گزارش پروژه سیگنال

پروژه تصویر

نیکی نزاکتی
۹۸۵۲۲۰۹۴

۱. ذخیره سازی تصویر

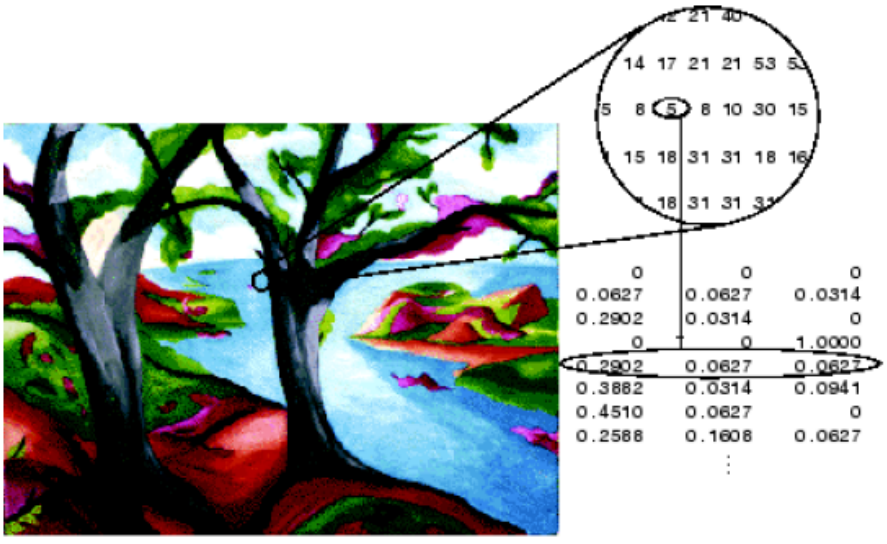
روش های ذخیره سازی تصویر

در جعبه ابزار پردازش تصویر متلب، 4 نوع عکس تعریف شده اند که خلاصه ای از آنها را در زیر مشاهده می کنید.

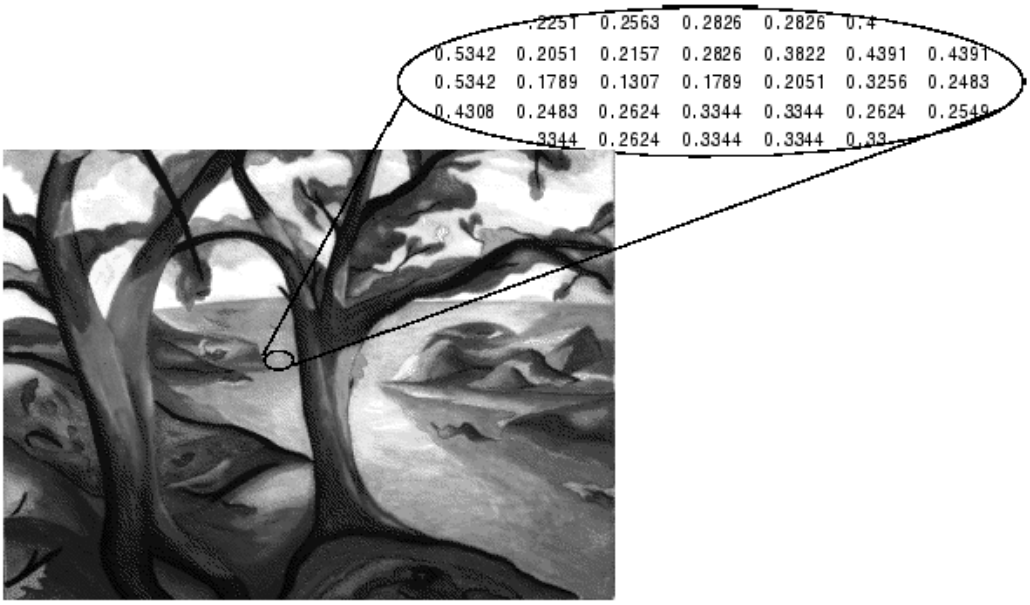
این عکس ها، مشخص می کنند که متلب چگونه داده های ماتریسی را بصورت مقادیر پیکسل ها تفسیر می کند.

باینری: یک ماتریس از نوع Logical است که تنها حاوی 0 و 1 است. این 0 ها و 1 ها را می توان بصورت رنگ های سیاه و سفید در نظر گرفت.

ایندکس شده (شبه رنگی): یک عکس اندیس گذاری شده، حاوی یک ماتریس و یک نقشه رنگ(colormap) است. مقادیر پیکسل های این ماتریس، مشخص کننده ی سطرهای نقشه ی رنگ هستند. هر نقطه ی رنگی(پیکسل) با یک سطر از نقشه ی رنگ مشخص می شود. در این کتاب، از متغیر X برای اشاره به آن ماتریس استفاده می شود و از متغیر map برای مشخص کردن ماتریس نقشه ی رنگ استفاده می شود.
ماتریس نقشه ی رنگ، یک ماتریس m×3 است(یعنی سه ستون دارد)، که کلاس آن از نوع double است و حاوی اعداد اعشاری بین بازه ی [0,1] است. هر سطر از map (نقشه رنگ)، تعیین کننده ی مولفه های قرمز، سبز و آبی از یک نقطه ی رنگی است. یک عکس ایندکس شده، از ترسیم مقادیر نقشه ی رنگ در پیکسل ها بوجود می آید. رنگ هر پیکسل از عکس با استفاده از یک سطر از نقشه ی رنگ (map) مشخص می شود. یک نقشه ی رنگ، همراه با یک ماتریس عکس ایندکس شده، ذخیره می شود و به طور اتوماتیک هنگامی که از تابع imread استفاده می کنیم همراه با ماتریس عکس بارگذاری می شود. پس از اینکه ماتریس عکس و نقشه ی رنگ در محیط workspace در متغیرهای جداگانه در متلب خوانده شدند، می توانید ارتباط بین عکس و نقشه ی رنگ را متوجه شوید. ارتباط بین مقادیر ماتریس یک عکس و نقشه ی رنگ آن عکس، به نوع ماتریس عکس بستگی دارد. در صورتی که ماتریس عکس مورد نظر از نوع single یا double باشد،این ماتریس حاوی مقادیر صحیحی از ۱ تا p است، به طوری که p مشخص کننده ی طول نقشه ی رنگ است. مقدار ۱، به اولین سطر در نقشه ی رنگ اشاره می کند و مقدار ۲ به دومین سطر اشاره می کند و همینطور الی آخر. در صورتی که ماتریس ما از نوع logical یا uint8 یا uint16 باشد، مقدار ۰ در این ماتریس، به اولین سطر در نقشه ی رنگ اشاره می کند و مقدار ۱ به دومین سطر اشاره می کند و همین طور الی آخر. عکس زیر، ساختار یک عکس ایندکس شده را نشان می دهد. در این عکس، ماتریس ما از نوع double است ، بنابراین مقدار ۵(که در زیر می بینید) به پنجمین سطر از نقشه ی رنگ اشاره می کند:



سیاه سفید: یک عکس سیاه و سفید، یک ماتریس است که به طوری که مقادیر آن شدت روشنایی را در یک محدوده مشخص می کنند. نرم افزار متلب یک عکس سیاه و سفید را بعنوان یک ماتریس منحصر بفرد ذخیره می کند به طوری که هر درایه از این ماتریس به یک پیکسل از یک عکس تعلق دارد. به طور قراردادی، در این کتاب از متغیر I برای اشاره به عکس های سیاه و سفید استفاده می کنیم. این ماتریس می تواند از نوع های uint8 یا uint16 یا int16 یا single یا double باشد. در حالی که عکس های سیاه و سفید به ندرت همراه با یک نقشه ی رنگ ذخیره می شوند، اما متلب برای نمایش آنها از نقشه ی رنگ استفاده می کند. برای یک ماتریس از نوع single یا double، که از یک نقشه ی رنگ پیش فرض استفاده می کنند، عدد 0 مشخص کننده ی شدت رنگ سیاه(black) است و عدد 1 مشخص کننده ی (شدت) رنگ سفید است. برای یک ماتریس از نوع uint8 یا uint16 یا unt16، شدت ((intmin(class(I مشخص کننده ی رنگ سیاه و شدت ((intmin(class(I مشخص کننده ی رنگ سفید است. عکس زیر، یک عکس سیاه و سفید از نوع double را مشخص می کند.



رنگی (RGB) : یک عکس رنگی(RGB)، عکسی است که هر پیکسل از آن با استفاده از سه مقدار قرمز(red) و سبز(green) و آبی(blue) مشخص می شوند. متلب عکس های رنگی را بصورت یک ماتریس m x n x 3 ذخیره می کند به طوری که هر یک از لایه های این ماتریس، مشخص کننده ی مقادیر قرمز(red) و سبز(green) و آبی(blue) است. عکس های رنگی، از یک نقشه ی رنگ، استفاده نمی کنند. رنگ هر پیکسل در یک عکس رنگی، با ترکیب شدت رنگ های قرمز و سبز و آبی آن عکس به دست می آید. فرمت های فایل های گرافیکی، عکس های رنگی را بصورت عکس های 24 بیتی ذخیره می کنند، به طوری که هرکدام از مولفه های قرمز و سبز و آبی از این عکس، 8 بیت را اشغال می کنند. این باعث می شود که بطور بالقوه 16 میلیون رنگ به وجود آیند. یک ماتریس عکس رنگی می تواند از نوع uint8 یا uint16 یا single یا double باشد.در یک ماتریس عکس رنگی که از نوع single یا double باشد، هر یک از مولفه های رنگی(قرمز و سبز و آبی) مقداری بین 0 تا 1 هستند. در یک عکس رنگی، یک پیکسل که مولفه های آن به صورت (0,0,0) هستند، به رنگ سیاه

نمایش داده می شود. و یک پیکسل دیگر که مولفه های آن صورت (1,1,1) باشند، به رنگ سفید نمایش داده می شود. این مولفه های سه تایی که مشخص کننده ی یک پیکسل هستند، با استفاده از سه بعد، ذخیره می شوند(یعنی هر کدام در یک لایه از ماتریس عکس رنگی قرار می گیرند). بعنوان مثال رنگ های قرمز و سبز و آبی از پیکسل با مختصات (10,5) به ترتیب در مکان های RGB(10,5,1) و RGB(10,5,2) و RGB(10,5,3) ذخیره می شوند. در عکس زیر، یک عکس رنگی از نوع double را مشاهده می کنید:

برای تعیین یک پیکسل که در نقطه مختصات (2,3) قرار دارد، می توانید به سه تایی RGB ذخیره شده در مکان های (2,3,1:3) نگاه کنید. فرض کنید که درایه ی (2,3,1) حاوی مقدار 0.5176 باشد و درایه ی (2,3,2) حاوی مقدار 0.1608 باشد و درایه ی (2,3,3) حاوی مقدار 0.0627 باشد، در این صورت، رنگ پیکسل (2,3) برابر خواهد بود با 0.5176 0.1608 0.0627.

انواع فرمت عکس

برای ذخیره کردن عکسها میتوان از فرمتهای مختلف استفاده کرد. فرمت معروف JPG یا مشابه آن JPEG و JFIF معروفترین فرمت عکس است اما فرمتهای دیگری مثل PNG و GIF و حتی BMP و TIFF یا TIF نیز در شرایط خاصی استفاده میشوند.

فرمت JPG یا JPEG یا JFIF: معروفترین فرمت عکسهای دیجیتال، jpg است و تقریباً تمام دوربینها و وسایل الکترونیکی از این فرمت پشتیبانی میکنند. فرمت JPG جزو فرمتهای **توأم با افت کیفیت** محسوب میشود اما بسته به تنظیمات فشردهسازی، کیفیت عکسها نزدیک به عکس اولیه است. با توجه به افت کیفیت، اگر عکسی را ویرایش کنیم و با این فرمت ذخیره کنیم، کیفیت آن اندکی پایینتر از عکس اصلی خواهد بود.

فرمت PNG: این فرمت منبع‌باز به عنوان جایگزین موفق فرمت GIF معرفی شده است و از ۱۶ میلیون رنگ (یا عمق رنگ ۲۴ بیت) و همین‌طور شفافیت یا Transparency پشتیبانی می‌کند. به عبارت دیگر می‌توان لوگو، متن یا ... دیگری را روی زمینه‌ی شفاف قرار داد و در طراحی وب، ویرایش ویدیو و واترمارک کردن آن و بسیاری کاربری‌های دیگر استفاده کرد. PNG به صورت تصویر متحرک نیز ساخته شده و فرمت موردبحث APNG نام دارد. PNG فرمتی فشرده اما **بدون افت کیفیت** است لذا معمولاً حجمی معادل چند برابر jpg دارد اما با ویرایش عکس‌های png، افت کیفیتی وجود ندارد.

فرمت BMP: فرمت bmp از جمله فرمت‌های Lossless یا **بدون افت کیفیت** است. به علاوه فایل‌هایی با پسوند bmp بدون فشرده‌سازی ذخیره می‌شوند و به همین علت است که حجمشان نسبتاً زیاد است و حتی با Zip کردن هم کاهش حجم نسبتاً زیادی خواهند داشت. این فرمت در سیستم عامل ویندوز کاربرد بیشتری نسبت به سایر سیستم عامل‌ها دارد.

۲. تشخیص لبه

الگوریتم های تشخیص لبه

آشکارسازی لبه (edge detection) معمولا برای تشخیص لبه های یک شی از بین چند شی دیگر مورد استفاده قرار می گیرد، برای این کار از تابعی به نام edge استفاده می شود.

تغییرات فیزیکی به صورت تغییر رنگ و تغییر شدت روشنایی به صورت لبه در تصویر نمایان می شوند. در محیط با مقادیر پیوسته، مشتق، تغییرات ناگهانی و شدت آن را مشخص می کند و در محیط گسسته محاسبه ی تغییرات نسبت به پیکسل های مجاور، تقریبی از مشتق را نمایان می سازد. در عملیات لبه برداری ورودی یک تصویر به فرمت intensity می باشد و در خروجی تصویر binary داده می شود، که در تصویر حاصل مرزهای بیرونی تصویر به صورت ۱ و مرزهای داخل به صورت ۰ نشان داده می شود.

```
I=rgb2gray(i1);
Bw=edge(I,'sobel')
```

Edge لبه ها را در تصاویر intensity پیدا می کند، این تابع یک تصویر باینری یا intensity را به عنوان ورودی می گیرد و یک تصویر باینری bw به همان اندازه ی تصویر اولی بر می گرداند، که جاهایی که تابع لبه ها را در تصویر پیدا می کند، در تصویر خروجی ۱ می کند و جاهایی دیگر را ۰ قرار می دهد.

برخی از الگوریتم های لبه برداری

۱. الگوریتم soble

۲. الگوریتم canny

۳. الگوریتم Roberts

۴. الگوریتم prewitt

۵. الگوریتمzero-cross

الگوریتم soble : این متد لبه ها را با استفاده از تخمین زدن مشتق پیدا می کند، که لبه ها را در آن نقاطی بر می گرداند که گرادیان تصویر max ، I است.

Bw= edge(I,'sobel',thresh)

مقدار thresh یک میزان آستانه را برای این متد مشخص می کند.

این تابع (edge) را از همه لبه هایی که قوی تر (بیشتر) از thresh نیستند چشم پوشی می کند و اگر ما مقدار این thresh را مشخص نکنیم یا اگر thresh خالی باشد {[]} ، تابع edge خود به طور اتوماتیک مقداری را انتخاب می کند.

Bw=edge(I,'sobel',thresh,direction)

در این syntax , directin جهت را مشخص می کند، یعنی رشته ای است که مشخص می کند که این تابع لبه های افقی یا عمودی و یا هردو را جستجو کند که به طور پیش فرض هردو را جستجو می کند.

'horizontal' افقی:

'vertical'عمودی:

Bw=edge(I,'sobel',...,options)

در این دستور تابع یک رشته ی اختیاری به عنوان ورودی می گیرد که رشته **'**nothinning**'** سرعت عملیات الگوریتم را بالا می برد؛ به این علت که در مرحله ی نازک شدن لبه های اضافی می گذرد (می پرد) و اگر رشته ی **'**nothinning**'** را انتخاب کنیم، الگوریتم لبه های نازک شده را نیز درخواست می کند.

edge(I,'sobel',...)=[Bw,thresh]

این دستور، مقدار threshold (آستانه) را برمی گرداند.

edge=[Bw,thresh,gv,gh](....)

Edge detection by mathworks.ir%

در این دستور، لبه های افقی و عمودی (gv,gh) را با توجه به عملگرهای گرادیان بر می گرداند.

دو متد Roberts و prewitt نیز هم به همین گونه هستند.

الگوریتم canny: این متد ل لبه ها را با جستجوی max های محلی (موضعی) گرادیان I ، که گرادیان از روی مشتق فیلتر گاوس (Gaussian) محاسبه می شود.

این متد از دو آستانه (thresholds) استفاده می کند تا لبه های ضعیف و قوی را پیدا کند که فقط شامل لبه هایی ضعیف در خروجی می باشد که آنها متصل به لبه های قوی باشند.

این روش بیشتر به کشف لبه های ضعیف به درستی می پردازد و کمتر فریب نویز را می خورد و از بقیه روش ها بهتر است.

Bw= edge(I,'canny',thresh)

این متد یک بدار آستانه (thresh) را مشخص می کند که المنت اول آن آستانه پایین و المنت دوم آن آستانه بالا را مشخص می کند.

اگر یک عدد را به عنوان (thresh) انتخاب کنیم این عدد به عنوان آستانه بالا (high threshold) و عدد (thresh×۰,۴) به عنوان آستانه پایین در نظر گرفته می شود و اگر هیچ عددی را برای thresh انتخاب نکنیم، تابع edge خود به طور اتوماتیک هردو المنت را انتخاب می کند.

edge(I,'canny',...)=[Bw,thresh]

دستور بالا یک بردار دو المنتی را برمی گداند که میزان آستانه بالا و پایین را مشخص می کند.

تشخیص لبه با متلب

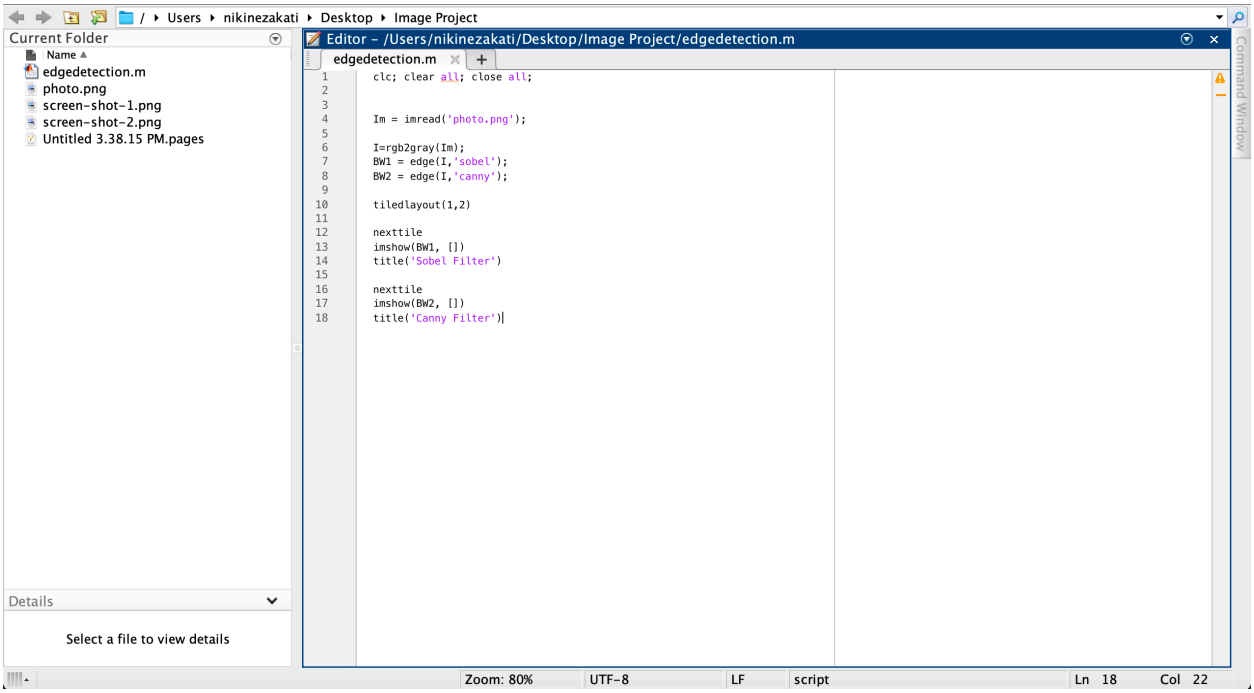
یک عکس رنگی را در فولدر برنامه قرار میدهیم و آن را با تابع imread صدا میزنیم. الگوریتم های تشخیص لبه متلب برای عکس های رنگی کار نمیکنند برای همین با استفاده از تابع rgb2gray آن را به عکس سیاه و سفید تبدیل میکنیم.

سپس با دو روش sobel و canny تشخیص لبه را روی عکس صدا میزنیم و عکس را نمایش میدهیم.

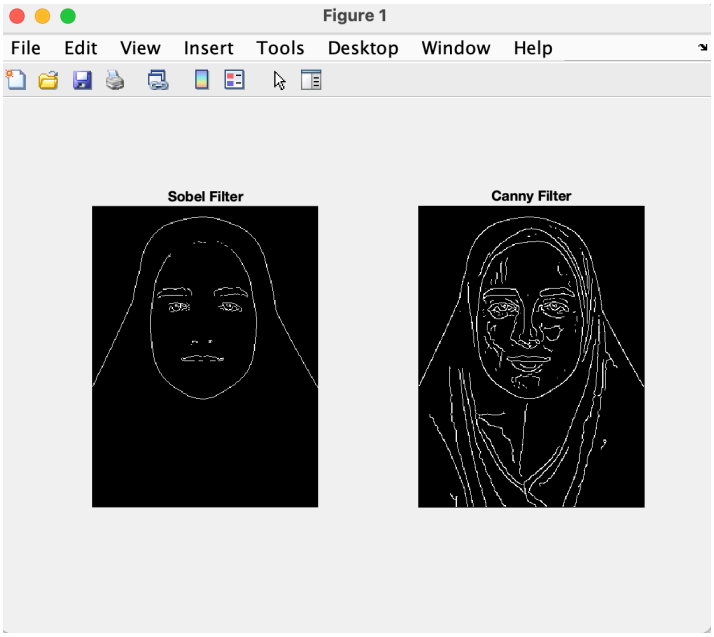
```
edge(I,'sobel')
edge(I,'canny')
```

عکس اولیه:

کد متلب:



خروجی:



۳. نویز

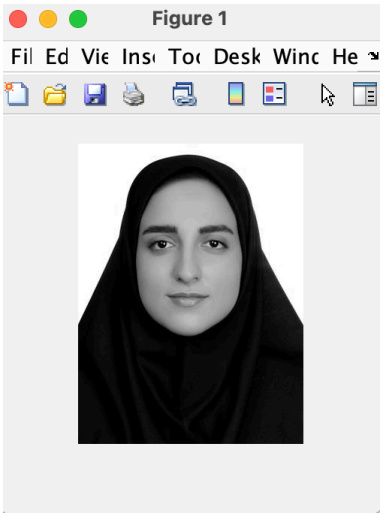
سیاه و سفید کردن عکس

با استفاده از تابع rgb2gray آن را به عکس سیاه و سفید تبدیل میکنیم.

کد:

```
Im = imread('photo.png');
I=rgb2gray(Im);
```

خروجی:



مفهوم SNR

نسبت سیگنال به نویز (Signal to Noise Ratio) برابر با نسبت توان سیگنال به توان نویز موجود در آن سیگنال است که به اختصار به آن SNR می‌گویند. نسبت سیگنال به نویز معیاری برای بیان عملکرد بهینه سیستم پردازش سیگنال محسوب می‌شود، البته به شرط اینکه نویز یک تابع گاوسی (Gaussian) باشد. نسبت سیگنال به نویز یا SNR را در حالت کلی به صورت زیر محاسبه می‌کنند:

SNR = Ps / Pn

اضافه کردن نویز به تصویر

برای اضافه کردن نویز به طوری که SNR برابر 6db شود ابتدا تصویر را به double تبدیل کرده و در متلب

از تابع زیر استفاده میکنیم:

```
J = awgn(img,6,'measured');
```

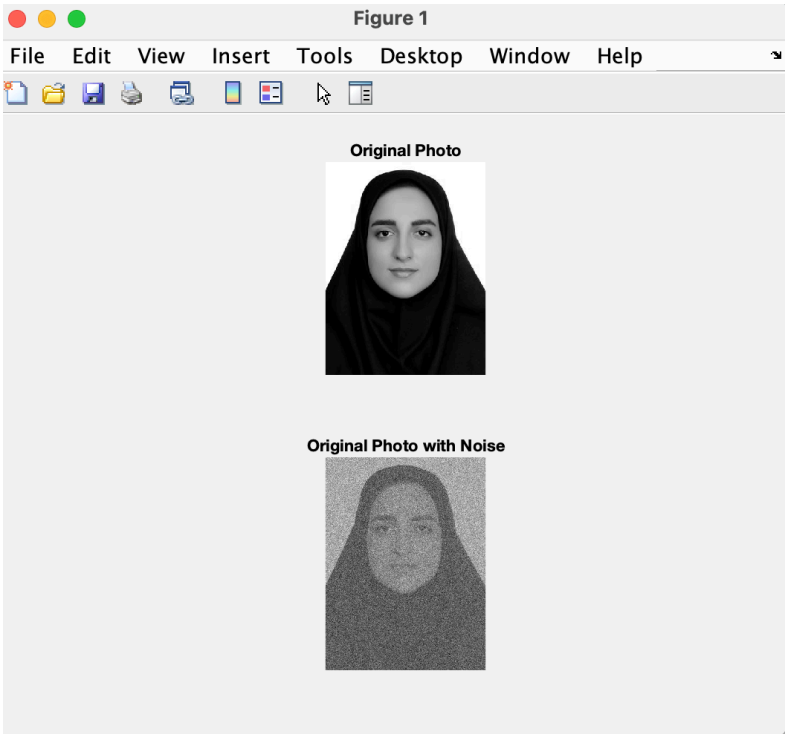
کد متلب:

Editor – /Users/nikinezakati/Desktop/Image Project/noise.m

noise.m | awgn.m

```
clc; clear all; close all;


Im = imread('photo.png');

I=rgb2gray(Im);
subplot(211); imshow(I, [])
title('Original Photo')

img=double(I);

J = awgn(img,6,'measured');
subplot(212);imshow(J, [])
title('Original Photo with Noise')

```

خروجی:



رفع نویز

برای رفع نویز از تابع زیر در متلب استفاده میکنیم:

```
xden = medfilt2(J);
```

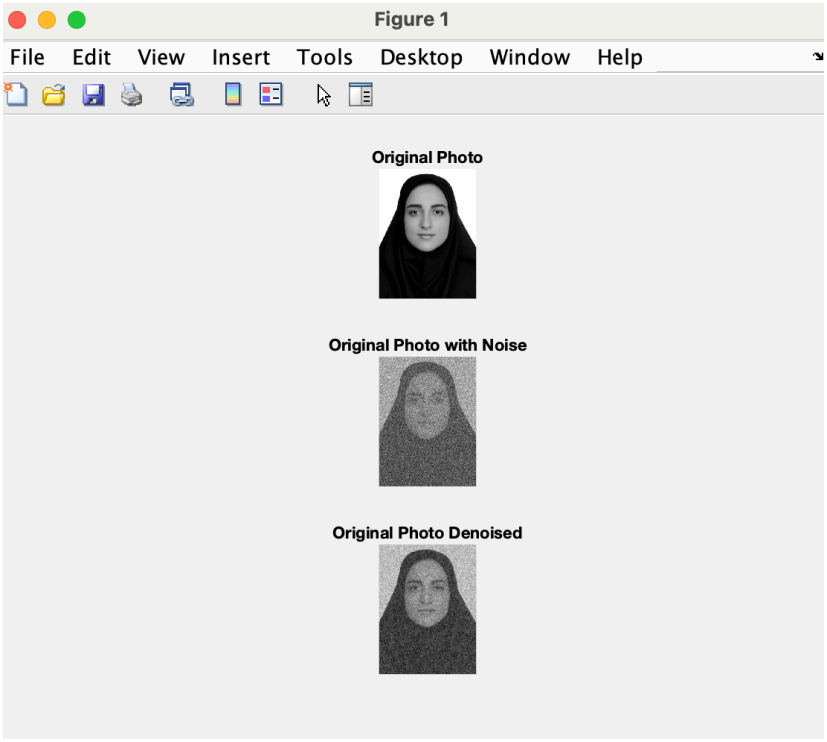
کد متلب:

Editor – /Users/nikinezakati/Desktop/Image Project/noise.m

noise.m | awgn.m

```
clc; clear all; close all;


Im = imread('photo.png');

I=rgb2gray(Im);
subplot(311); imshow(I, [])
title('Original Photo')

img=double(I);

J = awgn(img,6,'measured');
subplot(312);imshow(J, [])
title('Original Photo with Noise')

xden = medfilt2(J);
subplot(313); imshow(xden, [])
title('Original Photo Denoised')

```

خروجی:



برای مشخص شدن میزان دقت رفع نویز میتوانیم از تابع imshowpair(A,B,'diff') استفاده کنیم و عکس اولیه را با خودش، عکس نویز دار و عکس رفع نویز شده مقایسه میکنیم:

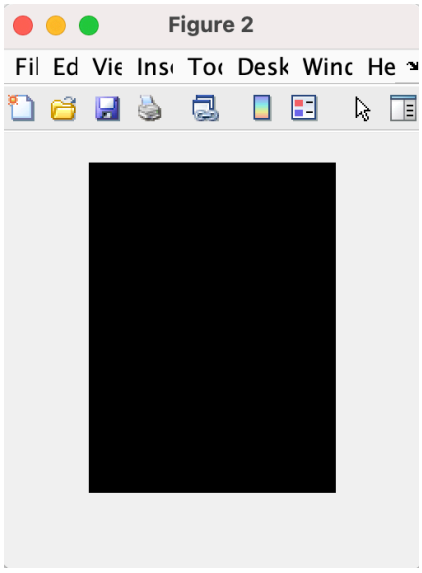
کد متلب:

Editor - /Users/nikinezakati/Desktop/Image Project/noise.m

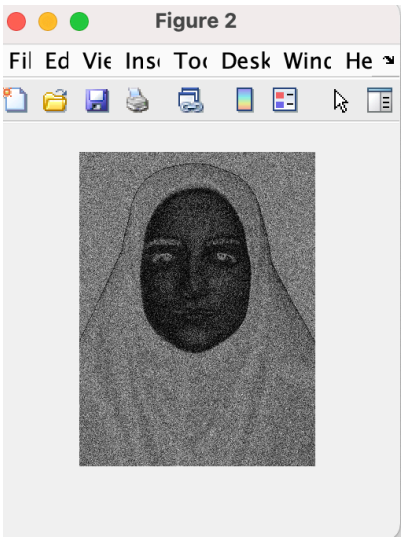
noise.m

```
clc; clear all; close all;


Im = imread('photo.png');

I=rgb2gray(Im);
subplot(311); imshow(I, [])
title('Original Photo')

img=double(I);

J = awgn(img,6,'measured');
subplot(312);imshow(J, [])
title('Original Photo with Noise')

xden = medfilt2(J);
subplot(313); imshow(xden, [])
title('Original Photo Denoised')

figure
imshowpair(img,img,'diff')

figure
imshowpair(img,J,'diff')

figure
imshowpair(img,xden,'diff')
```

خروجی مقایسه عکس با خودش:



خروجی مقایسه عکس با عکس نویزدار شده:



خروجی مقایسه عکس با عکس رفع نویز شده: