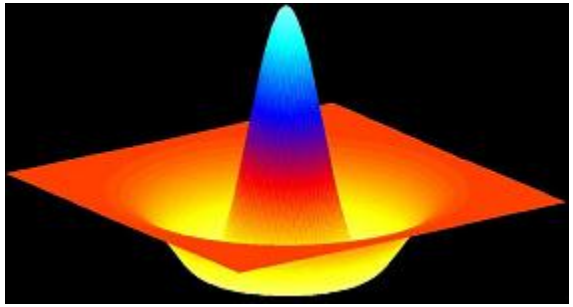


Procesamiento Digital de Imágenes

Restauración de Imágenes

Restauración de Imágenes

Restauración: Proceso por el cual se intenta reconstruir o recuperar una imagen que ha sido degradada, usando conocimiento *a priori* del proceso de degradación. Las técnicas de restauración se basan en modelos del proceso de degradación y en la aplicación del proceso inverso para recuperar la imagen original.

Modelo del Proceso de Degradación/Restauración

$$g(x,y) = H\left[\, f(x,y)\right] + \eta(x,y)$$

- $g(x,y)$: Imagen Degradada
- H: Función de Degradación
- $f(x,y)$: Imagen Original
- $\eta(x,y)$: Ruido Aditivo

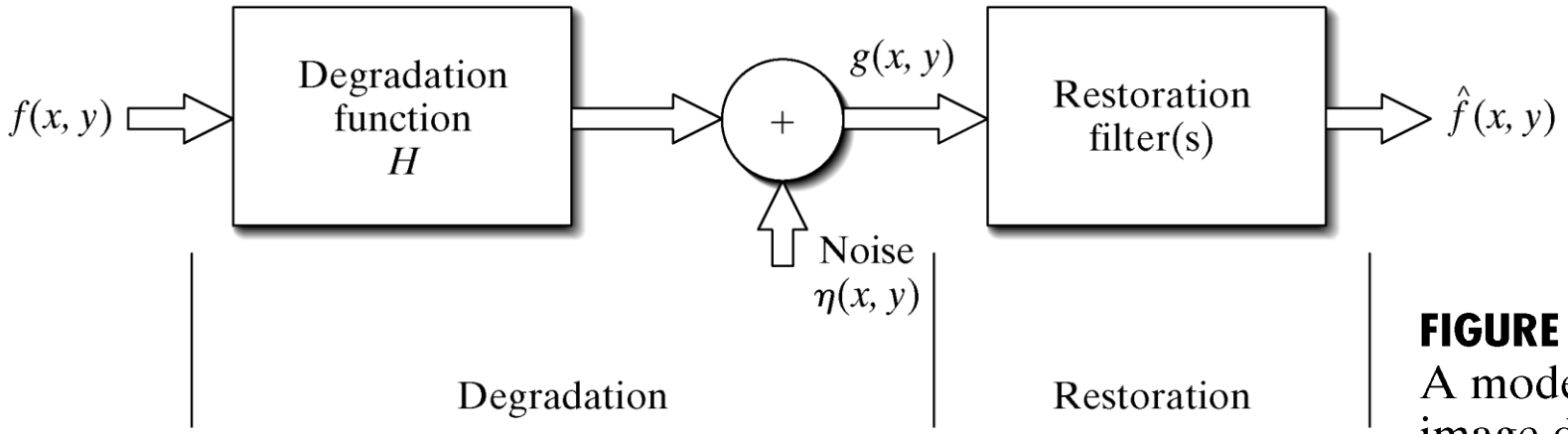


FIGURE 5.1
A model of the image degradation/ restoration process.

Objetivo: dada la imagen degradada $g(x,y)$, algún conocimiento de la función de degradación H, y algún conocimiento del ruido aditivo $\eta(x, y)$, el objetivo de la restauración es obtener una estima $\hat{f}(x, y)$ de la imagen original, que sea lo más parecida posible a la misma.

Si H es un proceso lineal y espacialmente invariante, entonces

$$g(x,y) = h(x,y) * f(x,y) + \eta(x,y)$$

donde $h(x,y)$ es la representación espacial de la función de degradación H. La representación equivalente en el dominio frecuencial resulta entonces

$$G(u,v) = H(u,v)F(u,v) + N(u,v)$$

$H(u,v)$ Optical Transfer Function (OTF)

$h(x,y)$ Point Spread Function (PSF)

Matlab → `otf2psf , psf2otf`

Modelos de Ruido

Consideraremos principalmente modelos de ruido en el dominio espacial y modelos de ruido en el dominio frecuencial. En general, consideraremos que el ruido es independiente de las coordenadas espaciales de la imagen.

Función Matlab para adicionar ruido: `imnoise`

```
g = imnoise(f, type, parameters)
```

Donde `f` es la imagen original, y `type` y `parameters`, refieren al tipo de ruido y a los parámetros que lo caracterizan, respectivamente.

Los tipos de ruido que consideraremos se muestran en la siguiente tabla.

TABLE 5.1 Generation of random variables.

Name	PDF	Mean and Variance	CDF	Generator†
Uniform	$p_z(z) = \begin{cases} \dfrac{1}{b-a} & \text{if } a \le z \le b \\ 0 & \text{otherwise} \end{cases}$	$m = \dfrac{a+b}{2}, \quad \sigma^2 = \dfrac{(b-a)^2}{12}$	$F_z(z) = \begin{cases} 0 & z < a \\ \dfrac{z-a}{b-a} & a \le z \le b \\ 1 & z > b \end{cases}$	MATLAB function `rand`
Gaussian	$p_z(z) = \dfrac{1}{\sqrt{2\pi}b} e^{-(z-a)^2/2b^2}$ $-\infty < z < \infty$	$m = a, \quad \sigma^2 = b^2$	$F_z(z) = \int_{-\infty}^{z} p_z(v)\, dv$	MATLAB function `randn`
Salt & Pepper	$p_z(z) = \begin{cases} P_a & \text{for } z = a \\ P_b & \text{for } z = b \\ 0 & \text{otherwise} \end{cases}$ $b > a$	$m = aP_a + bP_b$ $\sigma^2 = (a-m)^2 P_a + (b-m)^2 P_b$	$F_z(z) = \begin{cases} 0 & \text{for } z < a \\ P_a & \text{for } a \le z < b \\ P_a + P_b & \text{for } b \le z \end{cases}$	MATLAB function `rand` with some additional logic
Lognormal	$p_z(z) = \dfrac{1}{\sqrt{2\pi}bz} e^{-[\ln(z)-a]^2/2b^2}$ $z > 0$	$m = e^{a+(b^2/2)}, \sigma^2 = [e^{b^2} - 1]e^{2a+b^2}$	$F_z(z) = \int_0^z p_z(v)\, dv$	$z = ae^{bN(0,1)}$
Rayleigh	$p_z(z) = \begin{cases} \dfrac{2}{b}(z-a)e^{-(z-a)^2/b} & z \ge a \\ 0 & z < a \end{cases}$	$m = a + \sqrt{\pi b/4}, \sigma^2 = \dfrac{b(4-\pi)}{4}$	$F_z(z) = \begin{cases} 1 - e^{-(z-a)^2/b} & z \ge a \\ 0 & z < a \end{cases}$	$z = a + \sqrt{b \ln[1 - U(0,1)]}$
Exponential	$p_z(z) = \begin{cases} ae^{-az} & z \ge 0 \\ 0 & z < 0 \end{cases}$	$m = \dfrac{1}{a}, \quad \sigma^2 = \dfrac{1}{a^2}$	$F_z(z) = \begin{cases} 1 - e^{-az} & z \ge 0 \\ 0 & z < 0 \end{cases}$	$z = -\dfrac{1}{a}\ln[1 - U(0,1)]$
Erlang	$p_z(z) = \dfrac{a^b z^{b-1}}{(b-1)!} e^{-az}$ $z \ge 0$	$m = \dfrac{b}{a}, \quad \sigma^2 = \dfrac{b}{a^2}$	$F_z(z) = \left[1 - e^{-az} \sum_{n=0}^{b-1} \dfrac{(az)^n}{n!}\right]$ $z \ge 0$	$z = E_1 + E_2 + \cdots + E_b$ (The E's are exponential random numbers with parameter a.)

† $N(0,1)$ denotes normal (Gaussian) random numbers with mean 0 and a variance of 1. $U(0,1)$ denotes uniform random numbers in the range $(0,1)$.

Ruido Gaussiano: sensores de imagen actuando con bajos niveles de iluminación.

Ruido Salt & Pepper: fallas en dispositivos de conmutación

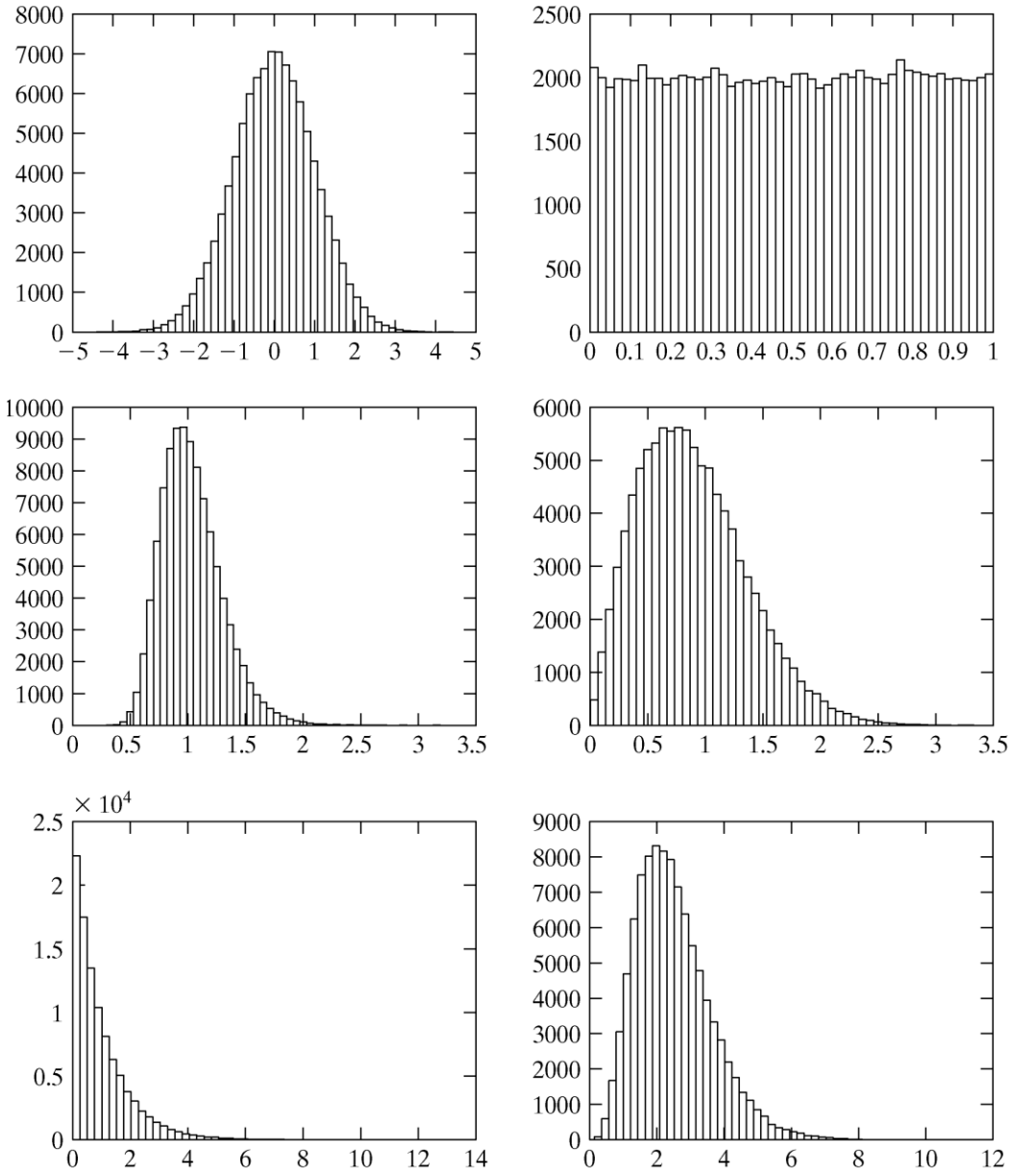
Ruido Lognormal: el tamaño de partículas de plata en emulsiones fotográficas tiene una distribución lognormal.

Ruido Rayleigh: aparece en *range imaging.*

Ruido Exponencial y Erlang: es de utilidad para describir el ruido en imágenes laser.

Función Matlab para generar ruido

```
R = imnoise2(type, M, N, a, b)
```



a b
c d
e f

FIGURE 5.2
Histograms of random numbers: (a) Gaussian, (b) uniform, (c) lognormal, (d) Rayleigh, (e) exponential, and (f) Erlang. In each case the default parameters listed in the explanation of function `imnoise2` were used.

Fig.5.2(a) fue generada con los comandos:

```
>>r = imnoise2('gaussian', 100000, 1, 0, 1);
>>p = hist(r, 50);
```

Ruido Periódico: aparece en imágenes debido a interferencia eléctrica o electromecánica durante la adquisición. Es dependiente de las coordenadas en la imagen, y típicamente se recurre a filtrado en el dominio frecuencial para su atenuación.

Lo modelamos como:

$$r(x,y) = A\sin\left[2\pi u_0(x+B_x)/M + 2\pi v_0(y+B_y)/N\right]$$

Donde A es la amplitud, u_0, v_0 son las frecuencias respecto a los ejes x e y, respectivamente, y B_x, B_y son desplazamientos de fase respecto al origen.

Función Matlab: **`imnoise3`**

```
>>[r,R,S] = imnoise3(M,N,C,A,B);
```

Ruido

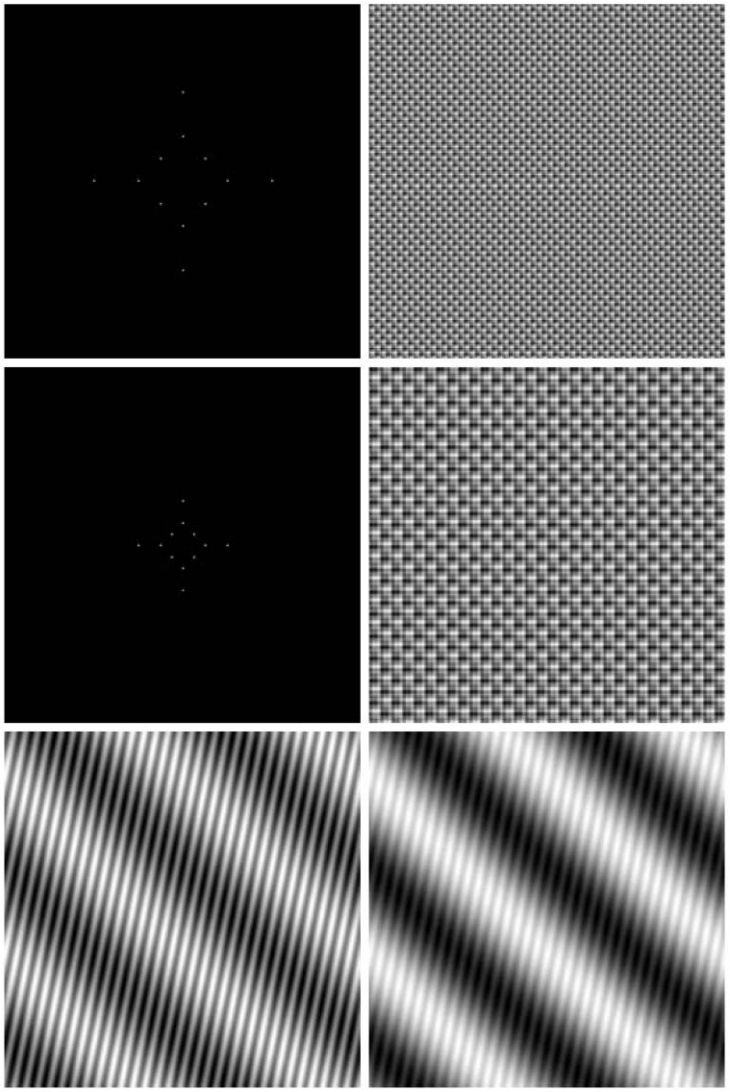
Transformada

Espectro

Coordenadas frecuenciales

Amplitudes

Fases de c/impulso



a b
c d
e f

FIGURE 5.3
(a) Spectrum of specified impulses. (b) Corresponding sine noise pattern. (c) and (d) A similar sequence. (e) and (f) Two other noise patterns. The dots in (a) and (c) were enlarged to make them easier to see.

```
>>C=[0 64;0 128;32 32;64 0;128 0;…
-32 32];

>>[r,R,S]=imnoise3(512,512,C);

>>imshow(S,[])

>>figure, imshow(r,[])
```

Estimación de los parámetros del ruido

Ruido periódico: los parámetros pueden estimarse analizando el espectro de Fourier de la imagen. Aparecen como impulsos de frecuencia que incluso pueden detectarse a simple vista.

Ruido Espacial: queda completamente caracterizado por la PDF, que puede aproximarse por el histograma de la imagen. Para describir la forma del histograma suelen usarse los **momentos centrados** (respecto a la media), definidos como

$$\mu_n = \sum_{i=0}^{L-1} (z_i - m)^n p(z_i)$$

donde n es el orden del momento, L es el número de valores de intensidad posible, $p(z_i)$ es una estima de la probabilidad de ocurrencia del valor de intensidad z_i (dada por histograma normalizado) , y m es el **valor medio** definido como

$$m = \sum_{i=0}^{L-1} z_i \, p(z_i)$$

Como el histograma está normalizado, la suma de sus componentes es 1, por lo que resulta $\mu_0 = 1$, $\mu_1 = 0$, y el momento de segundo orden es la **varianza**

$$\mu_2 = \sum_{i=0}^{L-1} (z_i - m)^2 \, p(z_i)$$

Función Matlab: `statmoments`

```
[v,unv] = statmoments(p,n)
```

Momentos en rango [0, 1]

Momentos en rango original

Nro. de momentos

histograma

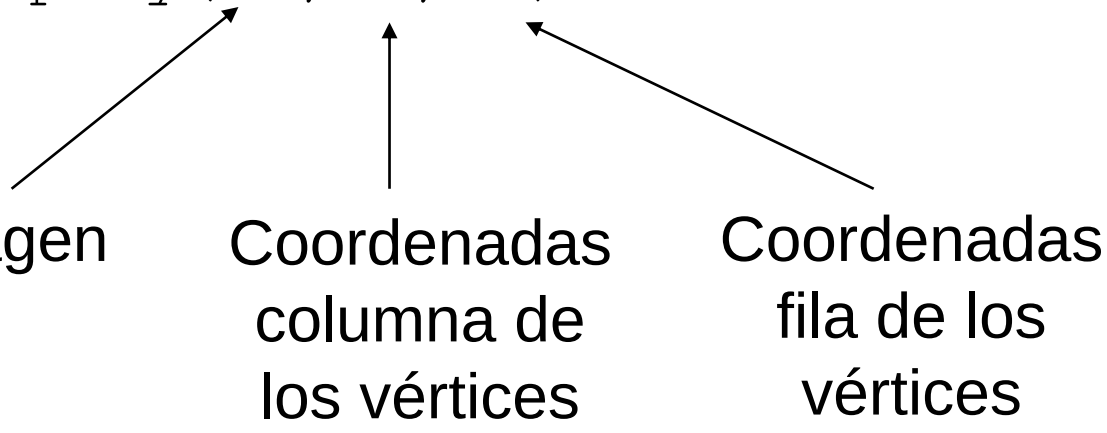
Como $\mu_0 = 1$, $\mu_1 = 0$, la función no los computa, y

$$v(1) = m \ , \ v(k) = \mu_k$$

Frecuentemente, los parámetros del ruido deben estimarse a partir de la imagen ruidosa, o de un conjunto de imágenes.

Para ello, debe seleccionarse una región de la imagen con un fondo lo más constante en intensidad posible, de manera que la variabilidad en los valores de intensidad pueda atribuirse principalmente al ruido. Para seleccionar una **Region de Interés (ROI: Region of Interest)** existe la función Matlab `roipoly` que genera una ROI poligonal.

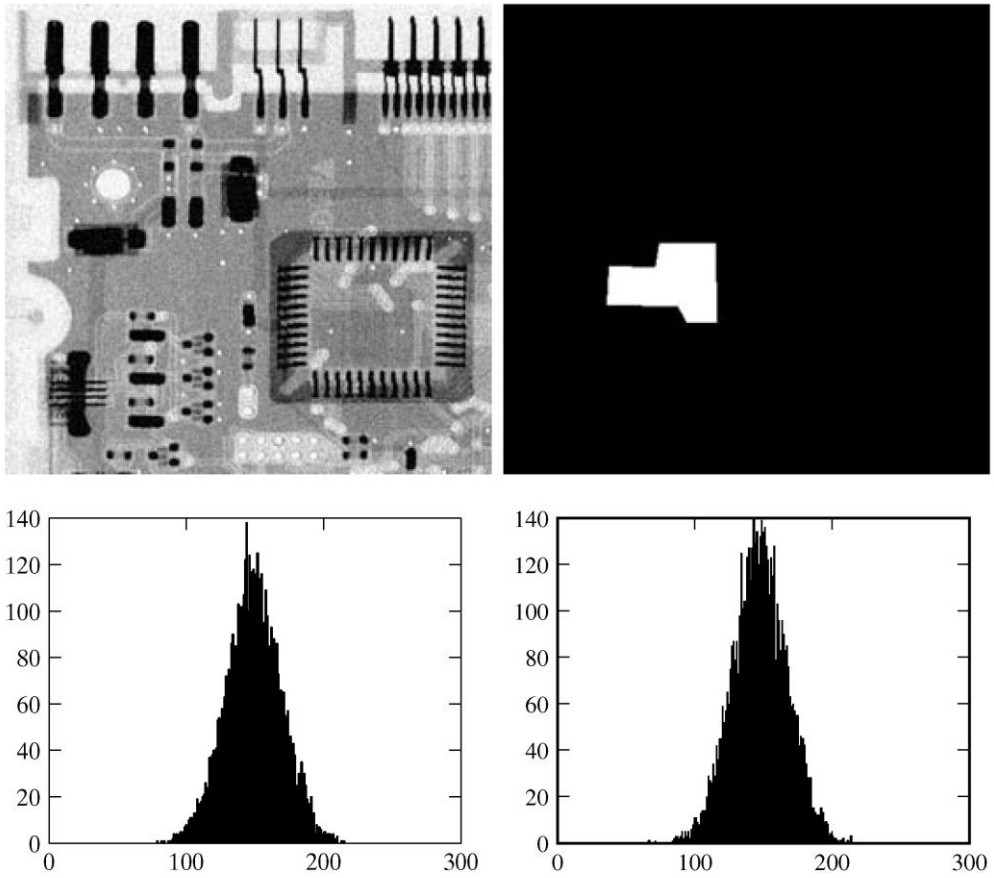
```
B = roipoly(f , c, r)
```



Ejemplo:

```
>>[B,c,r] = roipoly(f);
>>[p, npix] = histroi(f,c,r);
>>figure, bar(p,1)
```

```
>>[v,unv] = statmoments(p,2)
```



a b
c d

FIGURE 5.4
(a) Noisy image.
(b) ROI generated interactively.
(c) Histogram of ROI.
(d) Histogram of Gaussian data generated using function `imnoise2`.
(Original image courtesy of Lixi, Inc.)

Restauración en presencia sólo de ruido

En este caso el modelo de degradación resulta:

$$g(x,y) = f(x,y) + \eta(x,y)$$

y el método de reducción de ruido adecuado es filtrado espacial.

Funciones Matlab: `imfilter`, `medfilt2`, `ordfilt2`, `spfilt` (implementa los filtros de Table 5.2)

```
f = spfilt(g, type, m, n, parameter)
```

TABLE 5.2 Spatial filters. The variables m and n denote respectively the number of rows and columns of the filter neighborhhood.

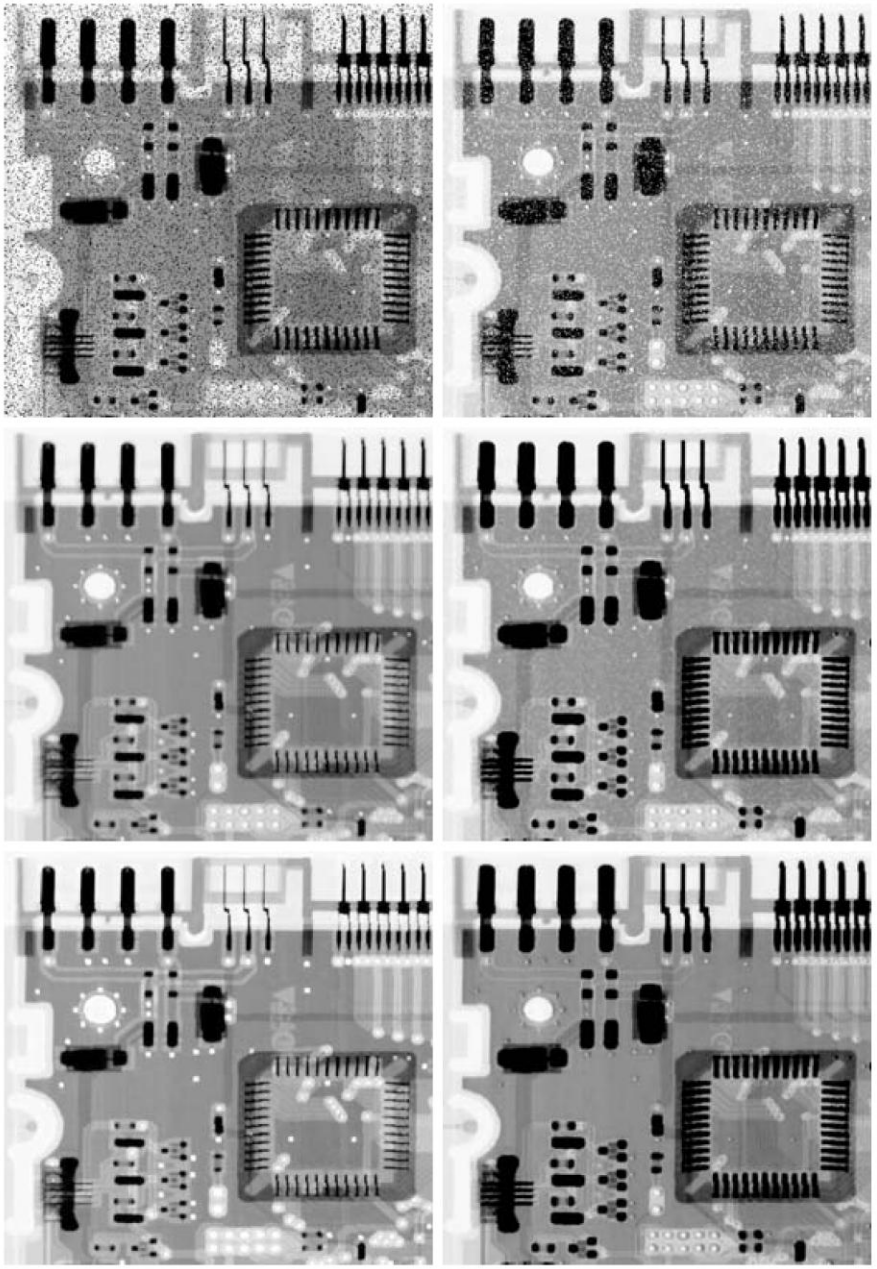
Filter Name	Equation	Comments
Arithmetic mean	$\hat{f}(x,y) = \frac{1}{mn}\sum_{(s,t)\in S_{xy}} g(s,t)$	Implemented using IPT functions `w = fspecial('average', [m, n])` and `f = imfilter(g, w)`.
Geometric mean	$\hat{f}(x,y) = \left[\prod_{(s,t)\in S_{xy}} g(s,t)\right]^{\frac{1}{mn}}$	This nonlinear filter is implemented using function `gmean` (see custom function `spfilt` in this section).
Harmonic mean	$\hat{f}(x,y) = \frac{mn}{\sum_{(s,t)\in S_{xy}} \frac{1}{g(s,t)}}$	This nonlinear filter is implemented using function `harmean` (see custom function `spfilt` in this section).
Contraharmonic mean	$\hat{f}(x,y) = \frac{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q+1}}{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q}}$	This nonlinear filter is implemented using function `charmean` (see custom function `spfilt` in this section).
Median	$\hat{f}(x,y) = \underset{(s,t)\in S_{xy}}{\text{median}}\{g(s,t)\}$	Implemented using IPT function `medfilt2`: `f = medfilt2(g, [m n])`.
Max	$\hat{f}(x,y) = \max_{(s,t)\in S_{xy}}\{g(s,t)\}$	Implemented using IPT function `ordfilt2`: `f = ordfilt2(g, m*n, ones(m, n))`.
Min	$\hat{f}(x,y) = \min_{(s,t)\in S_{xy}}\{g(s,t)\}$	Implemented using IPT function `ordfilt2`: `f = ordfilt2(g, 1, ones(m, n))`.
Midpoint	$\hat{f}(x,y) = \frac{1}{2}\left[\max_{(s,t)\in S_{xy}}\{g(s,t)\} + \min_{(s,t)\in S_{xy}}\{g(s,t)\}\right]$	Implemented as 0.5 times the sum of the max and min filtering operations.
Alpha-trimmed mean	$\hat{f}(x,y) = \frac{1}{mn-d}\sum_{(s,t)\in S_{xy}} g_r(s,t)$	The $d/2$ lowest and $d/2$ highest intensity levels of $g(s,t)$ in S_{xy} are deleted, $g_r(s,t)$ denotes the remaining $mn-d$ pixels in the neighborhood. Implemented using function `alphatrim` (see custom function `spfilt` in this section).

Ejemplo: Fig.5.5(a) es una imagen `uint8` que ha sido corrompida con ruido *pepper* con probabilidad 0.1.

```
>>[M,N]=size(f);
>>R=imnoise2('salt & pepper', M,N,0.1,0);
>>c=find(R==0);
>>gp=f;
>>gp(c)=0;
```

Fig.5.5(b) es la misma imagen corrompida con ruido *salt*

```
>>R=imnoise2('salt & pepper', M,N,0,0.1);
>>c=find(R==1);
>>gs=f;
>>gs(c)=255;
```



a b
c d
e f

FIGURE 5.5
(a) Image corrupted by pepper noise with probability 0.1. (b) Image corrupted by salt noise with the same probability. (c) Result of filtering (a) with a 3×3 contraharmonic filter of order $Q = 1.5$. (d) Result of filtering (b) with $Q = -1.5$. (e) Result of filtering (a) with a 3×3 max filter. (f) Result of filtering (b) with a 3×3 min filter.

Una buena estrategia para el filtrado del ruido *pepper* es el uso de un filtro contra-armónico con un valor positivo de *Q*

```
>>fp=spfilt(gp,'chmean',3,3,1.5);
```

Fig.5.5(c)

Similarmente, el ruido *salt* puede filtrarse usando un filtro contra-armónico, con un valor negativo de *Q*

```
>>fs=spfilt(gs,'chmean',3,3,-1.5);
```

Fig.5.5(d)

Otras alternativas son:

```
>>fpmax=spfilt(gp,'max',3,3);
```

Fig.5.5(e)

```
>>fsmin=spfilt(gs,'min',3,3);
```

Fig.5.5(f)

Comparación de técnicas de filtrado usando imágenes patrón

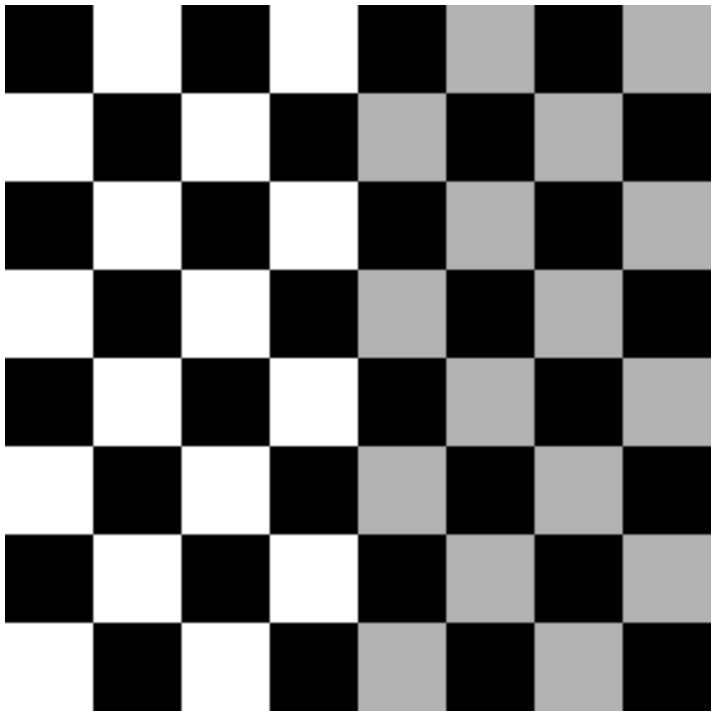
Para que la comparación tenga sentido se suele usar una misma imagen patrón sobre la cual se aplican las distintas técnicas de filtrado.

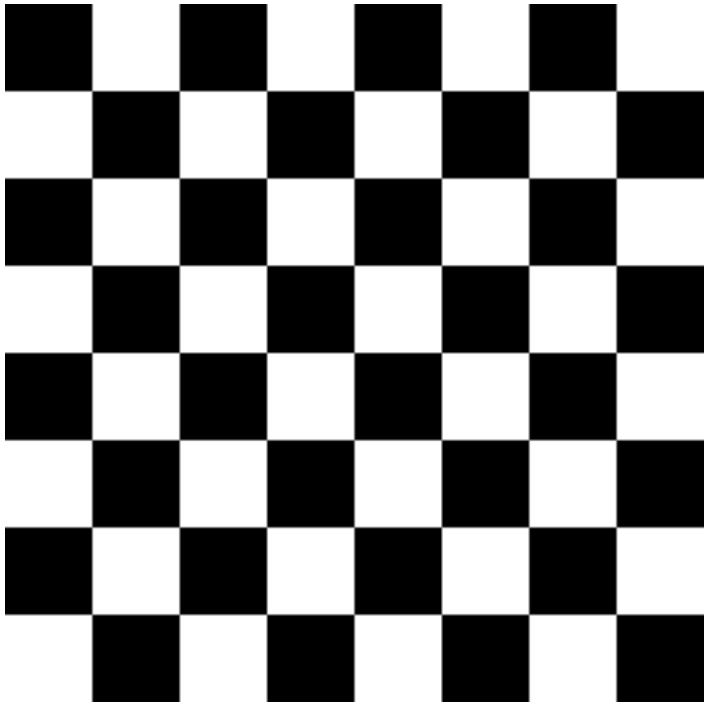
```
C = checkerboard(NP,M,N)
```

`NP:` longitud en pixels del lado de cada cuadrado (default: 10)

`M:` número de filas de la imagen

`N:` número de columnas de la imagen



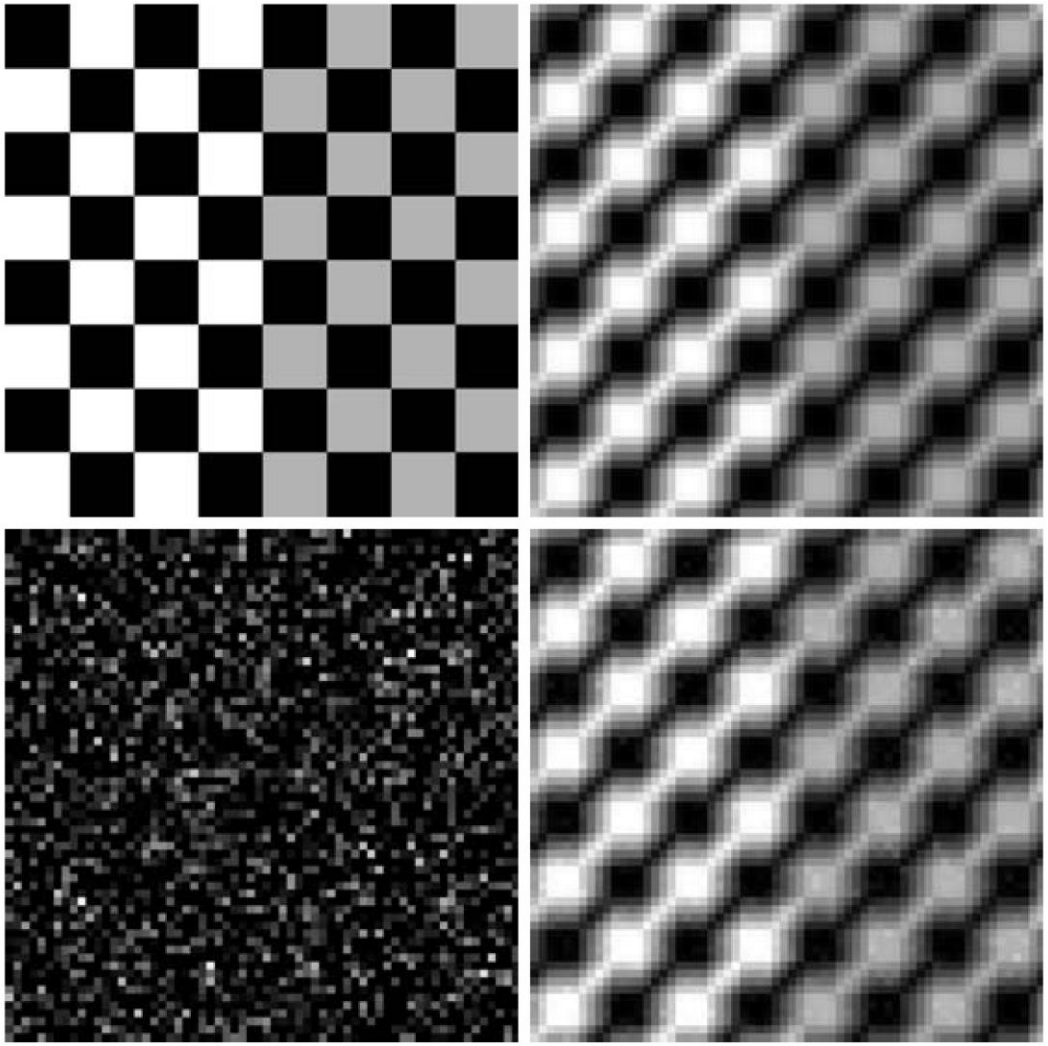


```
>> C=checkerboard(32);
>> imshow(C,[])
```

```
>> K=im2double(C>0.5);
>> imshow(K,[])
```

Ejemplo:

```
>> f = checkerboard(8);                    %imagen original
>> PSF = fspecial('motion',7,45);          %función de borrosidad
>> gb = imfilter(f,PSF,'circular');        %imagen con borrosidad
>> noise = imnoise(zeros(size(f)),'gaussian',0,0.001);
>> g = gb + noise;                         %imagen borrosa con ruido
```



a b
c d

FIGURE 5.7
(a) Original image. (b) Image blurred using `fspecial` with `len` = 7, and `theta` = –45 degrees. (c) Noise image. (d) Sum of (b) and (c).

Filtrado Inverso

La manera más simple de restaurar una imagen degradada sería formar una estima de su espectro de la forma

$$\hat{F}(u,v) = \frac{G(u,v)}{H(u,v)} = \frac{H(u,v)F(u,v) + N(u,v)}{H(u,v)}$$

$$= F(u,v) + \frac{N(u,v)}{H(u,v)}$$

y luego obtener la imagen restaurada mediante la transformada inversa de Fourier. Desafortunadamente, aunque se conozca la función de degradación $H(u,v)$, el ruido es desconocido.

Filtro de Wiener

La idea del Filtro de Wiener es buscar una estima $\hat{f}$ de la imagen original f, que minimice el error estadístico

$$e^2 = E\left\{\left(f - \hat{f} \right)^2\right\}$$

La solución en el dominio frecuencial viene dada por

$$\hat{F}(u,v) = \left[\frac{1}{H(u,v)} \frac{\left|H(u,v)\right|^2}{\left|H(u,v)\right|^2 + S_\eta(u,v) / S_f(u,v)} \right] G(u,v)$$

donde

$H(u,v)$ función de degradación

$S_\eta(u,v) = |N(u,v)|^2$ espectro de densidad de energía del ruido

$S_f(u,v) = |F(u,v)|^2$ espectro de densidad de energía de la imagen

$\frac{S_f(u,v)}{S_\eta(u,v)}$ relación señal-ruido (SNR)

La relación señal ruido se suele reemplazar por la relación señal ruido promedio

$$\frac{f_A}{\eta_A} = \frac{\frac{1}{MN}\sum_u\sum_v S_f(u,v)}{\frac{1}{MN}\sum_u\sum_v S_\eta(u,v)} = \frac{1}{R}$$

que es un escalar → **Filtro de Wiener paramétrico**

Función Matlab: **deconvwnr**

```
>>fr = deconvwnr(g,PSF);                          (1)

>>fr = deconvwnr(g,PSF,NSPR);                     (2)

>>fr = deconvwnr(g,PSF,NACORR,FACORR);            (3)
```

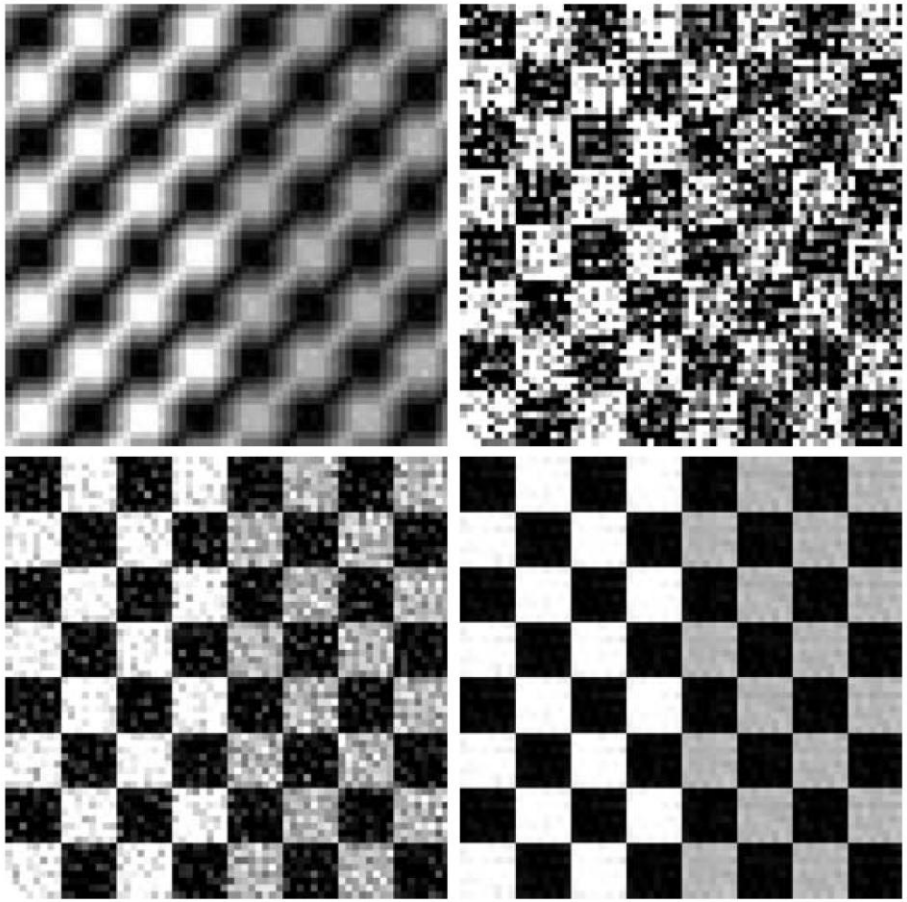
(1) Asume que el ruido es nulo.

(2) Asume que el ruido es conocido, ya sea una constante (Filtro de Wiener paramétrico) o una matriz.

(3) Asume conocidas las funciones de autocorrelación del ruido y de la imagen sin degradación

Ejemplo:

```
>> fr1 = deconvwnr(g,PSF);
```



a b
c d

FIGURE 5.8
(a) Blurred, noisy image. (b) Result of inverse filtering. (c) Result of Wiener filtering using a constant ratio. (d) Result of Wiener filtering using autocorrelation functions.

```
>> Sn = abs(fft2(noise)).^2;
>> nA = sum(Sn(:))/prod(size(noise));
>> Sf = abs(fft2(f)).^2;
>> fA = sum(Sf(:))/prod(size(f));
>> R = nA/fA;
>> fr2 = deconvwnr(g,PSF,R);
>> NCORR = fftshift(real(ifft2(Sn)));
>> FCORR = fftshift(real(ifft2(Sf)));
>> fr3 = deconvwnr(g,PSF,NCORR,FCORR);
```

Deconvolución con el algoritmo de Lucy-Richardson

El algoritmo de Lucy-Richarson permite restaurar un imagen que ha sido degradada por convolución con una PSF (Point Spread Function) y posiblemente con ruido aditivo. El algoritmo se basa en **maximizar la verosimilitud** (**maximum likelihood**) de que la imagen resultante sea una instancia de la imagen original bajo la hipótesis de una estadística de Poisson.

Maximizando la función de verosimilitud del modelo resulta en una ecuación que es satisfecha cuando la siguiente ecuación iterativa converge:

$$\hat{f}_{k+1}(x,y) = \hat{f}_k(x,y)\left[h(-x,-y) * \frac{g(x,y)}{h(x,y) * \hat{f}_k(x,y)} \right]$$

donde

$\hat{f}$ es una estima de la imagen original

g es la imagen degradada

h es la PSF

Función Matlab: **`deconvlucy`**

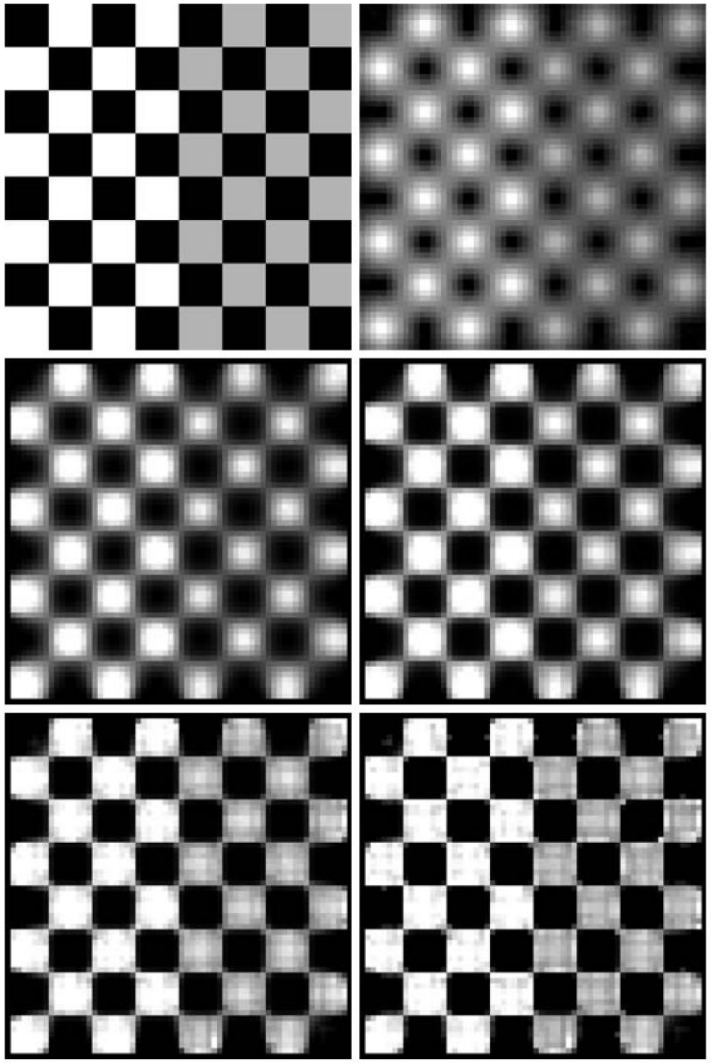
```
>>fr = deconvlucy(g,PSF,NUMIT,DAMPAR,WEIGHT);
```

`NUMIT:` número de iteraciones
`DAMPAR:` umbral de desviación de la imagen resultante respecto a la imagen degradada `g`
`WEIGHT:` matriz de pesos de cada pixel de la imagen, los pixeles de mala calidad pueden excluirse de la iteración asignándoles un peso 0.

Ejemplo:

```
I = checkerboard(8);
PSF = fspecial('gaussian',7,10);
V = .0001;
BlurredNoisy = imnoise(imfilter(I,PSF),'gaussian',0,V);
WT = zeros(size(I));
WT(5:end-4,5:end-4) = 1;
J1 = deconvlucy(BlurredNoisy,PSF);
J2 = deconvlucy(BlurredNoisy,PSF,20,sqrt(V));
J3 = deconvlucy(BlurredNoisy,PSF,20,sqrt(V),WT);

subplot(221);imshow(BlurredNoisy);
title('A = Blurred and Noisy');
subplot(222);imshow(J1);
title('deconvlucy(A,PSF)');
subplot(223);imshow(J2);
title('deconvlucy(A,PSF,NI,DP)');
subplot(224);imshow(J3);
title('deconvlucy(A,PSF,NI,DP,WT)');
```



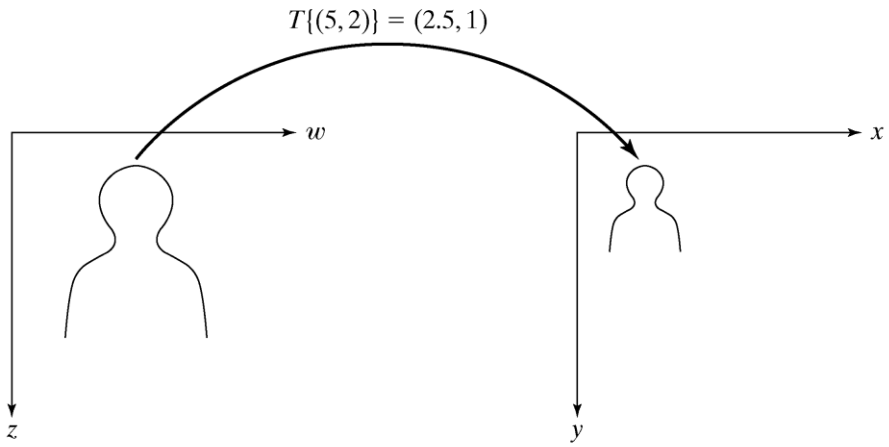
a b
c d
e f

FIGURE 5.10
(a) Original image. (b) Image blurred and corrupted by Gaussian noise. (c) through (f) Image (b) restored using the L-R algorithm with 5, 10, 20, and 100 iterations, respectively.

Transformaciones Geométricas

Supongamos que la imagen f definida sobre el sistema coordenado *(w, z)*, sufre una distorsión geométrica dando por resultado una imagen g definida sobre un sistema coordenado *(x, y)*. La transformación de coordenadas se puede expresar como

$$(x, y) = T\{(w, z)\}$$



$$(x,y) = T\{(w,z)\}=(w/2,z/2)$$

FIGURE 5.12 A simple spatial transformation. (Note that the *xy*-axes in this figure do not correspond to the image axis coordinate system defined in Section 2.1.1. As mentioned in that section, IPT on occasion uses the so-called spatial coordinate system in which *y* designates rows and *x* designates columns. This is the system used throughout this section in order to be consistent with IPT documentation on the topic of geometric transformations.)

Una forma muy usada de transformación geométrica es la denominada **transformación afín**

$$\begin{bmatrix} x & y & 1 \end{bmatrix} = \begin{bmatrix} w & z & 1 \end{bmatrix} T = \begin{bmatrix} w & z & 1 \end{bmatrix} \begin{bmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{bmatrix}$$

que permite escalar, rotar, trasladar, o deformar un conjunto de puntos, dependiendo del valor de los elementos de T.

Funciones Matlab: **`maketform, tformfwd, tforminv`**

```
tform=maketform(transf_type, transf_parameters))

XY = tformfwd(WZ,tform)

WZ = tforminv(XY,tform)
```

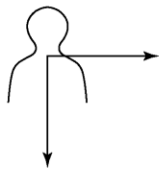
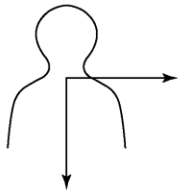
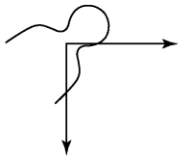
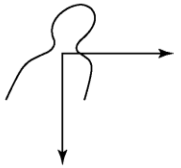
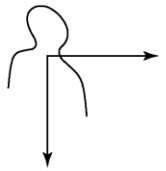
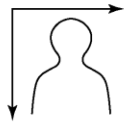
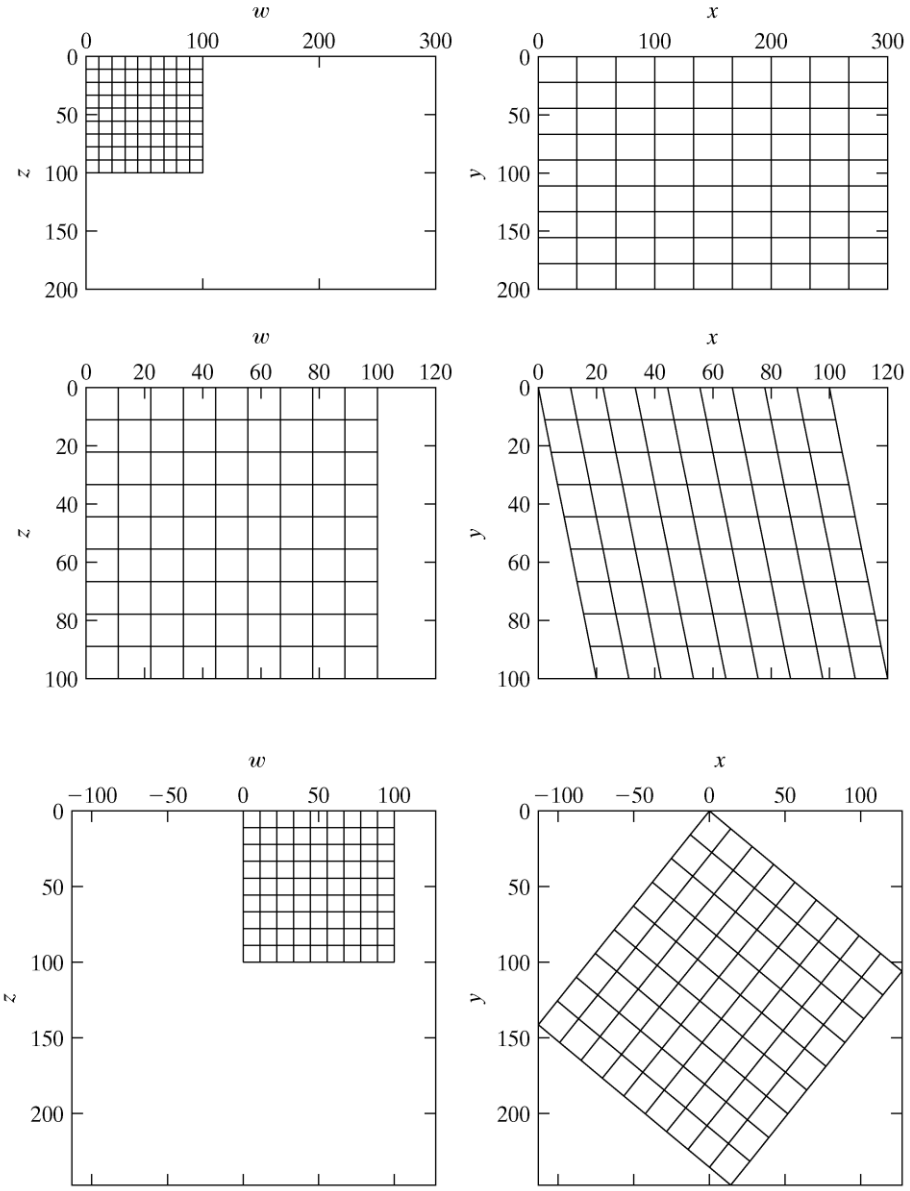
Type	Affine Matrix, T	Coordinate Equations	Diagram
Identity	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = w$ $y = z$	
Scaling	$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = s_x w$ $y = s_y z$	
Rotation	$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = w\cos\theta - z\sin\theta$ $y = w\sin\theta + z\cos\theta$	
Shear (horizontal)	$\begin{bmatrix} 1 & 0 & 0 \\ \alpha & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = w + \alpha z$ $y = z$	
Shear (vertical)	$\begin{bmatrix} 1 & \beta & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = w$ $y = \beta w + z$	
Translation	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \delta_x & \delta_y & 1 \end{bmatrix}$	$x = w + \delta_x$ $y = z + \delta_y$	

TABLE 5.3
Types of affine transformations.

La función `vistformfwd` permite visualizar el efecto de una transformación particular sobre un conjunto de puntos ubicados en una grilla.

Ejemplo:

```
>> T1=[3 0 0;0 2 0;0 0 1];
>> tform1=maketform('affine',T1);
>> vistformfwd(tform1,[0 100],[0 100]);
>> T2=[1 0 0;0.2 1 0;0 0 1];
>> tform2=maketform('affine',T2);
>> vistformfwd(tform2,[0 100],[0 100]);
>> Tscale=[1.5 0 0;0 2 0;0 0 1];
>> Trotation=[cos(pi/4) sin(pi/4) 0;
             -sin(pi/4) cos(pi/4) 0;
                  0          0     1];
>> Tshear=[1 0 0;0.2 1 0;0 0 1];
>> T3=Tscale*Trotation*Tshear;
>> tform3=maketform('affine',T3);
>> vistformfwd(tform3,[0 100],[0 100]);
```



a b
c d
e f

FIGURE 5.13
Visualizing affine transformations using grids.
(a) Grid 1.
(b) Grid 1 transformed using `tform1`.
(c) Grid 2.
(d) Grid 2 transformed using `tform2`.
(e) Grid 3.
(f) Grid 3 transformed using `tform3`.

Ejemplo: Consideraremos una **transformación lineal conforme,** que corresponde a un escalado, rotación y traslación, y que tiene asociada la siguiente matriz de transformación afín:

$$T = \begin{bmatrix} s\cos(\theta) & s\sin(\theta) & 0 \\ -s\sin(\theta) & s\cos(\theta) & 0 \\ \delta_x & \delta_y & 1 \end{bmatrix}$$

```
>>f=checkerboard(50);
>>s=0.8;
>>theta=pi/6;
>>T=[s*cos(theta) s*sin(theta) 0;
    -s*sin(theta) s*cos(theta) 0;
          0             0      1];
Tform=maketform('affine',T);
g=imtransform(f,Tform);
```

interpolación bilineal

```
>>g2=imtransform(f,Tform,'nearest');
```

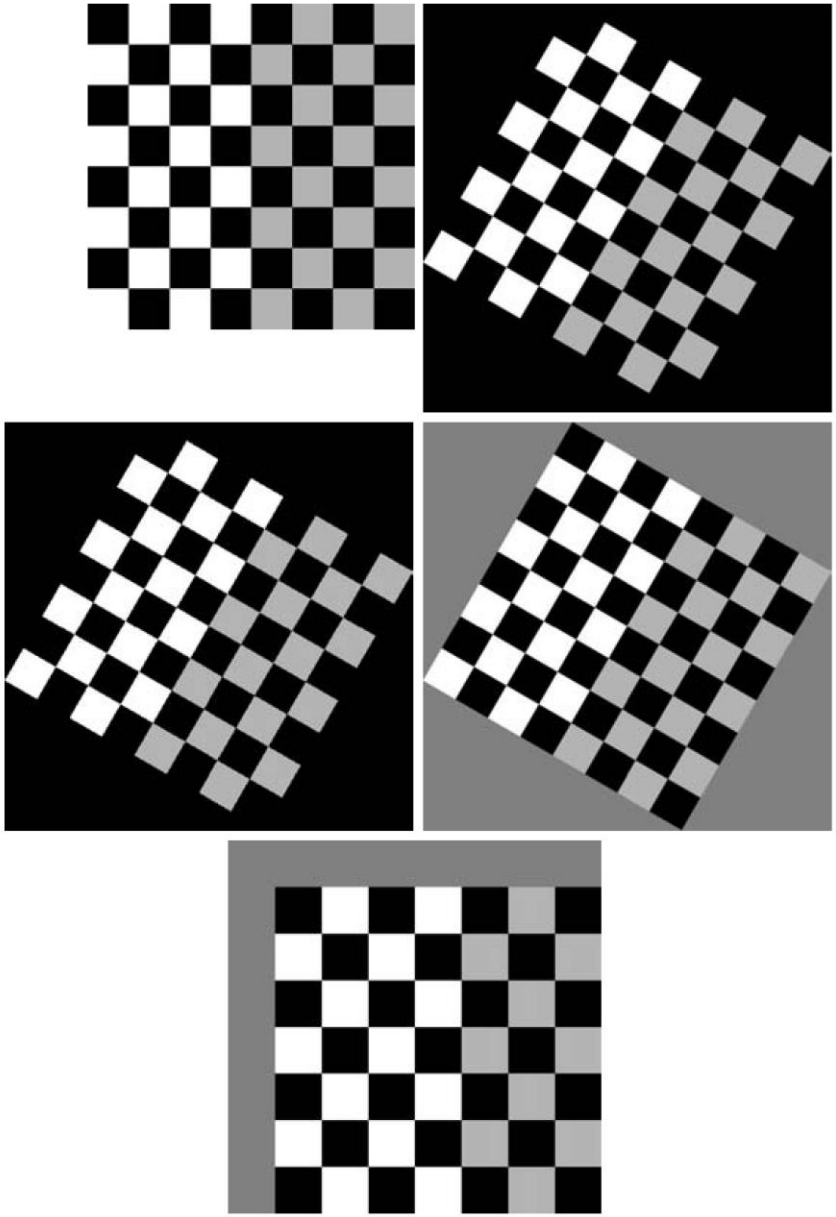
interpolación vecino más cercano

```
>>g3=imtransform(f,Tform,'FillValue',0.5);
```

especifica el nivel de intensidad de los pixeles fuera del dominio original de la imagen

Translación pura

```
>>T2=[1 0 0; 0 1 0; 50 50 1];
>>tform2=maketform('affine',T2);
>>g4=imtransform(f,tform2);
>>g5=imtransform(f,tform2,'Xdata',[1 400],'Ydata',...
     [1 400], 'FillValue',0.5);
```



a b
c d
e

FIGURE 5.14
Affine transformations of the checkerboard image.
(a) Original image. (b) Linear conformal transformation using the default interpolation (bilinear).
(c) Using nearest neighbor interpolation.
(d) Specifying an alternate fill value.
(e) Controlling the output space location so that translation is visible.

TABLE 5.4 Transformation types supported by `cp2tform` and `maketform`.

Transformation Type	Description	Functions
Affine	Combination of scaling, rotation, shearing, and translation. Straight lines remain straight and parallel lines remain parallel.	`maketform` `cp2tform`
Box	Independent scaling and translation along each dimension; a subset of affine.	`maketform`
Composite	A collection of spatial transformations that are applied sequentially.	`maketform`
Custom	User-defined spatial transform; user provides functions that define T and T^{-1}.	`maketform`
Linear conformal	Scaling (same in all dimensions), rotation, and translation; a subset of affine.	`cp2tform`
LWM	Local weighted mean; a locally-varying spatial transformation.	`cp2tform`
Piecewise linear	Locally varying spatial transformation.	`cp2tform`
Polynomial	Input spatial coordinates are a polynomial function of output spatial coordinates.	`cp2tform`
Projective	As with the affine transformation, straight lines remain straight, but parallel lines converge toward vanishing points.	`maketform` `cp2tform`