

تمرین سری **دو**

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
۸	۸ قسمت b
۱۰	۱۰ تمرین ۲
۱۰	۱۰ قسمت a
۱۳	۱۳ قسمت b
۲۰	۲۰ تمرین ۳
۲۰	۲۰ قسمت a
۲۲	۲۲ قسمت b
۲۴	۲۴ قسمت c
۲۴	۲۴ قسمت d
۲۶	۲۶ قسمت e
۲۹	۲۹ قسمت g
۳۳	۳۳ تمرین ۴
۳۳	۳۳ قسمت a
۳۴	۳۴ قسمت b
۳۶	۳۶ قسمت c
۳۸	۳۸ قسمت d
۴۰	۴۰ قسمت e
۴۱	۴۱ قسمت f
۴۳	۴۳ قسمت g
۴۶	۴۶ تمرین ۵
۴۶	۴۶ قسمت a
۴۷	۴۷ قسمت b
۴۹	۴۹ قسمت c

٥٣.	قسمت d
٥٥.	قسمت e
٥٧.	قسمت f
٥٨.	قسمت g
٦٢.	تمرین ٦
٦٢.	قسمت a
٦٩.	قسمت b
٧٧.	سوال ٧
٧٧.	قسمت a
٨٨.	قسمت b
٩٧.	قسمت c
٩٨.	قسمت d
١٠٢.	سوال ٨
١٠٢.	قسمت a
١٠٦.	قسمت b
١١١.	قسمت c
١١٦.	سوال ٩
١١٦.	قسمت a
١١٩.	قسمت b
١٢٢.	قسمت c
١٢٥.	قسمت d
١٢٨.	قسمت e
١٣٢.	سوال ١٠
١٣٢.	قسمت a
١٣٨.	قسمت b
١٤٥.	سوال ١١
١٤٥.	قسمت a
١٥٠.	قسمت b

١٥٦.....	قسمت c
١٥٦.....	قسمت d
١٦٠.....	سؤال ١٢
١٦٠.....	قسمت a
١٦١.....	قسمت b
١٦١.....	قسمت c
١٦١.....	قسمت d
١٦٢.....	قسمت e
١٦٢.....	قسمت f

ابزارهای استفاده شده

زبان برنامه نویسی: Matlab 2016 a

محیط توسعه: R Matlab 2016a

سیستم عامل: ویندوز ۱۰

تمرین ۱

کدها و خروجی‌های این تابع در پوشه‌ی **p1** قرار دارد.

قسمت a

کدهای این بخش از سوال در فایل‌های **p1a_run_p.m** و **p1_a_func_p.m** قرار دارند.

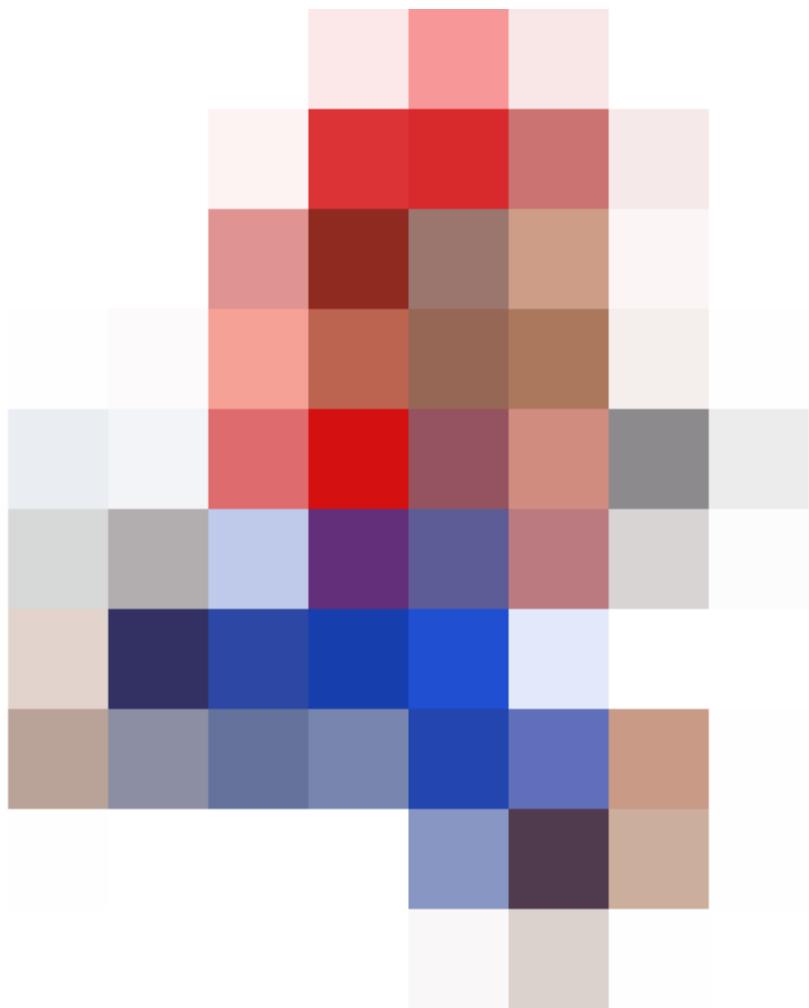
برای **pixelate** کردن تصویر و به طوری که اندازه‌ی تصویر ثابت بماند و تعداد پیکسل‌های درون تصویر به تعداد پیکسل‌های خواسته شده در ورودی بشود به روش زیر عمل کرده‌ام:

ابتدا طول و عرض تصویر را تقسیم بر تعداد پیکسل‌های طول و عرض خواسته شده می‌کنیم تا مشخص شود هر چند پیکسل را باید باهم یک گروه در نظر بگیریم و بعد آن‌ها را به یک پیکسل در تصویر جدید نسبت بدهیم. بعد از آن تصویر اصلی را متناسب با نسبت طول و عرض تصویر و تعداد پیکسل‌های خواسته شده به مربع‌هایی یکسان و تا حد ممکن جدا از هم و بدون هم پوشانی تقسیم می‌کنیم که هر کدام از مربع‌ها میانگین رنگ‌شان محاسبه می‌شود و سپس آن رنگ به تمام پیکسل‌های آن مربع داده می‌شود.

تصویر اصلی:



تصویر ۵۰۰*۵۰۰ حاوی ۱۰*۱۰ پیکسل(p1-a-p.png)



تصویر ۰۰۵۰۰ حاوی ۲۵*۲۵ پیکسل:(p1-b-p.png)



تصویر ۵۰۰*۵۰۰ حاوی ۵۰*۵۰ پیکسل:(p1-c-p.png)



تصویر ۵۰۰*۵۰۰ حاوی ۱۲۰*۱۲۰ پیکسل(p1-d-p.png)



تصویر ۵۰۰*۵۰۰ حاوی ۳۰۰*۳۰۰ پیکسل(p1-e-p.png)

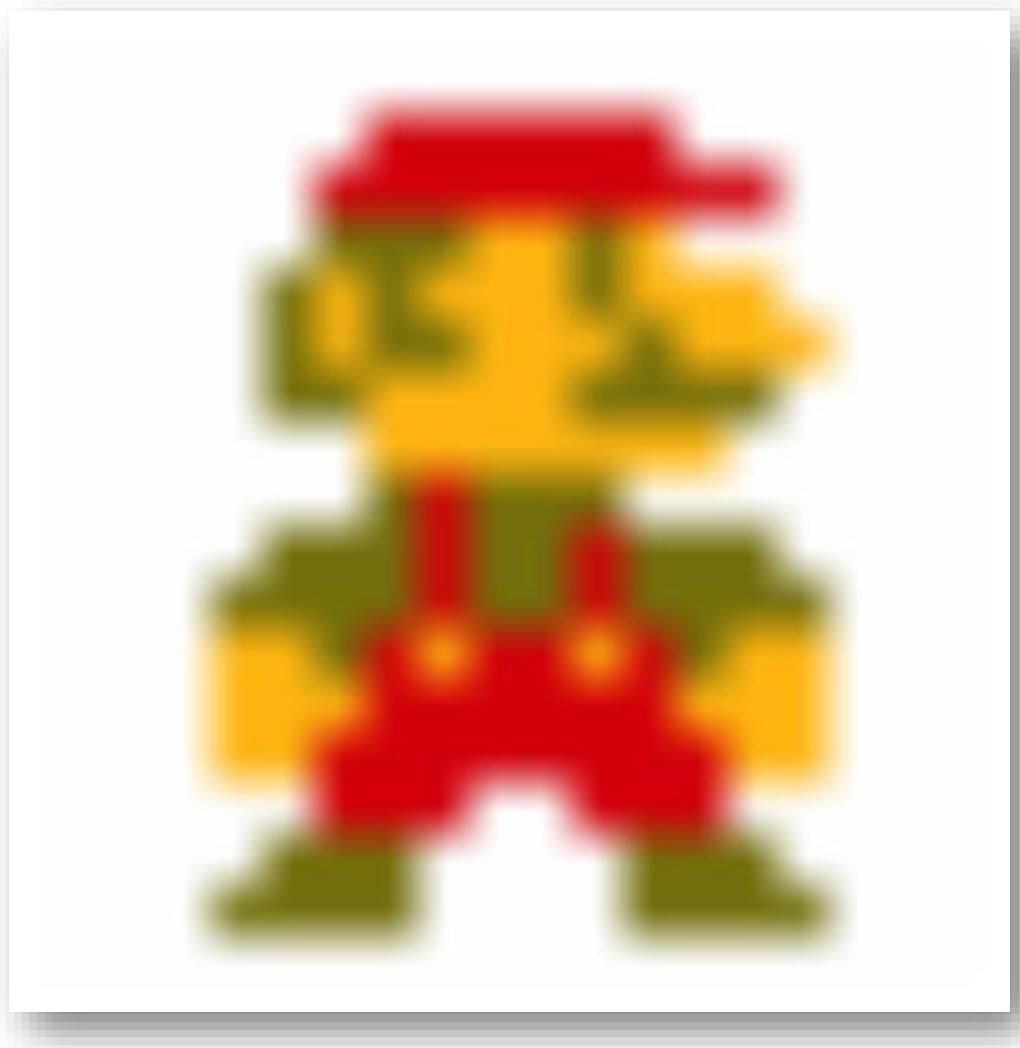


قسمت b

کد این قسمت در p1b_run.m قرار دارد. برای کم کردن شدت تصویر pixelate شده می‌توان از فیلتر های blur استفاده کرد مانند میانگین، Gaussian Blur و برای این قسمت از یک Gaussian Blur با اندازه‌ی ۳۱*۳۱ standard deviation برابر با ۱۳ استفاده کرده‌ام که با استفاده از فرمول زیر آن را تولید کرده‌ام.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

و بعد آن را با تصویر convolution کرده‌ام. در زیر خروجی این کار را مشاهده می‌کنید:



تصویر با نام gaussian_31_31_div_13 Blur.png در پوشه‌ی p1 ذخیره شده است.

تمرین ۲

قسمت a

کدهای این قسمت در پوشه‌ی p2 و در فایل‌های p2a_run.m و p2a_func.m قرار دارند.

در فایل p2a_func.m تابعی به فرم زیر نوشته‌ام:

```
function censored = p2a_func(img_RGB, point1, point2, k_size)
    % This function gets an RGB image and it censors a region that
    % specified by point1 and point2. It uses smoothing for censoring.
    % k_size is kernel size of smoothing filter, should be an odd number.
```

این تابع یک تصویر RGB را دریافت می‌کند و مستطیل بین دو نقطه‌ی point1 و point2 را با استفاده از Mean Filter بلور می‌کند که سایز Average Filter K_size مشخص می‌شود.

ابتدا یک فیلتر به سایز $3 * k_size * k_size$ می‌کنم که تمام درایه‌های آن برابر با یک است سپس این ماتریس سه بعدی را ضرب در $\frac{1}{k_size*k_size}$ می‌کنم. از این جهت ماتریس ۳ بعدی است که تصویر ورودی یک تصویر RGB است و در هر سه طیف باید کار بلور کردن را انجام دهیم و جهت استفاده از vectorization، فیلتر یک بعدی نیست.

بعد از ایجاد فیلتر از آنجایی که فیلتر متقارن است نیازی به چرخش فیلتر نیست. بعد از آن فیلتر را با نقطه‌های درون مستطیل convolution کرده‌ام. در زیر خروجی این تابع را برای سایزهای مختلف کرنل مشاهده می‌کنید:

تصویر ورودی:



خروجی با کرنل سایز ۹*۹ : (trump_9_in_9.png)



خروجی با کرنل سایز ۱۵*۱۵ : (trump_15_in_15.png)



خروجی با کرنل سایز (trump_25_in_25.png) ۲۵*۲۵



خروجی با کرنل سایز (trump_51_in_51.png) ۵۱ * ۵۱



قسمت b

کدهای این قسمت از سوال در پوشه‌ی p2 و در فایل‌های p2b_run.m و p2b_func.m قرار دارند. در p2b_func.m تابع زیر را نوشته‌ام:

```
function censord = p2b_func(img, point1, point2, desired_res)
    % This function gets an grayscale image and it censors a region that
    % specified by point1 and point2. It uses reducing resolution for censoring.
    % desired_res is new resolution of region.
```

این تابع یک تصویر Gray Scale را به عنوان ورودی دریافت می‌کند و سپس نسبت تعداد پیکسل‌های محور افقی و عمودی ناحیه بین دو نقطه‌ی point1 و point2 را نسبت به تعداد پیکسل‌های خواسته شده محاسبه می‌کند تا مشخص شود هر چند پیکسل معادل با یک پیکسل جدید می‌شوند. سپس در ناحیه‌ی خواسته شده، بر اساس نسبت بین پیکسل‌های ناحیه و پیکسل‌های خواسته شده، ناحیه به پیکسل‌هایی با سایز یکسان که همپوشانی ندارند تقسیم بندی می‌شوند و سپس میانگین شدت روشنایی پیکسل‌ها محاسبه می‌شود و شدت

روشنایی پیکسل جدید را برابر با شدت روشنایی میانگین قرار می دهیم. در فایل p2b-run.m این تابع برای رزولوشن‌های مختلف فراخوانده شده است. در زیر خروجی این کد را مشاهده می‌کنید:

تصویر اصلی:



تصویر با رزولوشن ۵ * ۵ : (Larry_5_in_5.png)



تصویر با رزولوشن ۱۰ * ۱۰ : (Larry_10_in_10.png)



تصویر با رزولوشن ۲۰ * ۲۰ : (Larry_20_in_20.png)



تصویر با رزولوشن ۳۰ * ۳۰ (: Larry_30_in_30.png)



تصویر با رزولوشن ۵۰*۵۰ (.png) (Larry_50_in_50.png)



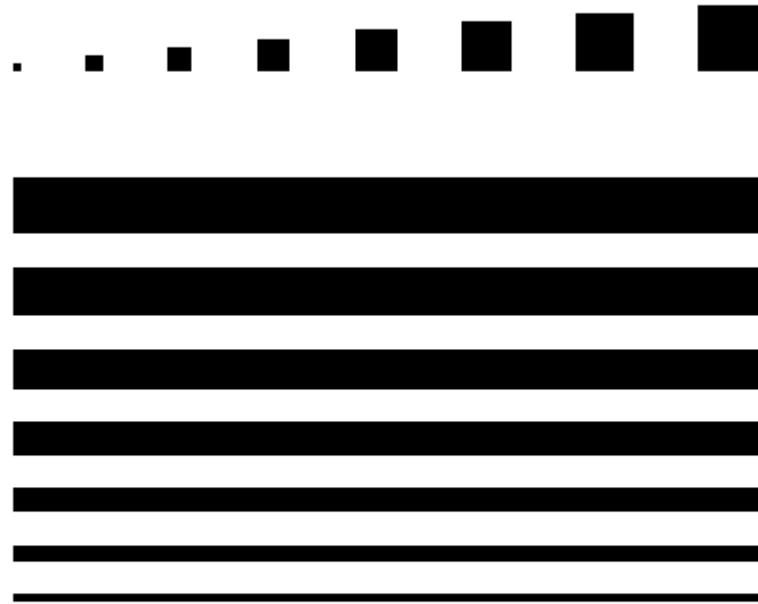
تمرین ۳

قسمت a

کد این قسمت از سوال در پوشه‌ی p3 و در فایل p3a_run قرار دارد. در این قسمت از سوال خواسته شده است که از فیلتری استفاده کنیم که فاصله‌ی بین خط‌ها را از بین ببرد. در کتاب گونزالز دو تصویر است که در یکی از آن‌ها عبارت MIN را نوشته است و در دیگری MAX و با اعمال فیلترهای min و max دو کلمه را بزرگ و کوچک کرده است به گونه‌ای که تغییر فرم خاصی رخ نداده است و به نظر می‌رسد که فونت کوچک و بزرگ شده است. برای از بین بردن فاصله‌ی بین خط‌ها از فیلتر min استفاده می‌کنم. در کد ابتدا تصویر را می‌خوانم سپس سایز فیلتر را یک فیلتر ۵*۵ انتخاب می‌کنم، سپس فیلتر را به اندازه‌ی لازم با استفاده از مقدار ۲۵۵ می‌کنم، سپس در درون تصویر شروع به حرکت می‌کنم و بر روی هر نقطه مربع ۵ * ۵ قرار می‌دهم و در آن همسایگی کمترین شدت روشنایی را پیدا می‌کنم و در مرکز مربع قرار می‌دهم. در آخر حاشیه‌ها را حذف می‌کنم و تصویر را نمایش می‌دهم. در زیر خروجی را مشاهده می‌کنید (برای دیدن خروجی در اندازه‌ی واقعی به فایل p3a.png مراجعه کنید):

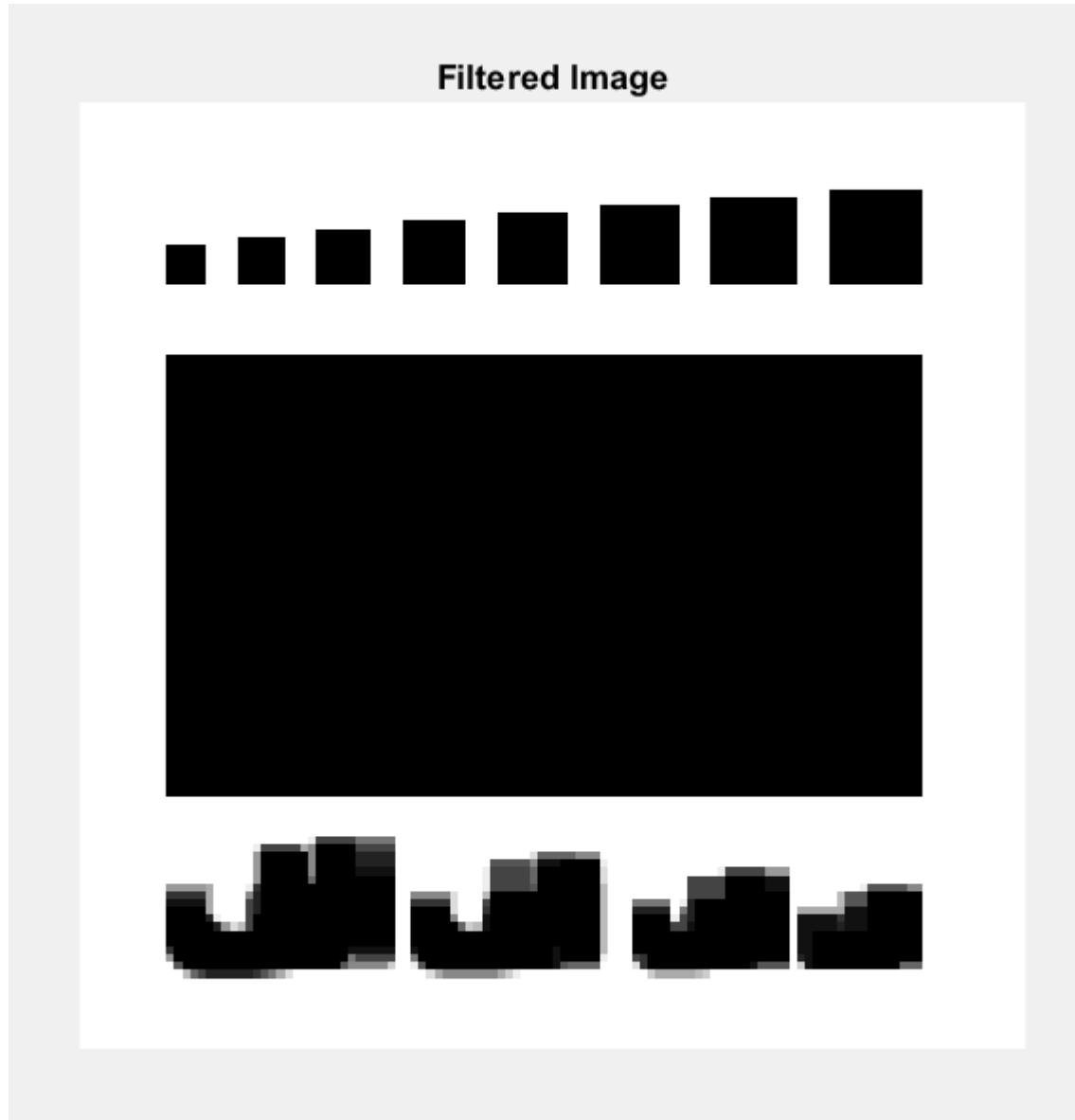
تصویر ورودی:

Input Image



الف الف الف الف

خروجی بعد از اعمال فیلتر 5×5 : min

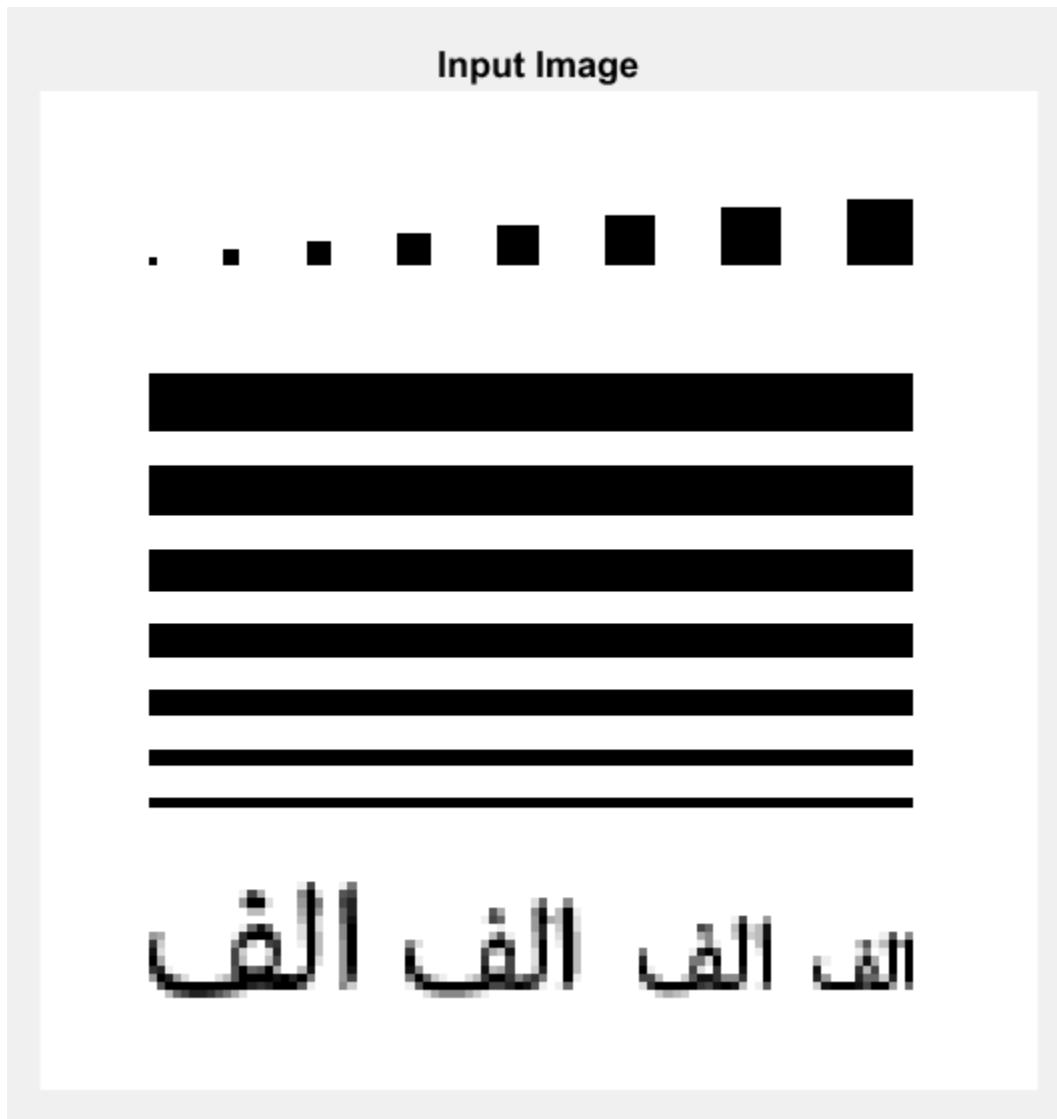


قسمت b

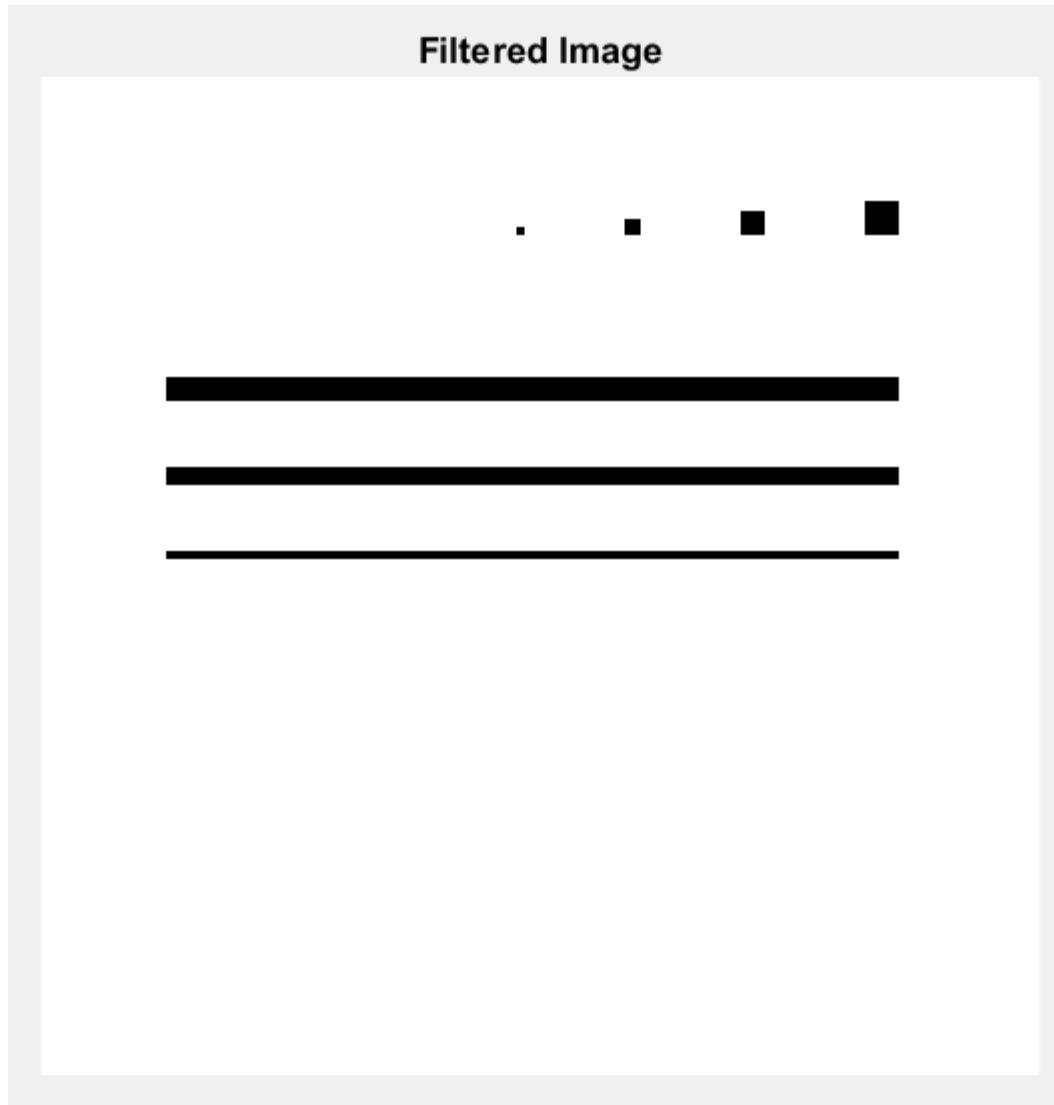
کد این قسمت از سوال در پوشه‌ی p3 و در فایل p3b_run.m قرار دارد در این قسمت از سوال از فیلتر غیر خطی 5×5 ماکریم استفاده کرده‌ام. در صورتی که یک همسایگی به قدر کافی بزرگ از پیکسل‌های حرف‌ها را انتخاب کنم، و ماکریم آن همسایگی را انتخاب کنم مقدار آن پیکسل‌ها از مقدار سیاه به سفید تغییر می‌کند و حرف حذف می‌شود. در کد ابتدا تصویر ورودی را خوانده‌ام سپس اندازه‌ی فیلتر را 5×5 در نظر گرفته‌ام. سپس تصویر را به اندازه‌ی کافی با پیکسل‌هایی با شدت روشنایی $\cdot pad$ کرده‌ام. در ادامه بر روی هر پیکسل فیلتر را اعمال کرده‌ام و همسایگی 5×5 آن پیکسل را پیدا کرده‌ام سپس ماکریم آن همسایگی را در آن نقطه قرار داده‌ام.

در زیر تصویر خروجی را مشاهده می کنید(برای دیدن تصویر در اندازه‌ی واقعی به p3b.png مراجعه کنید):

تصویر اصلی:



تصویر بعد از اعمال فیلتر ماکریم 5×5 :



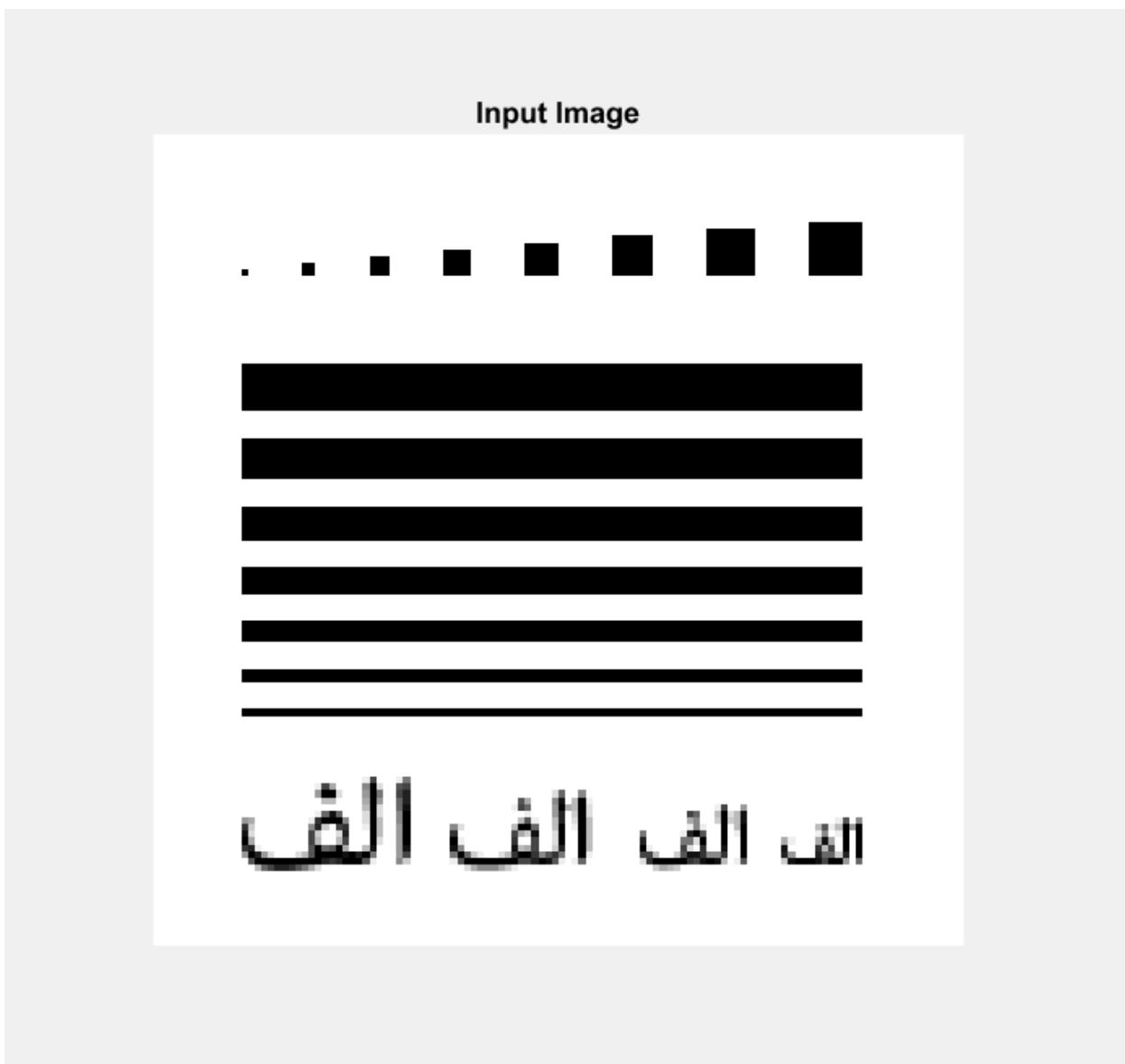
قسمت C

قسمت d

در این قسمت از سوال خواسته شده است که قسمت‌های سفید تصویر را به رنگ خاکستری و شدت روشنایی ۱۲۷ در آوریم. فیلتر identity فیلتر است که با طول و عرض مساوی که تمام درایه‌های آن برابر با صفر است بجز درایه در مرکز که برابر با یک است، که این فیلتر در صورت اعمال شدن همان تصویر قبل از فیلتر را می‌دهد.

برای اینکه قسمت سفید را به رنگ خاکستری با شدت روشنایی ۱۲۷ در بیاورم به جای یک د مرکز فیلتر عدد $\frac{127}{255}$ را قرار می‌دهم. کد این قسمت از سوال در پوشه‌ی p3 و در فایل p3d_rum.m قرار دارد. ابتدا تصویر را از فایل خوانده‌ام و سپس فیلتری که در بالا توضیح دادم را ایجاد می‌کنم و سپس فیلتر و تصویر را به تابع convo که نوشته‌ام و در بالا توضیح داده‌ام می‌دهم. در زیر تصویر خروجی کد را مشاهده می‌کنید (برای دیدن تصویر اصلی خروجی به p3ad.png مراجعه کنید):

تصویر ورودی:



تصویر خروجی:

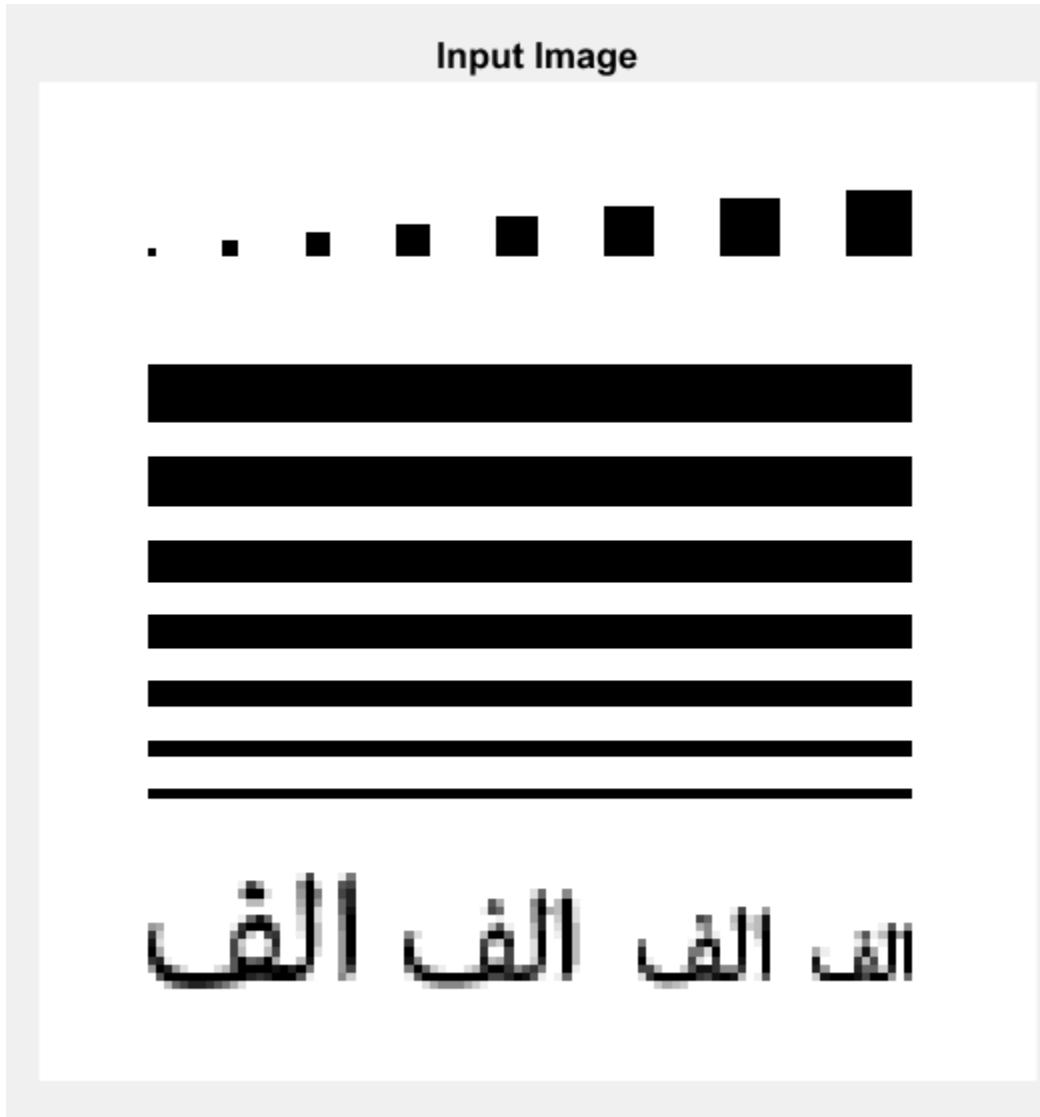


قسمت e

کد این قسمت در پوشه‌ی p3 و در فایل p3e_run.m قرار دارد. در این قسمت خواسته شده است که گوشی شکل‌های مربع/مستطیل را گرد کنیم. برای این کار یک فیلتر بلور کردن را مد نظر داشتم ولی از آنجایی که باید حالت دایره‌ای در گوشه‌ها ایجاد شود و هر چه از گوشه‌های تیز دور می‌شویم شدت روشنایی کمتر می‌شود، از

فیلتر گاوس استفاده می‌کنم. به همین دلیل از یک فیلتر 3×3 گاوس برای بلور کردن تصویر استفاده کرده‌ام. خروجی تصویر را در زیر مشاهده می‌کنید (برای دیدن تصویر در اندازه‌ی واقعی به [p3e.png](#) مراجعه کنید).

تصویر ورودی:



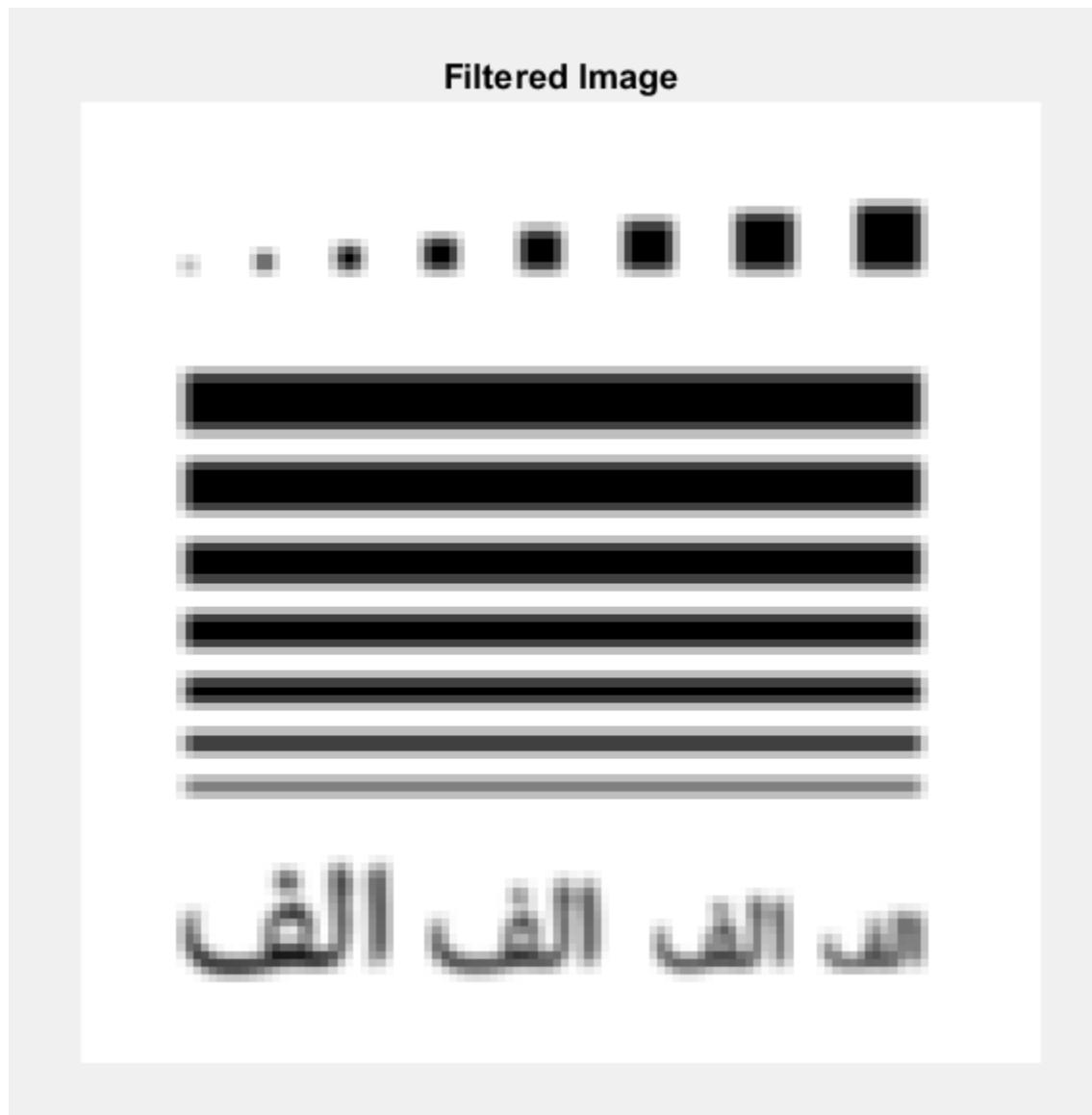
تصویر خروجی:

تصویر در سایز اصلی که گوشها گرد به نظر می‌رسند



ف الف الف

تصوير با بزرگنمایی:



قسمت g

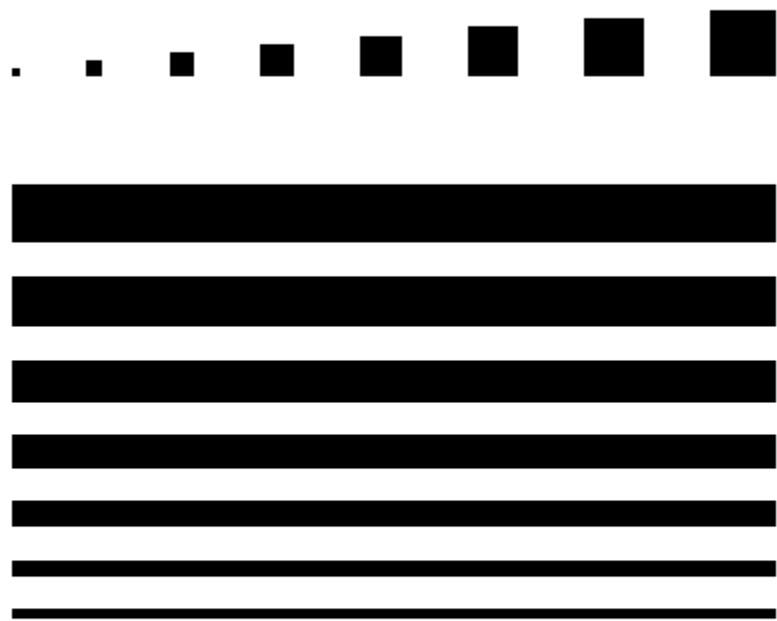
در این قسمت خواسته شده است که شکل های مستطیلی را به صورت خط های عمودی در بیاوریم. فیلتر های مشتق اول left sobel تغییرات در جهت افقی را نشان می دهد و نسبت به تغییرات عمودی حساس نیست و لبه ها را مشخص می کند و از آنجایی شکل های مستطیلی رنگ شان با پس زمینه متفاوت است سمت چت شکل به عنوان لبه تشخیص داده می شود و به صورت یک خط عمودی در می آید.

کد این بخش در پوشه‌ی p3 و در فایل p3g_run.m قرار دارد. ابتدا تصویر را خوانده ام سپس یک فیلتر 3×3 left sobel ایجاد کرده ام و با استفاده از تابع convo که نوشته ام و آن را در بالاتر توضیح داده ام تصویر را فیلتر کرده ام. البته رنگ تصویر بر عکس است و به همین خاطر ۲۵۵ را منهای مقدار پیکسل ها کرده ام تا رنگ تصویر مانند تصویر ورودی شود. در زیر خروجی ها را مشاهده می کنید (برای دیدن خروجی ها در اندازه واقعی به مراجعه کنید):

p3g-b.png و p3g-a.png

تصویر ورودی:

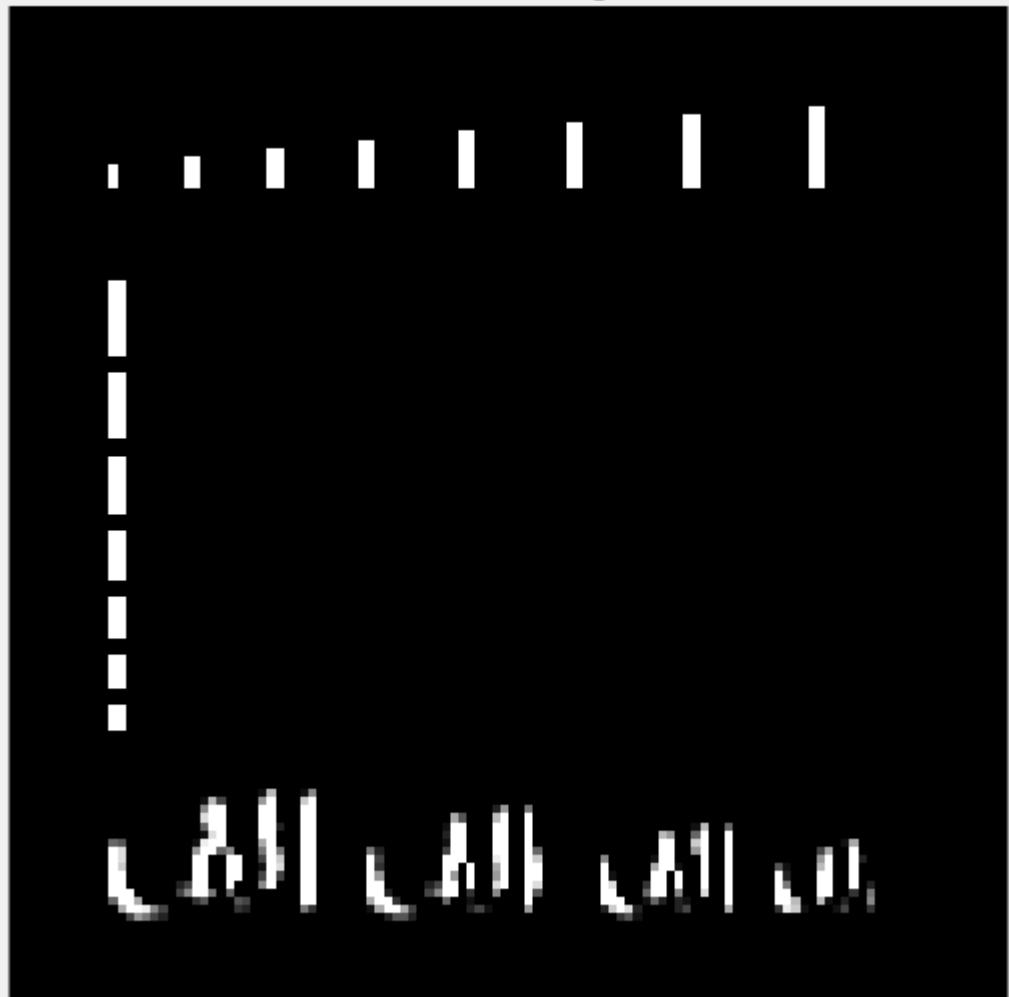
Input Image



الف الف الف

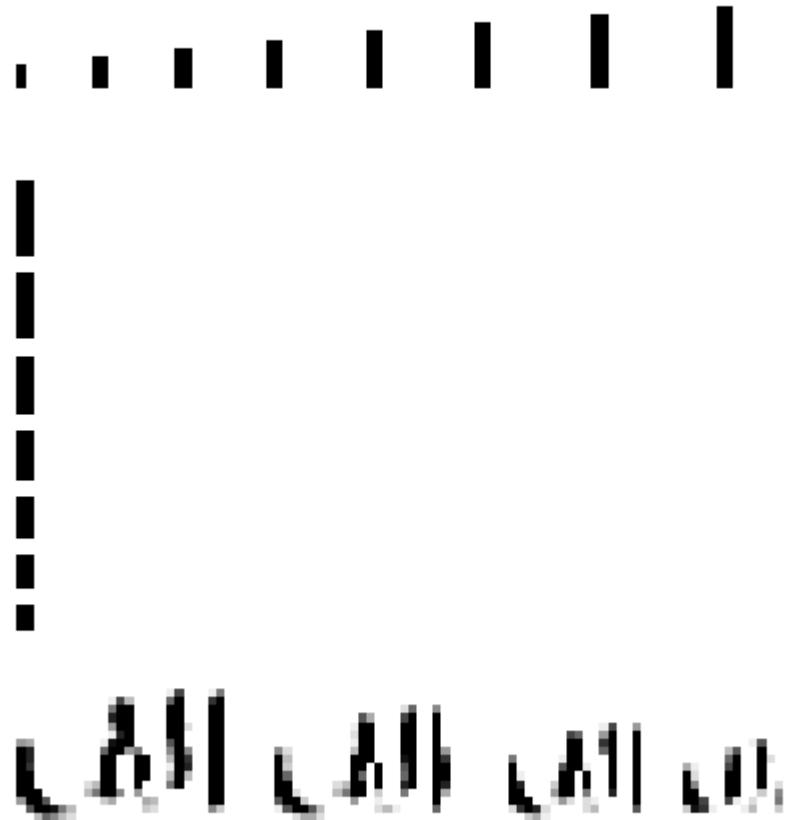
تصویر بعد از اعمال فیلتر sobel left :

Filtered Image



تصویر بعد از تغییر رنگ خروجی بالا:

Negative Filtered Image



تمرین ۴

کدهای این قسمت در پوشه‌ی p4 قرار دارد.

قسمت a

کد این قسمت در p4a.m قرار دارد. در kid_target_1.jpg لبه‌های افقی حفظ شده است پس باید فیلتر چیزی مانند sobel_x و چیزهای شبیه به آن باشد. با آزمون و خطا به این فیلتر رسیدم که خروجی بسیار شبیه به تصویر فیلتر شده است

```
% appropriate filter  
my_filter = [[0,0,0];...  
[1,0,-1];...  
[0,0,0]];
```

در کد تصویرها را خواندم و با استفاده از تابع conv0 که در مسیله‌ی قبل از آن استفاده کردم فیلتر را بر تصویر اعمال کردم. در زیر خروجی‌های کد را مشاهده می‌کنید:

تصویر فیلتر شده‌ی ورودی:



تصویر فیلتر شده حاصل از فیلتر طراحی شده (p4a-filtered-image.png)



با اینکه دو تصویر بسیار شبیه هم اند و با چشم تفاوتی در آنها دیده نمی‌شود ولی مقدار `isequal` برابر نشد که می‌تواند دلیل سیاست مرزی باشد.

قسمت b

کد این قسمت در p4b.m قرار دارد. در kid_target_2.jpg لبه‌های عمودی حفظ شده است پس باید فیلتر چیزی مانند `sobel` و چیزهای شبیه به آن باشد. با آزمون و خطا به این فیلتر رسیدم که خروجی بسیار شبیه به تصویر فیلتر شده است

```
% appropriate filter
my_filter = [[0,-1,0];...
              [0,0,0];...
              [0,1,0]];
```

در کد تصویرها را خواندم و با استفاده از تابع CONVO که در مسیله‌ی قبل از آن استفاده کردم فیلتر را بر تصویر اعمال کردم. در زیر خروجی‌های کد را مشاهده می‌کنید:

تصویر فیلتر شده‌ی ورودی:



تصویر فیلتر شده حاصل از فیلتر طراحی شده (p4b-filtered-image.png):



با اینکه دو تصویر بسیار شبیه هم اند و با چشم تفاوتی در آنها دیده نمی‌شود ولی مقدار `isequal` برابر نشد که می‌تواند دلیل سیاست مرزی باشد.

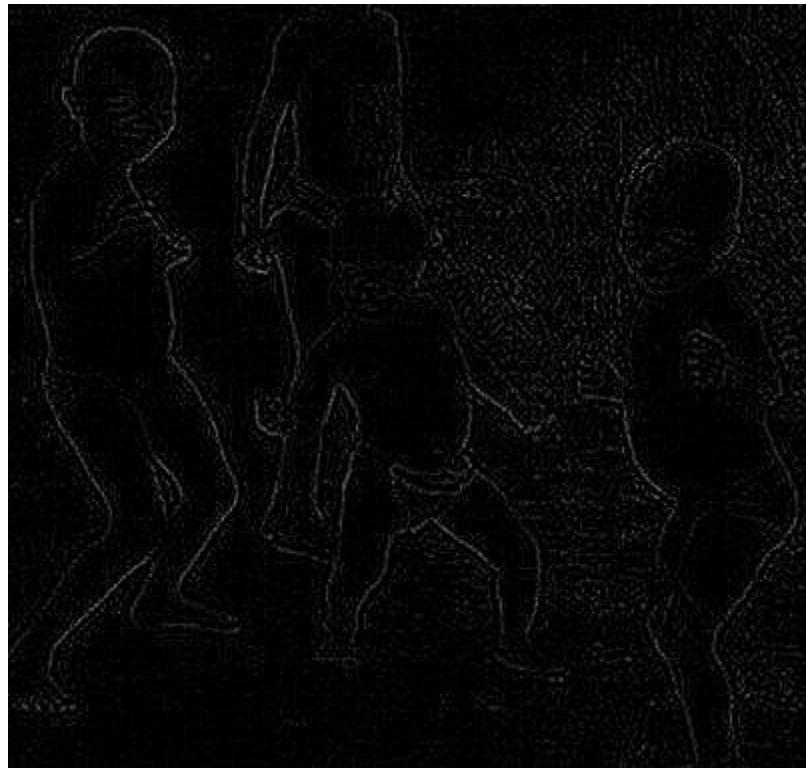
قسمت C

کد این قسمت در `p4c.m` قرار دارد. در `kid_target_3.jpg` لبه‌های و افقی عمودی حفظ شده اند و لبه‌ها باریک اند پس باید فیلتر چیزی مانند لاپلاسین و چیزهای شبیه به آن باشد. با آزمون و خطا به این فیلتر رسیدم که خروجی بسیار شبیه به تصویر فیلتر شده است

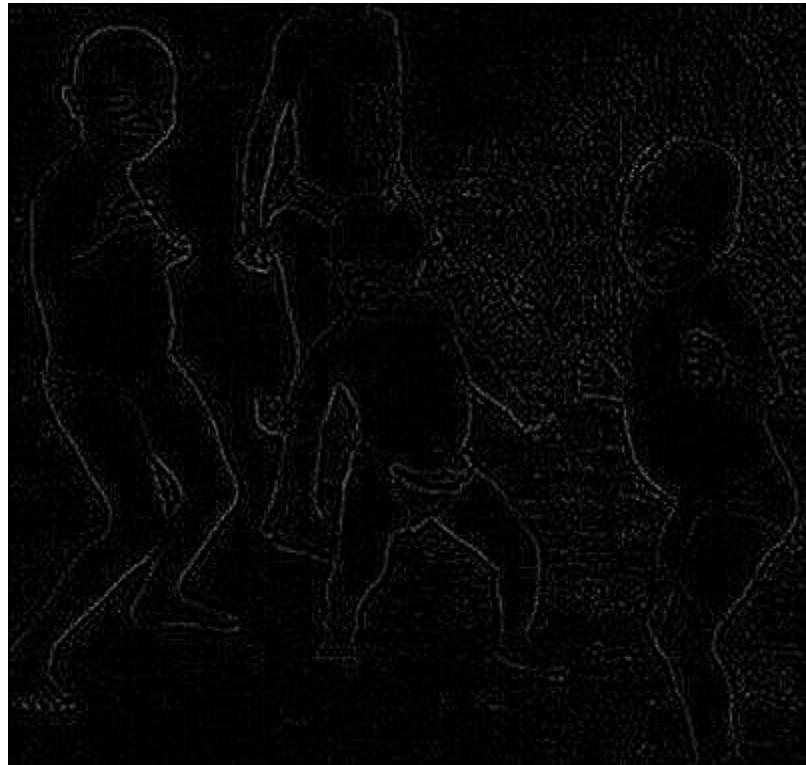
```
% appropriate filter
my_filter = [[0,1,0];
              [1,-4,1];
              [0,1,0]];
```

در کد تصویرها را خواندم و با استفاده از تابع `convO` که در مسیله‌ی قبل از آن استفاده کردم فیلتر را بر تصویر اعمال کردم. در زیر خروجی‌های کد را مشاهده می‌کنید:

تصویر فیلتر شده‌ی ورودی:



تصویر فیلتر شده حاصل از فیلتر طراحی شده (p4c-filtered-image.png)



با اینکه دو تصویر بسیار شبیه هم اند و با چشم تفاوتی در آنها دیده نمی‌شود ولی مقدار `isequal` برابر نشد که می‌تواند دلیل سیاست مرزی باشد.

d قسمت

کد این قسمت در `p4b.m` قرار دارد. در `kid_target_4.jpg` لبها برجسته‌تر شده‌اند و تصویر شارپ شده است، لبها عمودی و افقی حفظ شده اند پس باید تصویر با فیلتری مانند لاپلاس و چیزهای شبیه به آن شارپ شده باشد. با آزمون و خطا به این فیلتر رسیدم که خروجی بسیار شبیه به تصویر فیلتر شده است

```
% appropriate filter
my_filter = [ [0,-1,0];
               [-1,5,-1];
               [0,-1,0] ];
```

در کد تصویرها را خواندم و با استفاده از تابع `conv2` که در مسیله‌ی قبل از آن استفاده کردم فیلتر را بر تصویر اعمال کردم. در زیر خروجی‌های کد را مشاهده می‌کنید:

تصویر فیلتر شده‌ی ورودی:



تصویر فیلتر شده حاصل از فیلتر طراحی شده (p4d-filtered-image.png)



با اینکه دو تصویر بسیار شبیه هم اند و با چشم تفاوتی در آنها دیده نمی‌شود ولی مقدار `isequal` برابر نشد که می‌تواند دلیل سیاست مرزی باشد.

قسمت e

کد این قسمت در `p4e.m` قرار دارد. در `kid_target_5.jpg` لبه‌های در راستای قطر حفظ شده است با آزمون و خطابه این فیلتر رسیدم که خروجی بسیار شبیه به تصویر فیلتر شده است

```
% appropriate filter
my_filter = [ [-1,0,0];
               [0,0,0];
               [0,0,1]];
```

در کد تصویرها را خواندم و با استفاده از تابع `conv2` که در مسیله‌ی قبل از آن استفاده کردم فیلتر را بر تصویر اعمال کردم. در زیر خروجی‌های کد را مشاهده می‌کنید:

تصویر فیلتر شده‌ی ورودی:



تصویر فیلتر شده حاصل از فیلتر طراحی شده (p4e-filtered-image.png):



با اینکه دو تصویر بسیار شبیه هم اند و با چشم تفاوتی در آنها دیده نمی‌شود ولی مقدار `isequal` برابر نشد که می‌تواند دلیل سیاست مرزی باشد.

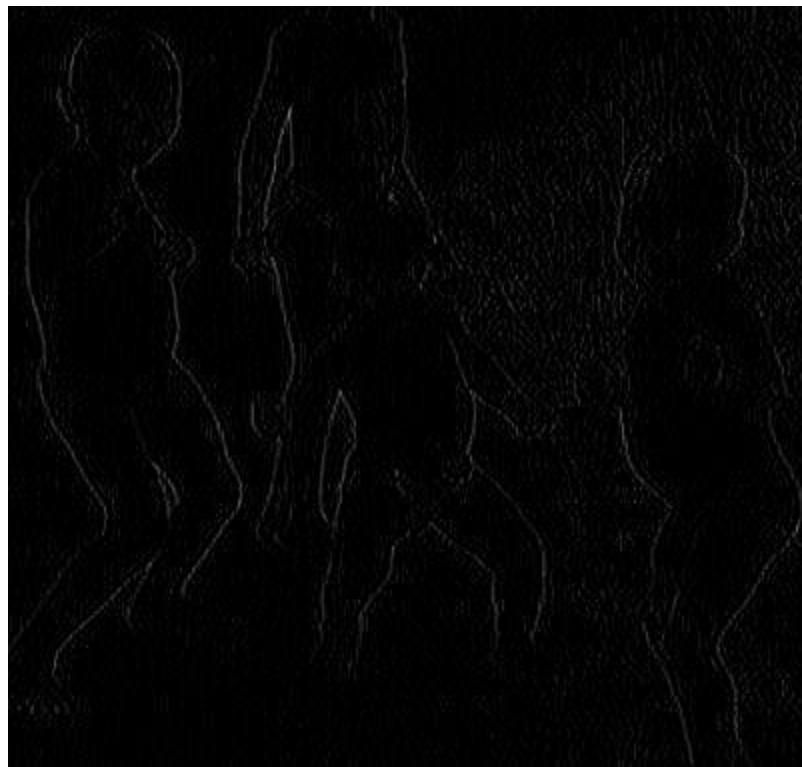
قسمت f

کد این قسمت در p4f.m قرار دارد. در kid_target_6.jpg لبه‌های افقی از سمت راست به چپ و چپ به راست حفظ شده اند. با آزمون و خطا به این فیلتر رسیدم که خروجی بسیار شبیه به تصویر فیلتر شده است

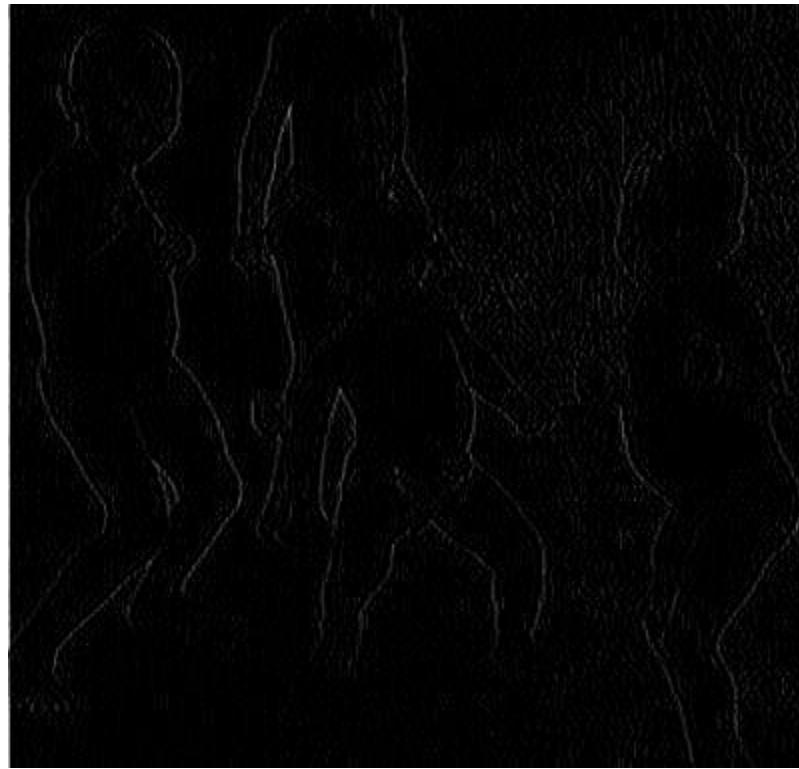
```
% appropriate filter
my_filter = [[0,0,0];...
              [-1,2,-1];...
              [0,0,0]];
```

در کد تصویرها را خواندم و با استفاده از تابع `CONVO` که در مسیله‌ی قبل از آن استفاده کردم فیلتر را بر تصویر اعمال کردم. در زیر خروجی‌های کد را مشاهده می‌کنید:

تصویر فیلتر شده‌ی ورودی:



تصویر فیلتر شده حاصل از فیلتر طراحی شده (p4f-filtered-image.png):



با اینکه دو تصویر بسیار شبیه هم اند و با چشم تفاوتی در آنها دیده نمی‌شود ولی مقدار `isequal` برابر نشد که می‌تواند دلیل سیاست مرزی باشد.

قسمت g

کد این قسمت در `p4g.m` قرار دارد. در `kid_target_7.jpg` لبه‌های عمودی و افقی برجسته‌تر شده‌اند و لبه‌ها باریک هستند پس تصویر شارپ شده‌است ولی شارپ شدن به گونه‌ای بوده است که شدت روشنایی کل تصویر را افزایش داده است. با آزمون و خطا به این فیلتر رسیدم که خروجی بسیار شبیه به تصویر فیلتر شده است

```
% appropriate filter
my_filter = [[-1,-1,-1];...
              [-1,12,-1];...
              [-1,-1,-1]];
```

در کد تصویرها را خواندم و با استفاده از تابع `conv2` که در مسیله‌ی قبل از آن استفاده کردم فیلتر را بر تصویر اعمال کردم. در زیر خروجی‌های کد را مشاهده می‌کنید:

تصویر فیلتر شده‌ی ورودی:



تصویر فیلتر شده حاصل از فیلتر طراحی شده (p4g-filtered-image.png)

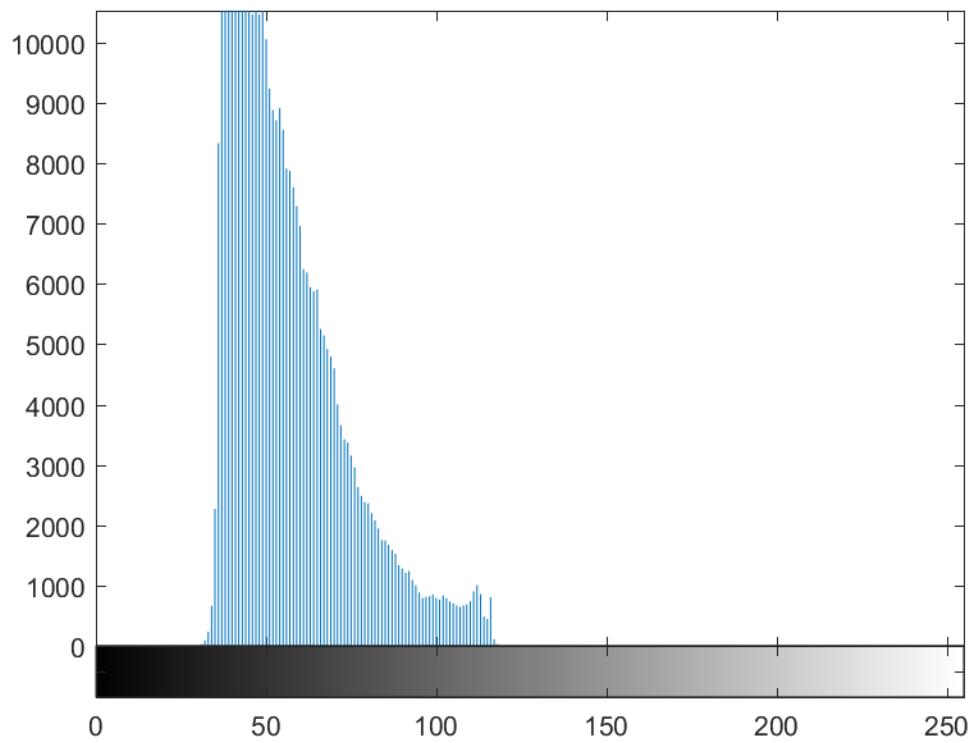


با اینکه دو تصویر بسیار شبیه هم اند و با چشم تفاوتی در آنها دیده نمی‌شود ولی مقدار `isequal` برابر نشد که می‌تواند دلیل سیاست مرزی باشد.

تمرین ۵

قسمت a

کد این قسمت در پوشه‌ی p5 و در فایل p5a_run.m قرار دارد. در این کد ابتدا فایل تصویر را خوانده‌ام و سپس با استفاده از دستور `imhist`، هیستوگرام تصویر را رسم می‌کنم. در زیر خروجی تصویر را مشاهده می‌کنید(p5a.png):

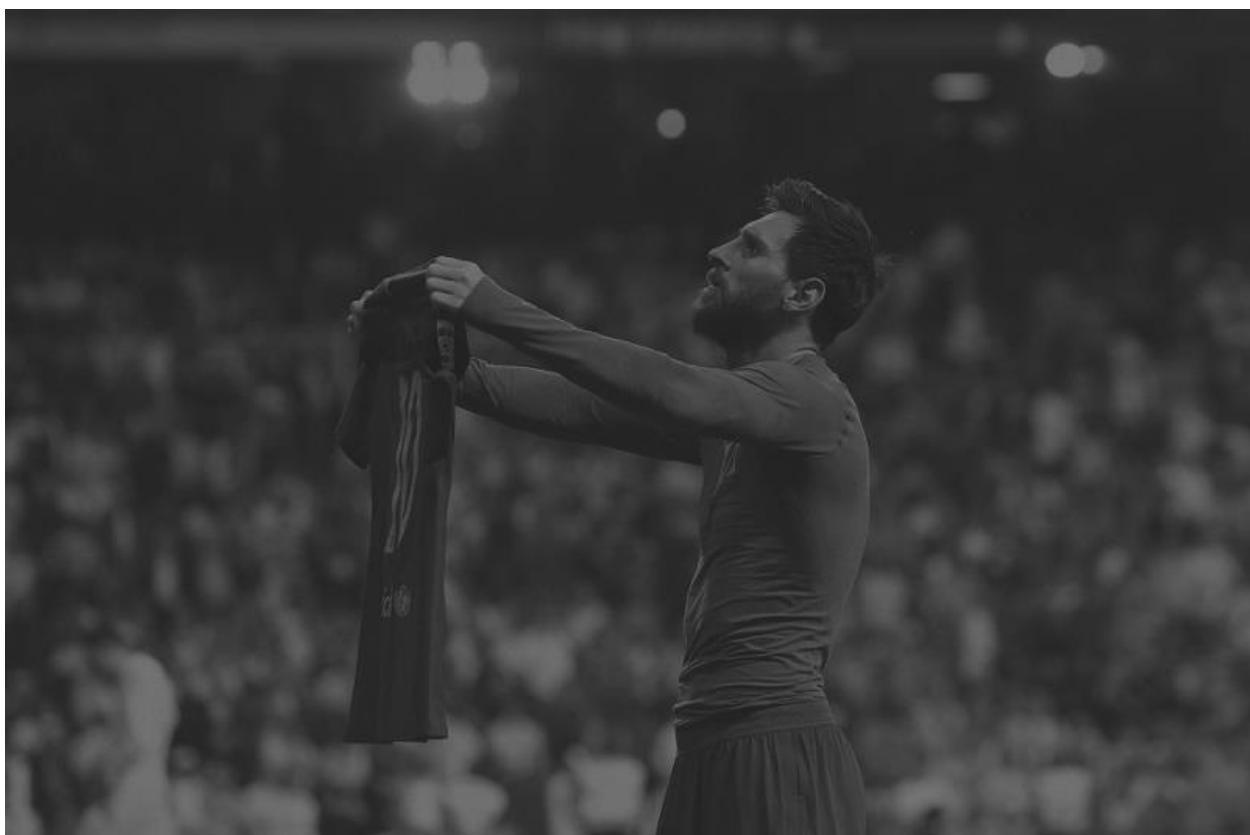


شدت روشنایی پیکسل‌ها در بازه‌ی ۲۵ تا ۱۳۰ است. بیشترین شدت روشنایی تکرار شده در تصویر مربوط به شدت روشنایی ۵۱ و همسایگانش است. شدت روشنایی استفاده شده در ناحیه‌ی تیره فشرده شده است. از شدت روشنایی بین ۰ تا ۲۵ و از ۱۳۰ تا ۲۵۵ استفاده نشده است به همین دلیل `contrast` تصویر پایین است.

قسمت b

کدهای این بخش در پوشه‌ی p5 و در فایل p5b_run قرار دارد. در این بخش، سوال خواسته است که با استفاده از تابع متلب histogram equalization را نمایش دهیم و خروجی و هیستوگرام آن را بررسی کنیم. در کد ابتدا تصویر ورودی را می‌خوانم و سپس آن را رسم می‌کنم و هیستوگرام آن را رسم می‌کنم. سپس با استفاده از تابع histeq متلب، histogram equalization را انجام می‌دهم و تصویر حاصل و هستوگرامش را رسم می‌کنم. در زیر خروجی‌های کد را مشاهده می‌کنید(خروجی‌های اصلی با نام p5b-2.png و 1.png ذخیره شده است):

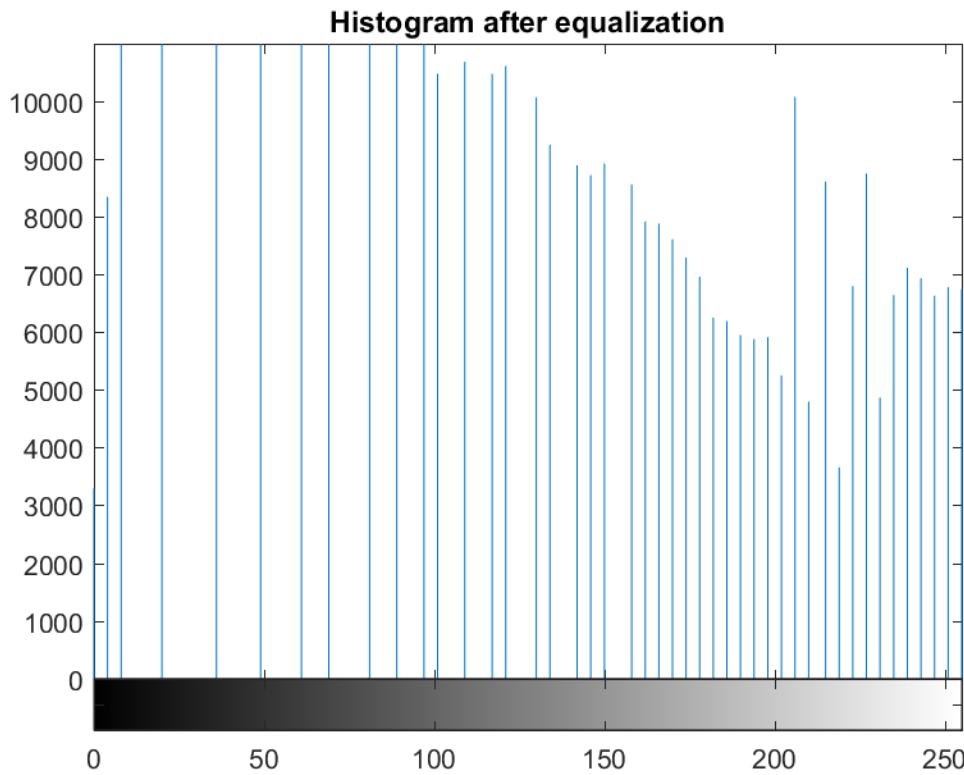
تصویر ورودی:



تصویر بعد از انجام :histogram equalization



هیستوگرام تصویر بعد از انجام :histogram equalization



همین طور که در هیستوگرام بالا دیده می شود شدت روشنایی در بازه‌ی ۰ تا ۲۵۵ پخش شده است و به همین دلیل تصویر contrast بالایی دارد. اگر مقادیر تصویر پیوسته بود هیستوگرام تصویر یک توزیع یکنواخت می شد ولی از آنجایی که مقادرهای پیکسل‌ها گسسته هستند هیستوگرام تا حد ممکن به توزیع یکنواخت نزدیک است. همین طور که در تصویری که histogram equalize آن مشاهده می شود، جزئیات قابل تشخیص تر است و تصویر بسیار بهتر نمایش داده می شود.

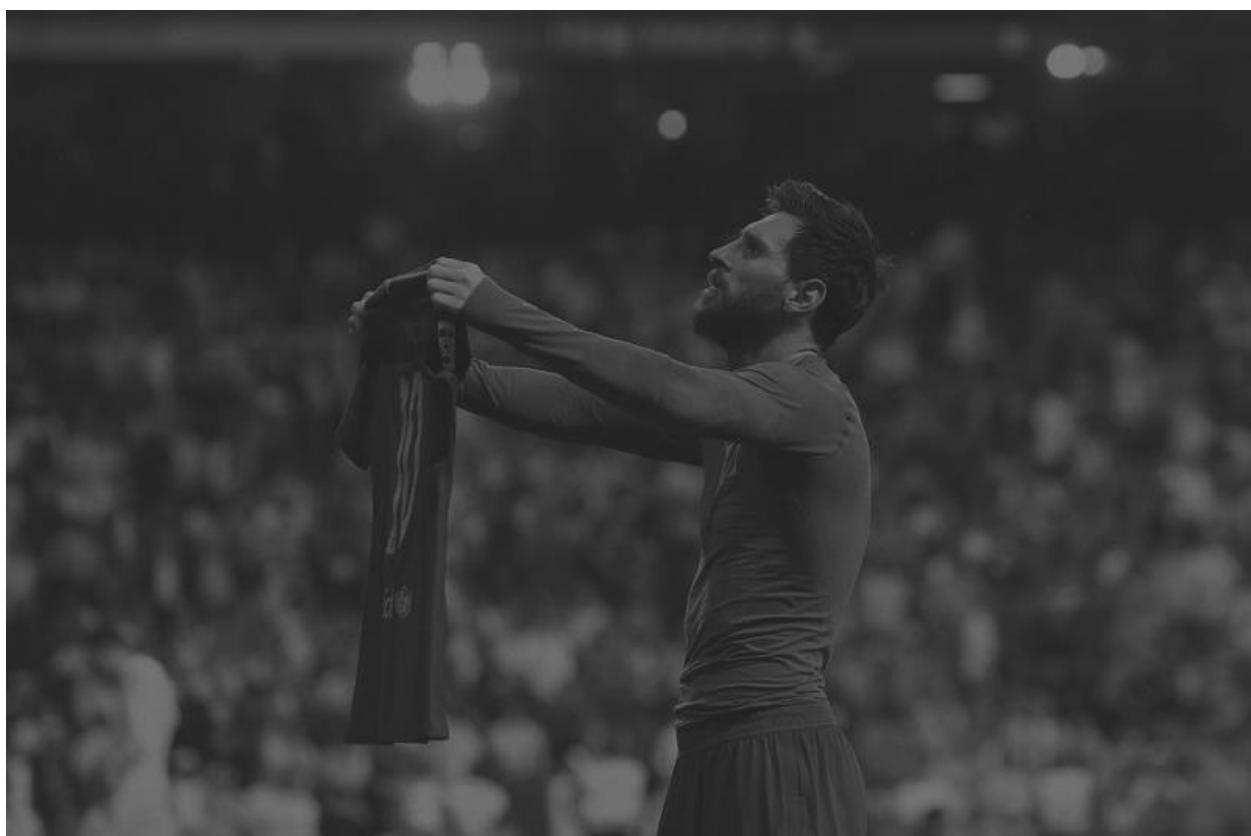
قسمت ۳

کدهای این بخش در p5c_run.m و histeq_mine.m قرار دارد. ابتدا تابع histeq_mine که یک تصویر را می‌گیرد و histogram equalization را روی آن انجام می‌دهد. برای این کار ابتدا یک آرایه به طول ۲۵۶ از صفرها ایجاد می‌کند تا تعداد تکرار هر شدت روشنایی را بشماریم. سپس تک تک پیکسل‌های تصویر را بررسی می‌کنیم و تعداد رخداد شدت روشنایی آن پیکسل را به علاوه‌ی یک می‌کنیم. بعد از این کار تعداد رخداد هر شدت روشنایی از ۰ تا ۲۵۵ را داریم. سپس آن‌ها را تقسیم بر تعداد کل پیکسل‌ها می‌کنیم تا احتمال رخداد

داد هر شدت روشنایی به دست بباید. بعد از آن فراوانی تجمعی احتمالات به دست آمده در مرحله قبل را حساب می‌کنیم و در مرحله‌ی بعد آن فراوانی‌های تجمعی را ضرب در ۲۵۵ می‌کنیم و آن‌ها را رند می‌کنیم. اکنون شدت روشنایی معادل هر شدت روشنایی در تصویر اصلی را داریم. در قسمت نهایی شدت روشنایی جدید هر پیکسل را با شدت روشنایی قبلی جایگزین می‌کنیم.

در کد p5c_run تصویر را خوانده‌ام و آن را نمایش می‌دهم و هستوگرام آن را رسم می‌کنم. بعد از آن تابع **histogram equalization** که نوشته‌ام را فرا می‌خوانم و **histogram equalization** را بر روی تصویر اعمال می‌کنم. سپس تصویر و هستوگرامش را نمایش می‌دهم. در زیر خروجی‌ها را مشاهده می‌کنید(خروجی‌های p5c-2.png و p5c-1.png در پوشه‌ی p5 موجود هستند):

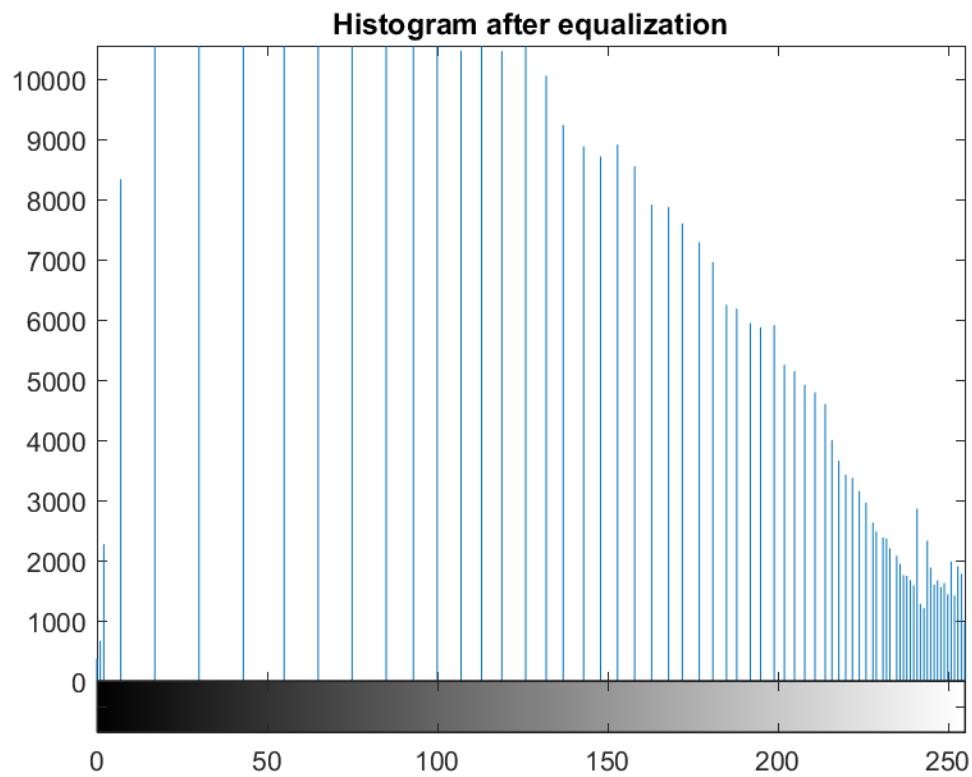
تصویر اصلی:



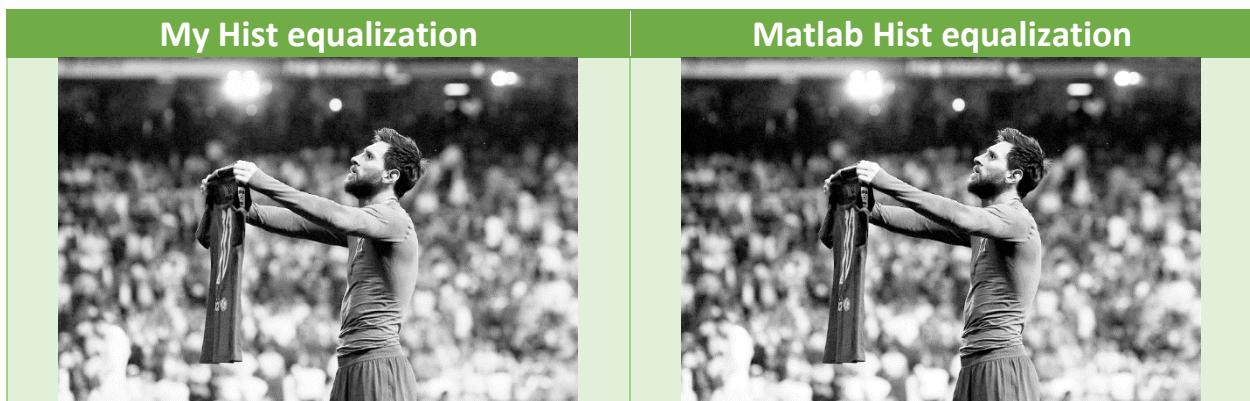
تصویر بعد از اعمال **:histogram equalization**



: histogram equalization هستوگرام تصویر حاصل از اعمال



وقتی دو تصویر حاصل از تابع **histogram equalization** خود و مطلب را کنار هم قرار می‌دهم سخت است تفاوت آن‌ها را با چشم متوجه شد ولی همانطور که در هستوگرام این قسم و قسمت قبل مشاهده می‌کنیم دو هیستوگرام متفاوت هستند چون مطلب از الگوریتم متفاوتی استفاده می‌کند.



قسمت d

کد این قسمت در پوشه‌ی p5 و در فایل p5d_run.m قرار دارد. در این قسمت histogram equalization را بر روی تصویر faces.jpg اعمال می‌کنیم. در زیر خروجی این عمل را مشاهده می‌کنید:

تصویر ورودی:



تصویر خروجی(p5d-1.png)



تصویر اول، تصویری است با contrast پایین که بیشتر پیکسل‌ها شدت روشنایی با مقدار کم دارند و شدت روشنایی بیشتر پیکسل‌ها در قسمت تاریک قرار دارد. همین طور که در تصویر ورودی مشاهده می‌شود، بیشتر صورت‌ها به کل قابل تشخیص نیستند از آنجایی که در بسیاری از کارهای مبنی بر بینایی انسان عملکرد بسیار خوبی دارد، این تصویر برای انسان هم بسیار سخت است. ولی بعد از histogram equalization جزیيات و صورت‌ها بسیار خوب قابل مشاهده اند. در تشخیص و بازناسی صورت از روش‌های مختلفی استفاده می‌کنیم مانند شبکه‌های عصبی، از آنجایی که تصویرها contrast های متفاوتی دارند و در موقعیت‌ها و شرایط نوری متفاوتی گرفته شده اند می‌توان از histogram equalization برای بالا بردن contrast تصویر استفاده کرد و این کار سبب افزایش دقت مدل و تسريع فرآیند آموزش می‌شود. فرض کنید که بیشتر تصویرهایی که برای آموزش مورد استفاده قرار گرفته اند از contrast پایینی برخوردار باشند و بیشتر پیکسل‌ها شدت روشنایی در یک رنج محدود داشته باشند، هنگامی که یک نمونه‌ی جدید برای تست به مدل می‌دهیم در صورتی که شدت روشنایی پیکسل‌ها مانند داده‌های آموزش نباشد مدل خطایش افزایش می‌یابد زیر داده‌های آموزش و تست از یک

توزیع یکسان نیستند. همچنین وقتی یک تصویر جدید به مدل می‌دهیم تا آن را `label` بزند با توجه به شرایط نوری که تصویر در آن گرفته شده است، ممکن است تصویر با نمونه‌های آموزش از نظر شدت روشنایی‌های به کار رفته متفاوت باشد که موجب `label`‌های اشتباه می‌شود که با `histogram equalization` از این مشکل دوری می‌کنیم.

تشخیص و بازشناسی چهره در بسیار از موارد برای دوربین‌های ناظارتی و ... مورد استفاده قرار می‌گیرد که در ساعت‌های مختلف روز فعال اند و شرایط نوری محیط قابل کنترل نیست `histogram equalization` سبب بهبود تشخیص و بازشناسی چهره توسط انسان و مدل‌های الگوریتمی می‌شود.

قسمت e

کد این قسمت در پوشه‌ی p5 و در فایل `p5e_run.m` قرار دارد. در ابتدا تصویر اثر انگشت را خوانده‌ام سپس آن را نمایش می‌دهم و بعد از آن `histogram equalization` را بر تصویر اعمال می‌کنم و خروجی را ذخیره می‌کنم و نمایش می‌دهم. در زیر خروجی حاصل را مشاهده می‌کنید:

تصویر ورودی:



تصویر خروجی(p5e.png)



همین طور که مشاهده می‌شود تصویر ورودی contrast پایینی دارد و بیشتر پیکسل‌ها شدت روشنایی با مقدار بالایی دارند و از بیشتر شدت روشنایی‌ها استفاده نشده است و بعضی از قسمت‌های اثر انگشت از پس زمینه قابل تمایز نیستند. تصویر خروجی contrast بالایی دارد از شدت روشنایی‌های ۰ تا ۲۵۵ به خوبی استفاده شده است، اثر انگشت به خوبی از پس زمینه قابل تشخیص است. اثر انگشت‌ها تحت شرایط نوری مختلفی گرفته می‌شوند و ممکن است contrast بالا یا پایینی داشته باشند که در صورتی که بخواهیم مدلی برای detection آن‌ها بسازیم، بدون شک بر عملکرد و دقت سیستم تاثیر می‌گذارد زیرا تفاوت در نورپردازی ربطی به تفاوت اثر انگشت‌ها و چیزی که آن‌ها را متمایز می‌کند ندارد و بخشی از زمان و قدرت مدل صرف تفاوت‌ها در شدت روشنایی می‌شود که سبب کم شدن دقت مدل می‌شود. در هنگام دادن یک نمونه به مدل برای label زنی ممکن است contrast تصویر پایین باشد به همین دلیل توزیع نمونه‌هایی که به مدل می‌دهیم با توزیع نمونه‌های آموزش متفاوت می‌شود و عملکرد مناسبی را نباید انتظار داشته باشیم.

اثر انگشت‌ها در صحنه‌ی جرم بعد از ریختن پودری بر روی اثر انگشت برای تمایز آن با سطحی که اثر انگشت بر روی آن قرار دارد با استفاده از یک دوربین با کیفیت بالا تصویر برداری می‌شوند و یا با استفاده از نوارهایی چسب مانند اثر انگشت برداشته می‌شود و بعداً اسکن می‌شود که histogram equalization به بهبود تصویر کمک می‌کند و باعث می‌شود فاکتور تمایز شدید در شدت روشنایی حاصل از شرایط نوری که تفاوت اصلی در اثر انگشت‌ها را باعث نمی‌شود کمتر شود.

قسمت f

کد این قسمت از سوال در پوشه‌ی p5 و در فایل p5_run.m قرار دارد. در ابتدا تصویر را می‌خوانم و نمایش می‌دهم و سپس با استفاده از تابع **histogram equalization** که نوشته‌ام نرمال سازی هیستوگرام تصویر را انجام می‌دهم. در انتخا خروجی را نمایش می‌دهم و آن را با نام p5f.png ذخیره می‌کنم. در زیر خروجی کد را مشاهده می‌کنید:

تصویر ورودی:



تصویر خروجی:



محیطی که تصویر ماشین‌ها جهت خواندن پلاکشان گرفته می‌شود یک محیط کنترل شده نیست و عامل‌های مختلفی همچون وجود ابر، آفتابی بودن هوا، روز، شب، آلودگی هوا، سایه و ... وجود دارد که کار تشخیص و خواندن پلاک را برای عامل انسانی و کامپیوتری سخت می‌کند. همین طور که در دو تصویر دیده می‌شود، تصویر اول تصویری است با **contrast** بسیار پایین و که از شدت روش‌نایی‌هایی با رنج محدود استفاده شده است. ولی بعد از اعمال **histogram equalization** تصویر بسیار بهبود یافته است و پلاک ماشین و سایر جزئیات به خوبی قابل دیدن هستند. اگر از روش‌های الگوریتمی و هوش مصنوعی برای تشخیص و خواندن پلاک‌ها استفاده کنیم، **contrast** پایین تصویر به گونه‌ای مانند نویز عمل می‌کند و دقت مدل را کم می‌کند چون چیزی باست که باعث تفاوت در تصویرهای مختلف شده است. به همین دلیل‌ها استفاده از این روش باعث می‌شود تاثیر بسیاری از شرایط محیطی را کم کنیم.

قسمت g

کد این قسمت از سوال در پوشه‌ی p5 و در فایل p5g.m قرار دارد. در ابتدا تصویر را می‌خوانم و نمایش می‌دهم و سپس با استفاده ازتابع **histogram equalization** که نوشته‌ام نرم‌ال سازی هیستوگرام تصویر را انجام می‌دهم. در انتخا خروجی را نمایش می‌دهم و آن را با نام p5g.png ذخیره می‌کنم. در زیر خروجی کد را مشاهده می‌کنید:

تصویر ورودی:



تصویر خروجی:

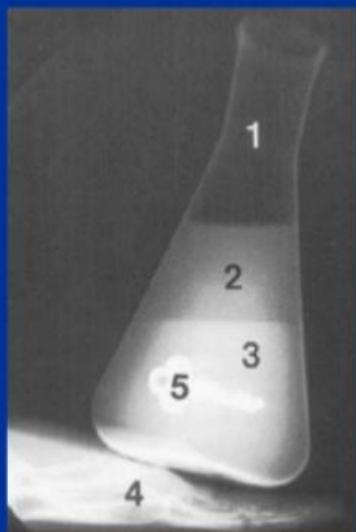


بیشتر بدن از قسمت‌هایی مانند استخوان، غضروف، چربی‌های بالا و بافت‌هایی با حجم مایعات بالا (کلیه و ...) تشکیل شده است که اشعه‌ی ایکس از آن‌ها به مقدار زیادی عبور نمی‌کند به همین دلیل بیشتر تصویرهای اسکن شده از contrast پایینی دارند و از بازه‌ی روشنی از شدت روشنایی استفاده می‌کنند. به همین دلیل بیشتر تصویرهای اشعه‌ی ایکس متمایل به سفید هستند. این موضوع تشخیص مشکلات و تمایز قابل شدن بین بخش‌های مختلف تصویر را سخت می‌کند با استفاده از histogram equalization و استفاده از بازه‌ی کامل شدت روشنایی‌ها، جزئیات بهتر نمایش داده می‌شود و پزشکان و افراد رادیولوژی راحت‌تر می‌توانند تصویر را بررسی کنند.

1. Air - Appears the blackest.
2. Fat - Lighter shade of grey than air.
3. Soft tissue or Fluid – Less black than Fat.
4. Calcium - Usually contained within bones.
5. Metal – Whitest.

Clip slide

(Objects of metal density are not normally present in the body.)



15

تمرین ۶

کدها و خروجی‌های این تمرین در پوشه‌ی p6 قرار دارد. در این قسمت از کد histogram equalization سوال قبل استفاده کرده‌ام به همین دلیل آن را دوباره توضیح نمی‌دهم.

قسمت a

کد این قسمت در p6a.m قرار دارد. ابتدا برای هر کدام از دو تصویر، تصویر را خوانده‌ام و سپس آن را نمایش داده‌ام و هیستوگرام آن را نمایش دادم‌ام، سپس histogram equalization را که خود نوشته‌ام بر آن‌ها اعمال می‌کنم و در انتهای توصیر خورجی و هستوگرام قبل و بعد از اعمال نرم‌ال سازی هستوگرام را نمایش و می‌دهم و سپس ذخیره می‌کنم.

در زیر خروجی‌های کد را مشاهده می‌کنید:

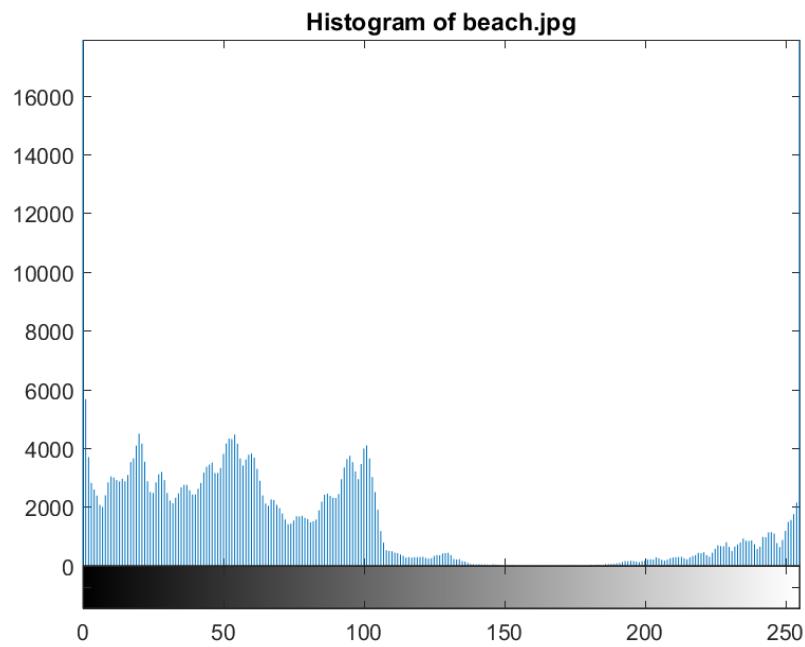
:beach.jpg تصویر ورودی



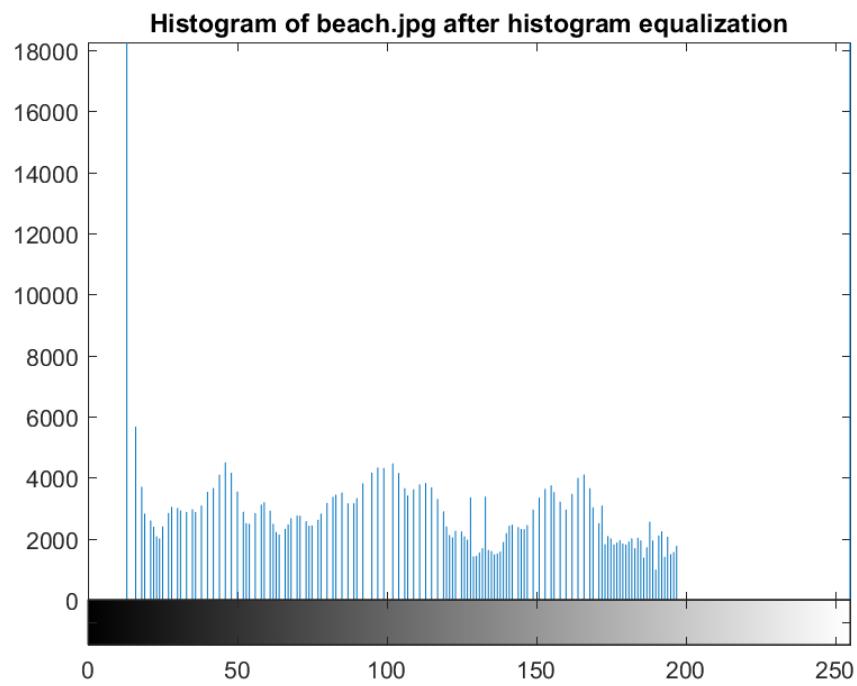
:p6a-1.png) Histogram Equalization beach.jpg تصویر بعد از



هیستوگرام تصویر ورودی (p6a-1-hist.png)



: (p6a-1-hist-hist_eq.png) histogram equalization هستوگرام تصویر بعد از اعمال



توجه: بر روی ۲۵۵ خطی آبی با ارتفاع زیاد موجود است که به خوبی دیده نمی‌شود.

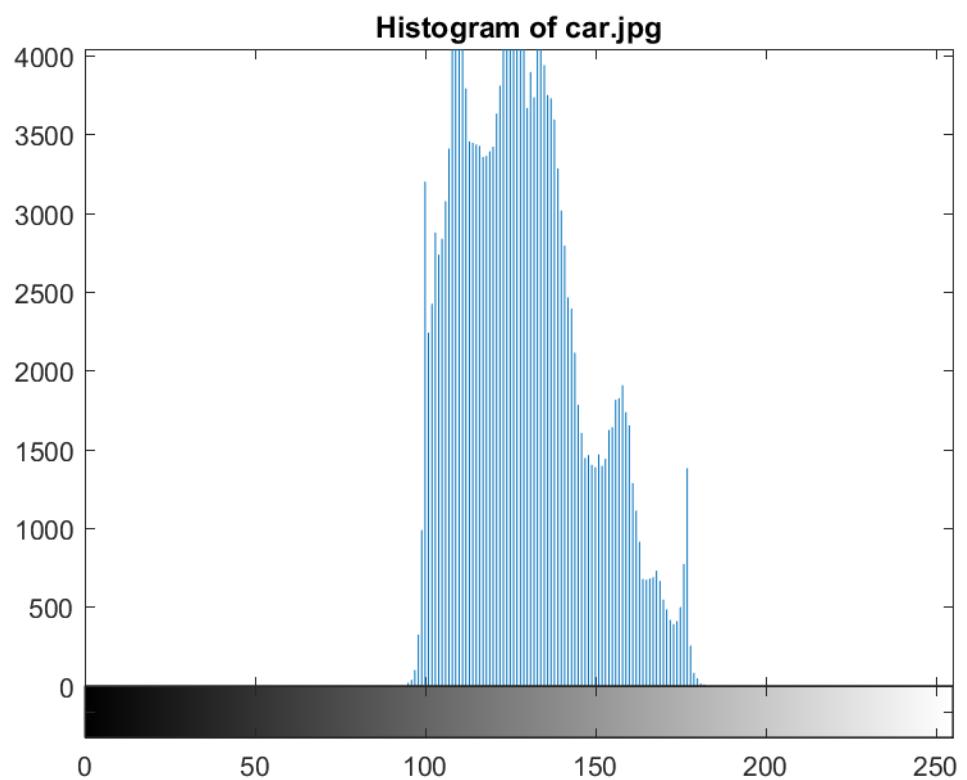
تصویر ورودی car.jpg



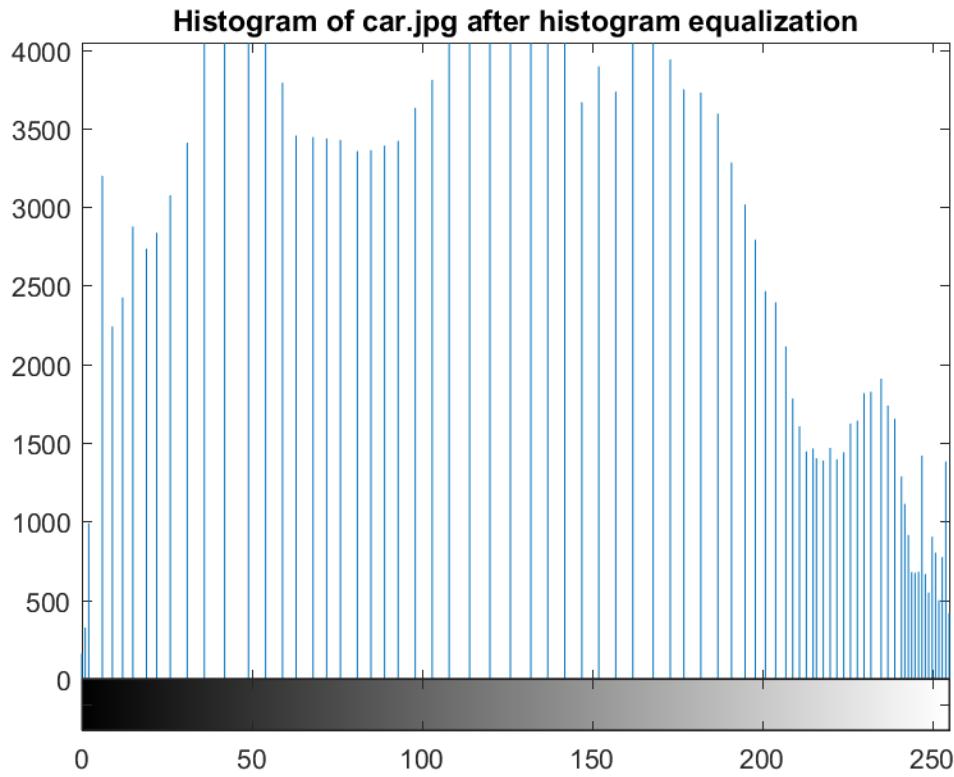
تصویر car.jpg بعد از :(p6a-2.png) Histogram Equalization



: (p6a-2-hist.png) هیستوگرام تصویر ورودی



هستوگرام تصویر بعد از اعمال `:p6a-2-hist-hist_eq.png` histogram equalization



هیستوگرام تصویر ماشین نشان می‌دهد شدت روشنایی‌ها تقریباً در بین بازه‌ی ۹۰ تا ۱۷۰ است و از سایر قسمت‌ها استفاده نشده است که بعد از اعمال **histogram equalization** تصویر ماشین بسیار خوب شده است جزیيات به خوبی قابل تشخیص اند. هیستوگرام تصویر نرمال شده نشان می‌دهد شدت روشنایی به خوبی در بازه‌ی ۰ تا ۲۵۵ پخش شده است و هیستوگرام تا جای ممکن به توزیع نرمال نزدیک است.

ولی در هیستوگرام تصویر ساحل همانطور که مشاهده می‌شود بسیاری از پیکسل‌ها شدت روشنایی بین ۰ تا تقریباً ۱۲۰ دارند و بعد از آن به شدت تعداد پیکسل‌هایی که از شدت روشنایی ۱۲۰ به بالا استفاده می‌کنند کم می‌شود و در انتهای دوباره تعداد پیکسل‌هایی که از شدت روشنایی ۲۰۰ و بالاتر استفاده می‌کنند زیاد می‌شود همین باعث می‌شود که عمل **histogram equalization** رو کل تصویر نتیجه‌ی مطلوبی ندهد. همین طور که در تصویر نرمال شده مشاهده می‌شود بخش زیادی از آسمان شدت روشنایی ۲۵۵ پیدا کرده است و داخل خانه مشخص نیست. اگر به هیستوگرام تصویر نرمال شده نگاه بکنیم مشاهده می‌کنیم بیشتر پیکسل‌ها شدت روشنایی‌ها بین ۰ تا حدوداً ۱۹۰ دارند و بعد از آن شدت روشنایی‌ها بالاتر تقریباً استفاده نشده است تا شدت روشنایی ۲۵۴ و بعد رشد چشمگیری رخ می‌دهد و می‌بینیم که بسیاری از پیکسل‌ها شدت روشنایی ۲۵۵ دارند.

کدهای این قسمت از سوال در پوشه‌ی pb6 و در فایل‌های histeq_mine.m, overlapping_hist_eq.m, pb6_run.m و non_overlapping_block_hist_eq.m قرار دارند.

کد histogram را در سوال قبل توضیح دادم که یک تصویر می‌گیرد و سپس کد equalization را بر آن اعمال می‌کند و بعد خروجی را باز می‌گرداند.

در کد non_overlapping_block_hist_eq.m تابع زیر قرار دارد

```
function adaptive_he_img = non_overlapping_block_hist_eq(img, m)
    % This function divide image to m*m non-overlapping blocks
    % then it applies histogram equalization on each one.
```

در این تابع تصویر به مربع‌هایی $m \times m$ تقسیم می‌شود و histogram equalization بر هر مربع اعمال می‌شود.

در کد overlapping_hist_eq.m تابع زیر قرار دارد:

```
function filtered_image = overlapping_hist_eq(img, m)
    % This function divide image to m*m overlapping blocks
    % then it applies histogram equalization on each
    % m*m neighbourhood of each pixel, it does padding.
```

این تابع یک تصویر را می‌گیرد و آن را به اندازه‌ی لازم pad می‌کند، سپس روی تک تک پیکسل‌ها حرکت می‌کند و برای همسایگی $m \times m$ پیکسل histogram equalization را اعمال می‌کند و بعد شدت روشنایی آن پیکسل را با شدت روشنایی جدید جایگزین می‌کند.

در کد pb6_run.m دو تصویر ساحل و ماشین را می‌خوانم و تابع overlapping_hist_eq.m را برای هر دو تصویر با سایز بلاک 200×200 فرامی‌خوانم. همین طور تابع non-overlapping histogram

را با سایز بلاک (همسایگی) ۷۱ برای هر دو تصویر فرا می خوانم و نتایج را نمایش می دهم و همین طور در فایل ذخیره می کنم.

در زیر خروجی های این کد را مشاهده می کنید:

:beach تصویر ورودی



: (p6b-beach-1.png) **non-overlapping** histogram equalization beach تصویر بعد از اعمال



: (p6b-beach-2.png) **overlapping** histogram equalization beach تصویر بعد از اعمال



نرمال سازی هستوگرام عادی(کلی تصویر):



در نرمال سازی هستوگرام عادی تصویر درون کلبه به خوبی دیده نمی شود فرد چهارمی در تصویر است که مشخص نیست، همین طوری زیر کلبه جزیيات قابل دیدن نیستند، در histogram equalization به وسیله‌ی بلاک‌های غیرهم پوشان، این مشکلات حل شده‌اند ولی تصویر به صورت بلاک‌های جدا از هم در آمده است. در روش histogram equalization به وسیله‌ی بلاک‌های هم‌پوشان مشکلات در روش نرمال سازی هیستوگرام عادی حل شده است و همچنین عملکرد بسیار بهتری نسبت به روش بلاک‌های غیر هم پوشان داشته‌ایم، زیرا تصویر بلاک بلاک نشده است، جزیيات بیشتر از نفر چهارم درون کلبه می‌بینیم، جزیيات بیشتری از زیر کلبه می‌بینیم، جزیيات بسیار بیشتر روی دیوارهای کلبه و شیشه‌ی در کلبه‌ی دوم قابل مشاهده است. در بین سه روش overlapping block histogram equalization با اختلاف روش histogram equalization بهتر از دو روش دیگر عمل کرده است.

تصویر ورودی :car



تصویر car بعد از اعمال :(p6b-car-1.png) non-overlapping histogram equalization



تصویر car بعد از اعمال :(p6b-car-2.png) overlapping histogram equalization



نرمال سازی هیستوگرام عادی(کلی) تصویر:



در روش histogram equalization عادی که بر روی کل تصویر اعمال می‌شود، نتیجه‌ی خوبی به دست آمده است، در روش histogram equalization non-overlapping blocks به طریق جزیيات بیشتری

قابل مشاهده است مانند ساختار توری مانند کنار چراغ روی سپر که در روش قبل قابل مشاهده نیست ولی تصویر بلاک بلاک شده است در روش histogram equalization به روش overlapping blocks نتیجه بسیار زیبا است، جزییات بازتاب محیط روی بدنه سمت راست تصویر، روی سپر و کاپوت بسیار خوب قابل مشاهده است، ساختار توری و شطرنجی مانند کنار چراغ روی سپر به خوبی قابل مشاهده است، سنتگفرش‌ها بسیار بهتر دیده می‌شوند همین طور جزییات بیشتر از صندلی راننده قابل دیدن است. روش overlapping blocks بسیار بهتر از سایر روش‌ها عمل کرده است.

سوال ۷

کدهای این سوال در پوشه‌ی p7 قرار دارد.

قسمت a

کدهای این قسمت در p7a.m قرار دارد. ابتدا تصویر برج آزادی را می‌خوانم و تصویر را نمایش می‌دهم و هیستوگرام آنرا رسم می‌کنم و سپس log transformation زیر را انجام می‌دهم:

$$s = c \times \log(1 + r)$$

و C را برابر با

$$c = \frac{255}{\log(1 + \text{maximum intensity})}$$

قرار می‌دهم به طوری که بزرگ‌ترین شدت روشنایی ۲۵۵ شود.

بعد از اعمال این transformation بر تصویر برج آزادی تصویر خروجی و هیستوگرامش را ذخیره می‌کنم و نمایش می‌دهم.

همین کار را هم برای تصویر تهران انجام می‌دهم. علاوه بر log transformation برای تصویر تهران، inverse log transformation نیز انجام می‌دهم.

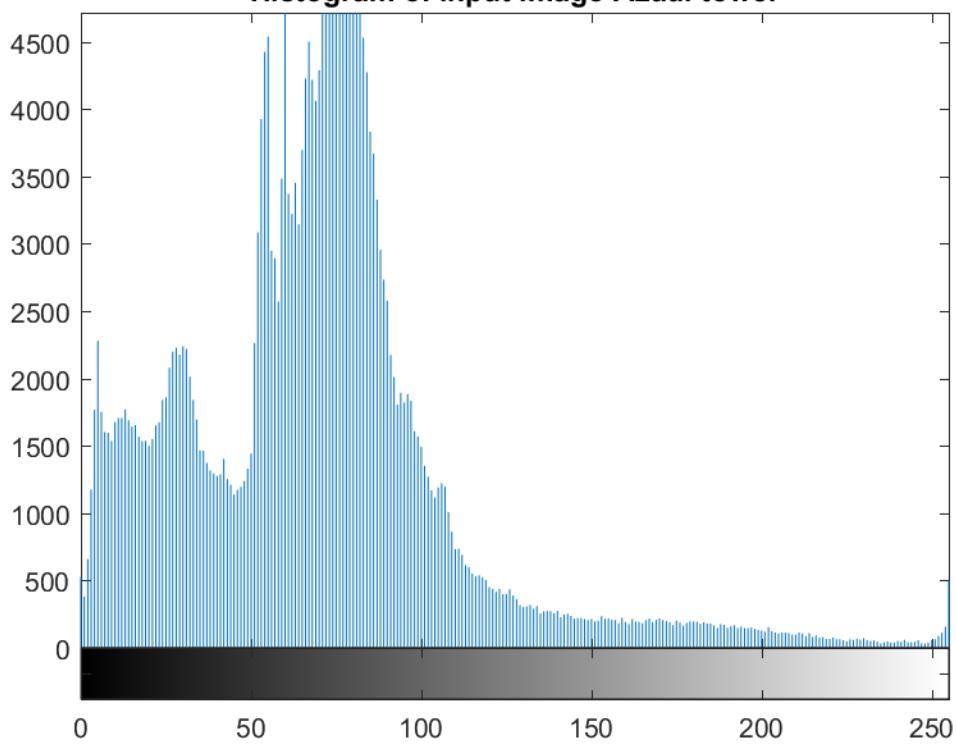
در زیر خروجی‌های کد را مشاهده می‌کنید:

تصویر ورودی برج آزادی:



هیستوگرام تصویر برج آزادی:

Histogram of input image-Azadi tower

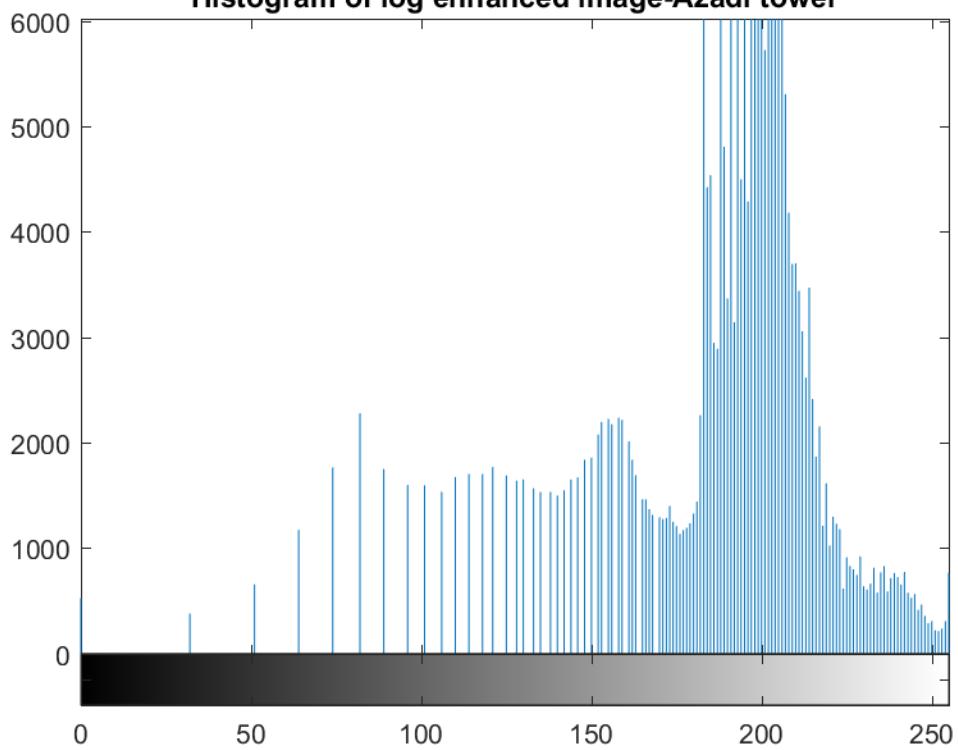


تصویر برج آزادی بعد از log transformation



هستوگرام تصویر برج آزادی بعد از log transformation

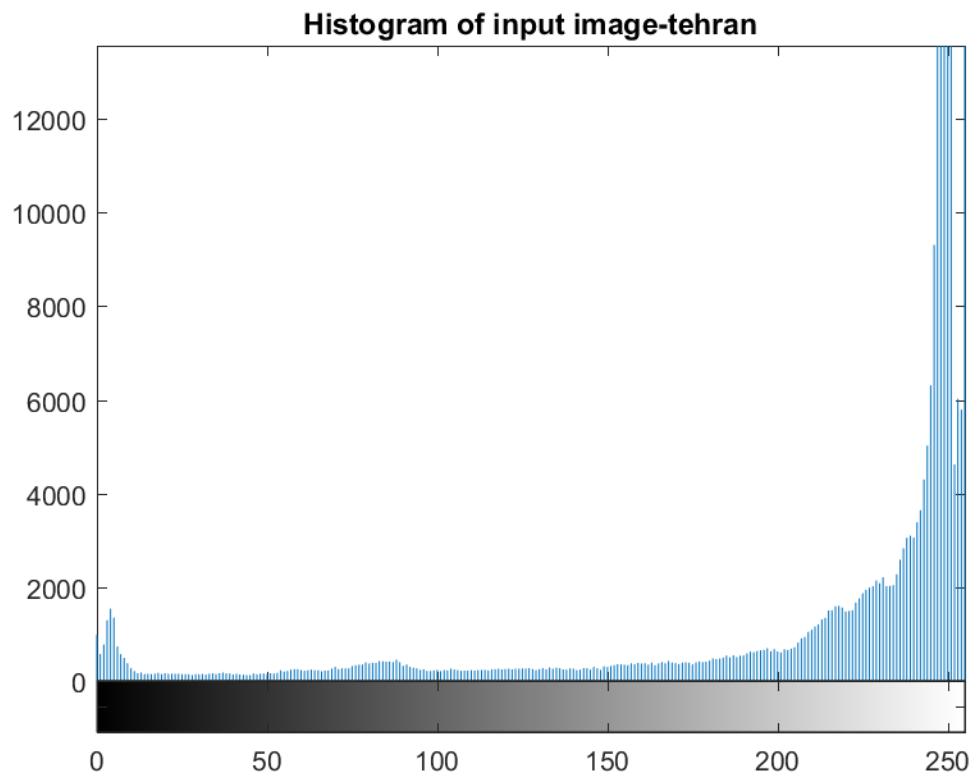
Histogram of log enhanced image-Azadi tower



تصویر ورودی تهران:



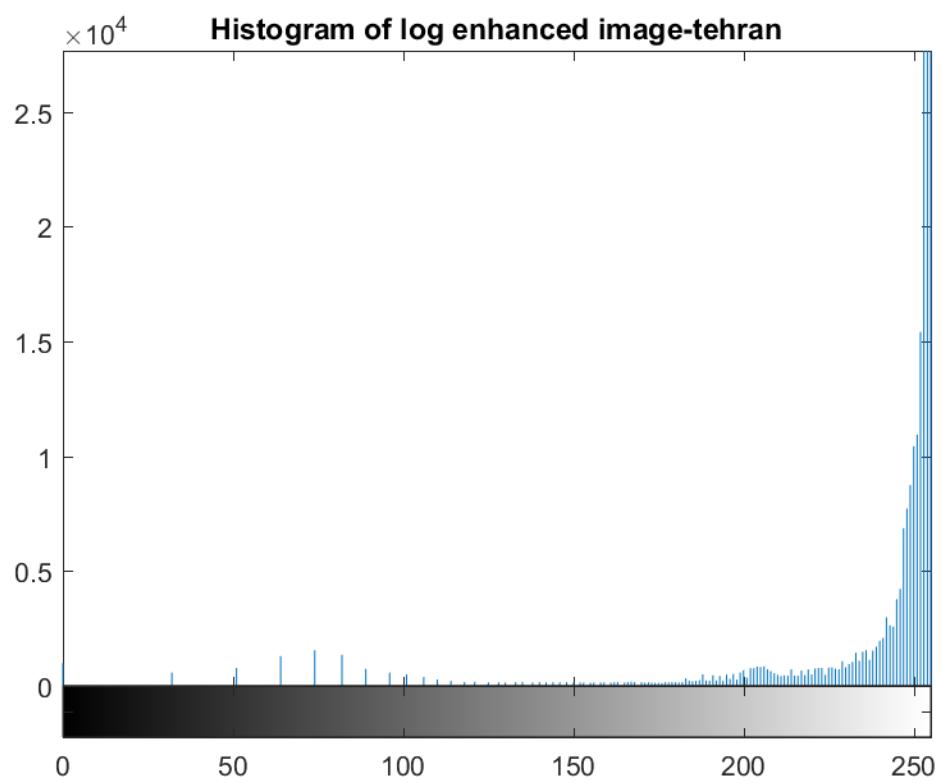
هیستوگرام تصویر تهران:



تصویر تهران بعد از log transformation



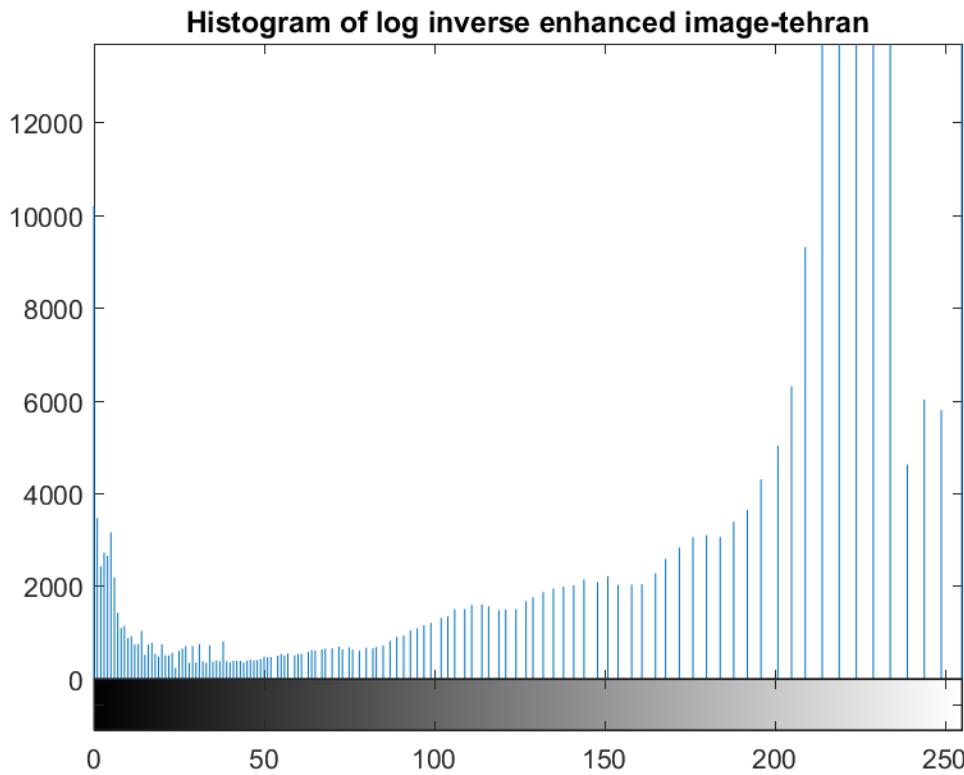
هستوگرام تصویر تهران بعد از log transformation



تصویر تهران بعد از Inverse log transformation



هیستوگرام تصویر تهران بعد از :Inverse log transformation



تصویر برج آزادی تصویری با کنترast پایین است که اگر به هیستوگرام آن نگاه کنیم متوجه می‌شویم، که بیشتر پیکسل‌ها شدت روشنایی کمتر ۱۲۰ دارند. بعد از استفاده از \log transform تصویر بسیار بهبود یافته است و جزییات تصویر به راحتی قابل مشاهده و تشخیص است و تصویر به طرز مناسبی روشن شده است، همین طور که مشاهده می‌شود در هیستوگرام تصویر جدید، میزان استفاده از شدت روشنایی‌ها بسیار بهتر در بازه‌ی ۰ تا ۲۵۵ پراکنده شده است و دیگر آن ناحیه فشرده در هیستوگرام اولیه وجود ندارد. البته این نتیجه قابل پیشبینی بود زیرا در هنگام \log transformation هر چه شدت روشنایی بزرگ‌تر باشد مقدار جدیدش بزرگ‌تر است. تصویر حاصل از \log transformation برج آزادی بهتر از تصویر اولیه است.

تصویر تهران تصویری بسیار روشن است و کنتراست مناسبی ندارد و با دقت در هیستوگرام آن می‌بینیم که بیشتر پیکسل‌ها شدت روشنایی بیشتر از ۲۱۰ دارند و بسیار کم از شدت روشنایی‌های دیگر استفاده شده است، بعد از اعمال \log transformation بر روی تصویر تهران متوجه می‌شویم که تصویر حاصل سفید تر از تصویر اولیه است و جزییات کمتری قابل تشخیص است، با توجه به هیستوگرام تصویر حاصل از \log transform می‌بینیم

که وضع استفاده از شدت روشنایی‌ها توسط پیسکل‌ها بدتر شده است و بیشتر پیکسل‌های شدت روشنایی بزرگتر مساوی ۲۳۵ پیدا کرده‌اند و میزان استفاده از شدت روشنایی‌های کمتر نسبت به حالت قبل حتی کمتر هم شده است. به همین دلیل این transformation برای تصویر تهران موجب بهبود نشده است. دلیل اینکه در تصویر تهرن بهبود نداریم آن است که بیشترین شدت روشنایی‌های استفاده شده در ناحیه تاریک نیست بلکه در ناحیه روشن است.

برای بهبود تصویر تهران از Inverse Log Transformation استفاده کردم زیر بهبودی حاصل نکرد. همین طور که مشاهده می‌شود تصویر حاصل از این کار بسیار بهبود یافته است و شدت روشنایی‌ها به گونه‌ای است که تصویر سفید مانند نیست و جزیات و خانه‌ها به خوبی قابل مشاهده است. اگر به هیستوگرام این تصویر نگاه نکنیم می‌بینیم دیگر بیشترین شدت روشنایی‌های استفاده شده در ناحیه ۲۱۰ و بزرگ‌تر از آن نیست و به طرز مناسبی هیستوگرام در بازه‌ی ۰ تا ۲۵۵ پخش شده است.

نتیجه می‌گیریم اگر بیشترین شدت روشنایی‌های استفاده شده در ناحیه تاریک باشد log transformation نتیجه‌ی خوبی می‌دهد.

قسمت b

کدهای این قسمت در p7b.m قرار دارد، در این کد برای gamma transformation از تابع زیر استفاده می‌کنم:

$$s = c \times r^\gamma$$

و مقدار c را بدین گونه حساب می‌کنم:

$$c = \frac{255}{(\max r)^\gamma}$$

در ابتدا تصویرها را می‌خوانم آن‌ها و هیستوگرامشان را نمایش می‌دهم و سپس gamma transformation را اعمال می‌کنم و تصویرهای حاصل و هیستوگرامشان را رسم می‌کنم.

از آنجایی که تصویر برج آزادی تصویری است که بیشتر پیکسل‌ها از شدت روشنایی‌های کم استفاده می‌کنند از گاما با مقدار ۴۵،۰ استفاده کرده‌ام. در تصویر تهران از آنجایی که بیشتر پیکسل‌های تصویر از شدت روشنایی‌های بالا دارند از گاما برابر با ۴ استفاده می‌کنم.

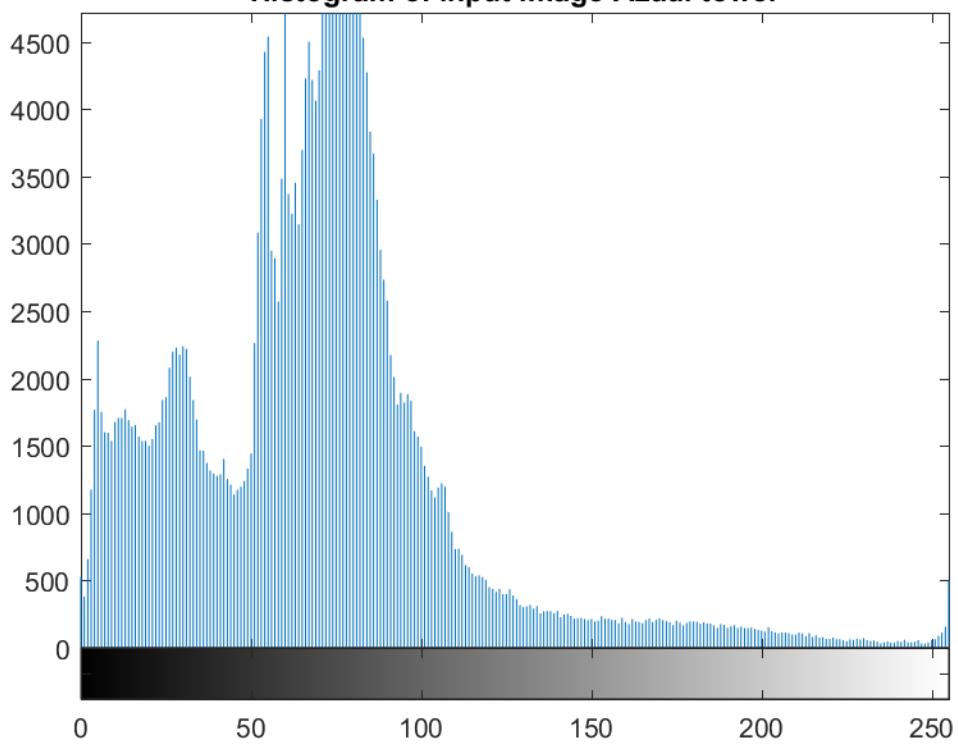
در زیر خروجی‌های کد را مشاهده می‌کنید:

تصویر ورودی برج آزادی:



هیستوگرام تصویر برج آزادی:

Histogram of input image-Azadi tower

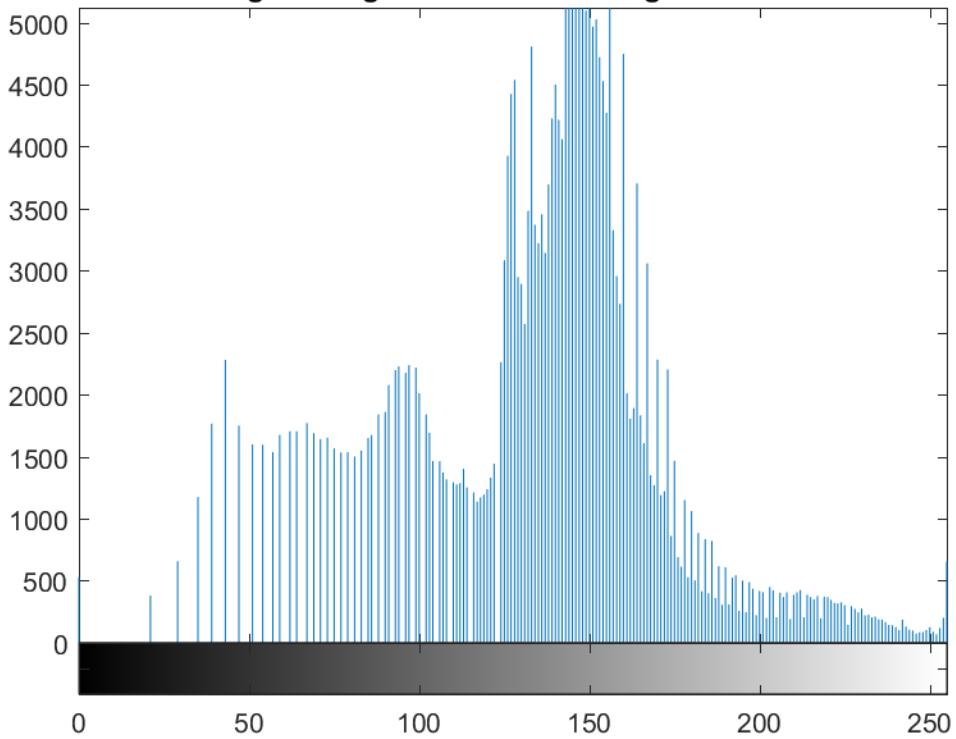


تصویر برج آزادی بعد از gamma transformation



هستوگرام تصویر برج آزادی بعد از gamma transformation

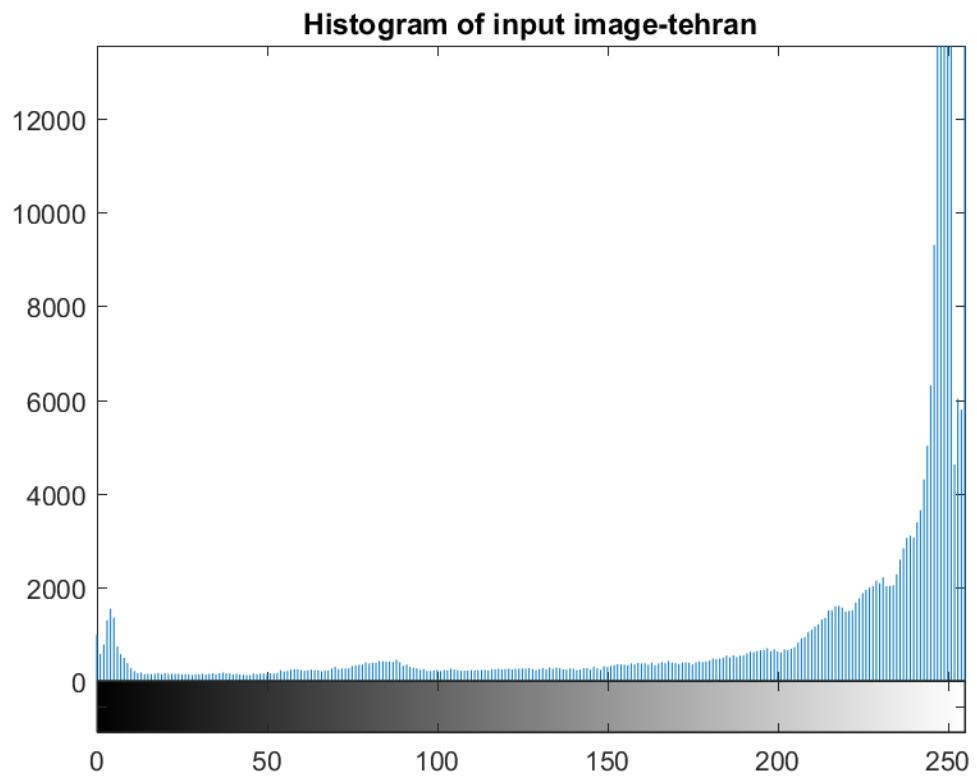
Histogram of gama enhanced image-Azadi tower



تصویر ورودی تهران:



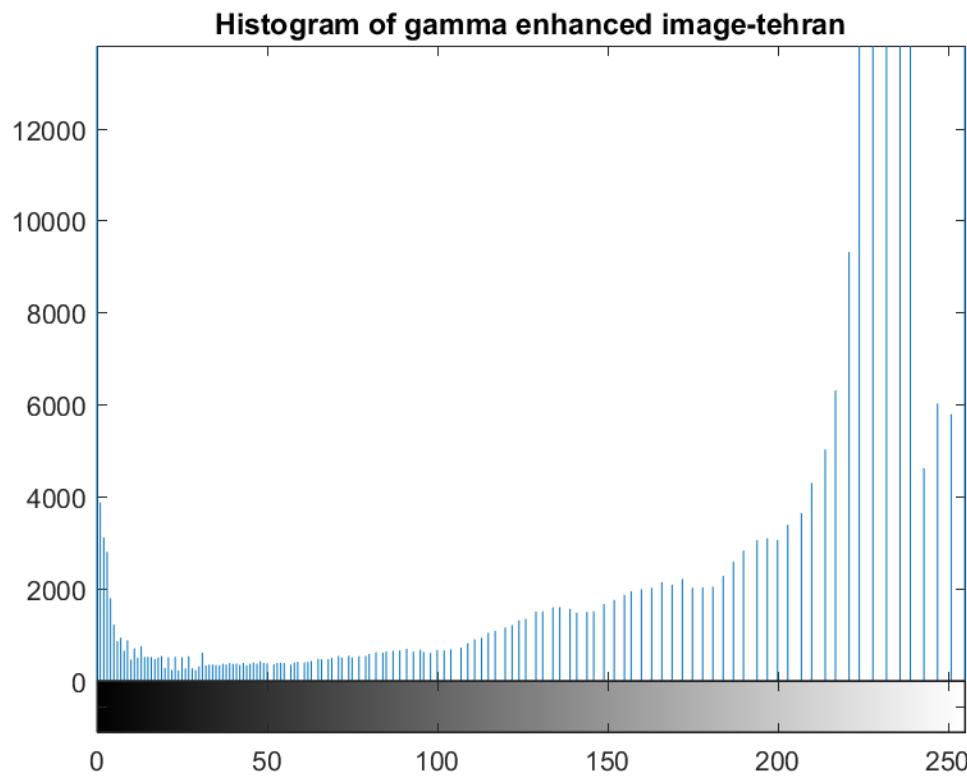
هیستوگرام تصویر تهران:



تصویر تهران بعد از gamma transformation



هستوگرام تصویر تهران بعد از log transformation



همین طور که مشاهده می شود با استفاده از gamma transformation و با گامای ۰،۴۵ تصویر برج آزادی بسیار بهبود پیدا کرده است و وقتی به هیستوگرام تصویر بعد از transformation نگاه می کنیم متوجه می شویم که در هیستوگرام نسبت به حالت قبل دیگر بیشتر پیکسل ها دارای شدت روشنایی تیره و در بازه کوچکی نیستند و به خوبی میزان استفاده از شدت روشنایی ها بین پیکسل ها پراکنده شده است.

با استفاده از gamma transformation بر روی تصویر تهران، تصویری بهتری به دست می آید و تصویر دیگر بسیار سفید نیست و اگر به هیستوگرام تصویر جدید نگاه کنیم می بینیم تعداد پیکسل های بسیار بیشتر از شدت روشنایی های کمتر از ۲۴۰ استفاده می کنند و جزئیات بیشتر از تصویر قابل دیدن است مانند خانه های دور دست و دیگر تصویر آن حالت سفیدی را ندارد.

برای تصویرهایی که بیشتر شدت روشنایی های استفاده شده در ناحیه تاریک است باید از gamma های کوچک تر از یک استفاده کنیم و برای تصویرهایی که بیشترین شدت روشنایی استفاده شده توسط آنها در ناحیه سفید است باید از gamma بزرگتر از یک استفاده کنیم.

قسمت C

کد این بخش در p7c.py قرار دارد، برای این قسمت هر درایه از ماتریس سه بعدی تصویر را از ۲۵۵ کم کرده‌ام.
در زیر تصویر خروجی را مشاهده می‌کنید:

تصویر ورودی:



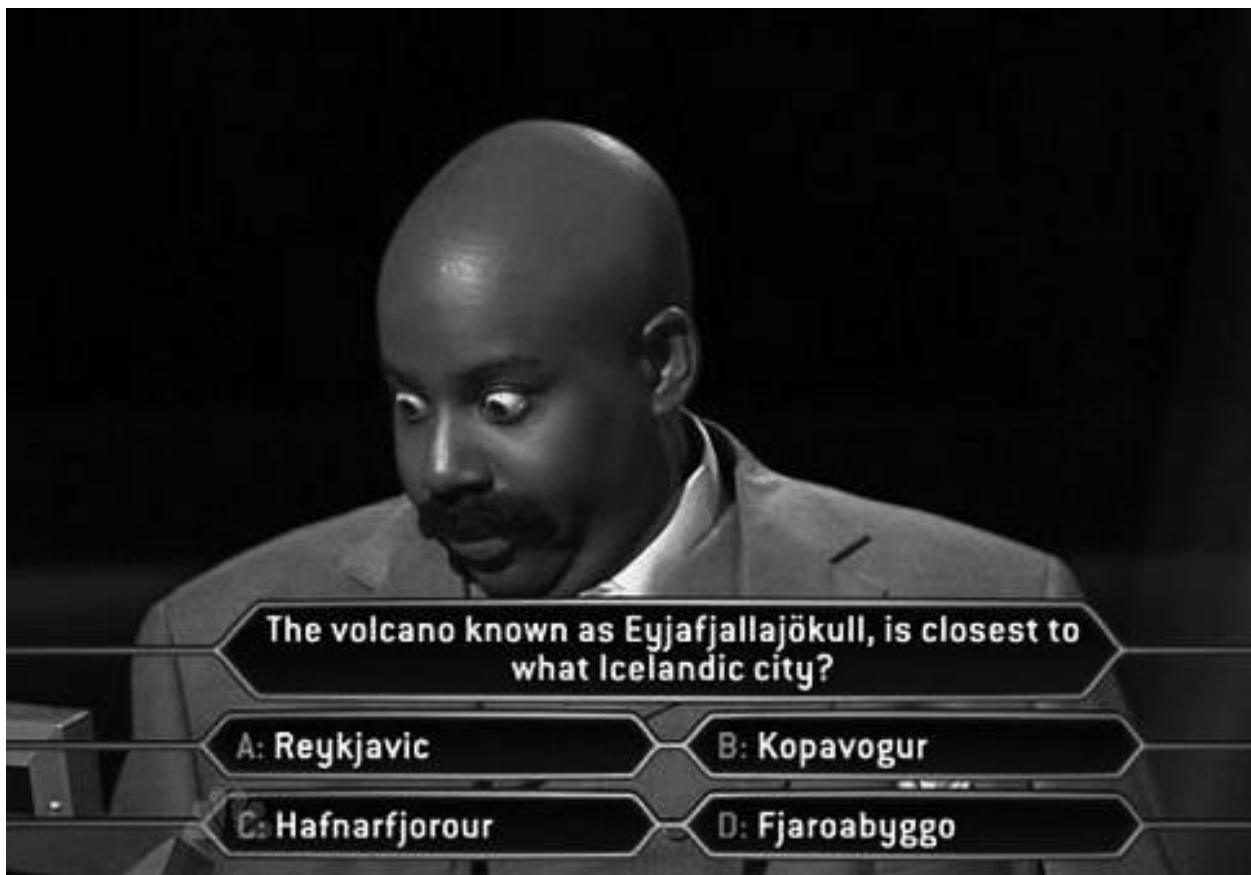
تصویر خروجی:



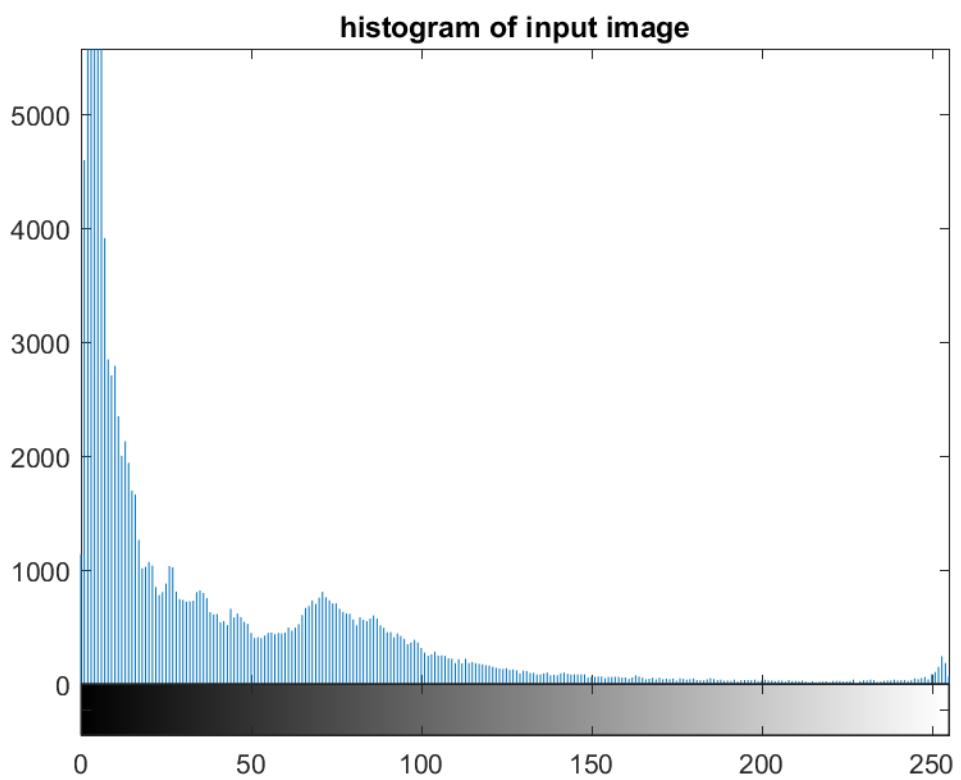
d قسمت

کدهای این قسمت در p7d.m قرار دارد. در این کد ابتدا تصویر را می‌خوانم و سپس هیستوگرام آن را رسم می‌کنم و در ادامه بر تصویر یک **gamma transformation** طبق تابع‌های قسمت b انجام می‌دهم و از مقدار گاما^۱ ۰.۲ استفاده می‌کنم در زیر خروجی تصویر را مشاهده می‌کنید.

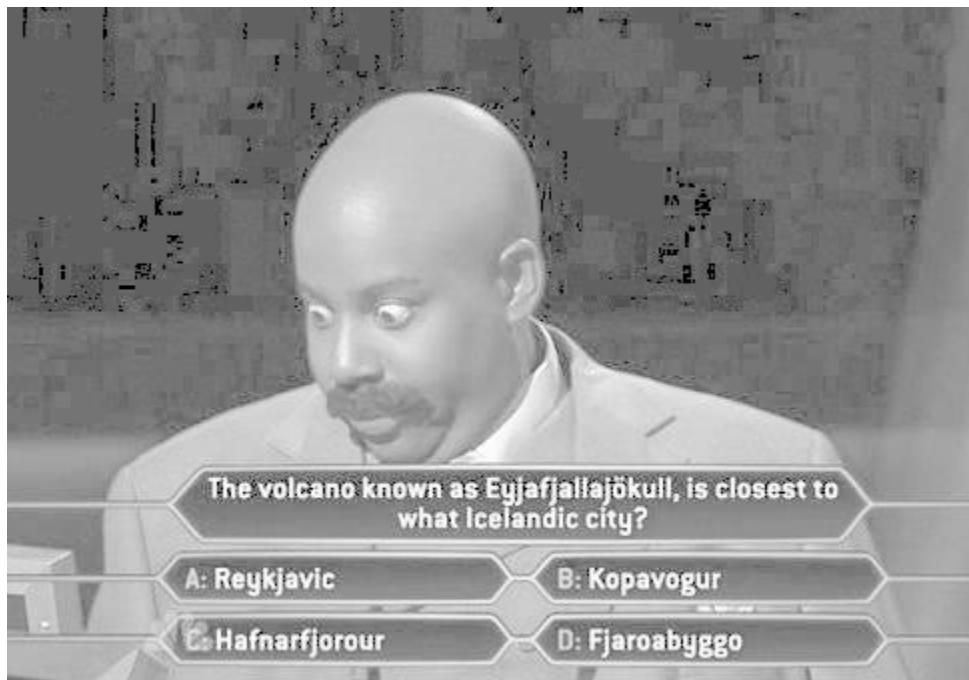
تصویر ورودی:



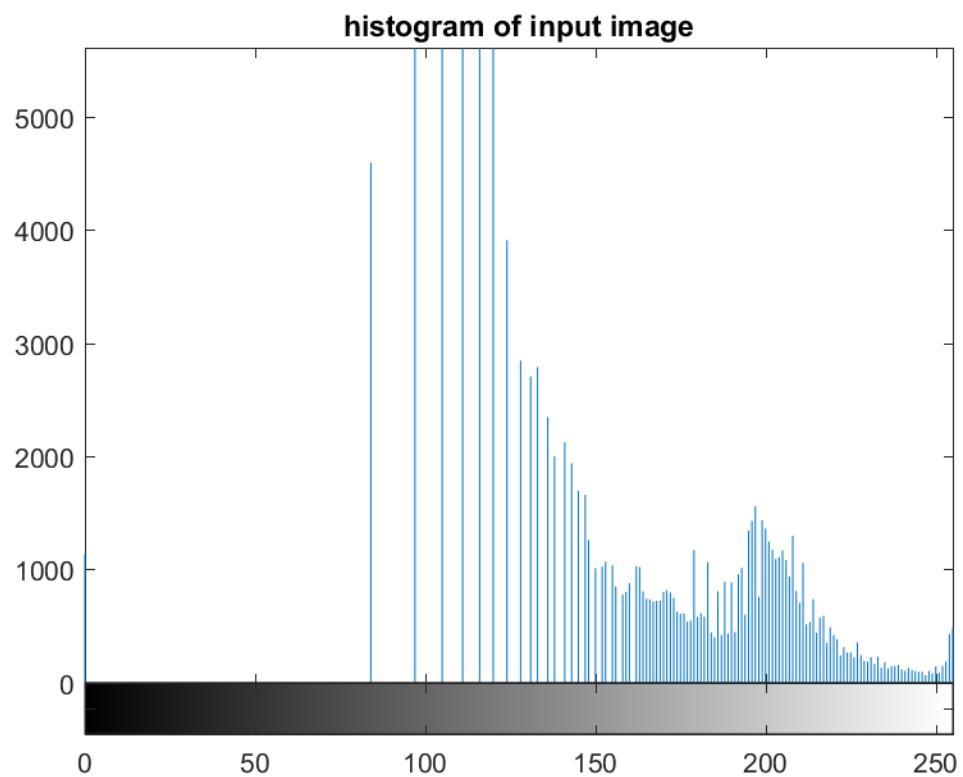
هیستوگرام تصویر ورودی:



تصویر خروجی که washed-out است:



هیستوگرام تصویر خروجی:



سوال ۸

کدهای این قسمت از سوال در پوشه‌ی p8 است.

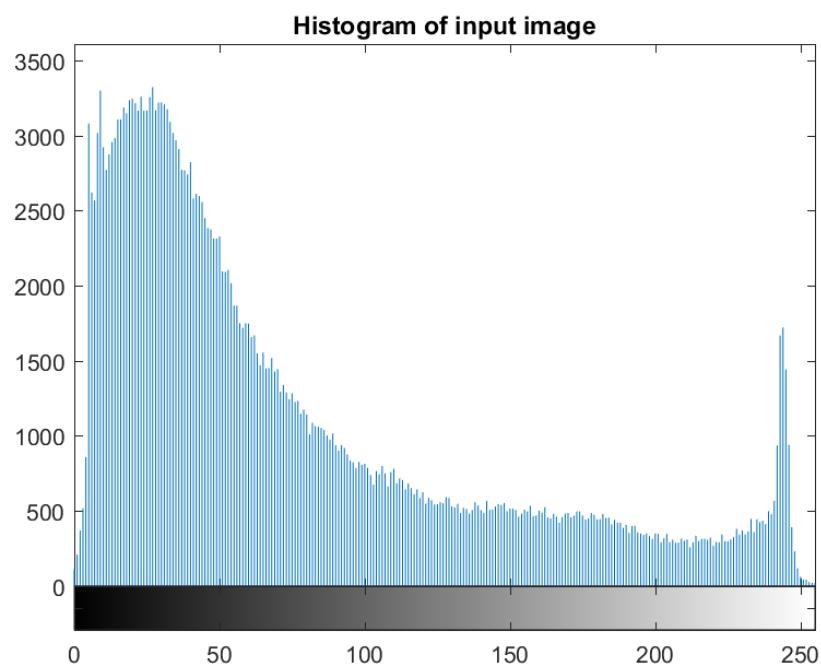
قسمت a

کد این قسمت از سوال در فایل p8a.m قرار دارد. در ابتدا تصویر را می‌خوانم و بعد از آن هیستوگرام تصویر را نشان می‌دهم، یک بار شدت روشنایی پیکسل‌های تصویر را به علاوه‌ی ۶۴ می‌کنم و تصویر حاصل و هیستوگرامش را نمایش می‌دهم. را نمایش می‌دهم و بار دیگر تصویر ورودی را منهای ۶۴ می‌کنم و تصویر حاصل و هیستوگرامش را نمایش می‌دهم.
در زیر خروجی‌های برنامه را مشاهده می‌کنید:

تصویر ورودی:



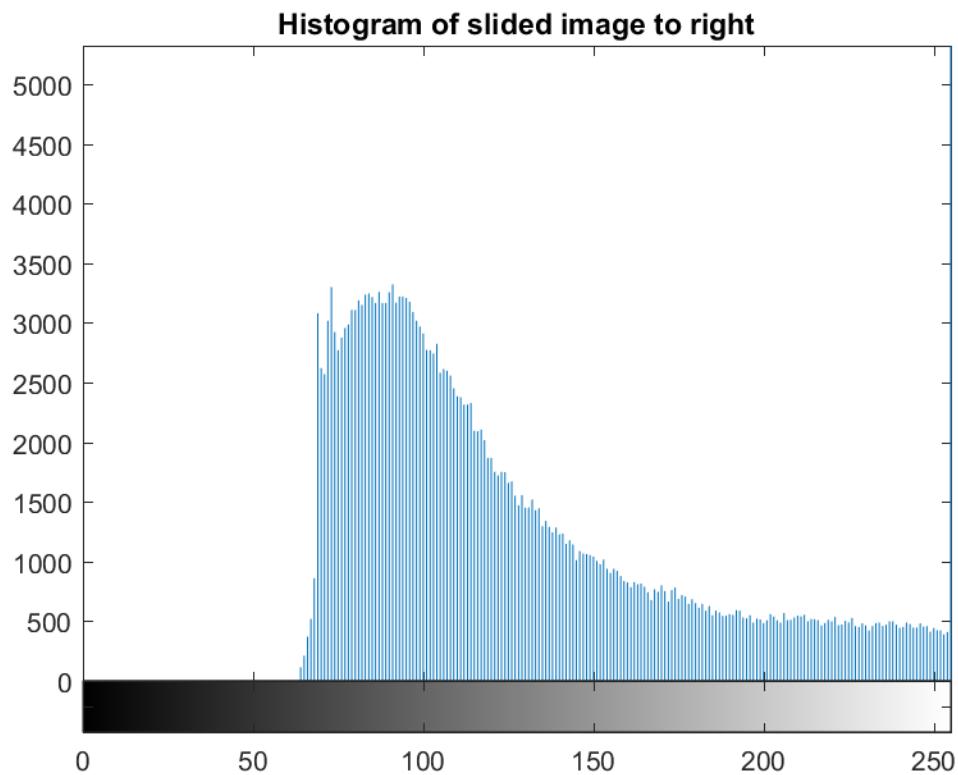
هیستوگرام تصویر ورودی (p8a-input-hist.png)



تصویر بعد از : (p8a-slide-right.png) slided histogram to right



هیستوگرام تصویر حاصل از : (p8a-image-slided-right.png) sliding to right

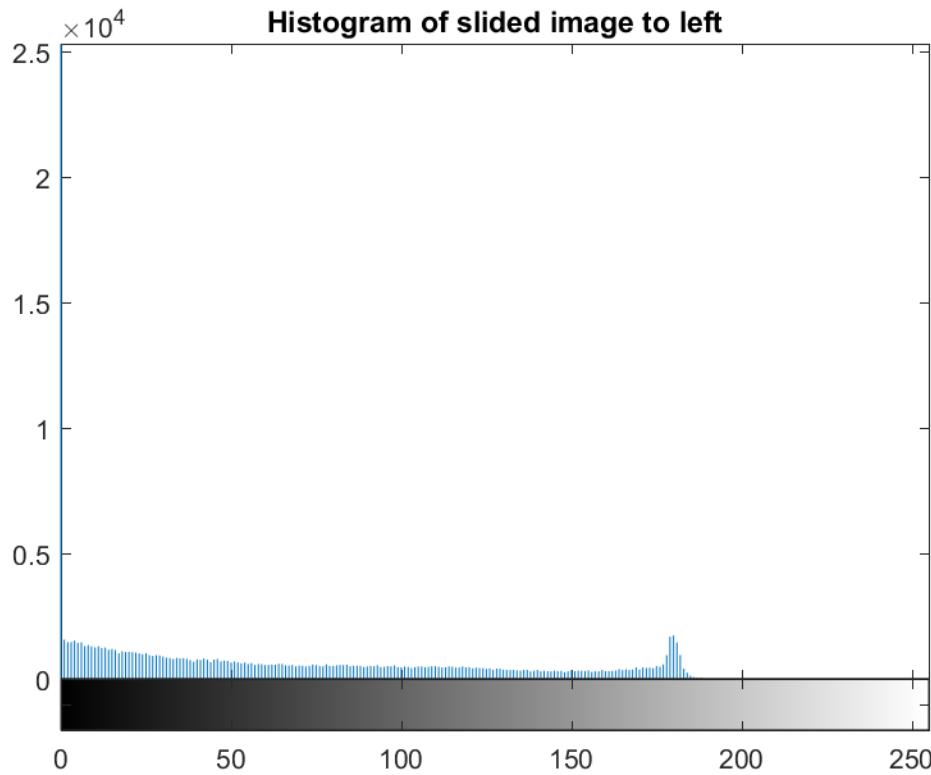


در تصویر ورودی کمترین شدت روشنایی برابر با صفر است، با جمع کردن شدت روشنایی پیکسل‌های تصویر با عدد ۶۴، همین طور که در هیستوگرام تصویر حاصل از شیفت به سمت راست می‌بینیم هیچ پیکسلی شدت روشنایی ۰ تا ۶۳ ندارد، کمترین شدت روشنایی در تصویر جدید برابر با ۶۴ است، و اگر هیستوگرام جدید را با هیستوگرام تصویر ورودی مقایسه کنیم مشاهده می‌کنیم دقیقاً همان هیستوگرام تصویر اصلی است با این تفاوت که ۶۴ واحد به سمت راست شیفت پیدا کرده است. کنترast تصویر کم شده است و تصویر washed out شده است.

تصویر بعد از : (p8a-slide-left.png) slided histogram to left



هیستوگرام تصویر حاصل از :(p8a-image-slided-left.png) sliding to left



در تصویر ورودی بیشترین شدت روشنایی برابر با ۲۵۵ است، با کم کردن ۶۴ از شدت روشنایی پیکسل های تصویر، همین طور که در هیستوگرام تصویر حاصل از شیفت به سمت چپ می بینیم هیچ پیکسلی شدت روشنایی ۱۹۲ تا ۲۵۵ ندارد، بیشترین شدت روشنایی در تصویر جدید برابر با ۱۹۱ است، و اگر هیستوگرام جدید را با هیستوگرام تصویر ورودی مقایسه کنیم مشاهده می کنیم دقیقا همان هیستوگرام تصویر اصلی است با این تفاوت که ۶۴ واحد به سمت چپ شیفت پیدا کرده است. کنتراست تصویر کم شده است و تصویر تیره شده است.

قسمت b

کدهای این قسمت در فایل های hist_matching.m و p8b.m قرار دارد. در فایل hist_matching.m زیر نوشته ام:

```

function hist_match_img = hist_matching(img, img_ref)
    % This function gets two images then it applies
    % histogram matching on first image. img_ref is
    % the picture that functions tries to turn histogram of img like it

```

این تابع دو تصویر به عنوان ورودی می‌گیرد که هیستوگرام تصویر اول را باید با استفاده از روش histogram matching شبیه به هیستوگرام تصویر دوم بکنیم. در این تابع ابتدا تعداد استفاده‌های هر شدت روشنایی توسط پیکسل‌های تصویر ورودی شمارده می‌شود سپس با تقسیم تعداد تکرار هر شدت روشنایی بر تعداد کل پیکسل‌ها احتمال هر شدت روشنایی در تصویر اصلی پیدا می‌شود، سپس فراوانی تجمعی احتمال شدت روشنایی‌ها را که حساب کردیم را پیدا می‌کنیم. بعد فراوانی تجمعی را ضرب در ۲۵۵ که بزرگ‌ترین شدت روشنایی است می‌کنیم سپس round می‌کنیم. تا اینجای کار آرایه‌ای به دست آورده‌یم که آن را S نام‌گذاری می‌کنیم و این کار تا اینجا شبیه به کاری بود که در histeq_mine.m در کد histogram equalization انجام دادیم. در صورتی که می‌خواستیم histogram matching انجام دهیم مقدار خانه‌ی آرایه S مشخص می‌کرد در تصویر ورودی شدت روشنایی 1-a را باید با مقدار [i]S جایگزین کنیم.

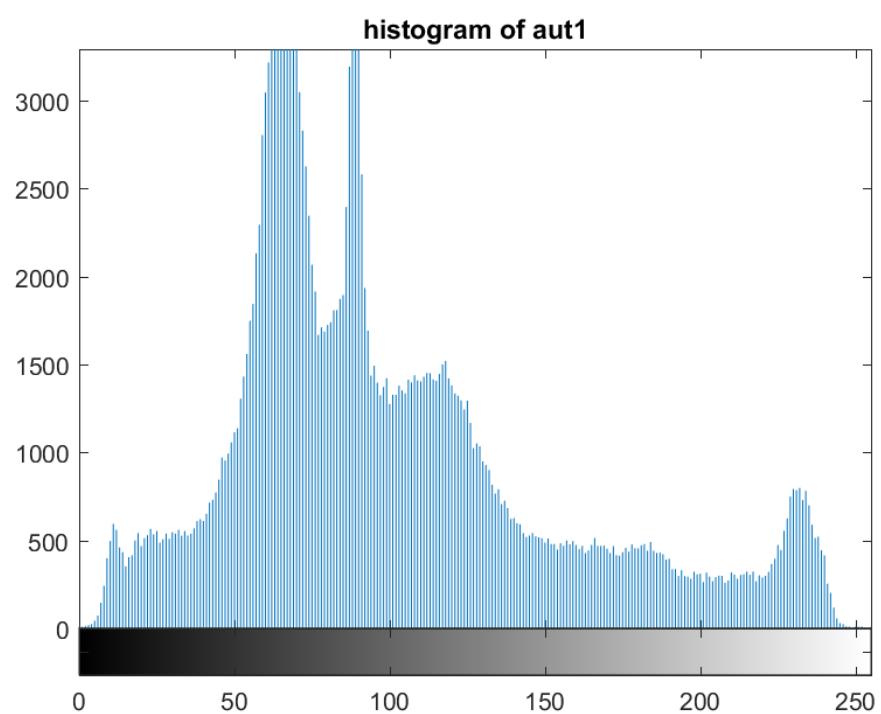
همین کار را هم برای تصویر مرجع (img_ref) انجام می‌دهیم، که نام آرایه حاصل را g می‌گذاریم. برای اینکه عمل histogram matching را انجام دهیم برای هر خانه a در S کمترین مقدار در g را پیدا می‌کنیم به طوری که مقدار [i]S به [j]g نزدیک‌تر از سایر مقدارها در آرایه g باشد. با تغییر شدت روشنایی a به z در تصویر ورودی، هیستوگرام تصویر ورودی تا جای ممکن شبیه به هیستوگرام تصویر رفرنس می‌شود، از آنجایی می‌گوییم تاجای ممکن به خاطر این که مقدار پیکسل‌ها گستته است و در کتاب گنزالس توضیح داده است در صورتی هیستوگرام دقیقاً یکسان با هیستوگرام تصویر مرجع می‌شود که مقدارها پیوسته باشند.

در p8b.m تصویرها را می‌خوانم و نمایش می‌دهم و هیستوگرام آن‌ها را رسم می‌کنم و سپس تابع histogram matching را بر روی آن‌ها فرا می‌خوانم و در انتهای تصویر حاصل از histogram matching و هیستوگرامش را رسم می‌کنم در زیر خروجی این کد را مشاهده می‌کنید:

:aut1 تصویر



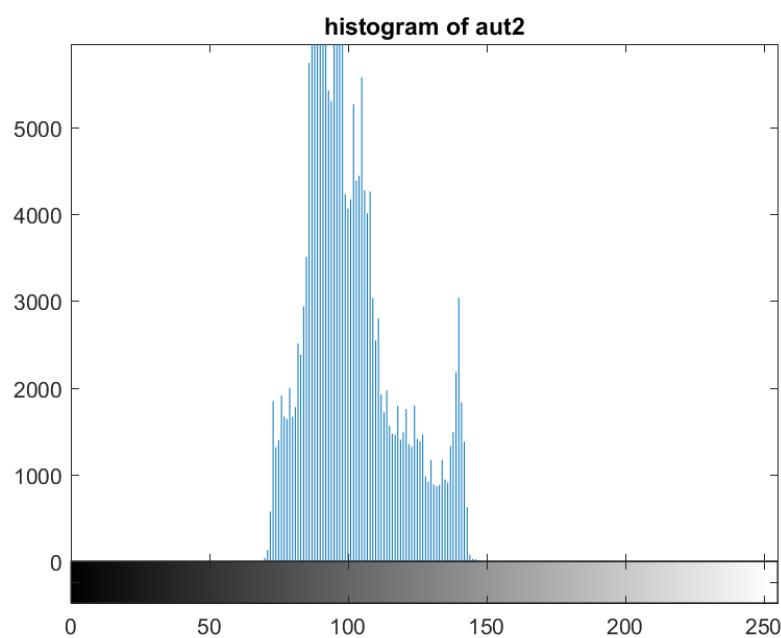
هیستوگرام aut1



تصویر : aut2



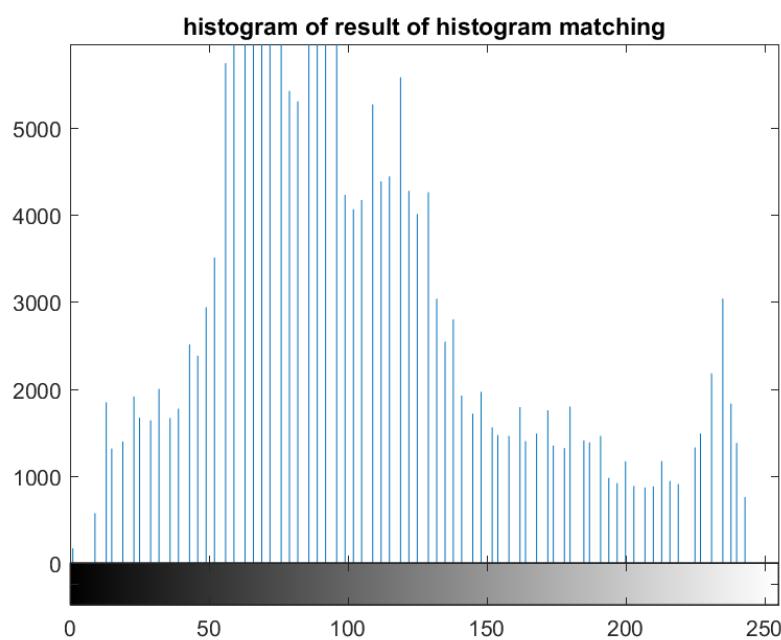
هیستوگرام : aut2



تصویر حاصل از histogram matching



هیستوگرام تصویر حاصل از histogram matching



همین طور که در تصویر حاصل از `histogram matching` دیده می‌شود، تصویر `aut2` بسیار شبیه به `aut1` شده به طوری که با چشم نمی‌توان تفاوتی دید. هیستوگرام تصویر حاصل شبیه به به هیستوگرام تصویر یک شده است.

قسمت C

کدهای این قسمت در `p8c.m` قرار دارد. ابتدا تصویر را خواندهام سپس تصویر و هیستوگرام آن را نمایش داده‌ام. برای `intensity windowing` طبق روش زیر عمل می‌کنم:

Intensity Windowing



- A clamp operation, then linearly stretching image intensities to fill possible range
- To window an image in $[a,b]$ with max intensity M

$$f(p) = \begin{cases} 0 & \text{if } p < a \\ M \times \frac{p-a}{b-a} & \text{if } a \leq p \leq b \\ M & \text{if } p > b \end{cases}$$

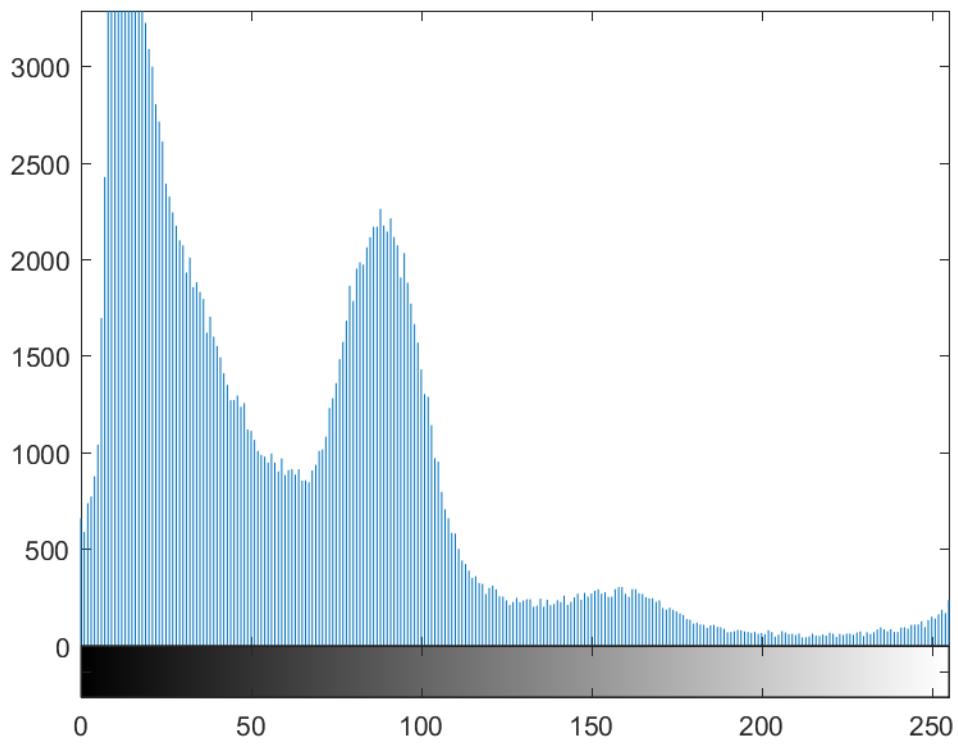
با استفاده از ابزار `imtool` شدت روشنایی گریه را بررسی کردم و a و b را بر اساس آن برابر ۲ و ۴۵ قرار دادم و مقدار M را برابر ۲۵۵ قرار دادم تا در تصویر نهایی شدت روشنایی ۴۵ در تصویر ورودی به شدت روشنایی ۲۵۵ تبدیل شود. در ادامه خروجی‌های کد را مشاهده می‌کنید:

تصویر ورودی:



You think this is a fucking joke?

هیستوگرام تصویر ورودی (p8c-input-hist.png)

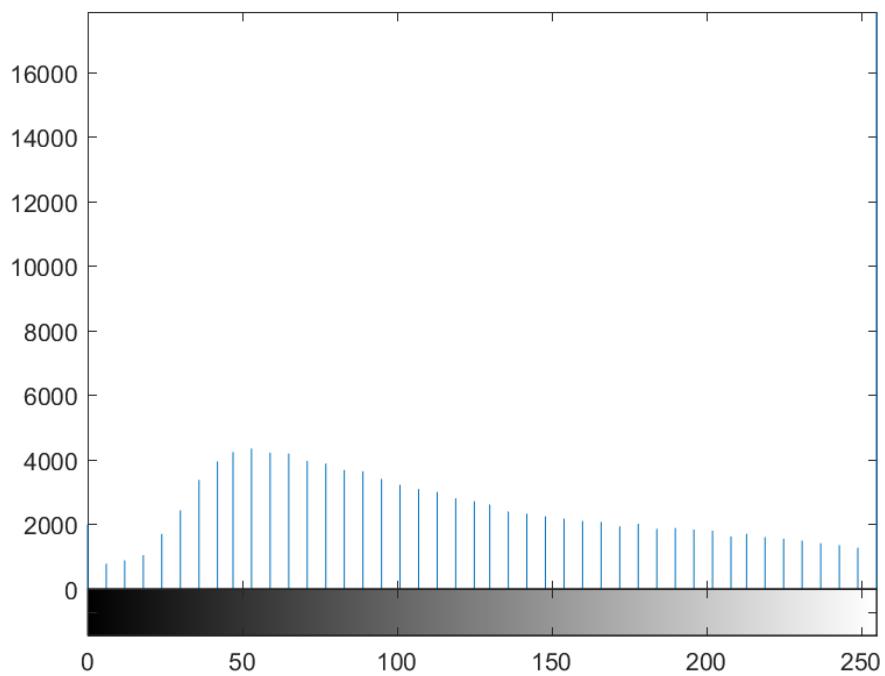


تصویر بعد از اعمال intensity windowing : (p8c-result of intensity windowing.png)



You think this is a fucking joke?

هیستوگرام تصویر بعد از اعمال intensity windowing : (windowing.png



همین طور که در تصویر خروجی مشاهده می شود پس زمنیه دمپایی ها و گربه حذف شده است و شدت روشنایی ۲ تا ۴۵ در هیستوگرام قبلی در بازه‌ی ۰ تا ۲۵۵ هیستوگرام جدید قرار گرفته است.

سوال ۹

کدهای این قسمت در پوشه‌ی p9 قرار دارد.

قسمت ۲

کد این قسمت و فایل p9a.m قرار دارد، در کد تصویرها را می‌خوانم و سپس از آن‌ها میانگین می‌گیرم. تصویرهای یک تا بیست تصویرهایی نویزی هستند که با میانگین گیری از آن‌ها می‌توان اثر نویز را کم کرد.

خروجی‌های کد را در زیر مشاهده می‌کنید:

میانگین دو تصویر اول:



میانگین پنج تصویر اول:



میانگین ده تصویر اول:



میانگین بیست تصویر اول:



همین طور که مشاهده می شود با با میانگین گیری از تصویرهای بیشتر اثر نویز کمتر شده است.

b قسمت

کدهای این قسمت در `p9_snr.m` و `p9b.m` قرار دارد. در فایل `p9_snr.m` تابع زیر قرار دارد

```

function snr_value = p9_snr(image, target_image)
    % this function gets two images and it calculates
    % SNR of them. First image is noisy image, second
    % image is target image (noise free).
    % SNR(Signal-to-noise ratio) is defined as the ratio of the power of a signal
    % (meaningful information) and the power of background noise (unwanted signal):

    snr_value = 10*log10(sum(sum(double(image).^2))/...
        sum(sum((double(target_image)-image).^2)));
end

```

که براساس تابع زیر

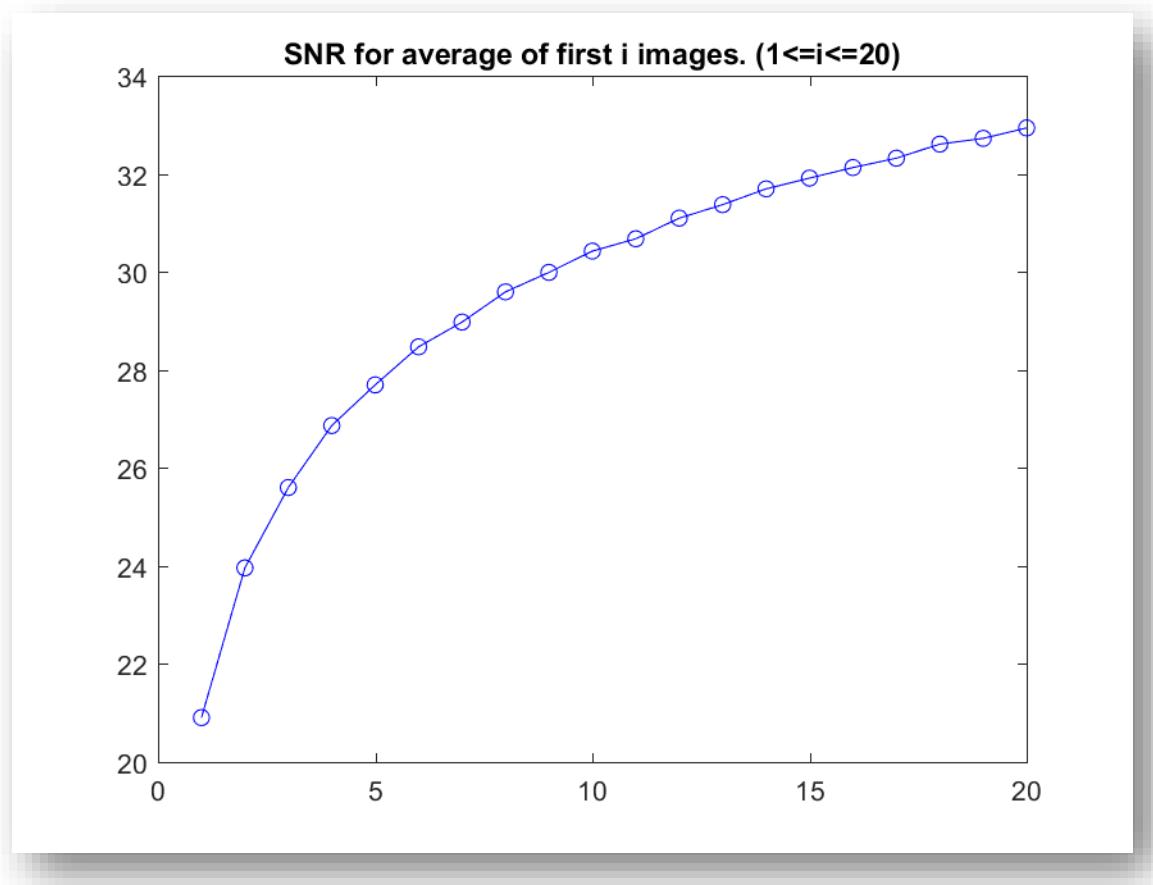
$$SNR = \frac{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \hat{f}(x, y)^2}{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [f(x, y) - \hat{f}(x, y)]^2}$$

البته از تابع زیر هم استفاده می‌شود که من از آن استفاده کرده‌ام.

$$SNR = 10 \times \log(\frac{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \hat{f}(x, y)^2}{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [f(x, y) - \hat{f}(x, y)]^2})$$

میزان SNR را محاسبه می‌کند. البته به جای ضرب در 10 می‌توان ضرب در 20 نیز استفاده کرد.

در کد p9b.m ابتدا تصویر اصلی و مرجع بدون نویز را می‌خوانم و سپس یک به یک تصویرهای نویزی یک تا بیست را می‌خوانم و میانگین تصویرها را حین خواندن هر تصویر جدید محاسبه می‌کنم و با استفاده از تابه p9_snr میزان SNR را برای تصویرهای حاصل از میانگین گیری محاسبه می‌کنم. در زیر نمودار SNR را به ازای میانگین گیری از n تصویر اول مشاهده می‌کنید، n از یک تا بیست تغییر می‌کند:



در زیر مقدار snr به ازای میانگین گیری از تصویرها را مشاهده می کنید:

Columns 1 through 10

20.9051	23.9601	25.6002	26.8643	27.6945	28.4706	28.9758	29.5917	29.9893	30.4237
---------	---------	---------	---------	---------	---------	---------	---------	---------	---------

Columns 11 through 20

30.6754	31.0958	31.3717	31.6936	31.9135	32.1288	32.3194	32.6069	32.7240	32.9371
---------	---------	---------	---------	---------	---------	---------	---------	---------	---------

همین طور که مشاهده می شود با میانگین گیری از تصویرهای بیشتر قدرت سیگنال به نویز افزایش یافته است، به عبارتی نویز کمتر شده است.

قسمت ۵

کد این قسمت در p9c.m قرار دارد، در این قسمت تصویر بدون نویز و مرجع را ابتدا می‌خوانم و بعد تصویر نویزی مدنظر را می‌خوانم در ادامه تصویرها را نمایش می‌دهم، سپس با استفاده از تابع `imgaussfilt` و با انحراف معیار برابر با ۱,۱ فیلتر را بر تصویر نویزی اعمال می‌کنم و تصویر را نمایش می‌دهم، در آخر `snr` تصویر نویزی قبل و بعد از انجام فیلتر را محاسبه می‌کنم.

در زیر خروجی‌های کد را مشاهده می‌کنید:

تصویر اصلی بدون نویز:



تصویر نویزی یک قبل از اعمال فیلتر:



تصویر نویزی یک بعد از اعمال فیلتر (p9c-gaussfilterd-image.png)



معیار SNR برای تصویر نویزی قبل و بعد از اعمال فیلتر:

```
>> p9c
```

```
SNR of noisy image before applying filter: 2.090507e+01
```

```
SNR of noisy image after applying filter: 2.483494e+01
```

همین طور که مشاهده می شود تصویر نویزی یک بعد از اعمال فیلتر نسبت حالت قبل از اعمال فیلتر بهتر شده است و SNR بزرگ تری دارد.

قسمت d

کد این قسمت در p9d.m قرار دارد، در این قسمت تصویر بدون نویز و مرجع را ابتدا می‌خوانم و بعد تصویر نویزی مدنظر را می‌خوانم در ادامه تصویرها را نمایش می‌دهم، سپس با استفاده از تابع `imfilter` و باسایز کرنل ۳، فیلتر را بر تصویر نویزی اعمال می‌کنم و تصویر را نمایش می‌دهم، در آخر `snr` تصویر نویزی قبل و بعد از انجام فیلتر را محاسبه می‌کنم.

فیلتر در این روش یک ماتریس مربعی است که المنت‌های آن برابر است با $\frac{1}{\text{size of kernel} * \text{size of kernel}}$

در زیر خروجی‌های کد را مشاهده می‌کنید:

تصویر اصلی بدون نویز:



تصویر نویزی یک قبل از اعمال فیلتر:



تصویر نویزی یک بعد از اعمال فیلتر (p9d-smoothed-image.png):



معیار SNR برای تصویر نویزی قبل و بعد از اعمال فیلتر:

>> p9d

SNR of noisy image before applying filter: 2.090507e+01

SNR of noisy image after applying filter: 2.333869e+01

همین طور که مشاهده می شود تصویر نویزی یک بعد از اعمال فیلتر نسبت حالت قبل از اعمال فیلتر بهتر شده است و SNR بزرگ تری دارد.

قسمت ۶

خروجی روش میانگیری از ۲۰ تصویر:



خروجی روش Gaussian filter



: smoothing روچی خروجی



میزان SNR سه روش:

SNR	
32.9371	میانگین گیری از ۲۰ تصویر اول
23.960	میانگین گیری از ۲ تصویر اول
25.600	میانگین گیری از ۳ تصویر اول
24.834	Gaussian Filter
23.338	Smoothing Filter

در بالا ۳ تصویر خروجی سه روش را مشاهده می‌کنید همین طور که مشاهده می‌شود میانگین ۲۰ تصویر اول بسیار به توصیر اصلی و بدون نویز نزدیک است و نتیجه‌ی بهتری نسبت به دو روش دیگر دارد. میانگین گیری از

۲۰ تصویر اول بالاترین SNR را در بین ۳ روش دارد و چیزی در حدود ۸ واحد از آن‌ها بیشتر. این روش در میان سه روش از همه بهتر عمل کرده است. حتی میانگین گیری از سه تصویر اول SNR بزرگتری نسبت به روش smoothing و Gaussian دارد.

در بین خروجی دو روش Gaussian و smoothing سخت است با چشم بتوان گفت کدام بهتر عمل کرده است. ولی SNR فیلتر Gaussian بیشتر شده است در نتیجه عملکرد بهتری نسبت به smoothing داشته است.

در صورتی که از یک تصویر چندین تصویر نویزی متفاوت داشته باشیم بهترین روش از میان سه روش بالا این است از تصویرها میانگین بگیریم. در این روش بلور شدن مشاهده نمی‌شود و همینطور برخلاف دو روش دیگر لازم به داشتن سیاستی برای حاشیه‌های تصویر نیستیم. لازمه‌ی این روش این است چند تصویر با شرایط یکسان (زاویه و مکان یکسان، نوع نویز یکسان، نور پردازی یکسان و ...).

روش Gaussian filter و smoothing filter هر دو با استفاده از Convolution انجام می‌شوند و هزینه‌ی محاسباتی بالایی دارند به خصوص اگر سایز تصویر و فیلتر بزرگ باشد. یکی بدی دیگر که این دو روش دارند آن است باید تصویر را pad کنیم و یا از جایی فیلتر را اعمال کنیم که فیلتر درون تصویر قرار بگیرد و حاشیه‌ی تصویر متفاوت می‌شود. این دو روش برای وقتی که مقدار نویز کم است خوب عمل می‌کنند ولی هر چه میزان نویز بیشتر می‌شود کاراییشان کم می‌شود و اگر بخواهیم از کرنل‌های بزرگ‌تری استفاده کنیم بلور شدن تصویر زیاد می‌شود. روش گاوویسن به پیکسل‌های نزدیک‌تر به مرکز کرنل اهمیت و وزن بیشتر می‌دهد به همین دلیل نتایج بهتری می‌دهد و طبیعی تر عمل می‌کند. ولی در smoothing filter تمام به درایه‌های کرنل وزن یکسانی دارند به همین دلیل میزان بلور شدن بیشتر است.

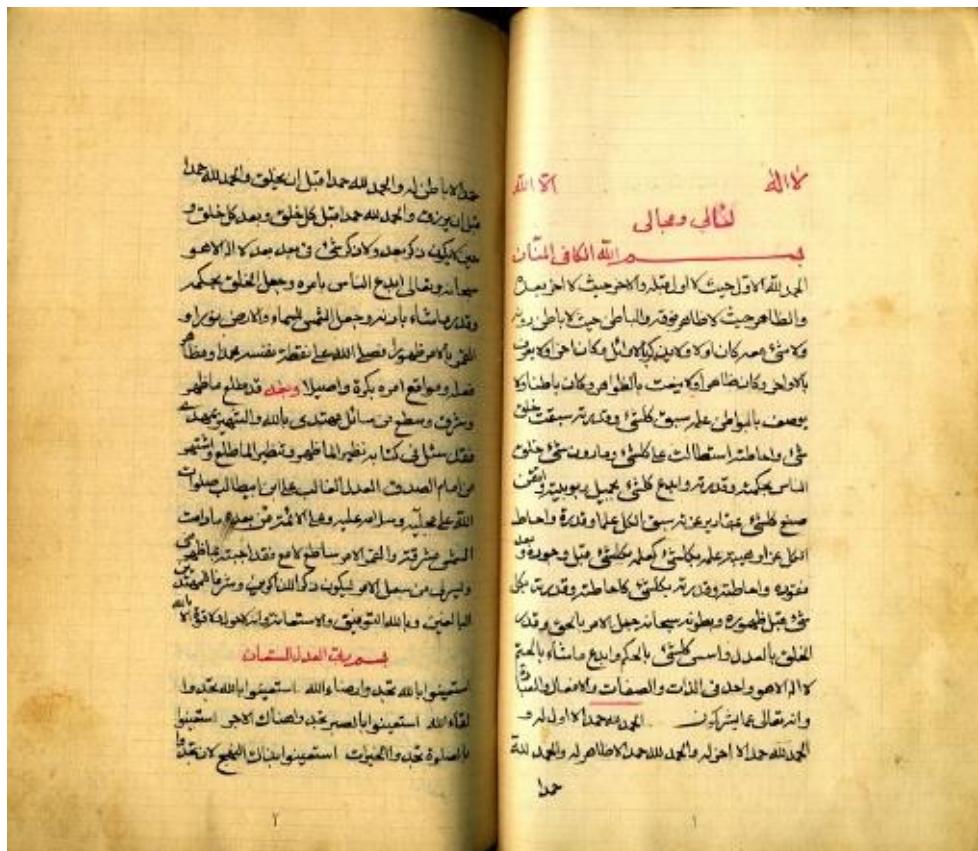
کدهای این قسمت در پوشه‌ی p10 قرار دارد.

قسمت ۲

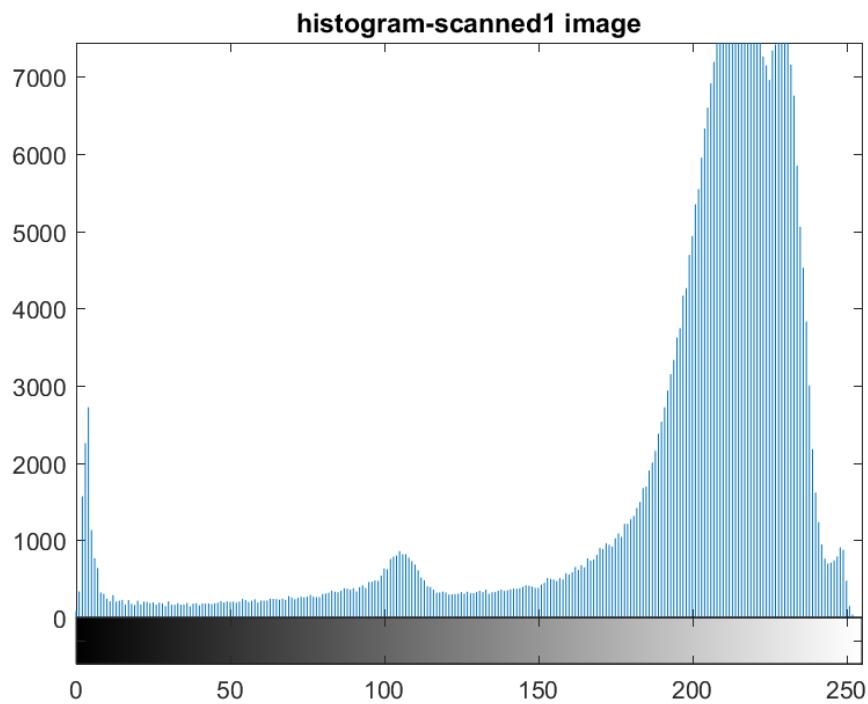
کدهای این قسمت در p10a.m قرار دارد. در این تصویر ابتدا هر تصویر را خوانده‌ام سپس هیستوگرام آن تصویر را رسم کرده‌ام و بر اساس آن و با آزمون خطا threshold را مشخص کرده‌ام. پیسکسل‌های بیش از و مساوی threshold را برابر با ۲۵۵ و کمتر از آن را برابر با ۰ قرار داده‌ام.

در زیر خروجی‌های کد را مشاهده می‌کنید:

: scanned1 تصویر



هیستوگرام تصویر scanned1 : (p10a-scanned1-hist.png)



همین طور که مشاهده می شود بعد از شدت روشنایی 90 بزرگ ترین peak شروع به ایجاد می شود که همان پس زمینه است، با آزمون خطا برای حفظ نوشتہ های این تصویر threshold برابر با 90 را انتخاب می کنم.

تصویر scanned1 بعد از thresholding : (p10-scanned1-thresholded.png)

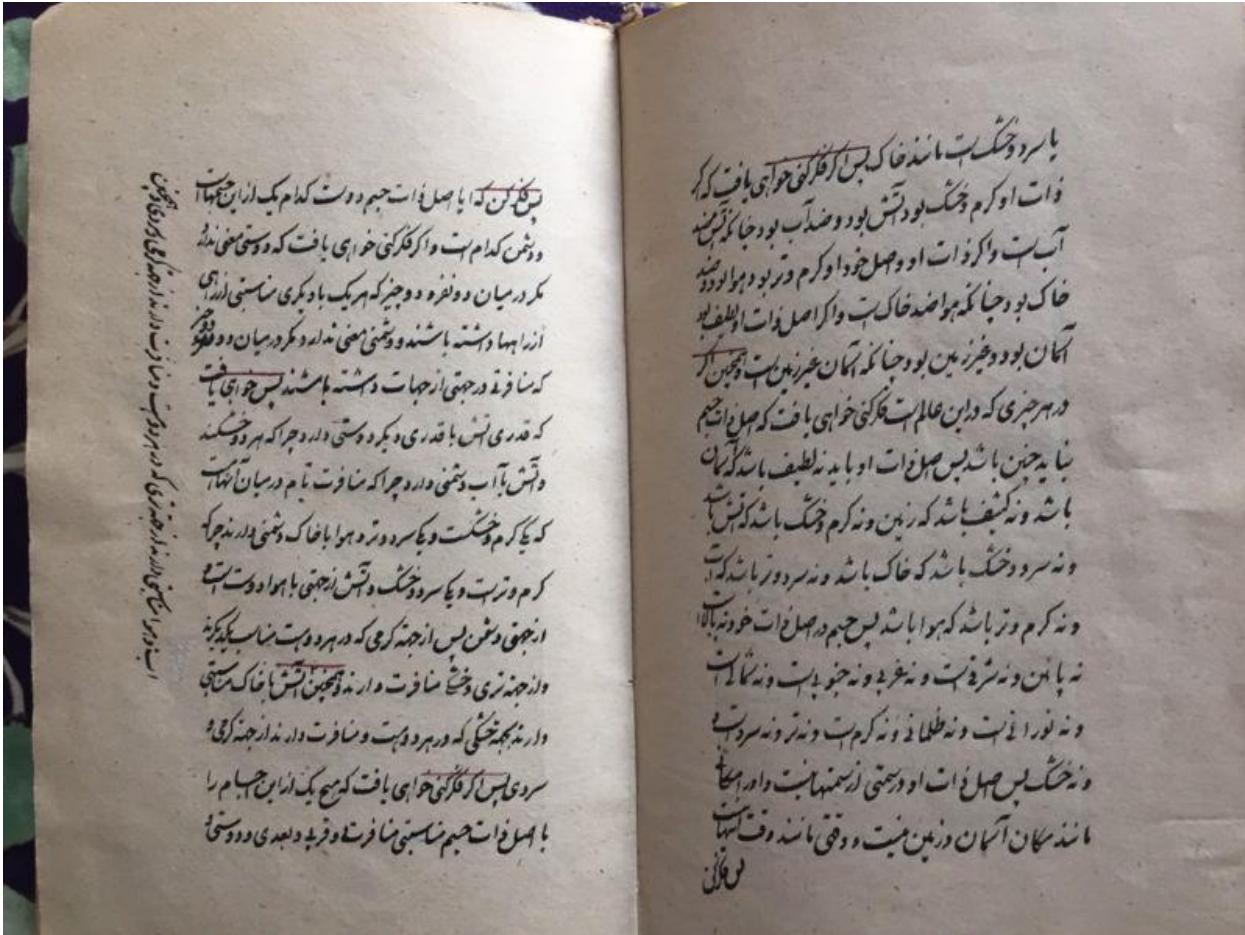
11

卷之三

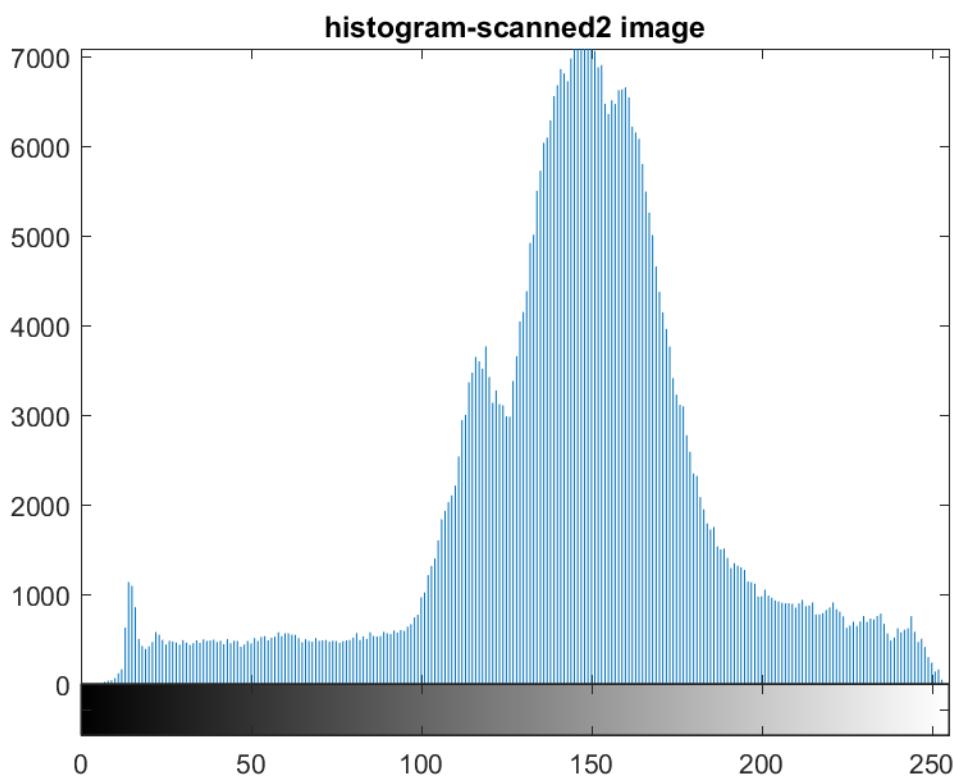
جداً باطن له والجود ملحداً قبل استحقاقه بالمجدلة جداً
قبل استحقاقه بالمجدلة جداً قبل كل حقوقه وبعد كل حقوق
هي كثيرة وكبيرة وكثيرة وكبيرة جداً لا إله إلا هو
يجادل وحال أبدع الناس بأمره يجعل الحقائق ينكر
وقد يحيى ما شاء بأمره يجعل الحقائق ينكر بغير إرادة
الله يأكل حظهموا فضلاً الله على تقطيره يفتك بهم وهذا عظيم
فلو لم يتحقق أمره بكرة صاصباً - عند تقطير ما ظهر
كسرت وسطع من سالي همته بالله والشهري يهدى
فتقى سقراً كثاً به لظهور المأهول وفضلاً لما تعلم وشهد
في أيام الصدقة العدد العالى على ابن أبي طالب صلوات
الله عليه عليه وسلم عليه وعلى الأئمة من صواب ما رأى من
المتشدقين تزور المهن الامر سلامة لامع فقد اجهزة يا اخي
ولم يرى من قبل لا اهم ليكون ذكر الله كفيه وشرف المهدى
الماضية وخلافة المؤمنين والاسلاميات اشكوا لك اتفاق الا
بس مررت بغير تفاصيل
استعينوا بالله تجدوا به ما استعنوا به الله تعالى بما
ذهبوا به استعينوا بالصبر عبد ولصان الاجي استعينوا
بالصلوة عبد والغافل استعينوا بذات المفعول كأنتم

4

تصویر2 :scanned2



هیستوگرام تصویر 2:(p10a-scanned2-hist.png) scanned2



همین طور که مشاهده می شود بعد از شدت روشنایی 100 بزرگترین peak شروع به ایجاد می شود که همان پس زمینه است، با آزمون خطا برای حفظ نوشه های این تصویر threshold برابر با 100 را انتخاب می کنم.

: (p10-scanned2-thresholded.png) thresholding scanned2 تصویر

پا سر و چکت است اند خاک بیش از گلکنی خواهی بافت که
ذات او کرم و حکمت بود و نیش بود و خذاب بود و جایگاهی نداشت
آبست داکر ذات او و معلم خود او کرم و فربود و برازو پرست
خاک بود و جایگاهی مراد خاک است داکر اصل ذات او عطفه
آشان بود و بجزرین بود و جایگاهی آشان بجزرین است همچنان که
در برجی که درین علم است گلکنی خواهی بافت که همین اتفاق
باشد چون باشد بس صدیقات او باشد و لطفیست باشد گلکنی
باشد و نه گفته باشد که زین و نه کرم و حکم باشد که نه
و نه سرمه و خلک باشد که خاک باشد و نه سرمه و باشد که نه
نه کرم و نه باشد که برو باشد بسیج هم و معلم ذات خود نه
نه باشد که برو باشد بسیج هم و معلم ذات خود نه
نه نور را نیست و نه غرد و نه جوز باشد و نه نارنگی
و نه خلک بسیج هم است او و سخنی نه سخنی باشد داده
اند سخان آشان زین نیست و حقیقی باشد وقتی که

در هر دو تصویر امکان حذف شیار بین دو صفحه نیست، زیر با این کار و تغییر **threshold** بخشی از متن را از دست می‌دهیم و متن خوانایی اش را از دست می‌دهد. یک مشکل دیگر که این روش دارد این است که در ناحیه‌های مختلف می‌توان حدهای مختلفی را برای **threshold** کردن استفاده کرد و کیفیت تصویر **threshold** شده را افزایش داد ولی این روش این امکان را نمی‌دهد. در هر دو متن به خوبی قابل مشاهده است و بیشتر پس زمنیه بجز شیار حذف شده است. همین طور در تصویر اول چند کلمه از نوشتة‌های صفحه‌ی سمت راست قسمت بالا حذف شده است.

قسمت b

کدهای این قسمت در p10b و adaptive_thresholding_non_overlapping.m قرار دارد، در فایل دوم تابه زیر را نوشتئام:

```
[function adaptive_thre_img = adaptive_thresholding_non_overlapping(img, m, threshold_ratio)
% This function divide image to m*m non-overlapping blocks
% then it applies histogram equalization on each one.
```

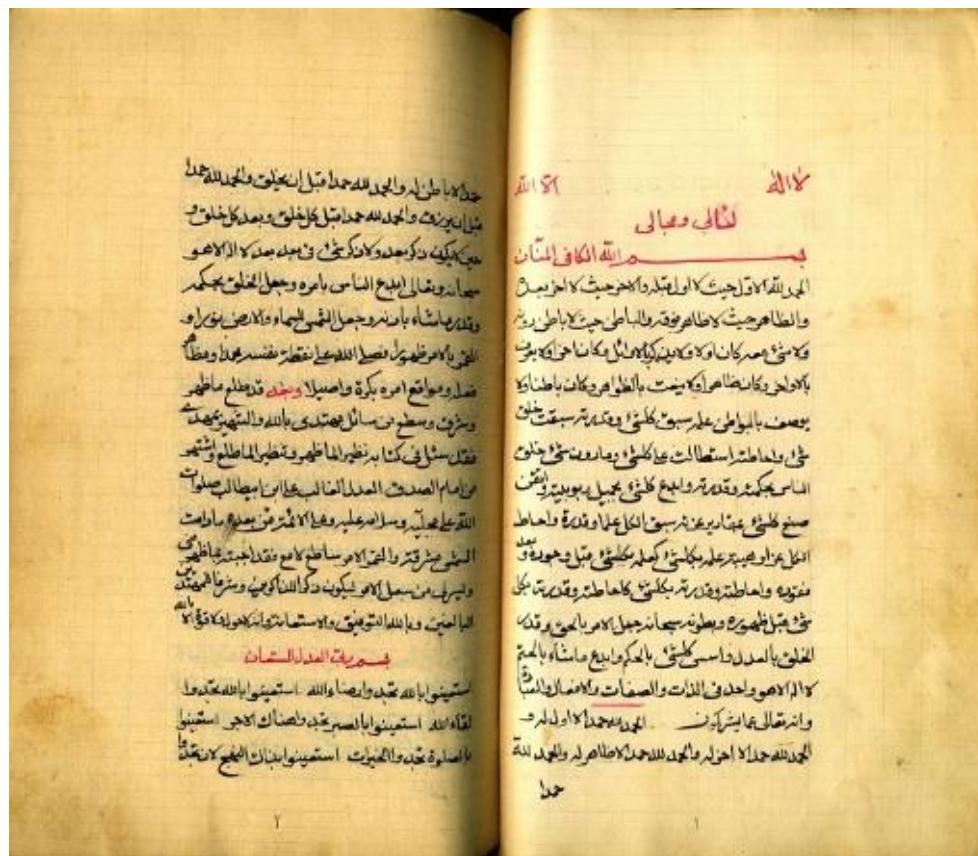
این تابه تصویر را به مربع‌های $m \times m$ که همپوشانی ندارند تقسیم می‌کند و سپس در هر بلاک کمترین و بیشترین شدت روشنایی را پیدا می‌کند و با توجه به کمترین و بیشترین شدت روشنایی موجود در هر بخش، درصد اول شدت روشنایی‌های آن بلاک را صفر می‌کند و باقی را ۲۵۵ threshold_ratio.

در کد p10b.m تصویرها را خواندهام و هیستوگرام آن‌ها را رسم کردهام سپستابع بالا را برای آن‌ها فرا می‌خواندهام و نتیجه‌ها را نمایش داده‌ام.

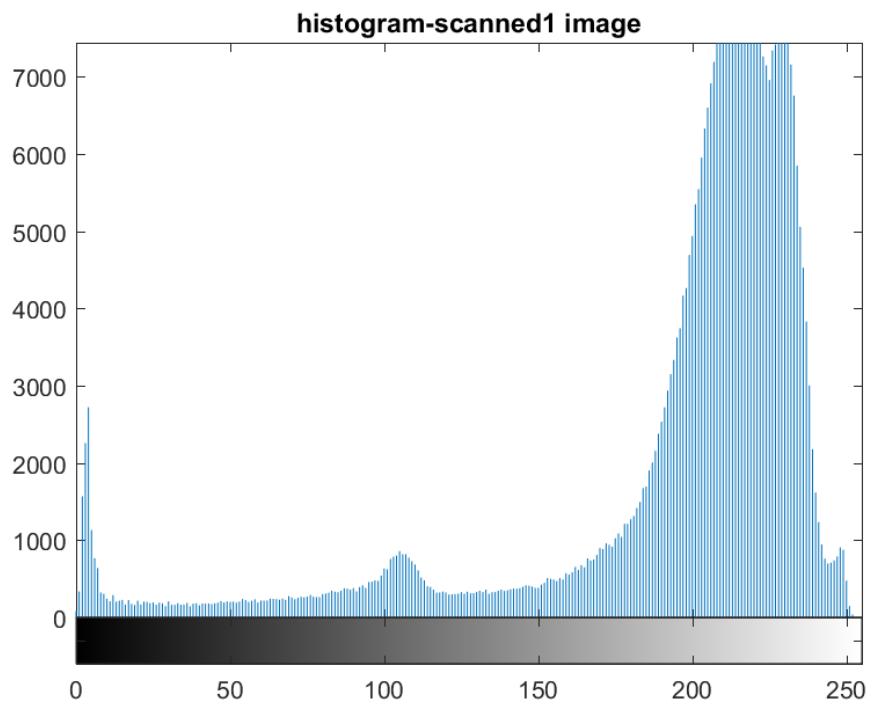
البته کد دیگر هم برای adaptive thresholding در فایل adaptive_thresholding.m نوشته‌ام که از روش بلاک‌های هم پوشان استفاده می‌کند از آنجایی که نتیجه‌ی مدنظرم را نداد از تابع بلاک‌های غیر همپوشان استفاده کرده‌ام.

در زیر خروجی‌های کد را مشاهده می‌کنید:

: scanned1 تصویر



: (p10b-scanned1-hist.png) scanned1 هیستوگرام تصویر



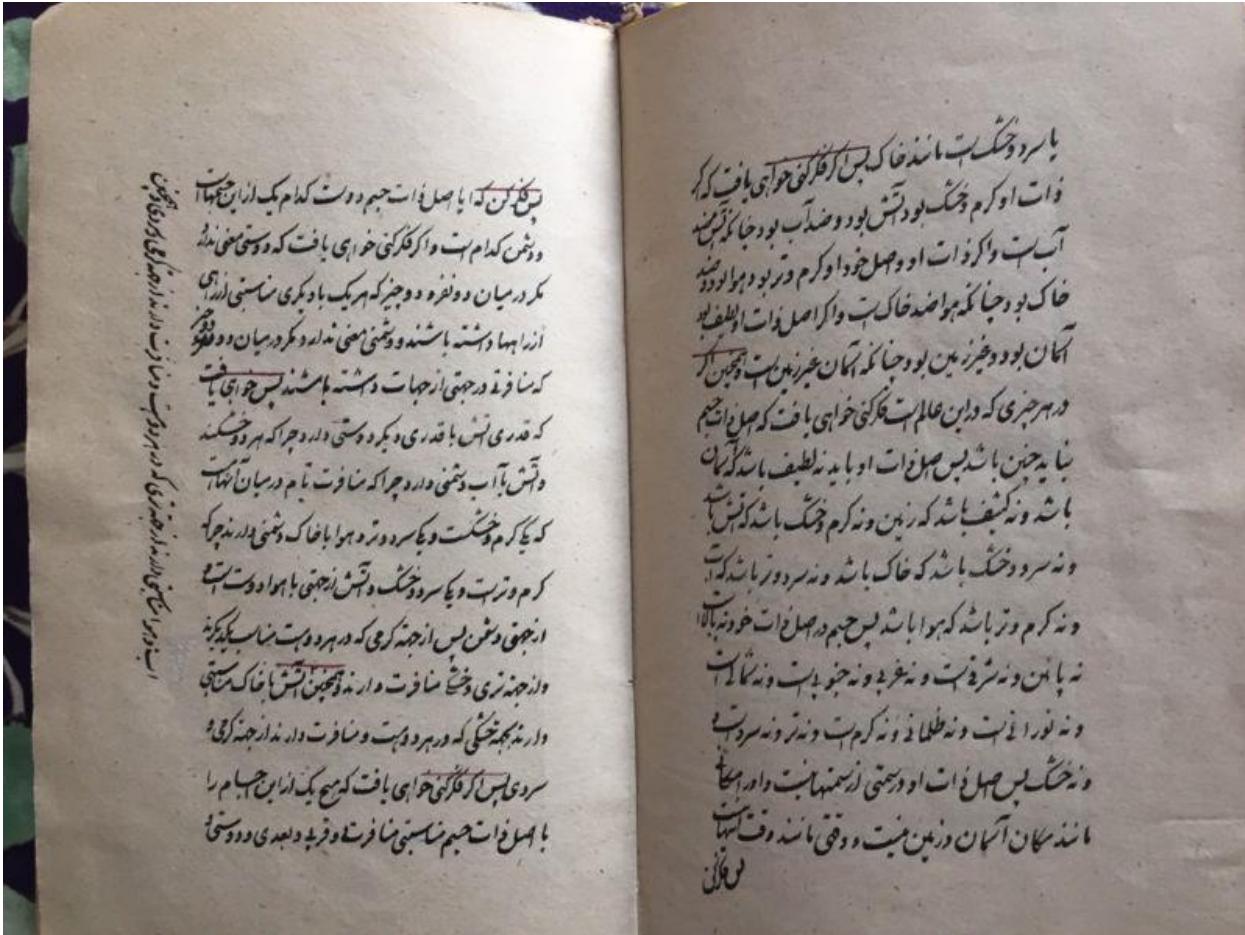
تصویر scanned1 بعد از thresholding : (p10b-scanned1-adaptive-thresholded.png)

✓

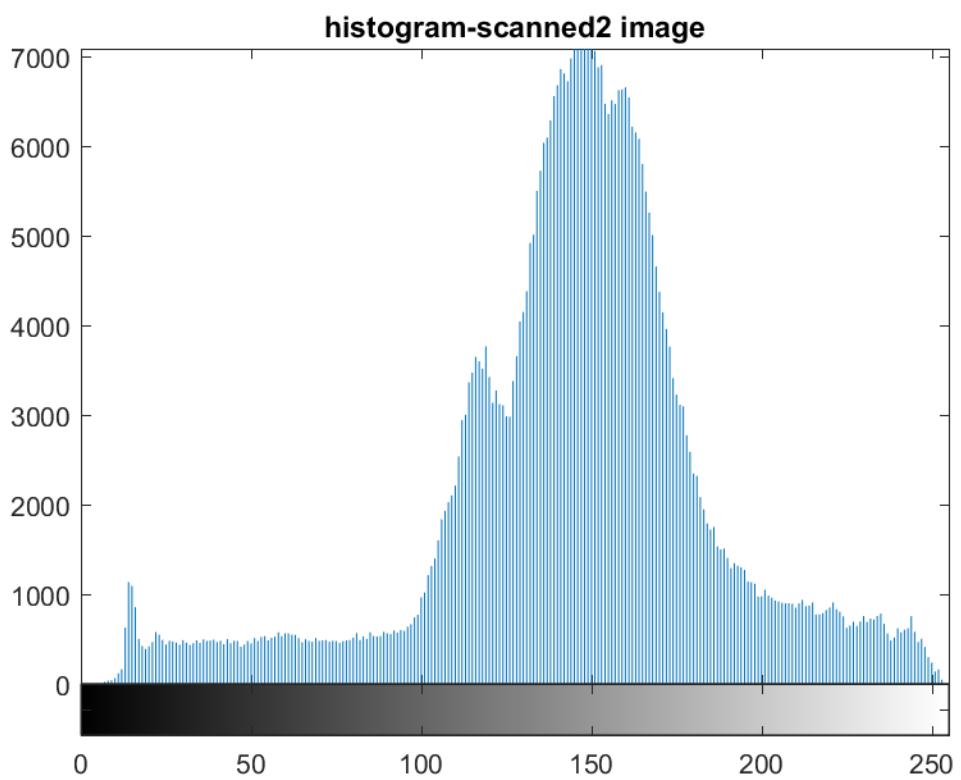
15 of 15

هذا اصله والحمد لله مدح ابي الحجاج والحمد لله هذا
بيان وبيان وبيان وبيان وبيان وبيان وبيان وبيان
اللهم يا حسنه ربنا ربنا ربنا ربنا ربنا ربنا ربنا ربنا
فعلم وساقع امره بذكره واصيلاً قد علم ما يضره
وسرق وسرق من ماله همدى بالله والشهى بهمدى
فعلم شفى كبر نفع لما يضره بظهور الماء وشفي
من اقام الصدف المدى العالى بغير ابصاره صفو
اللهم يا حسنه ربنا ربنا ربنا ربنا ربنا ربنا ربنا ربنا
اللهم شفى قر واللهم شفى قر واصطب لامع وندى اجيلا ضارها
ونشرى من جهل الامر يكوت ذلك المدى ايجي ونشر المهدى
بالعينين والله التوفيق والاستغاثة بربه فهو ينادي الله
استغاثة الله استغاثة الله استغاثة الله استغاثة الله

خليفة الله أكواز حيث لا دين ملدو الأداء حيث لا نوادر
والناظه حيث لا ناطه بغيره والباض حيث لا يضره
وكافيه مصدر ذاتي ولا يكتفي بالذات ركاب اهون بغيره
باكلور وكمار خالصي كيميت بالظهوه وكمار باطه
يعصف بالاوهن عذر سبقه على اهون ترسبت
شيء فعافت واستعادت على كل شيء وبدوره عجز عن انتقام
لأنه يكتفي وقدره ما عليه كل شيء بغيره يكتفي
معن هنلي يفتار عين شرقي المكان على اهونه وكماره ولها
الخل عذابه يربطه بكل شيء لكنه يكتفي بتلك الوجهة
فنهذه فعاظته وفخره بكل شيء كلامه وفوقه يكتفي
شيء مثل ظهوره وبطونه يحصل لأهونه اعنده
الخلفي بأهله دناءه كشيء يانه كويه ينبعه شفاءه اعنده
كمار الأهواء وأحدى الذئب الصفات والأصل الشفاعة
واندر على يمني البشارة كشيء يهدى أهلاه لآهله
الجود يهدى لا آخره وإنجذب له جملاً كاصفه بغيره ونحوه لشه



هیستوگرام تصویر 2:(p10b-scanned2-hist.png) scanned2



تصویر2 scanned2 (p10b-scanned2-adaptive-thresholded.png) thresholding از بعد

باید بگفت این میان سه ایامی است که فکر کنید و بازی بگردید
و دست و چشم و چشیدن بود و این سه بود و هر چند که کار نمایند
آنست و اگر آنست او و صلح خواهد اد کرم و فرزند بود و هر کسی که
خواهد در جا نمایند مراصد خواست و اگر میلاد است این طبقه
آنها بود و خوبی هم نداشتند آنها غیر عین هستند و چنان که
در سر برخی که در این عالم است نهایتی خرابی باخت که همه اینها
با همین ایام سه میلاد است او این طبقه همان کسانی است
که آش و آش بخواهند که بجن و زکر کرم حذف شوند که نیش آش
و نیش در حرف ناش که حاکم آش و زکر کرم حذف شوند که نیش آش
و زکر کرم و زکر آش لیستند و شده بجهنم و مصلحت هم نهادند هر کسی که
نیش باشند و میسر نباشد و میسر نباشد و میسر نباشد و میسر نباشد
و نیش نباشد و میسر نباشد و میسر نباشد و میسر نباشد و میسر نباشد
و میسر نباشد و میسر نباشد و میسر نباشد و میسر نباشد و میسر نباشد

در هر دو خروجی نوشته به خوبی از متن جدا شده است و نوشتہ‌ها خوانا هستند. همین طور که مشاهده می‌شود، مشکل تصویر یک در روش قبل که چند کلمه پاک شده بود از بین رفته است، بخش بسیار زیادی از شیارها از بین رفته‌اند و متن‌ها خواناتر هستند.

سوال ۱۱

کدهای این قسمت در p11 قرار دارد.

قسمت a

کد این قسمت در mse_template.m و p11a.m تابع زیر را نوشته‌ام:

```
function mse_map = mse_template(img, img_temp)
    % m = size(img_temp, 1)
    % This function divide image to m*m overlapping blocks
    % then it calculates MSE error of each blocks and template.
```

این تابع، یک تصویر و template را می‌گیرد و از اولین پیسکل تصویر که template در آن جا می‌شود، را از روی تک تک پیکسل‌ها عبور می‌دهد و میزان MSE و هر پیکسل همراه با همسایگی‌اش را حساب می‌کند، در آخر ماتریس mse_map را بازمی‌گرداند که هر درایه از آن برابر است با mse ان پیکسل و همسایگی‌اش با template. در کد p11a.m تصویر template و مجموعه سکه‌ها را می‌خوانم، سپس تابع mse_template را فرامی‌خوانم تا میزان mse را برای تمام پیکسل‌ها به ازای قرار دادن template بر روی پیکسل حساب کنم، بعد از آن کمترین پیکسلی را در mse_map پیدا می‌کنم که کمترین مقدار mse را دارد و محل سکه را نشان می‌دهد. بعد از آن در تصویر اصلی محل سکه‌ای که پیدا کرده‌ام را نگه می‌دارم و باقی عکس را سیاه می‌کنم. در ادامه تصویر حاصل و mse_map را رسم می‌کنم. در زیر خروجی‌های کد را مشاهده می‌کنید:

تصویر ورودی:



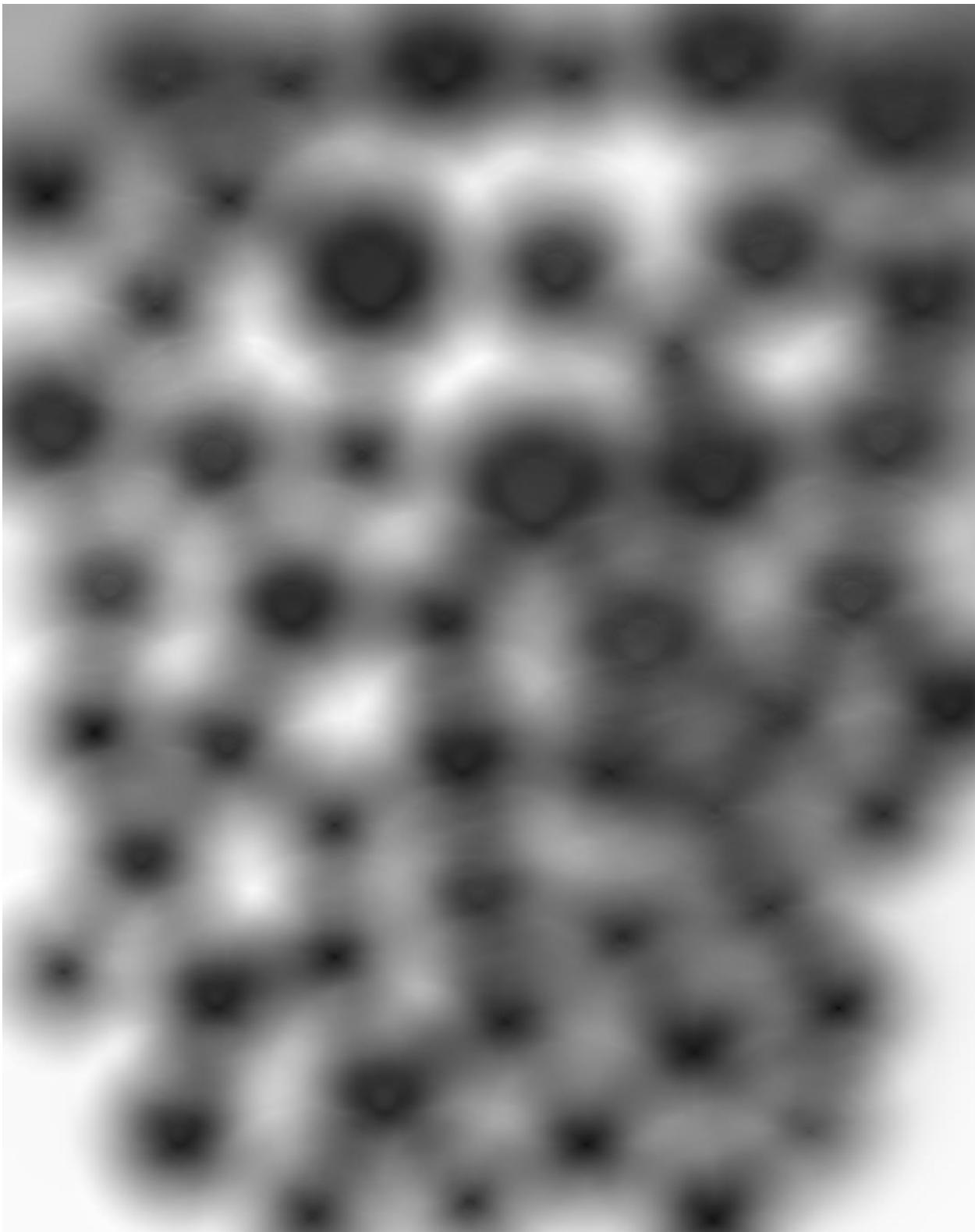
تصوير template



: (p11a-location-matched-coin.png) تصویر محل سکه‌ای که در تصویر اصلی پیدا شده است



تصویر : (p11a-mse-map.png) mse_map

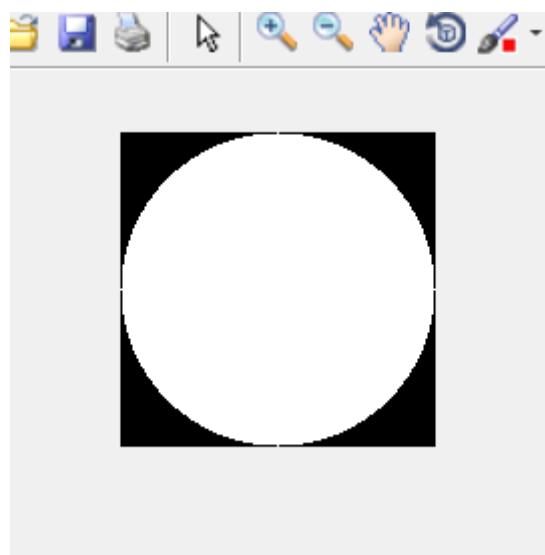


قسمت b

کدهای این قسمت در mse_template_2.m و p11b.m تابعی mse_template_2.m است، در فایل mse_template_2.m قرار دارد که مانند تابع mse_template_2 است ولی یک تفاوت کوچک دارد و آن این است بروز آنجایی که زاویه‌ی سکه مرجع و سکه درون تصویر یکی نیست باید سکه template را با زوایه‌های مختلف چرخش دهیم و سپس به این تابع بدھیم و چرخش template با imrotate باشد می‌شود بعد از چرخش گوشه‌های تصویر خالی بماند. مانند تصویر زیر:



به همین دلیل در این تابع mask زیر را می‌سازم:



و قبل از محاسبه‌ی MSE تصویر template و همسایگی یک پیکسل برای پیدا کردن شباهت، Mask بالا را ضرب نظری در نظری در هر دو می‌کنم به همین دلیل حاشیه‌ی سیاه حاصل از چرخش بی تاثیر می‌شود. باقی کد کاملاً شبیه به تابع mse_template قسمت قبل سوال است.

در فایل p11b.m تصویر مجموعه سکه‌ها و سکه‌ی template را می‌خوانم سپس سکه‌ی template را با استفاده از زاویه‌های 0° , 20° , 40° , ... 350° درجه چرخش می‌دهم و تابع mse_template_2 را فرامی‌خوانم، و هر بار نتیجه حاصل و mse (هر پیکسل) را رسم می‌کنم و در آخر زاویه‌ای را گزارش می‌کنم که کمترین MSE را باعث شده است که آن زاویه زاویه‌ای است که اگر template را با آن چرخش دهیم سکه مورد نظر پیدا می‌شود.

تصویر ورودی:



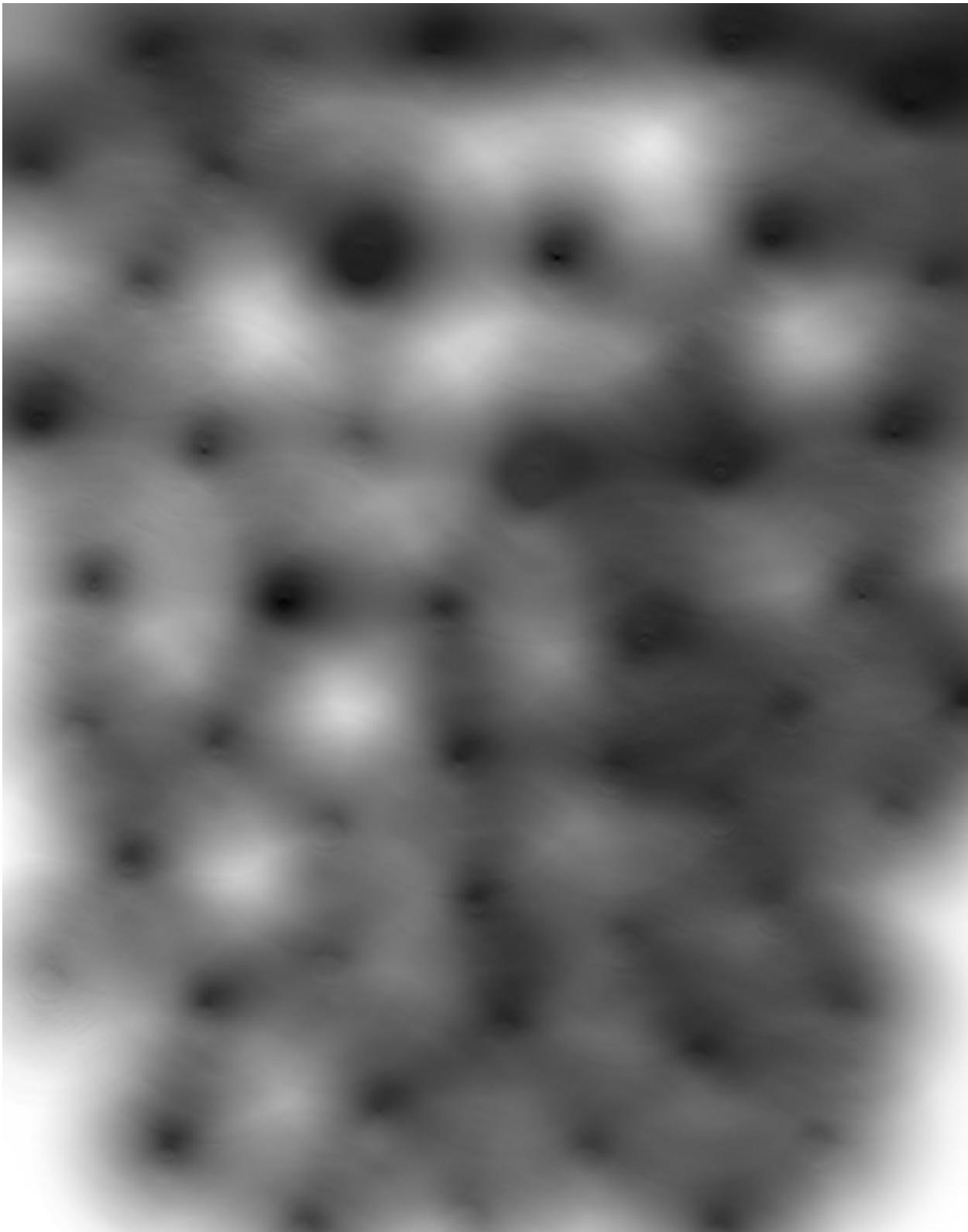
:template تصویر سکه‌ی



تصویر محل سکه‌ی مدنظر که بر روی تصویر پیدا شده است (p11b-location-matched-coin.png):



: (p11b-mse-map.png) MSE_map تصویر



البته این روش کمی هزینه زمانی بالایی دارد ولی از آنجایی که در صورت سوال به هزینه‌ی زمانی اشاره نشده است از همین روش استفاده کردم. در حالی که کارهای دیگری برای افزایش سرعت می‌شد انجام داد، مثلاً پس زمینه را از سکه‌ها جدا کرد و mse سکه‌ی template و همسایگی یک پیکسل را برای نقاط غیر از پس زمینه انجام داد.

قسمت C

کد این بخش از سوال در p11c.m است، کد دقیقاً مانند قسمت قبل از با این تفاوت coin template یک را استفاده می‌کنم. با روش rotate کردن template بر خلاف روش قبل نتوانستم سکه را پیدا کنم زیرا شکل سکه‌دوم با و template با اینکه یکی است ولی در شدت روشنایی متفاوت اند و نور پردازی متفاوتی دارند. به همین دلیل سکه‌ی دیگری به اشتباه تشخیص داده می‌شود.

قسمت d

کدهای این بخش در p11d.m و mse_template_3.m قرار دارد. در mse_template_3.m تابع زیر قرار دارد:

```
] function mse_map = mse_template_3(img, img_temp)
]     % m_1 = size(img_temp, 1) m_2 = size(img_temp, 2)
]         % This function divide rgb image to m*m overlapping blocks
-     % then it calculates MSE error of each blocks and template.
|
```

این تابع دقیقاً مانند تابع mse_template است که در بخش یک استفاده کردم، template بر روی تمام پیکسل‌ها حرکت می‌کند، وقتی template روی یک پیکسل قرار می‌گیرد mse پیکسل و همسایگی‌اش را با حساب می‌کنیم و تنها تفاوتی که این تابع با تابع بخش a دارد این است که اینجا تصویر ورودی و template به صورت RGB است.

در کد p11d.m تصویر را می‌خوانم، سپس تابع mse_template_3 را فرامی‌خوانم که این تابع یک متريس دو بعدی هم سایز تصویر بازی ماریو می‌دهم که مقدار هر درایه مشخص کننده MSE آن پیکسل و همسایگی‌اش با template است، بعد از آن کمترین MSE را بین پیکسل‌ها پیدا می‌کنم، سپس تمام مقدارهای MSE را صعودی مرتب می‌کنم و از اول آرایه MSE‌های مرتب شده یکی یکی MSE‌ها را جدا می‌کنم و این کار را تا زمانی انجام می‌دهم که جهشی بین دو MSE پشت سر هم مشاهده کنم که نشان می‌دهد پیکسل بعد نشان دهنده‌ی یک سکه نیست، این گونه سکه‌ها را می‌شمارم. بعد از آن محل سکه‌ها را روی تصویر مشخص می‌کنم و در انتهای MSE_map را نمایش می‌دهم. در زیر خروجی‌های کد را مشاهده می‌کنید:

تصویر ورودی:



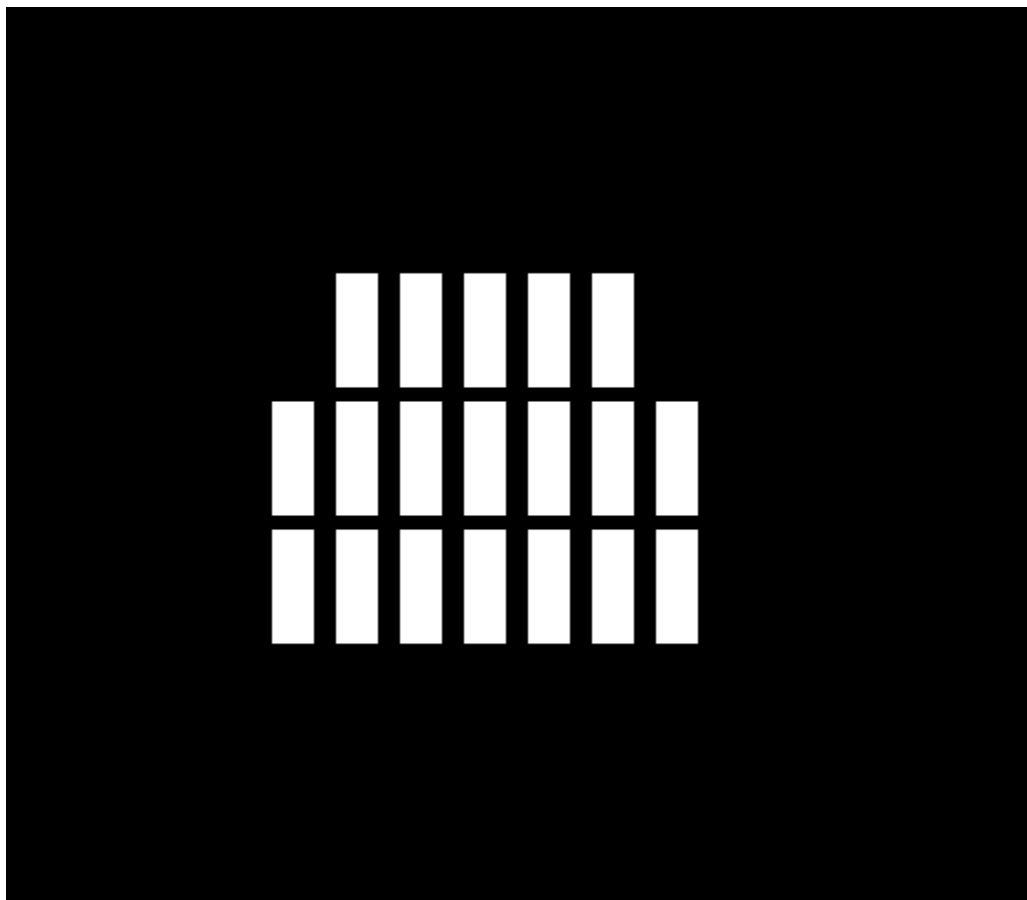
: Template تصویر



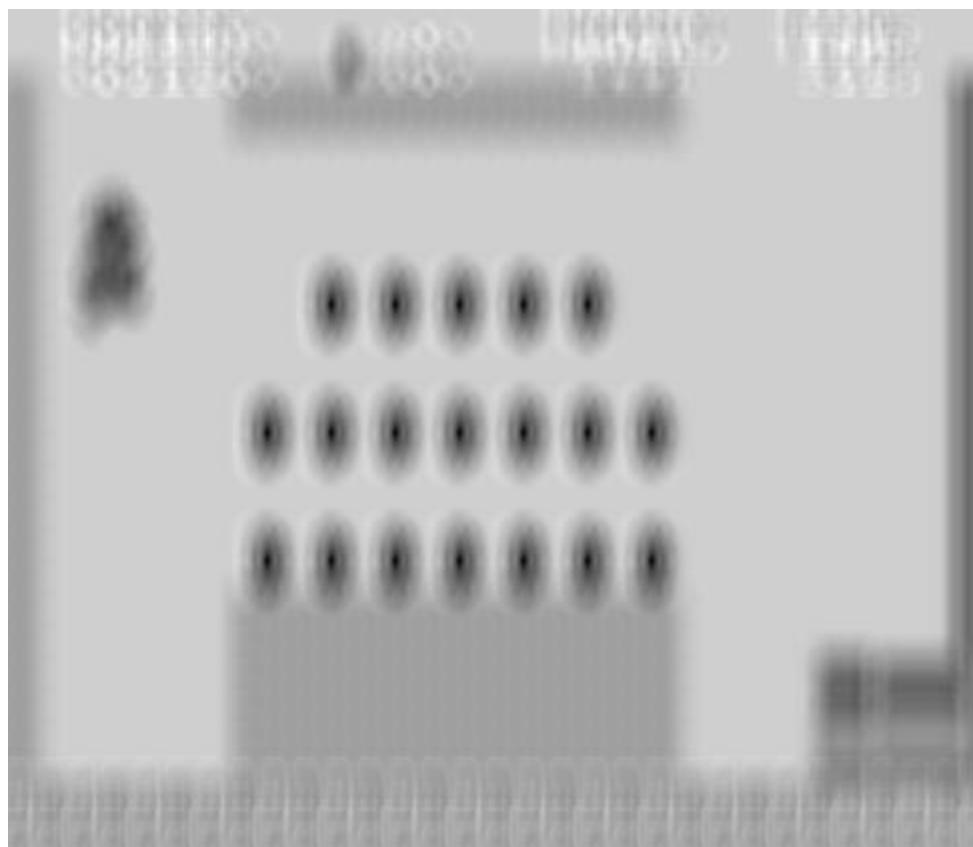
تعداد سکه‌ها:

```
>> p11d  
Number of coins: 19  
>>
```

: (p11d-position-coins.png) محل سکه‌ها

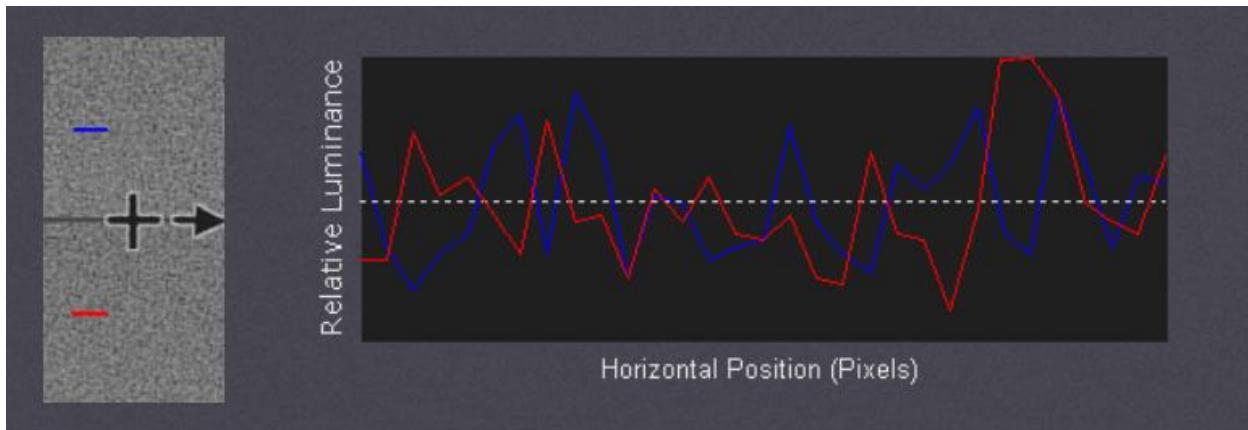


: MSE MAP

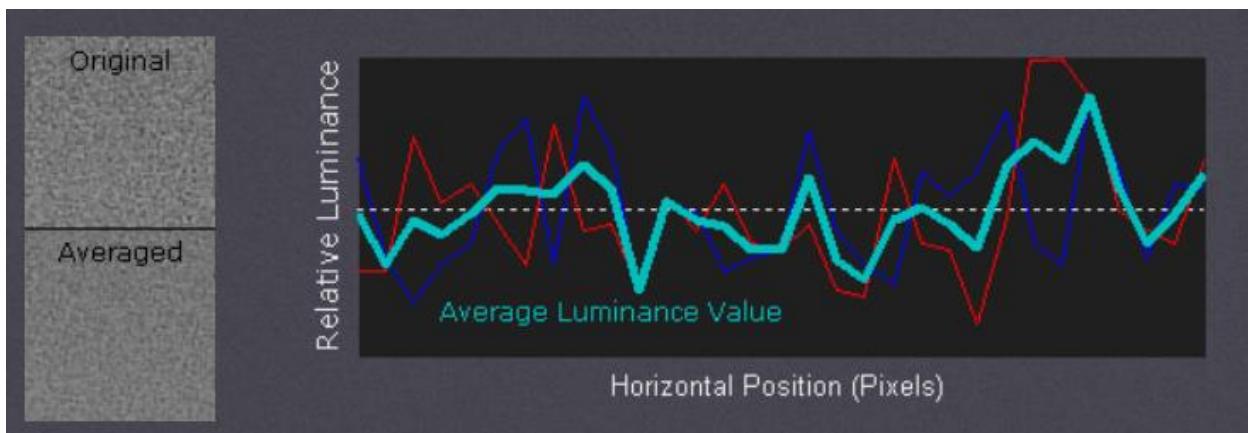


قسمت a

تحت این فرض انجام می‌شود که نویز رندم بوده و به همین دلیل با میانگین گیری از تصویرهای بیشتر به مرور نوسان شدت روشنایی هر پیسکل در تصویرهای مختلف که کمتر و بیشتر از مقدار واقعی هستند خنثاً می‌شوند.



نمودار سمت راست شدت تغییرات روشنایی در طول دو خط آبی و قرمز سمت چپ را نشان می‌دهد که با میانگین گیری به نتیجه‌ی زیر می‌رسیم:



همین طور که مشاهده می‌شود با میانگین گیری نوسان‌های بیشتر و کمتر از میزان واقعی هم را خنثاً می‌کنند.

قسمت b

انجام دوبار histogram equalization برابر با انجام یک بار histogram equalization است، زیرا histogram equalization شدت روش‌نایی‌های تصویر را به گونه‌ای تغییر می‌دهد که هیستوگرام تصویر جدید برابر با یک توزیع یکنواخت باشد ولی از آنجایی که تصویر گسسته است هیستوگرام تاجای ممکن شبیه به توزیع یکنواخت می‌شود و انجام دوباره‌ی آن نمی‌تواند histogram را از آن به توزیع یکنواخت نزدیک‌تر بکند. اگر طبق فرمول‌ها محاسبه‌ی histogram equalization جلو برویم باز به همان اعداد می‌رسیم.

قسمت c

اگر از spatial filters برای بلور کردن استفاده کنیم باید جمع المنت‌های فیلتر یک شود. اگر از spatial filters برای کارهایی همچون high pass filter استفاده کنیم باید مجموع المنت‌های فیلتر صفر شود. پس الزاماً در یک شدن المنت‌های فیلتر نیست.

اگر مجموع برابر با یک باشد DC حفظ می‌شود اگر برابر با صفر باشد DC صفر می‌شود.

قسمت d

در صورتی که از adaptive histogram equalization به گونه‌ای باشد که تصویر را به بلاک‌هایی تقسیم کنیم به طوری که آن بلاک‌ها هم پوشانی نداشته باشند و سپس histogram equalization را بر روی هر کدام از آن‌ها اعمال کنیم تصویر بلاک بلاک می‌شود و بین بلاک‌های مختلف تفاوت دیده می‌شود و بلاک‌ها قابل تشخیص اند، می‌توان adaptive histogram equalization را اینگونه انجام داد که برای هر پیکسل همسایگی $m * m$ آن را در نظر گرفت و histogram equalization را بر روی همسایگی پیکسل انجام داد و سپس مقدار پیکسل را آپدیت کرد. در روش قبل بلاک‌ها هم پوشانی نداشتند ولی در این روش بلاک‌ها هم پوشانی دارند.

هر دو روش را برای تمرین‌ها مربوط انجام دادم که روش histogram equalization به روش بلاک‌های هم پوشان بسیار بهتر عمل می‌کرد.

قسمت e

در histogram matching هدف تغییر شدت روشنایی‌ها به گونه‌ای است که هیستوگرام تصویر شبیه به هیستوگرامی شود که ما مشخص می‌کنیم که هیستوگرام هدف می‌تواند هر شکلی داشته باشد، اگر هیستوگرام histogram هدف را به گونه‌ای انتخاب کنیم که توزیع شدت روشنایی‌ها یک توزیع یکنواخت باشد، matching شدت روشنایی‌های تصویر را به گونه‌ای تغییر می‌دهد که شدت روشنایی‌های تصویر جدید توزیعی شبیه به توزیع یکنواخت داشته باشند که این کار همان histogram equalization است.

قسمت f

کاربردهای مختلفی دارد در طراحی و گرافیک کامپیوتر و ... دارد. بعضی از کاربردهای آن عبارت‌اند:

- استفاده از Complementary color در طراحی لوگو، وسایل، دکور، وسایل امداد و ... برای جذب حداکثری توجه افراد.
- استفاده از این روش در فیلم‌های سه بعدی که با کمک عینک‌هایی با دو شیشه با رنگ‌های متفاوت حس سه بعدی بودن را شبیه سازی می‌کنند.

