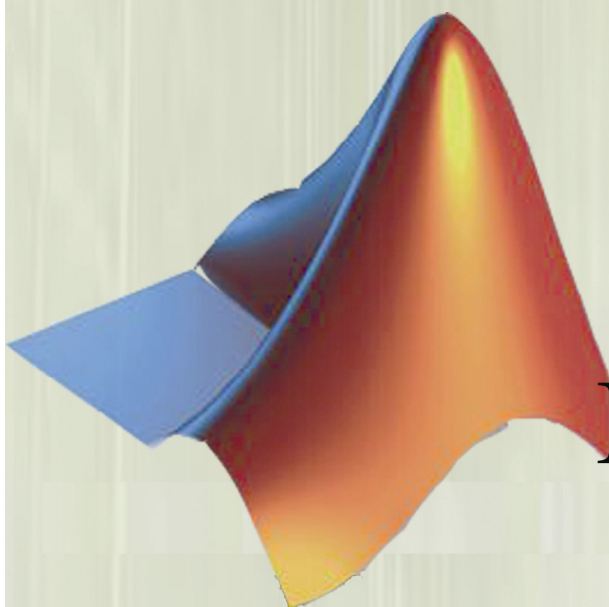


جزوه آموزش

پردازش تصویر

با استفاده از نرم افزار

MATLAB



گروه ریاتیک دانشگاه پیام نور قم

ترم پاییز 87-1386

به نام خدا

پردازش تصاویر

پردازش تصاویر امروزه بیشتر به موضوع پردازش تصویر دیجیتال گفته می‌شود که شاخه‌ای از دانش رایانه است که با پردازش سیگنال دیجیتال که نماینده تصاویر برداشته شده با دوربین دیجیتال یا پویش شده توسط پویشگر هستند سر و کار دارد. پردازش تصاویر دارای دو شاخه عمده بهبود تصاویر و بینایی ماشین است. بهبود تصاویر دربرگیرنده روش‌هایی چون استفاده از فیلتر محوکننده و افزایش تضاد برای بهتر کردن کیفیت دیداری تصاویر و اطمینان از نمایش درست آنها در محیط مقصد (مانند چاپگر یا نمایشگر رایانه) است، در حالی که بینایی ماشین به روش‌هایی می‌پردازد که به کمک آنها می‌توان معنی و محتوای تصاویر را درک کرد تا از آنها در کارهایی چون رباتیک و محور تصاویر استفاده شود.

بینایی رایانه‌ای

بینایی رایانه‌ای (Computer vision) یکی از شاخه‌های مدرن، و پرتنوع هوش مصنوعی است که با ترکیب روش‌های مربوط به پردازش تصاویر (Image processing) و ابزارهای تعلّم ماشینی رایانه‌ها را به بینایی اشیاء، مناظر، و "درک" هوشمند خصوصیات گوناگون آنها توانا می‌گرداند.

وظایف اصلی در بینایی رایانه‌ای

تشخیص شیء : تشخیص حضور و/ یا حالت شیء در یک تصویر. به عنوان مثال :

- جستجو برای تصاویر دیجیتال بر اساس محتوایشان (بازیابی محتوای محور تصاویر).
- شناسایی صورت انسان‌ها و موقعیت آنها در عکس‌ها.
- تخمین حالت سه بعدی انسان‌ها و اندام‌هایشان.

پیگیری : پیگیری اشیای شناخته شده در میان تعدادی تصویر پشت سر هم. به عنوان مثال:

- پیگیری یک شخص هنگامی که در یک مرکز خرید راه می‌رود.

تفسیر منظره : ساختن یک مدل از یک تصویر/تصویر متحرک. به عنوان مثال :

● ساختن یک مدل از ناحیه پیرامونی به کمک تصاویری که از دوربین نصب شده بر روی یک ربات گرفته می‌شوند.

خود مکان‌یابی: مشخص کردن مکان و حرکت خود دوربین به عنوان عضو بینایی رایانه. به‌عنوان مثال :

● مسیریابی یک ربات درون یک موزه.

سامانه‌های بینایی رایانه‌ای

یک سامانه نوعی بینایی رایانه‌ای را می‌توان به زیرسامانه‌های زیر تقسیم کرد :

تصویربرداری

تصویر یا دنباله تصاویر با یک سامانه تصویربرداری (دوربین، رادار، لیدار، سامانه توموگرافی) برداشته می‌شود. معمولاً سامانه تصویربرداری باید پیش از استفاده تنظیم شود.

پیش‌پردازش

در گام پیش‌پردازش، تصویر در معرض اعمال "سطح پایین" قرار می‌گیرد. هدف این گام کاهش نوفه (کاهش نویز - جدا کردن سیگنال از نویز) و کم کردن مقدار کلی داده هاست. این کار نوعاً با به‌کارگیری روش‌های گوناگون پردازش تصویر (دیجیتال) انجام می‌شود. مانند:

زیرنمونه‌گیری تصویر، اعمال فیلترهای دیجیتال، پیچش‌ها، همبستگی‌ها یا فیلترهای خطی لغزش‌ناسته، عملگر سوبل، محاسبه گرادیان X و Y (و احتمالاً گرادیان زمانی)، تقطیع تصویر، آستانه‌گیری پیکسلی، انجام یک ویژه‌تبدیل بر تصویر، تبدیل فوریه، انجام تخمین حرکت برای ناحیه‌های محلی تصویر که به نام تخمین شارش نوری هم شناخته می‌شود، تخمین ناهمسانی در تصاویر برجسته‌بینی و تحلیل چنددقتی.

استخراج ویژگی

هدف از استخراج ویژگی کاهش دادن بیشتر داده‌ها به مجموعه‌ای از ویژگی‌هاست، که باید به اغتشاشاتی چون شرایط نورپردازی، موقعیت دوربین.

● انجام آشکارسازی لبه.

● استخراج ویژگی‌های گوشه‌ای.

- استخراج تصاویر چرخش از نقشه‌های ژرفا.
- بدست آوردن خطوط تراز و احتمالاً گذر از صفرهای خمش.

ثبت

هدف گام ثبت برقراری تناظر میان ویژگی‌های مجموعه برداشت شده و ویژگی‌های اجسام شناخته شده در یک پایگاه داده‌های مدل و/ یا ویژگی‌های تصویر قبلی است. در گام ثبت باید به یک فرضیه نهایی رسید. چند روش این کار عبارت‌اند از:

- تخمین کمترین مربعات.
- تبدیل هاگ در انواع گوناگون.
- درهم‌سازی هندسی.
- پالودن ذره‌ای.

کاربردها

هوش مصنوعی، یادگیری ماشینی، کاوش‌های ماشینی در داده‌ها، کاوش‌های ماشینی در متون، محاسبات نرم، منطق فازی، پردازش تصاویر.

پردازش تصویر

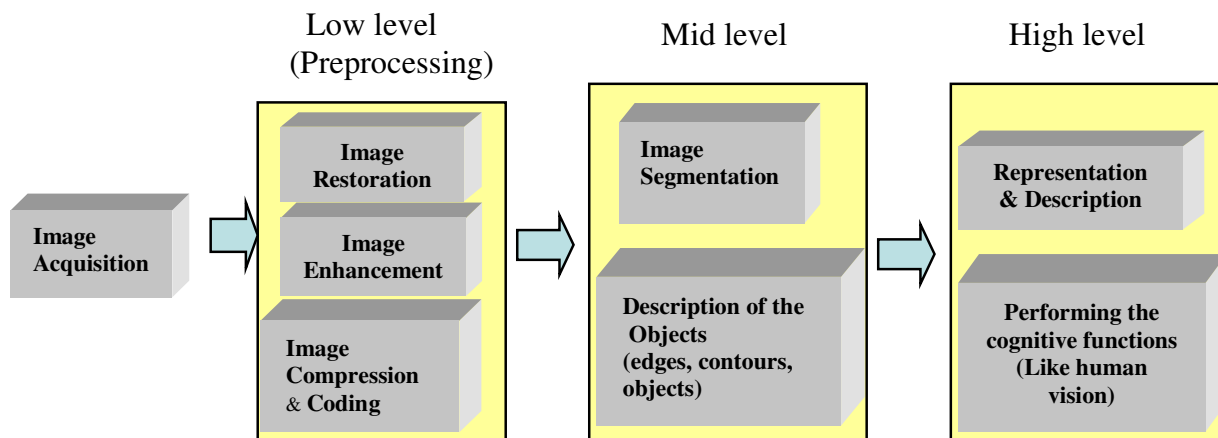
مرز مشخصی بین پردازش تصویر از یک طرف و بینایی ماشین (Computer Vision) از طرف دیگر نمی‌توان مشخص کرد، با این حال می‌توان سه نوع پردازش سطح پایین (Low level)، سطح میانی (Mid level)، و سطح بالا (High level) تشخیص داد.

❖ پردازش سطح پایین شامل پردازش‌های ابتدایی مانند پیش پردازش‌هایی برای حذف نویز، بهبود کنتراست (Contrast) و فیلتر کردن تصویر است. مشخصه این نوع پردازش این است که ورودی و خروجی آن تصویر هستند.

❖ پردازش سطح میانی شامل بخش بندی تصویر (Image Segmentation) به منظور تقسیم آن به نواحی و اشیاء مختلف، توصیف اشیاء به فرمی که برای پردازش کامپیوتر مناسب باشند و طبقه بندی یا تشخیص اشیاء مختلف است. ویژگی این پردازش این است که ورودی آن معمولاً تصویر و خروجی آن صفاتی از اشیاء تصویر مانند لبه‌ها، کانتورها و تشخیص اشیاء است.

❖ پردازش سطح بالا شامل فهمیدن روابط بین اشیاء تشخیص داده شده (Making sense)، استنباط و تفسیر صحنه و انجام تفسیر و تشخیص هایی است که سیستم بینایی انسان انجام می دهد. بسیاری از پردازش های سطح بالا در حیطه بینایی ماشین قرار می گیرند.

به طور مثال یک سیستم تشخیص اتوماتیک متن را در نظر بگیرید. پردازش برای مشخص کردن ناحیه حاوی متن، پیش پردازش تصویر حاصل، استخراج کاراکترها مختلف و تشخیص آن کاراکترها در حیطه بحث پردازش تصویر قرار دارند. استنباط مفهوم و محتوای متن مورد نظر در حیطه بینایی ماشین یا آنالیز تصویر قرار می گیرد.



تصویر دیجیتال چیست ؟

یک تصویر را می توان توسط تابع دوبعدی $f(x,y)$ که در آن X و Y را مختصات مکانی و مقدار f در هر نقطه را شدت روشنایی تصویر در آن نقطه می نامند. اصطلاح سطح خاکستری نیز به شدت روشنایی تصاویر مونوکروم اطلاق می شود. تصاویر رنگی نیز از تعدادی تصویر دوبعدی تشکیل می شود.

زمانی که مقادیر X و Y و مقدار $f(x,y)$ با مقادیر گسسته و محدود بیان شوند، تصویر را یک تصویر دیجیتالی می نامند. دیجیتال کردن مقادیر X و Y را Sampling و دیجیتال کردن مقدار $f(x,y)$ را quantization گویند.

یک تصویر دیجیتالی از تعدادی عناصر محدود با مقدار و موقعیت مختلف تشکیل شده است. این عناصر، عناصر تصویر (Picture element) یا پیکسل (Pixel) نامیده می شوند.

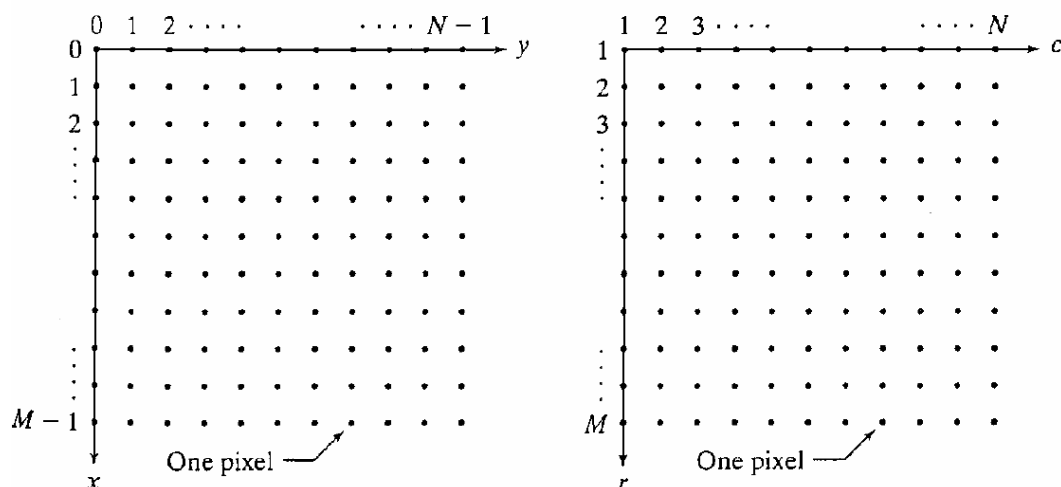
پردازش تصاویر دیجیتالی به معنی اعمال پردازش های مختلف روی یک تصویر دیجیتالی با استفاده از کامپیوتر دیجیتالی است.

نمایش تصویر دیجیتال

برای نمایش یک تصویر $M * N$ از یک آرایه دو بعدی (ماتریس) که M سطر و N ستون دارد استفاده می کنیم . مقدار هر عنصر از آرایه نشان دهنده شدت روشنایی تصویر در آن نقطه است. در تمام توابعی که پیاده سازی خواهیم کرد ، هر عنصر آرایه یک مقدار ۸ بیتی است که می تواند مقداری بین ۰ و ۲۵۵ داشته باشد. مقدار صفر نشان دهنده رنگ تیره (سیاه) و مقدار ۲۵۵ نشان دهنده رنگ روشن (سفید) است.

در بسیاری از کتابهای پردازش تصویر، مبدا تصویر به صورت زیر تعریف می شود: $(x,y)=(0,0)$. مولفه x شماره سطر و مولفه y شماره ستون یک پیکسل را نشان می دهند

به دلیل اینکه در Matlab اندیس مولفه آرایه ها از ۰ شروع نمی شود، در این محیط مبدا به صورت زیر تعریف می گردد: $(r,c)=(1,1)$



این تصویر به صورت زیر به عنوان یک ماتریس در محیط Matlab تعریف می شود.

$$f = \begin{bmatrix} f(1, 1) & f(1, 2) & \dots & f(1, N) \\ f(2, 1) & f(2, 2) & \dots & f(2, N) \\ \vdots & \vdots & & \vdots \\ f(M, 1) & f(M, 2) & \dots & f(M, N) \end{bmatrix}$$

انواع تصویر

تصاویر در Matlab شامل یک ماتریس داده می باشند و معمولاً به ماتریس جعبه رنگ وابسته هستند. سه نوع ماتریس داده برای تصاویر وجود دارد که هر کدام به صورت های مختلفی تفسیر می شوند:

- تصاویر اندیس گذاری شده (Indexed)
- تصاویر با شدت رنگ (Intensity)

- تصاویر RGB

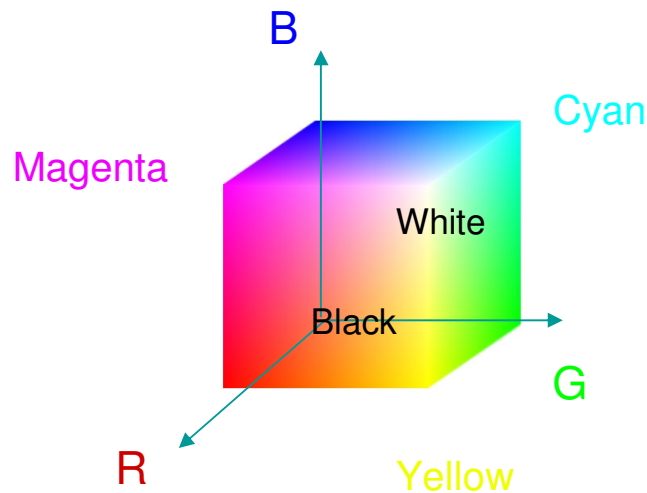
- تصاویر اندیس‌گذاری شده :

در واقع یک تصویر اندیس‌گذاری شده آرایه‌ای دو بعدی است که در هر خانه‌ی آن شماره رنگ پیکسل مورد نظر ذخیره می‌شود. یک تصویر اندیس‌گذاری شده، به یک جعبه رنگ نیاز دارد و داده‌های تصویر را به صورت اندیس‌هایی در ماتریس جعبه رنگ تفسیر می‌کند. ماتریس جعبه رنگ، یک جعبه رنگ استاندارد است که اندازه‌ی آن $M \times 3$ بوده و شامل مقادیر قابل قبول RGB است. با ارائه‌ی آرایه‌ی داده‌ی $X(i,j)$ که مربوط به تصویر است و آرایه‌ی جعبه رنگ `cmap`، رنگ هر پیکسل از تصویر با `cmap(X(i,j),:)` مشخص می‌شود که این رابطه نشانگر این است که مقادیر X ، اعداد صحیحی در بازه‌ی `[1 length(cmap)]` است. این تصویر را می‌توان با استفاده از دستور زیر نمایش داد:

```
image(X);  
colormap(cmap)
```

- تصاویر RGB :

تصاویر رنگی، از ترکیب چند تصویر دوبعدی تشکیل شده‌اند. به طور مثال در سیستم رنگی RGB هر تصویر از سه مولفه تصویر قرمز، سبز و آبی تشکیل شده است



- تصاویر با شدت رنگ:

همانطور که مشاهده کردیم در حالت عادی ۳ درجه رنگی از ۳ رنگ اصلی آبی و سبز و قرمز برای هر نقطه وجود دارد که با کم و زیاد شدن شدت هر مولفه اصلی، رنگ حاصل تغییر می‌کند. اما در پردازش تصویر این فرمت اصلاً مناسب نیست! علت این است که در پردازش تصویر معمولاً نیاز به تشخیص یک محدوده رنگی داریم، مثلاً رنگ نارنجی، در این حالت یافتن این محدوده رنگی با ترکیب ۳ رنگ اصلی ممکن نیست یا اینکه کاری بسیار دشوار و وقت گیر می‌باشد. سیستم یا روش دیگری که در این زمینه وجود دارد بر اساس قاعده زیر است.

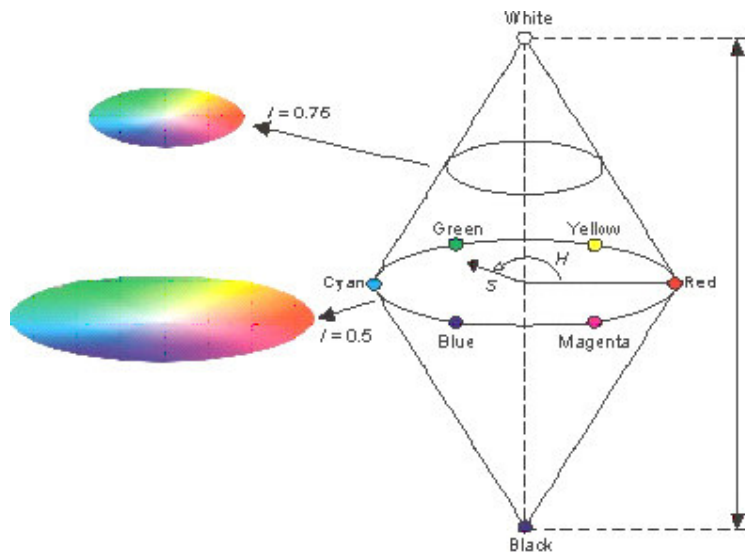
برای هر رنگ ۳ مشخصه وجود دارد:

۱- نام رنگ (Hue): که مشخص کننده نوع رنگ می باشد. به عنوان مثال آبی کمرنگ یا قرمز متمایل به نارنجی یا

...

۲- شدت رنگ (Saturation): هر رنگ می تواند پر رنگ یا کم رنگ باشد اما ماهیت ذاتی آن ثابت است و فقط کم رنگ تر یا پررنگ تر شده است

۳- روشنایی یا تیرگی رنگ (Intensity): این مؤلفه شدت رنگ را تعیین می کند که با نور تابیده شده به آن تغییر می کند. به عنوان مثال اگر نور تابیده شده کم باشد، رنگ رو به تیرگی رفته و تقریباً متمایل به سیاه می شود. بنابراین حوضه جدیدی از رنگ که الهام گرفته از چشم انسان می باشد قابل تعریف است. به این حوضه رنگی اصطلاحاً HSI گفته می شود که مخفف ۳ کلمه بالاست. شکل زیر نمودار تغییرات رنگ را با توجه به این ۳ مؤلفه بیان می کند:



H: بین ۰ تا ۳۶۰ قابل تغییر است، یعنی از قرمز تا سبز ۱۲۰ درجه و از سبز تا آبی ۱۲۰ درجه و از آبی تا قرمز ۱۲۰ درجه در خلاف جهت حرکت عقربه های ساعت.

S: بین ۰ تا ۱۰۰ قابل تغییر است که از کم رنگ (۰) تا پر رنگ (۱۰۰)

I: از ۰ تا ۱۰۰ تغییر می کند، یعنی از تاریک (۰) تا روشن (۱۰۰)

چند نمونه از رنگها در ۲ حوزه مذکور:

مقادیر HSI	مقادیر RGB	رنگ
------------	------------	-----

(۰،۰،۰)	(۲۵۵،۲۵۵،۲۵۵)	سیاه
(۰،۰،۱۰۰)	(۲۵۵،۲۵۵،۲۵۵)	سفید
(۰،۱۰۰،۱۰۰)	(۲۵۵،۰،۰)	قرمز
(۱۲۰،۱۰۰،۱۰۰)	(۰،۲۵۵،۰)	سبز
(۲۴۰،۱۰۰،۱۰۰)	(۰،۰،۲۵۵)	آبی
(۱۸۰،۵۰،۵۰)	(۶۴،۱۲۸،۱۲۸)	قهوه ای

کار کردن با سیستم HSV نیز دقیقاً معادل سیستم HIS می‌باشد. بنابراین به عنوان مثال با داشتن یک تصویر با فرمت HSV می‌توان به راحتی مکان نقاط با رنگ های مورد نظر را روی تصویر یافت.

اشکال گرافیکی دو بعدی

تابع **plot**: این تابع رسم شکل مجموعه ای از آرایه های داده ها را بر روی محورهای مختصات رسم کرده و نقاط تعیین شده را با استفاده از خطوط مستقیم به یکدیگر متصل می‌کند.

```
x=linspace (0,2*pi,30);
y=sin (x);
plot(x,y)
```

در مثال فوق ، ۳۰ نقطه در بازه ی $0 \leq x \leq 2\pi$ ایجاد می‌شود تا محور افقی نمودار را ساخته و بردار دیگری را به نام y که شامل سینوس آن نقاط در x می‌باشد ایجاد می‌کند تا بتواند نقاط را بر روی نمودار رسم کند. بعد از رسم نقاط آنها را با خطوط راست به یکدیگر متصل می‌کند.

می‌توان چند منحنی یا خط را روی یک نمودار رسم کرد.

```
z =cos(x)
plot(x,y,x,z)
```

چنانچه ترتیب آرگومان‌ها را تغییر دهید نمودار هم به اندازه ی ۹۰ درجه دوران می‌کند.

```
plot(x,y)
```

نوع خطوط، علائم و رنگها:

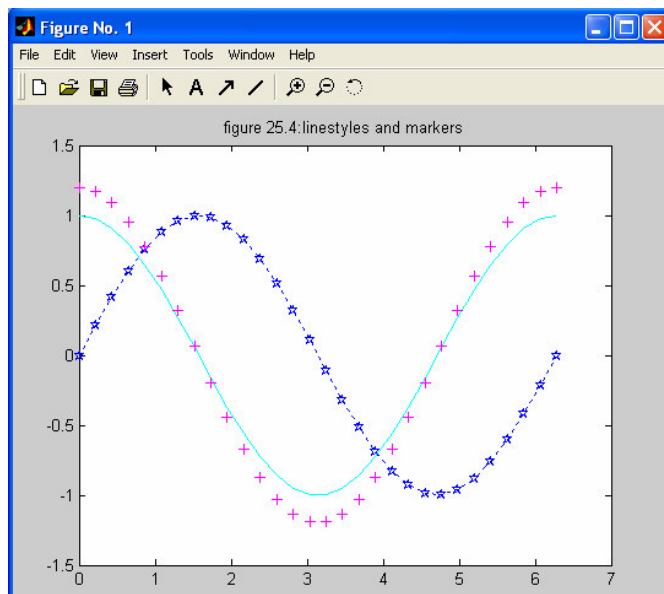
در مثال قبل دیدیم که اگر دو منحنی را در یک نمودار ایجاد کنیم Matlab به طور خودکار یکی از آنها را رنگی می‌کند و از رنگ‌های آبی و سبز برای این کار استفاده می‌کند اما می‌توان رنگ این منحنی را به دلخواه تعیین کرد. این گزینه اختیاری یک رشته کاراکتری می‌باشد که شامل یک یا چند کاراکتر از جدول زیر می‌باشد.

اگر رنگی را برای رسم نمودار خود، مشخص نکنید و از رنگ‌های پیش فرض استفاده کنید، Matlab از رنگ آبی شروع کرده و با اضافه شدن هر خط به نمودار، به ترتیب از رنگ‌های بعدی موجود در جدول استفاده می‌کند.

در صورتی که نوع خط را مشخص نکرده باشید، خط پیش فرض شما، خط توپر می‌باشد. برای علائم حالت پیش فرض وجود ندارد. اگر علامتی را مشخص نکرده باشید، علامتی هم رسم نمی‌شود.

نماد	رنگ	نماد	علائم	نماد	نوع خط
b	آبی		نقطه	:	نقطه چین
g	سبز	o	دایره	--	خط چین
r	قرمز	+	علامت جمع	—	تو پر
c	فیروزه ای	*	ستاره		
m	ارغوانی	s	مربع		
y	زرد	x	علامت ضربدر		
k	سیاه	d	نوری		
w	سفید	x	مثلث رو به پایین		
		^	مثلث رو به بالا		
		<	مثلث رو به چپ		
		>	مثلث رو به راست		
		p	ستاره پنج پر		
		h	ستاره شش پر		

اگر رنگ، علامت و نوع را در یک رشته نوشته باشید رنگ مورد نظر برای خط و علامت به کار برده می‌شود.
`plot(x,y,'b:p',x,z, 'c-',x,1.2*z, 'm+')`
`title ('figure 25.4:linestyles and markers')`



جدول بندی نمودارها، جعبه های مختصات و برچسب ها:

grid on ← این دستور خطوط شبکه ای مربوط به محورهای مختصات را برای نمودار جاری فعال می سازد.

grid off ← باعث غیر فعال شدن آن می شود.

اگر از دستور grid بدون هیچ آرگومانی استفاده کنید، اگر خطوط شبکه فعال باشد پاک می شود و اگر فعال نباشد خطوط شبکه به نمودار فوق اضافه می گردد.

به طور پیش فرض حالت grid off فعال است.

معمولاً محورهای دو بعدی توسط خطوطی در بر گرفته می شوند که به آنها جعبه محورهای مختصات گفته می شود. این جعبه را می توانید با استفاده از دستور box off غیر فعال کنید. دستور box on مجدداً محورهای مختصات را فعال می کند.

با اجرای دستور box اگر جعبه محورهای مختصات فعال باشد آن را غیرفعال کرده و اگر غیر فعال باشد آن را فعال می کند.

ylable,xlable ← با استفاده از این دستورات می توانید برای محورهای مختصات برچسب قرار دهید.

title ← عنوان را در بالای نمودار قرار می دهد.

```
y=sin(x);
z=cos(x);
plot(x,y,x,z);
box off
xlabel ('independent variable x');
ylabel ('dependent variable y and z');
title ('figure 25.5:sine curves,no box');
```

text ← با استفاده از تابع text می توانید متن یا برچسبی را در محل مورد نظر خود روی سطح نمودار قرار دهید. دستور text به صورت text (x,y,'string') مورد استفاده قرار می گیرد که در آن (x,y) مختصات گوشه سمت چپ متن برچسب واحد موجود بر روی محورهای مختصات نشان می دهد.

```
grid on
box on
text(25,0.7,'sin (x)')
```

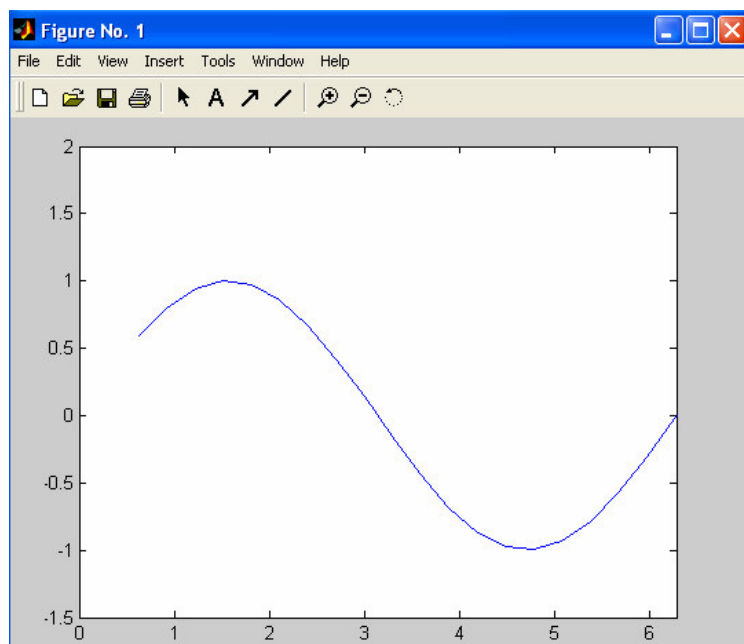
اگر بخواهید برچسبی را به نمودار اضافه کنید ولی نمی خواهید از مختصات محل مورد نظر استفاده کنید می توانید با ماوس این کار را انجام دهید. تابع gtext('text') کنترل را به پنجره figure جاری ارجاع می دهد؛ سپس خطوط متقاطع را روی صفحه جاری قرار می دهد که با حرکت ماوس جا به جا می شوند و هر جا که کلید ماوس و یا کلیدی از صفحه کلید را فشار دهید متن مورد نظر را در آنجا قرار می دهد.

تغییر دلفواه محورهای مختصات:

با استفاده از دستور **axis** می‌توانید کنترل کاملی روی درجه بندی و نحوه‌ی نمایش محورهای افقی و عمودی داشته باشید.

محدوده محورهای مختصات نمودار جاری را تنظیم میکند	<code>axis [xmin xmax ymin ymax]</code>
برداری سطری را بر می‌گرداند که شامل محدوده‌های محورهای جاری است	<code>X=axis</code>
درجه بندی محور مختصات را به حالت پیش فرض قرار می‌دهد	<code>Axis outo</code>
درجه بندی محورهای مختصات را ثابت نگه می‌دارد که اگر مقدار <code>hold</code> برابر <code>on</code> باشد، نمودارهای زیر رشته آن هم از همین محدوده محورهای مختصات استفاده می‌کنند.	<code>Axis manual</code>
محدوده محور مختصات را به اندازه بازه داده‌های رسم شده قرار میدهد	<code>Axis light</code>
محدوده محور مختصات و نسبت‌های مربوطه را تنظیم می‌کند به گونه‌ای که محور مختصات تمام فضای تخصیص داده شده به خود را پر می‌کند. این گزینه فقط زمانی کار می‌کند که <code>data aspect ratio mode</code> یا <code>plot box aspect ratio</code> به صورت <code>manual</code> باشد.	<code>Axis fill</code>

```
x = linspace(0.2*pi , 30);
y = sin(x);
plot (x,y);
axis( [0 2*pi -1.5 2])
```



هنگامی که محدوده‌ی محورهای مختصات را فقط روی یک محور تغییر دهید، استفاده از `axis` بسیار مشکل است؛ برای حل این مشکل از توابع `xlim` ، `ylim` استفاده می‌کند.

Hold : با استفاده از دستور hold می توان نمودارهای جدیدی را به نمودار موجود اضافه کرد. دستور hold on باعث می شود که نمودارهای جدید به نمودار جاری اضافه شود. با اجرای دستور hold off پنجره figure جاری برای نمودارهای جدید، پاک می شود. اگر دستور hold را بدون هیچ آرگومانی اجرا کنید، آنگاه اگر حالت جاری on باشد به off و اگر off باشد به on تبدیل می شود .

```
x = linspace(0,2*pi , 30);
y = sin (x);
z = cos(x);
plot(x,y);
hold on
ishold
ans=
    1
plot (x,2,'m');
hold off ;
ishold
ans=
    0
```

پنجره های Figure

می توان پنجره های متعددی ایجاد نمود و مجموعه داده های مختلفی را به روش های گوناگون در هر کدام از آنها رسم کرد . برای ایجاد پنجره های figure جدید، از دستور Command استفاده کنید، یا اینکه File → New Figure را اجرا کنید.

با بستن پنجره های figure آنها را حذف می کنید. البته دستور Close نیز همین کار را انجام می دهد .

clos	پنجره figure جاری را می بندد
close(h)	پنجره figure شماره h را می بندد
close all	تمام پنجره های figure را می بندد

اگر می خواهید محتویات پنجره figure را بدون حذف آن پاک کنید از دستور clf استفاده کنید.

clf محتویات پنجره جاری را پاک می کند

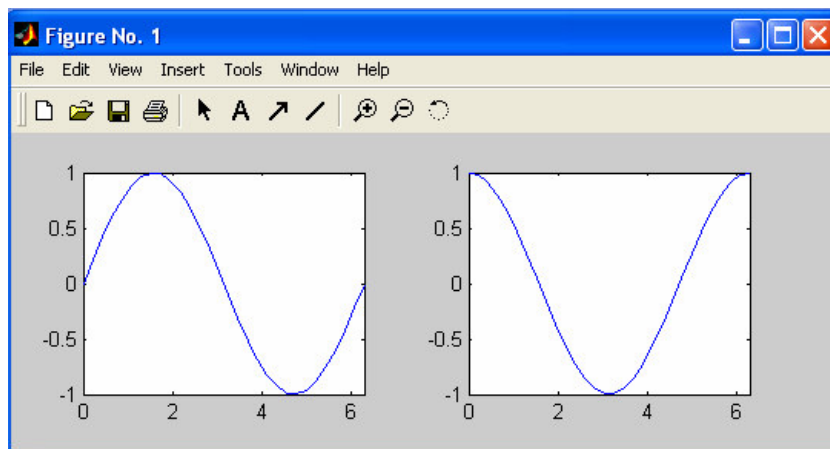
زیرنمودارها

یک پنجره figure می تواند بیش از یک مجموعه از محورهای مختصات را در خود نگه دارد. دستور subplot(m,n,p) پنجره figure جاری را به m*n ناحیه رسم نمودار تقسیم می کند و ناحیه p را به عنوان ناحیه فعال انتخاب می کند. به طور مثال:

```

x = linspace(0,2*pi,30);
y = sin(x);
z = cos(x);
subplot(2,2,1)
plot(x,y);
axis([0,2*pi,-1,1]);
subplot(2,2,2)
plot(x,z)
axis([0,2*pi,-1,1]);

```

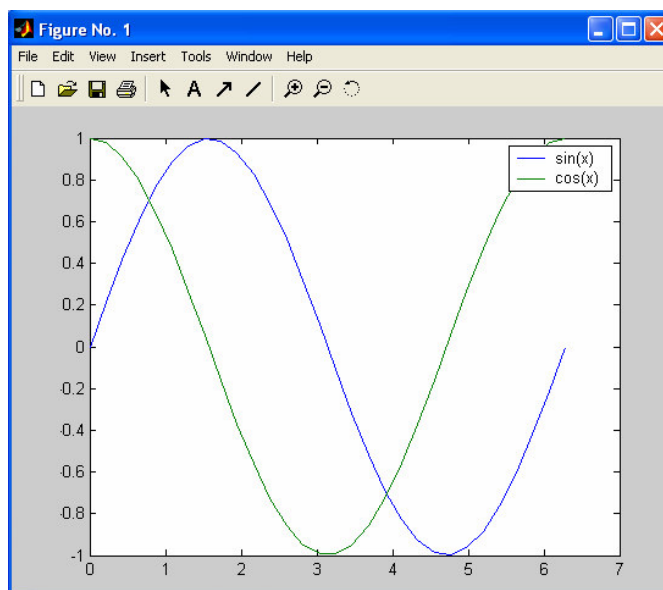


Legend: جعبه راهنمایی را روی نمودار ایجاد می کند که با استفاده از آن می توانید متن راهنمایی را برای هر خطی که بر روی نمودار قرار دارد ایجاد کنید. دستور legend off جعبه راهنما را حذف می کند.

```

x = linspace(0,2*pi,30);
y = sin(x);
z = cos(x);
plot(x,y,x,z);
legend('sin(x)', 'cos(x)');

```



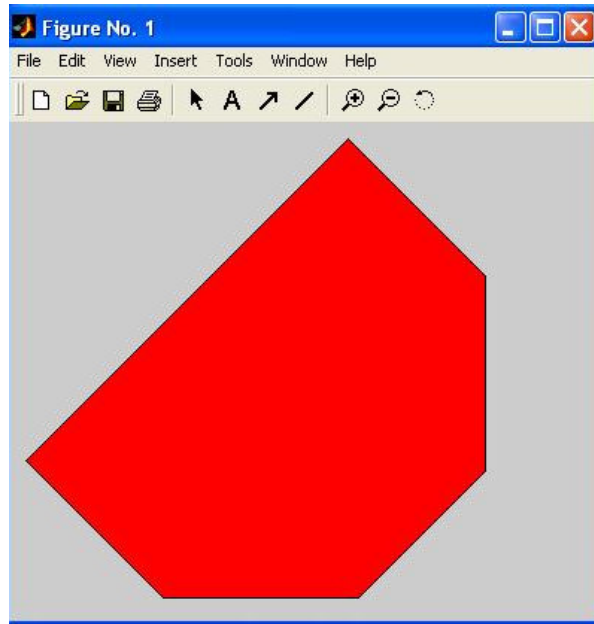
Zoom out: حالت بزرگنمایی.

با فشار دادن کلید سمت چپ ماوس، نمودار دو برابر بزرگ می شود و اگر کلید سمت راست ماوس را فشار دهید اندازه بزرگنمایی جاری نصف می شود.

Zoom(n) : نمودار را n مرتبه بزرگتر نشان می دهد.

Zoom off : حالت بزرگنمایی را غیر فعال می کند.

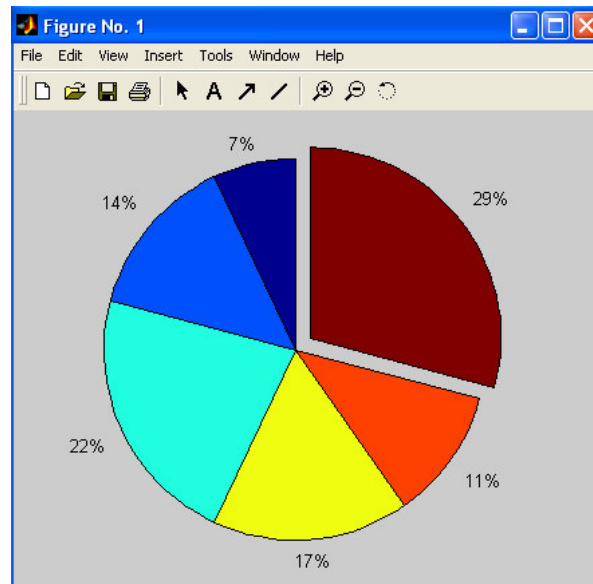
برای رنگ کردن یک چند ضلعی می توان از تابع Fill استفاده کرد. دستور Fill(x,y,'c') یک چند ضلعی را که بر حسب بردارهای ستونی X,y و رنگ c مشخص می شود را رنگ می کند.



```
t = (1:2:12)*pi / 8;  
x = sin(t);  
y = cos (t);  
fill (x,y,'r');  
axis square off
```

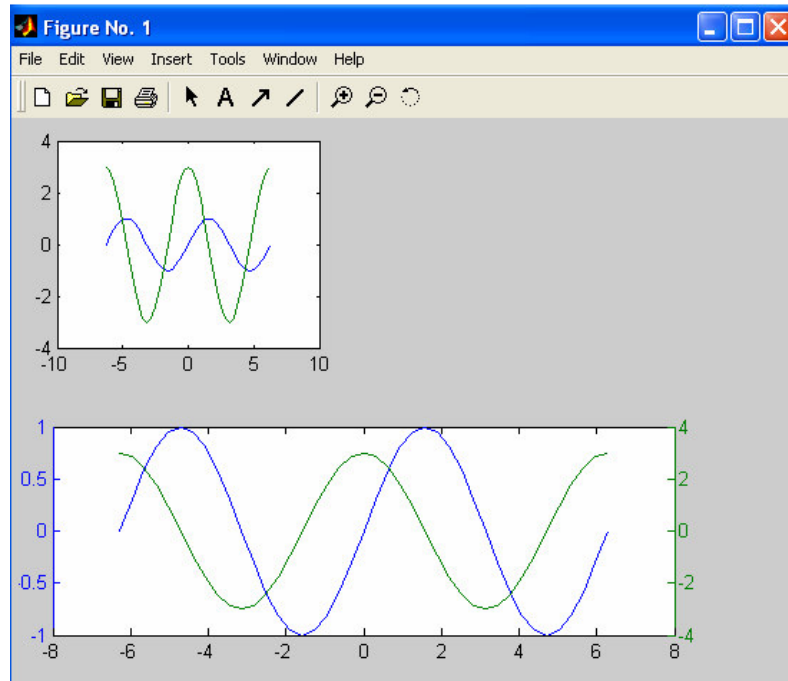
با استفاده از تابع pie(a,b) می توانید نمودارهای استاندارد کیکی ایجاد کنید که در آن **a** برداری از مقادیر و **b** یک بردار منطقی اختیاری است که قاچ یا قاچ هائی که می بایست از نمودار کیکی بیرون آورده شوند، را توصیف می کند.

```
a = [.5 1 1.6 1.2 .8 2.1 ];  
pie (a,a == max(a));
```



برخی مواقع بهتر است دو تابع مختلف را بر روی یک محور مختصات و با استفاده از مقیاس‌های مختلفی روی محور y رسم کنیم، تابع `plotyy` این کار را انجام می‌دهد.

```
x = -2*pi : pi/10 : 2*pi;
y = sin(x);
z = 3*cos(x);
subplot(2,2,1);
plot(x,y,x,z);
subplot(2,1,2);
plotyy(x,y,x,z);
```



پگونه یک فایل تصویر را در Matlab باز کنیم؟

• Matlab می‌تواند فایل‌های گرافیکی با فرمت‌های JPEG, TIFF, GIF, BMP, PNG, HDF, PCX, XWD, ICO, CUR را به عنوان فایل گرافیکی بخواند. مثلاً برای وارد کردن تصویری به نام cameraman.tif به فضای Matlab کافی است از دستور imread استفاده کنیم:

```
MyImage = imread('cameraman.tif','tif');
```

توجه داشته باشید که فایلی که دستور خواندنش را می‌دهید باید برای برنامه قابل دسترس باشد. یعنی یا باید در مسیر (Path) Matlab باشد یا اینکه در پرونده‌ای (folder) قرار داشته باشد که در حال حاضر برنامه به آن دسترسی دارد. برای اینکه بدانید که Matlab برای پیدا کردن فایلی که دستورش را دادید کجا را خواهد گشت این کارها را بکنید: از دستور path برای اینکه بدانید کدام پرونده‌ها جزء مسیر پیش‌فرض Matlab است و از دستور dir برای اینکه بدانید که Current Directory چیست: استفاده کنید.

همچنین می‌توان برای خواندن یک تصویر از آدرس مستقیم محل ذخیره سازی آن استفاده کرد.

```
f= imread('D:\myimages\chestxay.jpg');
```

نکته: اگر؛ در آخر دستور قرار داده نشود، هنگام خواندن تصویر محتوای ماتریس تصویر یعنی f را نیز نمایش خواهد داد.

با تابع size می‌توان ابعاد تصویر خوانده شده، یعنی ماتریس تصویر را مشخص کرد:

```
size(f)
ans=
    1024    1024
```

می‌توان تابع فوق را به فرم زیر نیز استفاده کرد.

```
[M,N]=size(f);
```

دستور فوق تعداد سطرها (M) و ستون‌های تصویر (N) را مشخص می‌کند.

تا اینجا یک فایل تصویر را در محیط Matlab وارد کرده‌ایم. همانطور که می‌دانیم یک تصویر دیجیتال بر روی کامپیوتر در قالب یک ماتریس ذخیره می‌شود. پس MyImage مثل همه متغیرهای Matlab یک ماتریس است. برای اینکه بدانیم فایل خوانده شده از چه فرمتی است (سیاه سفید، یا Gray Scale یا رنگی) می‌نویسیم:

```
imfinfo('cameraman.tif')
```

این دستور را اجرا کنید و ببینید چه می‌نویسد... اما اگر بخواهید بدانید که ماتریس ذخیره شده MyImage از چه نوعی است کافی است بنویسد: whos و لیست متغیرهای مقیم شده در حافظه و نوع و اندازه آنها را ببینید.

چطور تصویر را ببینیم؟

حالا می‌خواهیم تصویر را که در یک ماتریس ذخیره شده است را ببینیم. بنویسید:

```
imshow(MyImage)
```

با استفاده از دستور subplot می‌توانیم دو تصویر را با هم ببینیم:

```
YourImage = imread('tire.tif','tif');
```

```
Figure
```

```
subplot(1,2,1), imshow(MyImage), title('MyImage')
```

```
subplot(1,2,2), imshow(YourImage), title('YourImage')
```

نحو دیگر این تابع به شکل زیر است:

```
imshow (f,G)
```

که f ماتریس تصویر خوانده شده و G تعداد سطوح خاکستری است. مقدار پیش فرض آن ۲۵۶ است.

```
imshow(f,[low high])
```

با استفاده از فرمت زیر:

پیکسل‌هایی که مقادیر آنها کمتر یا مساوی low باشد سیاه و پیکسل‌هایی که مقدار آنها بزرگتر یا مساوی $high$ باشند، سفید نشان داده می‌شوند.

با استفاده از فرمت: `imshow(f,[])` مقدار low برابر مینیمم f و مقدار $high$ برابر ماکسیمم f در نظر گرفته می‌شود. برای تصاویری که دارای محدوده دینامیکی کمی هستند، این روش مناسب تر است.

```
f = imread('c:\matlab7\work\gradient.bmp');
```

مثال:

می‌توان دستور فوق را به صورت زیر نیز نوشت چرا که `c:\matlab7\work\` دایرکتوری فعلی (Current Directory) محسوب شده و نیازی به دادن مسیر نیست.

```
f= imread('gradient.bmp');
```

```
imshow (f,[ ])
```

```
size(f)
```

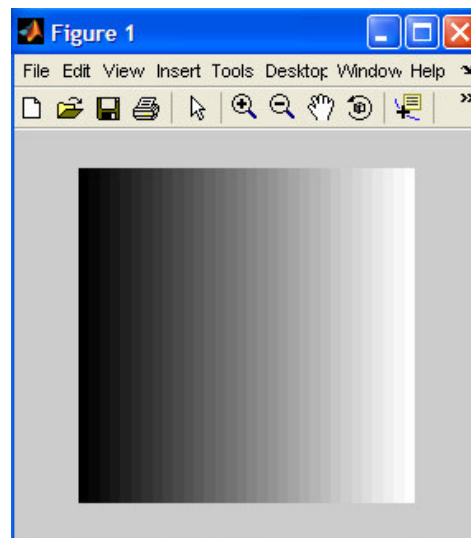
```
ans =
```

```
512 512
```

تصویر نمایش داده شده را می‌توان با استفاده از

Export در منوی File، می‌توان با فرمت

دلخواه ذخیره کرد



نوشتن تصویر

یک ماتریس تصویر با فرمت زیر می تواند به عنوان یک تصویر با فرمت دلخواه ذخیره گردد:

```
imwrite (f, 'filename')
```

فرمت های متداول برای نوشتن تصویر : TIFF, JPEG, BMP, PNG, PGM,

مثال:

```
imwrite (f, 'D:\gradient2.tif')
```

یا

```
imwrite (f, 'D:\gradient2', 'tif')
```

نوشتن بر روی یک تصویر نمایش داده شده:

برای نوشتن متن در خروجی از دستور زیر استفاده می کنیم، موقعیت نمایش متن در صفحه ، در دو پارامتر اول مشخص می شود و متن خروجی در پارامتر سوم مشخص می شود و اما پارامتر پنجم سایز متن و پارامتر هفتم نوع چینش متن را نمایش می دهد. مثال :

```
text(130,240,'Write by Robotic Group in 86/7/10 ',...  
'FontSize',7,'HorizontalAlignment','right')
```

هم چنین هر تصویر را می توان با یک عنوان نشان داد که دستور زیر این کار را انجام می دهد :

```
title('Please click & enter for get a point !');
```

بریدن قسمتی از تصویر

تابع **Imcrop** :

با این تابع می توان قسمتی از تصویر را به صورت (به شکل) مستطیل انتخاب کرده و در واقع یک تصویر دیگر به اندازه تصویر انتخابی ایجاد کنیم، که می توان محدوده ی انتخابی را با ماوس جدا کرد و یا به شکل دیگر این که مختصات محدوده ی انتخابی را در پارامتر های ورودی وارد کرد . هنگامی که imcrop بدون پارامتر است با فشار دادن کلید shift (همزمان با چپ کلیک یا راست کلیک) هنگام انتخاب مستطیل مورد نظر مستطیل به مربع تبدیل می شود .

```
I= imread('ic.tif');
```

```
I1=imcrop(I);
```

```
I2=imcrop(I,[60 40 100 90]);
```

```
I2=imcrop(I,rect) % rect= [x min, y min , width , heath] % مختصات محدوده ی انتخابی
```

دنبال کردن مسیری خاص در تصویر

: Improfile

این تابع در واقع در طول مسیر انتخابی (مسیری که با موس انتخاب می کنید) مقادیر پیکسل های مسیر را به صورت یک بردار به ما نشان می دهد یک نکته مهمی که وجود دارد این است که می توان inter pollution بگیریم، یعنی یک پارامتر ورودی به این تابع بدهیم تا به همان اندازه در مسیر انتخابی، نقاط را در بردار نشان دهد، مثلا اگر improfile(10) بدهیم، ۱۰ نقطه را در این مسیر انتخابی نشان می دهد، در واقع برای این کار از یک الگوریتم مسیر یابی استفاده می کند.

قالب های تصویر

نوع داده ای عددی پیش فرض در Double, Matlab است. این نوع داده به دقت Double یا ۶۴ بیتی است و اعداد نماد علمی را نیز شامل می شود. Matlab برای پشتیبانی از قالب های تصاویر، شامل قالب های داده ای کاراکتری ۱۶ بیتی یا نوع صحیح بدون علامت ۸ بیتی است.

نکته: توابع image و imagesc، می توانند تصاویر ۸ یا ۱۶ بیتی را نمایش دهند بدون این که قالب آنها را به Double تبدیل کنند.

بازه ی مقادیر داده ای برای داده های ۸ بیتی، [0 255] و بازه ی مقادیر داده ای برای داده های ۱۶ بیتی برابر [0 65535] می باشد.

برای تصاویر اندیس گذاری شده، تابع image مقدار ۰ را در اولین مقدار جعبه رنگ ۲۵۶ رنگی قرار می دهد و نیز مقدار ۲۵۵ را به عنوان آخرین مقدار در نظر می گیرد که به طور خودکار از نقطه ی شروع مناسبی استفاده می کند. از آنجایی که بازه ی معمولی داده های Double برای تصاویر اندیس گذاری شده، [1 length(Cmap)] است، بنابراین برای تبدیل داده های ۸ بیتی به Double و یا داده های ۱۶ بیتی به Double و یا برعکس باید مقادیر را به اندازه ی ۱ واحد اضافه کرد.

علاوه بر این، عملگرهای ریاضی، روی آرایه های ۸ بیتی تعریف نشده اند؛ بنابراین برای اجرای عملگر روی مقادیر صحیح بدن علامت می بایست آنها را به قالب Double تبدیل کنید. مثال:

```
Xdouble = double (Xuint8)+1 ;
```

```
Xuint8 = uint (Xdouble -1);
```

برای مثال در این مثال، داده ی ۸ بیتی در متغیر Xuint8 به Double تبدیل می شود و برعکس.

تصاویر متمرک

در Matlab توابع `getframe` و `movie` ابزاری را که برای نمایش و ایجاد تصاویر متحرک لازم باشند را در اختیار شما قرار می دهند.

`Getframe`، یک عکس از شکل جاری می گیرد و تابع `movie` بعد از ساخت فریم ها، آنها را به صورت دنباله ای پشت سر هم نشان می دهد. خروجی تابع `getframe`، ساختمانی است که شامل تمام اطلاعات مورد نیاز برای تابع `movie` می باشد. به مثال زیر توجه کنید:

```
[x, y, z] = peaks(50); → باعث کشیده شدن یک منحنی سه بعدی می شود
surf(x, y, z)
Axis([-3 3 -3 3 -10 10])
Axis vis3d off
Shading interp
colormap(copper)
for i = 1:15
    view(-37.5+15*(i-1), 30)
    m(i) = getframe;
end
Cla % clear axis for movie
movie(m)
```

Axis vis3d ← ضرایب تناسب را به گونه ای حفظ می کند که دوران شکل های سه بعدی را بدون تغییر اندازه ی محورهای مختصات میسر می سازد.

Shading ← سایه رنگ سطح و مسیر شی ها را کنترل می کند.

view ← شکل کلی آن به صورت `view([AZ,EL])` یا `view(AZ,EL)` است. این تابع زاویه ی دید را مطابق با آنچه که یک مشاهده کننده می بیند قرار می دهد. `AZ` ← چرخش در جهت افقی `EL` ← ارتفاع

مقادیر منفی باعث چرخش در جهت عقربه های ساعت می شود.

حالت پیش فرض ← `EL = 30` , `AZ = -37.5`

مقادیر `EL = 90` , `AZ = 0` باعث می شود تصویر مستقیماً از بالا نمایش داده شود.

تابع **getframe**: یک فریم فیلم را برمی گرداند. در واقع این تابع یک تصویر لحظه ای (فوری) از حالت جاری (شکل جاری) می گیرد که معمولاً از این تابع در یک حلقه استفاده می شود.

تابع **movie**: تابع movie(m) یک فیلم را در آرایه‌ی m قرار می‌دهد. m یک آرایه از فریم‌هاست. movie (m,n) فیلم را n بار اجرا می‌کند. اگر n منفی باشد فیلم یک بار به جلو و بار دیگر به عقب نمایش داده می‌شود؛ یعنی اگر n برابر با ۱۰- باشد، فیلم یک بار به جلو و بار دیگر در جهت عکس آن نمایش داده می‌شود تا ۱۰ بار اجرا شود. اگر n یک بردار باشد اولین عنصر تعداد دفعات تکرار است و بقیه‌ی عناصر شامل لیستی از فریم‌ها برای اجرا هستند. به عنوان مثال:

```
n = [10 4 4 2 1]
```

عدد ۱۰ تعداد تکرار را نشان می‌دهد. سپس ابتدا فریم ۴ اجرا می‌شود. دوباره فریم ۴ نمایش داده می‌شود، بعد فریم ۲ و در نهایت فریم ۱ نمایش داده می‌شود.

متغیر **m** شامل آرایه‌ی ساختمانی است که هر کدام از مقادیر آرایه، تنها شامل یک فریم است.

```
m =
```

```
1x15 struct array with fields:
```

```
  cdata
```

```
 colormap
```

```
size (m(1).cdata)
```

```
ans =
```

```
412 369 3
```

قابلیت‌های تصویر

با استفاده از توابع `im2frame` و `frame2im`، می‌توانید تصاویر اندیس‌گذاری شده و فریم‌های تصویر متحرک را به یکدیگر تبدیل کرد. مثال:

```
[X , cmap] = frame2im (m(n))
```

در این مثال n امین فریم از ماتریس تصویر متحرک m به تصویر اندیس‌گذاری شده X و جعبه رنگ cmap تبدیل می‌شود.

```
m(n) = im2frame (x , cmap)
```

در مثال فوق نیز تصاویر اندیس‌گذاری شده X و جعبه ابزار cmap را به n امین فریم از ماتریس تصویر متحرک m تبدیل می‌کند.

توجه داشته باشید که از `im2frame` می‌توان برای تبدیل یک سری از تصاویر ثابت به تصاویر متحرک استفاده کنید؛ شبیه به همان روشی که تابع `getframe` یک سری از figure ها یا axis ها را به تصاویر متحرک تبدیل می‌کند.

برچسب گذاری تصویر

تابع زیر برای برچسب گذاری بر روی یک تصویر سیاه و سفید استفاده می‌شود، یعنی اینکه از سمت چپ بالا رو به پایین حرکت می‌کند و جاهای سفید را که مقدار یک دارند به ترتیب مقادیر ۱ و ۲ و ۳ و ... قرار می‌دهد پس این تابع

هر کدام از تکه های سفید عکس را یک برچسب جدا قرار می دهد . به عنوان مثال سومین توده سفید را با عدد ۳ پر می کند :

```
l= bwlabel(bw);
```

دستور زیر اطلاعاتی در مورد هر یک از برچسب ها به ما می دهد که این اطلاعات در یک استراکچر قرار می گیرد ، پارامتر دوم مشخص می کند که کدام فیلهای این ساختار را لازم داریم اگر All باشد همه فیله ها را در متغیر قرار می دهد، استفاده از فیلد باندینگ باکس موجب می شود که برای هر برچسب یک محدوده چهار گوش به دست آوریم و همین طور باید دقت کنیم که پارامتر اول این تابع یک ماتریسی است که همان خروجی تابع bwlabel است. یعنی برای استفاده از این تابع ابتدا باید تصویر را برچسب گذاری کرده باشیم. در زیر تمام فیلدهای خروجی این تابع معرفی می شوند:

```
Stats = imfeature(1,'BoundingBox');
```

Area	مساحت هر لیبل در این متغیر است.
Centroid	این متغیر مختصات (X و y) نقطه مرکزی برچسب را داراست.
BoundingBox	در این متغیر چهار گوشه برای هر برچسب مشخص می شود.
MajorAxisLength	طول بزرگترین محور در این متغیر است.
MinorAxisLength	طول کوچکترین محور در این متغیر است .
Eccentricity	خروج از مرکز- نا هم مرکزی
Orientation	سو و جهت - تشخیص موقعیت
ConvexHull	برجستگی (تحدب) بدنه
ConvexImage	برجستگی (تحدب) تصویر
ConvexArea	برجستگی (تحدب) مساحت
Image	تصویر یک برچسب را در این ماتریس داریم.
FilledImage	پر شده تصویر برچسب (ماتریس)
FilledArea	پر شده مساحت برچسب (عدد)
EulerNumber	(یک عدد)
Extrema	(یک آرایه دو بعدی)
EquivDiameter	برابری قطری (یک عدد اعشاری)

Solidity	سفتی، سختی - (یک عدد اعشاری کوچکتر از یک)
Extent	گسترده‌گی، دامنه، درجه، مقدار - (یک عدد اعشاری کوچکتر از یک)
PixelList	لیستی از پیکسل‌ها در این ماتریس است.

a= stats(2).BoundingBox

b= stats(3).BoundingBox

d= stats(6).BoundingBox

e= stats(10).BoundingBox

بهینه‌سازی تصاویر

در مبحث بهینه‌سازی تصاویر موارد زیر را بررسی می‌کنیم:

- افزایش شدت نور مربوط به یک تصویر
- انجام عملیات چهارگانه‌ی محاسباتی بر روی تصاویر
- تغییر نمایش تصاویر رنگی به فرمت باینری
- نمایش تصاویر رنگی و افزودن نوار رنگی (یافتن اطلاعات رنگی درباره‌ی هر پیکسل)
- حذف نویز (noise) از تصاویر
- عملیات لبه‌برداری بر روی تصاویر
- الگوریتم‌های لبه برداری
- حذف ناحیه‌ی دلخواه از تصاویر
- پر کردن ناحیه‌ی دلخواه از تصاویر
- فیلتر کردن تصاویر
- طراحی فیلتر
- تعادل کنتراست یک تصویر
- عملیات Thresholding بر روی یک تصویر

افزایش شدت نور یک تصویر به دلایلی مانند بهبود کیفیت تصویر صورت می‌گیرد. عمل اصلی که در این مورد بر روی تصویر انجام می‌شود این است که سطح شدت نور (سطح خاکستری) هر پیکسل با مقدار ثابتی جمع می‌شود، تا مقدار بیشتری پیدا کند و به ناحیه‌ی روشن‌تر نزدیک شود. به این عمل «ارتقاء تمایز» گفته می‌شود و بدین معنی است که شدت نور هر پیکسل را به مقدار ثابتی افزایش می‌دهیم تا در کل شدت نور کل پیکسل‌های تصاویر به میزان یکسان بالا رود و ارتقاء پیدا کند؛ که نتیجه‌ی آن روشن‌تر شدن تصویر است.

در Matlab این امر به کمک تابع `histeq()` صورت می‌گیرد. مثال:

```
I= imread('food.tif');
```


imshow(I)
I2= histeq(I);
Figure
imshow(I2)

با استفاده از دستورات روبرو می‌توان افزایش شدت نور کلی تصویر را مشاهده کرد

همچنین می‌توان با استفاده از تابع `imhist (i)` نمودار مربوط به شدت نور هر یک از این تصاویر را مشاهده کرد.

انجام عملیات چهارگانه محاسباتی بر روی تصاویر:

- ۱- جمع دو تصویر (Adding Images)
- ۲- تفریق دو تصویر (Subtracting Images)
- ۳- ضرب دو تصویر (Multiplying Images)
- ۴- تقسیم دو تصویر (Dividing Images)

توابع محاسباتی

- 1) ippl.....باعث سرعت اجرا
- 2) imadd.....جمع تصاویر
- 3) imsubtract.....تفریق تصاویر
- 4) imdivide.....تقسیم تصاویر
- 5) immultiply.....ضرب تصاویر
- 6) imabsdiff.....قدر مطلق تفاضل تصاویر
- 7) imcomplement.....متمم تصاویر
- 8) imlincomb.....ترکیب خطی تصاویر

IPPL → Intel Performance Primitives Library

بررسی فعال یا غیر فعال بودن ippl

TF= ippl
[TF B]= ippl

توضیح:

- 1) ippl مخصوص شرکت intel است و فقط در دسترس پرسونل های intel است.
- 2) ippl برای یک مجموعه از توابع اساسی که در سیگنال و پردازش تصویر به کار می‌رود، تولید شده است.
- 3) ippl از توابع جعبه ابزار پردازش تصویر است و باعث می‌شود تا توابع محاسباتی با سرعت بیشتری اجرا شود

4) اگر ippl فعال باشد $TF=1$ (True) است و گرنه $TF=0$ (false) است.

TF: اطلاعاتی در مورد پروسسور B

B: آرایه سلولی که هر سطر آن یک رشته است.

5) عملکرد ippl به این صورت است که وقتی فعال است و از توابع محاسباتی مثل (imadd)

استفاده می کنیم به جای اینکه تک تک عناصر آرایه را با هم جمع کند از پردازش موازی استفاده کرده و در هر زمان گروهی از عناصر دو آرایه را با هم جمع می کند و بدین صورت باعث افزایش سرعت در اجرای این توابع می شود.

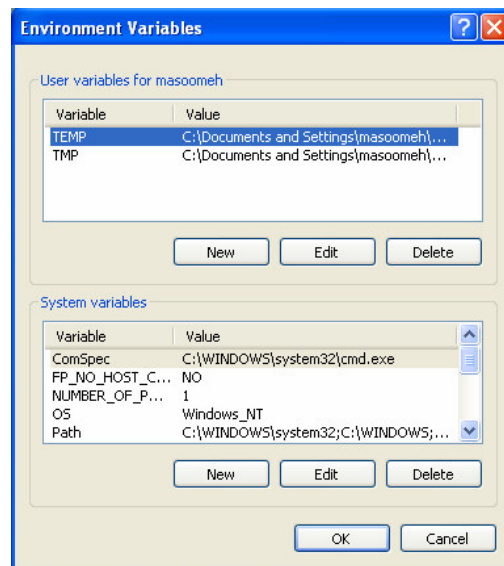
6) وقتی ippl در دسترس است، توابع محاسباتی مانند (imsubtract، immultiply، imlincomp، imadd)

، (imcomplement، imdivide، imfilter) از مزیت Ippل استفاده می کنند.

غیر فعال کردن ippl

برای غیر فعال کردن ippl در قسمت system environment variables یک LPT_IPPL_OFF را تعریف کنید و برای آن ارزش NO قرار دهید.

برای این کار روی پنجره my computer راست کلیک کرده و روی properties کلیک کنید و از پنجره باز شده، پنل advanced را انتخاب کرده و در بخش system variable روی قسمت new وارد کنید.



تابع Imadd

این تابع دو تصویر را با هم جمع می کند یا یک تصویر را با یک عدد ثابت جمع می کند.

$Z = \text{imadd}(x, y)$

$Z = \text{imadd}(x, y, 'w')$

توضیح:

این دستور آرایه x را با آرایه y جمع کرده و نتیجه را در آرایه Z ذخیره می کند.
class تصویر Z باید شبیه کلاس تصویر x است. مگر اینکه x از نوع logical باشد که در این اینصورت Z از نوع Double است.

تصویر y و x باید هم اندازه و هم نوع باشند. حاصل جمع X و y از محدوده مقدار نوع آرایه تصویر خارج نمی شود. مثلا اگر تصویر x و y از نوع Uint8 باشند عناصر در خروجی از محدوده ۰ تا ۲۵۵ خارج نمی شوند. همچنین اگر عناصر در خروجی به صورت اعشاری بدست آید، عدد بدست آمده گرد می شود
اگر x و y هر دو آرایه ای هم کلاس و یکی از کلاسهای logical، uint8، یا single باشد، ippl فعال شده است.
اگر y یک عدد اسکالر از نوع double باشد و x هم کلاس و یکی از کلاس های uint8، uint16 یا Single باشند، ippl فعال شده است.

آرگومان سوم ورودی در تابع Imadd

$Z = \text{imadd}(x, y, 'w')$

w می تواند یکی از Class های زیر باشد:

Double, int8, int16, int32, uint8, uint16, uint32

که آرگومان سوم این تابع کلاس تصویر خروجی را مشخص می کند.

مثال:

```
x= uint8([255 0 75;44 225 100]);
```

```
y= uint8([50 50 50 ;50 50 50]);
```

```
z= imadd(x,y)
```

```
z=
```

```
255 50 125  
94 255 150
```

جمع آرایه x با یک عدد اسکالر y :

```
Z = imadd(x,y)
```

x : آرایه ، y : یک عدد اسکالر، Z : آرایه

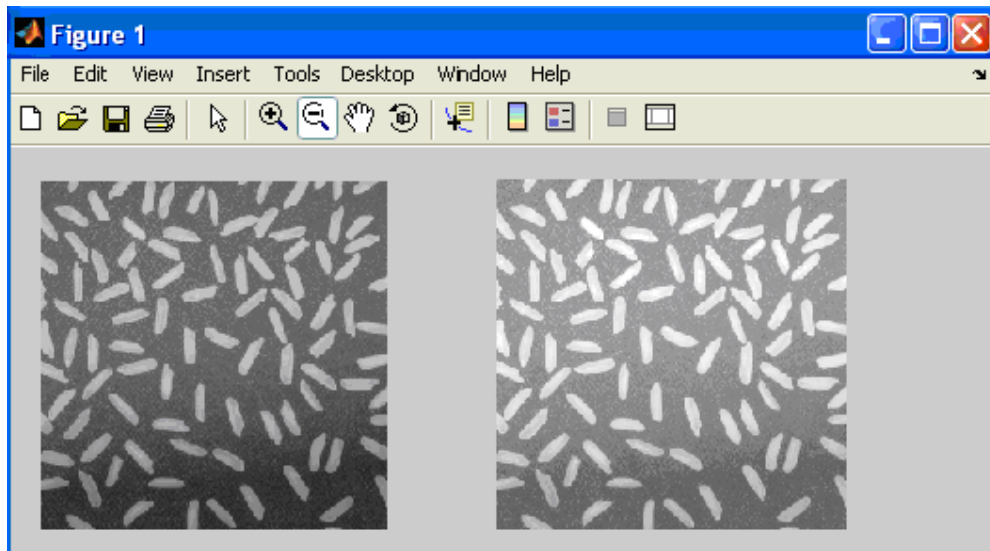
مثال: جمع کردن یک تصویر با یک عدد ثابت

```
I = imread('rice.png');
```

```
J = imadd(I,50);
```

```
subplot(1,2,1), imshow(I)
```

```
subplot(1,2,2), imshow(J)
```



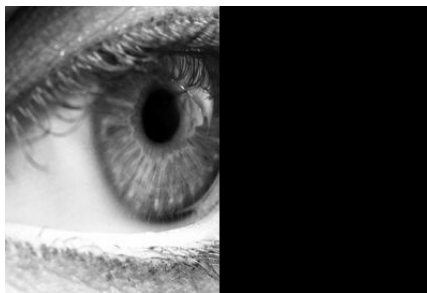
جمع تصویر x با تصویر Y:

$Z = \text{imadd}(x,y)$

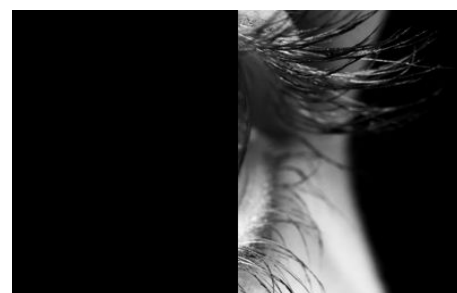
x: آرایه ، y: آرایه ، Z: آرایه

مثال : جمع دو تصویر

```
I = imread(Image1);3
J = imread(Image2);
K = imadd(I,J,'uint16');
imshow(K,[])
```



J



I

K :



تابع `Imsubtract`

یک تصویر را به یک تصویر دیگر تفریق می کند یا یک عدد را از یک تصویر کم می کند.

```
Z= imsubtract(x,y)
```

توضیح:

```
Z= imsubtract(x,y)
```

آرایه x را از آرایه y کم می کند و نتیجه را در آرایه Z می ریزد. تصاویر x و y باید هم اندازه و هم کلاس باشند و Z از محدوده مقدار کلاس تصویر خارج نمی شود.

اگر آرایه x از کلاس `uint8`، `uint16` یا `Single` باشد `ippl` فعال است.

مثال:

```
X = uint8([ 255 10 75; 44 225 100]);
```

```
Y = uint8([ 50 50 50; 50 50 50 ]);
```

```
Z = imsubtract(X,Y)
```

```
Z =
```

```
205  0 25
```

```
 0 175 50
```

تفریق یک عدد از یک تصویر

```
Z=imsubtract(X,Y);
```

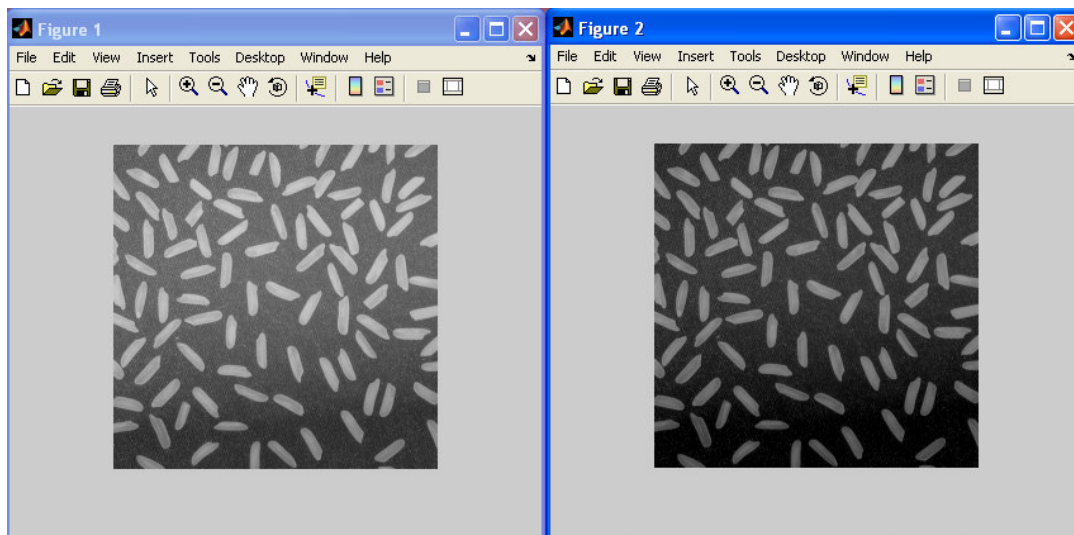
x : تصویر (آرایه) ، y : یک عدد اسکالر ، Z : تصویر (آرایه)

مثال:

```
I = imread('rice.png');
```

```
Iq = imsubtract(I,50);
```

```
figure, imshow(I), figure, imshow(Iq)
```



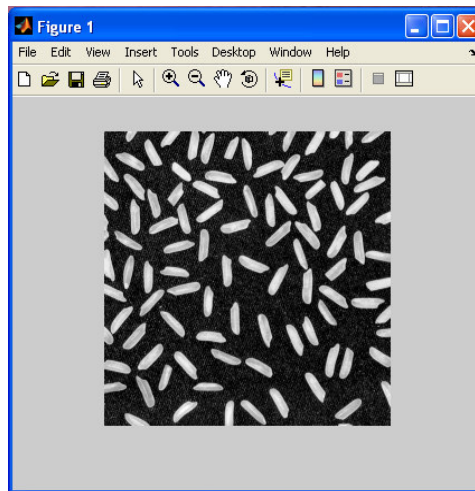
در این حالت یک مقدار عددی از تمام پیکسل‌های تصویر مورد نظر کم می‌شود که در این صورت به دلیل کوچک‌تر شدن ارزش پیکسل‌ها، شدت نور تصویر نهایی کاهش می‌یابد.

تفریق دو تصویر از هم

X: تصویر (آرایه) ، y: تصویر (آرایه) ، Z: تصویر (آرایه)

مثال:

```
I = imread('rice.png');  
background = imopen(I, strel('disk', 15));  
Ip = imsubtract(I, background);  
imshow(Ip, [])
```



در عملیات تفریق تفاضل بین هر یک از زوج پیکسل‌های متناظر دو تصویر $f(x,y)$ و $h(x,y)$ محاسبه می‌شود:

$$g(x,y) = f(x,y) - h(x,y)$$

قسمت‌هایی که برابر نیستند کاملاً روشن ظاهر می‌شوند، زیرا تفریق نمی‌شوند. از کاربردهای این معادله، مسئله‌ی پرتوپردازی حالت نقاب (در مسئله‌ی ارتقاء، مبحثی از تصویربرداری پزشکی) می‌باشد.

تابع **Imdivide**

این تابع دو تصویر را به هم تقسیم می‌کند یا یک تصویر را در یک عدد تقسیم می‌کند. که از این حالت معمولاً جهت کاهش شدت روشنایی یک تصویر استفاده می‌شود.

```
Z = imdivide(x,y)
```

توضیح:

این دستور هر عنصر از آرایه x را به عنصر متناظرش در آرایه y تقسیم می‌کنیم و نتیجه را در آرایه Z ذخیره می‌کند.

$$Z = x ./ y$$

تصاویر x و y باید هم اندازه باشند. اگر حاصل تقسیم یک عدد اعشاری شود، عدد اعشاری گرد می شود و بعد در Z ریخته می شود؛ حاصل تقسیم نیز از محدوده مقدار نوع داده خارج نمی شود.

ippl تنها زمانی که x و y ارایه هایی از کلاس `uint8`، `uint16` یا `single` هستند و سایز و کلاس x و y یکسان باشد، فعال است.

مثال:

```
x = uint8([255 10 75;44 255 100])
y = uint8([50 20 50;50 50 50 ])
z = imdivide(x,y)
z =
     5     1     2
     1     5     2
```

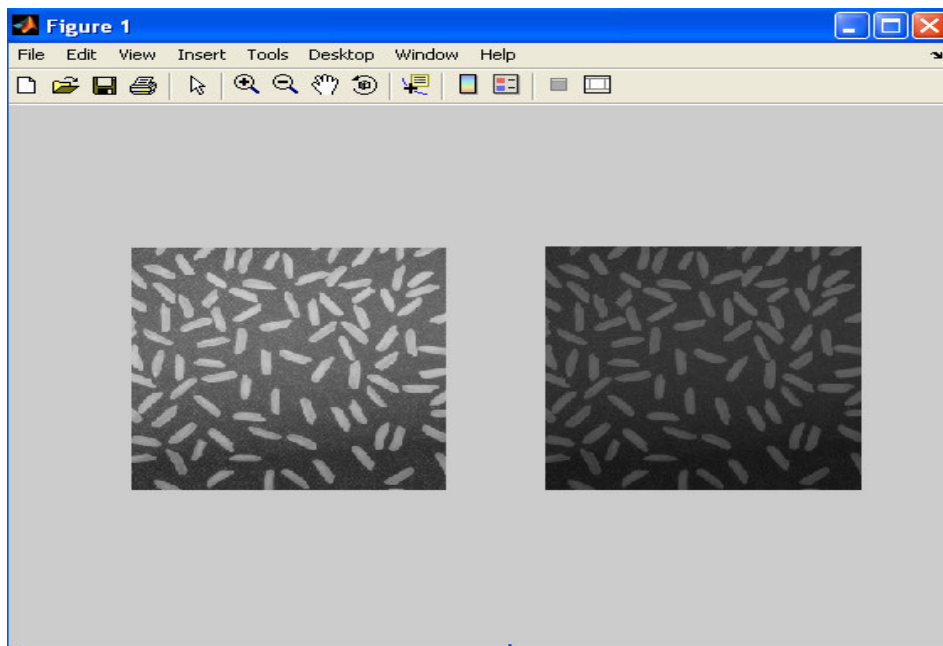
برای تقسیم تصویر در یک عدد

```
Z = imdivide(x,y)
```

x : تصویر ، y : یک عدد اسکالر، Z : تصویر

مثال:

```
I = imread('rice.png');
J = imdivide(I,2);
subplot(1,2,1), imshow(I)
subplot(1,2,2), imshow(J)
```

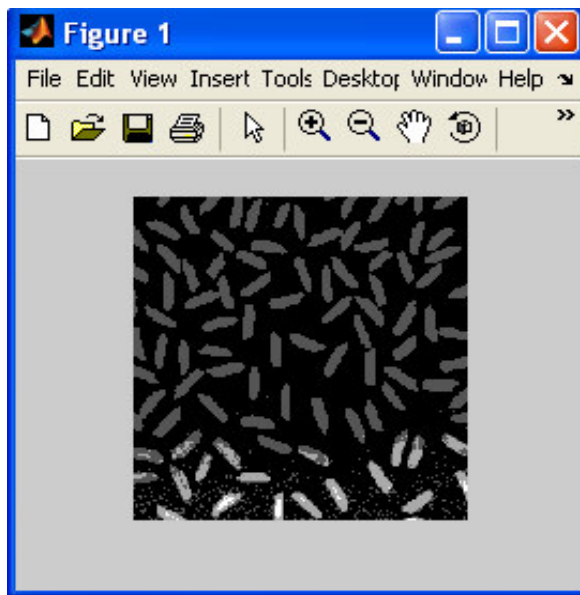


تقسیم یک تصویر بر تصویر دیگر

```
Z= imdivide(x,y)
```

X: تصویر ، y: تصویر، Z: تصویر
مثال:

```
I = imread('rice.png');  
background = imopen(I,strel('disk',15));  
Ip = imdivide(I,background);  
figure, imshow(Ip,[])
```



تابع Immultiply

دو تصویر را در هم ضرب می کند یا یک عدد را در یک تصویر ضرب می کند. معمولاً از این کار جهت افزایش شدت روشنایی یک تصویر استفاده می شود و علت افزایش شدت روشنایی آن است که در ضرب نظیر به نظیر، ارزش تمام پیکسل های هم شماره از هر دو تصویر در یکدیگر ضرب شده و در نتیجه پیکسل های تصویر نهایی ارزش بالاتری خواهند داشت.

```
Z= immultiply(x,y)
```

توضیح:

این دستور، هر عنصر از آرایه x را به عنصر متناظر آن در آرایه y ضرب می کند و نتیجه را در آرایه Z می ریزد.

```
Z= x.* y
```


تصاویر y و x باید سایز و کلاس یکسانی داشته باشند. کلاس Z شبیه کلاس x است. به جز این که:

اگر x از نوع `logical` و y از نوع `numeric` باشد، Z شبیه کلاس y است.

اگر x از نوع `numeric` و y از نوع `logical` باشد، Z شبیه کلاس x است.

Z از محدوده مقدار کلاس خارج نمی شود.

اگر x و y و z کلاس یکسان و از نوع `uint8`، `logical` یا `single` باشد `ippi` فعال است.

مثال:

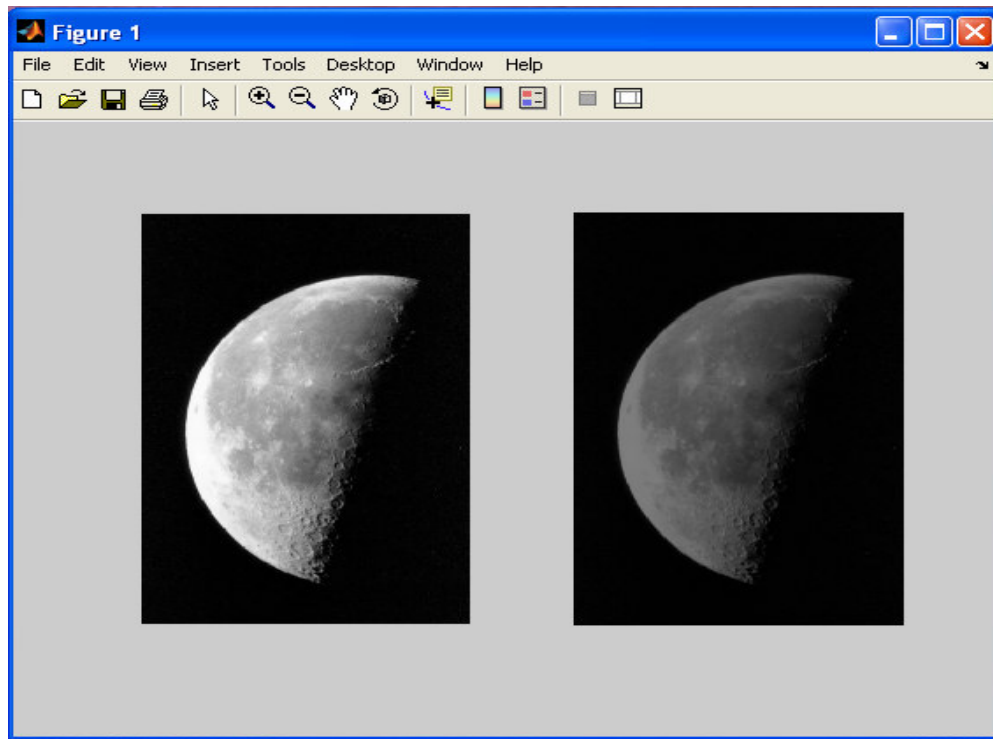
```
X=uint8([1 2 3 ;333 23 33])
Y= uint8([22 33 44;55 55 66])
Z= immultiply(X,Y)
Z=
    22    66   132
    255   255   255
```

ضرب یک عدد در تصویر

x : تصویر (آرایه) ، y : یک عدد اسکالر ، Z : تصویر (آرایه)

مثال:

```
I = imread('moon.tif');
J = immultiply(I,0.5);
subplot(1,2,1), imshow(I)
subplot(1,2,2), imshow(J)
```

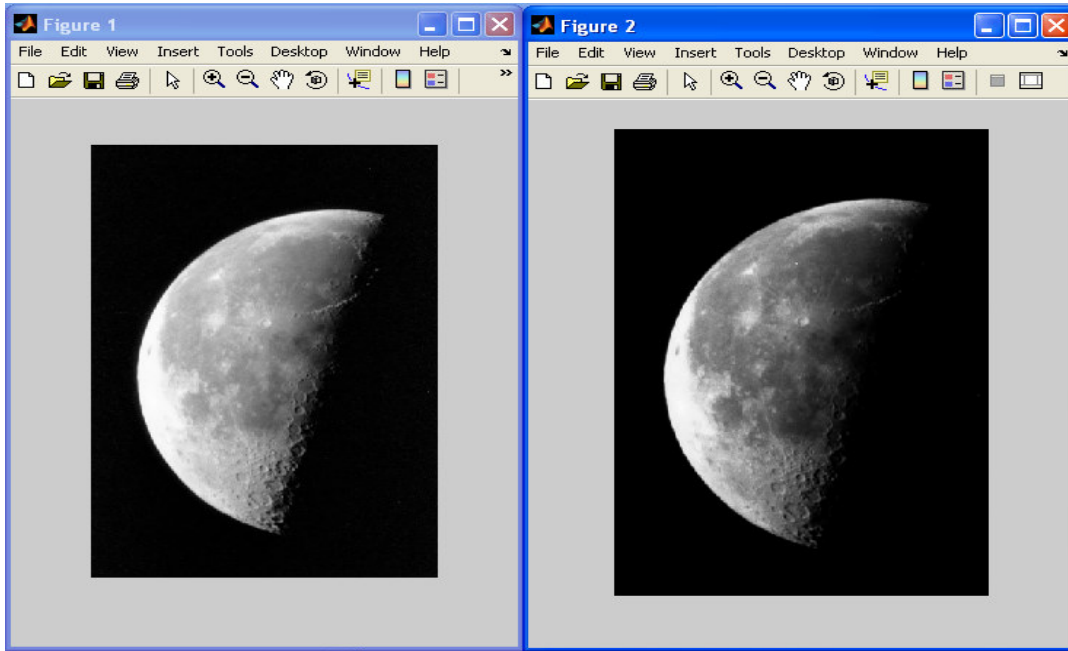


ضرب دو تصویر در هم

X: تصویر ، y: تصویر، Z:تصویر

مثال:

```
I = imread('moon.tif');  
I16 = uint16(I);  
J = immultiply(I16,I16);  
imshow(I), figure, imshow(J)
```



تابع Imabsdiff

قدر مطلق تفاضل دو تصویر

$$Z = \text{imabsdiff}(x,y)$$

توضیح:

خروجی این تابع به این صورت است $Z = \text{abs}(x-y)$

X و Y و Z باید اندازه و کلاس یکسانی داشته باشند. مقدار عناصر خروجی، خارج از محدوده مقدار کلاس تصویر قرار نمی گیرد.

اگر X و Y و Z کلاس یکسان و از یکی از کلاسهای `logical`، `uint8` یا `single` باشد ippl فعال است.

مثال:

```
X = uint8([ 255 10 75; 44 225 100]);
```

```
Y = uint8([ 50 50 50; 50 50 50 ]);
```

```
Z = imabsdiff(X,Y)
```

```
Z =
```

```
205 40 25
```

```
6 175 50
```

قدر مطلق تفاضل دو تصویر

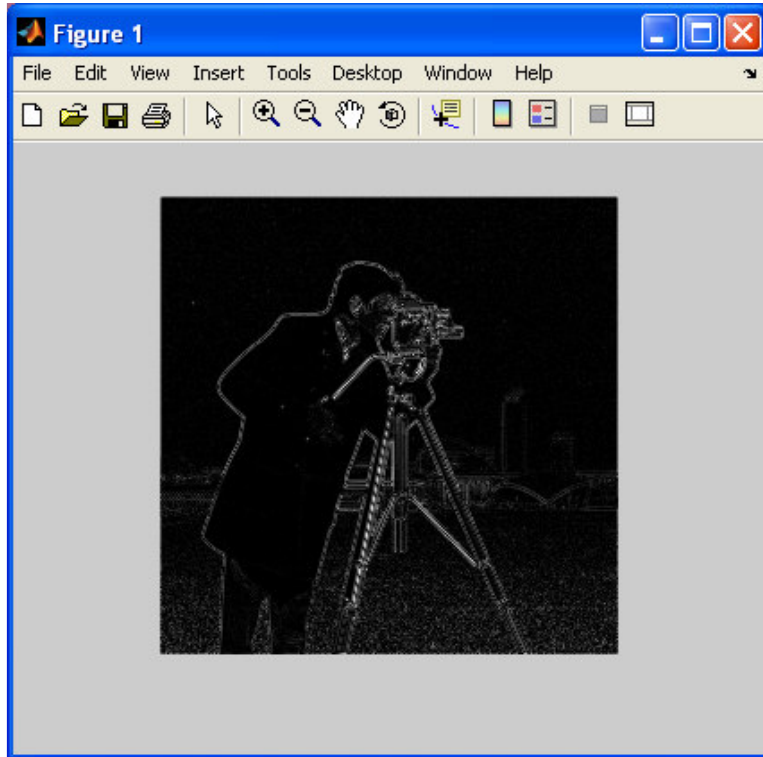
X : تصویر ، Y : تصویر، Z : تصویر

مثال:

```
I = imread('cameraman.tif');
```

```
J = uint8(filter2(fspecial('gaussian'), I));
```

```
K = imabsdiff(I,J);
imshow(K,[]) % [] = scale data automatically
```



تابع imcomplement

متمم تصویر

```
IM2= imcomplement(IM);
```

توضیح:

ورودی تابع از نوع: binary ، grayscale یا RGB

خروجی تابع: تصویر خروجی سایز و کلاسی شبیه تصویر ورودی دارد.

در متمم گیری از تصاویر باینری صفر به یک و یک به صفر تبدیل می شود.

به وسیله این تابع تصویر تیره به روشن و تصویر روشن به تیره تبدیل می شود.

IM2= ~IM اگر تصویر ورودی باینری باشد:

IM2= 1-IM اگر تصویر ورودی grayscale باشد:

IM2= 255-IM اگر تصویر ورودی RGB باشد:

مثال:

```
X= uint([255 10 75;44 225 100]);
```

```
X2= imcomplement(X);
```

```
X2=
```

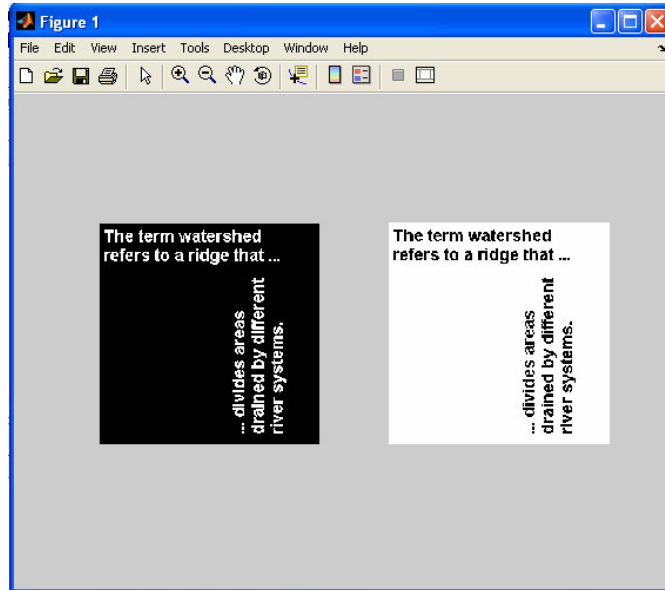
```
0 245 180
```

مثال:

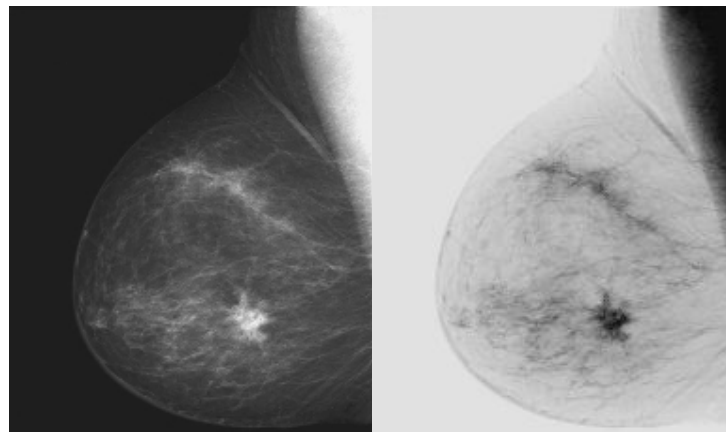
```

bw = imread('text.png');
bw2 = imcomplement(bw);
subplot(1,2,1),imshow(bw)
subplot(1,2,2),imshow(bw2)

```



مثالی از کاربرد این تابع:



شکل سمت چپ یک تصویر ماموگرام است و شکل سمت راست نیز مکمل آن را نشان می‌دهد.

تابع **imlincomb**

ترکیب فطی از تصویر

$$Z = \text{imlincomb}(K_1, A_1, K_2, A_2, \dots, K_n, A_n)$$

$$Z = \text{imlincomb}(K_1, A_1, K_2, A_2, \dots, K_n, A_n, K)$$

$$Z = \text{imlincomb}(\dots, \text{output_class})$$

توضیح:

$Z = \text{imlincomb}(K_1, A_1, K_2, A_2, \dots, K_n, A_n)$ مقدار زیر را محاسبه می‌کند:

$$K_1 * A_1 + K_2 * A_2 + \dots + K_n * A_n$$

Z کلاس و سائیزی شبیه A_1 دارد.

$Z = \text{imlincomb}(K_1, A_1, K_2, A_2, \dots, K_n, A_n, K)$ نیز مقدار زیر را محاسبه می کند:

$$K_1 * A_1 + K_2 * A_2 + \dots + K_n * A_n + K$$

اگر یکی از مورد های زیر برقرار باشد ippl فعال است:

$$Z = \text{imlincomb}(1.0, A_1, 1.0, A_2)$$

$$Z = \text{imlincomb}(1.0, A_1, -1.0, A_2)$$

$$Z = \text{imlincomb}(-1.0, A_1, 1.0, A_2)$$

$$Z = \text{imlincomb}(1.0, A_1, K)$$

وقتی A_1, A_2, K کلاس یکسانی دارند و یکی از کلاس های زیر هستند:

Single, int, 16, uint8

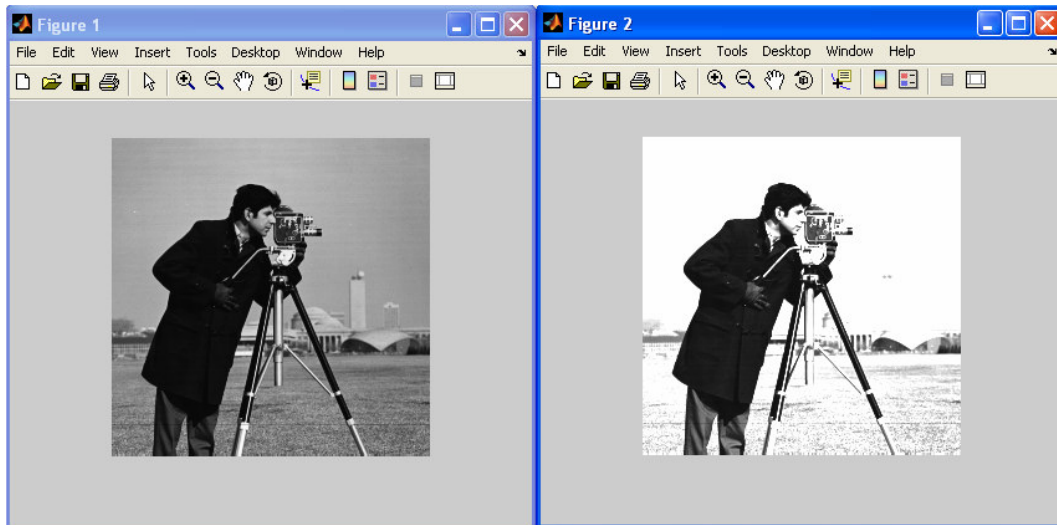
مثال:

```
I = imread('cameraman.tif');
```

```
J = imlincomb(2,I);
```

```
Imshow(I), figure, imshow(J)
```

تصویر I را در عدد ۲ ضرب می کند و نتیجه در تصویر ذخیره می کند.



مثال: حاصل تفاضل دو تصویر را با عدد ۱۲۸ جمع کردن

```
I = imread('cameraman.tif');
```

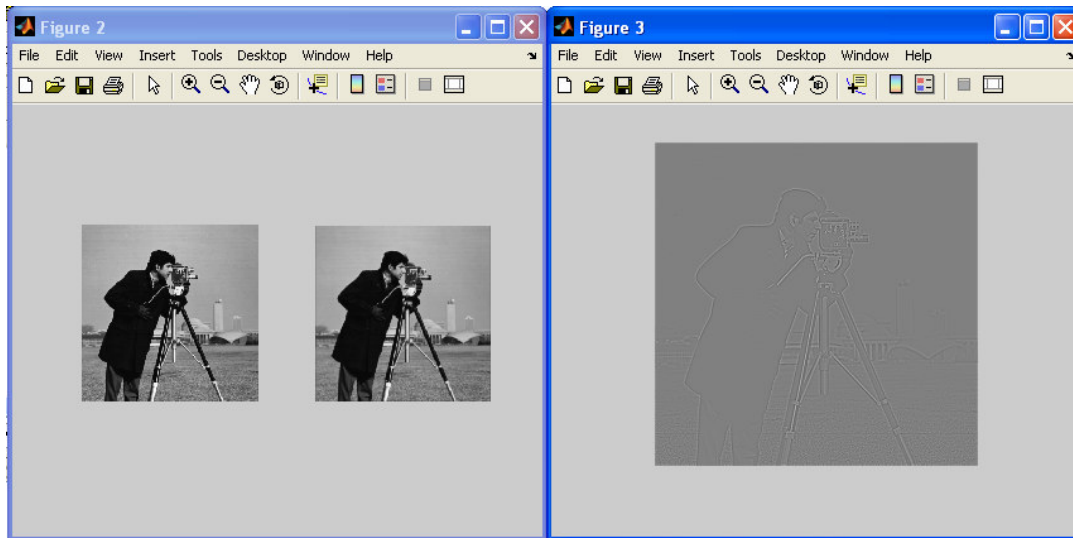
```
J = uint8(filter2(fspecial('gaussian'), I));
```

```
K = imlincomb(1,I,-1,J,128); % K(r,c) = I(r,c) - J(r,c) + 128
```

```
subplot(1,2,1), imshow(I)
```

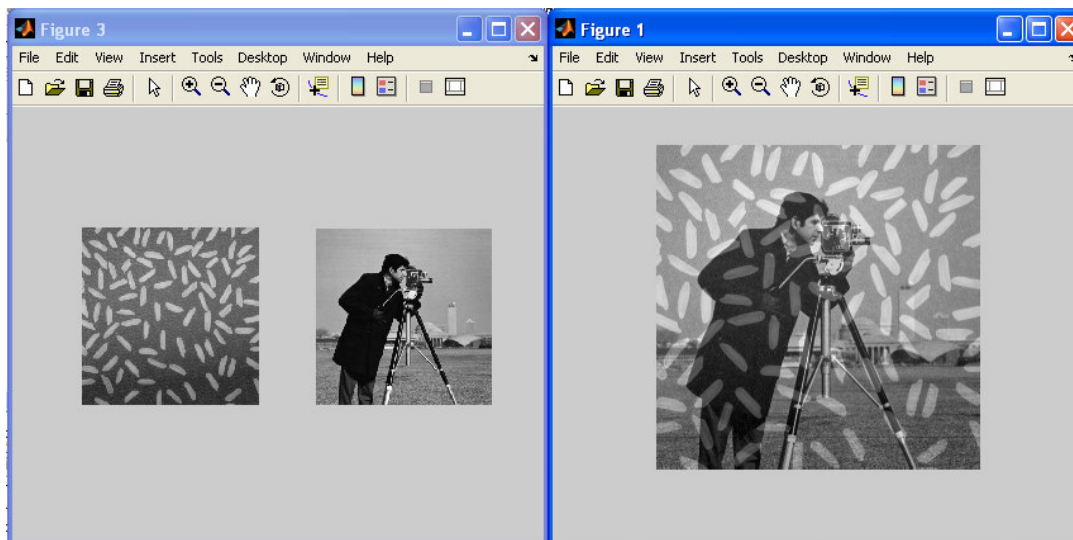
```
subplot(1,2,2), imshow(J)
```

```
figure,imshow(K)
```



مثال: جمع دو تصویر و مشخص کردن کلاس خروجی

```
I = imread('rice.png');
J = imread('cameraman.tif');
K = imlincomb(1,I,1,J,'uint16');
subplot(1,2,1),imshow(I),subplot(1,2,2)
imshow(J),figure, imshow(K,[])
```



مثال:

```
X = uint8([ 255 10 75; 44 225 100]);
Y = uint8([ 50 20 50; 50 50 50]);
Z = imdivide(imadd(X,Y),2)
Z=
 128   15   63
  47  128   75
```

در این مثال ابتدا تابع `imadd` انجام می‌شود و اگر حاصل جمع دو آرایه بزرگتر از ۲۵۵ شود نتیجه ۲۵۵ می‌شود و بعد تابع `imdivide` انجام می‌شود.

در حالی که در مثال زیر، تابع `imlincomb` جمع و تقسیم را در دقت `double` انجام می‌دهد و در آخر اگر نتیجه بزرگتر از ۲۵۵ شده باشد آن را ۲۵۵ می‌کند.

```
Z2 = imlincomb(.5,X,.5,Y) %Z2=0.5X+0.5Y  
Z2=
```

```
153 15 63  
47 138 75
```

نمودار هیستوگرام

در آن تعداد پیکسل‌ها (فراوانی) هر سطح روشنایی در تصویر ورودی مشخص می‌شود اگر نمودار هیستوگرام نمودار چسبیده و فشرده باشد، در این حالت پیکسل‌ها متمرکز و در مرکز هیستوگرام می‌باشند و کنتراست تصویر پایین است و در واقع سطح روشنایی نقاط خوب از هم شناسایی (تفکیک) نشده و کیفیت تصویر کم است. حال اگر نموداری گسسته و باز باشد و در واقع در محدوده‌ی ۰ تا ۲۵۵ پخش شده و در نتیجه تفکیک نقاط به خوبی صورت گرفته و تصویری با کنتراست بالا و کیفیتی خوب داریم. دو تابع برای افزایش کنتراست تصویرهای سایه و سفید و یا `grayscale` وجود دارد :

۱- `imadjust`

۲- `histeq` (تصویر خروجی به یک هیستوگرام پیش فرض تعیین شده `match` می‌شود)

که دومی هیستوگرام تساوی را نمایش می‌دهد.

نکته: برای جلوگیری از `noise` تصویر باید در بالا بردن کنتراست محدودیت قایل شد.

برای افزایش کنتراست تصویرهای رنگی، با دستکاری کردن درخشش و نور تصویر بدون اینکه سرچشمه و رنگ اصلی تصویر از بین برود، این کار را انجام می‌دهیم.

```
makeform(convert to l*a*b)
```

```
a (:, :, 1) = histez(L)*100
```

روی درخشش لایه تصویر کار می‌کند

انتخاب نامیه ی فاصی از تصویر

تابع `Bwselect`

همانطور که از اسم آن پیداست، می‌توانیم از تصاویر سیاه و سفید انتخابی داشته باشیم تا تنها محدوده انتخابی را به ما نشان دهد. به این صورت که با انتخاب نقطه مورد نظر محدوده ای که این نقطه (نقاط) در آن واقع است، به ما نشان می‌دهد، که هم می‌توان برنامه آن را صدا زد تا با موس بتوان نقاط را انتخاب کرد

```
bw2 = bwselect(BW1,N)
```


و هم می توان با دادن دو بردار r, c که یکی نقاط x و دیگری نقاط y را دارد، انتخاب را انجام داد.
 $bw1 = bwselect(bw1, c, r, n)$

که n می تواند ۴ یا ۸ باشد (مؤلفه برای پیوستگی نقاط است)

تغییر نوع تصویر

rgb2ind: تابعی است که یک تصویر rgb را به عنوان پارامتر ورودی گرفته و به یک تصویر $indexed$ (اندیس گذاری شده) تبدیل می کند که در واقع $indexed$ شامل یک آرایه دو بعدی (شامل x, y های تصویر) و یک map که دارای سه ستون است:

ستون اول: شدت رنگ قرمز

ستون دوم: شدت رنگ سبز

ستون سوم: شدت رنگ آبی

$[x, map] = rgb2ind(rgb, n)$

x : آرایه دوبعدی تصویر، map : آرایه سه بعدی، n : ردیف map که می تواند تا 65536 باشد

چند نمونه از map :

black: [0 0 0]
pure red: [1 0 0]
white: [1 1 1]
Gray: [0.5 0.5 0.5]

ind2rgb: عکس تابع قبلی است که این بار یک تصویر $indexed$ (مرتب شده با آرایه) را به یک تصویر rgb تبدیل می کند. که rgb یک آرایه $3 \times n \times m$ از کلاس $double$ است. و در تصویر (x, map) ، x می تواند از کلاس $uint8$ ، $uint16$ و یا $double$ باشد

rgb2hsv: تابعی است که یک تصویر rgb را به یک تصویر با شدت رنگ (hsv) تبدیل می کند که همانطور که قبلاً ذکر شد برای هر پیکسل سه مشخصه را ذخیره می کند: نام رنگ، شدت رنگ، روشنایی یا تیرگی رنگ. با توجه به اینکه تصویر خروجی دوربین ها معمولاً RGB است، اولین کار بعد از دریافت تصویر از دوربین این است که سیستم RGB را به HSV تبدیل کنیم. این کار در نرم افزار Matlab بسیار ساده می باشد و کافی است از تابع $A = rgb2hsv(k)$

استفاده کنیم. در عبارت بالا K تصویر با فرمت RGB و A همان تصویر قبلی با فرمت HSV می باشد. اگر بخواهیم اهمیت این تابع را از دید کاربردی بررسی کنیم مثالی کاربردی در این زمینه ارائه می دهیم. همانگونه که می دانیم یکی از مسائل مهم در پردازش تصویر در رباتیک (به طور مثال در ربات های فوتبالیست) تشخیص اجسام با

استفاده از رنگ آنهاست؛ که این امر با پردازش عکس هایی که ربات از محیط می گیرد انجام می شود، اما به دلیل تغییرات محیطی فراوان، به طور مثال تغییر نور و روشنایی محیط، ممکن است عکس هایی که در یک زمان و با فاصله های زمانی بسیار کم از یکدیگر از یک شیء گرفته می شوند، با یکدیگر متفاوت باشند. برای مثال با تغییر میزان نور محیط رنگ اجسام تیره تر و یا روشن تر می شود و چون فرمت عکس هایی که گرفته می شود RGB است، تشخیص آن جسم دشوار می شود برای برطرف کردن این مشکل از تابع تبدیل RGB به HSV، (RGB2HSV) استفاده می کنیم.

رنگی نمودن تصاویر باینری

عملیاتی وجود دارد که برای رنگی کردن تصاویر باینری استفاده می شود. مثلاً دستور زیر پیکسل های با مقدار صفر را به رنگ قرمز و پیکسل های ۱ را به رنگ آبی تبدیل می کند.

```
I= imread('apples.jpg');
```

```
BW = rgb2gray(I)
```

```
imshow(BW, [1 0 0,0 0 1])
```

← 1 0 0 ← پیکسل های با مقدار صفر ← رنگ قرمز

← 0 0 1 ← پیکسل های با مقدار ۱ ← رنگ آبی

نکته : برای تصاویر intensity می توان تا ۲۵۵ رنگ برای آن تعریف کرد.

حذف نامیه دلفواه از تصاویر:

تابع roifill :

از جمله توابع پر کاربرد و مهم در علم پردازش تصویر است، که از آن جهت حذف ناحیه زائد از تصویر به صورت دلخواه استفاده می شود که برای تصاویر gray scale کاربرد دارد.

دو روش استفاده:

۱- به صورت دستی و به کمک ماوس

۲- انتخاب محدوده دلخواه توسط مقادیر سطر و ستون مناسب

- در روش اول پس از فراخوانی تصویر اصلی با اجرای این تابع و انتخاب محدوده دلخواه توسط رسم یک چند ضلعی به کمک ماوس، محدوده مورد نظر را تعیین کرده و تصویر نهایی که قسمت انتخاب شده از آن ناحیه حذف شده، مشاهده می شود.

```
I= imread('aass.tif');
```

```
roifill(I);
```

تذکر:

۱- تصویری که این تابع به روی آن اعمال می شود، حتما باید به فرمت گری باشد.

۲- در صورت استفاده از روش اول اگر محدوده انتخابی مناسب نبود برای انتخاب محدوده delete و یا دیگر از Back space استفاده می‌شود.

- دومین روش، بدین صورت است که تصویر مورد نظر را فراخوانی کرده و سپس شماره ردیف و ستون های محدوده ای که باید از تصویر اصلی محو شود به صورت بردار داده می‌شود. که نقاط بردار همان نقاط رئوس چند ضلعی انتخابی ما می‌باشد.

Row=[a,b,c,d,...]

Coloumn=[t,y,u,I,...]

J= roifill(I,Coloumn,Row);

این تابع در واقع یک درونیایی درون مرزهای چند ضلعی انتخابی ما انجام می‌دهد تا بتواند ناحیه مورد نظر را انتخاب کرده و محو کند.

نحو دیگر این تابع به این صورت است :

J= roifill(I,BW);

در این دستور، BW یک تصویر باینری به همان سایز تصویر I است که به عنوان ماسک می‌باشد. به این صورت که، نواحی از تصویر I که متناظر با پیکسل های ۱ در تصویر BW می‌باشند، fill می‌شوند(محو می‌شوند).

اگر چندین ناحیه وجود داشته باشد، roifill یک درونیایی روی هر ناحیه به طور مستقل انجام می‌دهد.

[J,BW]= roifill(...)

این دستور، باینری ماسک را برای پیکسل‌هایی که محو شده اند برمی‌گرداند.

در واقع BW برای پیکسل‌هایی که در نواحی درونیایی شده هستند، مقدار عددی ۱ و برای دیگر نقاط مقدار ۰ را جایگزین می‌کند.

نحو دیگری از این تابع به صورت زیر است :

J= roifill(x,y,I,xi,yi)

در این دستور از دو عدد X و Y برای ایجاد یک سیستم مکانی جدید برای محو شدن استفاده می‌کند، به این صورت که محتوای دو بردار xi و yi که رئوس چند ضلعی تعیین شده را تعیین می‌کند، را از این دو عدد کم می‌کند و در مختصات جدید درونیایی را انجام می‌دهد.

تابع roipoly :

تابع دیگری در این زمینه، تابع roipoly می‌باشد. که این تابع هم همان دو روش گفته شده را برای استفاده دارد.

نحوه کار آن تاحدی شبیه تابع roifill می‌باشد، تنها تفاوت میان این دو تابع در این است که در تابع roifill محدوده انتخاب شده از تصویر محو می‌شود بدون اینکه تغییری در سایر قسمت‌های تصویر اصلی ایجاد شود، اما تابع roipoly در واقع محدوده انتخاب شده را به صورت ۱ (رنگ سفید) کنار گذاشته و بقیه محدوده تصویر را به صورت ۰ (رنگ سیاه) نمایش می‌دهد.

در واقع این تابع همان ماسک سیاه - سفید، تابع roifill می‌باشد.

پر کردن نامیه دلفواه از تصویر :

در برخی موارد، جهت پردازش بهتر بر روی تصاویر و افزایش دقت عملیات پردازش لازم است که نواحی و حفره‌های موجود در تصویر را پر کرده و هم سطح محدوده اصلی تصویر کرد، به کمک تابع bwfill می‌توان ناحیه background تصاویر به فرمت binary را پر کرد

$bw2 = bwfill(bw1, c, r, n)$

شروع عملکرد تابع bwfill از پیکسل (r, c) می‌باشد و در صورتی که r و c بردارهایی با طول مساوی باشند، عمل پر کردن به صورت موازی انجام خواهد گرفت، از موقعیت $c(k)$ و $r(k)$.

مقدار عددی n می‌تواند به صورت ۴ یا ۸ باشد که تعیین کننده connectivity می‌باشد. زمانی که عدد ۴ استفاده شود، چهار اتصال در تصویر foreground وجود دارد و در صورتی که عدد ۸ استفاده شود، ۸ اتصال در تصویر foreground وجود دارد.

$bw2 = bwfill(bw1, n)$

این دستور، تصویر را در صفحه نمایش نشان می‌دهد و اجازه می‌دهد که نقطه دلخواه به کمک mouse تعیین شود.

$[bw2, idx] = bwfill(...)$

این دستور مشخصات خطی تمام پیکسل‌های پر شده توسط این تابع را برمی‌گرداند.

$Bw2 = bwfill(bw1, 'holes', n)$

تابع bwfill در واقع تعیین می‌کند که کدام پیکسل‌ها جز حفره‌ها باشند و آنگاه ارزش عددی پیکسل‌های حفره را از صفر به یک تغییر می‌دهد.

پیش فرض مقدار n ، ۸ است.

تصویر ورودی حتماً باید به فرمت عددی یا منطقی باشد و تصویر خروجی به فرمت منطقی خواهد بود.

نویز در تصویر

ما در اینجا دو نویز 'salt&pepper' و 'speckle' را بررسی می‌کنیم ، که این دو نویز به معنای "فلفل و نمک " و "خال " هم می‌باشد.

نویز salt&pepper (فلفل و نمک)

$$J = \text{imnoise}(I, 'salt\&pepper', D)$$

این دستور پیکسل‌های صفر و یک (روشن و خاموش)، (نویز نمک و فلفل) را به تصویر اضافه می‌کند ، و این پیکسل‌های صفر و یک را با تراکم و فشردگی مقدار D ، که در واقع تقریباً روی (تعداد کل پیکسل‌ها $D * D$) پیکسل تصویر تاثیر می‌گذارد؛ که هر چه مقدار این D بیشتر شود، نویز تصویر نیز بیشتر می‌شود و به طور پیش فرض مقدار آن $0/05$ می‌باشد .

نویز speckle (خال)

$$j = \text{imnoise}(I, 'speckle', V)$$

این نوع نویز از یک معادله استفاده می‌کند : $j = I + n * I$

که n یک عدد تصادفی که به طور یکنواخت توزیع می‌شود با میانگین صفر و واریانس V ، که مقدار واریانس به طور پیش فرض $0/04$ می‌باشد .

مذف نویز:

توسط تابع `medfilt2` می‌توان نویز اضافه شده به یک تصویر را حذف نمود .

$$B = \text{medfilt2}(a, [m \ n])$$

که a همان تصویر نویز دار می‌باشد و $[m \ n]$ یک همسایگی اطراف پیکسل مربوطه در تصویر ورودی می‌باشد ، که پیکسل خروجی شامل متوسط مقدار در همسایگی m -by- n اطراف پیکسل ورودی می‌باشد که به طور پیش فرض این همسایگی 3 -by- 3 می‌باشد .

که این تابع لایه گذاری می‌کند، تصویر ورودی را با گذاشتن صفر روی لبه‌ها همچنین متوسط مقدار برای نقاط درون $[m \ n]/2$ از لبه‌ها محاسبه می‌کند .

آشکارسازی لبه

آشکارسازی لبه یکی از مفاهیم پردازش تصاویر است.

هدف آشکارسازی لبه نشان‌گذاری نقاطی از یک تصویر است که در آنها شدت روشنایی به تندی تغییر می‌کند. تغییرات تند در خصوصیات تصویر معمولاً نماینده رویدادهای مهم و تغییرات در خصوصیات محیط هستند. شناسایی لبه یک محدوده تحقیقاتی در پردازش تصویر و استخراج ویژگی است.

ویژگی‌های لبه

لبه‌ها ممکن است وابسته به دیدگاه باشند - یعنی می‌توانند با تغییر نقطه دید تغییر کنند، و نوعاً هندسه صحنه، اجسامی که جلوی همدیگر را گرفته‌اند و مانند آن را نشان می‌دهند یا ممکن است وابسته به دیدگاه باشند - که معمولاً نمایانگر ویژگی‌های اجسام دیده‌شده همچون نشان‌گذاری‌ها و شکل سطح باشند. در دو بعد و بالاتر مفهوم تصویر باید در نظر گرفته شود.

یک لبه نوعی ممکن است (برای نمونه) مرز میان یک بخش قرمز رنگ و یک بخش سیاه رنگ باشد؛ حال آنکه یک خط می‌تواند تعداد کمی پیکسل‌های ناهم‌رنگ در یک زمینه یکنواخت باشد. در هر سوی خط یک لبه وجود خواهد داشت. لبه‌ها نقش مهمی در کاربردهای پردازش تصویر دارند.

آشکارسازی لبه

لبه مرز بین نواحی با خواص نسبتاً متفاوت سطح خاکستری است. نظریه‌ی پایه در بیشتر روش‌های آشکارسازی لبه، محاسبه یک عملگر مشتق محلی است. در این مقطع توجه شود که لبه (گذر از تاریک به روشن) به صورت یک تغییر آرام، نه سریع، سطح خاکستری مدل می‌شود. این مدل نشان می‌دهد که معمولاً لبه‌های تصاویر رقمی بر اثر نمونه‌برداری، کمی مات می‌شوند. مشتق اول مقطع سطح خاکستری در لبه جلویی گذر، مثبت است، در لبه عقبی آن منفی است و همان‌طور که مورد انتظار است، در نواحی با سطح خاکستری ثابت صفر است. مشتق دوم برای قسمتی از گذر که در طرف تیره لبه است، مثبت است، برای قسمت دیگر گذر که در طرف روشن لبه است، منفی است، و در نواحی با سطح خاکستری ثابت، صفر است. بنابراین، از بزرگی مشتق اول می‌توان برای تعیین این که آیا پیکسل در روی لبه قرار دارد، استفاده کرد. مشتق دوم در نقطه وسطی هر گذر سطح خاکستری یک عبور از صفر دارد. عبور از صفرها راهی قوی برای تعیین محل لبه‌های تصویر فراهم می‌آورند. اندازه‌ی مشتق اول تصویر در هر نقطه برابر بزرگی گرادیان است. مشتق دوم نیز با استفاده از لاپلاسین به دست می‌آید. اگر یک لبه را به عنوان تغییر در شدت روشنایی که در طول چند پیکسل دیده می‌شود در نظر بگیریم، الگوریتم‌های آشکارسازی لبه به طور کلی مشتقی از این تغییر شدت روشنایی را محاسبه می‌کنند. برای ساده‌سازی، به آشکارسازی لبه در یک بعد می‌پردازیم. در این نمونه، داده‌های ما می‌تواند یک تک خط از شدت روشنایی پیکسل‌ها باشد. برای نمونه بین پیکسل‌های چهارم و پنجم در داده‌های ۱- بعدی زیر به روشنی می‌توان لبه‌ای را آشکار کرد.

۱۴۹	۱۴۸	۱۵۲	۴	۶	۷	۵
-----	-----	-----	---	---	---	---

محاسبه مشتق اول

تعداد زیادی از عملگرهای آشکارسازی لبه بر پایه مشتق اول شدت روشنایی کار می‌کنند، یعنی با گرادیان شدت روشنایی داده‌های اصلی سروکار داریم. با این اطلاعات می‌توانیم تصویری را برای قله‌های گرادیان روشنایی جستجو کنیم.

اگر $I(x)$ نماینده شدت روشنایی پیکسل x ، و $I'(x)$ نماینده مشتق اول (گرادیان شدت روشنایی) در پیکسل x باشد، بنابراین داریم:

$$I'(x) = -1 \cdot I(x-1) + 0 \cdot I(x) + 1 \cdot I(x+1).$$

برای پردازش تصویر با عملکرد بهتر، مشتق اول را می‌توان (در یک بعد) با پیچش دادن دادن با ماسک زیر بدست آورد:

۱	۰	-۱
---	---	----

مماسبهٔ مشتق دوم

برخی دیگر از الگوریتم‌های آشکارسازی لبه بر اساس مشتق دوم شدت روشنایی کار می‌کنند که در واقع نرخ تغییرات گرادیان شدت روشنایی است و برای آشکارسازی خط‌ها بهترین است، زیرا بدانگونه که در بالا گفتیم هر خط یک لبه دوگانه است، بنابراین در یک سوی خط یک گرادیان روشنایی و در سوی دیگر گرادیان مخالف آن دیده می‌شود. پس می‌توانیم منتظر تغییر بسیار زیاد در گرادیان شدت روشنایی در محل یک خط باشیم. برای یافتن خط‌ها می‌توانیم گذر از صفرهای تغییر گرادیان را در نتایج جستجو کنیم.

اگر $I(x)$ نمایشگر شدت نور در نقطه x و $I''(x)$ مشتق دوم در نقطه x باشد:

$$I''(x) = 1 \cdot I(x-1) - 2 \cdot I(x) + 1 \cdot I(x+1).$$

اینجا نیز بیشتر الگوریتم‌ها از یک ماسک پیچش برای پردازش سریع داده‌های تصویر سود می‌برند:

۱	-۲	۱
---	----	---

عملگرهای آشکارسازی لبه

- مرتبه نخست: چلیپای رابرتز، پرویت، سوبل، کنی، اسپیسک
- مرتبه دوم: لاپلاسی، مار-هیلدرث

اکنون، عملگر کنی و پس از آن مار-هیلدرث بیشترین کاربرد را دارد. عملگرهای زیادی تاکنون منتشر شده‌اند اما هیچیک برتری قابل ملاحظه‌ای بر عملگر کنی در شرایط کلی نداشته‌اند. کار بر روش‌های چندمقیاسی هنوز بیشتر در آزمایشگاه‌هاست.

عملیات لبه برداری بر روی تصاویر:

آشکار سازی لبه (edge detection) معمولاً 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 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 شوند .

```
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 تصویر اولی بر می گرداند ، که جاهایی که تابع لبه ها را در تصویر پیدا می کند ، در تصویر خروجی ۱ می کند و جاهای دیگر را ۰ قرار می دهد .

برخی از الگوریتم های لبه برداری :

- ۱- الگوریتم sobel
- ۲- الگوریتم canny
- ۳- الگوریتم Roberts
- ۴- الگوریتم prewitt
- ۵- الگوریتم zero-cross

الگوریتم sobel :

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 I ، max است .

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

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

این تابع (edge) از همه لبه هایی که قویتر (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، direction جهت را مشخص می کند، یعنی رشته ای است که مشخص می کند که این تابع لبه های افقی یا عمودی و یا هر دو را جستجو کند. که به طور پیش فرض هر دو را جستجو می کند.

'horizontal' افقی:

'vertical' عمودی:

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

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

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

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

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

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 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 نویزها را می خورد و از بقیه روش ها بهتر است.

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

این متد یک بردار 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 (0.4×thresh) به عنوان آستانه پایین در نظر گرفته می‌شود، و اگر هیچ عددی را برای thresh انتخاب نکنیم، تابع edge خود به طور اتوماتیک هر دو المنت را انتخاب می‌کند.

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

دستور بالا یک بردار دو المنتی را بر می‌گرداند که میزان آستانه بالا و پایین را مشخص می‌کند.

معرفی پند تابع

● برای گرفتن هر نقطه از تصویری که در خروجی نمایش داده می‌شود می‌توان از تابع زیر استفاده کرد توجه شود که نقطه مورد نظر توسط کلیک ماوس مشخص شده و برای ورود آن باید کلید Enter را نیز فشار داد :

[X,Y] = getpts(GCF);

● **impixel** : معمولاً برای تعیین مقادیر رنگی مربوط به یک پیکسل از این تابع استفاده می‌شود، که این تابع یک تصویر به عنوان ورودی می‌گیرد و خروجی آن مقادیر رنگ‌های قرمز، سبز و آبی در آن پیکسل است که این پیکسل نیز توسط کاربر انتخاب می‌شود. یکی از اشکال استفاده از این تابع به صورت زیر است:

Z = imread ('Imag');

P= impixel (Z);

باید توجه داشت که قبل از استفاده از این تابع حتماً باید تصویر مورد نظر فراخوانی شده باشد. این تابع تصویر را روی صفحه نمایش نشان می‌دهد و به شما امکان استفاده از ماوس برای انتخاب نقطه‌ی مورد نظر را می‌دهد. به این صورت که به کمک ماوس نقاطی از تصویر را انتخاب کرده و با زدن کلید Enter، مقادیر رنگی مربوط به R، G و B پیکسل‌های آن نقطه در خروجی دریافت خواهد شد. در واقع توسط این تابعی توان ارزش رنگی هر پیکسل از تصویر را به دست آورد. که با استفاده از دو کلید Backspace و Delete، می‌توانید به قسمت انتخابی قبلی بازگردید.

اگر تصویر اصلی به صورت سیاه - سفید باشد و این تابع بر روی آن اجرا شود، مقادیر R، G و B مربوط به رنگ سیاه، هر سه مقدار عددی صفر و برای رنگ سفید مقدار عددی ۲۵۵ را نمایش می‌دهد.

● **pixval** : این تابع اطلاعاتی درباره‌ی پیکسل‌های (نقاط) انتخابی که با ماوس آن نقاط را انتخاب کرده ایم، ارائه می‌دهد و حتی فاصله بین دو نقطه از تصویر را نیز می‌توان با انتخاب این دو نقطه در روی تصویر به دست آورد. این تابع همچنین اطلاعاتی راجع به رنگ نقاط ارائه می‌دهد.

● **imfill**: این تابع برای تصاویر سیاه و سفید کاربرد دارد که باعث می‌شود با انتخاب نقطه‌ای خاص در محدوده (segment) مشخص و کلیک روی آن نقطه، رنگ آن نقطه به رنگ محدوده‌ی مشخصی که حاوی آن نقطه است درآید. به عنوان مثال با استفاده از این تابع با کلیک کردن بر روی نقاط سیاه درون یک دایره‌ی سفید، رنگ آن نقاط به رنگ دایره (سفید) در می‌آید.

```
BW1 = imread('rc.tif');  
BW2 = imfill (BW1);
```

تهیه شده توسط گروه رباتیک دانشگاه پیام نور مرکز قم