Introducción á Toolbox de Imaxe de Matlab

Extraído do Proxecto Fin de Carreira:
"Curso Práctico de Procesado de Imaxe Usando Matlab"
Autor: Roberto Carlos Montoto Díaz.
Tutor: Fernando Martín Rodríguez.

1.- Introducción

O material presente neste tutorial está pensado para que sexa usado polo alumno da materia de Procesado de Imaxe. Nel atopará unha serie de fundamentos teórico-prácticos para poder utilizar Matlab como unha ferramenta apta para o procesado de imaxe e aprenderá a usar algunhas das funcións básicas de procesado de imaxe que lle permitirán abordar con éxito a realización das prácticas da materia.

Neste documento veremos algúns fundamentos esenciais para poder facer procesado de imaxe con Matlab. Falaremos da representación das imaxes en Matlab, dos tipos de datos, dos tipos de imaxes e como traballar con elas. Veremos tamén algunhas das funcións mais importantes do toolbox de imaxe e para que se usan. Falaremos de operacións xeométricas (cambios de tamaño da imaxe, rotacións, recortes), operacións de veciñanza, filtrado lineal e non lineal, deseño de filtros , a transformada de Fourier e as súas aplicacións, análise e realzado de imaxes (histogramas, axuste de intensidade, ecualización de histograma, eliminación de ruído usando filtrado lineal e filtrado de mediana) e operacións morfolóxicas (erosión, dilatación e etiquetado). Con todo esto teremos feita unha presentación bastante profunda do toolbox de imaxe de Matlab 5, e o alumno disporá dunha base teórico-práctica importante que lle permitirá coñecer a fondo o entorno de traballo (Matlab 5) e a súa ferramenta principal (o toolbox de imaxe).

2.-Matlab versión 5

Neste apartado veremo-las principais características da ferramenta que eliximos para a realización das Prácticas de Procesado de Imaxe : Matlab 5.

En concreto veremos con certo detalle o Toolbox de Imaxe, que ven sendo un conxunto de funcións para Procesado de Imaxe e tamén a interface de Matlab con C (API Matlab-C), ferramenta que nos permitirá realizar outras funcións aptas para traballar con imaxes que non veñen incluídas no toolbox de imaxe.

2.1.-Toolbox de Imaxe

Neste apartado verémo-los distintos tipos de imaxes soportados e como son representados por Matlab. Tamén veremos algúns conceptos fundamentais e algunhas das funcións máis usuais do procesado de imaxe.

2.1.1.-As imaxes en Matlab

A estructura básica de datos en Matlab é o array, un conxunto ordenado de elementos reais ou complexos. O array é por tanto un obxecto axeitado para a representación de imaxes, que serán matrices reais con datos de cor e intensidade.

Matlab representa a maioría das imaxes como arrays bidimensionais (matrices) no que cada elemento da matriz representa un píxel. Por exemplo, unha imaxe de 200 filas e 300 columnas de puntos de distintas cores será representada por Matlab como unha matriz de 200x300. Algunhas imaxes, como as RGB, requiren un array tridimensional, onde cada plano da dimensión representa a información de intensidades de vermello, verde e azul respectivamente.

Este convenio fai que traballar con imaxes en Matlab veña sendo como traballar con calquera tipo de matrices o que nos proporciona toda a facilidade que Matlab ofrece para manipula-los datos.

2.1.1.1.-Tipos de datos

Por defecto, Matlab almacena a maioría dos datos en arrays de tipo *double* . Os datos nestes arrays son representados como números de 64 bits en coma flotante. Tódalas funcións e capacidades de Matlab traballan con este tipo de arrays.

Pero para o procesado de imaxe esta representación dos datos non sempre será a máis axeitada. Supoñamos unha imaxe 1000x1000 píxels. Se optasemos pola representación anterior, esta imaxe ocuparía ¡¡ 8 megabytes!!. Para reduci-los requerimentos de memoria, Matlab soporta o almacenamento de imaxes en arrays de tipo *uint8* (enteiro sen signo de 8 bits) e *uint16* (enteiro sen signo de 16 bits)

característica moi importante non soportada pola versión anterior de Matlab e que foi unha das principais razóns polas que se optou por Matlab 5 para a realización deste traballo. Haberá que ter isto moi en conta á hora de programar ficheiros MEX xa que tratar arrays de tipo *double* cando son de tipo *uint8* é un erro moi frecuente que pode ser evitado se comprobamos primeiro o tipo de datos que estamos manexando.

2.1.1.2.-Tipos de imaxes en Matlab

O toolbox de Imaxe de Matlab 5 soporta 4 tipos básicos de imaxes:

. Imaxes indexadas.
. Imaxes en intensidade ou en escala de grises.
. Imaxes binarias.
. Imaxes RGB.

Neste apartado veremos como Matlab e o toolbox de Imaxe representan cada un destes tipos de imaxes.

2.1.1.2.1.- Imaxes indexadas

Unha imaxe indexada consiste nunha matriz, *X* (matriz-imaxe), e un mapa de cor, *map*. *map* é un array *m*x3 de tipo *double* que contén números reais en coma flotante de 64 bits cun rango [0 ,1]. Cada fila de *map*, representa os componentes vermello, verde e azul dunha cor concreta. Cada elemento da matriz *X* representa un pixel da imaxe e o seu valor indica un elemento do array *map* . Polo tanto a cor de cada pixel da imaxe é determinada usando o correspondente valor de *X* como un índice do array *map*. O valor 1 apunta á primeira columna de *map* , o valor 2 apunta á segunda columna , e así sucesivamente.

A seguinte figura ilustra a estructura dunha imaxe indexada.

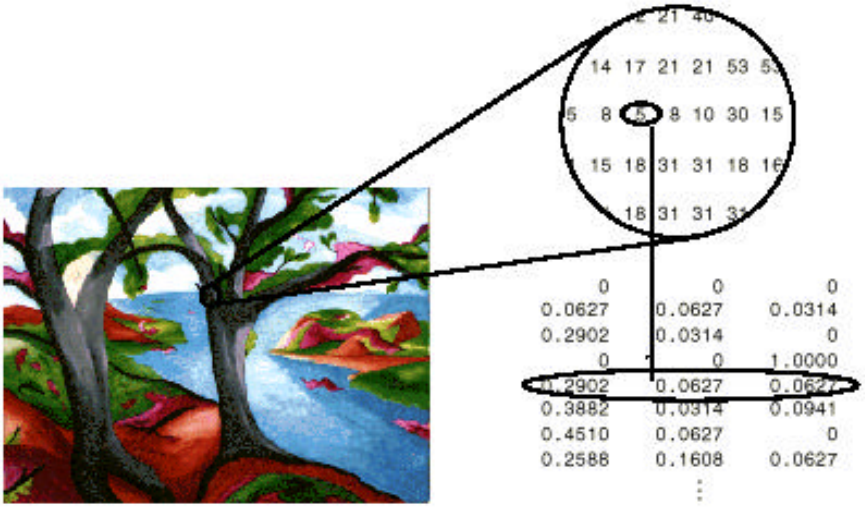

Figura 2.1. Imaxe indexada.

Os píxels na imaxe están representados por enteiros, que son punteiros (índices) a cores almacenadas en *colormap*. A relación entre os valores na matriz *X*, e no array *colormap* dependen do tipo que sexa a matriz *X*.

.

Se a matriz-imaxe é de tipo *double*, o valor 1 apunta á primeira fila do *colormap,* o valor 2 á segunda e así sucesivamente. Se a matriz-imaxe é de tipo *uint8* ou *uint16*, o valor 0 apunta á primeira fila, o valor 1 apunta á segunda fila e así sucesivamente. Na imaxe anterior a matriz-imaxe é de tipo double e polo tanto o valor 5 apunta á quinta fila do *colormap.*

O máis usual, e o máis recomendable, en imaxes indexadas é que a matriz-imaxe sexa de tipo *uint8*, e que o *colormap* sexa de 256 filas. Ademais, a maioría de funcións do toolbox de imaxe operan con imaxes neste formato.

2.1.1.2.2.- Imaxes en escala de grises

Unha imaxe en escala de grises é representada en Matlab 5, por unha matriz *I*, na que os seus valores representan intensidades de gris. A matriz pode ser de tipo *double, uint8* ou *uint16.* O valor 0 (cero) de intensidade normalmente representa o negro e o máximo (1,255,65535) normalmente representa o branco (segundo a matriz *I* sexa de tipo *double, uint8* ou *uint16).*

A figura seguinte representa unha imaxe en escala de grises de tipo *double.*

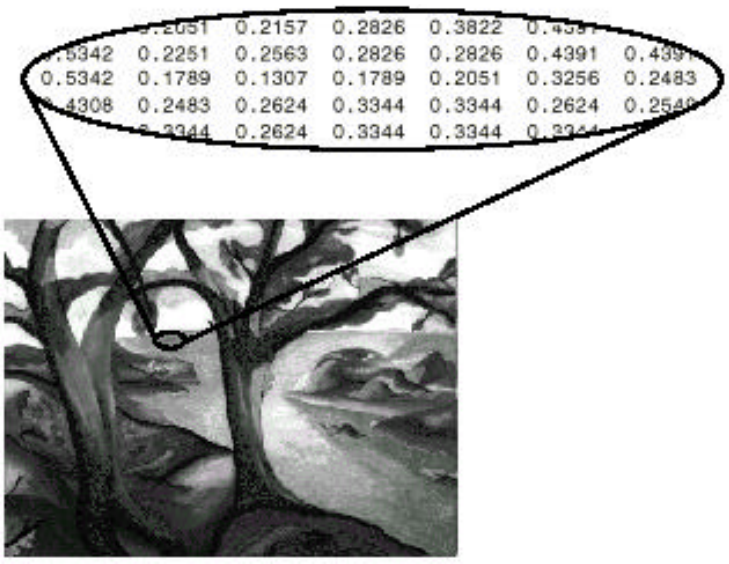

Figura 2.2. Imaxe en escala de grises.

O máis usual é toparse con imaxes en escala de grises representadas por matrices de tipo *uint8* e polo tanto con 256 niveis de gris.

2.1.1.2.3.-Imaxes binarias

Nunha imaxe binaria cada píxel pode ter so dous valores, normalmente 0 (negro) ou 1 (branco). Unha imaxe binaria é almacenada como unha matriz bidimensional de ceros e uns.

As imaxes binarias poden gardarse en arrays de tipo *double ou uint8*. Un array de tipo *uint8* é xeralmente preferible a un array de tipo *double* porque o primeiro usa menos memoria. No toolbox de imaxe de Matlab 5 calquera función que devolva unha imaxe binaria devólvea como un array lóxico de tipo *uint8.* O toolbox usa un flag para indica-lo rango dos datos dun array lóxico de tipo *uint8.* Se o flag está en "on" o rango [0,1]; se o flag está en "off" Matlab asume que o rango é [0,255].

A figura seguinte mostra unha imaxe binaria.

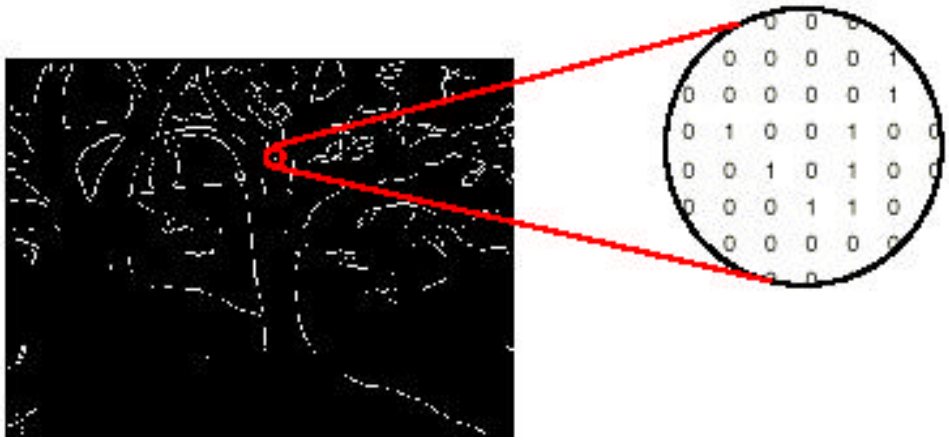

Figura 2.3. Imaxe binaria

2.1.1.2.4.- Imaxes RGB

Unha imaxe RGB, tamén chamadas imaxes en cor verdadeira (truecolor), son almacenadas en Matlab como un array *m*x*n*x3 que definen os componentes vermello, verde e azul de cada píxel. As imaxes RGB non usan unha paleta de cor, é dicir un *colormap*, como as indexadas. A cor de cada pixel ven determinada pola combinación das intensidades vermella , verde e azul gardadas en cada plano de cor na posición do píxel. As imaxes RGB representan cada píxel con 24 bits, oito bits para cada componente de cor, polo tanto o rango é de 16 millóns de cores.

Un array que represente a unha imaxe RGB pode ser de tipo *double, uint8* ou *uint16*. Nun array RGB de tipo *double*, cada componente de cor é un valor entre cero e un. Un píxel no que as súas componentes de cor sexan (0,0,0) será un punto negro e se fosen (1,1,1) sería un punto branco. As tres componentes de cor para cada pixel son gardadas deste xeito. O vermello no primeiro plano, o verde no segundo e o azul no terceiro. Por exemplo, o pixel (10,5) dunha imaxe RGB terá a súa componente vermella

no elemento (10,5,1) da matriz , a verde no elemento (10,5,2) e o azul no elemento (10,5,3) da matriz.

Na figura seguinte vemos unha imaxe RGB de tipo d*ouble.*

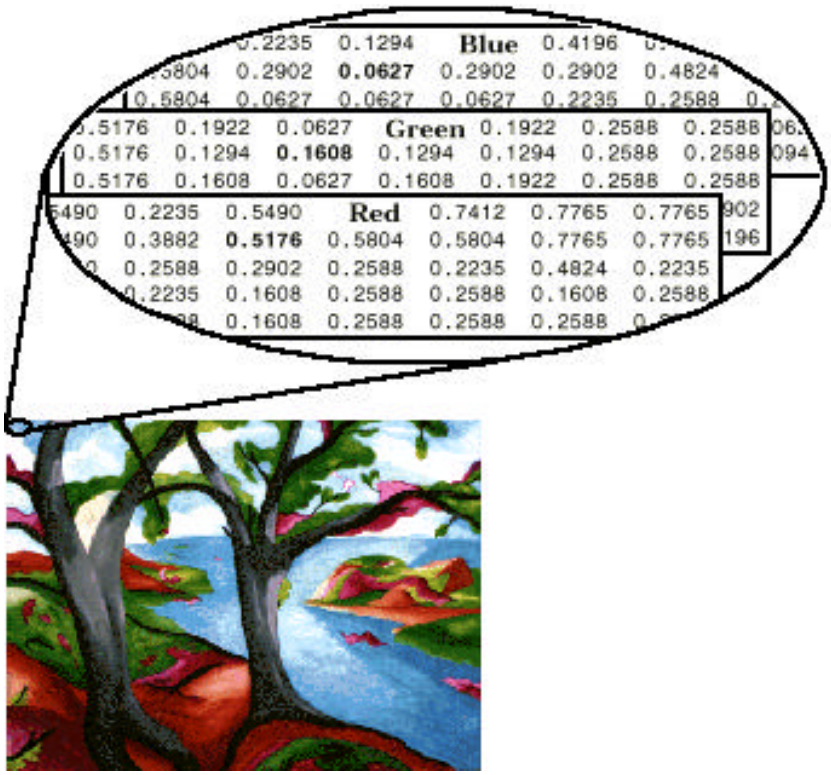

Figura 2.4. Imaxe RGB.

Para determina-la cor do pixel (2,3) temos que mirar na tripleta de tres valores almacenados en (2,3,1:3). No caso da figura a cor do pixel sería

0.5176 0.1608 0.0627

2.1.1.2.5.- Resumo dos tipos de imaxes e do tipos de datos numéricos

Nesta táboa resumimo-lo xeito no que Matlab 5 interpreta os elementos das matrices como cores de píxels, dependendo do tipo de imaxe e do tipo de datos da matriz.

Tipo de Imaxe	**Datos tipo *double***	**Datos tipo *uint8/uint16***
Binaria	A imaxe é un array mxn de enteiros no rango [0,1] onde o flag lóxico está en "on"	A imaxe é un array mxn de enteiros no rango [0,1] co flag lóxico en "on"
Indexada	A imaxe é un array mxn de enteiros no rango [1,p]. A paleta de cor é un array px3 de valores en coma flotante no rango [0,1]	A imaxe é un array mxn de enteiros no rango [1,p-1], onde p soe valer 256. A paleta de cor é un array px3 de valores en coma flotante no rango [0,1]
Escala de grises	A imaxe é un array mxn de números en coma flotante que representan intensidades de gris. O rango típico é [0,1]	A imaxe é un array mxn de números enteiros que representan intensidades de gris. O rango típico é [0,255] ou [0,65535]
RGB	A imaxe é un array mxnx3 de números en coma flotante no rango [0,1]	A imaxe é un array mxnx3 de enteiros no rango [0,255] ou [0,65535]

Táboa 2.1. Tipos de imaxes en Matlab 5.

2.1.1.3.- Traballar con Imaxes en Matlab 5

O toolbox de imaxe de Matlab 5 dispón de comandos para ler, escribir e visualizar varios tipos de formatos de imaxes :

- •BMP (Microsoft Windows Bitmap)
- •HDF (Hierarchical Data Format)
- •JPEG (Joint Photographic Experts Group)
- •PCX (Paintbrush)
- •PNG (Portable Network Graphics)
- •TIFF (Tagged Image File Format)
- •XWD (X Window Dump)

Hai que destacar que con esta versión de Matlab podemos visualizar directamente do ficheiro.

2.1.1.3.1.- Ler e escribir unha imaxe

Aquí, a principal diferencia coa versión anterior de Matlab é que agora non temos una función de lectura e escritura para cada tipo de formato, senón que o facemos todo coas funcións *imread* e *imwrite*.

A función *imread* le unha imaxe dende un ficheiro que conteña una imaxe nalgún dos formatos soportados por Matlab. Matlab vai almacenar en memoria a maioría das imaxes que imos ler como matrices de tipo *uint8*.

Para imaxes indexadas , imread sempre almacena a paleta de cor nun array de tipo *double* aínda que a matriz-imaxe pode ser de tipo *uint8.*

O uso básico da función *imread* é o seguinte :

Este código le unha imaxe gardada no ficheiro "lola.jpg" e gárdaa na variable RGB.

```
RGB=imread('lola.jpg');
```

Tamén podemos gardar unha imaxe que teñamos nunha variable nun ficheiro coa función *imread.* Supoñamos que temo-la matriz imaxe *X* e a paleta *map,* se escribimos esto

```
imwrite(X,map,'lola.bmp');
```

crearemos un ficheiro BMP que contén a imaxe.

Consulta-la referencia en liña de Matlab para máis información sobre o uso de *imread* e *imwrite.*

2.1.1.3.2.- Obter información sobre unha imaxe.

A función *iminfo* permítenos obter información sobre un ficheiro con algún dos formatos especificados anteriormente onde teñamos gardada unha imaxe. A información que obteñamos depende do tipo de ficheiro, pero sempre incluirá polo menos o seguinte:

. Nome do ficheiro, incluído o path.
. Formato do ficheiro.
. Número de versión do formato de ficheiro.
. Data da última modificación do ficheiro.
. Tamaño do ficheiro en bytes.
. Ancho da imaxe en píxels.
. Longo da imaxe en píxels.
. Número de bits por píxel.
. Tipo de imaxe : RGB, escala de grises ou indexada.

Consulta-la referencia en liña de Matlab para máis información sobre o uso de *iminfo.*

2.1.1.3.3.- Converter unha imaxe dun tipo a outro

Para certas operacións, pode ser útil converter unha imaxe a un tipo diferente. Por exemplo, se queremos filtrar unha imaxe en cor que está gardada como indexada, primeiro temos que pasala a formato RGB xa que Matlab aplícalle o filtro ós valores de intensidade da imaxe. Polo tanto se lle aplicamo-lo filtro a imaxe indexada o resultado será impredicible.

O toolbox de Matlab dispón dunha serie de funcións que nos permiten converter unha imaxe dun tipo a outro tipo distinto. As funcións teñen nomes mnemotécnicos (en inglés claro): por exemplo, *ind2gray* converte una imaxe indexada ó formato de escala de grises. Na seguinte táboa enuméranse algunhas das funcións de conversión máis usadas. Para máis información sobre o uso destas funcións consulta-lo manual de referencia en liña de Matlab.

Función	**Propósito**
gray2ind	Crea una imaxe indexada a partir dunha imaxe en escala de grises
im2bw	Crea una imaxe binaria a partir dunha imaxe en escala de grises, RGB ou indexada
ind2gray	Crea una en escala de grises partir dunha imaxe indexada

ind2rgb	Crea una imaxe RGB partir dunha imaxe indexada
mat2gray	Crea unha imaxe en escala de grises a partir dunha matriz de datos, mediante un escalado da mesma. Moi útil cando se usan filtros
rgb2gray	Crea una imaxe en escala de grises a partir dunha imaxe RGB
rgb2ind	Crea una imaxe indexada a partir dunha imaxe RGB

Táboa 2.2. Funcións de conversión máis usadas en Matlab 5.

2.1.1.3.4.- Traballar con datos tipo *uint8 e uint16*

Usamos *imread* para ler imaxes e gardalas como arrays de tipo *uint8* ou *uint16* (*uint8* o máis típico); usamos *imshow* para visualizar esas imaxes; e usamos *imwrite* para salvalas. A maioría das funcións do toolbox de imaxe de Matlab 5 aceptan datos de tipo *uint8* ou *uint16.* Consulta-la referencia en liña de Matlab 5 para ve-las descricións concretas das distintas funcións.

Ademais de ler, escribir e visualizar arrays de tipo *uint8* e *uint16*, en procesado de imaxe precisaremos facer máis cousas coas matrices polo tanto á hora de facer operacións con arrays de tipo *uint8* e *uint16*, teremos que ter en conta que non todas van ser soportadas por Matlab 5.

2.1.1.3.5.- Operacións matemáticas soportadas para arrays de tipo *uint8* e *uint16*

As seguintes operacións matemáticas de Matlab SI soportan datos de tipo *uint8* e *uint16*: *conv2, convn, fft2, fftn, sum.* Nestes casos a saída destas operacións é unha matriz de tipo *double.*

Se intentásemos realizar unha operación non soportada nun destes arrays recibiríamos un erro. Por exemplo,

BW3=BW1+BW2
??? Function '+' not defined for variables of class 'uint8'

2.1.1.3.6.- Converte-lo tipo de datos dunha imaxe

Se queremos realizar algunha operación que non sexa soportada para arrays de tipo *uint8* ou *uint16*, teremos que pasa-los datos da matriz a tipo *double* usando a función *double.* Por exemplo ,

BW =double (BW1) + double (BW2)

Hai que ter en conta , que converte-lo tipo de datos dunha matriz, cambia o xeito no que Matlab e o toolbox de imaxe interpretan eses datos. Se queremos que o array resultante sexa interpretado correctamente como unha imaxe, temos que facer un reescalado ou aplicar un offset ós datos cando son convertidos. Por exemplo nunha imaxe indexada os elementos da matriz son índices a unha paleta de cor, e se a matriz é de tipo *double* un 5 indica a quinta fila da paleta, pero si a matriz é de tipo *uint8* un 5 apunta á sexta fila da paleta.

Hai que ter en conta que so podemos reescalar ou engadir un offset en arrays de tipo *double* , porque os arrays de tipo *uint8* non soportan operacións aritméticas. Cando convertemos de *double* a *uint8,* primeiro realizamos tódalas operacións aritméticas necesarias antes de chamar á función *uint8,* que é a que realiza a conversión propiamente dita; cando convertemos de *uint8* a *double* primeiro temos que chamar á función *double* e despois realiza-la operacións aritméticas convenientes.

A táboa seguinte resume como converter imaxes de tipo *uint8* a imaxes de tipo *double.*

Tipo de Imaxe	**Exemplo**
Indexada	*B=double(A)+1;*
Escala de grises ou RGB	*B=double(A)/255;*
Binaria	*B=double(A);*

Táboa 2.3. Conversión de imaxes tipo *uint8* a outros tipos

A táboa seguinte resume como converter imaxes de tipo *double* a imaxes de tipo *uint8.*

Tipo de Imaxe	**Exemplo**
Indexada	*B=uint8(round(A-1));*
Escala de grises ou RGB	*B=uint8(round(A*255));*
Binaria	*B=logical(uint8(round(A)));*

Táboa 2.4. Conversión de imaxes tipo *double* a outros tipos

En versións máis recentes do toolbox de imaxe de Matlab, introdúcense unha serie de funcións para realiza-la conversión dun xeito máis doado :

im2double, im2uint8 e im2uint16

Estas funcións realizan automaticamente as conversións de tipos de datos tendo en conta todas estas consideracións. Por exemplo se escribimo-lo seguinte na cadro de comandos de Matlab

RGB2=im2uint8(RGB1);

teremos en RGB2 unha matriz de tipo *uint8* con rango entre [0,255], obtida a partir de RGB1 que é de tipo *double* con rango de [0,1].

2.1.1.3.7.- Visualizar imaxes con imshow

Para visualizar imaxes, o toolbox de Imaxe de Matlab 5 proporciónano-la función *imshow* coa que podemos visualizar imaxes de tipo binario, escala de grises, RGB e indexadas.

-Para visualizar unha imaxe binaria:
se temos unha imaxe binaria gardada nunha variable *BW*,
imshow(BW)

-Para visualizar unha imaxe escala de grises:
se temos unha imaxe en escala de grises gardada nunha variable *I*,
imshow(I)

-Para visualizar unha imaxe indexada:
se temos unha imaxe indexada gardada nunha variable *X*, e con paleta de cor *map*
imshow(X,map)

-Para visualizar unha imaxe binaria:
se temos unha imaxe RGB gardada nunha variable *RGB*,
imshow(RGB)

2.1.2.- Operacións Xeométricas

Neste apartado veremos algúns conceptos básicos sobre un tipo de funcións moi usados en procesado de imaxe: as operacións xeométricas. Estas funcións modifican a xeometría dunha imaxe, rotándoa, cambiando o seu tamaño ou seleccionando un anaquiño de imaxe. Estas funcións soportan tódolos tipos de imaxe.

Comezaremos este capítulo cunha breve discusión sobre a interpolación, unha operación común a moitas das funcións xeométricas. Despois describiremos algunhas das funcións xeométricas por separado, e veremos como se empregan.

2.1.2.1.- Interpolación

A función *imresize* e a función *imrotate* usan interpolación bidimensional como unha parte das operación que realizan. Por exemplo, se cambiamo-lo tamaño dunha imaxe , de tal xeito que a nova imaxe é máis grande que a orixinal, a función *imresize* ten que calcula-los valores dos novos píxels, e faino mediante interpolación.

O toolbox de imaxe de Matlab proporciona 3 métodos de interpolación.

. Interpolación de veciño máis próximo
. Interpolación bilineal
. Interpolación bicúbica

Tódolos métodos de interpolación traballan dun xeito similar. En cada caso, para determina-lo valor dun píxel interpolado, primeiro buscamo-lo punto na imaxe orixinal o que corresponde ó píxel de saída. Despois asignamos un valor ó píxel de saída, calculado unha media ponderada dun conxunto de píxels na veciñanza do píxel en cuestión.

Os diferentes métodos diferéncianse no conxunto de píxels a considerar.

. Para interpolación do veciño máis próximo, o píxel de saída asignámoslle o valor do píxel de entrada. Ningún outro píxel é tomado en conta.
. Para a interpolación bilineal, o valor do píxel de saída é unha media ponderada dos píxels nunha veciñanza de 2x2 píxels
. Para a interpolación bicúbica, o valor do píxel de saída é unha media ponderada dos píxels nunha veciñanza de 4x4 píxels.

O número de píxels considerados afecta a complexidade dos cálculos, de tal xeito que a interpolación bilineal, tarda máis que a do veciño máis próximo, e a interpolación bicúbica tarda máis que a bilineal.

Tamén hai que ter en conta que canto maior sexa o número de píxels considerados, o efecto será mellor. Temos polo tanto un compromiso entre o tempo de proceso da operación de interpolado e a calidade da mesma.

2.1.2.1.1.- Tipos de Imaxes

As funcións que usan interpolación levan un argumento onde se lle especifica o método de interpolación que se quere usar. O método do veciño máis próximo produce resultados aceptables para tódolos tipos de imaxe, e é o único método aceptable para imaxes indexadas. Para imaxes en escala de grises ou imaxes RGB, deberiamos usar interpolación bilineal ou interpolación bicúbica porque estes métodos producen mellores resultados que a interpolación do veciño máis próximo.

Para imaxes RGB, a interpolación é realizada nos planos vermello, verde e azul individualmente.

Para imaxes binarias, a interpolación ten efectos que deberiamos ter en conta. Se usamos interpolación bilineal ou bicúbica, os valores computados para os píxels da imaxe de saída, non serán todos 0 ou 1.

O efecto na imaxe de saída resultante depende do tipo de datos da imaxe de entrada:

. Se a imaxe de entrada é de tipo *double*, a imaxe de saída será unha imaxe en escala de grises de tipo *double*.
. Se a imaxe de entrada é de tipo *uint8*, a imaxe de saída será unha imaxe binaria de tipo *uint8*. Os valores dos píxels interpolados estarán redondeados a 0 ou a 1.

Se usamos interpolación do veciño máis próximo, o resultado será sempre binario, porque os valores dos píxels interpolados son tomados directamente dos píxels da imaxe de entrada.

2.1.2.2.- Cambio do tamaño dunha imaxe.

A función *imresize* do toolbox de imaxe de Matlab cambia o tamaño dunha imaxe usando o método de interpolación que lle indiquemos. Se non lle indicamos ningún usará interpolación do veciño máis próximo.

Podemos usar *imresize* para magnificar unha imaxe dándolle un factor de magnificación. Para agrandar unha imaxe temos que especificar un factor maior que 1. Por exemplo este comando dobra o número de píxels en X en cada dirección:

```
Y=imresize(X,2);
```

Para reducir unha imaxe teremos que especificar un factor entre 0 e 1.

Tamén podemos especifica-lo tamaño da imaxe de saída. Este comando crea unha imaxe de saída de 100x150 píxels.

```
Y=imresize(X,[100 150]);
```

Hai que ter en conta que se o novo tamaño no ten o mesma relación de aspecto que o tamaño orixinal a imaxe aparecerá distorsionada.

2.1.2.3.- Rotacións

A función *imrotate* xira unha imaxe usando o método de interpolación que lle indiquemos así como o ángulo de xiro en graos. Se non especificamos un método de interpolación a función usará interpolación do veciño máis próximo.

Se o ángulo de xiro en graos é negativo a imaxe xirará no sentido das agullas do reloxo. Se é positivo no sentido contrario.

Os seguintes comandos xiran unha imaxe 35º no sentido das agullas do reloxo usando interpolación bilineal.

```
I=imread('ic.tif');
J=imrotate(I,35,'bilinear');
imshow(I)
figure, imshow(J);
```

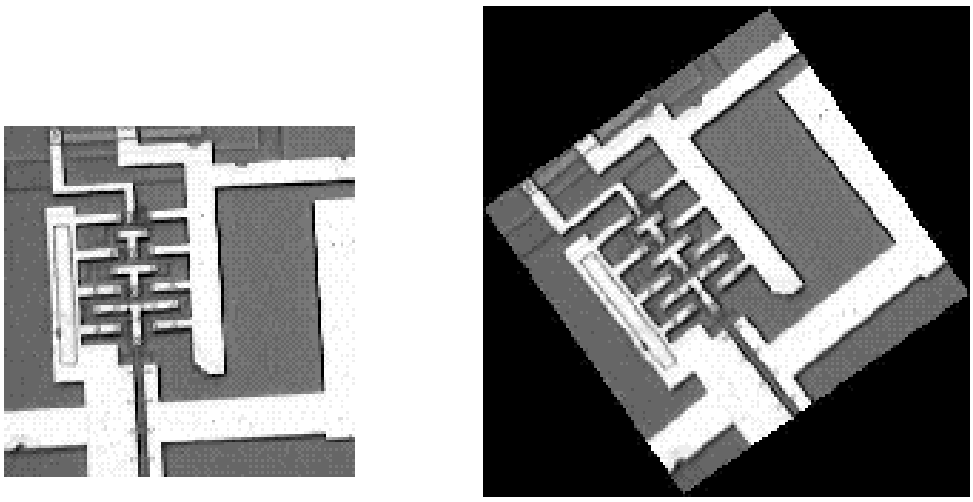

Figura 2.5. Imaxe xirada 35º.

Hai que ter en conta que a imaxe de saída é maior que a imaxe de entrada, para poder incluí-la imaxe orixinal enteira. *imrotate* engade 0´s (píxels negros) a imaxe de saída para enche-la área fora da imaxe orixinal.

imrotate ten unha opción para recorta-la imaxe de saída ó mesmo tamaño que a imaxe de entrada. Para máis información sobre a función *imrotate* consulta-la referencia en liña de Matlab.

2.1.2.4.- Recortar unha imaxe.

A función *imcrop* extrae unha porción rectangular da imaxe. Podemos especifica-lo rectángulo a través de argumentos de entrada ou ben seleccionalo co rato.

Se chamamos á función *imcrop* sen especificar un rectángulo, o cursor toma a forma dunha cruz cando está sobre a imaxe. Se facemos click nunha esquina da nosa rexión de interese, e arrastramo-lo punteiro do rato ata a esquina contraria , teremos seleccionada a nosa rexión de interese, e *imcrop* terá creada unha nova imaxe que contén a rexión que marcamos previamente.

Neste exemplo, visualizamos unha imaxe e facemos unha chamada a *imcrop*. O rectángulo seleccionado está marcado en vermello.

```
imshow ic.tif
I=imcrop;
imshow(I);
```

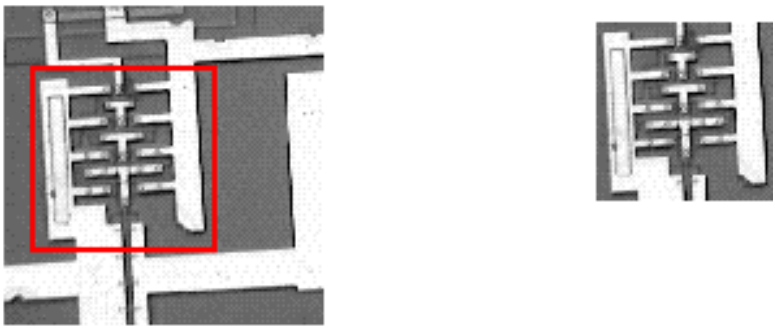

Figura 2.6. Na imaxe da esquerda seleccionamos unha área para recortar e visualizámo-la imaxe recortada na da dereita.

2.1.3.- Operacións de veciñanza

Algunhas operacións de procesado de imaxe procesan a imaxe en seccións, en vez de facelo con toda a imaxe completa. Por exemplo, moitas operacións de filtrado lineal e operacións binarias son de este tipo.

Neste apartado veremos algunhas funcións deste tipo, concretamente aquelas que realizan operacións de veciñanza deslizante.

Nas operacións de veciñanza deslizante, a imaxe de entrada é procesada píxel a píxel pero considerando tamén os seus píxels veciños.

É dicir, por cada píxel na imaxe de entrada, realízanse unha serie de operacións para determina-lo valor do correspondente píxel da imaxe de saída. Estas operacións están baseadas no bloque de píxels que forman a veciñanza do píxel de entrada en cuestión. Como cada píxel ten a súa veciñanza, vemos que para calcula-lo valor do píxel seguinte o bloque de píxels a ter en conta é distinto e por iso a estas operacións se lles chama operacións de veciñanza deslizante.

2.1.3.1.- Operacións de veciñanza deslizante

Unha operación de veciñanza deslizante é unha operación que é realizada cun píxel de cada vez. O valor de cada píxel da imaxe de saída, é determinado aplicándolle un algoritmo determinado ós píxels que forman a veciñanza do correspondente píxel de entrada. Os píxels que forman a veciñanza dun píxel (que chamaremos *píxel central)* forman un bloque rectangular, que conforme nos imos movendo dun elemento ó seguinte da matriz imaxe, o bloque da veciñanza vaise deslizándo na mesma dirección.

A figura seguinte mostra as veciñanzas para algúns elementos nunha matriz 6x5 con bloques de veciñanza de 2x3 píxels. O píxel central para cada un dos bloques está marcado cun puntiño.

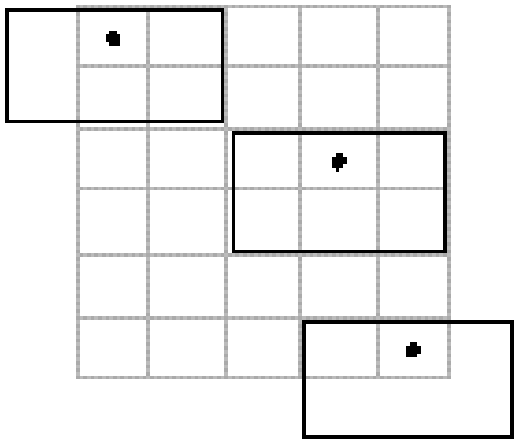

Figura 2.7. Matriz na que se indica a veciñanza dalgúns dos seus puntos.

O píxel central é o correspondente píxel da imaxe de saída para o que se quere determina-lo valor. Se a veciñanza ten un número impar de filas e columnas, o píxel central é o centro do rectángulo. Se algunha das dimensións do rectángulo que forma a veciñanza é par, o píxel central está xusto á esquerda do centro ou xusto sobre o centro. Por exemplo nunha veciñanza de 2x2 píxels o píxel central é o píxel de arriba á esquerda.

Para calquera veciñanza de tamaño mxn o píxel central é :

floor(([m n]+1)/2)

Na veciñanza da figura anterior o píxel central é o (1,2).

Unha operación de veciñanza deslizante procede do seguinte xeito:

1. Selecciona un píxel.
2. Determina a veciñanza do píxel.
3. Aplica unha función ós valores dos píxels da veciñanza. Esta función devolve un escalar.
4. Busca o píxel da imaxe de saída que corresponda coa posición do píxel central da imaxe de entrada. O valor deste píxel de saída será o valor calculado coa función.
5. Repite os pasos 3 a 4 para cada píxel da imaxe de entrada.

Por exemplo supoñamos que a figura anterior representa unha operación de promediado. A función debería suma-los valores dos seis píxels da veciñanza e despois dividir por 6. O resultado é o valor do píxel de saída.

2.1.3.1.1.- Recheo dos Bordes

Cando se procesan os píxels que forman os bordes da imaxe pode que non existan os píxels da veciñanza. Supoñamos que establecemos unha veciñanza de 2x3. Como vemos nesta figura algúns dos píxels da veciñanza dos píxels dos bordes

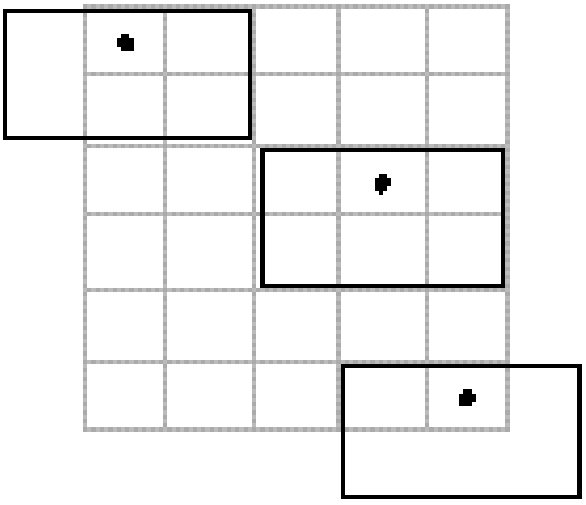

Figura2.8. Matriz na que vemos que algúns puntos da veciñanza non están na matriz.

simplemente no existen na imaxe. Para procesar estes píxels as operacións de veciñanza.
deslizante "cobren" os bordes da imaxe con ceros. É dicir as funcións que realizan operacións deste tipo procesan os píxels dos bordes supoñendo que a imaxe está rodeada de ceros. Estas filas e columnas adicionais de ceros non formarán parte da imaxe de saída. Hai que ter en conta que este recheo con ceros non sempre será a solución máis axeitada. Por exemplo nunha operación de promediado.

2.1.3.2.- Filtrado lineal e non lineal

Podemos usar tamén as funcións de veciñanza para realizar operacións de filtrado. Un exemplo claro de operación de veciñanza deslizante é a convolución, que é usada para filtrado lineal. Matlab 5 dispón das funcións *conv2* e *filter2* para realiza-la convolución

Ademais da convolución hai outras operacións de filtrado que podemos realizar usando operacións de veciñanza deslizante. Moitas destas operacións son de natureza non lineal. Por exemplo, podemos realizar unha operación de veciñanza deslizante onde o valor do píxel de saída sexa a desviación típica dos valores dos píxels da veciñanza.

Podemos usa-la función *nlfilter*, para implementar unha gran variedade de operacións de veciñanza deslizante. *nlfilter* toma como argumentos de entrada unha imaxe, un tamaño de veciñanza e unha función que devolve un escalar, e o resultado de *nfilter* é unha imaxe do mesmo tamaño que a imaxe de entrada á que se lle aplicou a operación de veciñanza deslizante coa función e o tamaño de veciñanza especificados nos parámetros de entrada. Por exemplo, a seguinte liña de comandos computa cada píxel de saída tomando a desviación típica dos valores da veciñanza 3x3 do píxel central, é dicir o píxel central e os seus 8 veciños.

```
I2 = nlfilter(I,[3 3],'std2');
```

Podemos ademais (isto é algo moi útil), implementa-la función que nós queiramos nun ficheiro .m e usar esa función con *nlfilter*. Por exemplo, o seguinte comando procesa a matriz I, cunha veciñanza de 2x3 píxels e cunha función chamada *amiñafuncion.m:*

```
nlfilter(I,[2 3],'amiñafuncion');
```

Podemos tamén usar unha función inline, por exemplo :

```
f = inline('sqrt(min(x(:)))');
I2 = nlfilter(I,[2 2],f);
```

O exemplo seguinte usa *nlfilter* para facer que cada píxel da imaxe de saída tome o valor máximo dos píxels da veciñanza. Vemo-los resultados na figura 2.9.

```
I = imread('tire.tif');
f = inline('max(x(:))');
I2 = nlfilter(I,[3 3],f);
```

imshow(I)
figure, imshow(I2)

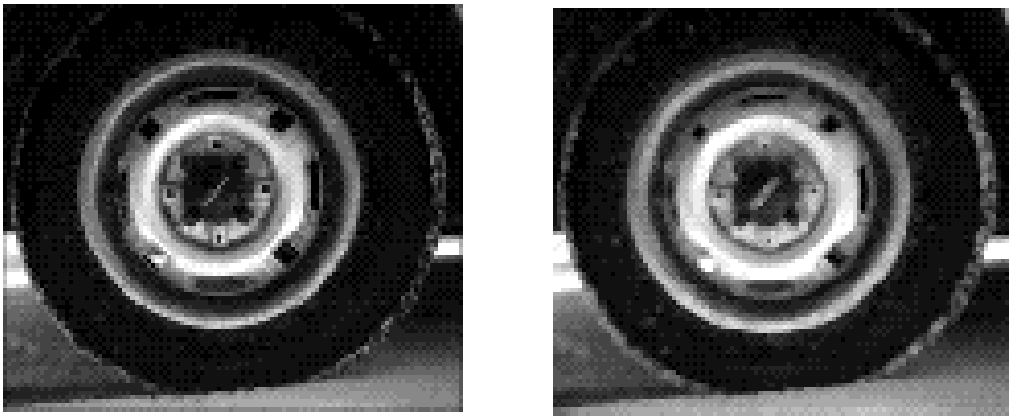

Figura 2.9.

Neste exemplo vemos que o efecto é como o dunha dilatación dunha imaxe con niveis de gris, xa que vemos na imaxe da dereita como medran as zonas brillantes.

2.1.4.- Filtrado lineal e Deseño de Filtros

O toolbox de imaxe de Matlab dispón dunha serie de funcións para deseñar e implementar filtros bidimensionais para traballar con imaxes. Este apartado describe estas ferramentas e como usalas.

O filtrado é unha técnica para modificar ou realzar unha imaxe. Por exemplo, podemos filtrar unha imaxe para resaltar certos aspectos da mesma ou o contrario filtramo-la imaxe para eliminar certas características.

O filtrado é unha operación de veciñanza, na que o valor dun píxel na imaxe de saída ven dado pola aplicación dun determinado algoritmo ós valores dos píxels veciños do correspondente píxel da imaxe de entrada.

O filtrado lineal é un tipo de filtrado no que o valor do píxel de saída é unha combinación lineal dos valores dos píxels veciños ó correspondente píxel da imaxe de entrada. Por exemplo un algoritmo que calcule a media da veciñanza é un tipo de operación de filtrado lineal. Nesta sección veremos cómo Matlab realiza o filtrado lineal usando a convolución e veremos tamén como usar filtros predefinidos.

2.1.4.1.- Convolución

En Matlab, o filtrado lineal é realizado a través da convolución bidimensional. Na convolución o valor dun píxel de saída é calculado multiplicando elementos de dúas matrices e sumando os resultados. Unha desa matrices representa a imaxe mentres que a outra é o filtro. Por exemplo, un filtro podería ser:

```
k = [4    -3    1
     4     6    2]
```

Esta representación do filtro é coñecida como o *kernel da convolución.* A función *conv2* de Matlab implementa o filtrado da imaxe aplicándolle o *kernel* a unha matriz imaxe. *conv2* toma como argumentos de entrada unha imaxe e o filtro, e devolve unha imaxe de saída. Por exemplo, nesta chamada, *k* é o *kernel da convolución*, *A* é a imaxe de entrada, e *B* é a imaxe de saída:

```
B = conv2(A,k);
```

conv2 produce unha imaxe de saída realizando as seguintes accións:

1) rota o *kernel da convolución* 180º para producir unha *molécula computacional.*
2) Determina o píxel central da *molécula computacional.*
3) Aplícalle a *molécula computacional* a cada un dos píxels da imaxe de entrada.

Veremos agora con máis detalle cada un destes pasos:

1) Rotación do *kernel*

Na convolución bidimensional , os cálculos son realizados usando unha *molécula computacional*, que é simplemente o *kernel* rotado 180º, como na seguinte chamada:

```
h = rot90(k,2)
h =
     2     6     4
     1    -3     4
```

Para xira-la imaxe 180º usamo-la función *rot90* (que so xira 90º) pasándolle como argumentos de entrada o kernel e o número de veces que queremos que se realice o xiro (neste caso 2 veces para xirar 180º).

2) Determinación do píxel central

Para poder aplica-la *molécula computacional* primeiro temos que determinar cal é o píxel central. O píxel central ven definido por *floor((size(h)+1)/2).* Por exemplo, nunha molécula 5x5 o píxel central é o (3,3). Na *molécula computacional h*, do noso exemplo o píxel central é o (1,2).

3) Aplicación da *molécula computacional*

O valor dun píxel dado da imaxe de saída B, virá determinado pola aplicación da *molécula computacional h*, ó correspondente píxel en *A.* Podemos pensar nesta operación coma unha superposición de *h* sobre *A*, co píxel central de *h* sobre o píxel de interese en *A.* Despois multiplicamos cada elemento de *h* polo correspondente píxel en A, e sumamo-los resultados.

Por exemplo para determina-lo valor do píxel (4,6) en *B*, superpoñemos *h* sobre *A* co píxel central de *h* sobre o píxel (4,6) de *A*. O píxel central aparece rodeado por un círculo na figura .

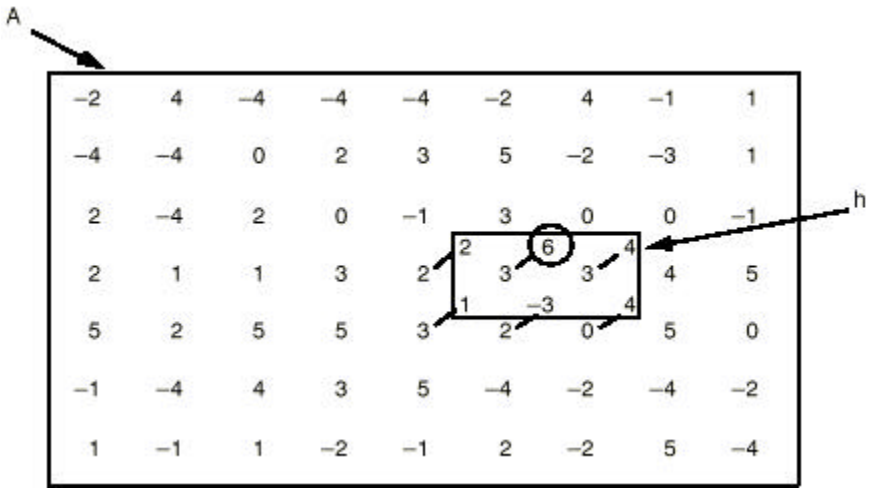

Figura 2.10. Pixel central e a súa veciñanza.

Agora se collemo-los seis píxels de *A* cubertos por *h* e multiplicamos cada valor do pixel polo valor en *h*, e sumamo-lo resultado, teremo-lo valor do pixel (4,6) de *B*.

*B(4,6)=2*2 + 3*6 + 3*4 + 3*1 +2*-3 + 0*4 = 31*

Se realizamos este procedemento para cada pixel en *A*, ó final teremos tódolos valores de *B*.

2.1.4.1.1.- Recheo dos bordes

Cando aplicamos un filtro ós píxels dos bordes dunha imaxe, algúns dos elementos da *molécula computacional*, pode que no se superpoñan sobre ningún píxel da imaxe de entrada. Por exemplo se a molécula é 3x3 e estamos calculado o valor para un píxel da fila de arriba de todo da imaxe, algúns elementos da *molécula computacional* estarán fora dos límites da imaxe.

Esta figura ilustra unha *molécula computacional* de 3x3 elementos aplicada ó pixel (1,3) dunha matriz 5x5. O pixel central está indicado cun punto

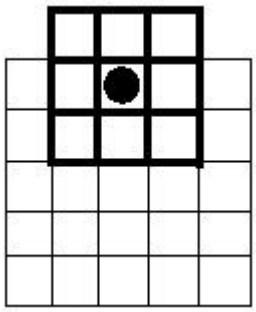

Figura 2.11. Non tódolos píxels da veciñanza están na imaxe.

Para poder calcula-los valores da imaxe de saída dos píxels dos bordes, *conv2* recubre a matriz imaxe con ceros. Noutras palabras, os valores de saída son calculados supoñendo que a imaxe de entrada está rodeada de ceros. Na figura anterior os elementos da fila de arriba da *molécula computacional* é coma se multiplicaran por ceros.

Dependendo do que queiramos facer, podemos descarta-los píxels con valores que dependen do recheo de ceros. Para indicar qué porción da convolución queremos, a *conv2* podemos pasarlle un terceiro argumento de entrada, chamado parámetro de forma e que pode tomar un dos seguintes 3 valores:

. 'valid' - devolve só os píxels con valores que non foron calculados usando recheo de ceros. A imaxe de saída resultante é máis pequena que a imaxe orixinal. No noso exemplo 3x3. (imaxe de entrada 5x5, molécula 3x3).

. 'same' - devolve o conxunto de píxels que poden ser computados aplicándolle o filtro a tódolos píxels que son actualmente parte da imaxe. Os píxels dos bordes son calculados usando recheo con ceros. O resultado é unha imaxe do mesmo tamaño que a imaxe de entrada.

. 'full' - Devolve unha convolución completa. Esto significa que *conv2* devolve tódolos píxels para os cales calquera dos píxels da molécula computacional se superpoñan a píxels da imaxe orixinal, aínda que o pixel central estea fora da imaxe de entrada. A imaxe de saída resultante é máis grande que a imaxe de entrada. Neste exemplo e imaxe de saída é 7x7.

conv2 devolve a convolución 'full' por defecto. Na seguinte figura vemos como aplica-la *molécula computacional* a tres diferentes lugares na matriz imaxe:

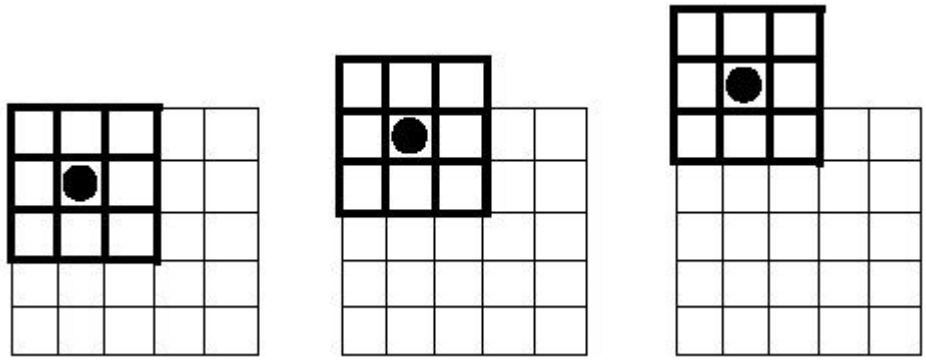

Figura 2.12.

Na primeira imaxe da figura 2.12 vemos que a *molécula computacional* superponse só sobre píxels que están na imaxe orixinal. O resultado é incluído na imaxe de saída sen importa-lo parámetro de forma especificado.

Na segunda imaxe da figura 2.12 vemos que a *molécula computacional* superponse a algúns píxels fora da imaxe orixinal, pero o pixel central está sobre un pixel pertencente á imaxe de entrada. O resultado é incluído na imaxe de saída se o parámetro de forma é 'full' ou 'same'.

Na terceira imaxe da figura 2.12 vemos que a *molécula computacional* superponse a algúns píxels fora da imaxe orixinal incluído o pixel central. O resultado é incluído na matriz de saída se o parámetro de saída é 'full'.

2.1.4.2.- A función *filter2*

Ademais da función *conv2* Matlab tamén dispón da función *filter2* para filtrado lineal bidimensional. *filter2* produce os mesmos resultados que *conv2* sendo a única diferencia que toma como argumento de entrada a *molécula computacional* e non o *kernel.* A operación que realiza *filter2* recibe o nome de *correlación.* Se *k* é o *kernel da convolución, h* é a *molécula computacional* correspondente, e *A* é unha matriz-imaxe as seguintes chamadas producen os mesmos resultados:

B = conv2(A,k,'same');

e

B=filter2(h,A,'same');

2.1.4.3.- Convolución n-dimensional

Para realizar convolucións bidimensionais podemos usar *conv2* ou *filter2*. Para realizar convolucións n-dimensionais usaremo-la función *convn*. *convn* toma como argumentos de entrada un array de datos e un *kernel*. Ambos poden ser de calquera dimensión. *convn* devolve un array tal que a súa dimensión é a dimensión máis grande dos arrays de entrada. *convn* tamén toma como argumento de entrada un parámetro de forma que acepta os mesmos valores que *conv2* e *filter2* e con efectos análogos.

Unha importante aplicación da función *convn* é filtrar arrays que teñen múltiples planos ou tomas. Por exemplo, supoñamos un array A que contén cinco imaxes RGB e que queremos filtrar usando un *kernel k* bidemensional. Se usáramos *conv2* ou *filter2* teríamos que chamar á función 15 veces, mentres que se usamos *convn* so teríamos que facelo unha vez

B=convn(A,k);

Neste exemplo como A ten 15 planos a función *convn* xa extende k automaticamente.

2.1.4.4.- Filtros Predefinidos

A función *fspecial* produce varios tipos de filtros predefinidos, en forma de *moléculas computacionais*. Logo de crear un filtro con *fspecial*, podemos aplicalo directamente á imaxe usando *filter2* ou rotala molécula e usar *conv2* ou *convn.*

Un filtro moi simple que podemos crear con *fspecial* é un filtro de promediado. Este tipo de filtro computa un valor dun pixel da imaxe de saída promediando o valor dos seus veciños.

O tamaño por defecto do filtro de promediado que crea *fspecial* é 3x3, é dicir *fspecial* devolve por defecto unha *molécula computacional* 3x3, pero tamén podemos especificar un tamaño diferente. O valor de cada elemento é *1/length(h(:))*. Por exemplo, un filtro de promediado 5x5 sería :

0.0400	0.0400	0.0400	0.0400	0.0400
0.0400	0.0400	0.0400	0.0400	0.0400
0.0400	0.0400	0.0400	0.0400	0.0400
0.0400	0.0400	0.0400	0.0400	0.0400
0.0400	0.0400	0.0400	0.0400	0.0400

Aplicar este filtro a un pixel é equivalente a suma-los valores dos 25 veciños e dividir por 25. Esto ten o efecto de suaviza-los brillos e desdibuxa-los bordes dunha imaxe. O exemplo seguinte ilustra o efecto de aplicarlle un filtro de promediado a unha imaxe de escala de grises

```
I=imread('blood1.tif');
h=fspecial('average',5);
I2=uint8(round(filter2(h,I)));
imshow(I)
figure , imshow(I2)
```

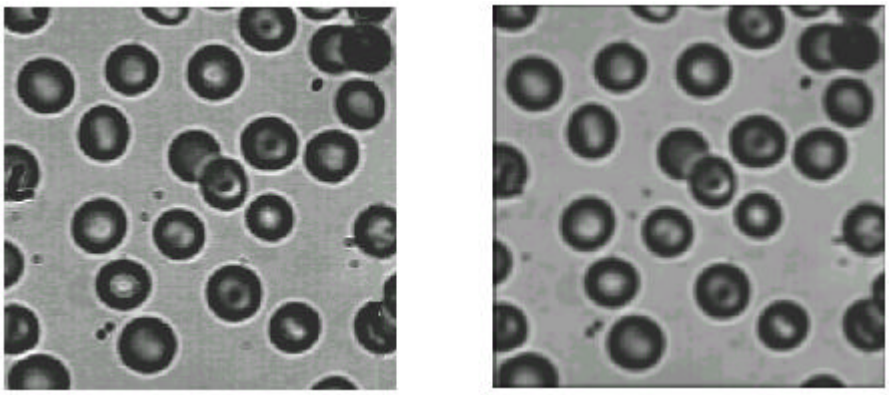

Figura 2.13. Resultados obtidos co filtro de promediado

Hai que ter en conta que a imaxe de saída obtida con *filter2 (*e con *conv2* e *convn)* e sempre de clase *double*. No exemplo anterior, a imaxe de entrada, é de clase *uint8*, e a saída obtida coa aplicación de *filter2* consiste en valores en dobre precisión no rango [0,255]. A chamada á función *uint8*, converte a imaxe de saída a unha de clase *uint8.*

Outro filtro que podemos implementar usando *fspecial* é o filtro de Sobel que é efectivo detectando os bordes horizontais de obxectos nunha imaxe.

```
h=fspecial('sobel');
h=
     1     2     1
     0     0     0
    -1    -2    -1
```

h é a matriz que calcula o gradiente vertical. Tamén existe a matriz do gradiente horizontal, que sería así:

```
hhor=
    -1     0     1
    -2     0     2
    -1     0     1.
```

O filtro de Sobel produce valores fora do rango dos datos de entrada. Por exemplo se a imaxe de entrada é de clase *double*, a saída pode tomar valores fora do

rango [0,1]. Para poder ve-la saída coma unha imaxe, podemos usar *imshow* especificándolle o rango dos datos ou usa-la función *mat2gray* para converte-los valores ó rango [0,1].

Hai que ter en conta que se a imaxe de entrada é de tipo *uint8* , non só teremos que converte-lo array de saída á mesma clase que a imaxe de entrada, senón que tamén teremos que reescala-los valores . Veremos esto co exemplo seguinte:

```
h=fspecial('sobel');
I2=filter2(h,I);
J=uint8(round(mat2gray(I2)*255));
```

Podemos tamén usar *imshow* para visualiza-lo array de saída sen reescalar primeiro os datos. O seguinte exemplo crea un filtro de Sobel e usa *filter2* para aplicarlle o filtro á imaxe blood1.tif. Hai que ter en conta que na chamada á *imshow*, o rango de intensidades está especificado como unha matriz valeira ([]). Isto dille á función *imshow* que "pinte" o valor mínimo de I2 como se fose negro e o máximo como branco, e os valores entre ambos como intensidades intermedias de gris:

```
I=imread('blood1.tif');
H=fspecial('sobel');
I2=filter2(h,I);
Imshow(I2,[])
```

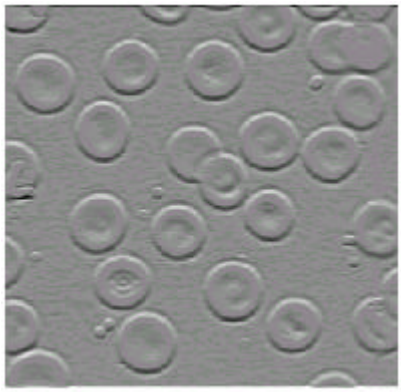

Figura 2.14. Resultado obtido co filtro de Sobel

Para unha descrición de tódolos tipos de filtros que podemos crear con *fspecial*, consulta-la referencia en liña de Matlab.

2.1.4.5.- Deseño de filtros

Esta sección describe como traballar no dominio da frecuencia para deseñar filtros. Veremos por tanto os seguintes temas:

1)Filtros FIR (Finite Impulse Response), a clase de filtros que soporta Matlab
2)O método da transformada en frecuencia, que transforma un filtro FIR unidimensional nun filtro FIR bidimensional
3)Método da mostraxe en frecuencia, que crea un filtro FIR bidimensional a partires dunha resposta en frecuencia ideal.
4)Como calcula-la resposta en frecuencia dun filtro

1)Filtros FIR (Finite Impulse Response)

Os filtros FIR teñen unha serie de características que fan deles que sexan unha ferramenta moi axeitada para o procesado de imaxe no entorno Matlab.

. Os filtros FIR son fáciles de representar como matrices de coeficientes.
. Os filtros FIR bidimensionais son extensións naturais dos filtros FIR unidimensionais.
. Xa hai unha serie de métodos ben coñecidos para deseñar filtros FIR
.Os filtros FIR son fáciles de implementar
. Os filtros FIR poden ser deseñados para que teñan unha fase lineal, que axuda a previ-la distorsión.

Hai outra clase de filtros, os IIR que non son tan aquelados para aplicacións de procesado de imaxe. Carecen da estabilidade e da facilidade de deseño e implementación dos filtros FIR. O toolbox de imaxe de Matlab 5 non soporta filtros IIR.

2)Método da Transformación de Frecuencia

O método da transformación de frecuencia transforma un filtro FIR unidimensional nun filtro FIR bidimensional. Este método preserva a maioría das características do filtro unidimensional, en particular a banda de paso e as características de rizado. Este método fai uso dunha matriz de transformación, un conxunto de elementos que definen a transformación de frecuencia.

A función *ftrans2* implementa o método da transformación de frecuencia. A matriz de transformación por defecto produce filtros con simetría circular. Definindo unha matriz de transformación distinta podemos obter distintas simetrías pero iso é algo que nós non veremos.

Este método produce xeralmente moi bos resultados, xa que é máis doado deseñar un filtro unidimensional con características determinadas que o seu correspondente filtro bidimensional. No seguinte exemplo deseñamos un filtro FIR unidimensional e usamos *ftrans2* para crear un filtro bidimensional con características similares. A forma da resposta en frecuencia para o filtro FIR unidimensional pode verse claramente no correspondente filtro FIR bidimensional.

```
b = remez(10,[0 0.4 0.6 1],[1 1 0 0]);
h = ftrans2(b);
[H,w] = freqz(b,1,64,'whole');
colormap(jet(64))
plot(w/pi–1,fftshift(abs(H)))
figure, freqz2(h,[32 32])
```

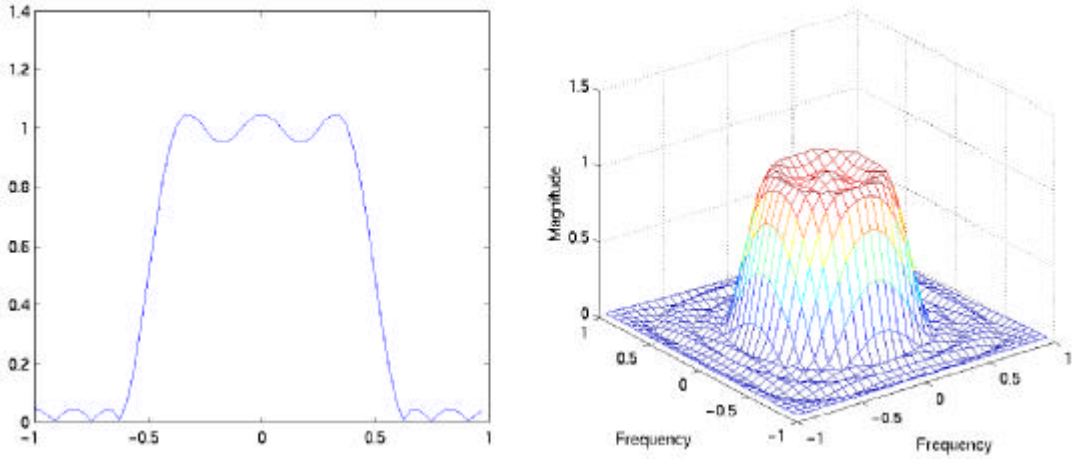

Figure 2.15. Resposta en frecuencia dos filtros creados no exemplo anterior

3)Método da mostraxe en frecuencia

Este método crea un filtro baseado nunha resposta en frecuencia que nos lle indicamos. Dada unha matriz de puntos que definen a forma dunha resposta en frecuencia, este método crea un filtro tal que a súa resposta en frecuencia pase por eses puntos.

A función *fsamp2* implementa este método de deseño de filtros FIR bidimensionais. *fsamp2* devolve un filtro *h* cunha resposta en frecuencia que pasa polos puntos da matriz de entrada *Hd*. O exemplo seguinte crea un filtro 11x11 usando *fsamp2*, e debuxa a resposta en frecuencia do filtro resultante. A función *freqz2* neste exemplo calcula a resposta en frecuencia do filtro bidimensional

```
Hd = zeros(11,11); Hd(4:8,4:8) = 1;
[f1,f2] = freqspace(11,'meshgrid');
mesh(f1,f2,Hd), axis([–1 1 –1 1 0 1.2]), colormap(jet(64))
h = fsamp2(Hd);
figure, freqz2(h,[32 32]), axis([–1 1 –1 1 0 1.2])
```

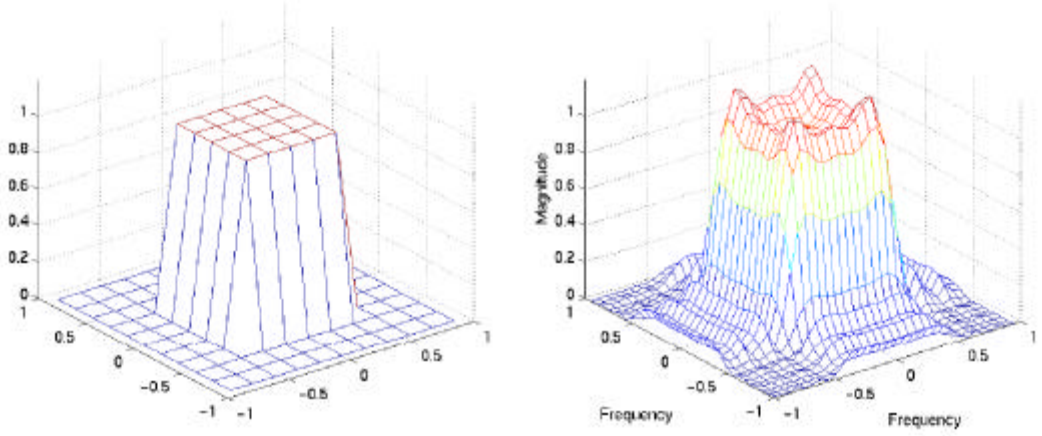

Figura 2.16.

Nestas 2 gráficas da Figura 2.16, pintamos á esquerda a resposta en frecuencia desexada e na outra a resposta obtida co método da mostraxe en frecuencia

Hai que ter en conta o efecto de rizado na resposta en frecuencia que obtemos coa aplicación deste método e que ocorre cando hai transicións bruscas na resposta en frecuencia desexada. As oscilacións baixan ó aumenta-lo tamaño do FIR. Para conseguir unha aproximación máis suave habería que considera-la utilización doutros métodos como o método da transformación de frecuencia ou métodos de fiestra que non veremos neste traballo.

4)Calculo da resposta en frecuencia dun filtro.

A función *freqz2* calcula a resposta en frecuencia dun filtro bidimensional. Esta función non ten argumentos de saída, e crea unha malla que mostra a resposta en frecuencia. Por exemplo, se temo-lo seguinte filtro FIR

```
h = [0.1667     0.6667     0.1667
     0.6667    -3.3333     0.6667
     0.1667     0.6667     0.1667 ];
```

O comando seguinte calcula e visualiza a resposta en frecuencia de *h* de 64x64 puntos.

```
freqz2(h)
```

que vemos na figura 2.17:

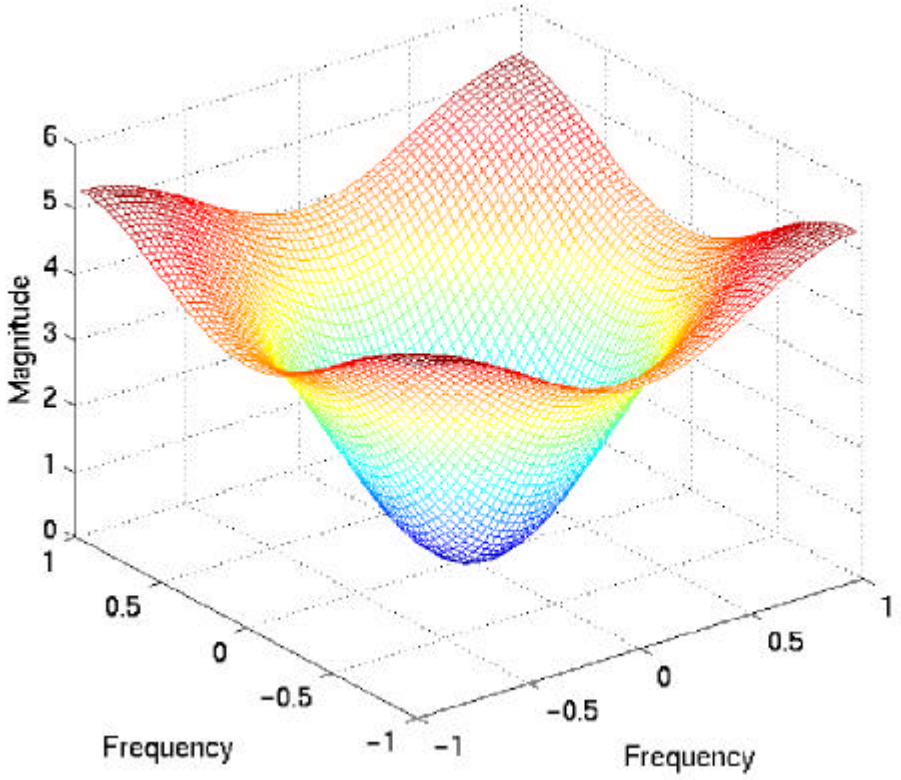

Figura 2.17. Resposta en frecuencia do filtro h.

Para obte-la matriz da resposta en frecuencia *H* e os vectores de puntos *f1* e *f2*, usaremos argumentos de saída :

[H,f1,f2] = freqz2(h);

freqz2 normaliza as frecuencias *f1* e *f2* de tal xeito que o valor 1.0 corresponde coa metade da frecuencia de mostraxe, é dicir π radiáns.

2.1.5.- Transformada de Fourier.

A representación máis usual dunha imaxe é unha función de 2 variables espaciais: f(x,y) . O valor da función para unha posición concreta (x,y) representa a intensidade da imaxe nese punto. O termo *transformada* refírese a unha representación matemática alternativa dunha imaxe.

Por exemplo , a transformada de Fourier é unha representación dunha imaxe como unha suma de exponenciais complexas con magnitudes , frecuencias e fases variables. Este tipo de representación é moi útil para moitas aplicacións como análise de imaxes, restauración e filtrado.

2.1.5.1.- Fundamentos Teóricos

Definición.

Se f(m,n) é unha función de dúas variables espaciais m e n, entón definiremo-la transformada de Fourier bidimensional de f(m,n) como:

$$F(\omega_1, \omega_2) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(m, n) e^{-j\omega_1 m} e^{-j\omega_2 n}$$

As variables ω_1 e ω_2 son variables de frecuencia; as súas unidades son radiáns por mostra. $F(\omega_1,\omega_2)$ é a representación de f(m,n) no dominio da frecuencia. $F(\omega_1,\omega_2)$ é unha función que toma valores complexos, e é periódica en ω_1 e ω_2 con período 2π . O máis usual é que se visualice o rango $-\pi \leq \omega 1,\ \omega 2 \leq \pi$ debido precisamente á periodicidade. Hai que ter en conta que F(0,0) é a suma de tódolos valores de f(m,n). Por esta razón a F(0,0) tamén se lle chama a componente de continua ou componente DC da transformada de Fourier.

A transformada de Fourier inversa bidimensional é :

$$f(m, n) = \frac{1}{2\pi} \int_{\omega_1=-\pi}^{\pi} \int_{\omega_2=-\pi}^{\pi} F(\omega_1, \omega_2) e^{j\omega_1 m} e^{j\omega_2 n}\ d\omega_1\ d\omega_2$$

Esta ecuación ven a significar que f(m,n) pode ser representada como unha suma dun número infinito de exponenciais complexas (sinusoides) con diferentes frecuencias. A magnitude e a fase da contribución das frecuencias ω1 e ω2 ven dada por F(ω1,ω2) .

Exemplo

Consideremos unha función f(m,n) igual a 1 nunha rexión rectangular dada e 0 no resto:

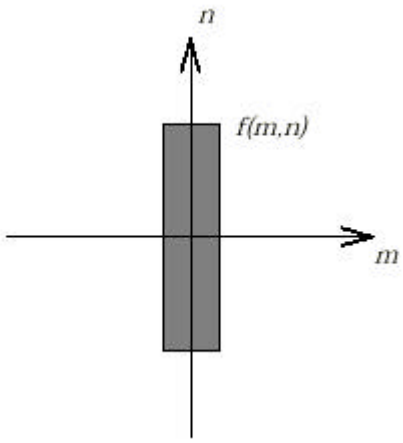

Figura 2.18. f(m,n).

Para simplifica-lo diagrama representamos f(m,n) como unha función continua aínda que as variables m e n sexan discretas.

A magnitude da transformada de Fourier $|F(\omega 1,\omega 2)|$ está representada na seguinte figura:

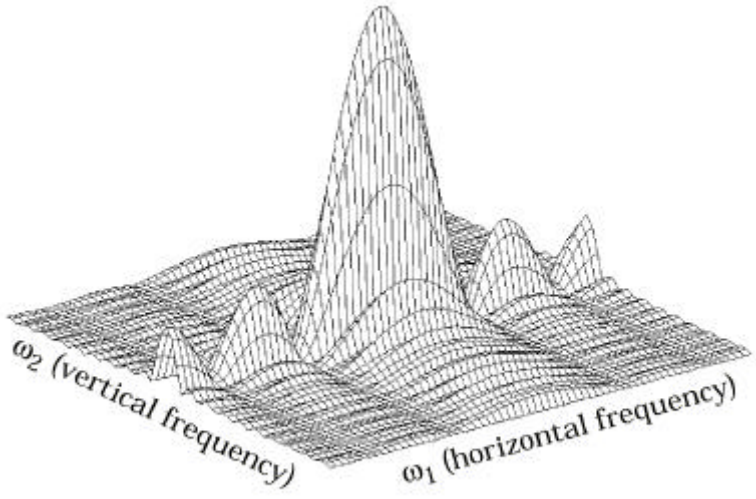

Figura 2.19. $|F(\omega_1,\omega_2)|$

O pico do centro do debuxo é F(0,0), que é a suma de tódolos valores en f(m,n).

Outra maneira de visualiza-la transformada de Fourier é visualizar log | F(ω1,ω2)| como unha imaxe :

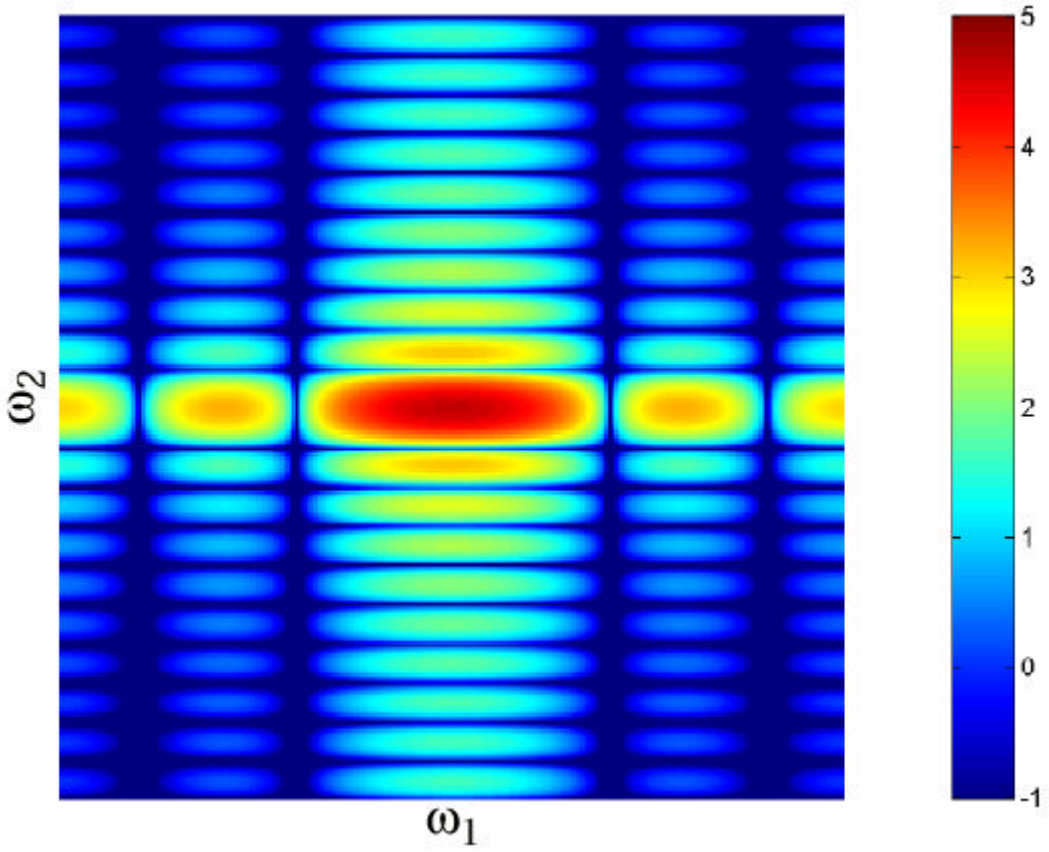

Figura 2.20. log|F(ω_1,ω_2)|.

Usar este tipo de representación axúdanos a ver con máis claridade rexións da transformada de Fourier é moi próxima a cero.

Nas seguintes figuras vemos algúns exemplos de transformadas de Fourier:

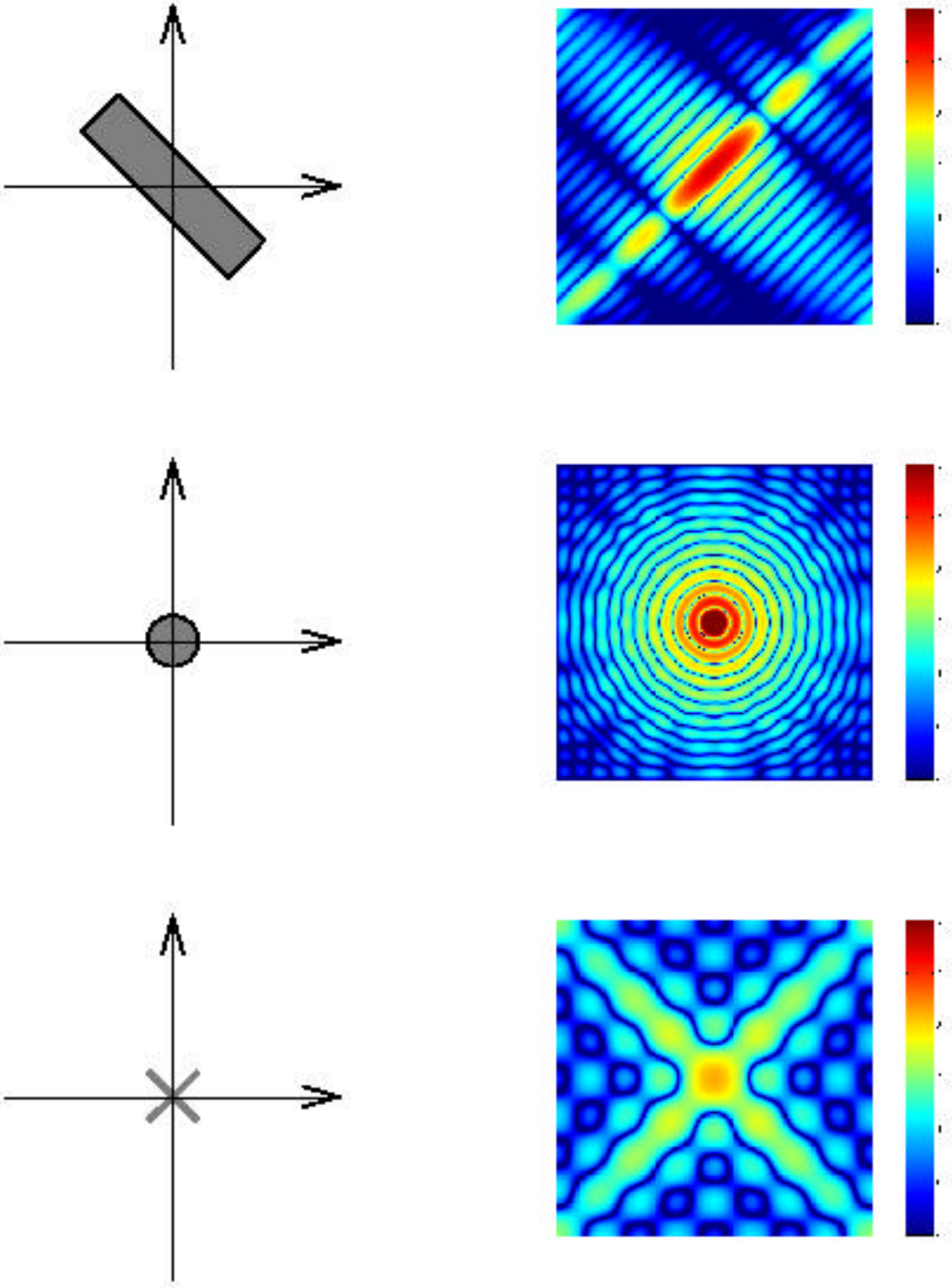

Figura 2.21. Tres exemplos de transformadas de Fourier

Nas imaxes da figura 2.21 comprobamos como unha raia produce unha transformada de Fourier en dirección perpendicular.

2.1.5.2.- A transformada de Fourier Discreta

Traballar coa transformada de Fourier nun ordenador implica usar unha forma da transformada coñecida como transformada de Fourier discreta DFT. Hai dúas principais razóns para usar esta variante :

. A entrada e a saída da DFT son discretas o que fai que sexan máis axeitadas para traballar con ordenador
. Hai un algoritmo rápido para calcula-la DFT coñecido como transformada de Fourier rápida (FFT).

A DFT está definida para unha función discreta f(m,n) que é non cero so sobre a rexión $0 \le m \le M-1$ e $0 \le n \le N-1$. As definicións DFT bidimensional MxN e a inversa da DFT MxN son :

$$F(p,q) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n) e^{-j(2\pi/M)pm} e^{-j(2\pi/N)qn} \qquad \begin{matrix} p = 0, 1, \ldots, M-1 \\ q = 0, 1, \ldots, N-1 \end{matrix}$$

$$f(m,n) = \frac{1}{MN} \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} F(p,q) e^{j(2\pi/M)pm} e^{j(2\pi/N)qn} \qquad \begin{matrix} m = 0, 1, \ldots, M-1 \\ n = 0, 1, \ldots, N-1 \end{matrix}$$

As funcións *fft, fft2 e fftn* de Matlab implementan o algoritmo da transformada de Fourier rápida para calcula-la DFT unidimensional, a DFT bidimensional, e a DFT N-dimensional, respectivamente. As funcións *ifft, ifft2* e *ifftn* computan a inversa da DFT.Os coeficientes da DFT F(p,q) son mostras da transformada de Fourier F(ω1,ω2):

$$F(p,q) = F(\omega_1, \omega_2)\Big|_{\substack{\omega_1 = 2\pi p/M \\ \omega_2 = 2\pi q/N}} \qquad \begin{matrix} p = 0, 1, \ldots, M-1 \\ q = 0, 1, \ldots, N-1 \end{matrix}$$

Exemplo

Imos construír unha matriz f similar á función f(m,n) do exemplo anterior

```
f=zeros(30,30)
f(5:24,13:17)=1;
imshow(f,'notruesize');
```

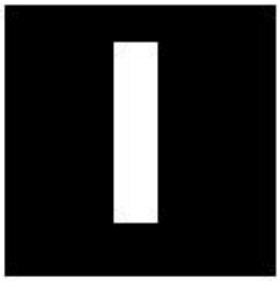

Figura 2.22. f(m,n).

Computamos e visualizamos unha DFT de 30x30 de f cos seguintes comandos:

```
F=fft2(f);
F2=log(abs(F));
imshow(F2,[-1 5],'notruesize');colormap(jet);colorbar
```

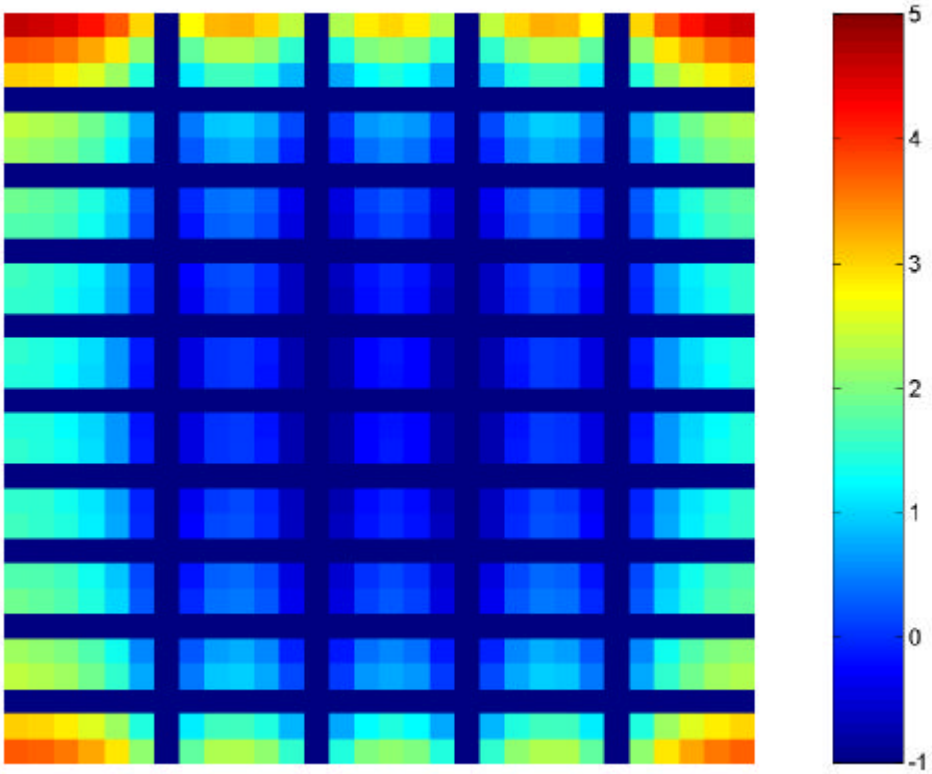

Figura 2.23. DFT 30x30 de f

Este debuxo é distinto ó debuxo da transformada anterior en varios aspectos . Primeiro, a resolución é moito menor e segundo o coeficiente DC aparece nas esquinas do debuxo e non no centro.

A resolución pode ser incrementada cubrindo con ceros f cando computamo-la súa DFT. O recheo con ceros e o cálculo da DFT podemos facelo nun so paso con este comando:

F=fft2(f,256,256);

Este comando enche con ceros f para que sexa unha matriz 256x256 antes de calcula-la DFT. O resultado é que tomamos moitas máis mostras da transformada de Fourier.

Imshow(log(abs(F)),[–1 5]); colormap(jet); colorbar

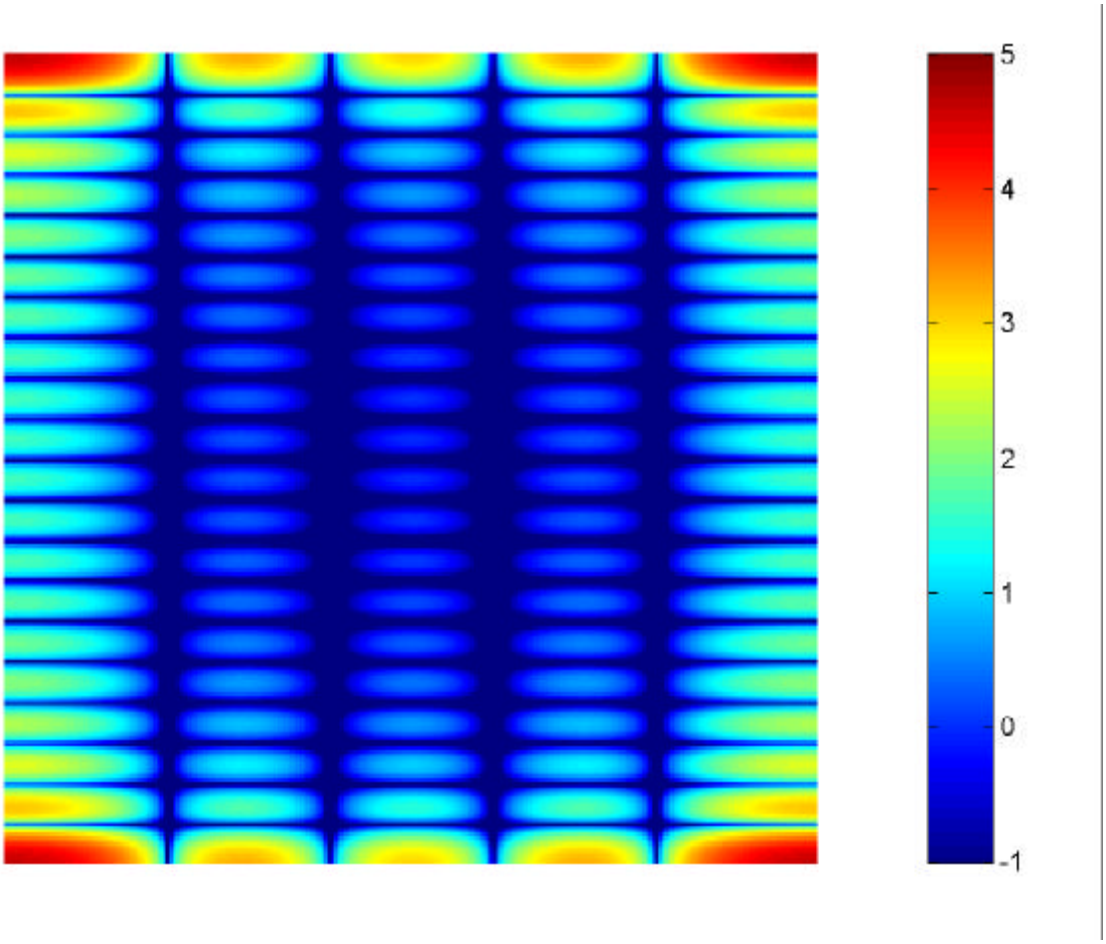

Figura 2.24. Resultado obtido se engadimos ceros

O coeficiente DC aínda segue nas esquinas do debuxo. Para iso podemos usa-la función *fftshift.*

```
F = fft2(f,256,256);
F2 = fftshift(F);
imshow(log(abs(F2)),[–1 5]); colormap(jet); colorbar
```

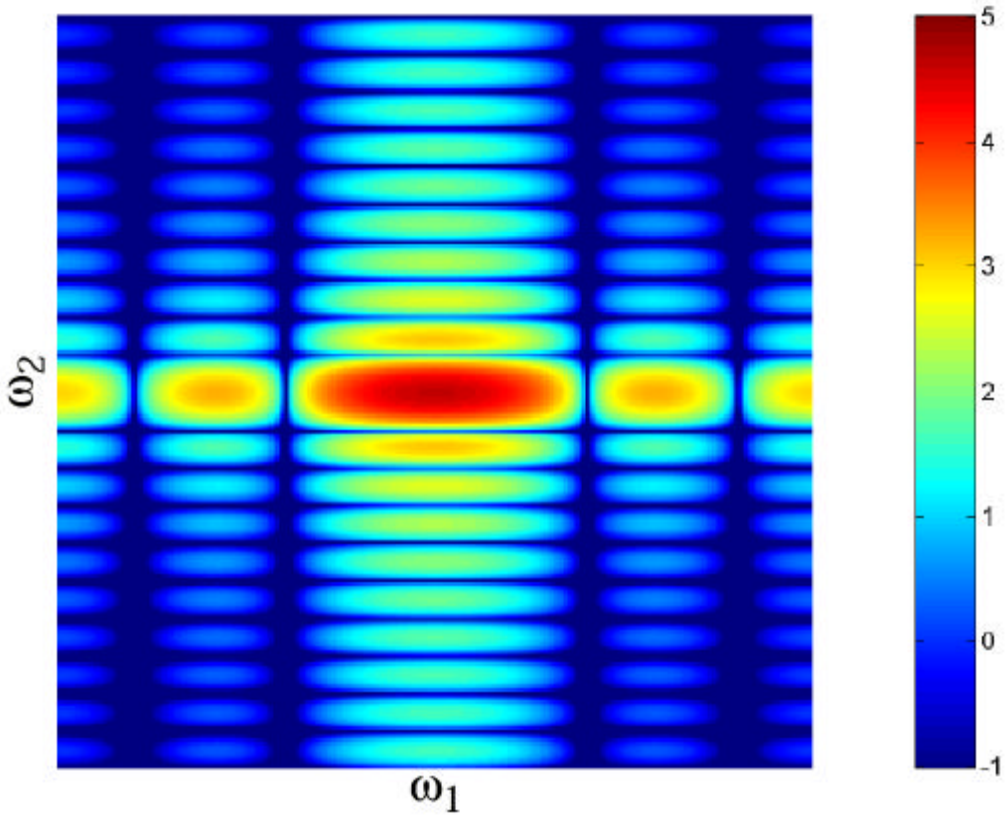

Figura 2.25. Resultado despois de aplicar *fftshift.*

2.1.5.3.- Aplicacións

Neste apartado veremos unha serie de aplicacións de procesado de imaxe relacionadas coa transformada de Fourier.

Resposta en frecuencia de filtros lineais.

A transformada de Fourier da resposta impulsional dun filtro lineal dano-la resposta en frecuencia de dito filtro. A función *freqz2* calcula e visualiza a resposta en frecuencia dun filtro. A resposta en frecuencia dun *kernel* Gaussiano mostra que o filtro "deixa pasar" as frecuencias baixas e atenúa as frecuencias altas.

```
h=fspecial('gaussian');
freqz2(h)
```

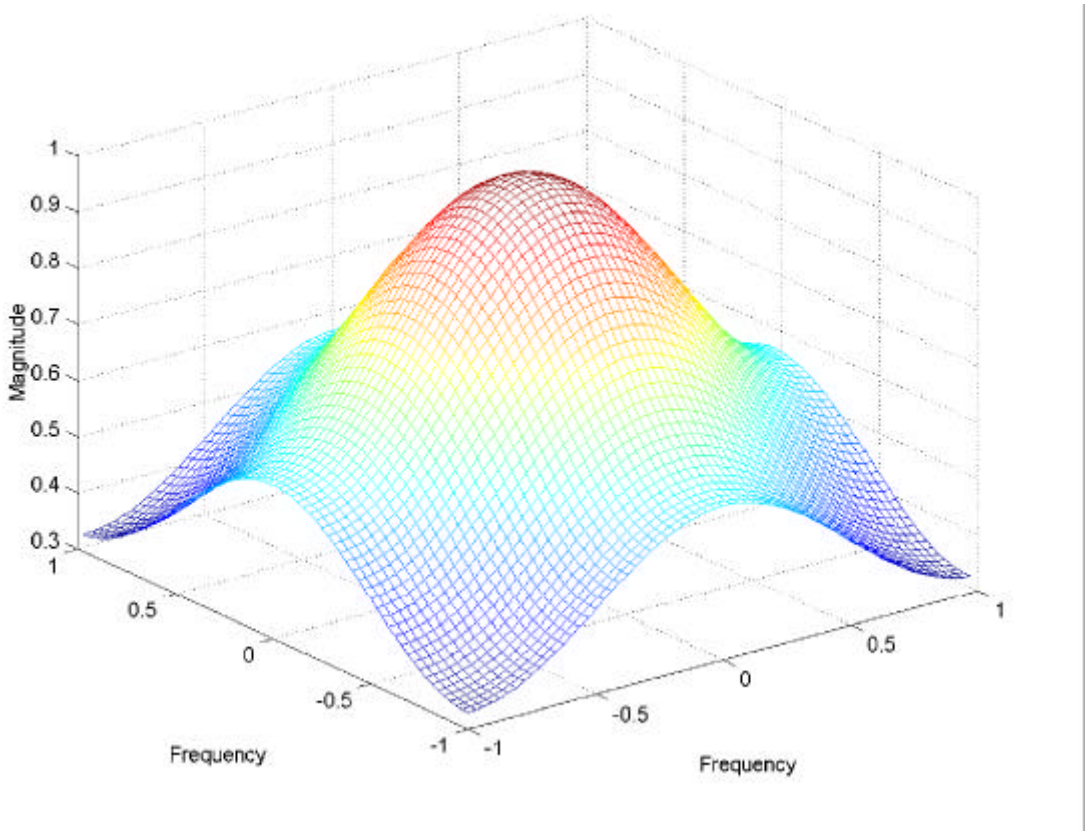

Figura 2.26. Resposta en frecuencia dun filtro gaussiano.

Consultar apartado sobre "Filtrado Lineal e Deseño de Filtros" para máis información sobre filtrado lineal, deseño de filtros e respostas en frecuencia.

Algoritmo convolución rápida (Fast Convolution).

Unha das propiedades máis importantes da transformada de Fourier é que que a multiplicación de 2 transformadas de Fourier correspóndese coa convolución asociada as funcións espaciais. Esta propiedade xunto coa FFT é a base do algoritmo de convolución rápida.

Supoñamos que *A* é unha matriz MxN e *B* é unha matriz PxQ. Podemos calcula-la convolución de *A* e *B* seguindo os seguintes pasos:

. . Engadir ceros a *A* e *B* de tal xeito que as súas novas dimensións sexan polo menos (M+P-1)x(N+Q-1). O máis usual é facer que as novas dimensións sexan potencia de 2 porque a función *fft2* é máis rápida con matrices dese tamaño.
. Calcula-la DFT bidimensional de *A* e *B* usando *fft2*.
. Multiplica-las dúas DFTs
. Calcula-la inversa da DFT da multiplicación resultante do 3º paso usando *ifft2*.

Por exemplo:

```
A = magic(3);
B = ones(3);
A(8,8) = 0; % zero–pad A to be 8–by–8
B(8,8) = 0; % zero–pad B to be 8–by–8
C = ifft2(fft2(A).*fft2(B));
C = C(1:5,1:5); % extract the nonzero portion
C = real(C) % remove imaginary part caused by roundoff error
C =
8.0000    9.0000   15.0000    7.0000    6.0000
11.0000   17.0000   30.0000   19.0000   13.0000
15.0000   30.0000   45.0000   30.0000   15.0000
7.0000   21.0000   30.0000   23.0000    9.0000
4.0000   13.0000   15.0000   11.0000    2.0000
```

Este método baseado na FFT é usado máis frecuentemente para matrices de entrada máis grandes. Para matrices pequenas é máis rápido usar *filter2* ou *conv2*.

Localización de características na imaxe.

A transformada de Fourier pode ser utilizada tamén para calcular correlacións unha operación moi relacionada coa convolución.

A correlación pode ser usada para localizar características nunha imaxe, por exemplo supoñamos que queremos localiza-las ocorrencias da letra "a" na imaxe *text.tif*. A seguinte figura mostra a imaxe e a letra "a".

Figura 2.27. Texto no que se quere localiza-la letra "a" e imaxe de referencia utilizada

Podemos calcula-la correlación da imaxe coa letra "a" do seguinte xeito:

. Primeiro rotamo-la imaxe da letra "a" 180º
. Aplicamo-lo algoritmo de convolución rápida ás dúas imaxes.

Na figura 2.28 a imaxe da esquerda é o resultado da correlación; os picos brillantes correspóndense coas ocorrencias da letra "a". Podemos localiza-la posición deses picos con maior exactitude se lle establecemos un limiar á imaxe resultante da correlación. A imaxe da dereita é a resultante de establecer un limiar á imaxe da esquerda, e os puntiños brancos representan as posicións da letra "a" na imaxe.

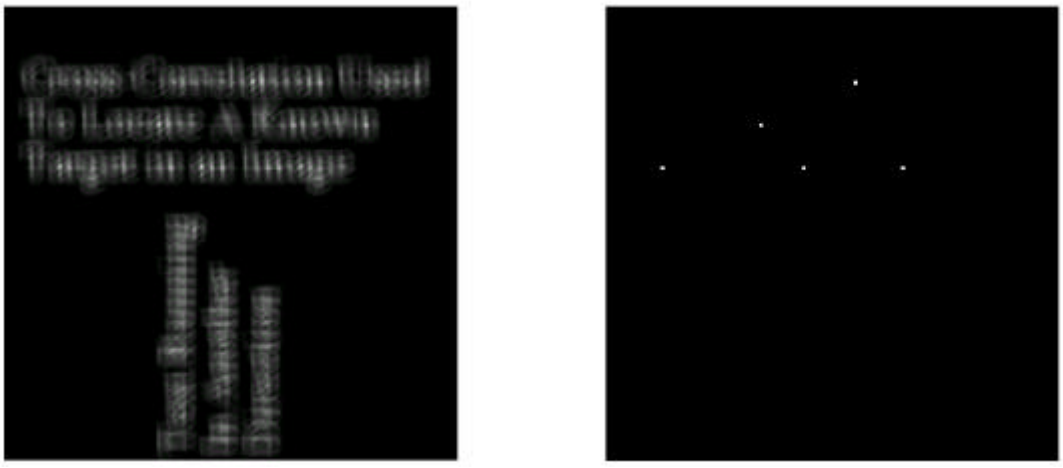

Figura 2.28.

2.1.6.- Análise e realzado de imaxes

2.1.6.1.- Histograma dunha imaxe

O histograma dunha imaxe é un gráfico que mostra a distribución de intensidades nunha imaxe indexada ou en escala de grises. A función *imhist* de Matlab é a que calcula o histograma dunha imaxe. Supoñamos que temos unha imaxe en escala de grises. O que fai a función *imhist* e conta-lo número de píxels con valor de intensidade 0, o número de píxels con valor de intensidade 1, 2, e así sucesivamente e pintar esa distribución. O proceso para imaxes indexadas é análogo.

Tamén lle podemos especificar á función *imhist* o número de divisións do eixo de ordenadas.

O exemplo seguinte mostra unha imaxe en escala de grises e o seu histograma correspondente

```
I = imread('rice.tif');
imshow(I)
figure, imhist(I,64)
```

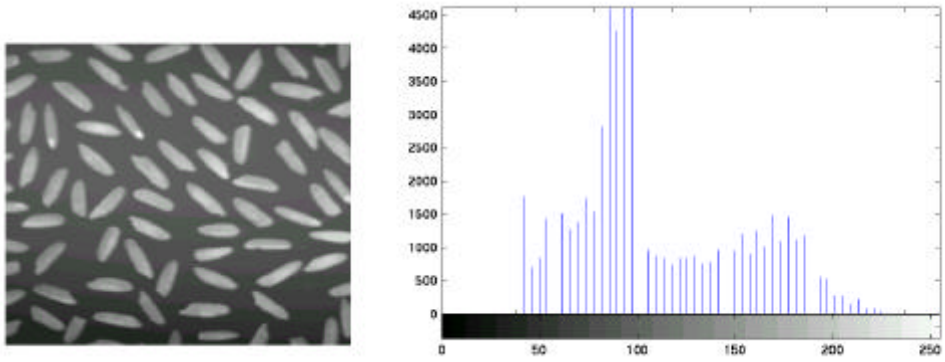

Figura 2.29. Á dereita vemo-lo histograma da imaxe da esquerda

Este histograma mostra un pico arredor do 100, debido ó gris escuro predominante no fondo da imaxe.

2.1.6.2.- Realzado de Imaxes

Dentro do realzado de imaxes incluímos toda unha serie de técnicas usadas por exemplo para incrementa-la relación sinal-ruído ou para facer máis visibles certos trazos da imaxe. Nesta sección discutimo-las seguintes técnicas de realzado de imaxes:

. Axuste de intensidade.
. Eliminación de ruído.

As funcións descritas nesta sección aplícanse principalmente a imaxes en escala de grises. De tódolos xeitos, algunhas destas funcións tamén se poden usar con imaxes de cor. Para máis información consulta-la referencia en liña de Matlab.

2.1.6.2.1.- Axuste de intensidade

O axuste de intensidade é unha técnica para mapea-los valores de intensidade a un novo rango. Por exemplo, se lle botamos unha ollada a imaxe de arroz e o seu histograma vemos que non ten moito contraste, e que o histograma non ten valores por debaixo de 40. Se remapeamo-los valores dos datos para que cubran todo o rango de intensidades de [0 255] podemos incrementa-lo contraste da imaxe.

Podemos facer este tipo de axustes coa función *imadjust*. Por exemplo, este código realiza o axuste descrito anteriormente:

```
I = imread('rice.tif');
J = imadjust(I,[0.15 0.9],[0 1]);
```

Hai que fixarse no feito de que as intensidades están especificadas no rango de 0 a 1, pola clase de I. Se I fose de clase *uint8* os valores estarían multiplicados por 255.

Na figura seguinte visualizamo-la imaxe axustada e o seu histograma

```
imshow(J)
figure , imhist(J,64)
```

Como vemos aumentou o contraste da imaxe e o histograma agora ocupa todo o rango de valores.

Do mesmo xeito podemos diminuí-lo contraste dunha imaxe estreitando o rango dos datos, escribindo o seguinte na liña de comandos de Matlab:

```
J=imadjust (I,[0,1],[0.3 0.8]);
```

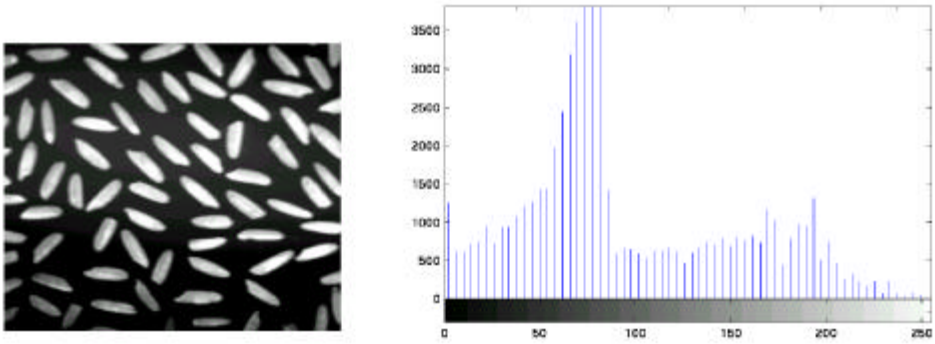

Figura 2.30. Histograma da imaxe despois do axuste.

A sintaxe xeral é

J = imadjust(I,[low high],[bottom top])

Onde os valores *low* e *high* son as intensidades na imaxe de entrada que son mapeadas a *bottom* e *top* na imaxe de saída.

Ademais de poder aumentar e diminuí-lo contraste podemos realizar outra serie de operacións de realzado da imaxe con *imadjust*. No exemplo seguinte vemos na figura que o abrigo do home é moi escuro para revelar calquera detalle. A chamada a *imadjust* mapea o rango [0,51] na imaxe (que é de tipo *uint8)* a un novo rango [128,255]. Esto fai que a imaxe teña máis brillo e expande o rango dinámico das porcións escuras da imaxe orixinal, facendo máis doado distinguir detalles do abrigo do home.

```
I = imread('cameraman.tif');
J = imadjust(I,[0 0.2],[0.5 1]);
imshow(I)
figure, imshow(J)
```

Figura 2.31. Imaxes antes e despois de aplicar *imadjust.*

Hai que ter en conta que con esta operación moitas partes da imaxe foron "borradas" isto é así porque tódolos valores po riba de 51 na imaxe orixinal foron mapeados a 255 na imaxe axustada.

2.1.6.2.2.- Ecualización de histograma

O proceso de axusta-la intensidade nas imaxes podemos face-lo automaticamente usando a función *histeq. histeq* realiza un ecualizado de histograma, operación que consiste en transforma-los valores de intensidade da imaxe de tal xeito que o histograma da imaxe resultante "encaixe" máis ou menos un histograma especificado. (Por defecto a función *histeq* intenta facer que o histograma da nova imaxe sexa un histograma o máis plano posible)

O exemplo seguinte ilustra o uso de *histeq* para facer un axuste de histograma. A imaxe orixinal ten pouco contraste, coa maioría dos valores no medio do rango de intensidades. *histeq* produce unha imaxe de saída tendo valores o máis igualmente distribuídos posible a longo do rango:

```
I = imread('pout.tif');
J = histeq(I);
imshow(I)
figure, imshow(J)
```

Figura 2.32. Imaxes antes e despois dunha ecualización de histograma

Na figura 2.33 vemo-los histogramas das imaxes anteriores

```
imhist(I)
figure, imhist(J)
```

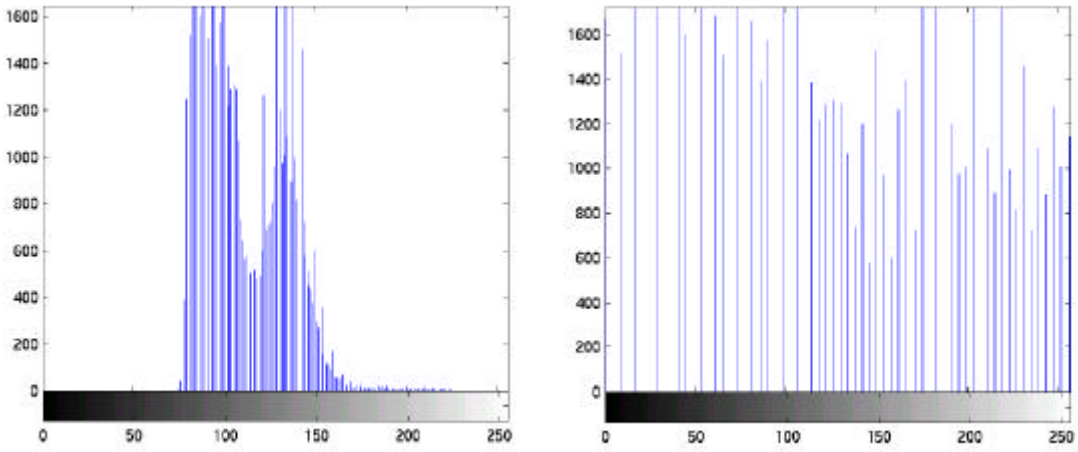

Figura 2.33.

2.1.6.2.3.- Eliminación de ruído

Nas imaxes dixitais pode haber ruído que pode ser introducido na imaxe de moitos xeitos, dependendo de cómo sexa creada a imaxe.

O toolbox de imaxe de Matlab dispón dunha serie de ferramentas para poder elimina-lo ruído nunha imaxe . Métodos distintos producen resultados mellores ou peores segundo sexa o tipo de ruído.

Algúns métodos son:

Filtrado lineal
Filtrado de mediana

Para poder simula-los efectos dos distintos tipos de ruído o toolbox dispón da función *imnoise*, que podemos usar para engadirlle a unha imaxe ruído e poder proba-los distintos métodos de eliminación de ruído.

2.1.6.2.3.1.- Filtrado lineal

Podemos usar filtrado lineal para eliminar certos tipos de ruído. Algúns filtros lineais como o filtro de promediado ou o filtro gaussiano son axeitados para ese propósito. Por exemplo, un filtro de promediado pode ser útil para eliminar ruído granular. Este tipo de filtros son filtros paso baixo, xa que eliminan o ruído de alta frecuencia. (o ruído granular é ruído de alta frecuencia).

2.1.6.2.3.2.- Filtrado de mediana

O filtrado de mediana é similar ó filtro de promediado, no sentido de que o valor de cada pixel toma o valor dunha "media" dos píxels veciños. No filtrado de mediana o

valor do pixel de saída ven determinado polo "valor mediano" dos píxels da veciñanza. Este valor é moito máis robusto co valor medio ós valores extremos.

A función do toolbox de Imaxe que implementa o filtrado de mediana é *medfilt2*. No exemplo seguinte comparamo-lo filtrado de mediana co promediado. O tipo de ruído que se lle engade á imaxe chámase "salt and pepper" que consiste en que píxels aleatorios son postos a branco ou negro (os extremos do rango de valores). En ámbolos dous casos o tamaño da veciñanza é 3x3.

Primeiro lerémo-la imaxe é engadirémoslle ruído

```
I = imread('eight.tif');
J = imnoise(I,'salt & pepper',0.02);
imshow(I)
figure, imshow(J)
```

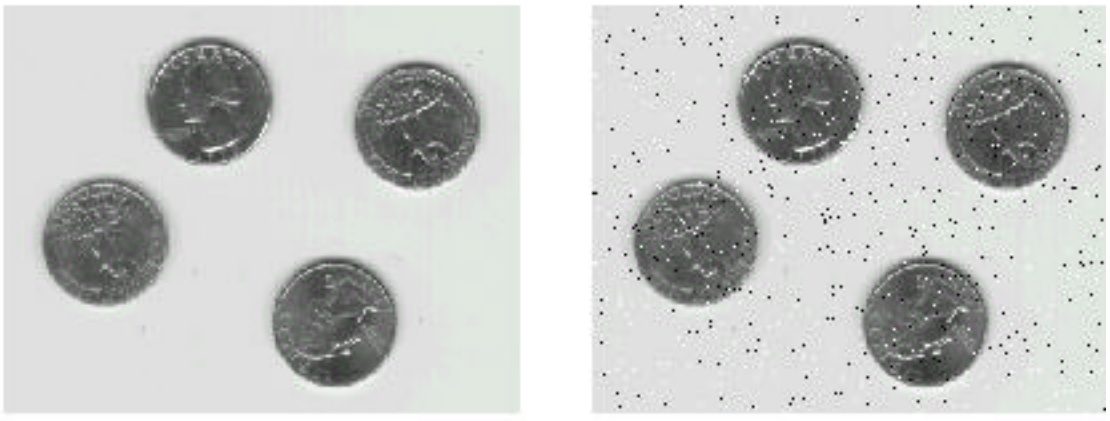

Figura 2.34. Imaxe orixinal e imaxe á que se lle engade ruído "salt and pepper".

Agora filtramo-lo ruído, visualizamo-los resultados e vemos que co filtrado de mediana conseguimos mellores resultados que co filtrado promediado.

```
K = filter2(fspecial('average',3),J)/255;
L = medfilt2(J,[3 3]);
imshow(K)
figure, imshow(L)
```

Os resultados obtidos son os da figura 2.35. Na imaxe da esquerda temo-la imaxe resultante de aplicar un filtro promediado á imaxe da dereita da figura 2.34. E na da esquerda o resultado de aplicar un filtro de mediana. Vemos que para o ruído granular o filtro que da mellores resultados é o filtro de mediana, xa que é, como dixemos antes máis robusto ós valores extremos.

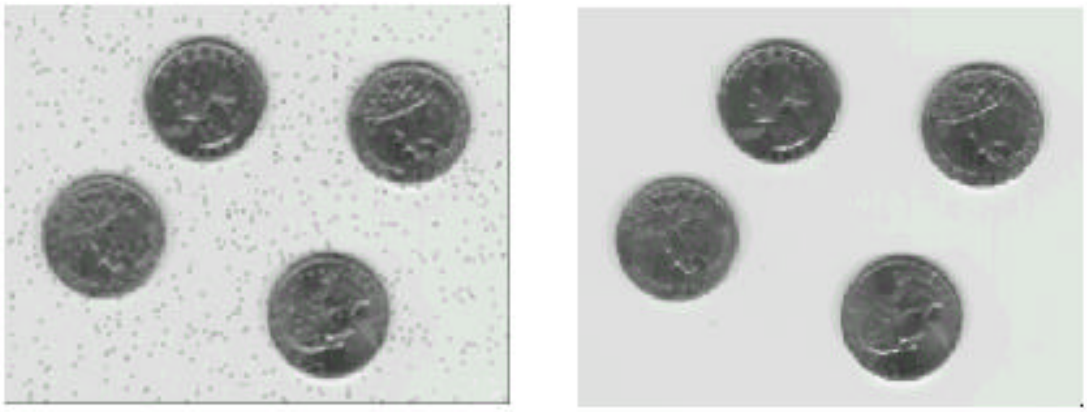

Figura 2.35. Na imaxe da dereita vemo-lo resultado de aplicar un filtro de mediana. E na imaxe da esquerda un filtro de promediado.

2.1.7.- Operacións morfolóxicas

As operacións morfolóxicas son métodos para procesar imaxes binarias baseados en formas. Estas operacións toman unha imaxe binaria como entrada e devolven unha imaxe binaria como saída. O valor de cada pixel na imaxe de saída está calculado en base ó correspondente pixel de entrada e os seus veciños. Elixindo a forma da veciñanza axeitadamente podemos construír unha operación morfolóxica sensible a formas específicas na imaxe de entrada.

2.1.7.1.- Dilatación e Erosión

As principais funcións que realizan operacións morfolóxicas son *dilation* e *erosion.* A dilatación e a erosión son operacións relacionadas aínda que producen resultados moi distintos. A dilatación engade píxels ás "fronteiras" dos obxectos, é dicir cambíaos de *off* a *on*. A erosión en cambio quita píxels dos bordes dos obxectos, é dicir cambíaos de *on* a *off.*

Cada dilatación e cada erosión usan unha veciñanza específica. O estado (*on-off)* dun píxel na imaxe de saída ven determinado pola aplicación dunha regra á veciñanza do píxel correspondente na imaxe de entrada. A regra usada define se a operación é unha dilatación ou unha erosión:

. Para unha dilatación , se algún pixel na veciñanza do pixel da imaxe de entrada está en *on*, o pixel de saída estará en *on.* Se non o pixel estará en *off.*
. Para unha erosión, se tódolos píxels na veciñanza do pixel da imaxe de entrada están a *on* o pixel de saída estará a *on.* Se non o pixel de saída estará a *off.*

A veciñanza para unha dilatación ou unha erosión pode ser de calquera forma e tamaño. A veciñanza está representada por unha matriz formada só por 0´s e 1´s . O pixel central representará ó pixel de interese mentres que os elementos da matriz que estean a 1 definirán a veciñanza.

O pixel central é o elemento

floor((size(SE)+1)/2)

da matriz que define a veciñanza (SE é dita matriz). Por exemplo nunha matriz 4x7 o pixel central será o (2,4). Cando construímos unha matriz para definir unha veciñanza temos que estar seguros de que o píxel central é o píxel que nos queremos. Por exemplo se queremos unha veciñanza 3x3 onde o píxel central sexa o píxel da esquina superior esquerda teremos que definir unha matriz como esta

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

Imos chamarlle a esta matriz *elemento estructurante.* Para unha erosión, a veciñanza consistirá nos píxels que están a 1 no *elemento estructurante.*

Para unha dilatación a veciñanza consistirá nos píxels que están a 1 no *elemento estructurante* rotado 180º. O pixel central é seleccionado antes da rotación. As operacións morfolóxicas erosión e dilatación son tamén operacións de veciñanza deslizante e polo tanto traballan do xeito que xa vimos no apartado de operacións de veciñanza deslizante.

As funcións *erode* e *dilate* do toolbox de imaxe de Matlab, son as que realizan a erosión e a dilatación respectivamente. Cada unha destas funcións toman como argumentos de entrada unha imaxe binaria e un elemento estructurante. Para unha erosión a veciñanza consistirá nos píxels que están a 1 no *elemento estructurante.* Devolven unha imaxe binaria.

O exemplo seguinte mostra unha erosion:

```
BW1 = imread('circbw.tif');
SE = eye(5);
BW2 = erode(BW1,SE);
imshow(BW1)
figure, imshow(BW2)
```

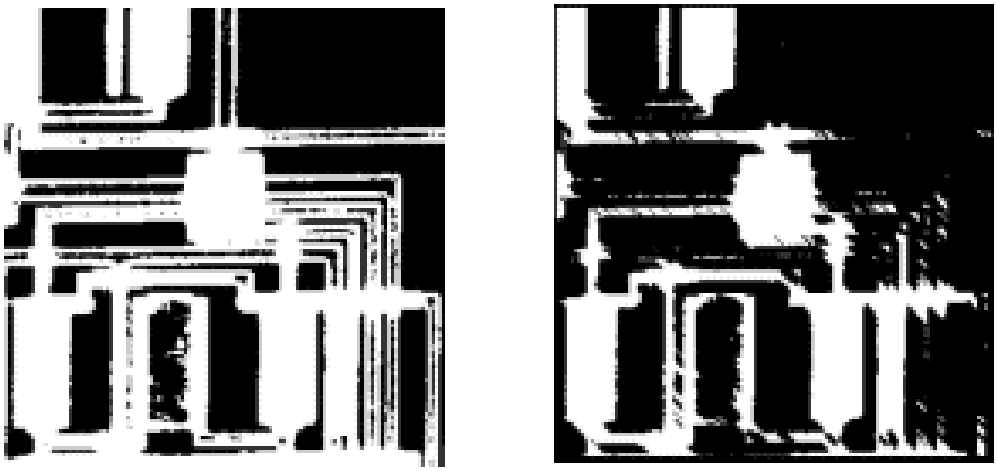

Figura 2.36. Imaxe antes (esquerda) e despois (dereita) dunha erosión.

2.1.7.1.1.- Operacións relacionadas

Hai outras operacións morfolóxicas relacionadas coa dilatación e a erosión:

.-closing : consiste nunha dilatación seguida dunha erosión co mesmo elemento estructurante.
.-opening: o contrario de closing. Consiste nunha erosión seguida dunha dilatación co mesmo elemento estructurante.

Por exemplo, supoñamos que na imaxe seguinte :

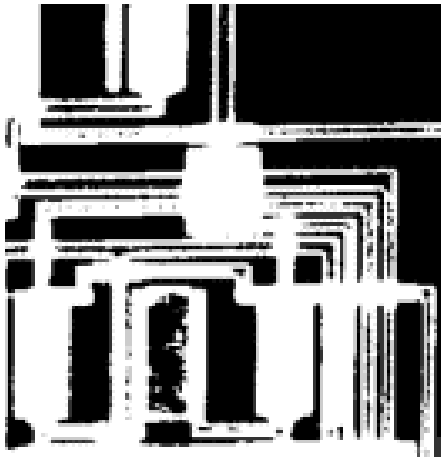

Figura 2.37.

queremos quitar tódalas liñas máis finas deixando so os rectángulos máis grandes. Podemos facelo cun opening.

Primeiro eliximos un elemento estructurante, que debería ser grande abondo para quita-las liñas cando fagamos un erode pero non tan grande como para quita-los rectángulos grandes. É dicir , eliximos un elemento estructurante que non caiba nas liñas pero si nos rectángulos. Elixiremos

SE=ones(40,30);

Agora realizamo-la erosión. Esta operación quitará as liñas finas e fará os rectángulos mais pequenos.

```
BW2 = erode(BW1,SE);
imshow(BW2)
```

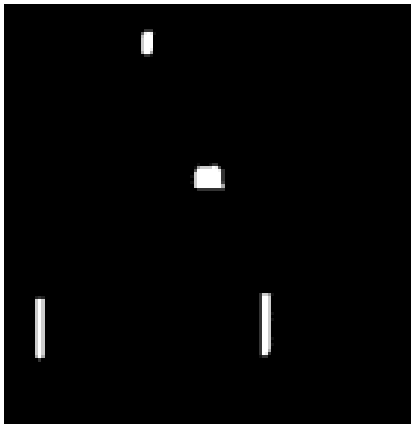

Figura 2.38. Imaxe resultante de aplicar unha erosión á figura 2.39

Despois realizamo-la dilatación co mesmo elemento estructurante para deixa-los rectángulos como estaban.

BW3 = dilate(BW2,SE);
imshow(BW3)

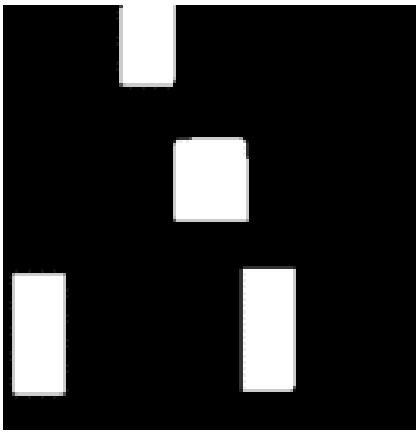

Figura 2.40. Imaxe resultante de aplicar unha dilatación á figura 2.39.

2.1.7.2.- Etiquetado

Podemos usa-la función *bwlabel* para etiquetar imaxes binarias, que é un método moi usado para marcar cada obxecto da imaxe. Definindo un tipo de conectividade e dándolle unha imaxe de entrada binaria a función *bwlabel* devolve unha matriz do mesmo tamaño que a imaxe orixinal. Os diferentes obxectos na imaxe de entrada están marcados na matriz de saída con diferentes valores de número enteiros.

Hai dous tipos de conectividade:

-veciñanza a 4 : a figura seguinte ilustra como é a veciñanza a 4, na que vemos cales son os veciños a considerar do pixel central

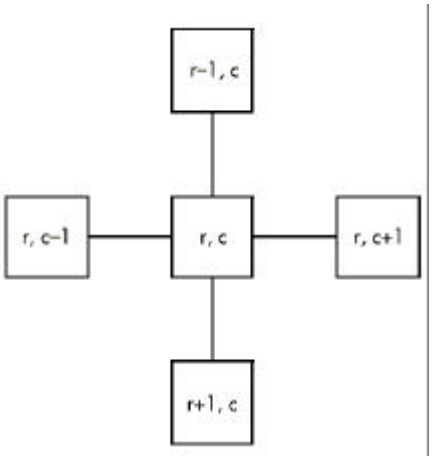

Figura 2.41. Veciñanza a 4.

-veciñanza a 8 : a figura seguinte ilustra como é a veciñanza a 8, na que vemos cales son os veciños a considerar do pixel central

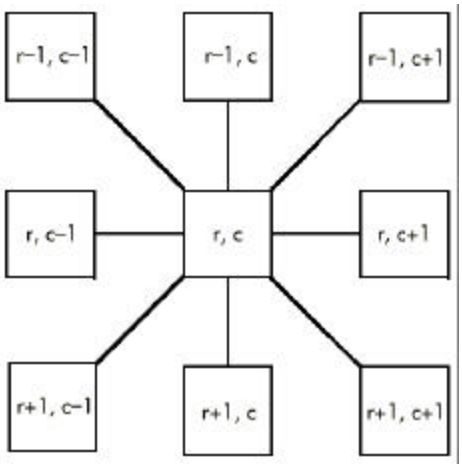

Figura 2.42. Veciñanza a 8.

Por exemplo supoñamos que temo-la seguinte imaxe binaria:

BW1 =

0	*0*	*0*	*0*	*0*	*0*	*0*	*0*
0	*1*	*1*	*0*	*0*	*1*	*1*	*1*
0	*1*	*1*	*0*	*0*	*0*	*1*	*1*
0	*1*	*1*	*0*	*0*	*0*	*0*	*0*
0	*0*	*0*	*1*	*1*	*0*	*0*	*0*
0	*0*	*0*	*1*	*1*	*0*	*0*	*0*
0	*0*	*0*	*1*	*1*	*0*	*0*	*0*
0	*0*	*0*	*0*	*0*	*0*	*0*	*0*

Se agora executamos *bwlabel*, especificando como tipo de conectividade a veciñanza a 4

bwlabel(BW1,4)

ans=

0	*0*	*0*	*0*	*0*	*0*	*0*	*0*
0	*1*	*1*	*0*	*0*	*3*	*3*	*3*
0	*1*	*1*	*0*	*0*	*0*	*3*	*3*
0	*1*	*1*	*0*	*0*	*0*	*0*	*0*
0	*0*	*0*	*2*	*2*	*0*	*0*	*0*
0	*0*	*0*	*2*	*2*	*0*	*0*	*0*
0	*0*	*0*	*2*	*2*	*0*	*0*	*0*
0	*0*	*0*	*0*	*0*	*0*	*0*	*0*

Na imaxe de saída os 1´s representan o primeiro obxecto, os 2´s o segundo e así sucesivamente. Hai que ter en conta que se usaramos conectividade a 8 (a que usa *bwlabel* por defecto) na imaxe de saída so habería 2 obxectos.

As operacións morfolóxicas realizadas polas funcións *dilate* e *erode* son de gran importancia para o procesado de imaxe e serán moi usadas ó longo deste traballo e nas prácticas da materia.