در p4d.m قرار دارد، همین طور که در تصویر داده شده مشاهده می‌شود، مشخص است قسمت سفید دچار فرسایش شده است. اگر به حرف T توجه کنیم مشاهده می‌کنیم که در قسمت بالایی تغییری رخ نداده است ولی در قسمت غیر بالایی آن، حرف T باریک شده است، به همین دلیل باید از Erosion با ساختاری

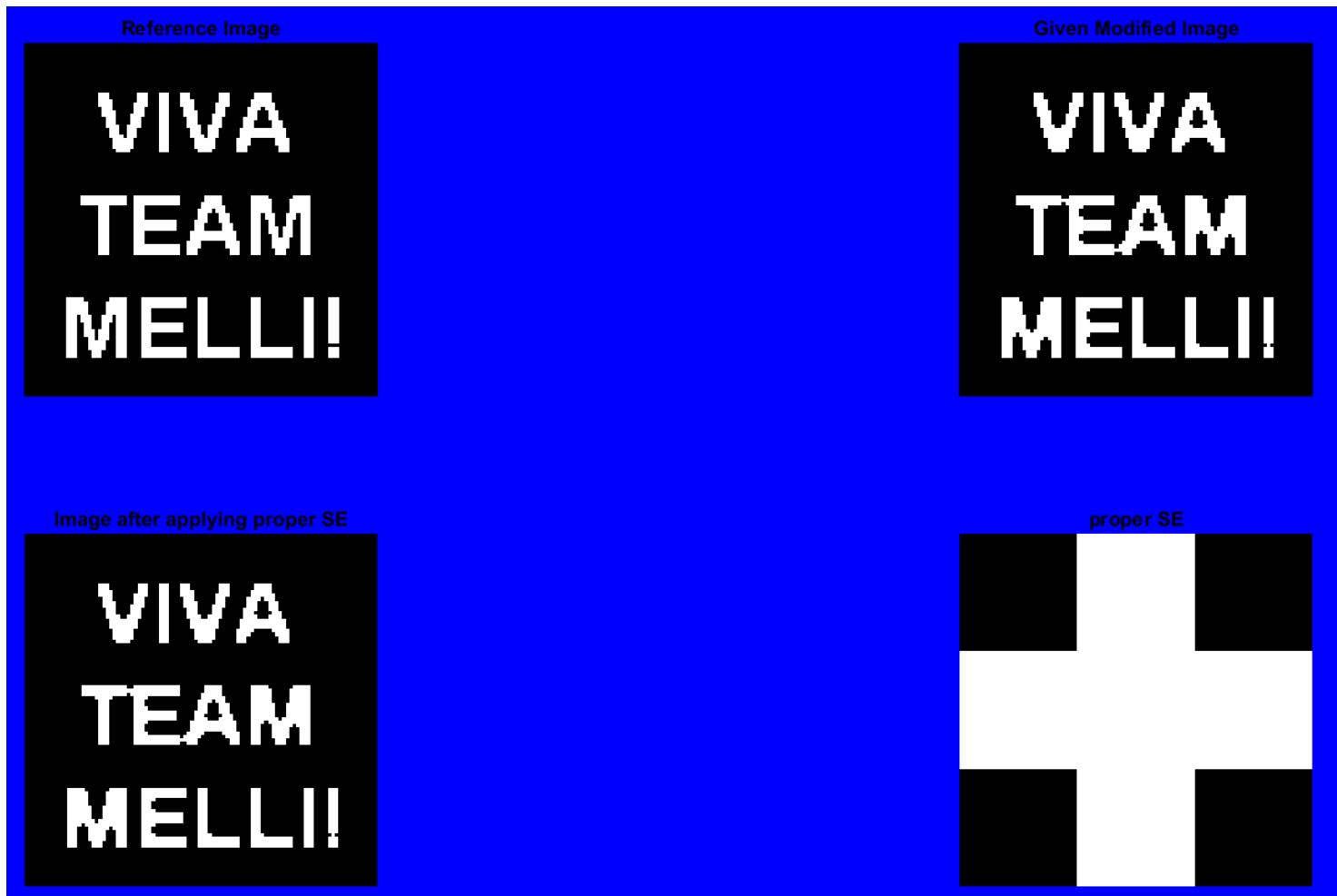
استفاده شده باشد که سطر بالایی و پایینی آن صفر باشد. همین طور اگر به قسمت دره مانند M توجه کنیم یک پیکسل در آن نقطه اضافه شده است، به همین دلیل سطر وسط بجز درایه‌ی میانی برابر با یک است. برای اطمینان بیشتر تمام ۵۱۲ ساختار ممکن را در کد تست می‌کنم. در زیر خروجی حاصل از اجرای کد را مشاهده می‌کنید(p4d.png):



e قسمت

کدهای این قسمت در p4e.m قرار دارد. اگر به حروفهای T و E در تصویر داده شده نگاه کنیم می‌بینیم که اندازه‌یشان با آن چیزی که در تصویر اصلی است یکسان است و تنها چند پیکسل در بعضی نقطه‌ها مانند اتصال

بین T و E بیشتر دارد، به همین دلیل باید عمل closing رخ داده باشد. اگر به درون حرف A نگاه کنیم در جهت‌های مختلف اثر یک ساختار + را مشاهده می‌کنیم. برای اطمینان بیشتر تمام ۵۱۲ ساختار را در کد چک می‌کنم. در زیر خروجی کد را مشاهده می‌کنید (p4e.png):



قسمت f

کدهای این قسمت در p4f.m قرار دارد، همین طور که در تصویر داده شده مشاهده می‌شود، قسمت‌های سفید گسترش یافته‌اند به همین دلیل عملیات dilation رخ داده است. و این گسترش در راستای افقی است به همین

دلیل سطر وسط ساختار باید یک باشد و سایر سطرهای صفر باشند. برای اطمینان در کد بیشتر تمام ۵۱۲ ساختار ممکن را تست می‌کنم. در زیر خروجی کد را مشاهده می‌کنید : (p4f.png)

Reference Image



Given Modified Image



Image after applying proper SE



proper SE



تمرین ۵

کدهای این قسمت پوشه‌ی p5 قرار دارد.

قسمت a

کد این قسمت در p5a.m قرار دارد. در این قسمت باید تعداد پیکسل‌های مرزی را بشماریم. ابتدا تصویر را به صورت باینری در می‌آوریم، برای این کار در تصویر رنگ زرد (0, 242, 255) را برابر صفر می‌گذاریم و سایر نقطه‌ها را برابر با یک قرار می‌دهیم.

بعد از آن با استفاده از رابطه‌ی زیر مرز را به دست می‌آوریم:

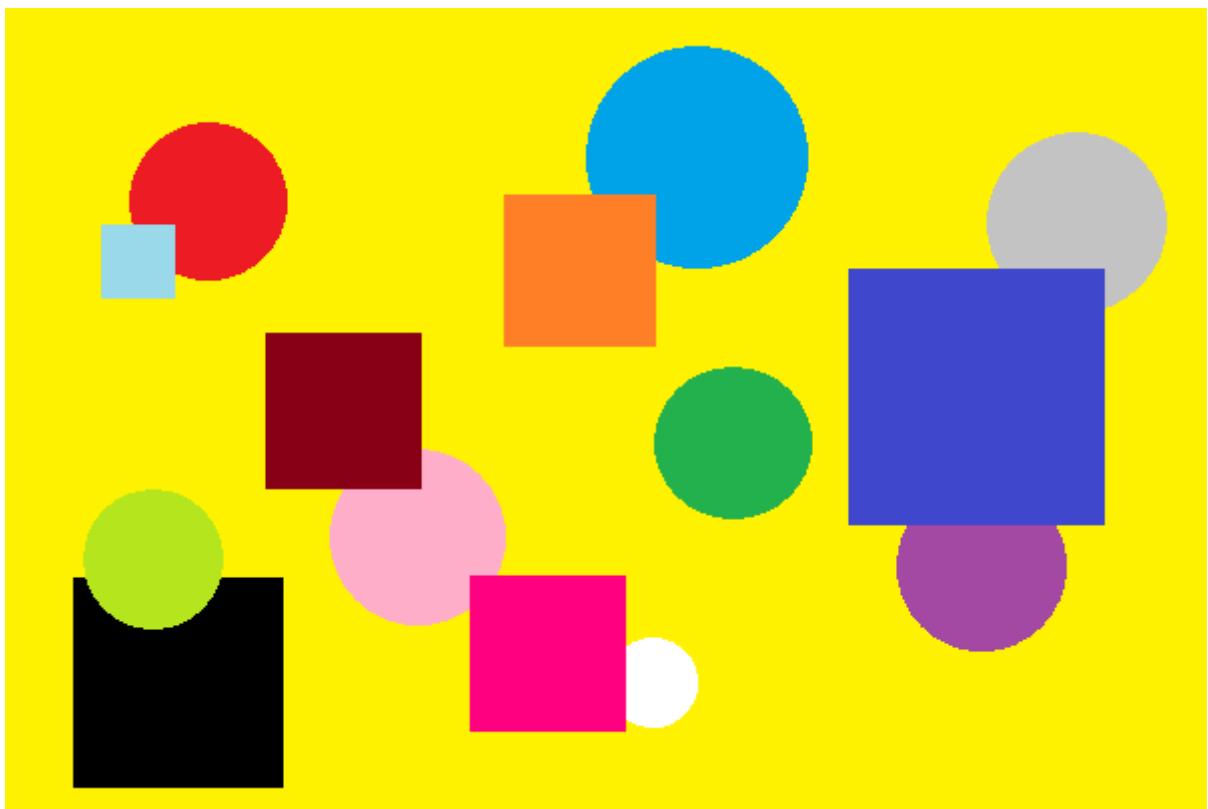
$$\beta(A) = A - (A \ominus B)$$

از ساختار زیر استفاده می‌کنیم:

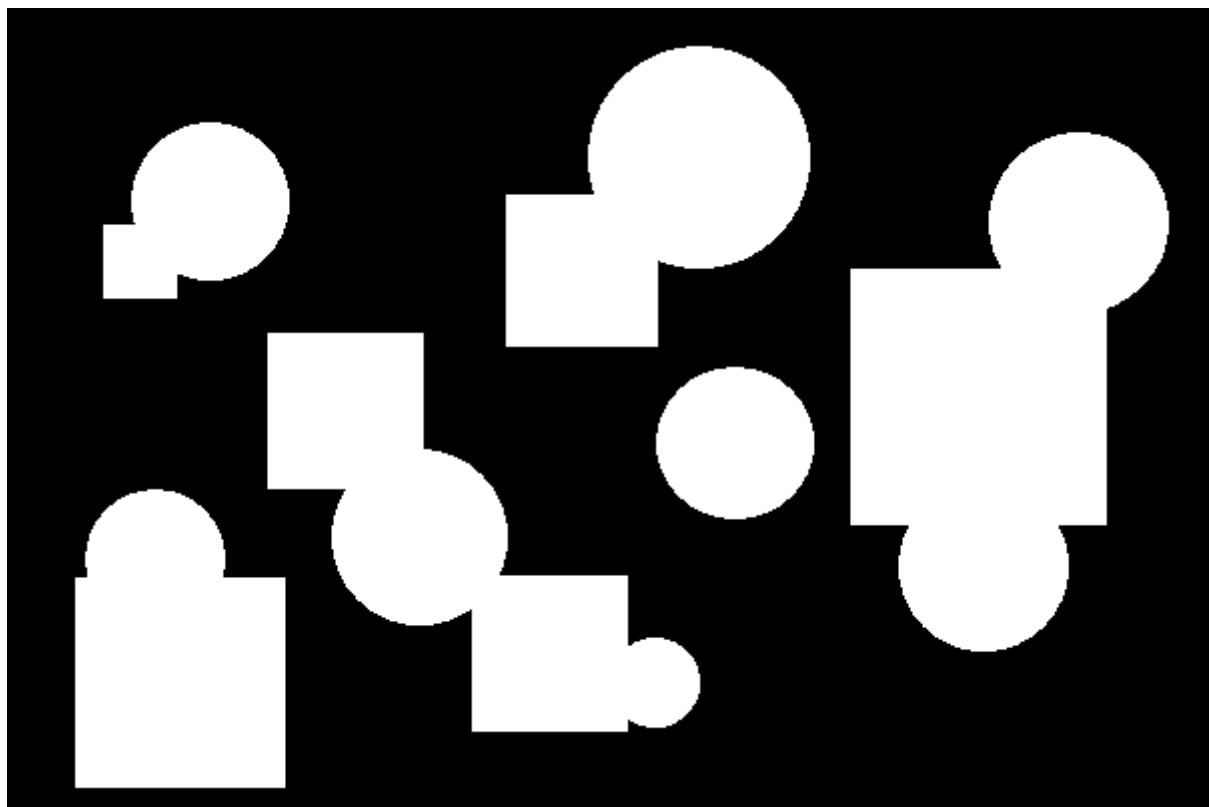
```
% structuring element  
st = strel('arbitrary', ones(3, 3));
```

خروجی‌های کد را در زیر مشاهده می‌کنید:

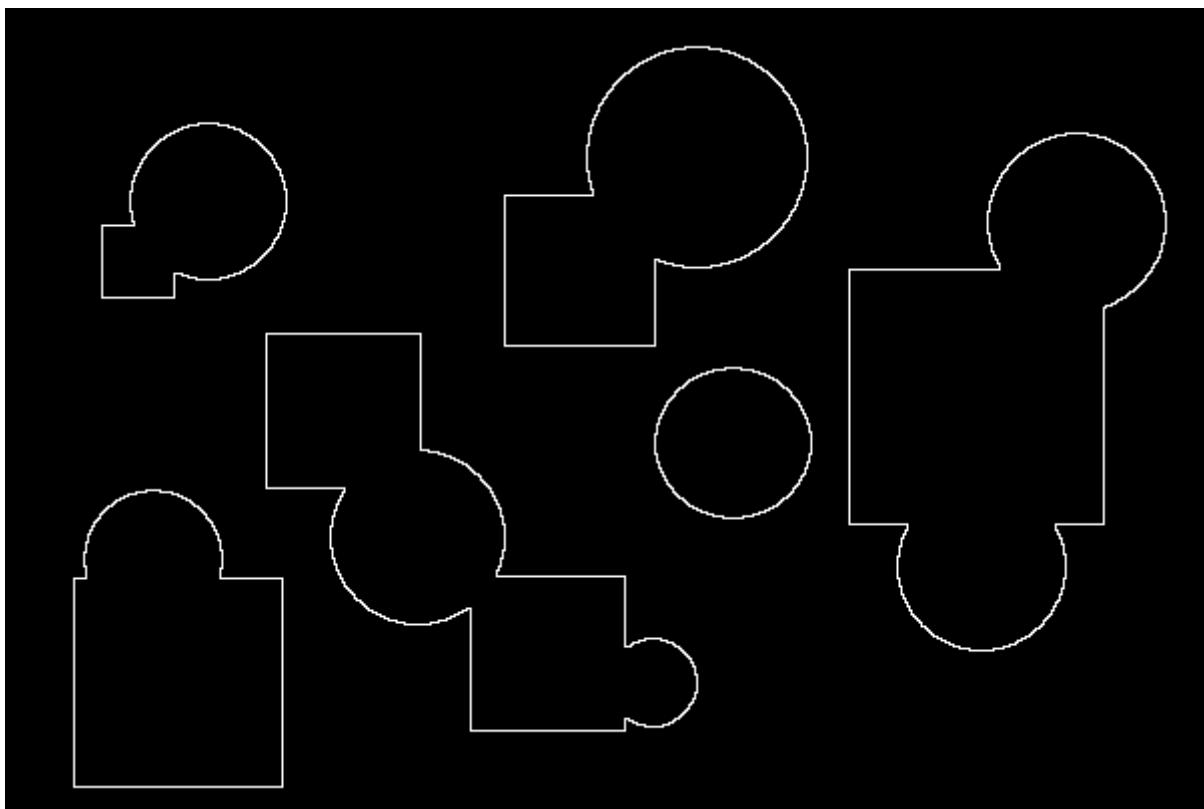
وروودی:



تصویر سیاه و سفید:



تصویر مرز به دستآمده (p5a.png)



تعداد پیکسل‌های مرز:

>> p5a

Number of bounccary pixels:

3540

قسمت b

کدهای این قسمت در p5b.m قرار دارد. این سوال را به دو صورت مختلف متوجه شدم به همین دلیل هر دو حالت را پیاده‌سازی کرده‌ام:

حالت اول:

در حالت اول، فرض کرده‌ام سوال خواسته است که نقاط دایره و ستاره مانند روی جام را با الگوریتم region که به صورت زیر است حذف کنیم. برای این کار از روش زیر استفاده می‌کنیم:

$$\begin{array}{ccc} \text{expansion} & & \text{stop at the boundary} \\ Y_0 = P & \downarrow & \nearrow \\ Y_k = (Y_{k-1} \oplus B) \cap X^c, k=1,2,3\dots & & \end{array}$$

از هر کدام از ناحیه‌هایی که می‌خواهیم آن را پر کنیم یک پیکسل را انتخاب کرده‌ام، سپس با استفاده از روش بالا آن ناحیه را پر می‌کنیم و این کار را برای تمام ناحیه‌ها تکرار می‌کنیم. خروجی این کار را در زیر مشاهده می‌کنید:

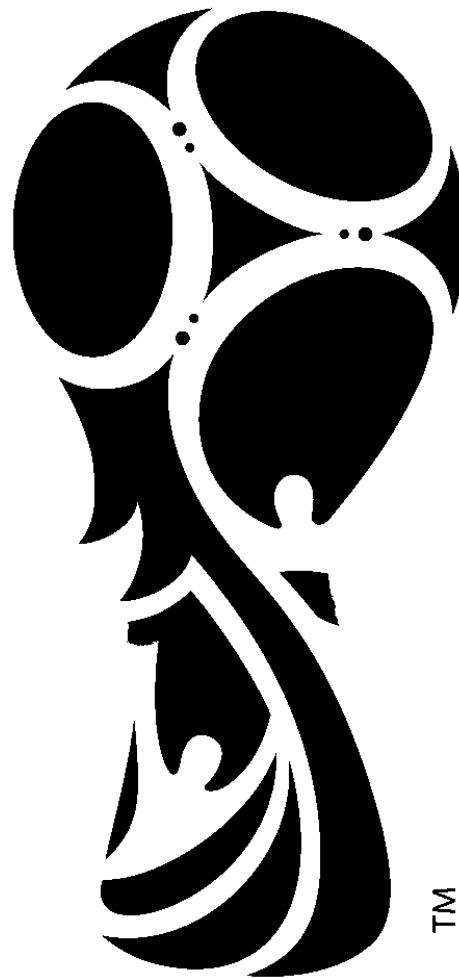
وروودی:



TM

**FIFA WORLD CUP
RUSSIA 2018**

: (p5b-manner-1.png) خروجی حالت یک



**FIFA WORLD CUP
RUSSIA 2018**

حالت دوم:

در این حالت فرض کردہ‌ام که سوال خواسته است علاوه بر ناحیه‌های ستاره و دایره مانند سفید بر روی جام، طرح‌های انسان مانند را نیز حذف کنیم، به همین دلیل از opening با یک ساختار $65*65$ که همه‌ی درایه‌های آن برابر با ۱ است استفاده می‌کنیم. در زیر خروجی آن را مشاهده می‌کنید (p5b-manner-2.png):



همین طور که در بالا اشاره شد، این سوال را به دو حالت مختلف متوجه شدم به همین دلیل هر دو روش را پیاده کردم، در زیر خروجی هر دو روش را مشاهده می‌کنید:

ورودی



FIFA WORLD CUP
RUSSIA 2018

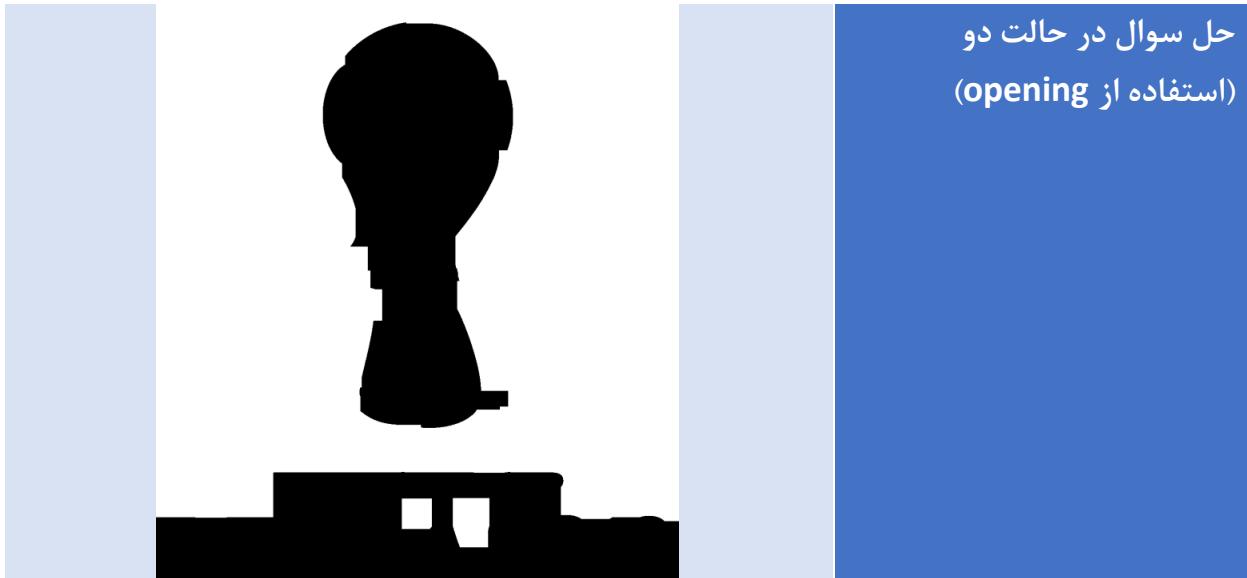
حل سوال در حالت یک

(استفاده از Region Filling)



FIFA WORLD CUP
RUSSIA 2018

حل سوال در حالت دو (opening از استفاده)

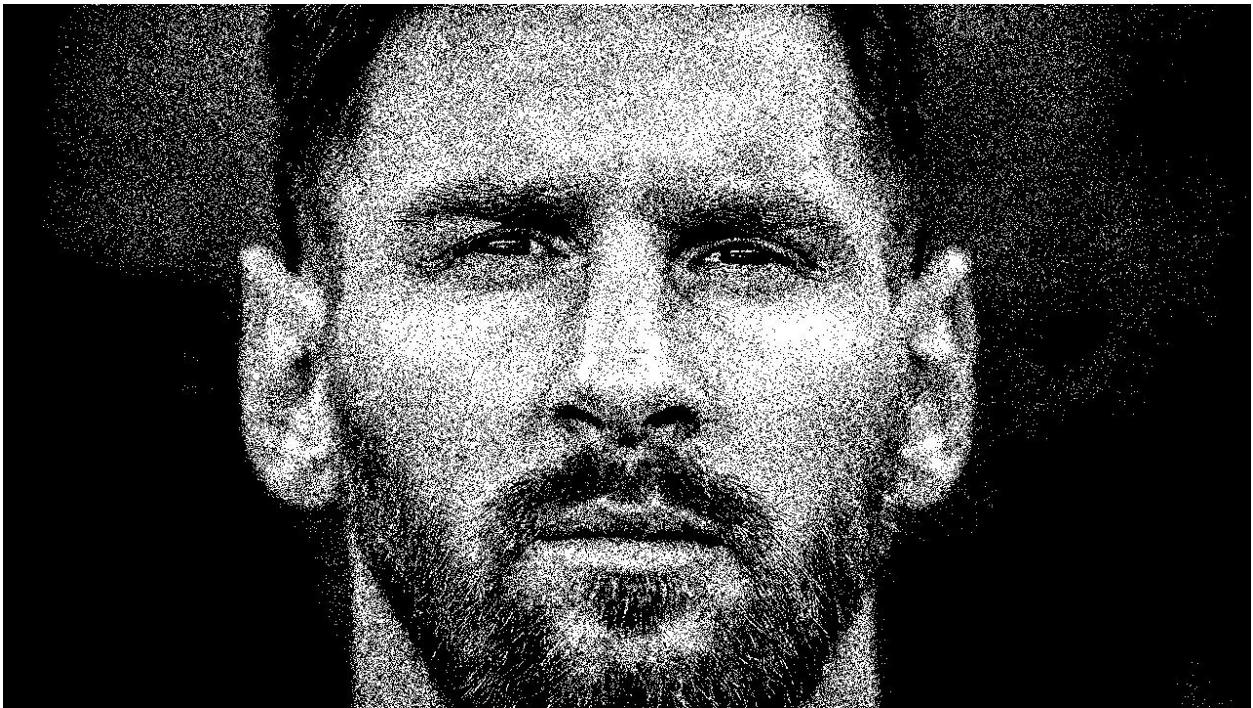


قسمت C

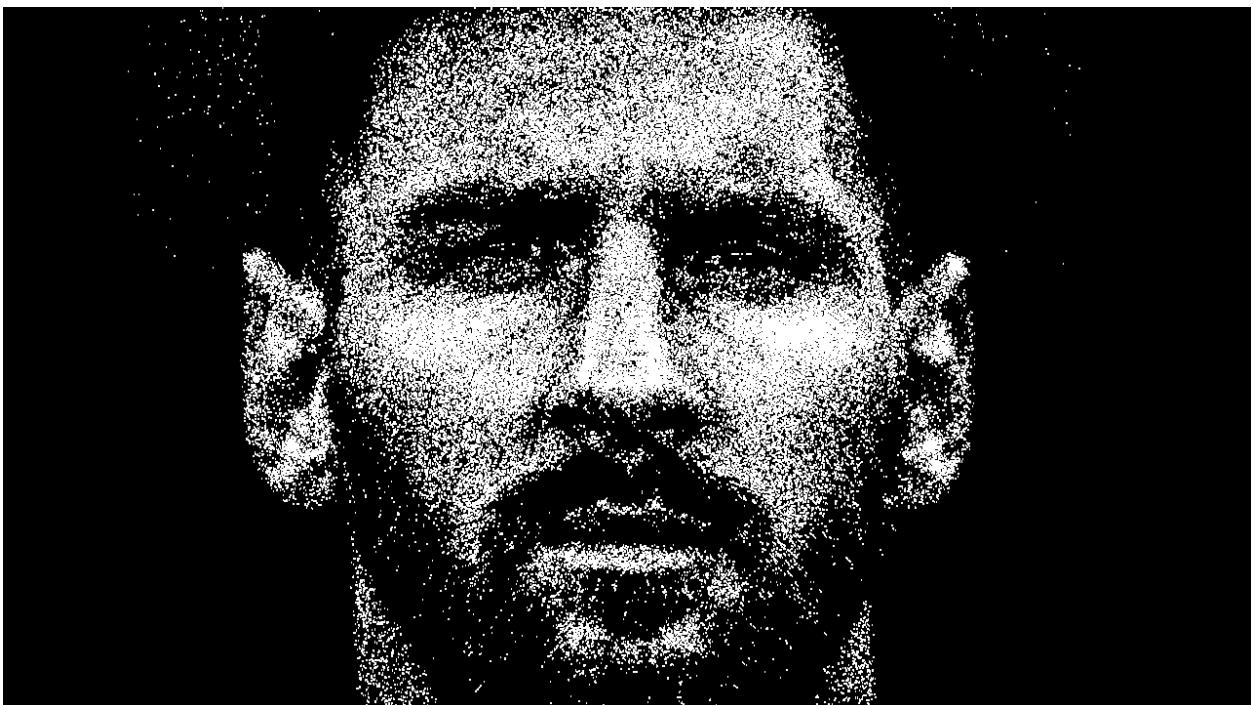
کد این قسمت در p5c.m قرار دارد، این سوال را با **دو روش مختلف** حل کرده‌ایم، در روش اول Opening را بر کل تصویر با ساختار $\begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix}$ اعمال کرده‌ایم و سپس Opening را بار دیگر با ساختار $\begin{matrix} 1 \\ 1 \end{matrix}$ اعمال کرده‌ایم.

در روش دوم Opening را با ساختار $\begin{matrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{matrix}$ فقط در ناحیه‌های اطراف لیون مسی اعمال کرده‌ایم. در زیر خروجی‌های کد را مشاهده می‌کنید:

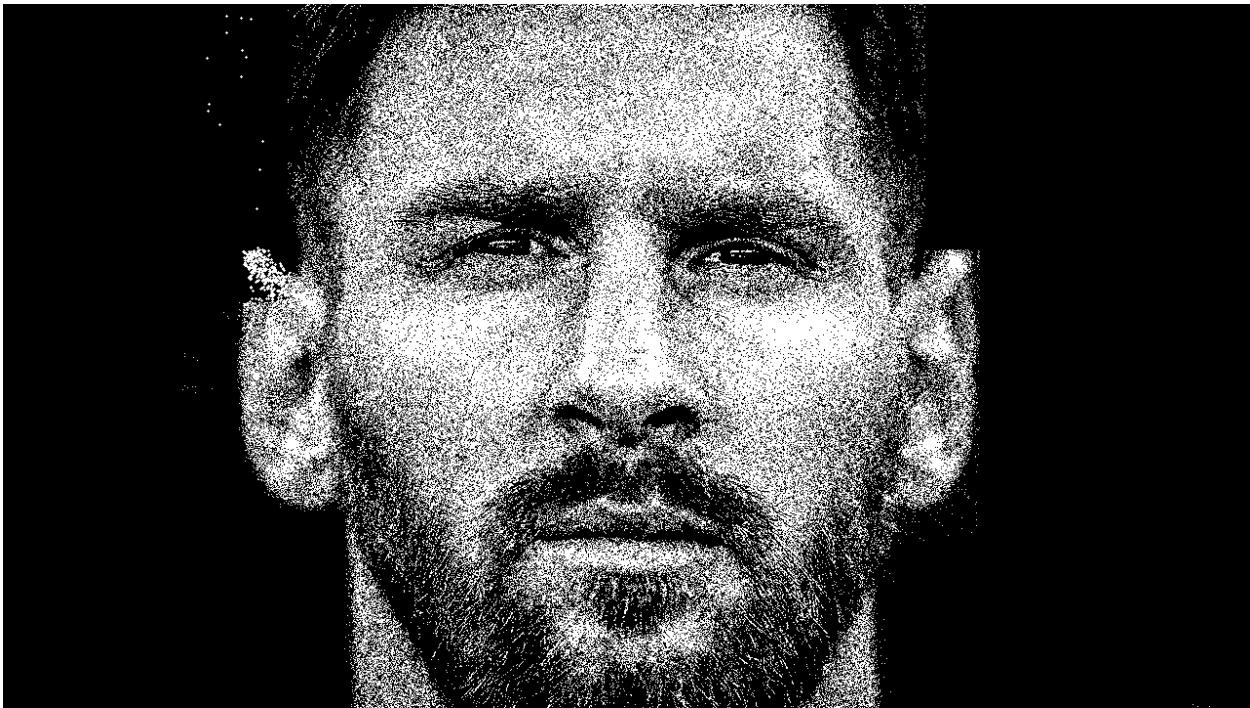
ورودی:



خروجی را روش اول (p5c-manner-1.png)



خروجی با روش دوم (p5c-manner-2.png)



قسمت d

کد این قسمت در p5d.m قرار دارد. برای پر کردن فاصله‌های سفید بین ناحیه‌های سیاه از opening با ساختار ۷*۷ که تمام درایه‌های آن برابر با یک است استفاده می‌کنیم. در زیر خروجی کد را مشاهده می‌کنید:

وروودی:



خروجی (p5d.png)



قسمت e

کدهای این قسمت در p5e.m قرار دارد. ابتدا تصویر و فیلتر را negative می‌کنم تا کاراکترها به رنگ سفید بشوند و پس زمینه سیاه بشود. برای حل این سوال راههای مختلفی را تست کردم، ولی کاراکتر e داده شده با های درون تصویر مقداری متفاوت است، برای رفع این مشکل کاراکتر e داده شده را erode می‌کنم. سپس erosion را بر تصویر داده شده با کاراکتر e اعمال می‌کنم، که بر اثر این کار شش نقطه‌ی سفید باقی می‌ماند که برابر با محل و تعداد e ها در تصویر است. در زیر خروجی‌های کد را مشاهده می‌کنید:

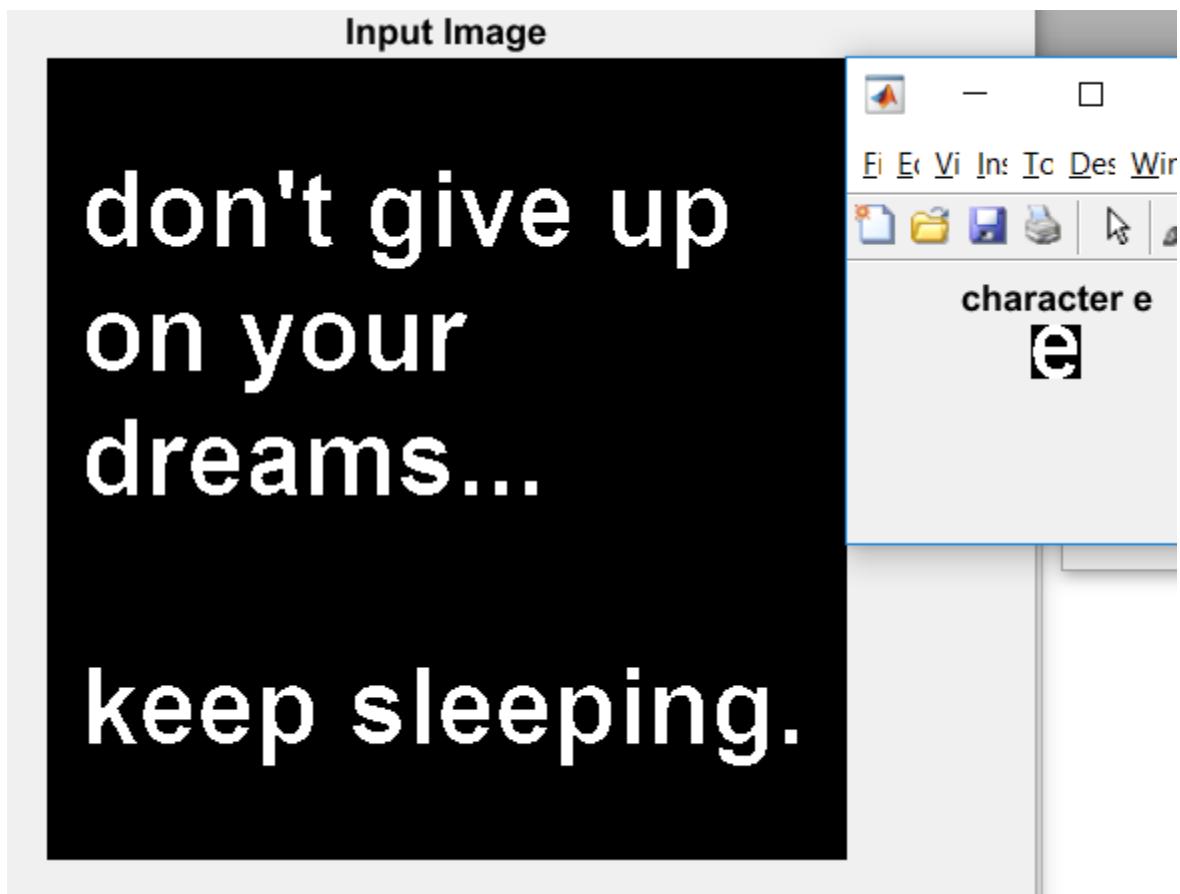
ورودی‌ها:

**don't give up
on your
dreams...**

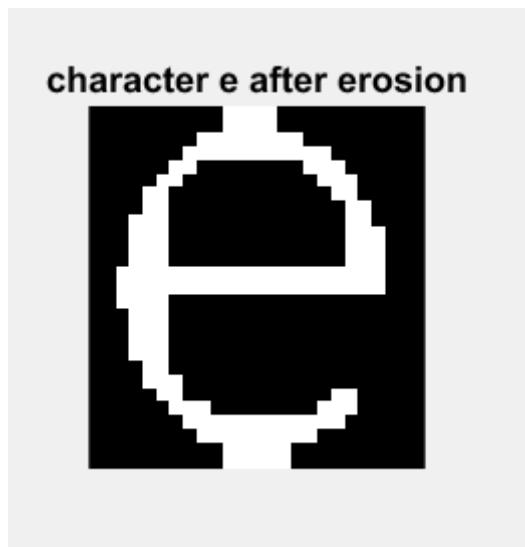
keep sleeping.

e

تصویرهای ورودی بعد از تبدیل یک به صفر و بر عکس:



اعمال erosion با ساختار 3×3 که تمام درایه‌های آن برابر با یک است:



اعمال erosion بر تصویر متن با ساختار کاراکتر:



همان‌طور که در تصویر دیده می‌شود ۶ نقطه‌ی سفید قابل مشاهده است که برابر با e ها در تصویر است.

```
>> p5e
number of e:
6
```

کدهای این قسمت در p5f.m قرار دارد. ابتدا تصویر را negative می‌کنم سپس عمل closing را با دو ساختار زیر اعمال می‌کنم و بعد دوباره تصویر را negative می‌کنم.

1
1
1

1
1
1

از این جهت که در زبان پارسی حروفها به صورت پایین به بالا و یا قطری از سمت راست به چپ هستند این دو ساختار را انتخاب کرده‌ام در زیر خروجی کد را مشاهده می‌کنید:

وروودی:

چو بخت عرب بر عجم پیروکشت
 جهان را دگرگونه شد رسماً دراد
 زمی نشز و نبه از چک رفت
 ادب خوازکشت و همزده بال
 جهان پرشد از خوی اهی سی
 کنون بی عمان را پس حاجت بی
 که در بزم این هر زه گردان خام
 بحاتی که خشکیده باشد گیا
 چو با تخت بزر برادر شود
 ز شیر شتر خوردان و سوار
 که تاج کیان کند آرزو
 درین است ایران که ویران شود

همه روز از اینان تیره گشت
 تو کوئی تابد و گر مسرد ندا
 ذکل عذر و صحنی بفریانک رفت
 به ستمد اندر شه را پر و بال
 زبان مسرود زیده و دل و شمنی
 کران را چه سودی ز آوازی فی
 گناه است و گرگردش آریم خام
 هدر دادن آب باشه گناه
 هد نام بوبکر و عمر شور
 عرب را بجانی رسیده است کار
 تقویت تو ای چرخ گردان تقویت
 کنام پکان و شیران شور

: خروجی()

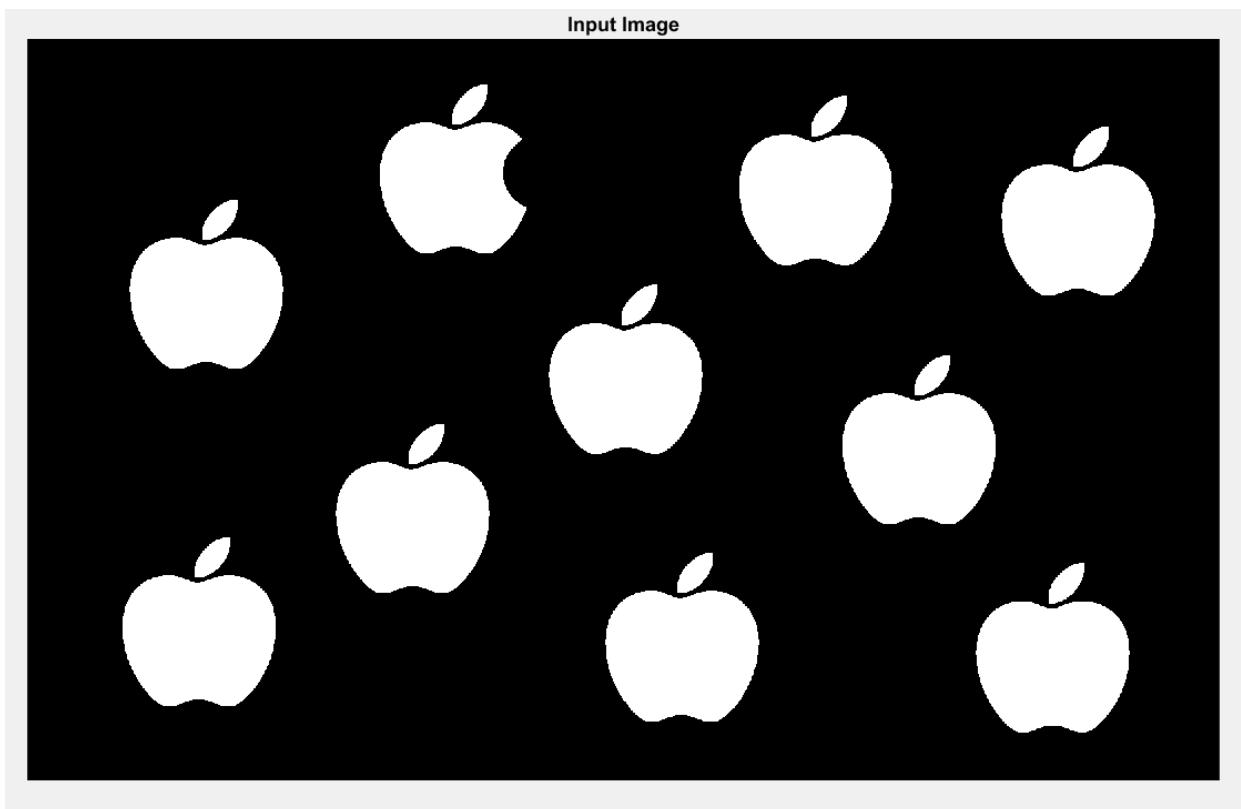
بود روز ایرانیان تیرگشت
 جهان را دگرگونه شد می‌زد
 زی نشاد خبر از چنگ رفت
 او بخواست و هم‌شدو بال
 جهان پرشد از خیابانی
 گذن بل عمان را پمچ است بی
 که در می‌این هر زده گردان خام
 بجانی که ملکیده باشد گیا
 چو باخت بزر برادر شود
 ز شیر شتر خوردن و سوار
 که تاج کیان گند آرزو
 درین است ایران که ویران شود

بود روز ایرانیان تیرگشت
 تو کوئی تابد و گرمه می‌زد
 ذکی صدر مسیح فریانگ رفت
 به سمته انبوشه را پر و بال
 زبان صمره و زیده و دل و شن
 کران را په مودی ز آوازی لی
 گناه است و گردش آریم خام
 هر دادن آب باشد گناه
 بد نام بوجلد و سر ثور
 عرب را بجانی رسیده است کار
 خوب را ای چرخ گردون خود
 کلام پیکان و شیران شود

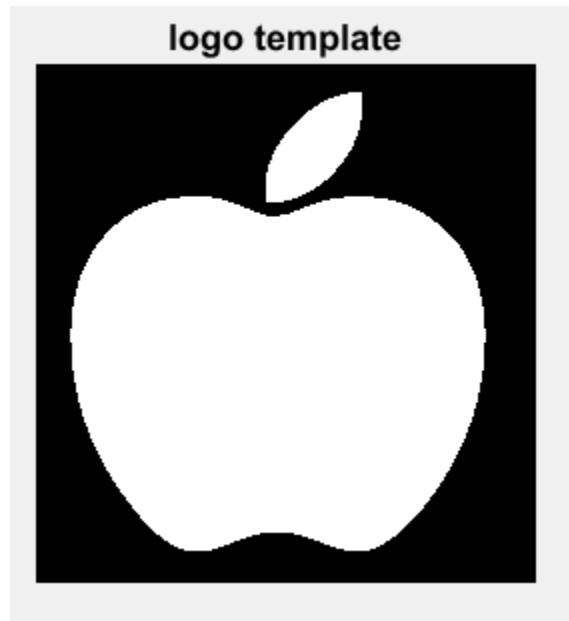
سوال گفته است سیبی که ناقص است (لوگوی اپل) را پیدا کنید. از آنجایی که گفته است شکل دارای خرابی را پیدا کنید می‌شود از شکل سالم استفاده کرد. ابتدا تصویر را سیاه سفید می‌کنم و بعد از آن تصویر را نگاتیم می‌کنم تا سیب‌ها سفید شوند. در ادامه یک سیب سالم را به عنوان الگو انتخاب می‌کنم. از آنجایی که ممکن است سیب‌های سالم در بعضی از پیکسل‌های لبه باهم متفاوت باشند، الگوی سیب سالم را با عمل erosion کمی فرایش می‌دهم. سپس تصویر را با آن opening می‌کنم، در این صورت اگر سیبی مشکلی داشته باشد به کلی حذف می‌شود. در ادامه با کم کردن تصویر اصلی و تصویر opening شده، سیب‌های (های) دارای نقص را باقی می‌ماند. تصویر سیاه سفید به دست آمده را در تصویر رنگی ضرب می‌کنم تا تصویر سیب نهایی به دست بیايد:

تصویر ورودی:

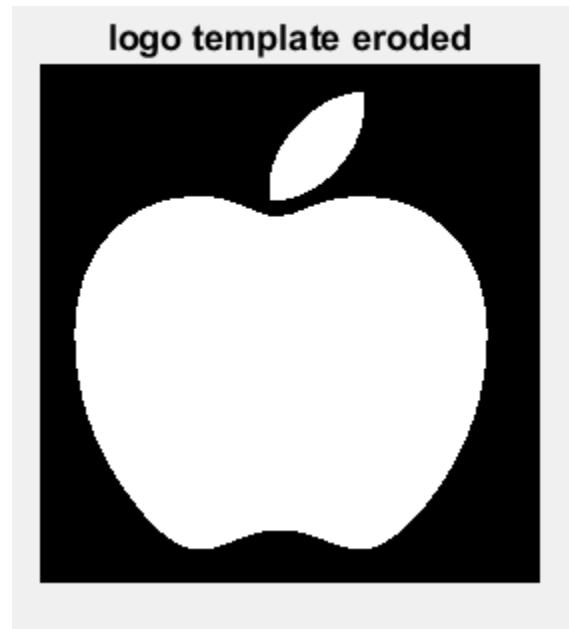
تصویر ورودی بعد از سیاه و سفید کردن:



انتخاب یک سیب سالم:

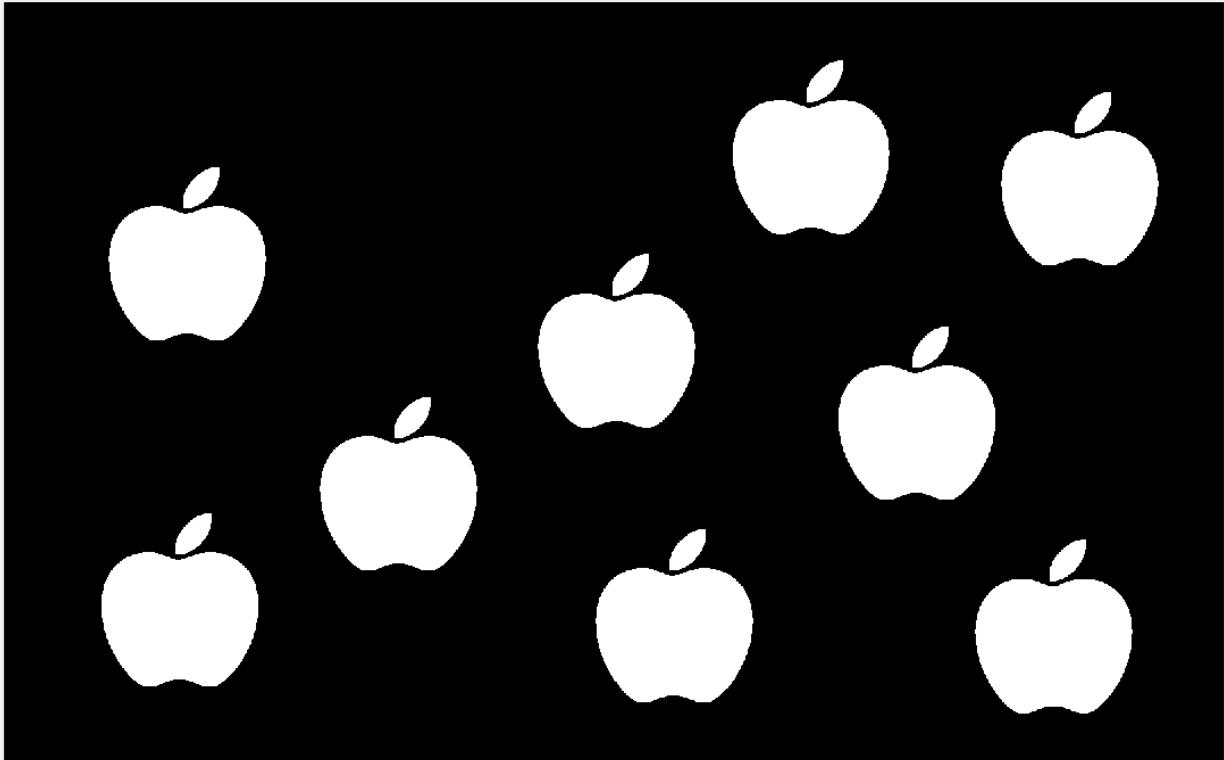


سیب سالم بعد از erosion با فیلتر 2×2 با درایه‌های یک:

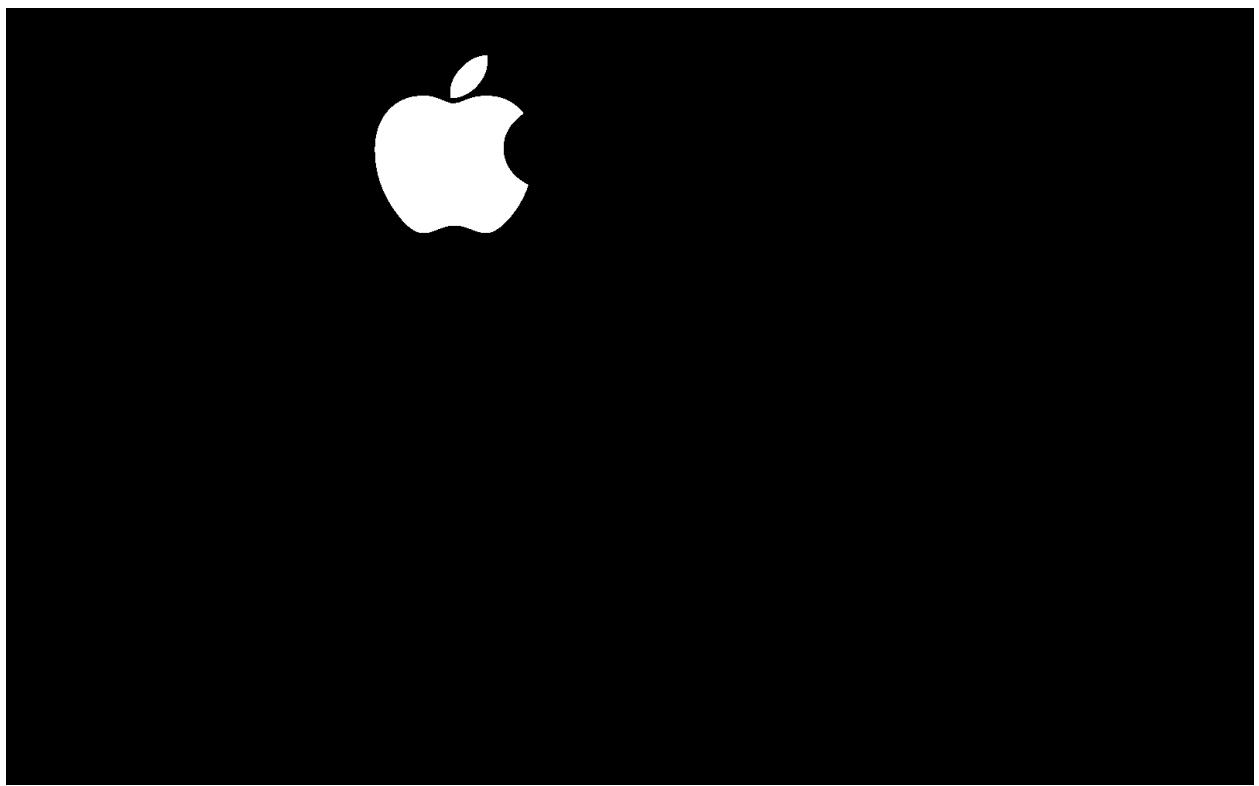


:opening بعد از عمل تصویر

Image after opening



: (p5g-bw.png) opening تصویر بعد از کم کردن تصویر و حاصل



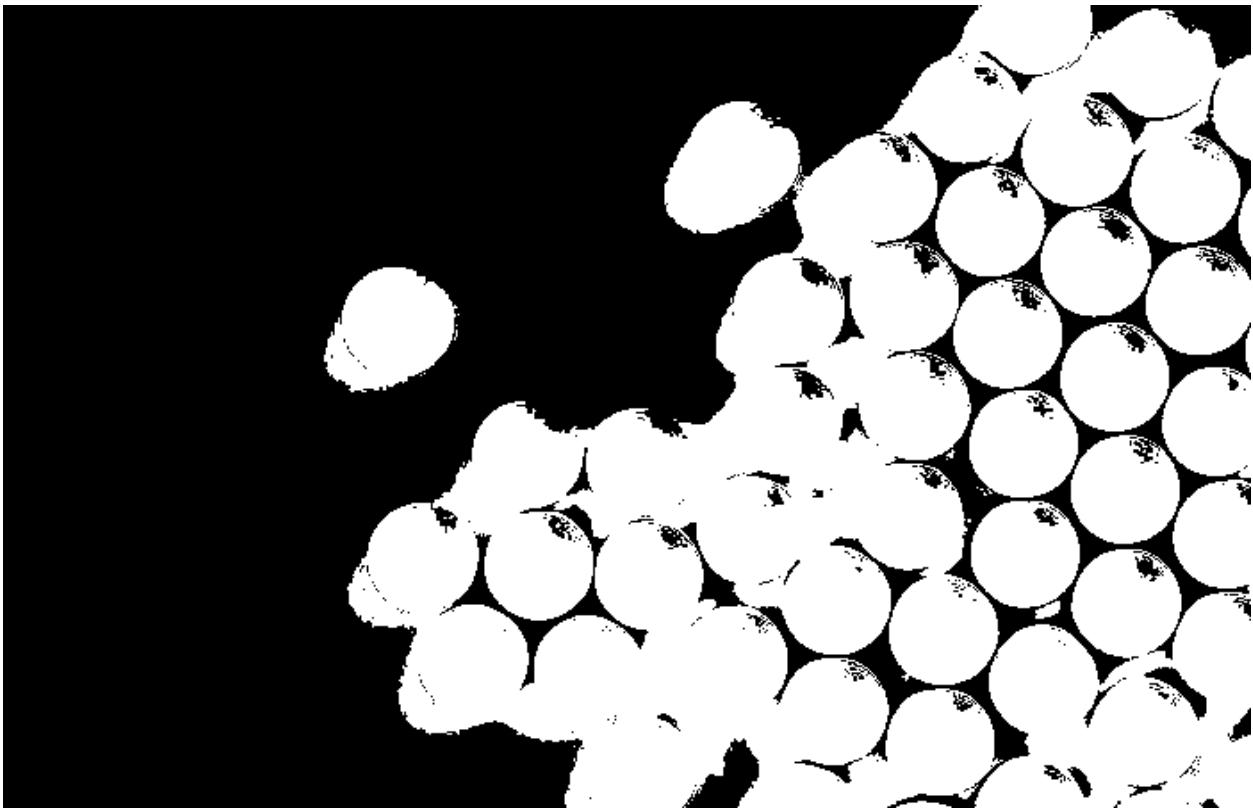
سیب ناقص در تصویر رنگی (p5g-rgb.png)



h قسمت

کد این قسمت در p5h.m قرار دارد، در ابتدا تصویر را می‌خوانم و آن را از Gray scale به RGB تبدیل می‌کنم
در ادامه سطح روشنایی‌های بین ۱۲۰ تا ۲۳۳ را نگه می‌دارم و باقی را برابر با صفر قرار می‌دهم، اینگونه تا جای
ممکن قرص‌ها را نگه می‌دارم و پس زمینه را حذف می‌کنم، در زیر خروجی این قدم را مشاهده می‌کنید

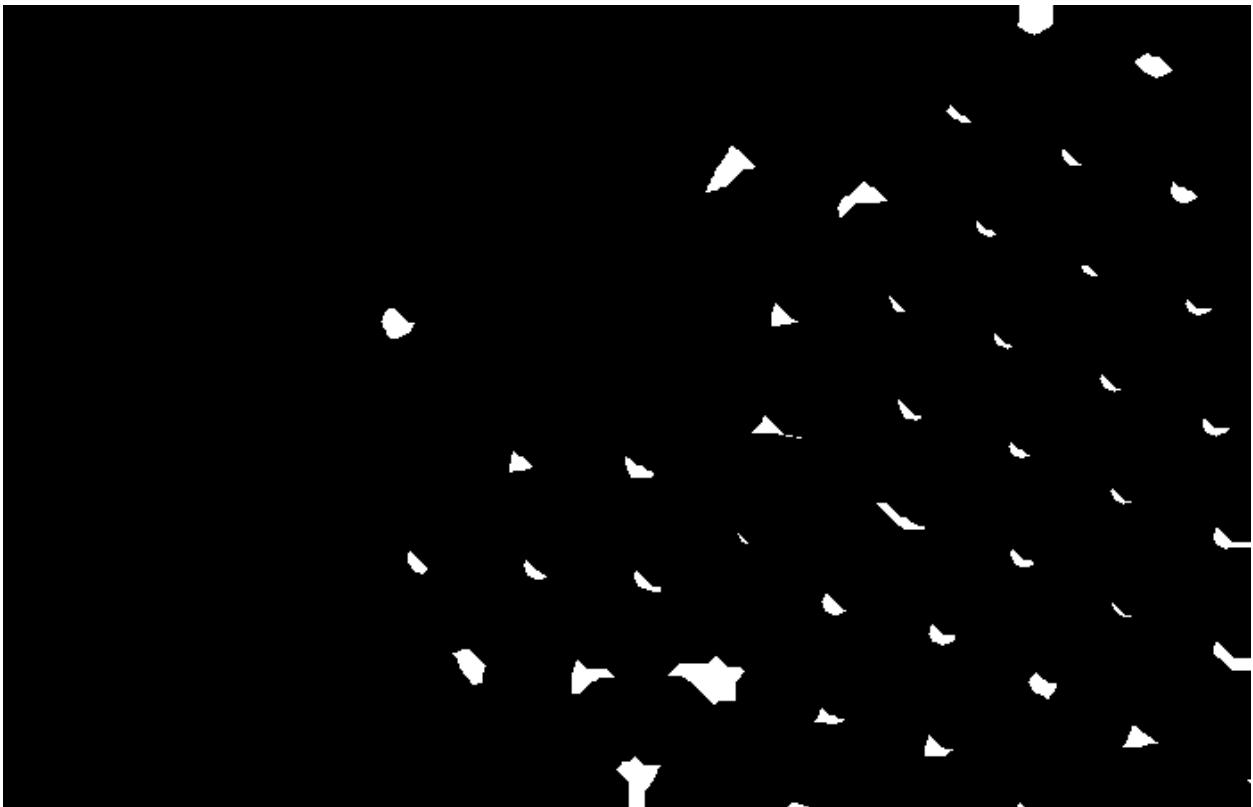
:(p5h-0.png)



برای جدا کردن قرص ها از هم، تصویر بالا را با ساختار زیر فرسایش می دهم:

```
se = strel('disk', 22);
```

که خروجی زیر حاصل می شود(p5h-1.png)

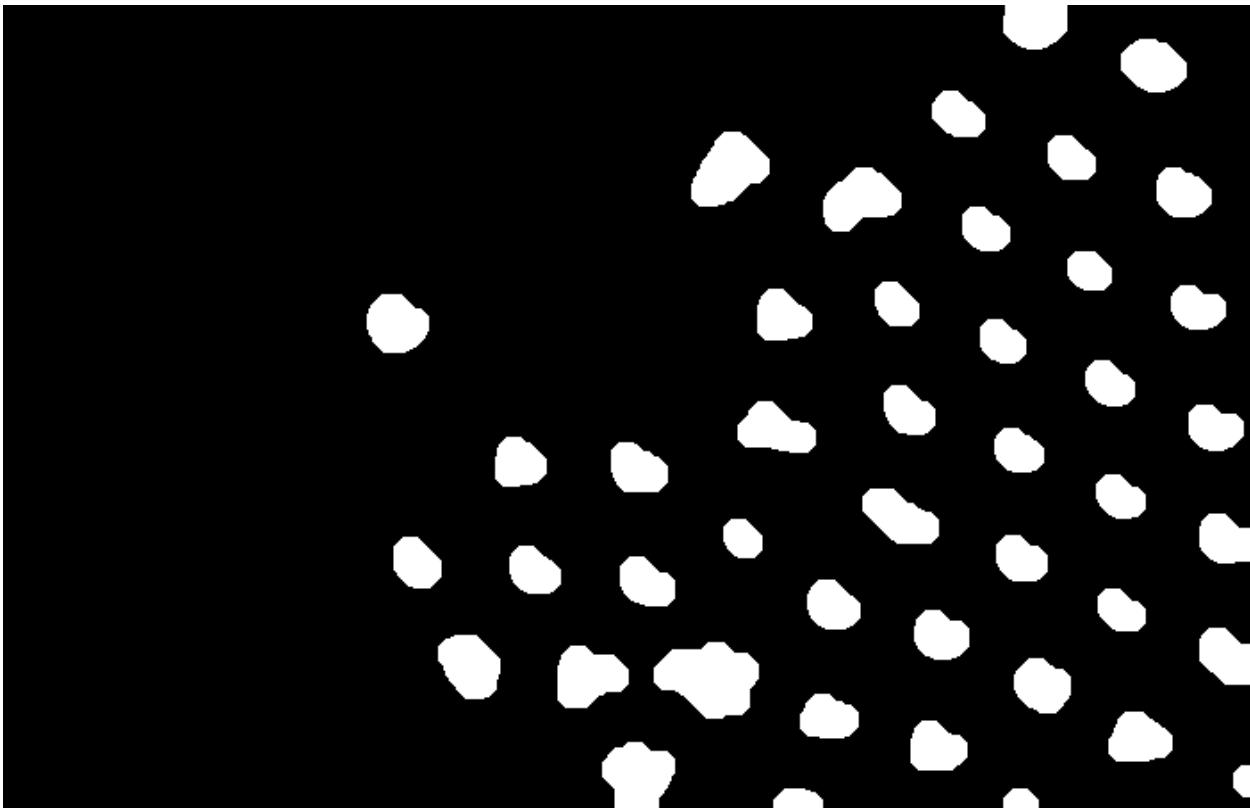


در بعضی ناحیه‌ها بین پیکسل‌های باقی مانده از بعضی قرص‌ها جدایی وجود دارد که آن را با dilation با ساختار

```
se = strel('disk', 9);
```

برطرف می‌کنم.

خروجی این قسمت را در زیر مشاهده می‌کنید(p5h-2.png)



اکنون قرص‌ها را به صورت ناحیه‌هایی سفید از هم داریم، برای شمارش آن‌ها از Connected Component به با رابطه‌ی زیر استفاده می‌کنم:

$$X_k = (X_{k-1} \oplus B) \cap A \quad k = 1, 2, 3, \dots$$

اینگونه با دیدن یک نقطه‌ی سفید، connected component مربوط به آن را پیدا می‌کنم و آن را از تصویر حذف می‌کنم و به سراغ پیکسل سفید بعدی می‌روم و این کار را تکرار می‌کنم تا پیکسل سفیدی باقی نماند. در زیر خروجی این قسمت که تعداد قرص‌ها را نمایش می‌دهد را نشان داده‌ایم:

```
Command Window
>> p5h
Connected Component:
44
fx >>
```

تمرین ۶

کدهای این قسمت در پوشه‌ی p6 قرار دارد.

قسمت ۲

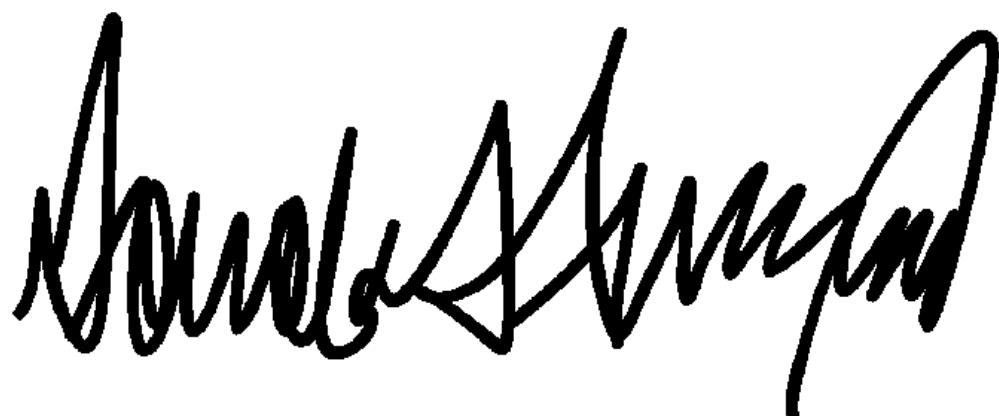
کد این قسمت در p6a.m قرار دارد. در کد ابتدا تصویر را می‌خوانیم، سپس تصویر را باینری می‌کنیم و از آنجایی که امضاها سیاه هستند، تصویر را نگاتیو می‌کنیم. سپس اسکلت امضاها را استخراج می‌کنیم، در زیر خروجی این کار را مشاهده می‌کنید:

ورودی:

Rouhani input signature

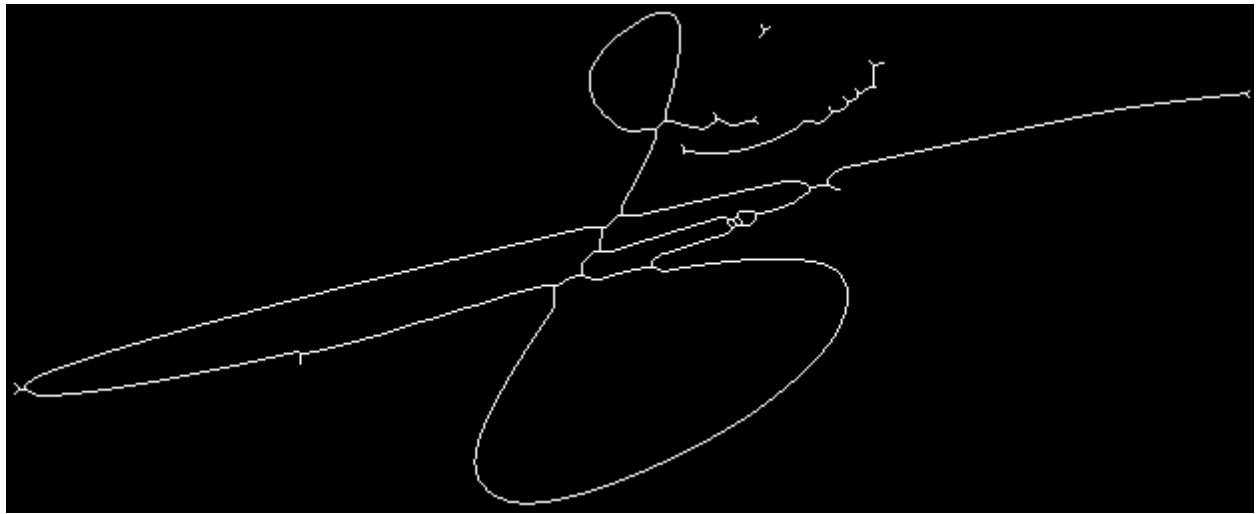


Trump input signature

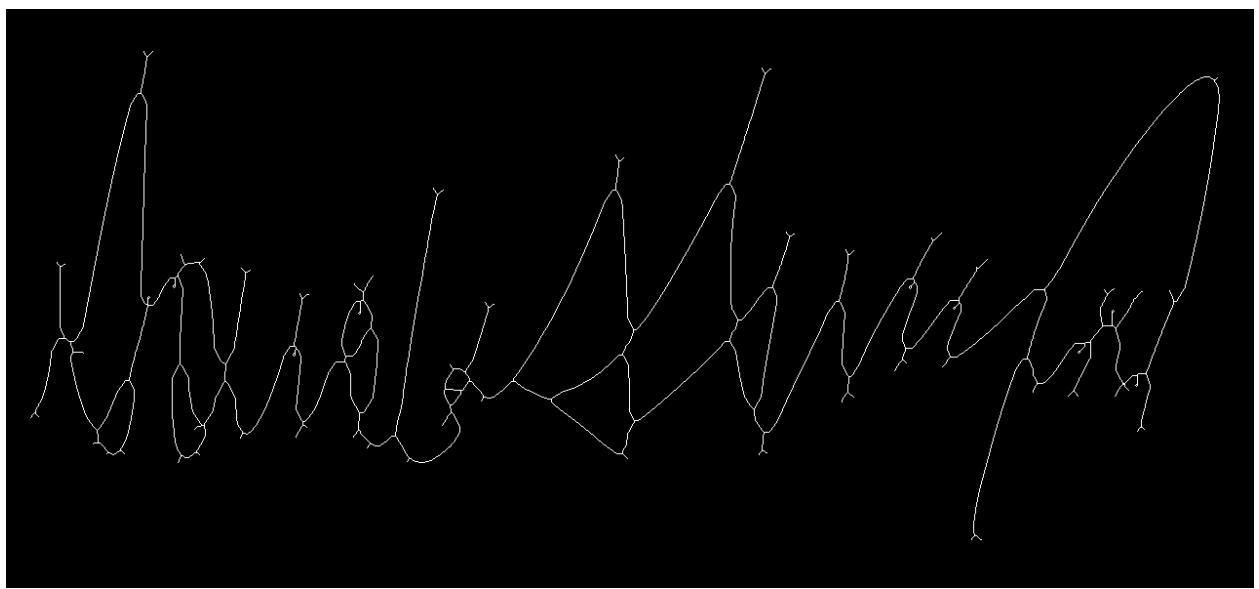


خروجی:

p6a-rouhani_signature.png



p6a-trump_signature.png

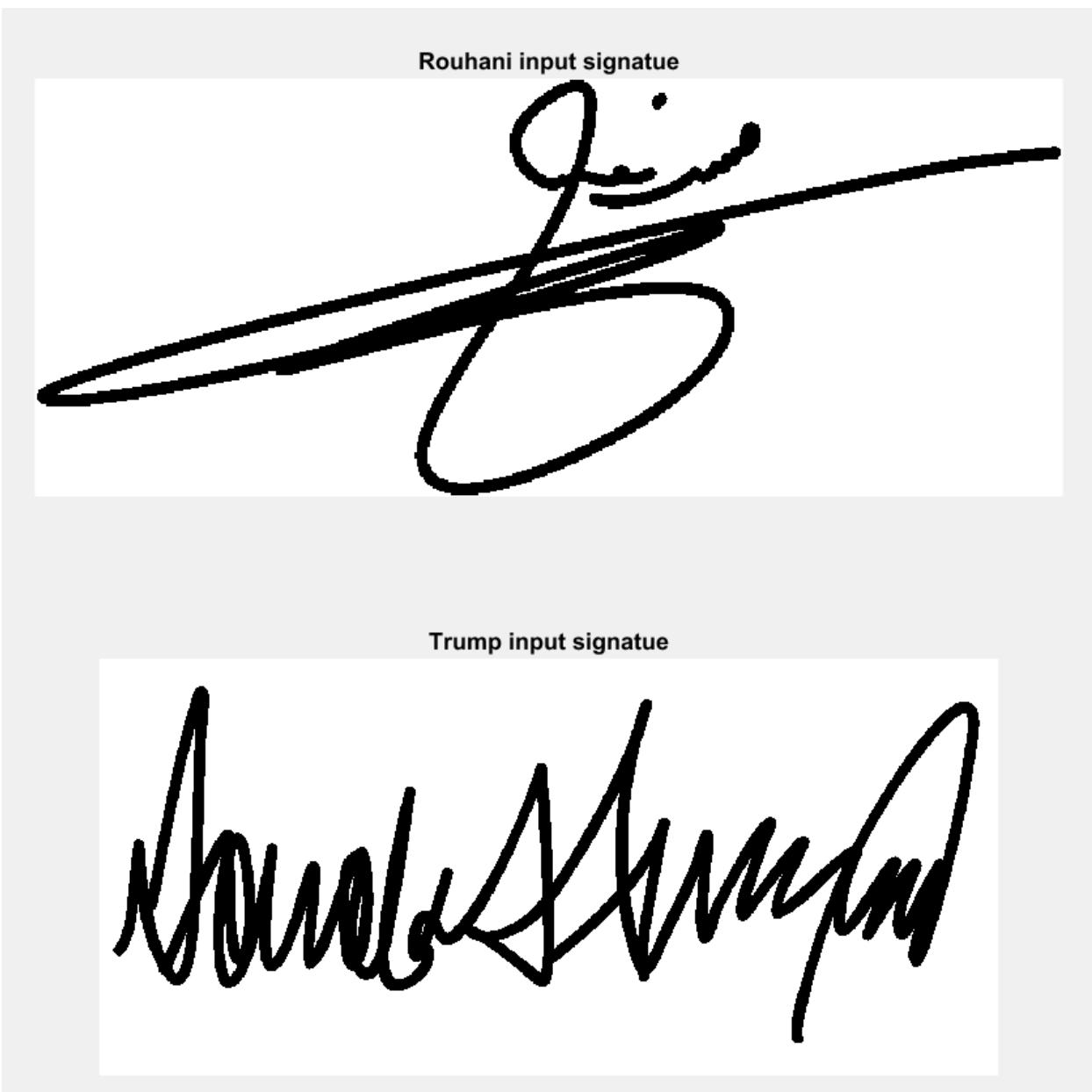


این ویژگی در تمام امضاهای یک فرد به شکل‌هایی با تغییرات اندک وجود دارد، بعضی از نیم خطها در امضاهای مختلف یک فرد می‌تواند کشیده‌تر باشد، شبیشان متفاوت باشد و ... ولی ساختار کلی اسکلت در تمام آن‌ها به صورت کلی یکسان است. در نتیجه می‌توان از میزان شباهت دو اسکلت برای تشخیص امضای یک فرد استفاده کرد.

قسمت b

کد این قسمت در p6b.m قرار دارد. در کد ابتدا تصویر را می‌خوانیم، سپس تصویر را باینری می‌کنیم و از آنجایی که امضاها سیاه هستند، تصویر را نگاتیو می‌کنیم. سپس اسکلت امضاها را استخراج می‌کنیم، در ادامه الگوریتم End Point را بر روی اسکلت‌ها اعمال می‌کنیم، در زیر خروجی این کار را مشاهده می‌کنید:

وروودی:

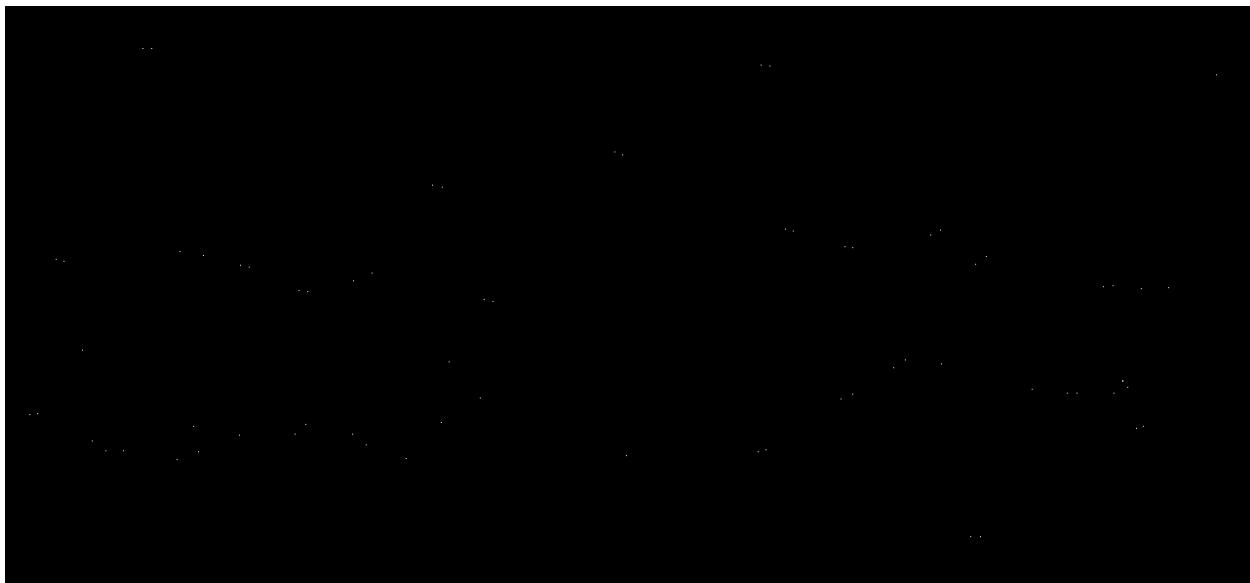


خروجی:

p6b-rouhani_signature.png



p6b-trump_signature.png



در هر دو تصویر بالا نقاط سفید زیادی دیده می‌شود که با زوم کردن بهتر مشخص خواهند بود، در امضاهای مختلف یک فرد تعداد این نقاط بسیار نزدیک به هم است و موقعیت مکانی این نقاط نسبت به هم بسیار شبیه خواهد بود به همین دلیل این ویژگی بسیار Stable است. در امضاهای یک فرد، شکل کلی end point ها نسبت به هم و تعدادشان بسیار شبیه است به همین دلیل به عنوان یک ویژگی می‌توان از آن‌ها استفاده کرد.

قسمت C

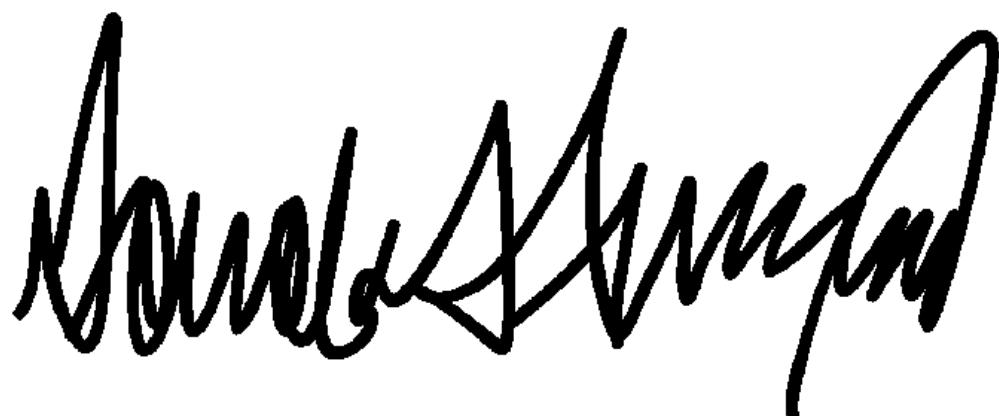
کد این قسمت در p6c.m قرار دارد. در کد ابتدا تصویر را می‌خوانیم، سپس تصویر را باینری می‌کنیم و از آنجایی که امضاها سیاه هستند، تصویر را نگاتیو می‌کنیم. سپس حالت Thin شده‌ی امضاها را استخراج می‌کنیم، در ادامه الگوریتم Branch Point ها را بر روی امضاهای نازک شده اعمال می‌کنیم، در زیر خروجی این کار را مشاهده می‌کنید:

ورودی:

Rouhani input signature



Trump input signature

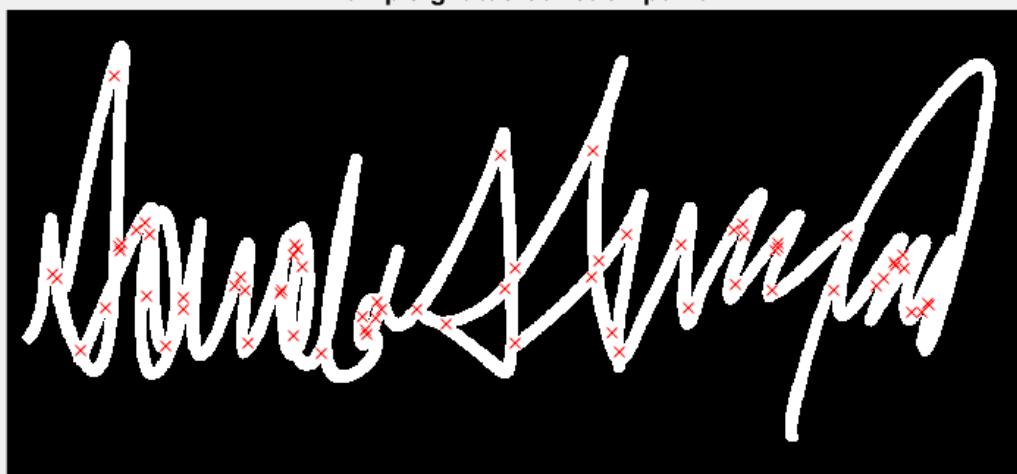


خروجی:

Rouhani signature-Junction point



Trump signature-Junction point



شکل کلی امضاهای یک فرد یکسان است و معمولاً کاراکترها و خطها در نقاط خاصی باهم تلاقی پیدا می‌کنند به همین دلیل تعداد این نقاط و موقعیت مکانی آن‌ها در امضاهای مختلف نسبت به هم بسیار شبیه است. در نتیجه این ویژگی از stability بالایی برخوردار خواهد بود.

تمرین ۷

کدهای این قسمت در پوشه‌ی p7 قرار دارد.

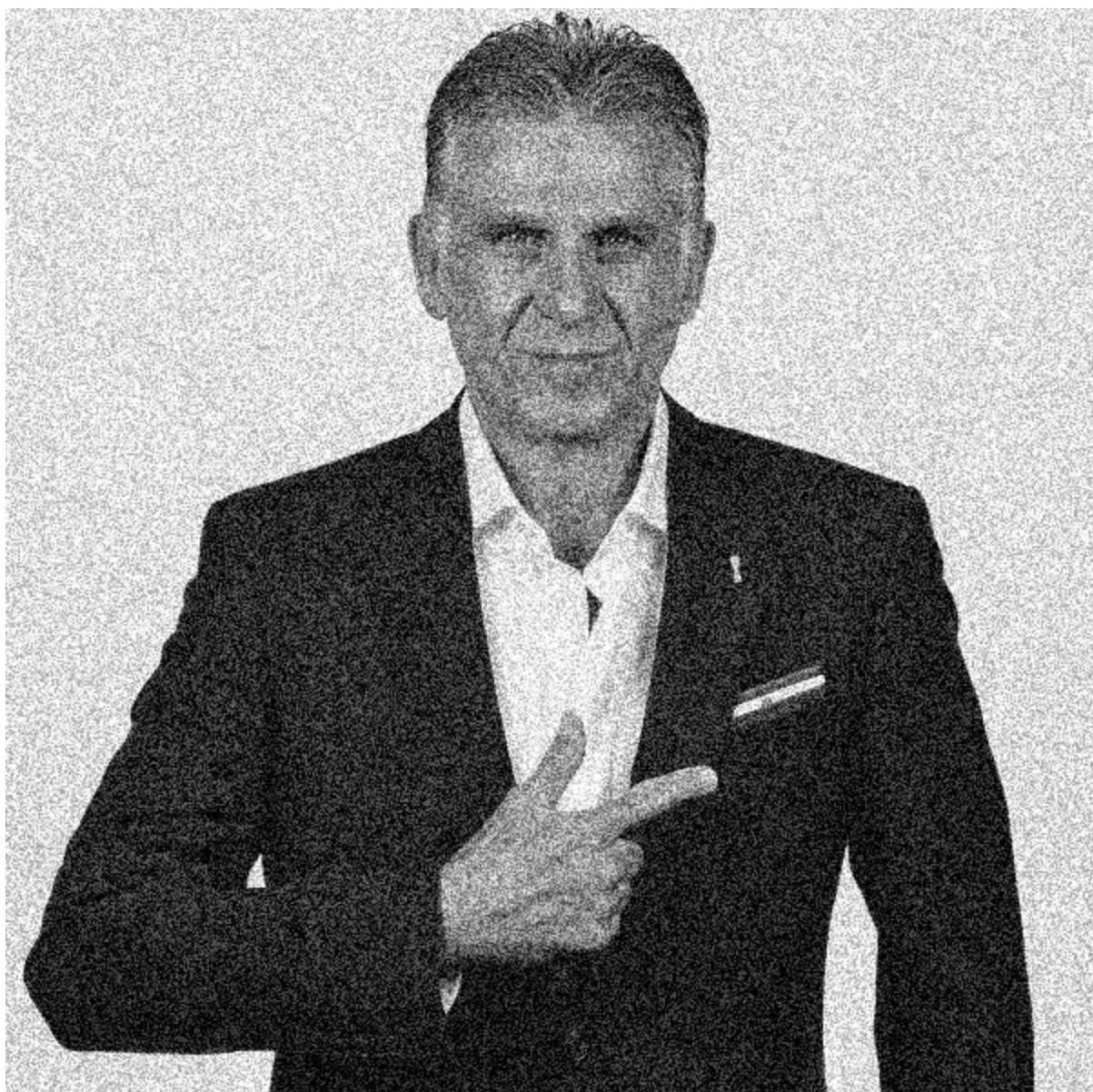
قسمت a

کد این قسمت در p7a.m قرار دارد. ابتدا Erosion و Dilation تصویر کیروش را با ساختار یک مربع 4×4 حساب می‌کنیم که تمام درایه‌های آن یک است. سپس از هر دو میانگین می‌گیریم:

$$[dyt_B f](x) = \left[\frac{1}{2}(\delta_B + \varepsilon_B) \right] f(x)$$

خروجی کد بالا را در ادامه مشاهده می‌کنید:

تصویر ورودی:



: تصویر خروجی (p7a.png)



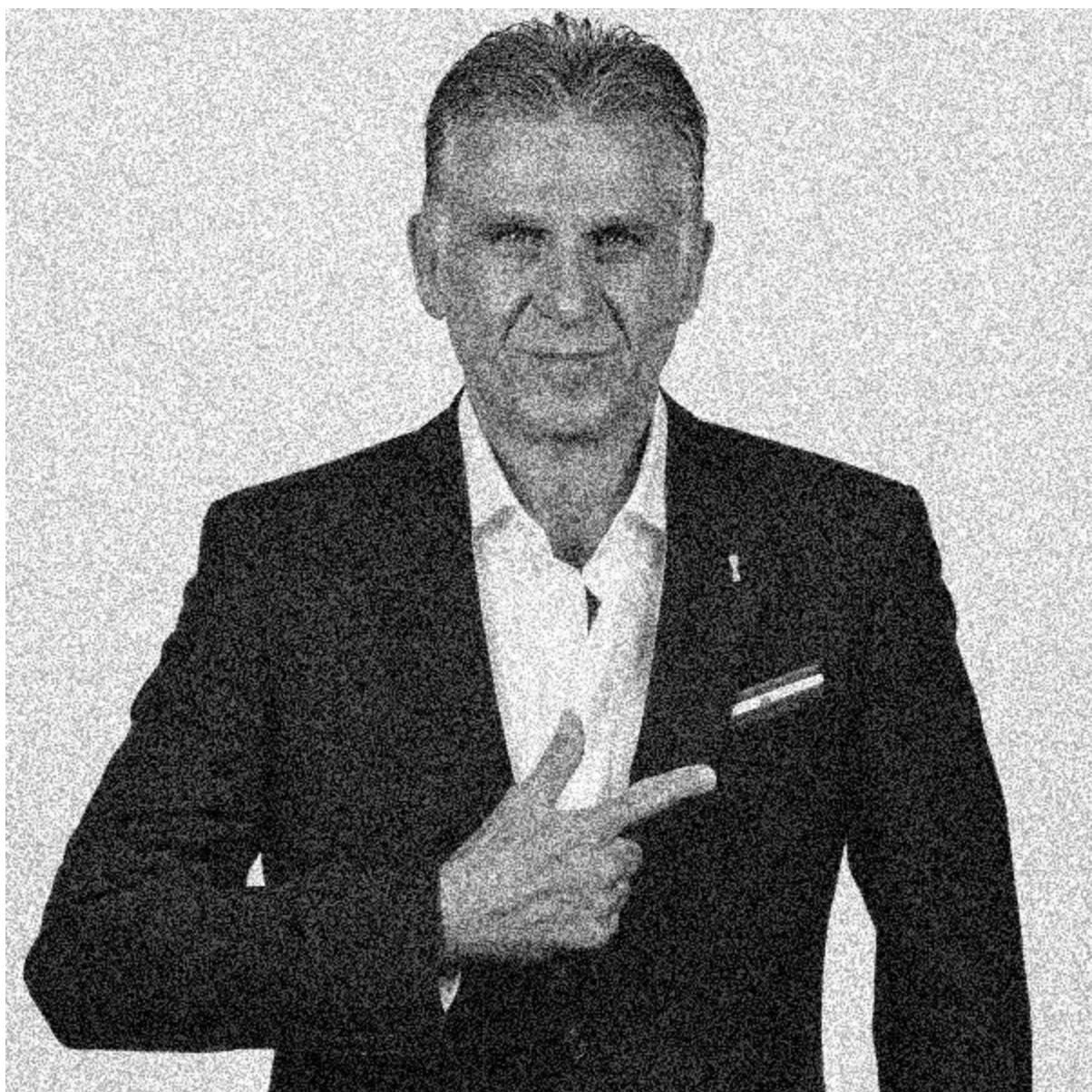
قسمت b

کد این قسمت در p7b.m قرار دارد. ابتدا Closing و Opening تصویر کیروش را با ساختار یک مریع 4×4 حساب می‌کنیم که تمام درایه‌های آن یک است. سپس از هر دو میانگین می‌گیریم:

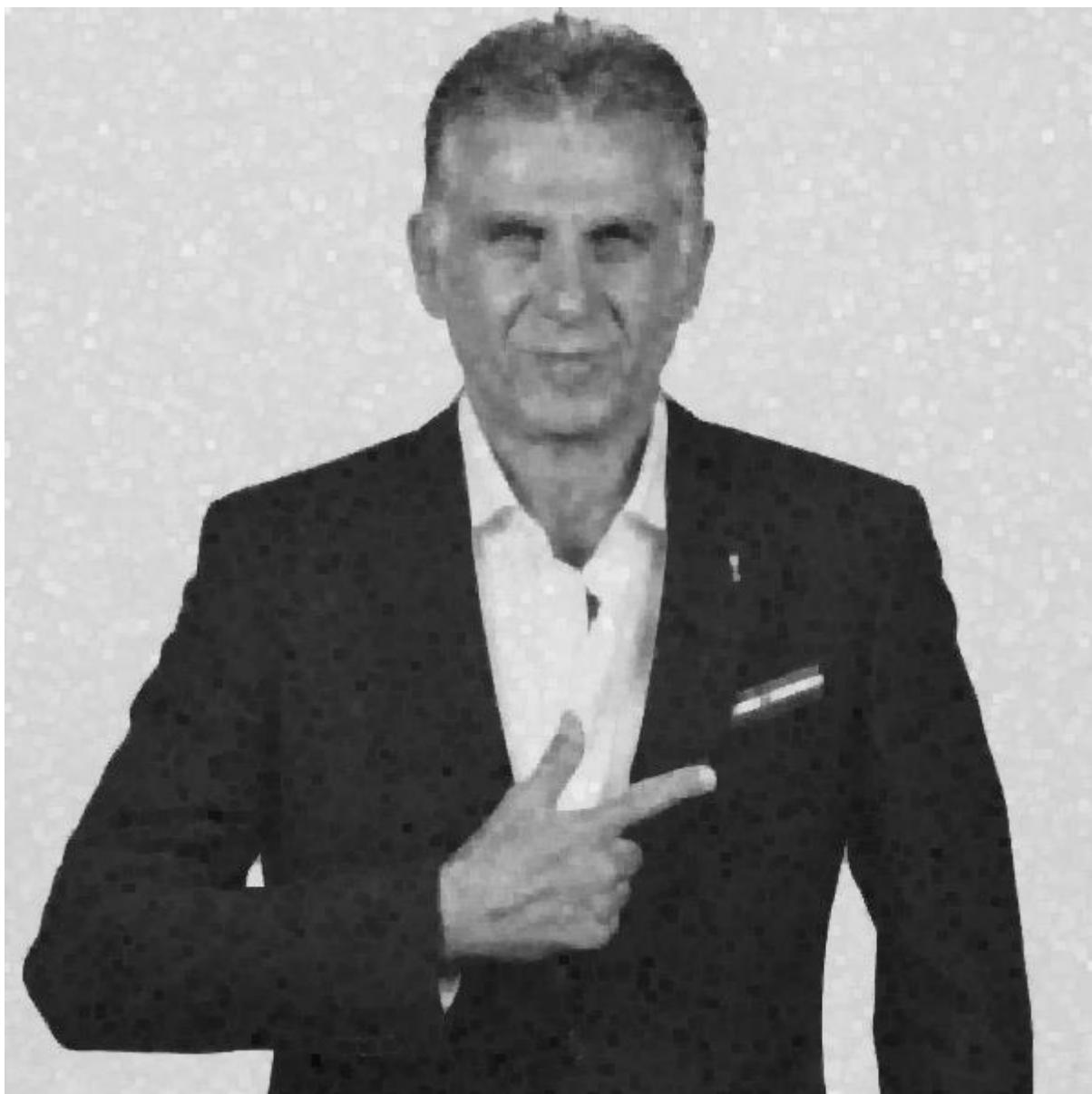
$$[\text{tet}_B f](x) = [\frac{1}{2}(\phi_B + \gamma_B)]f(x)$$

خروجی کد بالا را در ادامه مشاهده می‌کنید:

وروودی:



خروجی (p7b.png)

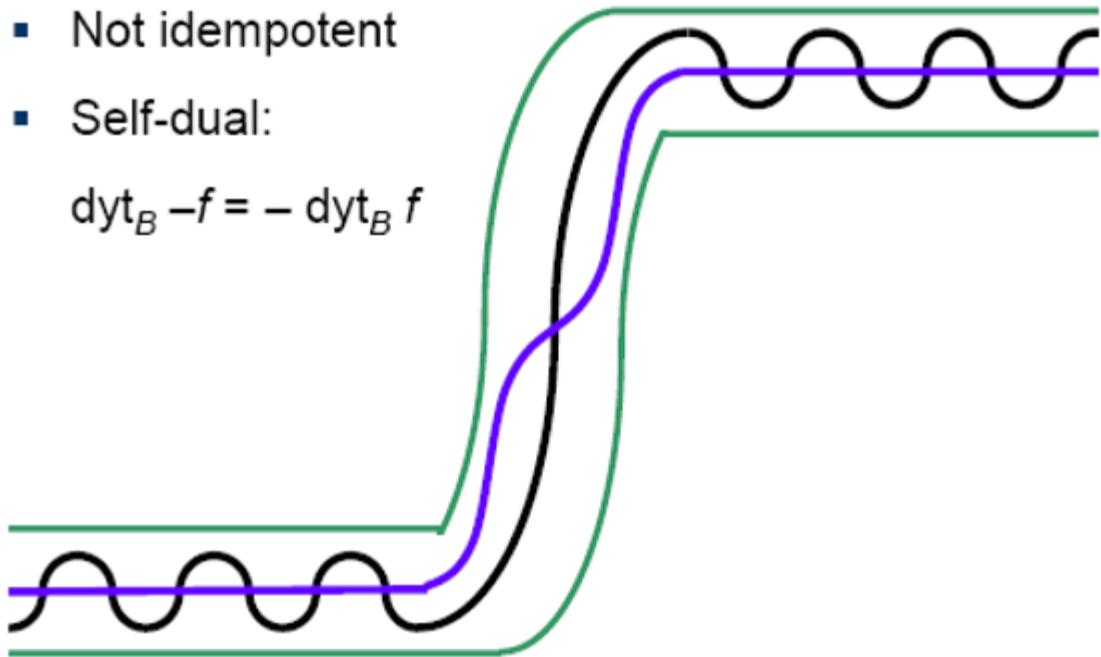


در قسمت‌های مختلف هر دو روش عملکرد یکسانی دارند **بجز** در لبه‌ها که متفاوت عمل می‌کنند. روش اول در ناحیه‌های نویزی و لبه‌ها به این صورت عمل می‌کند:

- Not idempotent

- Self-dual:

$$dyt_B -f = - dyt_B f$$

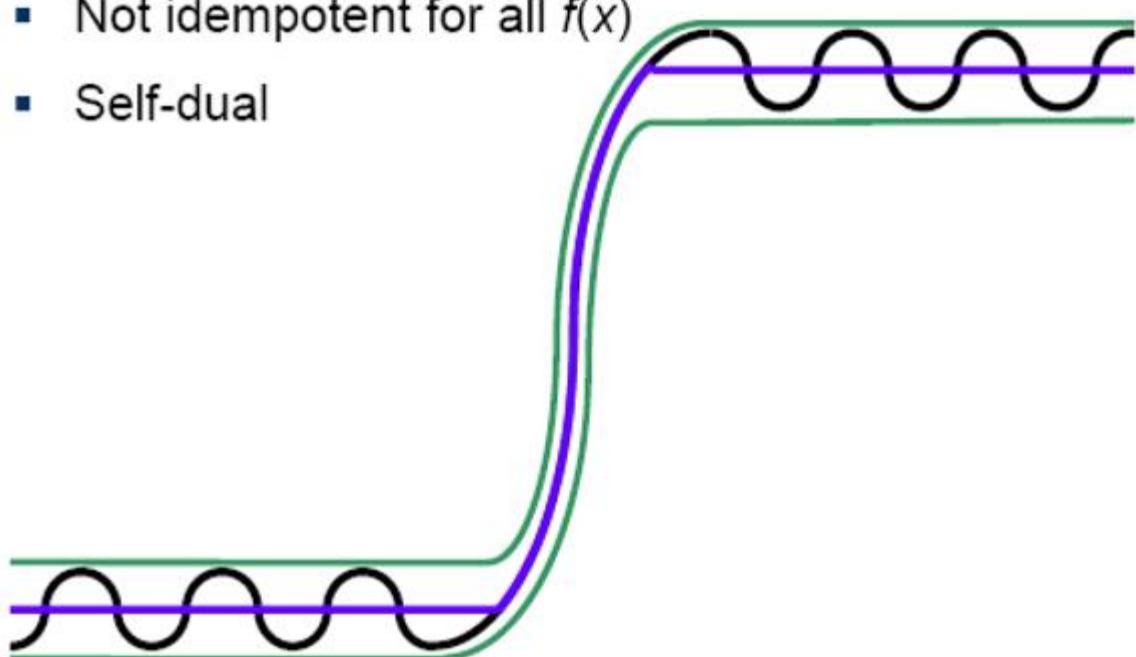


همین طور که دیده می‌شود لبه‌ها حفظ نمی‌شوند و دچار تغییر می‌شوند.

روش دوم به این صورت عمل می‌کند:

- Not idempotent for all $f(x)$

- Self-dual



در روش دوم لبه‌ها حفظ می‌شوند.

هر دو روش تصویر را **smooth** می‌کنند، نویزی هم که در تصویر وجود دارد یک نویز گوسی است که یکی از راه‌های حذف آن بلور کردن تصویر است.

قسمت C

کد این قسمت در p7c.m قرار دارد. ابتدا **dilation** و **erosion** تصویر را حساب می‌کنیم. سپس میانگین هر دو را از تصویر اصلی کم می‌کنیم. این گونه **2th Derivation** تصویر را محاسبه می‌کنیم. در زیر خروجی را مشاهده می‌کنید:

$$\left[I - \frac{1}{2}(\delta_B + \varepsilon_B) \right] f(x)$$

وروودی:



: (p7c.png) خروجی



شدت روشنایی در تصویر برای نمایش بهتر تنظیم شده است.

d قسمت

کد این قسمت در p7d.m قرار دارد. ابتدا opening و closing تصویر را حساب می‌کنیم. سپس میانگین هر دو را از تصویر اصلی کم می‌کنیم. این گونه 2th Derivation تصویر را محاسبه می‌کنیم. در زیر خروجی را مشاهده می‌کنید:

$$: \left[\mathbf{I} - \frac{1}{2}(\phi_B + \gamma_B) \right] f(x)$$

وروڈی:



: (p7d.png خروجی



شدت روشنایی در تصویر برای نمایش بهتر تنظیم شده است.

قسمت e

کد این قسمت در p7e.m قرار دارد. از dilation ، erosion ، opening ، closing و 2th derivation محاسبه‌ی استفاده می‌کنیم. در زیر خروجی را مشاهده می‌کنید:

$$\text{img} - \left[\frac{1}{2}(\phi_B - \delta_B) + \frac{1}{2}(\gamma_B - \varepsilon_B) \right]$$

وروڈی:



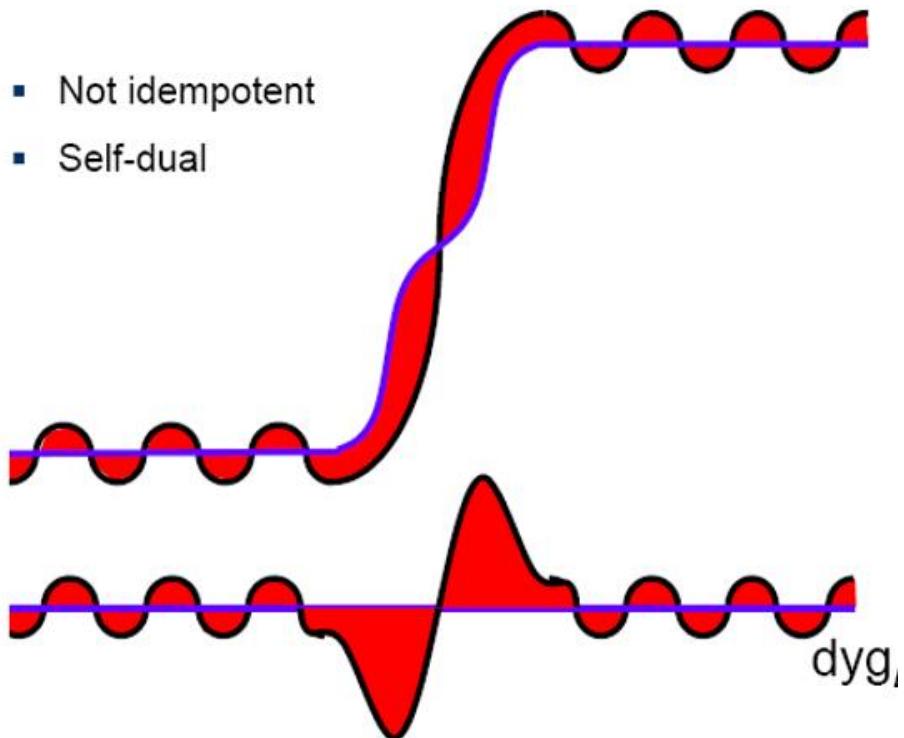
:p7e.png خروجی



شدت روشنایی در تصویر برای نمایش بهتر تنظیم شده است.

روش اول در لبه‌ها و ناحیه‌های نویزی بدین شکل عمل می‌کند:

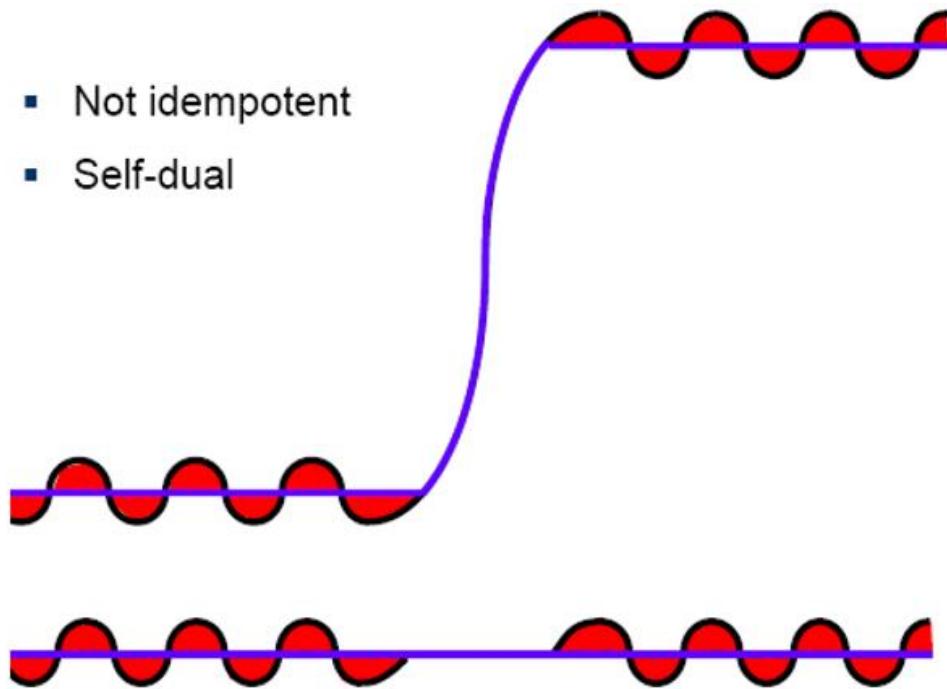
- Not idempotent
- Self-dual



در 2th derivation با این روش اثر نویز و لبه‌ها در خروجی مشخص است.

روش دوم در لبه‌ها و ناحیه‌های نویزی بدین شکل عمل می‌کند:

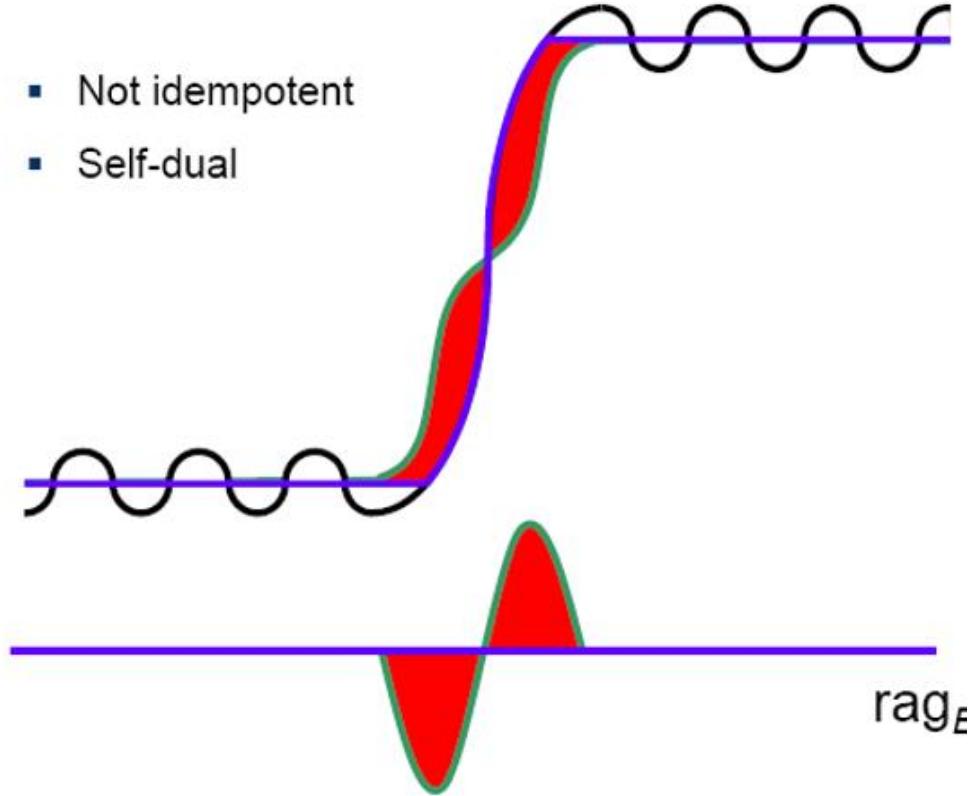
- Not idempotent
- Self-dual



در 2th derivation با این روش اثر نویز در خروجی مشخص است.

روش سوم در لبه‌ها و ناحیه‌های نویزی بدین شکل عمل می‌کند:

- Not idempotent
- Self-dual



در 2th derivation با این روش اثر لبه‌ها در خروجی مشخص است و نویزی بودن و یا نبودن تاثیری در خروجی ندارد.

قسمت f

کد این قسمت در p7f.m قرار دارد. ابتدا Erosion و Dilatation تصویر را حساب می‌کنم سپس طبق فرمول زیر عمل می‌کنم:

$$[\text{dyr}_B f](x) = [\delta_B - \varepsilon_B]f(x)$$

وروڈی:



:p7f.png خروجی



قسمت g

کد این قسمت در p7g.m قرار دارد. ابتدا Closing و Opening تصویر را حساب می‌کنم سپس طبق فرمول زیر عمل می‌کنم:

$$[\text{ter}_B f](x) = [\phi_B - \gamma_B]f(x)$$

وروڈی:



:p7g.png) خروجی



قسمت h

کد این قسمت در p7g.m قرار دارد. ابتدا تصویر را حساب می‌کنم سپس طبق فرمول زیر عمل می‌کنم:

$$[\text{rar}_B f](x) = [(\delta_B - \phi_B) + (\gamma_B - \varepsilon_B)]f(x)$$

وروڈی:



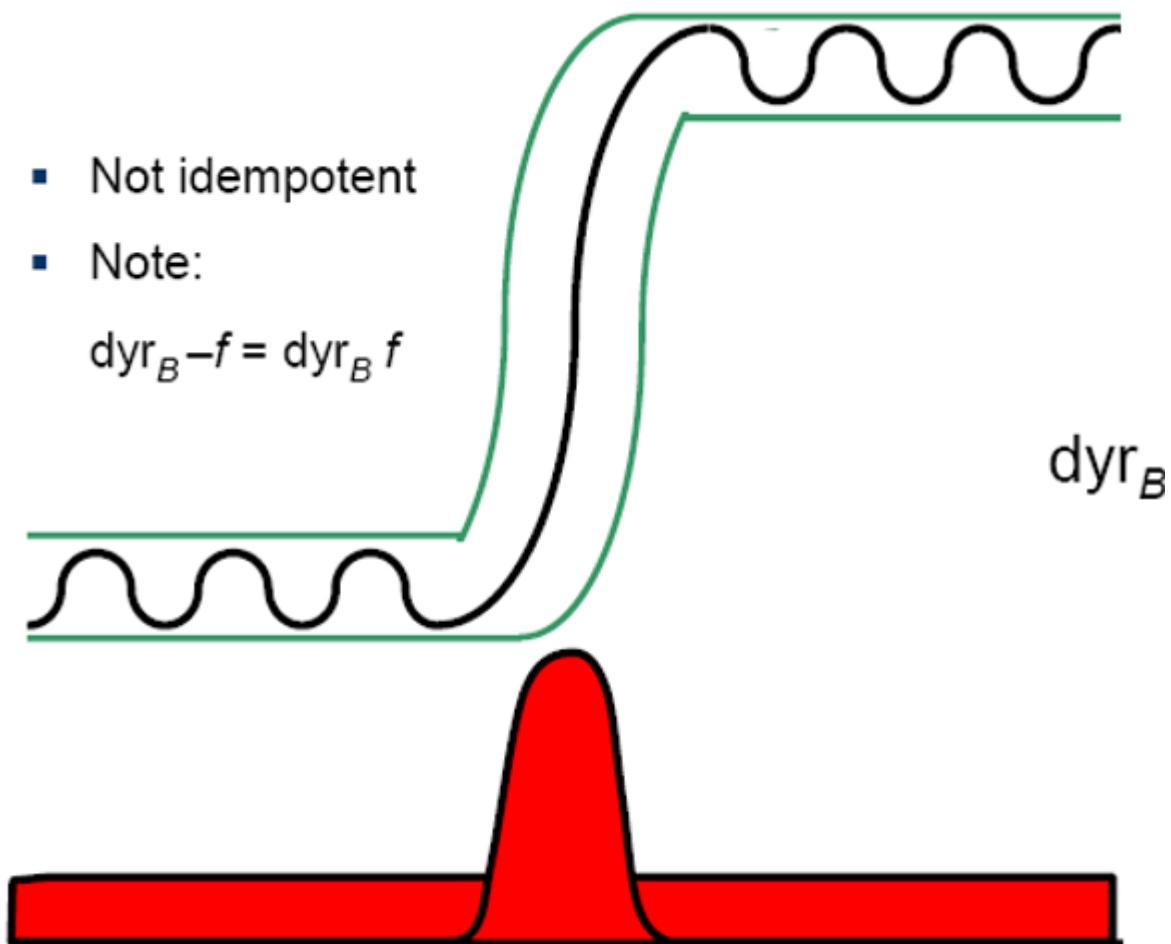
:p7h.png خروجی



روش اول در برخورد با لبه‌ها و ناحیه‌های نویزی و با تغییرات کم به شکل زیر عمل می‌کند:

- Not idempotent
- Note:

$$\text{dyr}_B - f = \text{dyr}_B f$$



همین طور که مشاهده می شود این روش نسبت به نویز و لبه حساس است.

روش دوم در برخورد با لبه ها و ناحیه های نویزی و با تغییرات کم به شکل زیر عمل می کند:

- Not idempotent
- Note:

$$\text{ter}_B - f = \text{ter}_B f$$



همین طور مه مشاهده می‌شود این روش نسبت به نویز حساس است ولی نسبت به لبه حساس نیست.

روش سه در برخورد با لبه‌ها و ناحیه‌های نویزی و با تغییرات کم به شکل زیر عمل می‌کند:

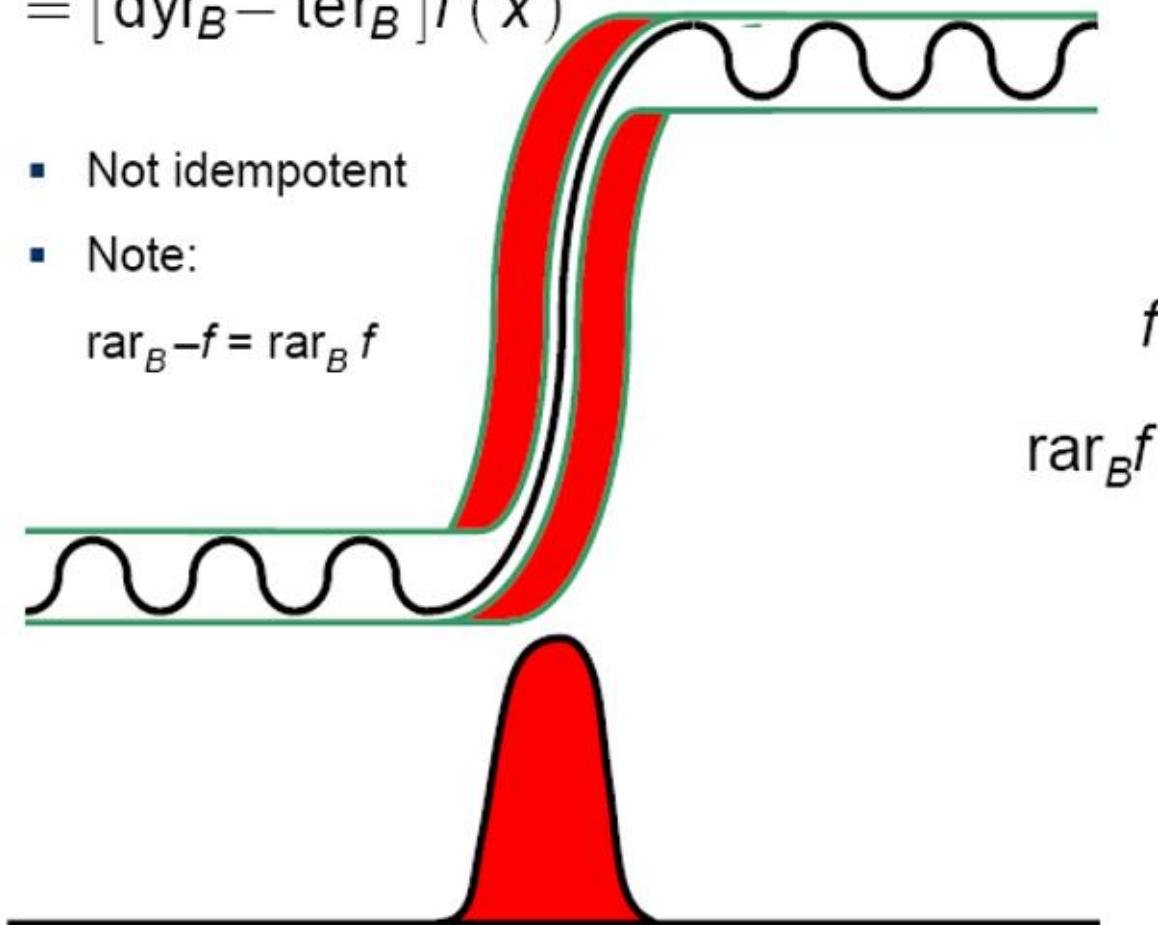
$$[\text{rar}_B f](x) = [(\delta_B - \phi_B) + (\gamma_B - \varepsilon_B)]f(x)$$

$$= [\text{dyr}_B - \text{ter}_B]f(x)$$

- Not idempotent

- Note:

$$\text{rar}_B - f = \text{rar}_B f$$



همین طور مه مشاهده می شود این روش نسبت به لبه حساس است ولی نسبت به نویز حساس نیست.

در سه خروجی بالا هم، همین اثرها به ترتیب دیده می شود.

تمرین ۸

کدهای این قسمت در پوشه‌ی p8 قرار دارد.

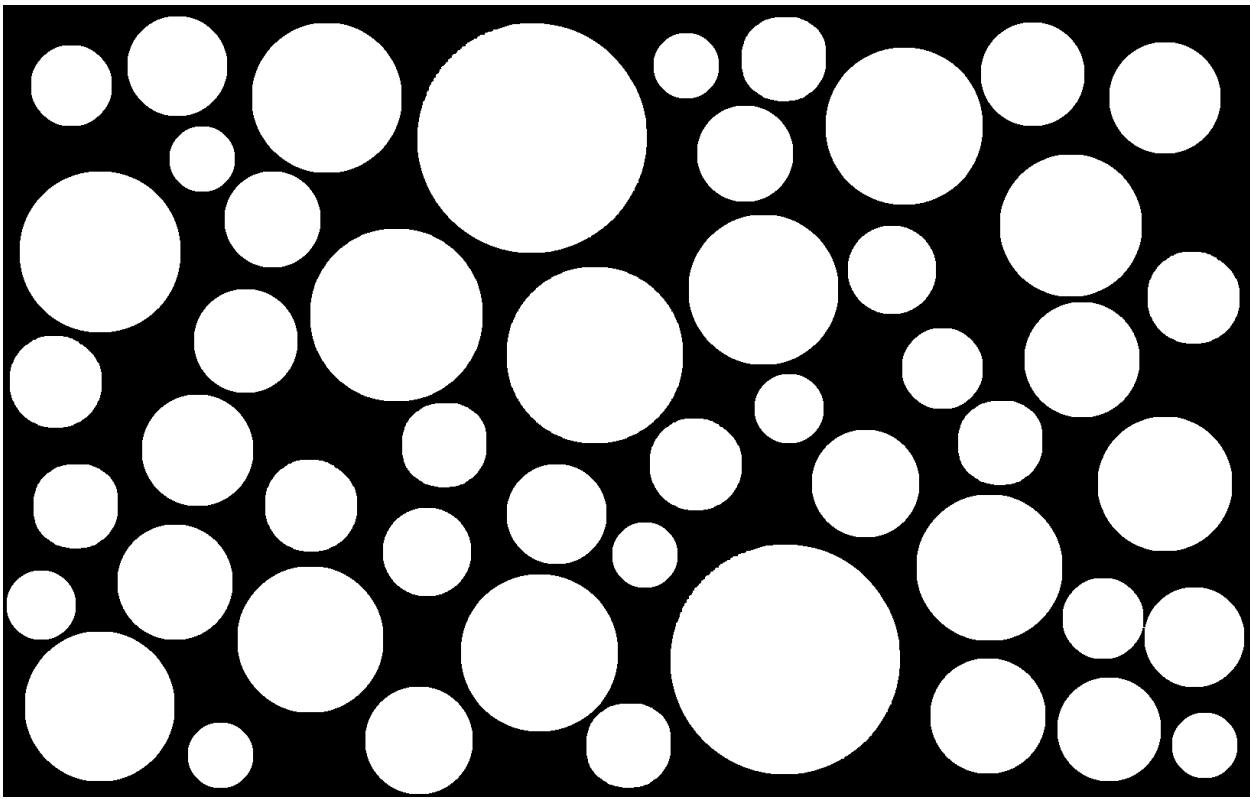
قسمت ۲

کد این قسمت در p8a.m قرار دارد. در ابتدا تصویر توپ‌ها را می‌خوانیم، سپس تصویر را به استفاده از Thresholding به یک تصویر باینری تبدیل می‌کنیم. برای اینکه توپ‌ها سفید شوند و پس زمینه سیاه شود، تصویر به دست آمده را Negative می‌کنیم. بعد از اینکار توپ‌ها سفید می‌شوند و پس زمینه سیاه، ولی مشکلی که به وجود می‌آید این است که در بعضی ناحیه‌ها درون توپ‌ها نقاطی سیاه رنگ دیده می‌شود که با استفاده از Closing آن‌ها را برطرف می‌کنیم تا به اینجا تصویر زیر به دست می‌آید:

تصویر ورودی:



تصویر ورودی باینری (p8a-1.png)



اکنون برای پیدا کردن بزرگ‌ترین و کوچک‌ترین توب به این شکل عمل می‌کنیم:
- هر توب یک **Connected Component** است.

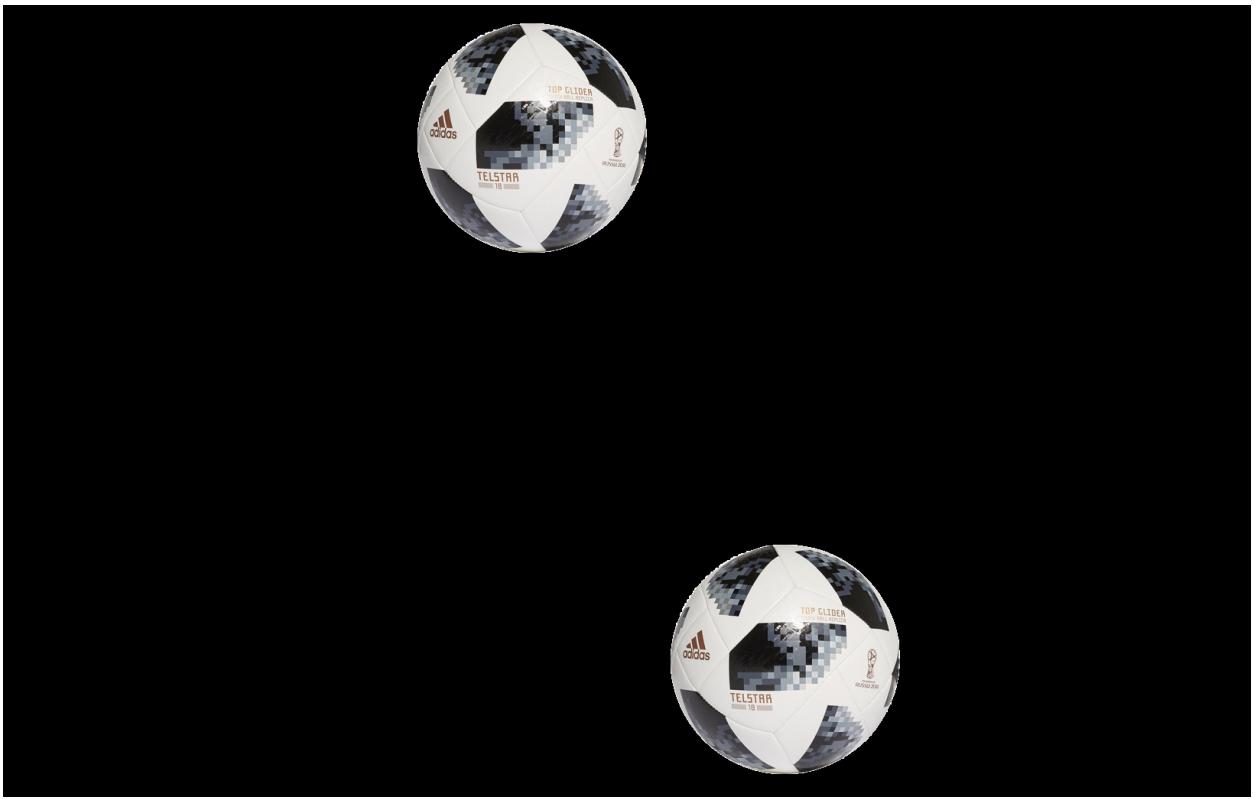
- با استفاده از الگوریتم **Connected Component**‌ها که بر اساس رابطه‌ی زیر کار می‌کنند:

$$X_k = (X_{k-1} \oplus B) \cap A \quad k = 1, 2, 3, \dots$$

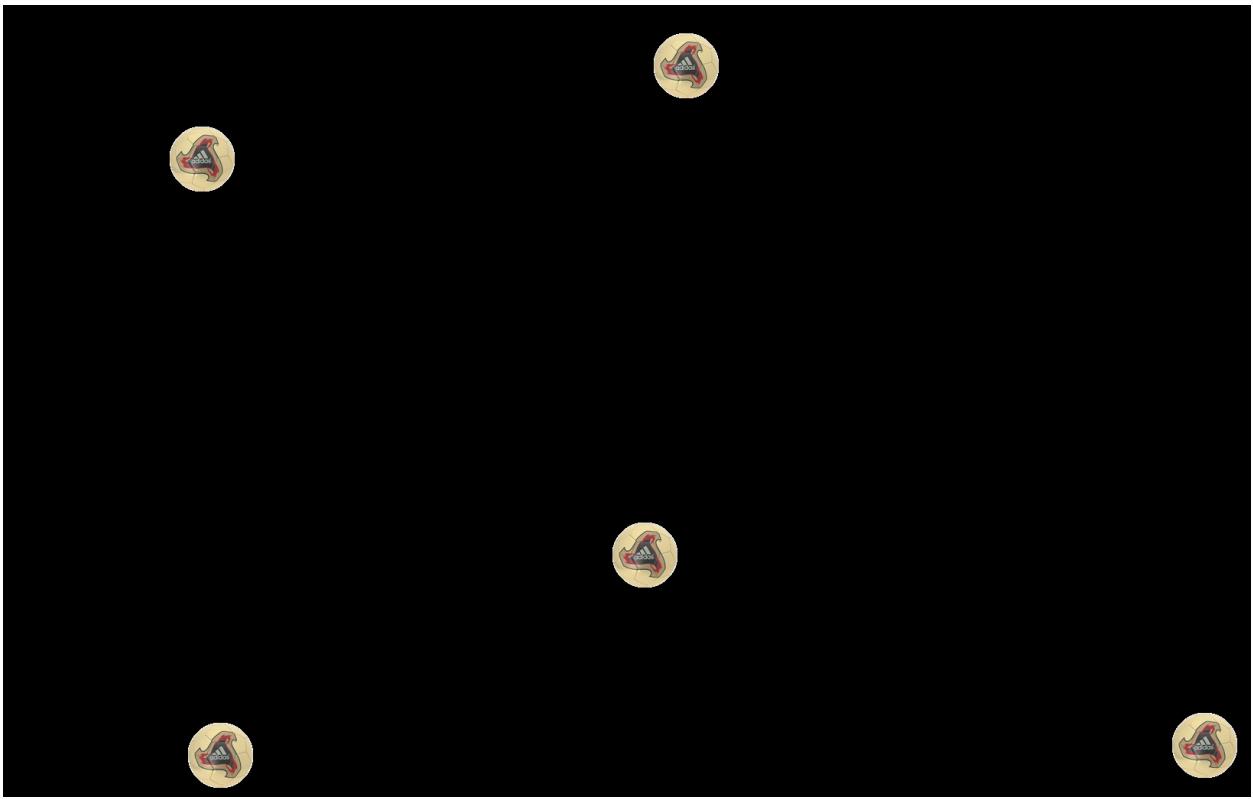
در هر مرحله یک توب را پیدا می‌کنیم و تعداد پیکسل‌های آن را می‌شماریم و توب را از تصویر حذف می‌کنیم و به سراغ توب بعدی می‌رویم.

- اکنون تعداد پیکسل‌های هر توب در تصویر را داریم، در نتیجه **connected component**‌های توب‌ها با بزرگ‌ترین و کوچک‌ترین مساحت (تعداد پیکسل) را جدا می‌کنیم و آن‌ها را نمایش می‌دهیم:

در زیر بزرگ‌ترین توب‌ها را مشاهده می‌کنید (p8a-2.png):



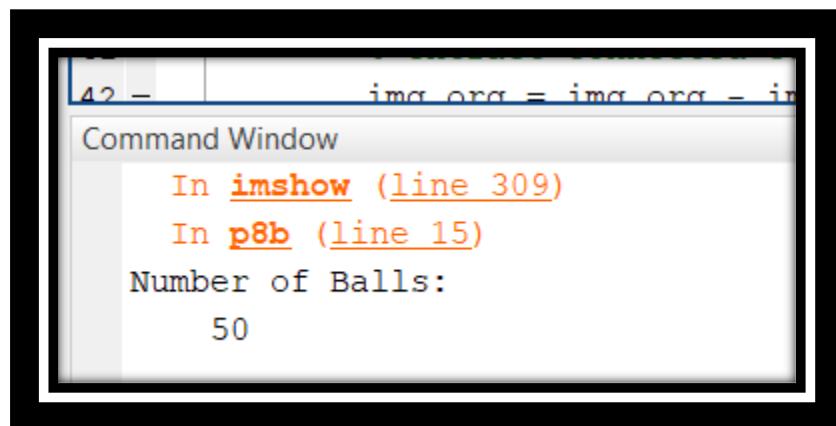
در زیر کوچک ترین توپ‌ها را مشاهده می‌کنید (p8a-3.png)



قسمت b

کد این قسمت در p8b.m قرار دارد، در این قسمت دقیقاً مانند قسمت قبل عمل می‌کنیم، تصویر را سیاه و سفید می‌کنیم، به طوری که توپ‌ها سفید باشند و پس زمینه سیاه باشد، در ادامه از روش Connected Component که در قسمت a توضیح دادیم، می‌کنیم و تعداد توپ‌ها را می‌شماریم

در زیر خروجی کد را مشاهده می‌کنید:

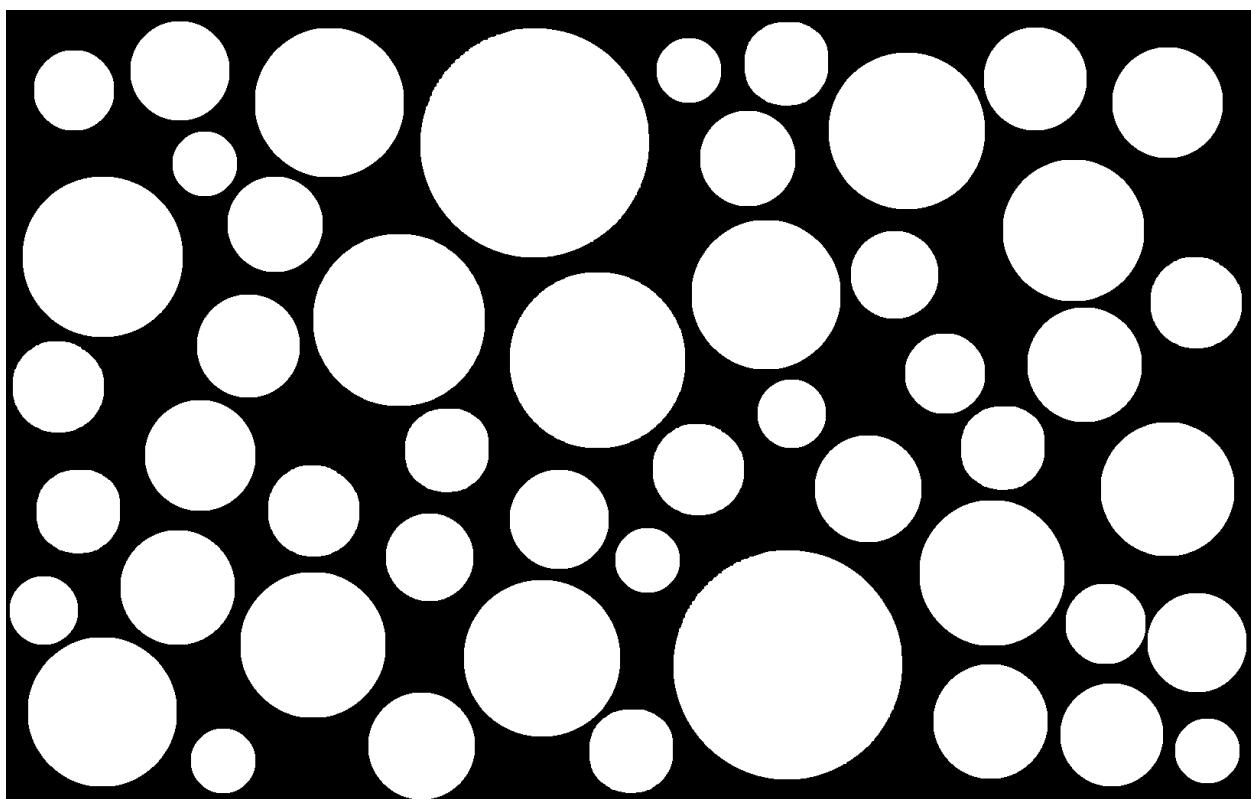


در تصویر 50 توپ وجود دارد.

کد این قسمت در p8c.m قرار دارد، در این قسمت در ابتدا مانند قسمت a عمل می‌کنیم، ابتدا تصویر را می‌خوانیم، سپس با استفاده از Closing و Thresholding تصویر را باینری می‌کنیم، در ادامه با استفاده از connected component زیر، تک تک توپ‌ها را از هم جدا می‌کنیم.

$$X_k = (X_{k-1} \oplus B) \cap A \quad k = 1, 2, 3, \dots$$

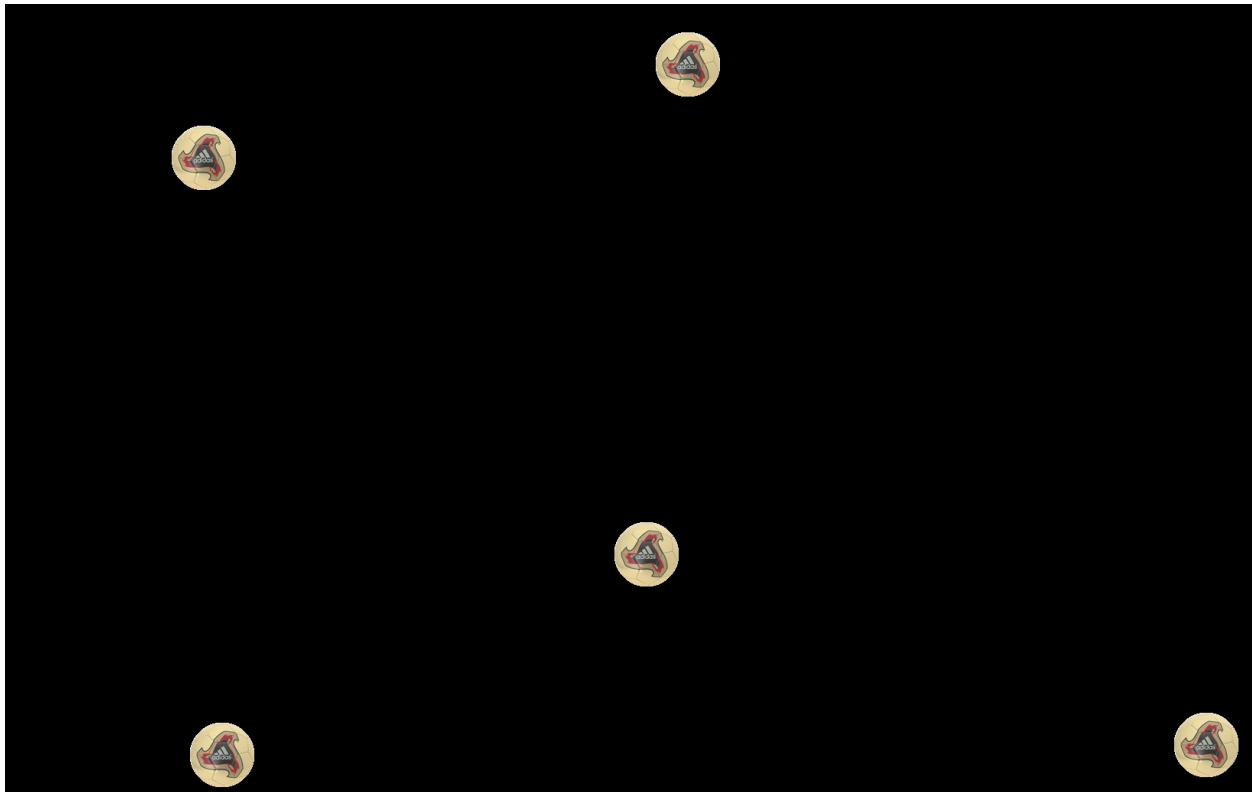
توپ‌ها و پس زمینه در تصویر باینری:



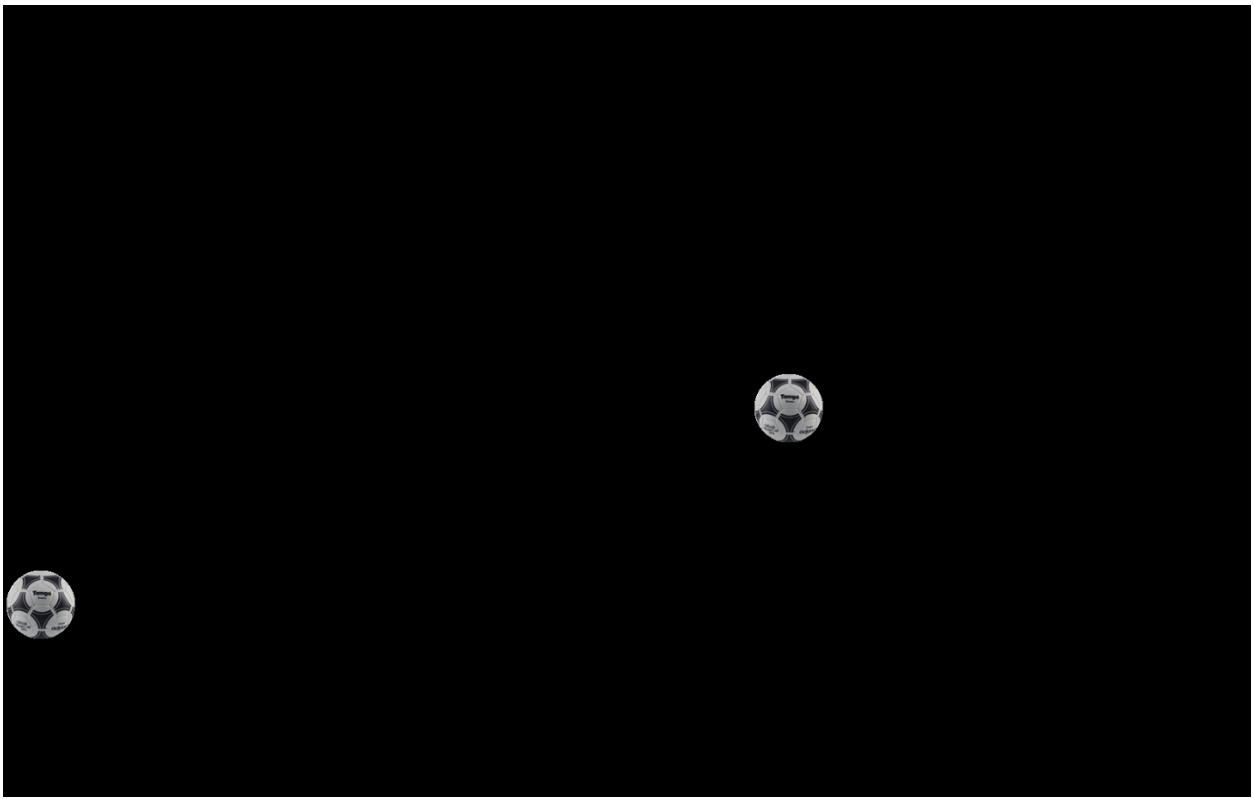
در ادامه، توپ‌ها با مساحت (تعداد پیکسل‌های یکسان) را در یک دسته قرار می‌دهیم. بدین ترتیب هر نوع توپ در یک دسته قرار می‌گیرد.

خروجی این قسمت، برای نوع‌های مختلف توپ را مشاهده می‌کنید:

نوع اول توب:



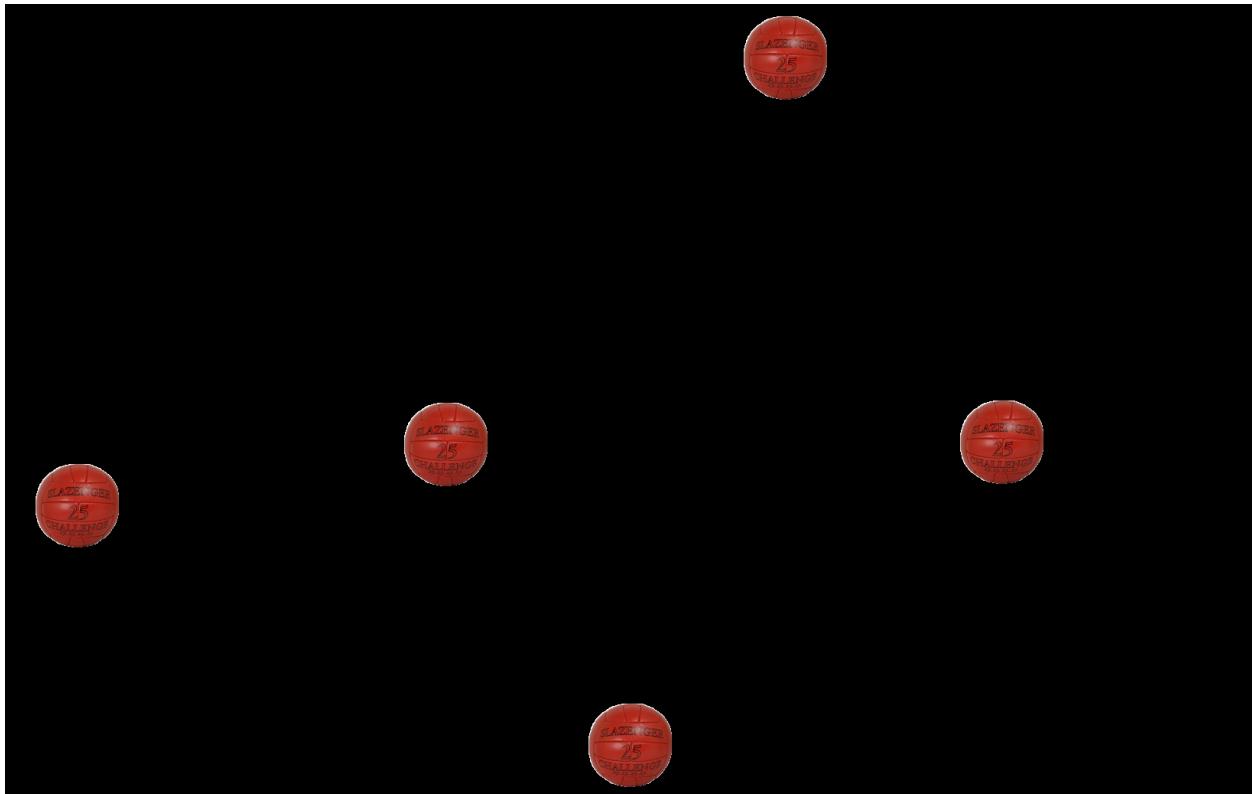
نوع دوم توب:



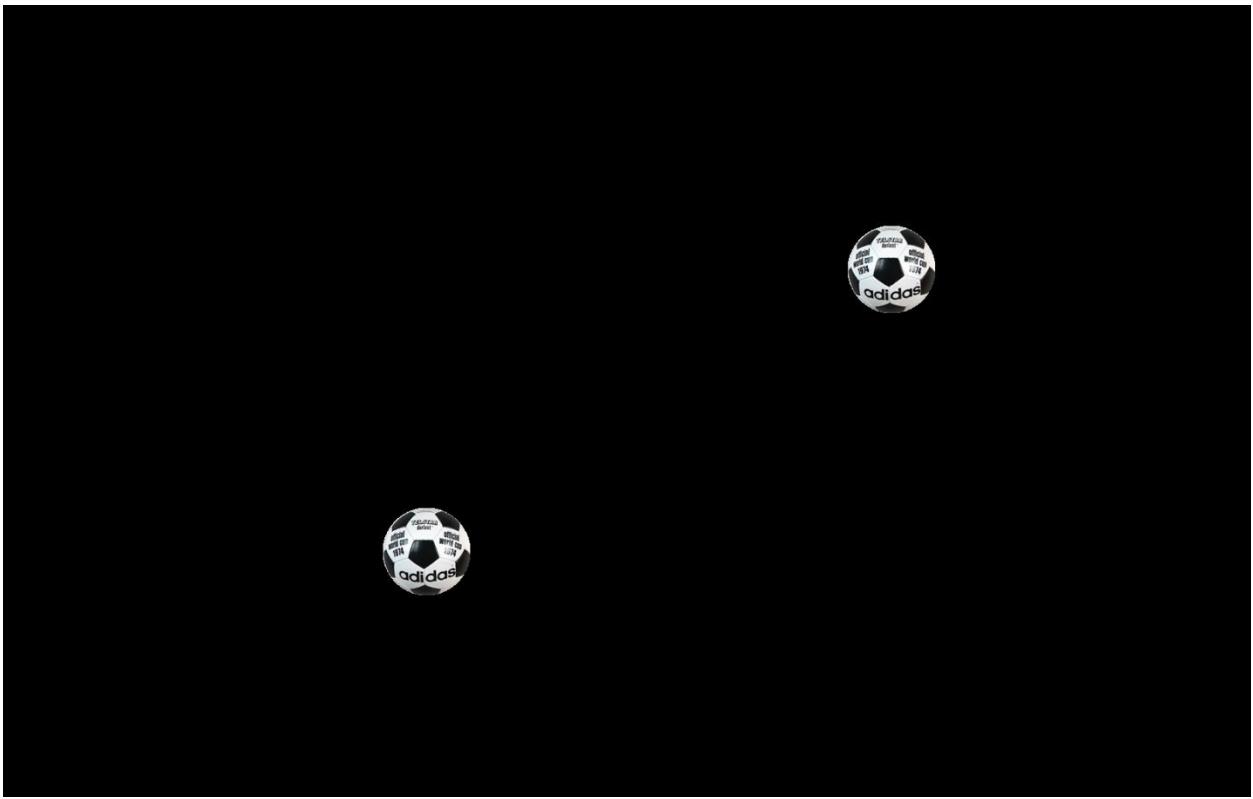
نوع سوم توپ:



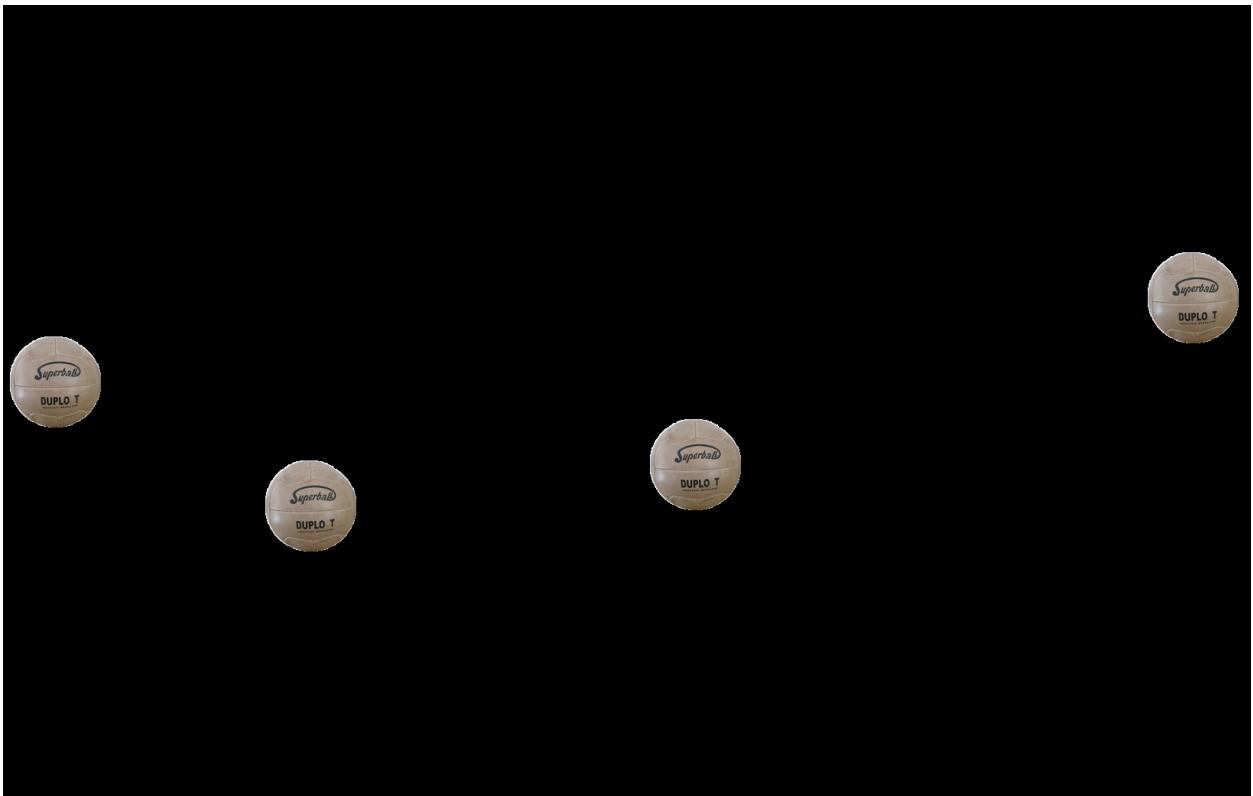
نوع چهارم توپ:



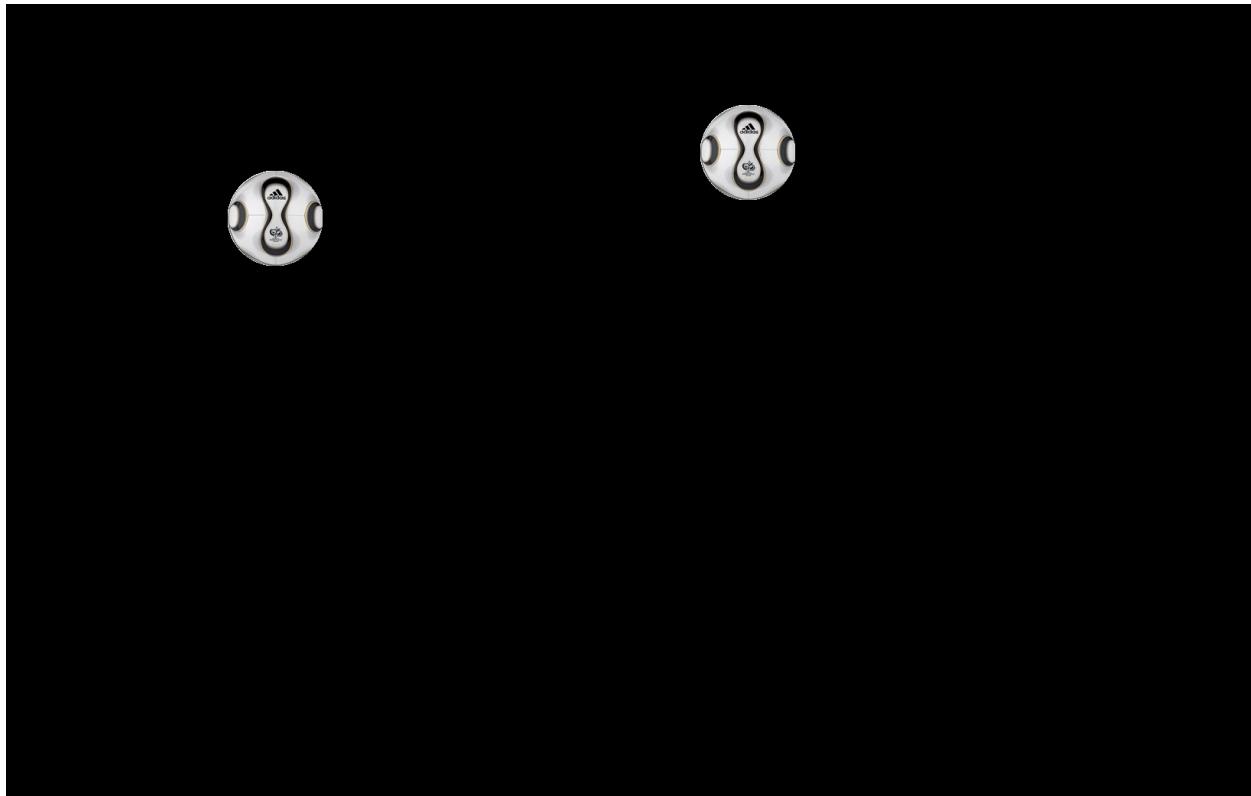
نوع پنجم توپ:



نوع ششم توب:



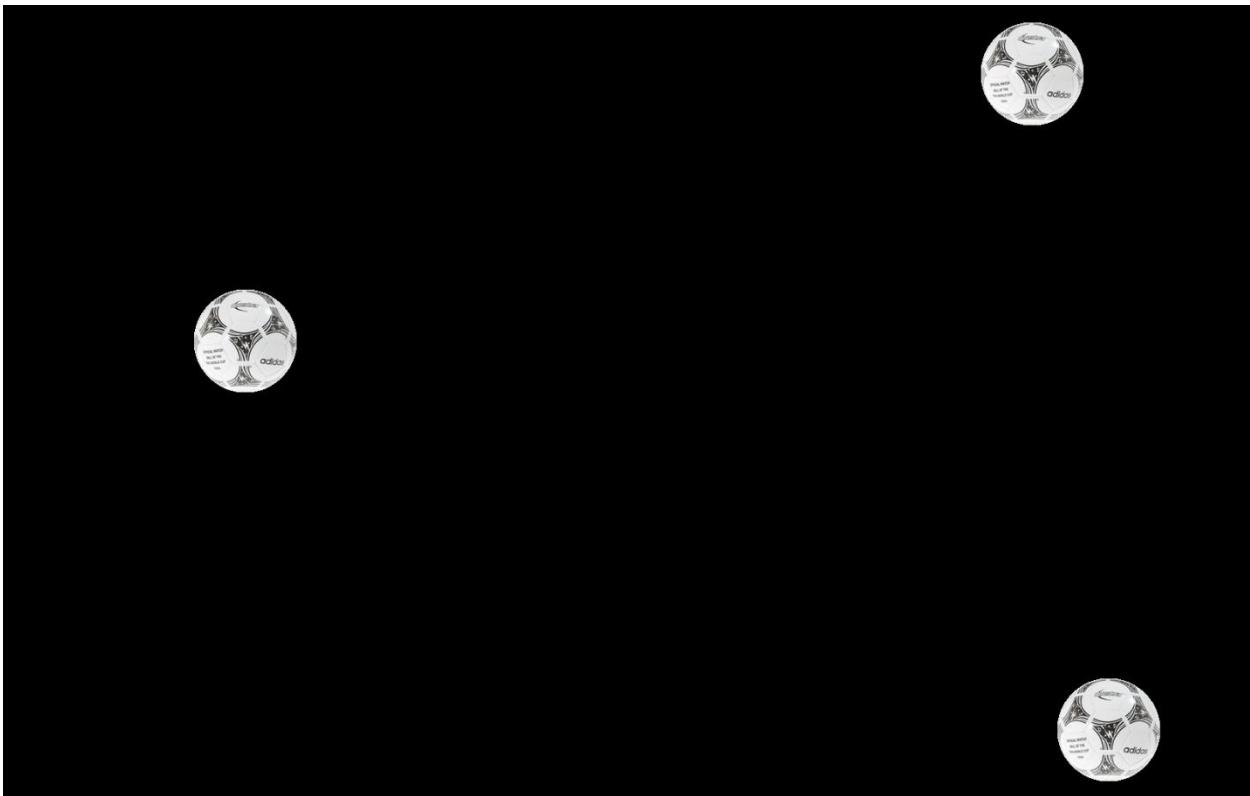
نوع هفتم توپ:



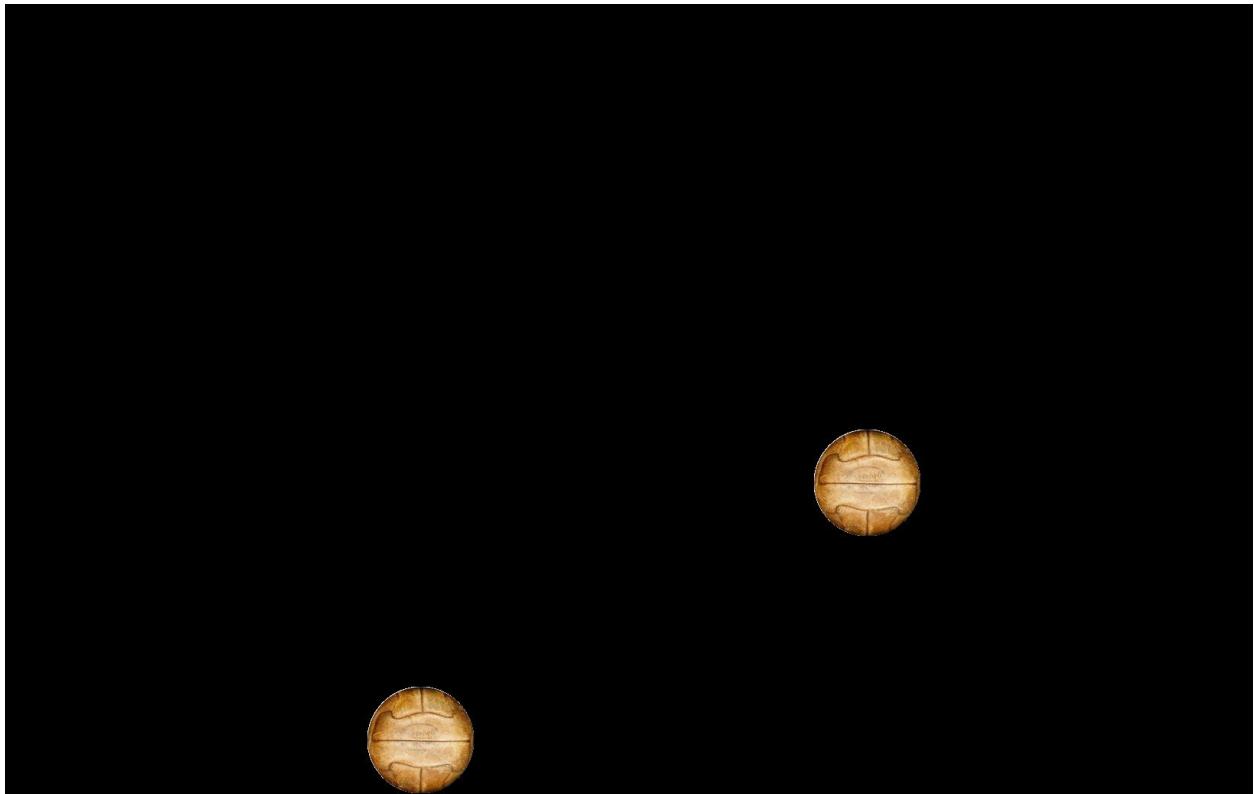
نوع هشتم توپ:



نوع نهم توب:



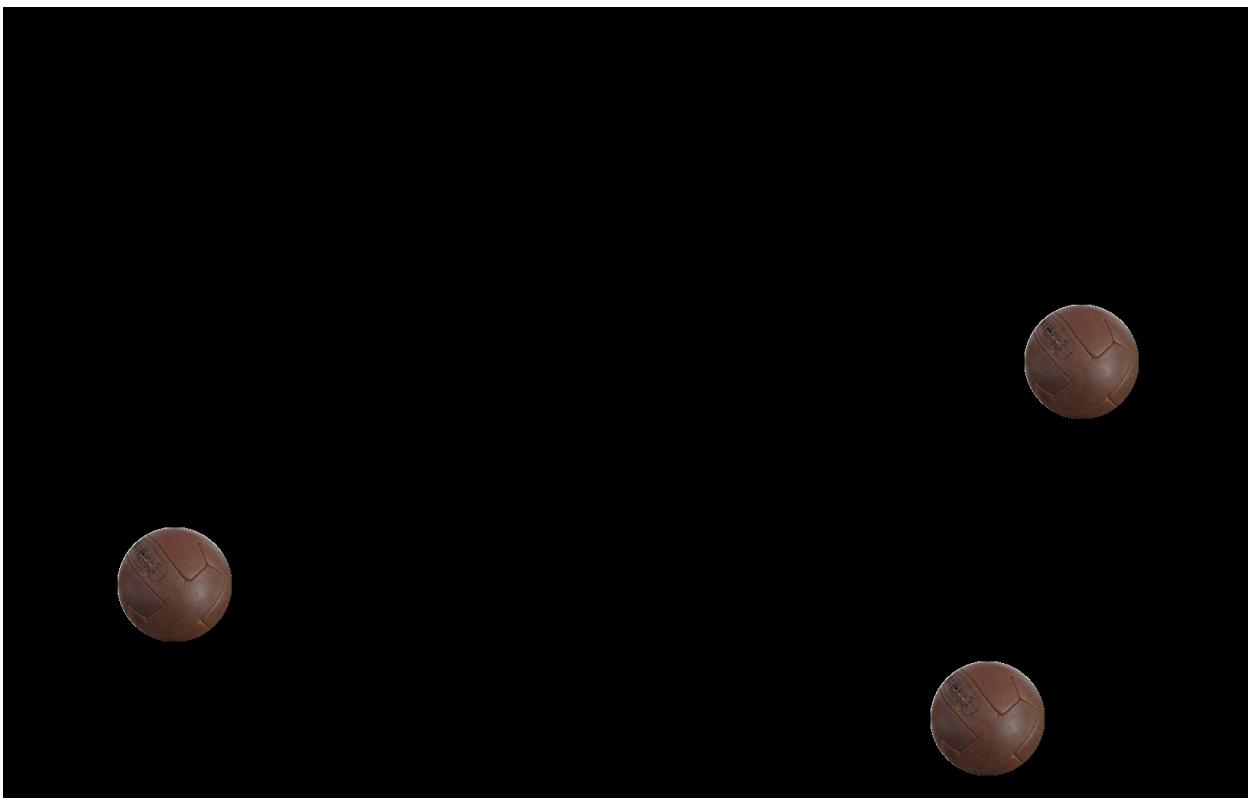
نوع دهم توپ:



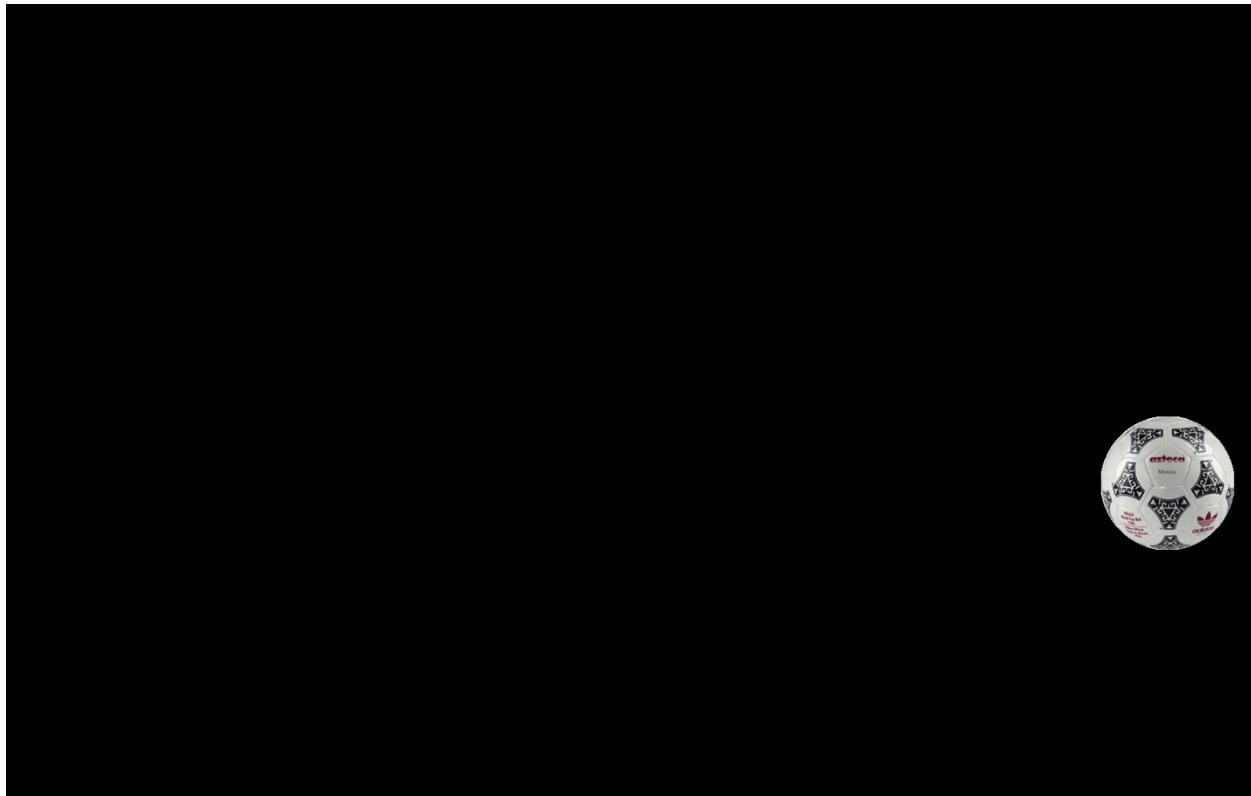
نوع یازدهم توپ:



نوع دوازدهم توب:



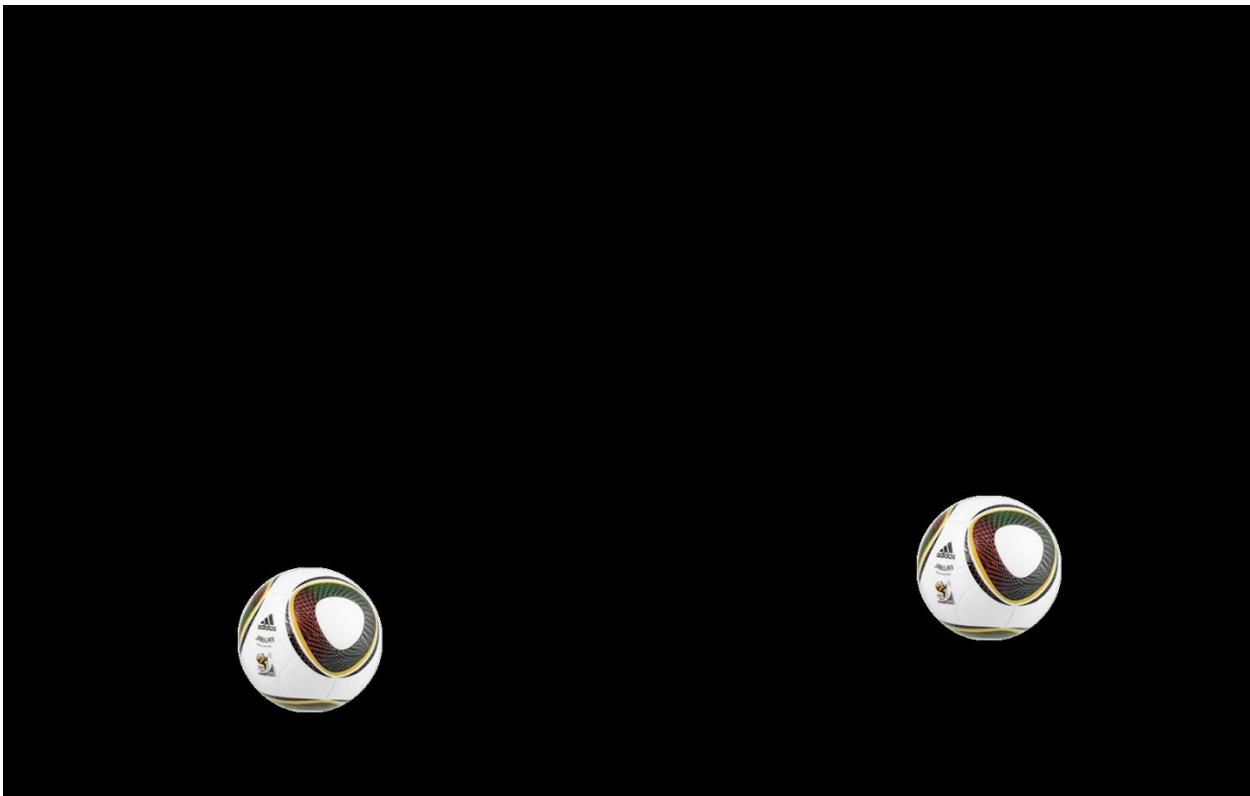
نوع سیزدهم توپ:



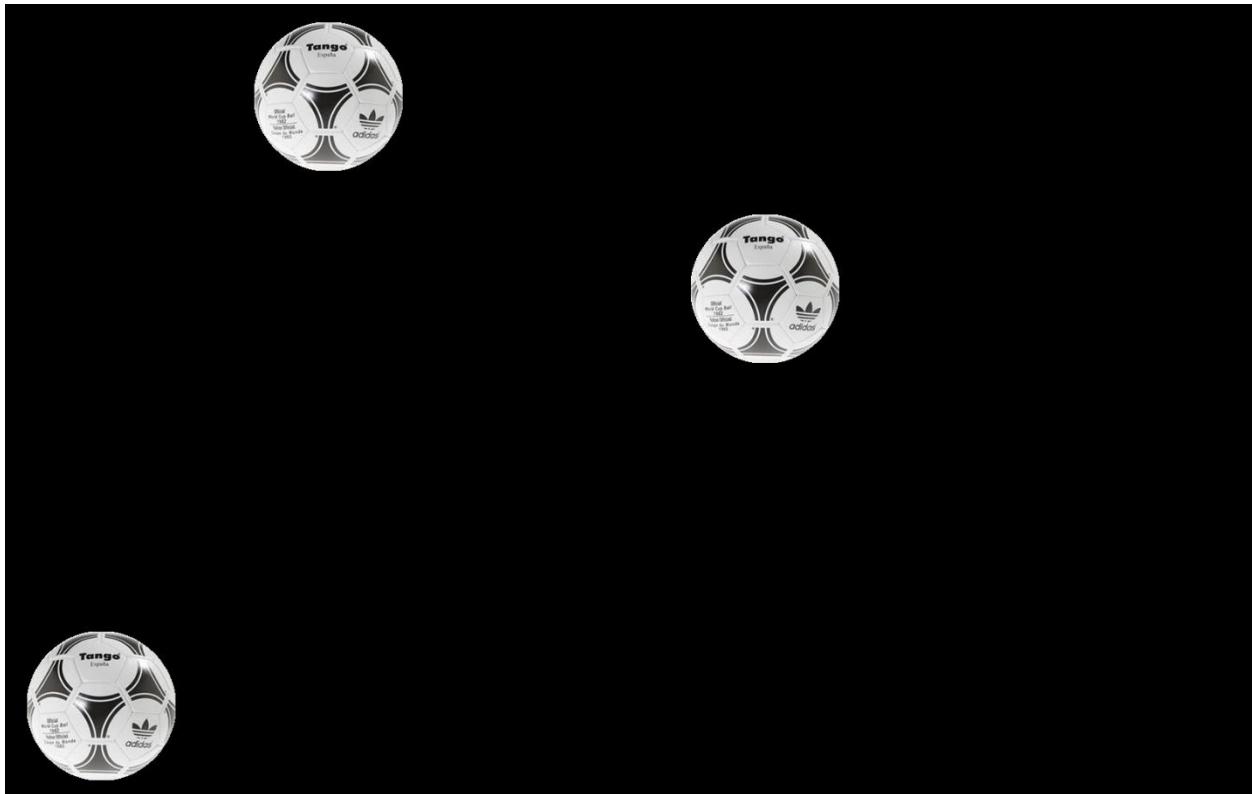
نوع چهاردهم توپ:



نوع پانزدهم توپ:



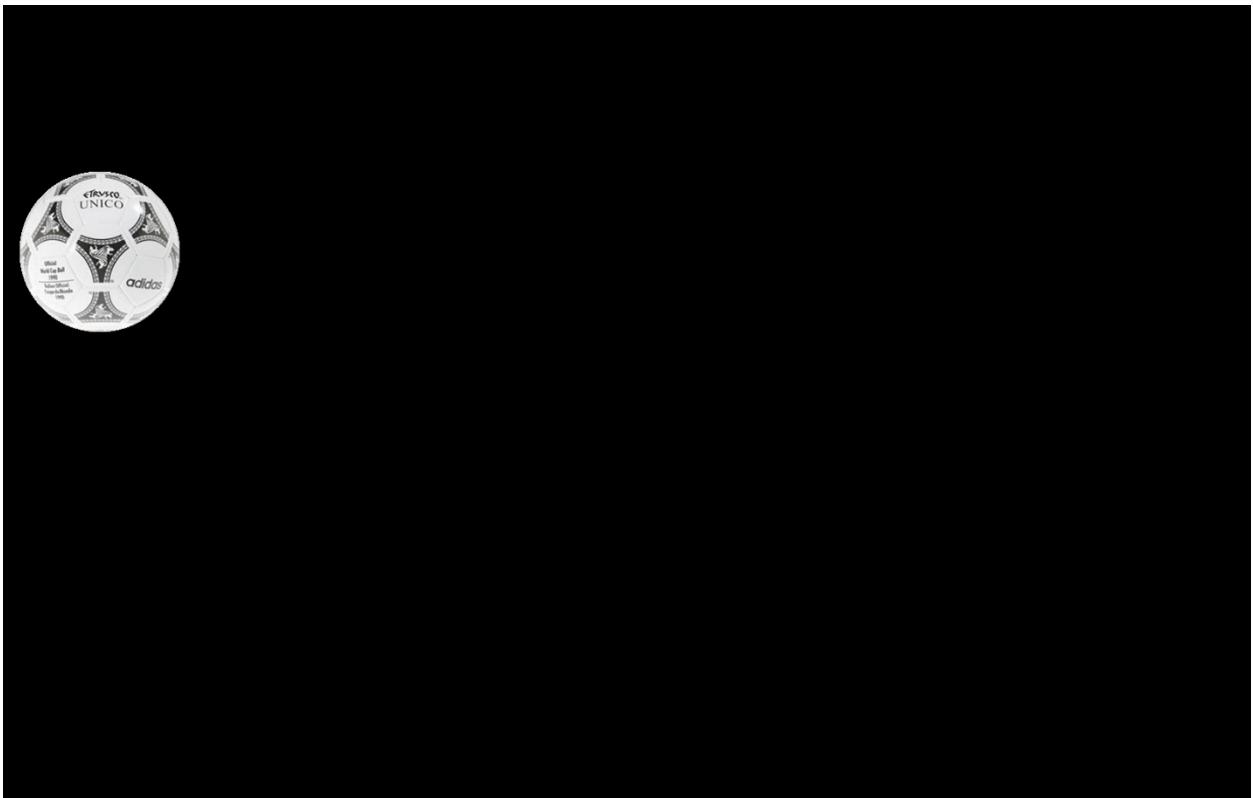
نوع شانزدهم توب:



نوع هفدهم توب:



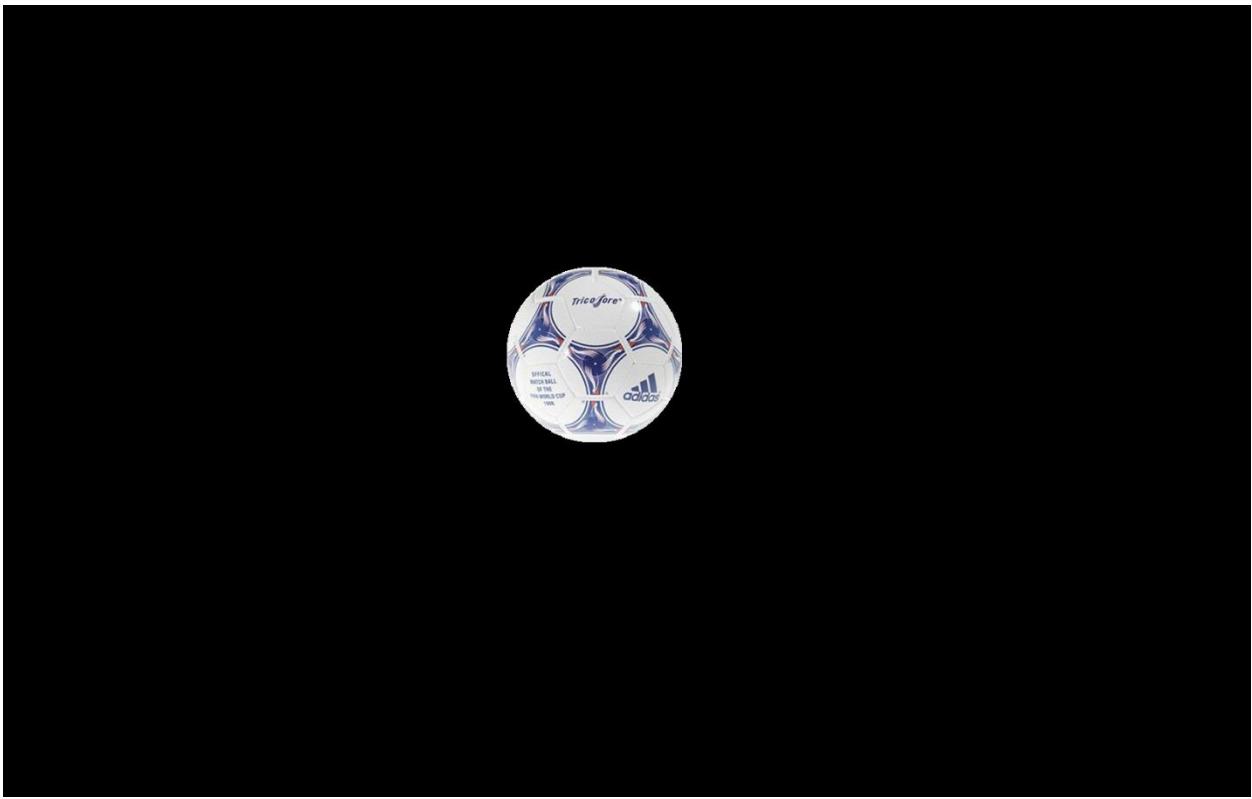
نوع هجدهم توب:



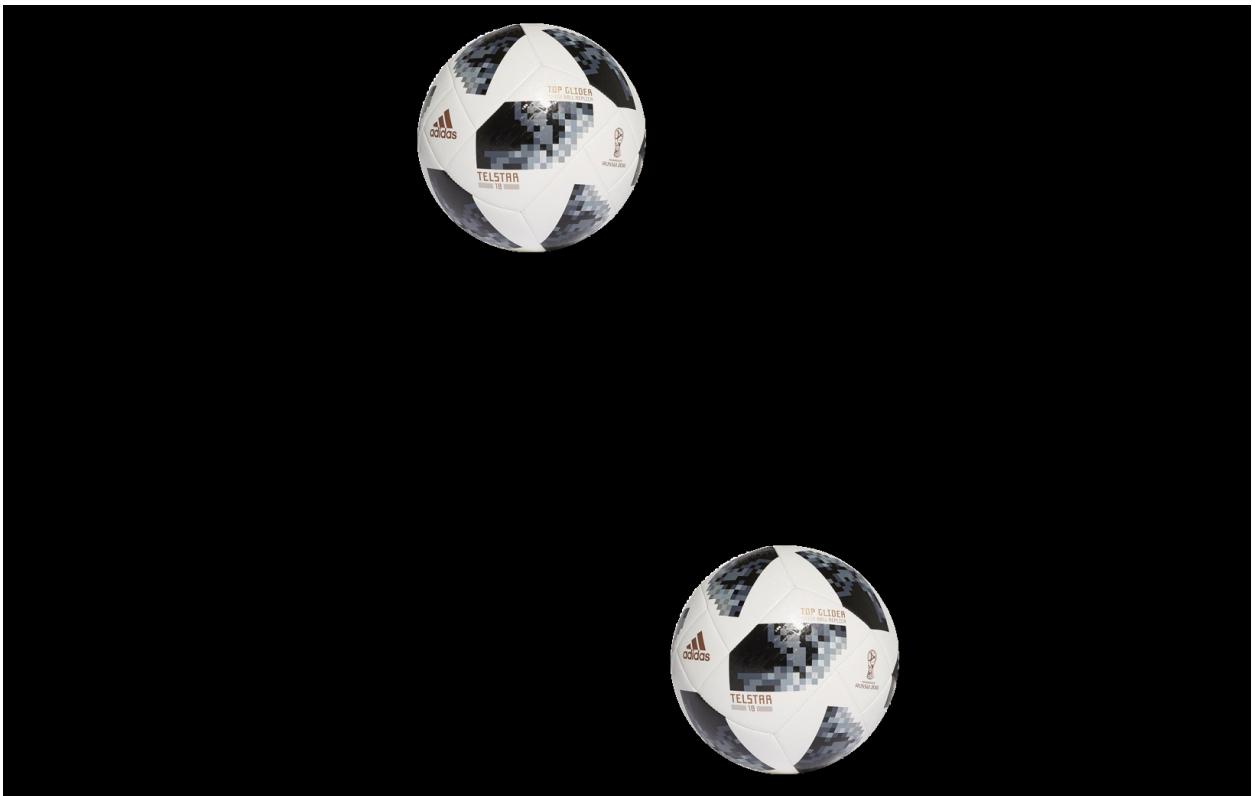
نوع نوزدهم توب:



نوع بیستم توب:



نوع بيستم و يكم توب:



تعداد توپ‌های هر نوع:

```
Command Window
> In images.internal.initSize (line 1)
In imshow (line 309)
In p8c (line 118)
Number of balls of each type
Columns 1 through 19

      5    2    3    5    2    4    2    3    3    2    2    3    1    1    2    3    2    1    1

Columns 20 through 21

      1    2

fx >> |
```

اکنون که تمام توپ‌ها را بر اساس نوعشان در تصویر جدا کردیم و تعداد هر کدام را شماردیم و می‌دانیم که هر نوع توپ مربوط به کدام برنده است، می‌توانیم تعداد توپ‌های برنده آدیداس را بشماریم. در زیر خروجی این کار را مشاهده می‌کنید:

توپ‌های برنده آدیداس:



تعداد:

Number of ball which belong to Adidas

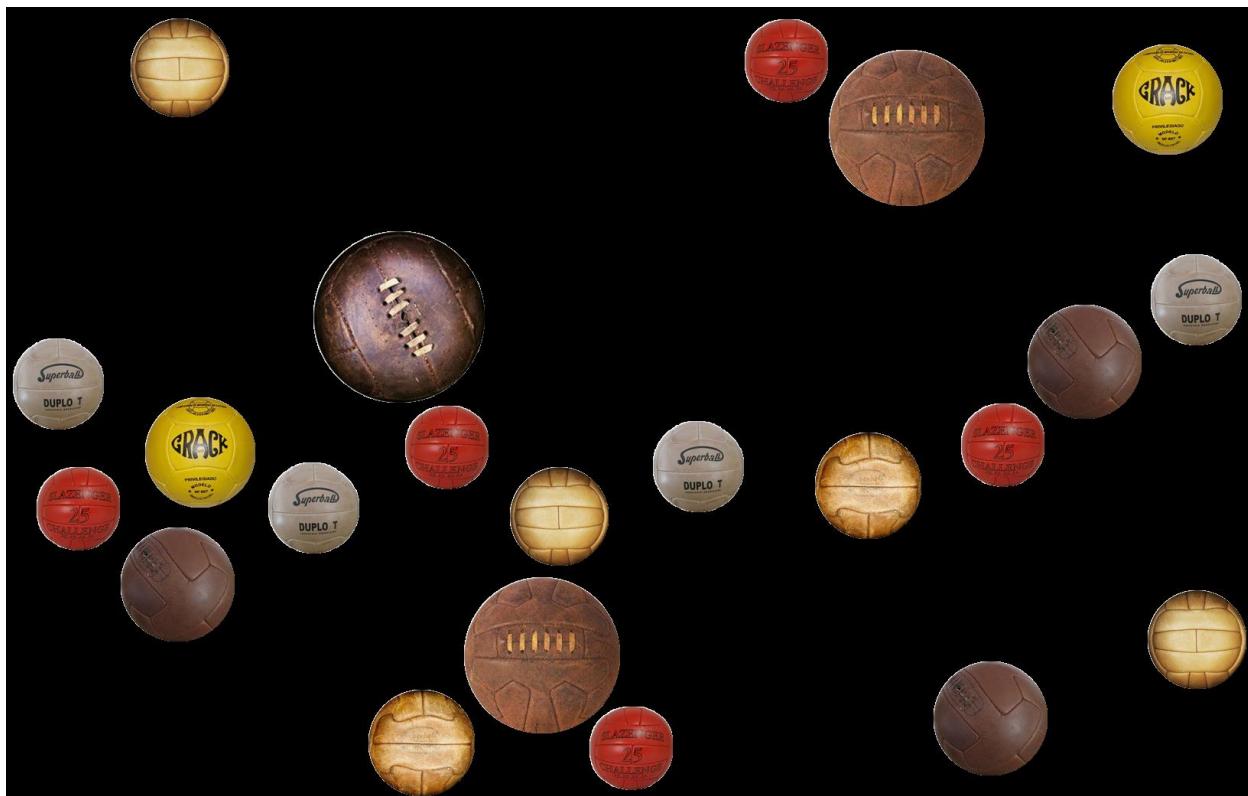
28

f_x >>

d قسمت

کدهای این قسمت در p8d.m قرار دارد. در این قسمت مانند قسمت قبل عمل می‌کنیم و فقط در قسمت آخر تغییر کوچکی به وجود می‌آوریم و توپ‌های غیر آدیداس را می‌شماریم.

توپ‌هایی که سازنده‌ی آن‌ها آدیداس نیست:



Number of ball which do not belong to Adidas

22

fx >>

e قسمت

کدهای این قسمت در p8e.m قرار دارد. در این قسمت مانند قسمت قبل عمل می‌کنیم و فقط در قسمت آخر تغییر کوچکی به وجود می‌آوریم و توپ‌هایی را که در جام جهانی‌هایی که در اروپا برگزار شده‌اند را می‌شماریم.

توپ‌هایی که در بازی‌هایی که در اروپا برگزار شده‌اند استفاده شده‌اند:



Number of ball which used in Europe
24

fx >>

تمرین ۹

قسمت ۲

دلیل استفاده از YCbCr این است که اطلاعات که در ۷ ذخیره می‌شود بسیار ارزشمند است. اطلاعات ۷ شدت روشنایی یا مقدار Gray-Scale Value است. اطلاعات **Cr** و **Cb**، اطلاعات رنگ هستند که اطلاعات شدت روشنایی از آن‌ها جدا شده است.

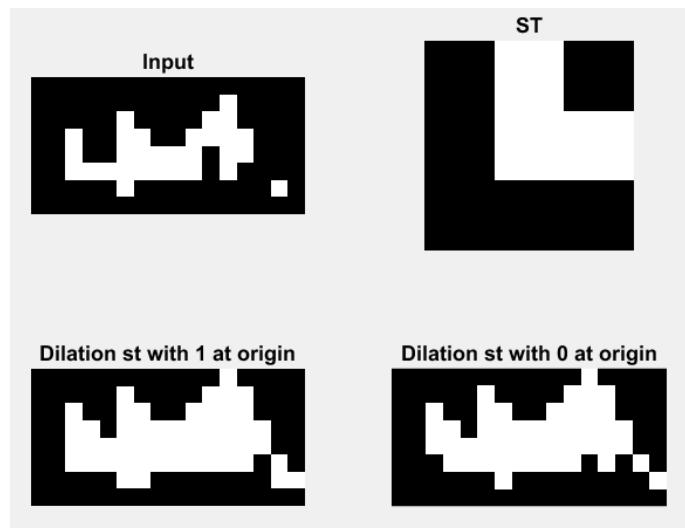
به دلیل کم بودن تعداد سلول‌های تشخیص رنگ در چشم نسبت به سلول‌های تشخیص روشنایی، چشمان ما در تشخیص تفاوت‌های رنگی جزئی ضعیف است ولی چشم نسبت به تغییرات شدت روشنایی بسیار قوی عمل می‌کند.

برای استفاده از این موضوع در **JPEG**، یک تصویر با با رزولوشن کم از **Cb** و **Cr** ذخیره می‌شود در حالی که ۷ با رزولوشن کامل ذخیره می‌شود. اطلاعات از دست رفته‌ی **Cr** و **Cb** در هنگام نیاز، با upscaling تخمین زده می‌شود که از روش‌های مختلفی می‌توان استفاده کرد. مانند استفاده از اطلاعات یکی از همسایگان یا .linear interpolation

قسمت b

کد این قسمت در p9b.m در پوشه‌ی P9 قرار دارد.

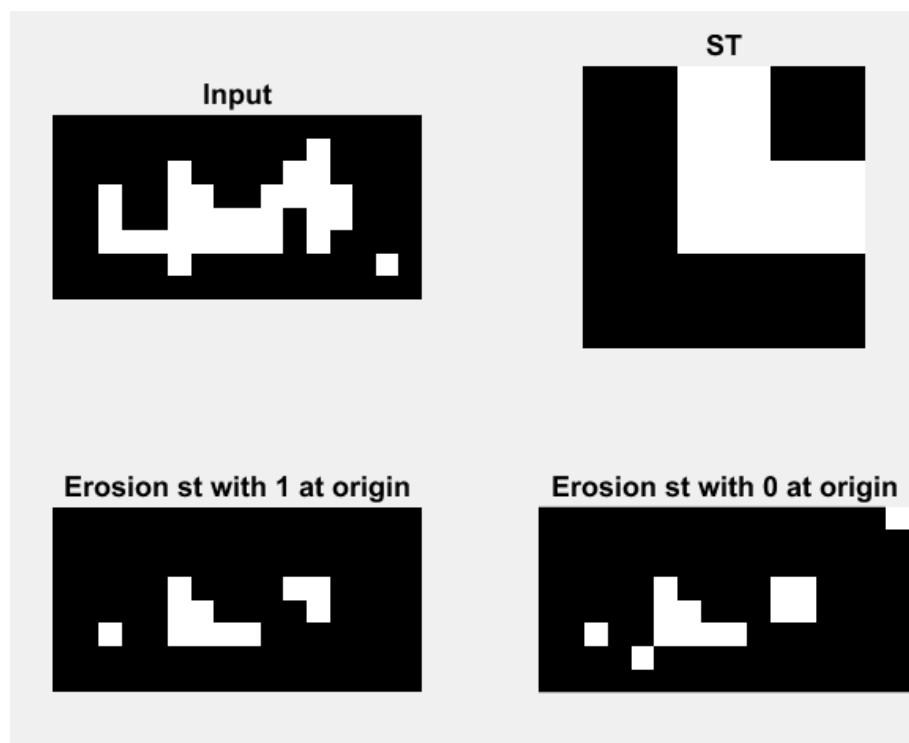
:Dilation در



در Dilation در صورتی که مرکز ساختار صفر باشد، در صورت دوران ۱۸۰ درجه‌ای ساختار و قرار دادن آن بر روی پیکسل زرآ، فقط در صورتی در خروجی مقدار پیکسل زرآ یک می‌شود که حداقل یکی از پیکسل‌های تصویر در همسایگی که مقدار یک دارد با یکی از یک‌های درون ساختار روی هم قرار بگیرند. در غیر این صورت صفر و یا یک بودن مقدار پیکسل در زرآ تفاوتی نمی‌کند. این موضوع را در تصویر بالا می‌بینید.

خروجی حاصل با ساختار با مرکز صفر زیر مجموعه‌ی خروجی ساختار با مذکور یک است.

: Erosion در



در Erosion ، با قرار دادن ساختار بر روی پیکسل زرآ، در صورتی در خروجی مقدار پیکسل زرآ یک می‌شود که تمام پیکسل‌های ساختار که مقدار یک دارد بر روی مقدارهای یک در همسایگی پیکسل زرآ تصویر قرار بگیرند. به همین دلیل در صورتی که مرکز ساختار صفر باشد دیگر مهم نیست مرکز همسایگی بر روی تصویر مقدار صفر و یا یک دارد.

خروجی حاصل با ساختار با مرکز یک زیر مجموعه‌ی خروجی ساختار با مذکور صفر است.

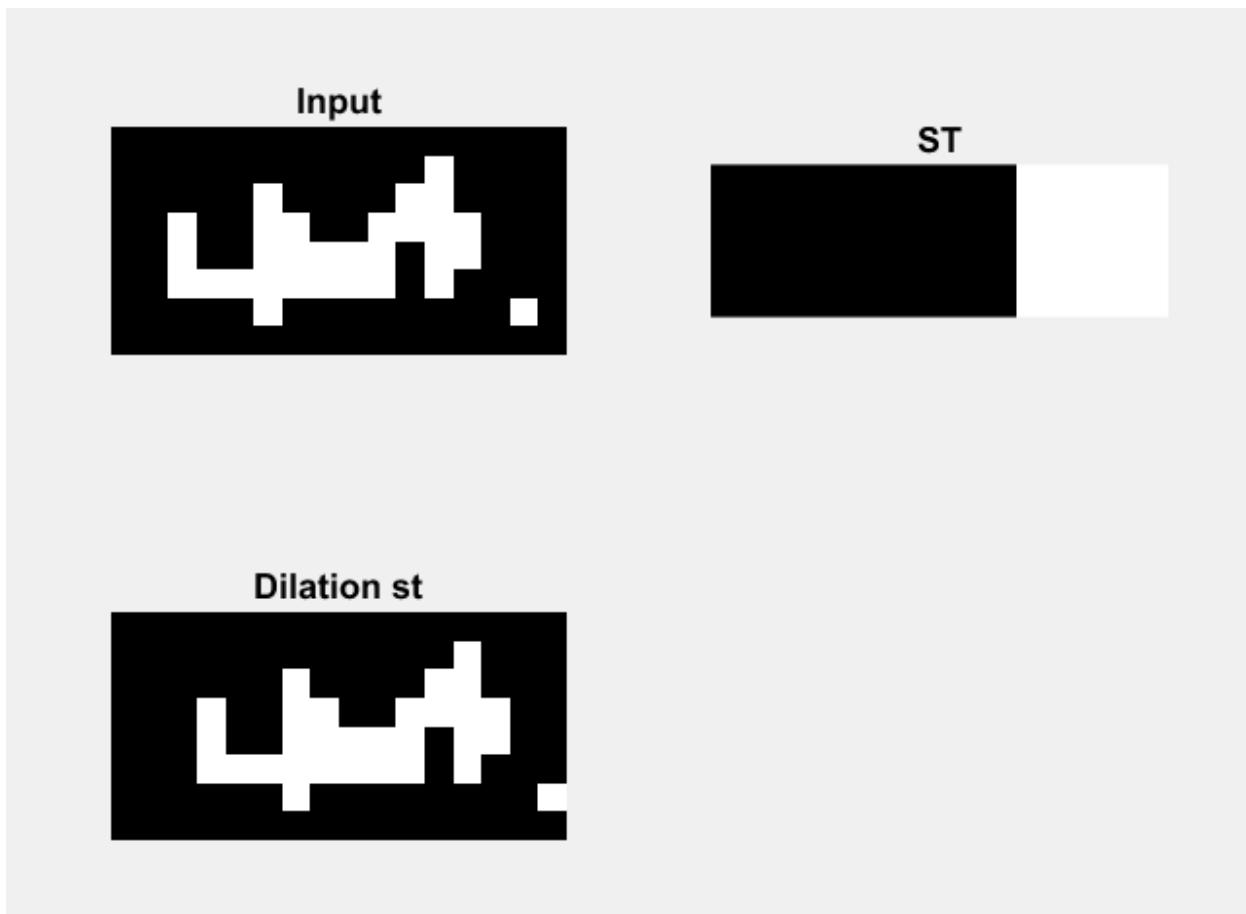
برای بررسی تغییرات در سایر روش‌های مورفولوژی به همین ترتیب بررسی می‌کنیم.

قسمت C

بر اساس مطلب گفته شده در صفحه ۶۲۸ و ۶۲۷ کتاب گنزالس، ویرایش سه، عملگرهای مورفولوژی یک سری عملگر بر روی مجموعه‌هایی با Z^n است، که برای موفولوژی باینری n برابر با ۲ است و برای موفولوژی Gray-Scale مقدار n برابر با ۳ است. برای تصویرهای رنگی $n > 3$ است. پس استفاده از آن‌ها ممکن است.

قسمت d

بله امکان پذیر است، به عنوان مثال کدی نوشتہام که در P9d.m قرار دارد. که خروجی این کد را مشاهده می‌کنید. با استفاده از ساختار [0 0 1] محتویات تصویر را یک پیکسل به راست، شیفت داده‌ام.



قسمت e

بله، این کار امکان پذیر است، در مقاله‌ی زیر این موضوع شرح داده شده است:

<https://ieeexplore.ieee.org/document/4766880/?arnumber=4766880>

The screenshot shows a web browser displaying an IEEE Xplore document page. The URL in the address bar is <https://ieeexplore.ieee.org/document/4766880/?arnumber=4766880>. The page header includes the IEEE Xplore logo and navigation links like 'Browse Journals & Magazines', 'Volume: PAMI-1 Issue: 1', and 'REQUEST A FREE TRIAL'. The main title of the article is 'Thinning Algorithms for Gray-Scale Pictures'. Below the title, there are three metrics: 53 Paper Citations, 5 Patent Citations, and 280 Full Text Views. The authors listed are Charles R. Dyer and Azriel Rosenfeld. A 'View All Authors' link is also present. To the right, a 'Related Articles' sidebar lists two papers: 'Modeling of the Douro River Plume Size, Obtained Through Image Segmentation of M...' and 'Performance-based classifier combination in atlas-based image segmentation using...'. A 'View All' link is at the bottom of the sidebar. At the bottom of the main content area, there are tabs for 'Abstract', 'Authors', 'Figures', 'References', 'Citations', 'Keywords', 'Metrics', and 'Media'. The 'Abstract' tab is currently selected.

ایده‌ی این مقاله استفاده از یک تعریف جدید مبنی بر وزن دار کردن ارتباط است. دو پیکسل به هم متصل هستند اگر یک مسیری بین دو پیکسل موجود باشد که از هیچ کدام از دو پیکسل روش‌تر نباشند.