دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

پردازش تصاویر دیجیتال

گزارش تمرین سری پنجم-فصل ۶

دانشجو

سید محمد جواد موسوی

استاد کلاس

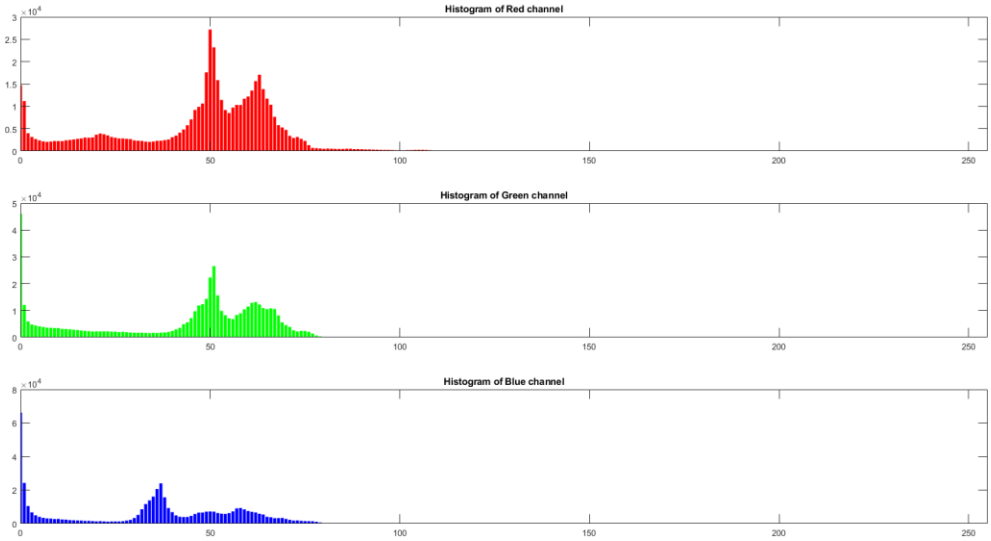
دکتر حمید سلطانیان‌زاده

سوال اول

ابتدا با کد زیر تصویر بارگزاری شده و بررسی می شود که اگر به صورت RGB نیست با دستور cat سه زیر تصویر جدا شود:

```
%% Part a
% Load the image
img = imread("images\flowers.png");

% Ensure the image is in RGB format
if size(img, 3) ~= 3
    img = cat(3, img, img, img)
end

% Display image
figure(1);
imshow(img);
title("Original RGB Image");
```

در مرحله بعد با دستور imhist هستوگرام شدت روشنایی سه زیر تصویر بدست آمد:



مشاهده می شود که شدت روشنایی در بازه باریکی قرار دارد و همه بازه ۰ تا ۲۵۵ را نپوشیده است.

بخش سوم و چهارم و پنجم

در این بخش ابتدا با دستور histeq تعدیل هیستوگرام به صورت جداگانه بر روی هر کانال اعمال شده و نتایج به نمایش گذاشته شده است:

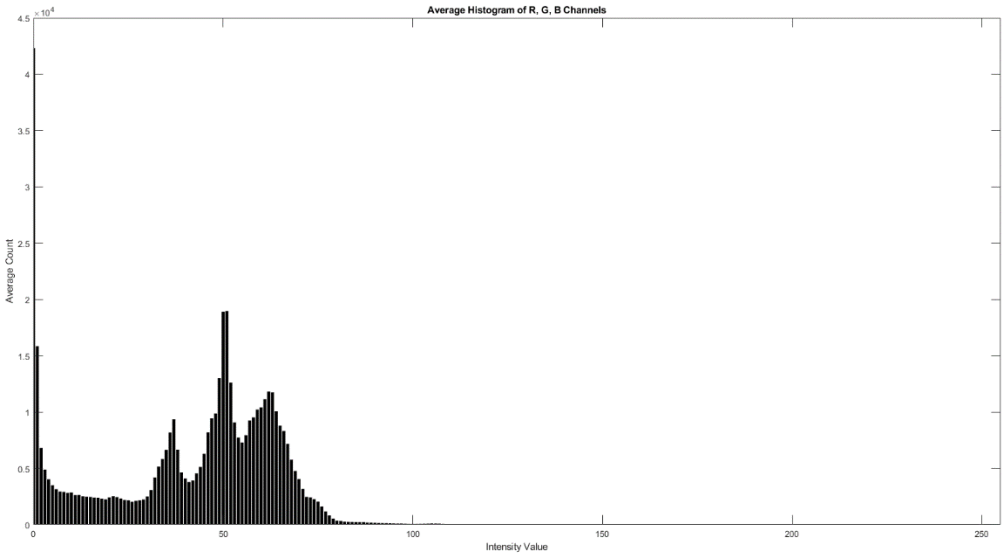
```
%% Part c and d and e
% Apply histogram equalization to each channel
R_eq = histeq(R);
G_eq = histeq(G);
B_eq = histeq(B);

% Combine the equalized channels back into an RGB image
img_eq = cat(3, R_eq, G_eq, B_eq);

% Display the equalized image
figure(٣);

subplot(١,٢,١);
imshow(img);
title("Original RGB Image");

subplot(١,٢,٢);
imshow(img_eq);
title('Histogram Equalized RGB Image (Channel-wise)');

```

بخش ششم

در این بخش میانگین هستوگرام سه زیر تصویری که در بخش دوم محاسبه شدند گرفته شده است:

```
%% Part f
% % Compute histograms of the equalized channels
% [countR_eq, ~] = imhist(R_eq);
% [countG_eq, ~] = imhist(G_eq);
% [countB_eq, ~] = imhist(B_eq);

% Compute the avrage histogram
avg_hist = (double(countR) + double(countG) + double(countB)) / 3;

% Plot the average histogram
```

```
figure(٤);

bar(٠:٢٥٥, avg_hist, 'k');
title('Average Histogram of R, G, B Channels');
xlim([٠ ٢٥٥]);
xlabel('Intensity Value');
ylabel('Average Count');

```

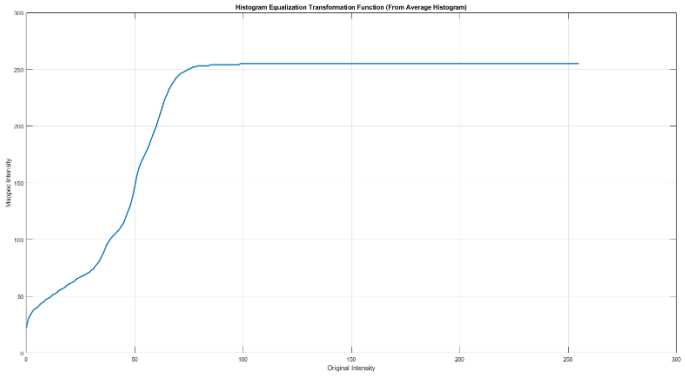


بخش هفت

در این بخش عمل تعدیل هیستوگرام با استفاده از تابع cdf محاسبه شده است:

```
%% Part G
% Normalize the average histogram to get a probability distribution
avg_hist_norm = avg_hist / sum(avg_hist);

% Compute the cumulative distribution function (CDF)
cdf_avg = cumsum(avg_hist_norm);

% Create the intensity transformation function
T_avg = uint8(round(255 * cdf_avg));  % Scale CDF to [0, 255] and convert to uint8

% Plot the transformation function
figure(٥);
plot(٠:٢٥٥, T_avg, 'LineWidth', ٢);
title('Histogram Equalization Transformation Function (From Average Histogram)');
xlabel('Original Intensity');
ylabel('Mapped Intensity');
grid on;

```

تابع تبدیل محاسبه شده به قرار زیر است:



در نهایت بعد از اعمال این تابع تبدیل داریم:

```
%% Part H and I
% Apply the average transformation function to each RGB channel
R_trans = T_avg(double(R) + 1);  % +1 because MATLAB indexing starts from 1
G_trans = T_avg(double(G) + 1);
B_trans = T_avg(double(B) + 1);

% Combine the transformed channels into one RGB image
img_trans = cat(3, R_trans, G_trans, B_trans);

% Display the transformed image
figure(٦);
imshow(img_trans);
title('RGB Image After Applying Average Histogram Equalization Transformation');
```

سوال دوم

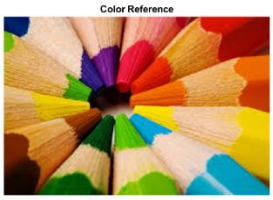
بخش اول

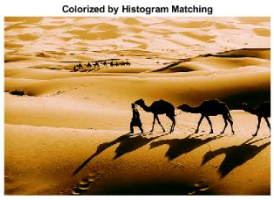
تطبیق هیستوگرام (Histogram Matching) روشی است که می‌توان با استفاده از آن به‌صورت مصنوعی به تصاویر خاکستری رنگ اضافه کرد. در این روش، یک تصویر رنگی به عنوان مرجع انتخاب می‌شود. این تصویر دارای سه کانال رنگی (قرمز، سبز، آبی) است که هرکدام هیستوگرام خاص خود را دارند. از طرفی، تصویر خاکستری تنها یک کانال شدت روشنایی دارد و بنابراین فقط یک هیستوگرام دارد. برای رنگی کردن تصویر خاکستری، ابتدا تصویر خاکستری را به سه نسخه‌ی مشابه تقسیم می‌کنیم تا سه کانال اولیه برای ساخت تصویر رنگی به دست آوریم. سپس برای هر کانال R ، G، B، هیستوگرام تصویر خاکستری را طوری تغییر می‌دهیم که با هیستوگرام کانال متناظر در تصویر مرجع تطابق پیدا کند. به این ترتیب، هر کانال خروجی دارای توزیع آماری مشابه با کانال متناظر در تصویر مرجع خواهد بود. در نهایت، این سه کانال را با هم ترکیب می‌کنیم تا تصویر رنگی نهایی ساخته شود.

بخش دوم

در این بخش ابتدا دو تصویر خوانده شدنه اند. سپس با توجه به اینکه تصویر خاکستری به صورت RGB بود ابتدا با دستور rgb۲gray به خاکستری تبدیل شد. سپس ابعاد تصویر رفرنس نیز با ابعاد تصویر خاکستری با دستور imreasize تغییر یافته است. سپس با دستور imhistmatch هر سه کانال با تصویر خاکسزی تطبیق داده شده اند:

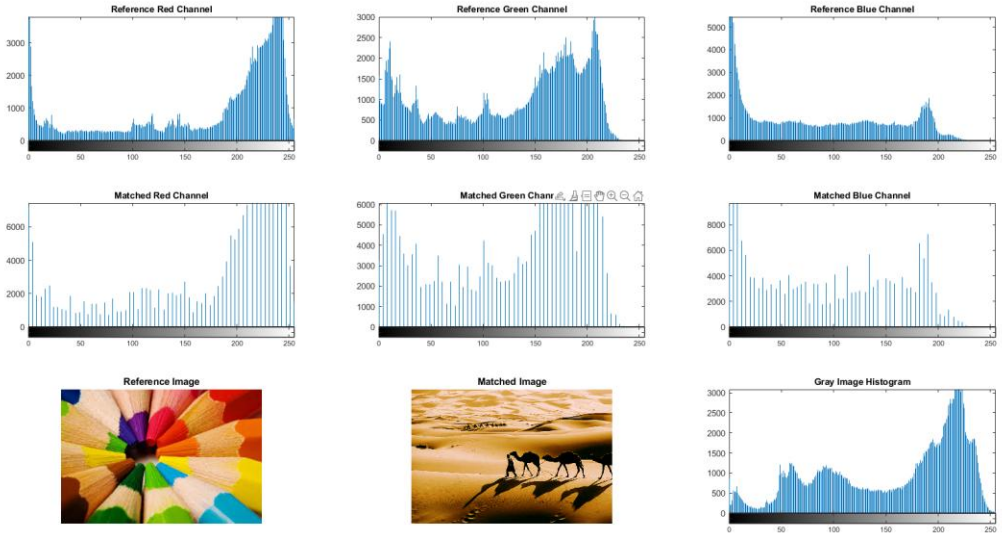
```
%% Part b
% Load images
gray_img = imread("images\dessert.jpg");
ref_img = imread("images\pencils.jpeg");

% Convert gray image(RGB) to grayscale
if size(gray_img, 3) == 3
    gray_img = rgb2gray(gray_img);
end

% Resize reference image(۱۷۷*۲۸٤*۳) to match grayscale image size(٤۰۸*٦۱۲*۳)
ref_img = imresize(ref_img, size(gray_img));

% Initialize output image
matched_img = zeros([size(gray_img), ۳], 'uint۸');

% Apply histogram matching channel-by-channel
for channel = ۱:۳
    ref_channel = ref_img(:,:,channel);
    matched_img(:,:,channel) = imhistmatch(gray_img, ref_channel);
end
```





بخش سوم

تصویری که ارائه شده است به‌خوبی نشان می‌دهد که فرآیند همسان‌سازی هیستوگرام تا حد زیادی با آنچه در بخش (a) انتظار می‌رفت مطابقت دارد. با دقت در هیستوگرام‌ها می‌توان روند عملکرد این روش را بهتر درک کرد. در تصویر خاکستری اولیه، همان‌طور که انتظار می‌رود، تنها یک هیستوگرام برای شدت روشنایی وجود دارد. این هیستوگرام معمولاً در ناحیه‌های میانی روشنایی متمرکز است و اطلاعاتی درباره رنگ ندارد. در مقابل، تصویر مرجع رنگی (مانند تصویر مدادهای رنگی) دارای سه هیستوگرام جداگانه برای کانال‌های R ، G و B است که به‌خوبی تنوع رنگی تصویر را نشان می‌دهند. در مرحله تطبیق، تصویر خاکستری به سه نسخه مشابه تقسیم شده و برای هر نسخه، هیستوگرام آن با یکی از کانال‌های R ، G یا B مرجع تطبیق داده شده است. نتیجه‌ی این فرآیند در هیستوگرام‌های تصویر نهایی مشخص است: کانال‌های رنگی R ، G و B در تصویر حاصل، توزیع‌هایی شبیه به تصویر مرجع پیدا کرده‌اند. اگرچه تفاوت‌هایی طبیعی وجود دارد (به‌دلیل تفاوت ساختار تصویری)، اما به‌وضوح می‌توان دید که رنگ و تُن کلی تصویر نهایی به‌شدت تحت تأثیر هیستوگرام تصویر مرجع قرار گرفته است.



سوال ۳

بخش اول و دوم و سوم

در ابتدا با دستور زیر تصویر بارگزاری شده است:

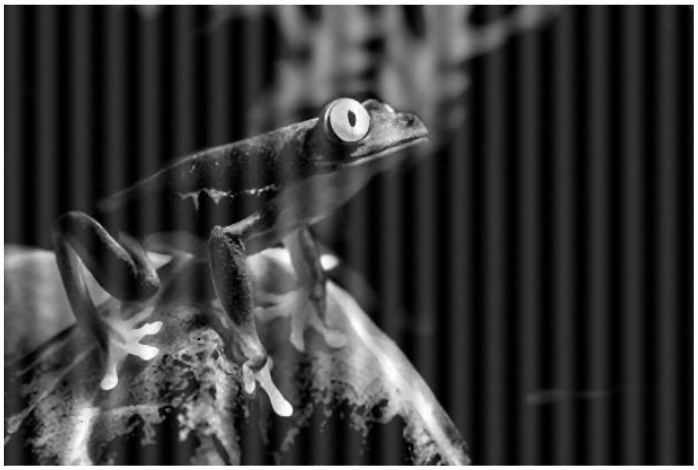
```
١. %% Part a
٢. % Load the image
٣. img = imread("images\frog_noisy.png");
٤.
٥. R = img(:,:,١);
٦. G = img(:,:,٢);
٧. B = img(:,:,٣);
```

تصـویر دارای نویز نمک فلفلی اسـت. علاوه بر این یک سـری خطور سـبز افقی خطوط قرمز عمودی در تصـویر دیده می شـود. برای حذف نویز نمک فلفلی یک روش کاراد اسـتفاده از فیلتر مدین اسـت. به همین منظور ابتدا تصاویر RGB را استخراج کرده و به طور جداگانه بر روی آنها فیلتر میانه زده شده است:

```
1. % Remove salt and pepper noise using median filtering
2. R_med = medfilt2(R, [3,3]);
3. G_med = medfilt2(G, [3,3]);
4. B_med = medfilt2(B, [3,3]);
5.
6. % Optional: show filtered channels
7. figure, imshow(R_med), title('Red Channel after Median Filter');
8. figure, imshow(G_med), title('Green Channel after Median Filter');
9. figure, imshow(B_med), title('Blue Channel after Median Filter');
```

دسـتور medfilt۲ دو ورودی گرفته اسـت. ورودی اول تصـویر و ورودی دوم اندازه کرنل اسـت که در اینجا ۳ در ۳ تنظیم شده است.

Red Channel after Median Filter



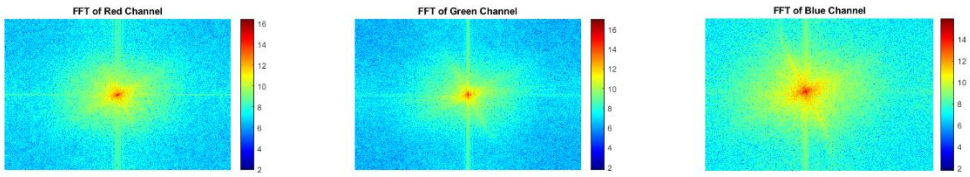
Green Channel after Median Filter

Blue Channel after Median Filter

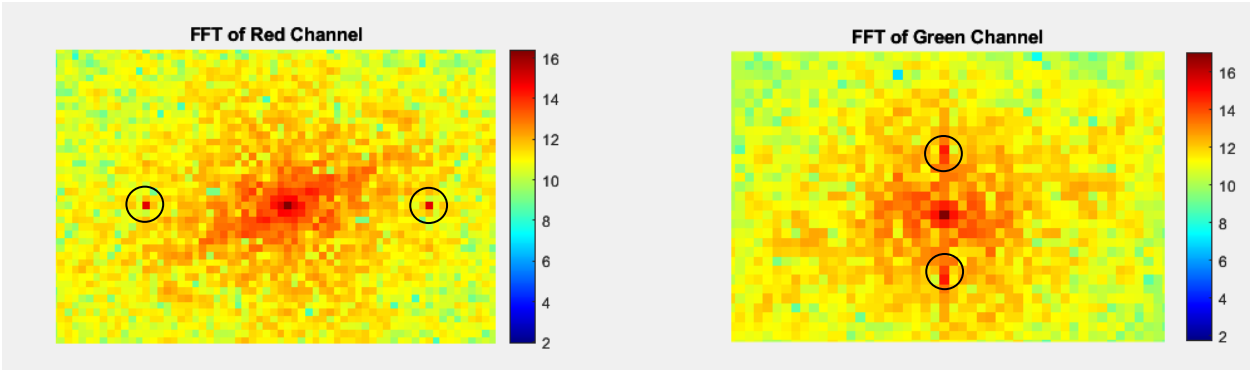
واضح است که نویز های متناوبی در تصویر قرمز و سبز وجود دارد. به همین منظور برای حذف این نویزها به تحلیل فوریه این تصاویر پرداخته شده است.

```
% Compute the 2D Fourier Transform
F_R = fft2(double(R_med));
F_G = fft2(double(G_med));
F_B = fft2(double(B_med));

% Shift the zero-frequency component to the center of the spectrum
F_R_shift = fftshift(F_R);
F_G_shift = fftshift(F_G);
F_B_shift = fftshift(F_B);

% Compute the magnitude spectrum (use log for better visualization)
spectrum_R = log(1 + abs(F_R_shift));
spectrum_G = log(1 + abs(F_G_shift));
spectrum_B = log(1 + abs(F_B_shift));
```

ابتدا با دستور fft۲ تبدیل فوریه دو بعدی تصویر گرفته شده است. دلیل استفاده از double این است که تصاویر در ابتدا مقادیر شدت روشنایی صفر تا ۲۵۵ را دارا هستند ولی ما در حوزه فوریه اعشار هم برایمان مهم است به همین منظور شدت روشنایی ها را به صورت اعشاری به حوزه فوریه میبریم. سپس با دستور fftshift فرکانس صفر به مرکز تصویر منتقل می شود. در نهایت به صورت لگاریتمی طیف فوریه را به نمایش می گذاریم:



برای اینکه مختصات نقاطی که در تصویر سبز و قرمز باعث ایجاد نویز متنفاوب شده اند را پیدا کنیم لازم اس تا به اندازه کافی تصویر تبدیل فوریه را زوم کنیم که داریم:



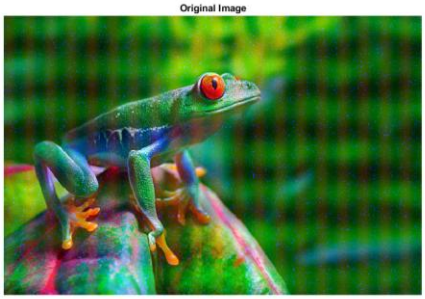
تاثیر نویز بر تبدیل فوریه به صورت نقاطی با مختصات متقارن و با شدت بالا مشخص شده است(دایره های کوچک). بنابراین باید این نقاط را فیلتر کنیم. مختصات نقاط به قرار زیر است:

```
% Apply notch filter
notch_coords_red = [286 204; 326 204];
notch_coords_green = [306 198; 306 210];
```

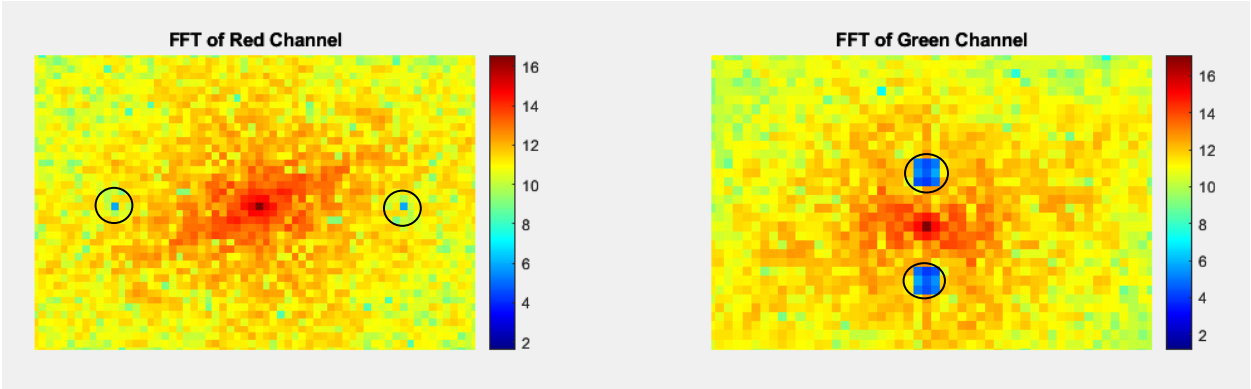
از تابع زیر برای پیاده سازی دستی فیلتر ناچ استفاده شده است:

```
function denoised_img = notch_filter_fft(input_img, notch_coords, notch_size)
    % Convert image to double for processing
    img_d = double(input_img);

    % Get image size
    [M, N] = size(img_d);

    % Compute the 2D FFT and shift the zero-frequency to the center
    F = fft2(img_d);
    F_shift = fftshift(F);

    % Create a notch mask (initialize to 1 = pass all frequencies)
    mask = ones(M, N);

    % Loop through all user-defined notch coordinates
    for i = 1:size(notch_coords, 1)
        cx = notch_coords(i,1);  % x (column) coordinate
        cy = notch_coords(i,2);  % y (row) coordinate

        % Zero out a square area around each notch coordinate
        mask(max(1,cy-notch_size):min(M,cy+notch_size), ...
             max(1,cx-notch_size):min(N,cx+notch_size)) = 0;
    end

    % Apply the mask to the shifted FFT
    F_filtered = F_shift .* mask;

    % Inverse FFT to get the filtered image
    F_ishift = ifftshift(F_filtered);
    img_filtered = real(ifft2(F_ishift));

    % Normalize and convert back to uint8
    img_filtered = mat2gray(img_filtered);  % Normalize to [0,1]
    denoised_img = im2uint8(img_filtered);  % Convert to 8-bit image
end
```

بعد از اعمال این فیلتر بر روی تصویر نتایج به قرار زیر است:



مشاهده می شود که نویز ها به خوبی حذف شده اند و دیگر نویز متنفاوبی در تصویر وجود ندارد. برای اطمینان مختصات تبدیل فوریه دو تصویر سبز و قرمز را بعد از اعمال فیلتر دوباره بررسی می کنیم:



مشاهده می شود که نقاط خیلی خوب فیلتر شده و نویز به طور کامل حذف شده است.

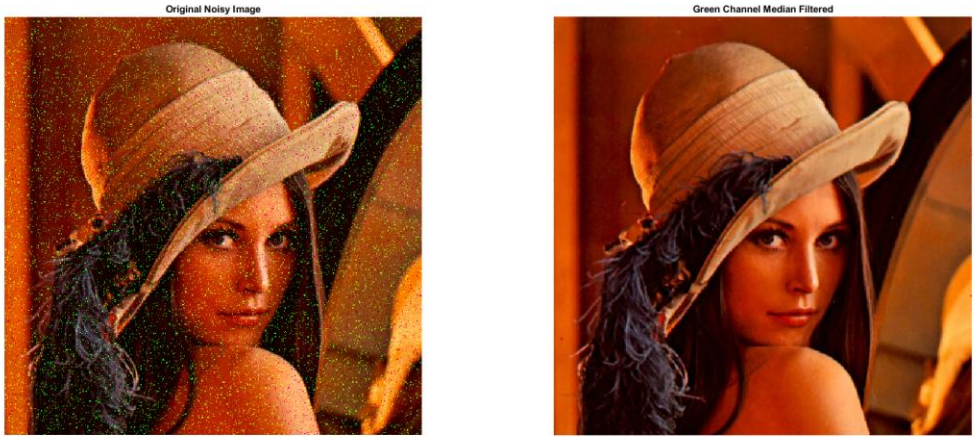
بخش چهارم و پنجم

در تصویر lenna[۱] مشاهده می شود که نویز نمک فلفلی به رنگ سبز اضافه شده است. بنابراین کافیست تا فیلتر میانه را فقط برای زیر تصویر سبز اعمال کرده که کد استفاده شده در زیر آورده شده است. سایز کرنل نیز برابر با ۳ در ۳ تنظیم شده است:

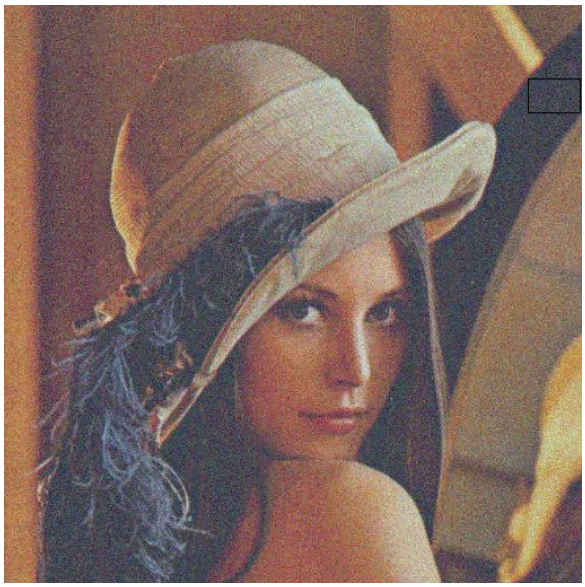
```
1. % load the images
2. lena1 = imread("images\lenna_noisy1.tif");
3. lena2 = imread("images\lenna_noisy2.tif");
4.
5. % Extract the green channel of lenna 1
6. green_channel_1 = lena1(:,:,2);
7.
8. % Apply median filtering
9. green_med_1 = medfilt2(green_channel_1, [3,3]);
۱۰.
```

```
% Replace the green channel in the original image with the filtered one
lena1_filtered = lena1;
lena1_filtered(:,:,2) = green_med_1;

% Display original and filtered image side by side
figure(1);
subplot(1,2,1); imshow(lena1); title('Original Noisy Image');
subplot(1,2,2); imshow(lena1_filtered); title('Green Channel Median Filtered');
```

نتیجه به قرار زیراست:



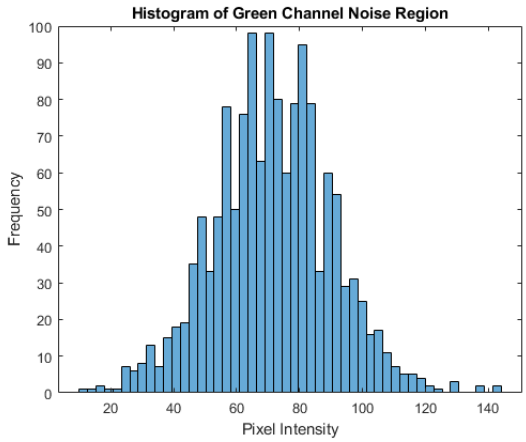
برای تصویر lena۲ باید توزیع نویز را تخمین بزنیم. مشاهده می شود که نویز به صورت نقاط سبز رنگ هستند بنابراین با زیر تصویر سبز کار را ادامه می دهیم. منطقه ای مربعی شکل در نوار مشکی داخل تصویر که در زیر نیز نمایش داده شده است را در در نظر گرفته و میانگین و واریانس و توزیع شدت این ناحیه را بررسی می کنیم. با توجه به اینکه ناحیه تقریبا دارای شدت روشنایی یکنواخت باید باشد توزیع بدست آمده را می توان به توزیع نویز به همراه توزیع شدت نسبت داد.



مختصات ناحیه انتخاب شده به قرار زیر است:

```
% Define the rectangular region (you should set these coordinates)
x1 = 464;   % Starting column
y1 = 65;  % Starting row
x2 = 509;  % Ending column
y2 = 95;  % Ending row
```

توزیع شدت در ناحیه انتخاب شده به قرار زیر است:

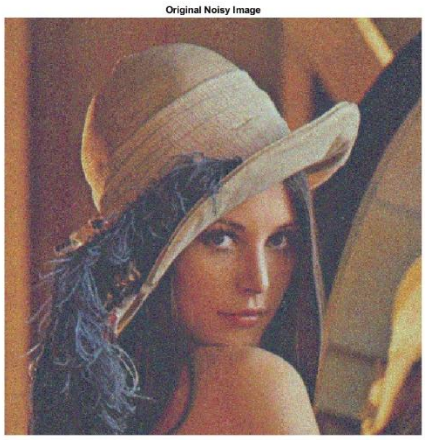


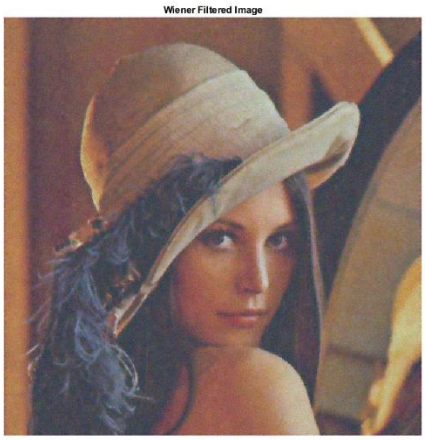
پر واضح است که توزیع نویز به صورت گوسی با واریانس حدودی زیر است:

```
Variance of selected region: 350.50
```

برای حذف نویز گوسی راهکاری که استفاده شده استفاده از فیلتر وینر با کرنل ۷ در ۷ بر روی تصویر سبز است:

```
% Apply Wiener filter
green_wiener_2 = wiener2(green_Channe2, [7 7]);
```





واضح است که نویز به طور خیلی خوبی حذف شده و کیفیت تصویر ارتقا یافته است.

سوال ۴

بخش اول

برای پیاده سازی تابعی که از فضای RGB به HSI منتقل شویم از تابع زیر استفاده شده که روابز محاسبه در زیر آورده شده است:

$$H = \begin{cases} \theta & \text{if } B \le G \\ 360 - \theta & \text{if } B > G \end{cases}$$

$$\theta = \cos^{-1}\left\{\frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-B)(G-B)]^{1/2}}\right\}$$

$$S = 1 - \frac{3}{(R+G+B)}[\min(R,G,B)]$$

$$I = \frac{1}{3}(R+G+B)$$

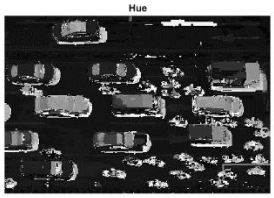
```
1. function [H, S, I] = rgb2hsi_custom(img_rgb)
2. % Convert to double and normalize to [0,1]
3. img = double(img_rgb);
4.
5. R = img(:,:,1) / 255;
6. G = img(:,:,2) / 255;
7. B = img(:,:,3) / 255;
8.
9. % Intensity
١٠. I = (R + G + B) / ٣;
```

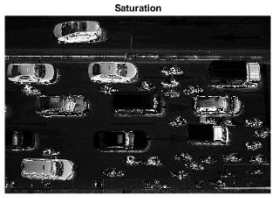
```
١١.
١٢. % Saturation
١٣. min_rgb = min(min(R, G), B);
١٤. sum_rgb = R + G + B;
١٥. S = ١ - (٣ .* min_rgb ./ (sum_rgb + eps));
١٦.
١٧. % Hue calculation
١٨. num = ٠,٥ .* ((R - G) + (R - B)) ;
١٩. den = sqrt((R - G).^٢ + (R - B).*(G - B)) + eps;
٢٠. theta = acos(num ./ den);
٢١.
٢٢. H = theta; % Default when B <= G
٢٣. H(B > G) = ٢*pi - H(B > G);
٢٤.
٢٥. H = H / (٢ * pi); % Normalize to [٠,١]
٢٦.
٢٧. % Clip values just in case (due to floating point)
٢٨. H = max(٠, min(١, H));
٢٩. S = max(٠, min(١, S));
٣٠. I = max(٠, min(١, I));
٣١.
٣٢. end
٣٣.
```

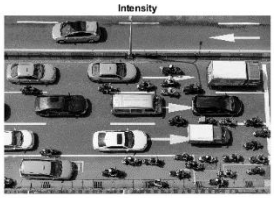
بخش دوم

در این مرحله این تابع بر روی تصویر داده شده اعمال شده است:

```
 1. %% Part b
 2. % Load the image
 3. RGB = imread('images\cars.jpg');
 4.
 5. % Convert RGB to HSI
 6. [H, S, I] = rgb2hsi_custom(RGB);
 7.
 8. % Display results
 9. figure(1);
١٠. subplot(١, ٣, ١);
١١. imshow(H);
١٢. title('Hue');
١٣.
١٤. subplot(١, ٣, ٢);
١٥. imshow(S);
١٦. title('Saturation');
١٧.
١٨. subplot(١, ٣, ٣);
١٩. imshow(I);
٢٠. title('Intensity');
٢١.
```



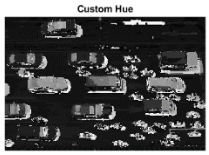


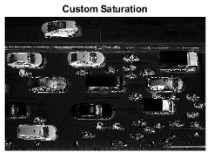


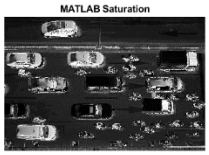
بخش سوم

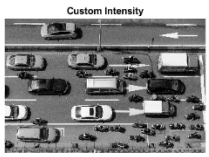
در این بخش با استفاده از rgb۲hsv سه ترم HSI محاسبه شده است.

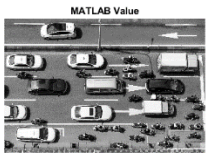
```
%% Par c
% MATLAB's HSV conversion
HSV = rgb2hsv(im2double(RGB));
H_matlab = HSV(:,:,1);
S_matlab = HSV(:,:,2);
I_matlab = HSV(:,:,3);

% Display comparison
figure(2);
subplot(۳, ۲, ۱); imshow(H); title('Custom Hue');
subplot(۳, ۲, ۲); imshow(H_matlab); title('MATLAB Hue');

subplot(۳, ۲, ۳); imshow(S); title('Custom Saturation');
subplot(۳, ۲, ٤); imshow(S_matlab); title('MATLAB Saturation');

subplot(۳, ۲, ٥); imshow(I); title('Custom Intensity');
subplot(۳, ۲, ٦); imshow(I_matlab); title('MATLAB Value');

```











در هر دو فضای رنگی HSI وHSV ، hue با استفاده از زاویه تعریف می‌شود و تفاوت چندانی ندارند، اما ممکن است به دلیل تفاوت‌های عددی جزئی یا روش‌های محاسباتی، مقدار آن‌ها کمی متفاوت باشد. saturation در این دو مدل مفهومی متفاوت دارد. در فضای رنگیhsi ، اشباع نشان‌دهنده‌ی فاصله‌ی رنگ از خاکستری است. یعنی هرچه یک رنگ کم‌رنگ‌تر یا به خاکستری نزدیک‌تر باشد، مقدار اشباع آن کمتر است. فرمول آن به صورت

S = ۱ - (min / sum) تعریف می‌شود که در آن min کمترین مقدار بینR ، GوB ، و sum مجموع آن‌هاست. در حالی که درhsv ، اشباع نشان می‌دهد که رنگ تا چه حد از روشنایی فاصله دارد، و با فرمول / (max - min) max محاسبه می‌شود. در اینجا max بیشترین مقدار بین R ، GوB است. در مورد intensity در hsi و value در hsv هم تفاوت مهمی وجود دارد. در فضـای hsi ، شـدت روشـنایی با میانگین مقدار کانال‌های R ، GوB محاسبه می‌شود، بنابراین نتیجه معمولاً نرم‌تر و هموارتر است. در مقابل، درhsv ، مقدار روشنایی همان بیشترین مقدار بینR ، GوB است، که باعث می‌شود تصویر روشن‌تر و درخشان‌تر به نظر برسد.

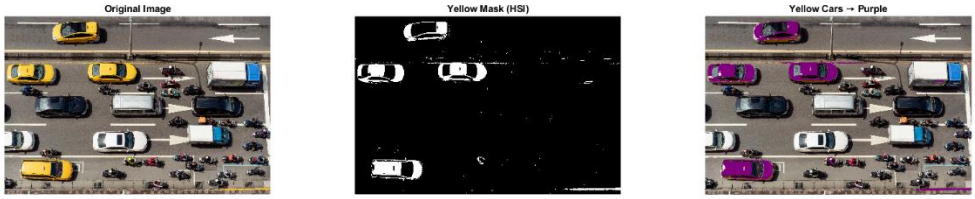
وقتی از HSI اسـتفاده می‌شـود، دیدن روشـنایی‌ها و رنگ‌های کم‌رنگ راحت‌تر اسـت چون این مدل به "درک بصـری انسـان" نزدیک‌تر اسـت. اما HSV برای برخی کارها مثلاً threshold کردن یا تنظیم روشـنایی راحت‌تر استفاده میشود.

بخش چهارم

در این بخش ابتدا تصـویر حاوی خودروها بارگذاری شـده و به فضـای رنگ HSI تبدیل می‌گردد. فضـای HSI به دلیل جداسازی مؤلفه‌های رنگ، اشباع و شدت، برای تحلیل‌های رنگی دقیق‌تر نسبت به فضای RGB مناسب‌تر اسـت. پس از تبدیل تصـویر بهHSI ، یک ماسـک بر اسـاس محدوده مشـخصـی از مؤلفه‌هایHue ، Saturationو Intensityتعریف می‌شود تا نواحی دارای رنگ زرد شناسایی شوند. در اینجا رنگ زرد به‌صورت تقریبی با Hue بین ۰.۱۰ تا ۰.۱۸، اشباع بیشتر از ۰.۲۵ و شدت بیشتر از ۰.۱۵ مشخص شده است. در ادامه، با استفاده از این ماسک، پیکسل‌های شناسایی‌شده از تصویر اصلی استخراج شده و به جای آن‌ها رنگ بنفش با مقادیرR=۱۲۸ ، G=۰، B=۱۲۸جایگزین می‌شود. این تغییر رنگ به‌صورت دستی و پیکسل به پیکسل انجام گرفته است.

```
%% Part d
% Load the image
car_img = imread("images\cars.jpg");

% Convert to HSI using your custom function
[H, S, I] = rgb2hsi_custom(car_img);

% Create mask for yellow (Hue ~ 0.12-0.17, Saturation > 0.4, Intensity > 0.4)
% mask = (H > 0.12 & H < 0.17) & (S > 0.4) & (I > 0.4);
% mask = (H > ۰,۱ & H < ۰,۱۹) & (S > ۰,۲۲) & (I > ۰,۱۸);
mask = (H >= ۰,۱۰ & H <= ۰,۱۸) & (S > ۰,۲۵) & (I > ۰,۱۵);

% Initialize output image
img_purple = car_img;

% Apply purple color (R=۱۲۸, G=۰, B=۱۲۸) where mask = ۱
[m, n] = size(mask);
for i = ۱:m
    for j = ۱:n
        if mask(i,j)
```

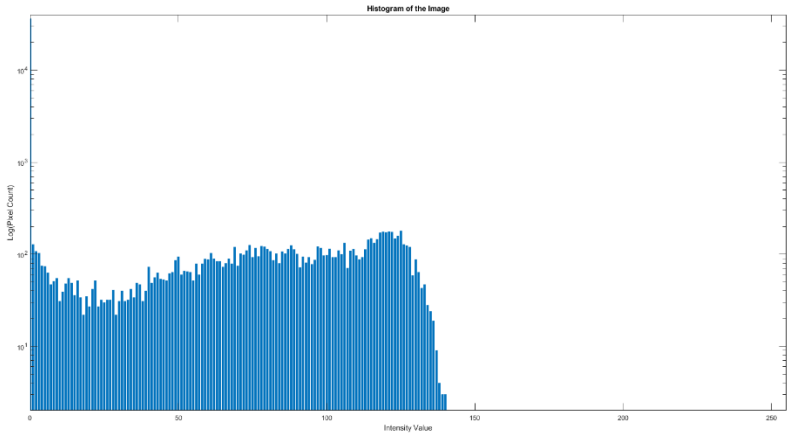
```
            img_purple(i,j,١) = ١٢٨;   % R
            img_purple(i,j,٢) = ٠;     % G
            img_purple(i,j,٣) = ١٢٨;   % B
        end
    end
end

% Display results
figure(٣);
subplot(١,٣,١);
imshow(car_img);
title('Original Image');

subplot(١,٣,٢);
imshow(mask);
title('Yellow Mask (HSI)');

subplot(١,٣,٣);
imshow(img_purple);
title('Yellow Cars → Purple');
```



سوال ۵

بخش اول

در این بخش ابتدا با دستور imread تصویر بارگزاری شد و هستوگرام تصویر به نمایش گذاشته شد. با توجه به اینکه بیشتر شدت روشنایی در سطح شدت صفر قرار داشت محور y به صورت لگاریتمی اسکیل شد تا نمایش هستوگرام بهتر باشد.



بخش دوم

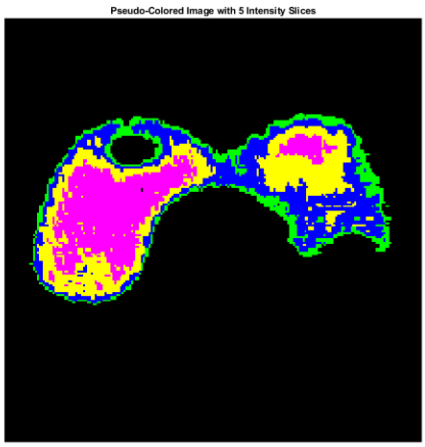
در بخش قبل مشاهده شد که تغییرات شدت روشنایی تصویر از صفر تا ۱۴۲ است. بنابراین در این بخش این بازه به ۵ بخش تقسیم شده است. برای هر بخش شدت روشنایی ها به صورت زیر تقسیم شده است:

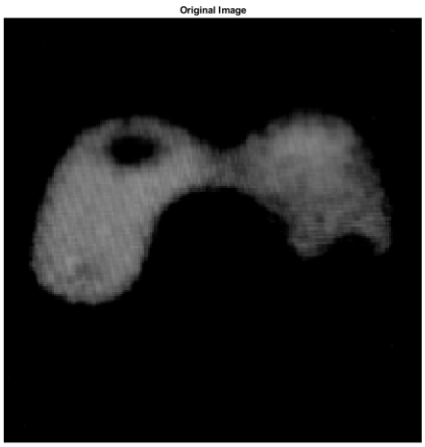
```
% Define 5 arbitrary RGB colors (in [R G B] format, values from 0 to 255)
colors = [
    0,   0,   0;      % Black
      0, 255,   0;    % Green
      0,   0, 255;    % Blue
    255, 255,   0;    % Yellow
    255,   0, 255     % Magenta
];
```

برای مثال شدت روشنایی های بین صفر تا ۲۹ در تصویر اصلی به شد تروشنایی مشکی تبدیل می شوند یا شدت روشنایی بین ۲۹ تا ۵۸ به رنگ سبز تبدیل شده است. بقییه تقسیم بندی به ترتیب مطابق کد بالا است. برای هر بازه شدت روشنایی ییک ماسک طراحی شده که برای مقادیر با شدت روشناییی داخل بازه انتخاب شده مقدار یک و برای مابقی شدتها صفر در نظر می گیرد. سپس برای ماسک در هر مرحله برای بخشهایی که مقدار یک دارد در تصویر اصلی شدت روشنایی هر کانال را مطابق تغییرات شدت روشنایی که در بالا در colors ذکر شد انتخاب می کند.

```
% Convert grayscale image to double for processing
phantom_double = double(phantom_img);

% Get image size
[rows, cols] = size(phantom_img);

% Initialize a color image (RGB)
pseudo_img = zeros(rows, cols, 3, 'uint8');

% Define intensity breakpoints for ٥ intervals (assuming range ٠-٢٥٥)
breaks = linspace(٠, ١٥٠, ٦); % gives ٦ points: [٠ ٥١,٢ ١٠٢,٤ ١٥٣,٦ ٢٠٤,٨ ٢٥٦]

% Assign colors based on intensity slicing
for i = ١:٥
    mask = (phantom_double >= breaks(i)) & (phantom_double < breaks(i+١));
    for c = ١:٣
        channel = pseudo_img(:,:,c);
        channel(mask) = colors(i,c);
        pseudo_img(:,:,c) = channel;
    end
end
```

در نهایت خروجی به قرار زیر است:





سوال ۶

بخش اول

تابع modifyColor تغییر شدت یک رنگ خاص در تصاویر RGB طراحی شده است. این تابع دارای سه ورودی است: تصویرRGB ، نوع رنگ cyan ، magenta، yellow یا black و نوع تغییر (ضعف یا شدت). ابتدا تصویر به نوع داده‌ای double تبدیل می‌شود تا انجام محاسبات عددی دقیق‌تر امکان‌پذیر باشد. سپس کانال‌های R ، G و B از تصویر استخراج می‌گردند. در ادامه، بسته به رنگ انتخاب‌شده، یکی از رنگ‌های مکمل مورد استفاده قرار می‌گیرد. به عنوان مثال، برای رنگ cyan از کانال قرمز استفاده می‌شود، چرا که cyan مکمل قرمز است. برای ایجاد ضعف یا شدت در رنگ، مقدار رنگ مکمل محاسبه و سپس به میزان مشخصی کاهش یا افزایش داده می‌شود. پس از آن، کانال اصلی با استفاده از مقدار جدید رنگ مکمل بازسازی می‌گردد. در مورد رنگ black ، ابتدا کمترین مقدار در میان کانال‌های R ، G و B انتخاب و به عنوان مقدار K در نظر گرفته می‌شود. سپس تغییر مورد نظر بر K اعمال می‌گردد. در ادامه، نسبت K جدید به K اولیه محاسبه شده و این نسبت بر روی تمامی کانال‌ها اعمال می‌شود تا شدت رنگ سیاه در تصویر افزایش یا کاهش یابد. در پایان، کانال‌های اصلاح‌شده با یکدیگر ترکیب و به نوع uint۸ تبدیل می‌شوند تا تصویر نهایی تولید گردد.

```
function output = modifyColor(img, colorType, effect)
    img = double(img);
    R = img(:,:,1);
    G = img(:,:,2);
    B = img(:,:,3);

```

```
    switch lower(colorType)
        case 'cyan'
            C = 255 - R;
            change = ٤٠;
            if strcmp(effect, 'weakness')
                C = max(C - change, ٠);
            else
                C = min(C + change, ٢٥٥);
            end
            R = ٢٥٥ - C;

        case 'magenta'
            M = ٢٥٥ - G;
            change = ٤٠;
            if strcmp(effect, 'weakness')
                M = max(M - change, ٠);
            else
                M = min(M + change, ٢٥٥);
            end
            G = ٢٥٥ - M;

        case 'yellow'
            Y = ٢٥٥ - B;
            change = ٤٠;
            if strcmp(effect, 'weakness')
                Y = max(Y - change, ٠);
            else
                Y = min(Y + change, ٢٥٥);
            end
            B = ٢٥٥ - Y;

        case 'black'
            K = min(min(R, G), B);
            change = ٣٠;
            if strcmp(effect, 'weakness')
                K = max(K - change, ٠);
            else
                K = min(K + change, ٢٥٥);
            end
            % Apply same change to R,G,B
            scale = K ./ (min(min(R, G), B) + ١e-٣);
            R = R .* scale;
            G = G .* scale;
            B = B .* scale;

        otherwise
            error('Invalid color type.');
    end

    % Recombine
    output = uint٨(cat(٣, R, G, B));
end
```

در نهایت با کد زیر نمودار خواسته شده برای ۸ حالت مختلف ترسی شده است:

```
%% Part a
% Load image
peppers_img = imread("images\woman_baby.tif");

colors = {'cyan', 'yellow', 'magenta', 'black'};
effects = {'weakness', 'heaviness'};

for i = 1:4
```

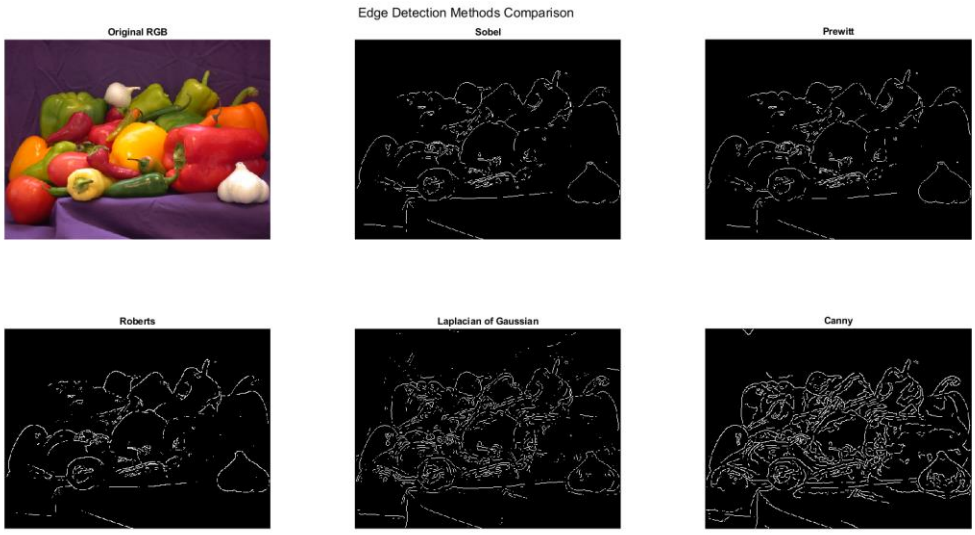
```
    subplot(2,4,i);  % Row 1: Weakness
    imshow(modifyColor(peppers_img, colors{i}, effects{1}));
    title(['Weak ', colors{i}]);

    subplot(2,4,i+4);  % Row 2: Heaviness
    imshow(modifyColor(peppers_img, colors{i}, effects{2}));
    title(['Heavy ', colors{i}]);
end
```

بخش دوم

در این بخش از پروژه، هدف مقایسه عملکرد روش‌های مختلف تشخیص لبه بر روی یک تصویر رنگی است. ابتدا تصویر رنگی peppers.png بارگذاری شده و با استفاده از تابع rgb2gray به تصویر خاکستری تبدیل می‌شود، زیرا بیشتر الگوریتم‌های تشخیص لبه بر روی تصاویر تک‌کاناله اعمال می‌شوند. سپس با استفاده از تابع edge در متلب، پنج روش مختلف برای شناسایی لبه‌های تصویر به کار گرفته می‌شود که شامل روش‌های Sobel ، Prewitt، Roberts، Laplacian of Gaussian (LoG) و Canny است. روش‌های Sobel و Prewitt از فیلترهای مشتق‌گیر برای تشخیص تغییرات شدت پیکسل‌ها استفاده می‌کنند و به تغییرات ناگهانی روشنایی حساس هستند. روش Roberts از فیلترهایی با اندازه کوچکتر استفاده می‌کند و می‌تواند لبه‌های باریک‌تری تولید کند. الگوریتم LoG ابتدا تصویر را با فیلتر گوسین هموار می‌سازد و سپس از مشتق دوم برای یافتن نواحی با تغییر شدید استفاده می‌کند. روش Canny یکی از دقیق‌ترین روش‌های تشخیص لبه است که شامل چند مرحله از جمله کاهش نویز، محاسبه گرادیان، نازک‌سازی لبه‌ها و آستانه‌گذاری دوگانه است. در نهایت، نتایج به همراه

تصویر اصلی در یک پنجره با چینش ۲ سطر و ۳ ستون نمایش داده می‌شوند تا بتوان تفاوت عملکرد هر یک از روش‌ها را به صورت بصری مقایسه کرد.

```
%% Part b
% Load the image
peppers_img = imread('peppers.png');

% Convert RGB to Grayscale
gray_img = rgb2gray(peppers_img);

% Apply edge detection methods
edge_sobel   = edge(gray_img, 'sobel');
edge_prewitt = edge(gray_img, 'prewitt');
edge_roberts = edge(gray_img, 'roberts');
edge_log     = edge(gray_img, 'log');
edge_canny   = edge(gray_img, 'canny');

% Display all results in a ٢x٣ grid
figure(٢);

subplot(٢,٣,١);
imshow(peppers_img);
title('Original RGB');

subplot(٢,٣,٢);
imshow(edge_sobel);
title('Sobel');

subplot(٢,٣,٣);
imshow(edge_prewitt);
title('Prewitt');

subplot(٢,٣,٤);
imshow(edge_roberts);
title('Roberts');

subplot(٢,٣,٥);
imshow(edge_log);
title('Laplacian of Gaussian');

subplot(٢,٣,٦);
imshow(edge_canny);
title('Canny');

sgtitle('Edge Detection Methods Comparison');
```



در روش **Sobel** لبه‌ها به شکل واضحی قابل مشاهده هستند، مخصوصاً در نواحی با تغییرات روشنایی بیشتر. این روش در مقایسه با روش‌های ساده‌تر مانند Roberts عملکرد نسبتاً بهتری در حذف نویز دارد **Prewitt** . عملکردی مشابه Sobel دارد، ولی ممکن است نسبت به نویز کمی حساس‌تر باشد. در هر دو روش، لبه‌های خارجی اجسام به خوبی استخراج شده‌اند. روش **Roberts** از فیلترهای کوچک‌تری استفاده می‌کند و در نتیجه لبه‌های باریک‌تری تولید می‌کند. با این حال، این روش در مقابل نویز حساس‌تر است و در تصویر بالا مشاهده می‌شود که برخی جزئیات ظریف‌تر به وضوح دیده نمی‌شوند. در **Laplacian of Gaussian (LoG)**جزئیات بیشتری از لبه‌ها استخراج شده و لبه‌های داخلی اجسام نیز نمایان‌تر هستند. این روش معمولاً لبه‌ها را به صورت متراکم‌تری نشان می‌دهد، زیرا از مشتق دوم تصویر استفاده می‌کند. در نهایت، **Canny** که یکی از دقیق‌ترین و پرکاربردترین روش‌ها است، هم لبه‌های داخلی و هم لبه‌های خارجی اجسام را با دقت بالا و نویز کمتر شناسایی کرده است. ساختار کلی تصویر با جزئیات بیشتری حفظ شده و این روش بهترین تعادل را میان حساسیت و دقت ارائه می‌دهد.

در مجموع، اگر هدف استخراج لبه‌های واضح با حذف مناسب نویز باشد، روش Canny انتخاب مناسبی است. برای کاربردهای سریع‌تر یا ساده‌تر، روش‌های Sobel و Prewitt عملکرد خوبی دارند Roberts .و LoG نیز بسته به نیاز، در شرایط خاص می‌توانند مفید باشند.

سوال 5) در ترکیب مقادیر RGB با نسبت زیر جمع می‌شوند:

$$\frac{1}{2}R+\frac{1}{2}B+G = \underbrace{\frac{1}{2}(R+G+B)}_{\text{نصف مجموع کل رنگ‌ها}} + \frac{1}{2}G$$

= خاکستری + ...

بنابراین چشم انسان رنگ سبز خالصی را می‌بیند که یک مؤلفه خاکستری نیز به آن اضافه شده و به این رنگ سبز خالصی است که روشنایی افزایش یافته است.

سوال 22) برای کاهش رنگ زرد چندین روش وجود دارد:

① مستقیم مقدار رنگ زرد را در ماتریس کم کنیم.

② چون رنگ آبی مکمل رنگ زرد هست با افزایش رنگ آبی، اثر رنگ زرد کم می‌شود.

③ افزایش رنگ فیروزه‌ای یا ارغوانی (magenta): این دو رنگ نیز وقتی با هم ترکیب می‌شوند با رنگ آبی و زرد ترکیب می‌شوند، اثر رنگ زرد را کاهش می‌دهند.

④ چون رنگ زرد از ترکیب قرمز و سبز تشکیل می‌شود با کاهش هر کدام از این دو رنگ (قرمز یا سبز) می‌توان زرد را کاهش داد.

(23

$$L^* = 116\, h\left(\frac{Y}{Y_w}\right) - 16$$

$$a^* = 500\left(h\left(\frac{X}{X_w}\right) - h\left(\frac{Y}{Y_w}\right)\right)$$

$$b^* = 200\left(h\left(\frac{Y}{Y_w}\right) - h\left(\frac{Z}{Z_w}\right)\right)$$

$$h(q) = \begin{cases} \sqrt[3]{q} & q \geq 0.008856 \\ 7.787q + \frac{16}{116} & q \leq 0.008856 \end{cases}$$

حال با توجه به مقادیر جدول زیر، مقادیر L^* و b^*, a^* را به دست می‌آوریم:

	R	G	B	$h(X/X_w)$	$h(Y/Y_w)$	$h(Z/Z_w)$
قرمز	1	0	0	0.85	0.66	0.14
زرد	1	1	0	0.94	0.96	0.4
سبز	0	1	0	0.57	0.85	0.4
فیروزه‌ای	0	1	1	0.73	0.8	1
آبی	0	0	1	0.556	0.446	0.96
بنفش	1	0	1	0.95	0.74	0.98
سفید	1	1	1	1	1	1
مشکی	0	0	0	0.14	0.14	0.14

سوال 26) دقت $C=I$ کروی، $\Sigma=I$ نامساوی.

$$D(z,a) = \sqrt{(z-a)^T(z-a)}$$

از همین رابطه داخل رادیکال داریم:

$$\|z-a\| = (z-a)^T(z-a) \rightarrow D(z,a) = \sqrt{\|z-a\|} = \sqrt{(z-a)^T(z-a)}$$