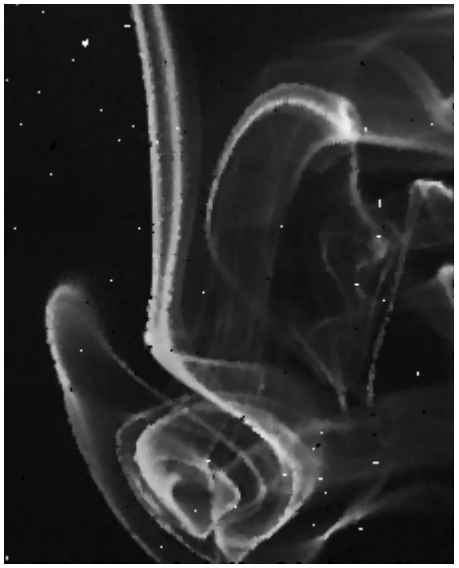
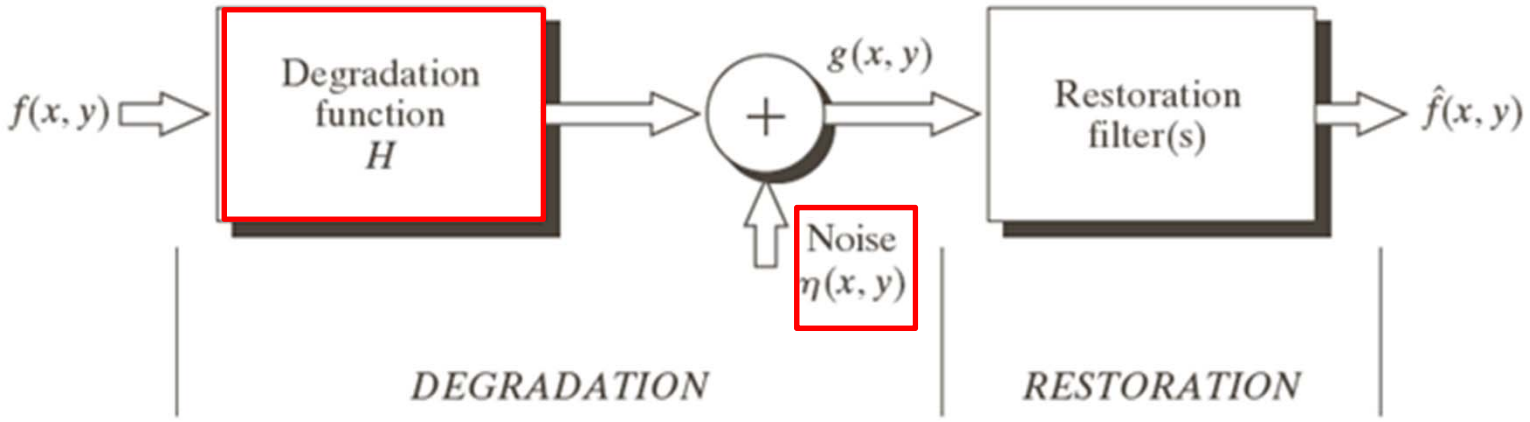


Chapter 8:

Image Restoration

Image Restoration

- **Definition**: A process that attempts to reconstruct or recover an image that has been degraded by using some *a priori* knowledge of the degradation phenomenon (e.g., deblurring).
- **Goal:** Modeling the degradation and applying the inverse process to recover the original image.
- More knowledge of *H* and η makes $\hat{f}(x, y)$ closer to *f(x,y).*



Noise Models

- Noise everywhere => video phone call using CCD camera: light levels, sensor temperature, wireless network channels

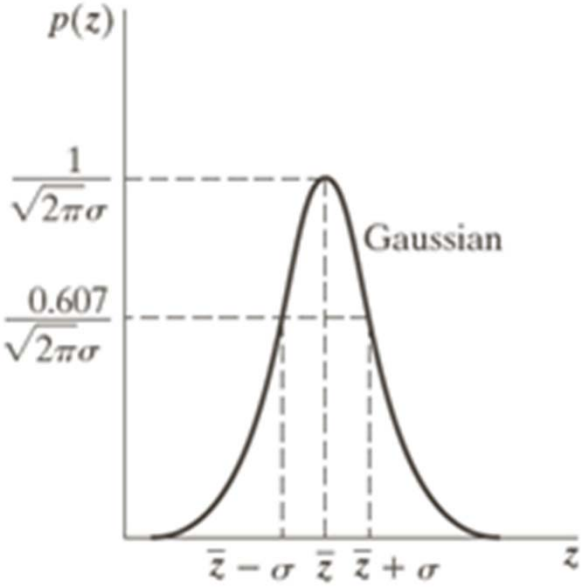
1. Gaussian(or normal) noise
   - most popular due to mathematical tractability and central limit theorem(중심극한정리)

$$p(z) = \frac{1}{\sqrt{2\pi\sigma}} e^{-(z-\bar{z})^2/2\sigma^2}$$

mean of z

variance

$$\sigma^2 = E[(z-\bar{z})^2]$$



~70% will be in $[(\bar{z}-\sigma),(\bar{z}+\sigma)]$

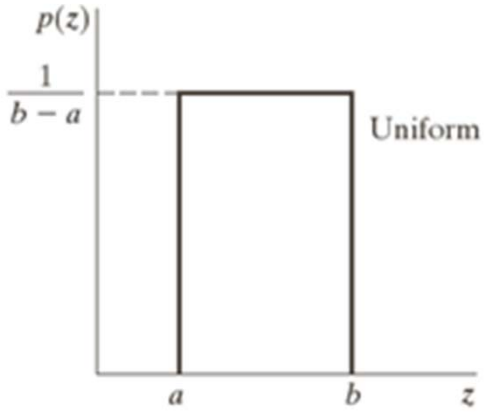
~95% will be in $[(\bar{z}-2\sigma),(\bar{z}+2\sigma)]$

Noise Models

2. Uniform noise

$$p(z)=\begin{cases}\dfrac{1}{b-a} & \text{if } a\le z\le b\\ 0 & \text{otherwise}\end{cases}$$

$$\bar{z}=\frac{a+b}{2},\quad \sigma^2=\frac{(b-a)^2}{12}$$

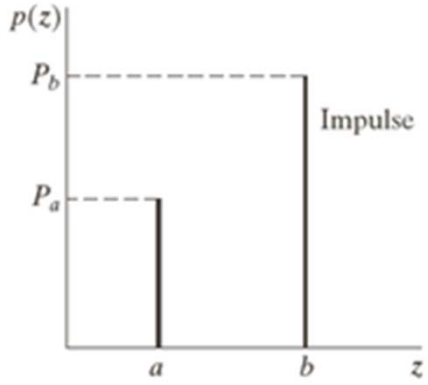


3. Salt-and-pepper noise

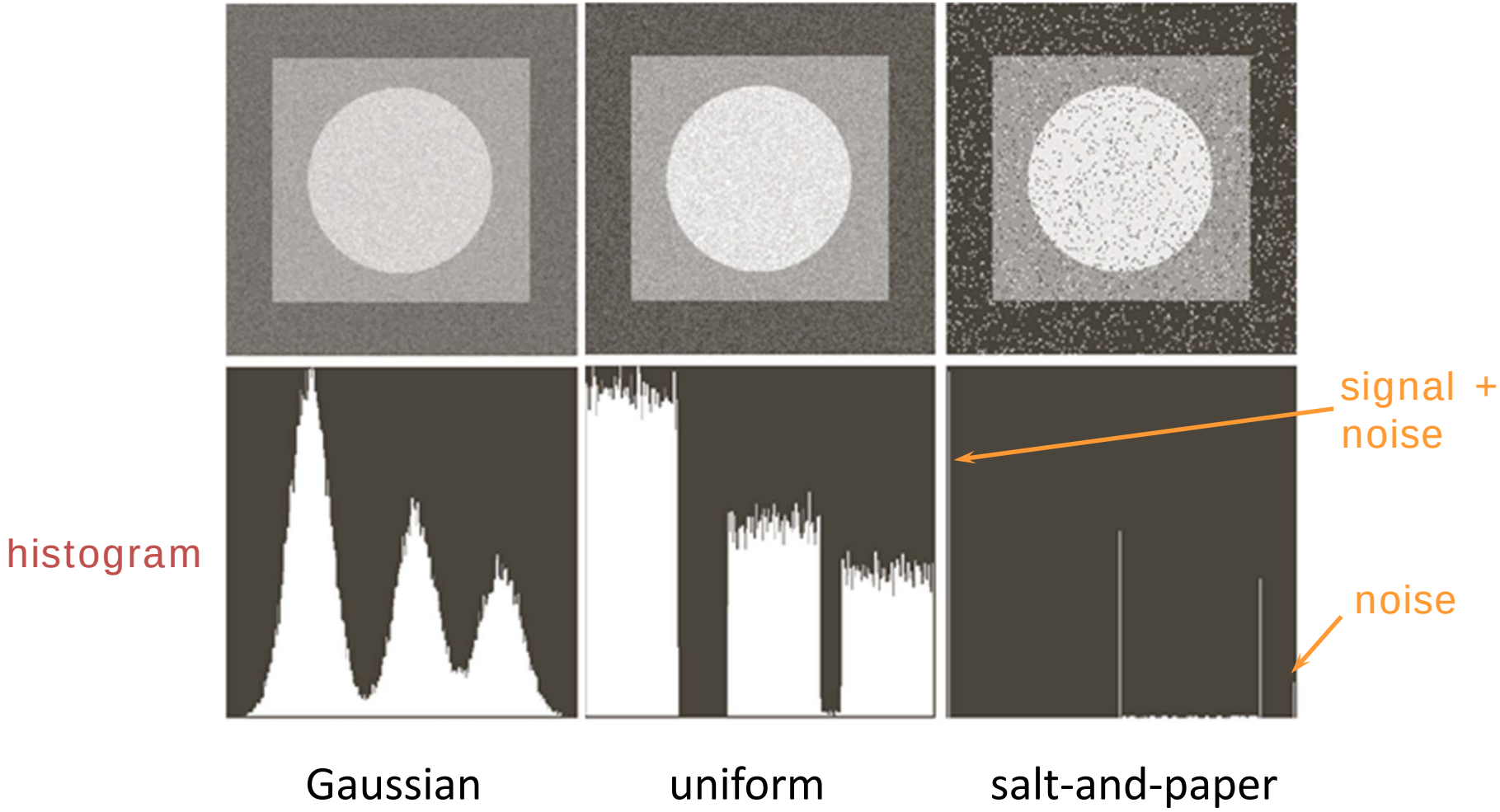
$$p(z)=\begin{cases}p_a & \text{for } z=a\\ p_b & \text{for } z=b\\ 0 & \text{otherwise}\end{cases}$$

= impulse noise

If b>a, b is light dots and a is dark dots.



Noise Models



Estimation of Noise Models

1. Capture a set of images from "flat" environments.
2. Calculate mean and variance.

$$\bar{z} = \sum_{i=0}^{L-1} z_i p_s(z_i) \qquad \sigma^2 = \sum_{i=0}^{L-1} (z_i - \bar{z})^2 p_s(z_i)$$

S: strip area (red box)

$p_s(z_i)$: probability density function(PDF) of pixel value z_i in S

$i = 0, 1, 2, \ldots, L\text{-}1$

- Salt-and-pepper noise: mid-gray area with histogram to get the parameters a and b

Salt and Pepper Noise in Matlab

- Also called **impulse noise**, shot noise, or binary noise, salt and pepper degradation can be caused by sharp, sudden disturbances in the image signal.
- Its appearance is randomly scattered white or black (or both) pixels over the image.

```
>> tw=imread('twins.tif');
>> t=rgb2gray(tw);
```

```
>> t_sp=imnoise(t,'salt & pepper');
```

- 3rd parameter with the range between 0 and 1 indicates the fraction of pixels to be corrupted.

```
>> imnoise(t, 'salt  pepper', 0.2); % 20 percent corruption
```

Salt & Papper Noise Example

```
>> imshow(t)
```

(a)

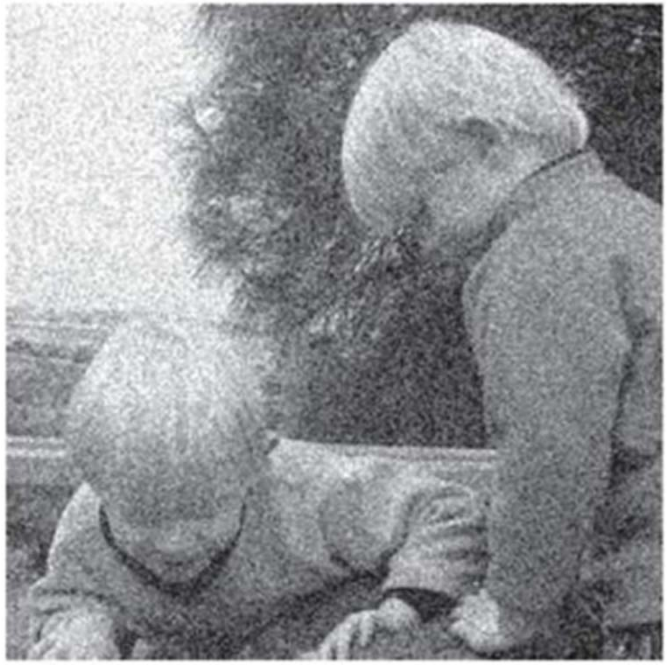
```
>> figure, imshow(t_sp)
```

(b)

FIGURE 8.1 *Noise on an image. (a) Original image. (b) With added salt and pepper noise.*

Gaussian Noise in Matlab

- **Gaussian noise** is an idealized form of **white noise,** which is caused by random fluctuations in the signal



- If the image is represented as I, and the Gaussian noise by N, then we can model a noisy image by simply adding the two => $I + N$

```
>> t_ga=imnoise(t,'gaussian');
```

Periodic Noise in Matlab

```
>> s=size(t);
>> [x,y]=meshgrid(1:s(1),1:s(2));
>> p=sin(x/3+y/5)+1;
>> t_pn=(im2double(t)+p/2)/2;
```

- Requires frequency domain filtering

Restoration from Salt and Pepper Noise: Lowpass Filter

```
>> a3=fspecial('average');
>> t_sp_a3=filter2(a3,t_sp);
```

```
>> a7=fspecial('average',[7,7]);
>> t_sp_a7=filter2(a7,t_sp);
```

(a)

(b)

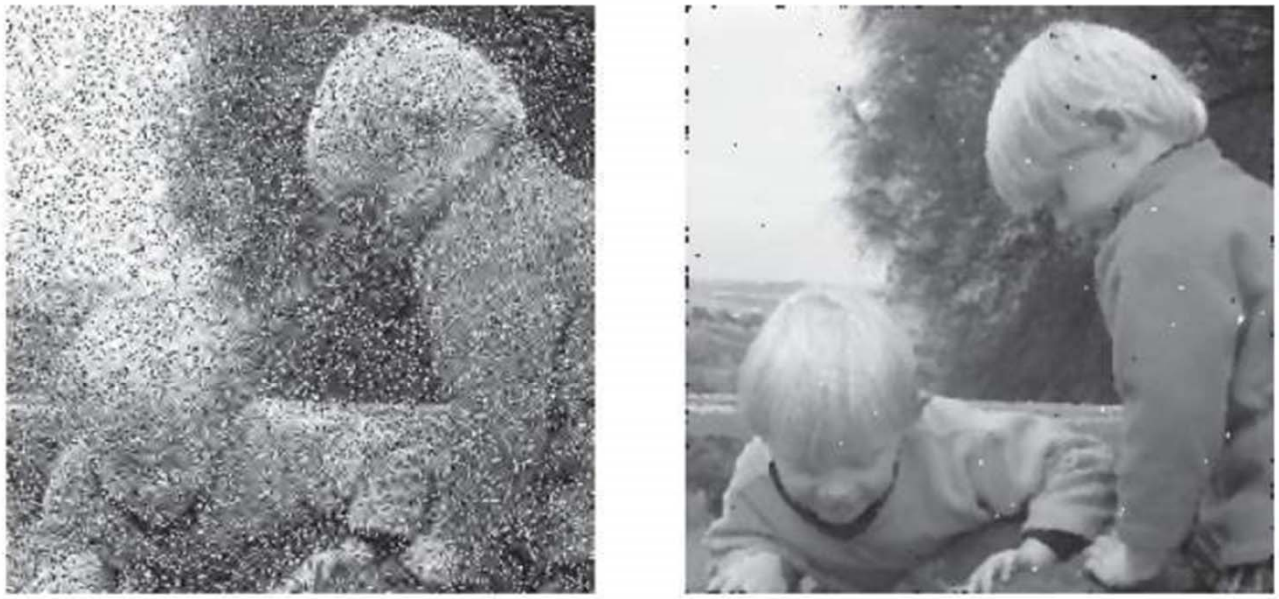
FIGURE 8.4 *Attempting to clean salt and pepper noise with average filtering. (a) 3 × 3 averaging. (b) 7 × 7 averaging.*

Restoration from Salt & Pepper Noise
Median Filter

50	65	52
63	255	58
61	60	57

$\longrightarrow$ 50 52 57 58 **60** 61 63 65 255 $\longrightarrow$ 60

```
>> t_sp2=imnoise(t,'salt & pepper',0.2);
```



(a) (b)

FIGURE 8.6 *Using a 3 × 3 median filter on more noise. (a) 20% salt and pepper noise. (b) After median fitering.*

Restoration from Salt & Pepper Noise
Median Filter

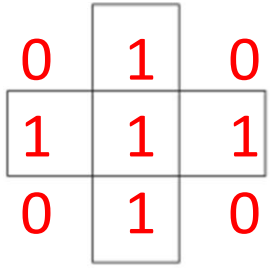
```
>> t_sp2_m5=medfilt2(t_sp2,[5,5]);
```

(a) (b)

FIGURE 8.7 *Cleaning 20% salt and pepper noise with median filtering. (a) Using* `medfilt2` *twice. (b) Using a* 5×5 *median filter.*

Restoration from Salt & Pepper Noise: Rank-Order Filtering

- Median filtering is a special case of a more general process called **rank-order filtering.**
- Set shape of mask(domain), order the values in the domain, and take the nth value.
- Median filter of 5x5 mask: RO filter with n=13.
- 3rd parameter gives the area mask to be considered for ordering. : Domain




```
>> ordfilt2(t_sp,3,[0 1 0;1 1 1;0 1 0]);
```

cross area mask

Restoration from Salt & Pepper Noise: Outlier Method

- Applying the median filter can in general be a slow operation: each pixel requires the sorting of at least nine values.
- Removes S&P noise by treating noisy pixels as outliers: Pixels whose gray values are significantly different from their neighbors will be removed.
- **Procedures**
  - ✓ Choose a threshold value D.
  - ✓ For a given pixel, compare its value p with the mean m of the values of its eight neighbors.
  - ✓ If $|p - m| > D$, then classify the pixel p as noise, otherwise not.
  - ✓ If the pixel is noisy, replace its value with m; otherwise leave its value unchanged.

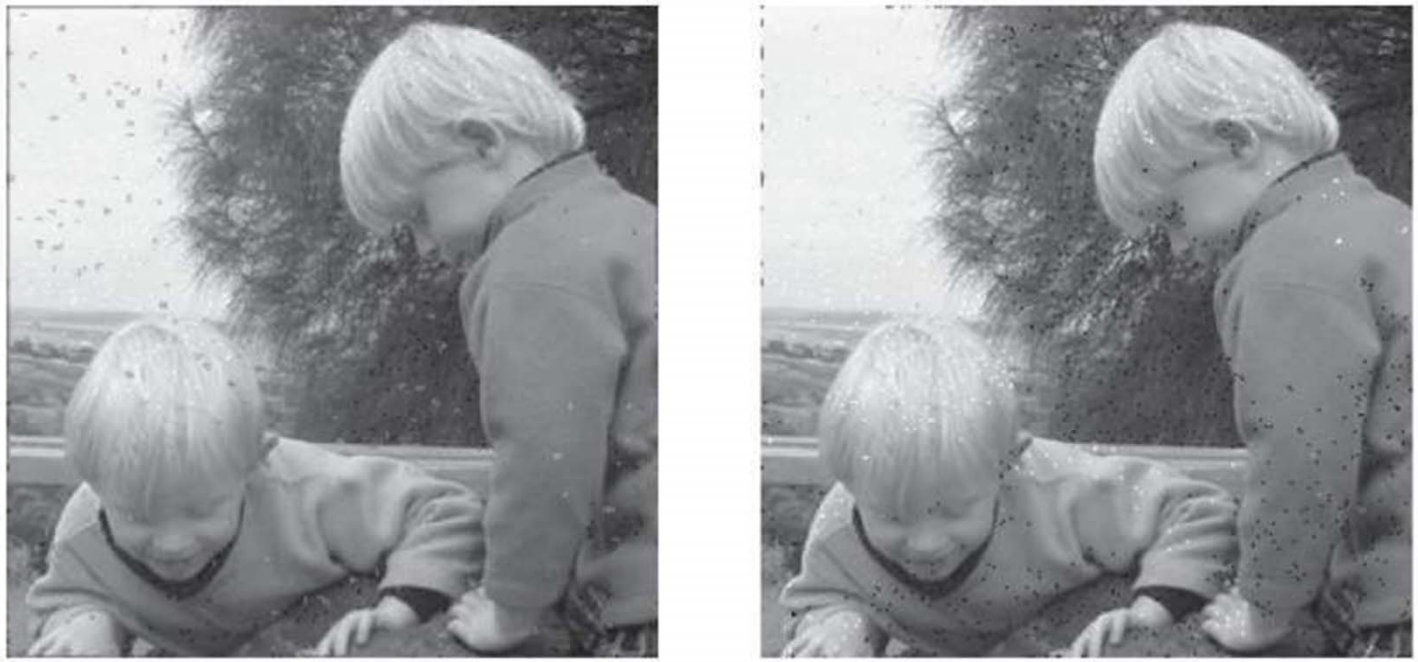
Outlier Function in Matlab

```
function res=outlier(im,d)
% OUTLIER(IMAGE,D) removes salt and pepper noise using an outlier method.
% This is done by using the following algorithm:
%
% For each pixel in the image, if the difference between its gray value
% and the average of its eight neighbors is greater than D, it is
% classified as noisy, and its grey value is changed to that of the
% average of its neighbors.
%
% IMAGE can be of type UINT8 or DOUBLE; the output is of type
% UINT8.  The threshold value D must be chosen to be between 0 and 1.

f=[0.125 0.125 0.125; 0.125 0 0.125; 0.125 0.125 0.125];
imd=im2double(im);
imf=filter2(f,imd);
r=abs(imd-imf)-d>0;
res=im2uint8(r.*imf+(1-r).*imd);
```

FIGURE 8.8 *A MATLAB function for cleaning salt and pepper noise using an outlier method.*

Outlier Method



(a) (b)

FIGURE 8.9 *Applying the outlier method to 10% salt and pepper noise. (a) $D = 0.2$. (b) $D = 0.4$.*

Restoration from Gaussian Noise N_i: Image Averaging

- suppose we have 100 copies of image, each with noise

$$M' = \frac{1}{100}\sum_{i=1}^{100}(M + N_i)$$

$$= \frac{1}{100}\sum_{i=1}^{100} M + \frac{1}{100}\sum_{i=1}^{100} N_i$$

$$= M + \frac{1}{100}\sum_{i=1}^{100} N_i.$$

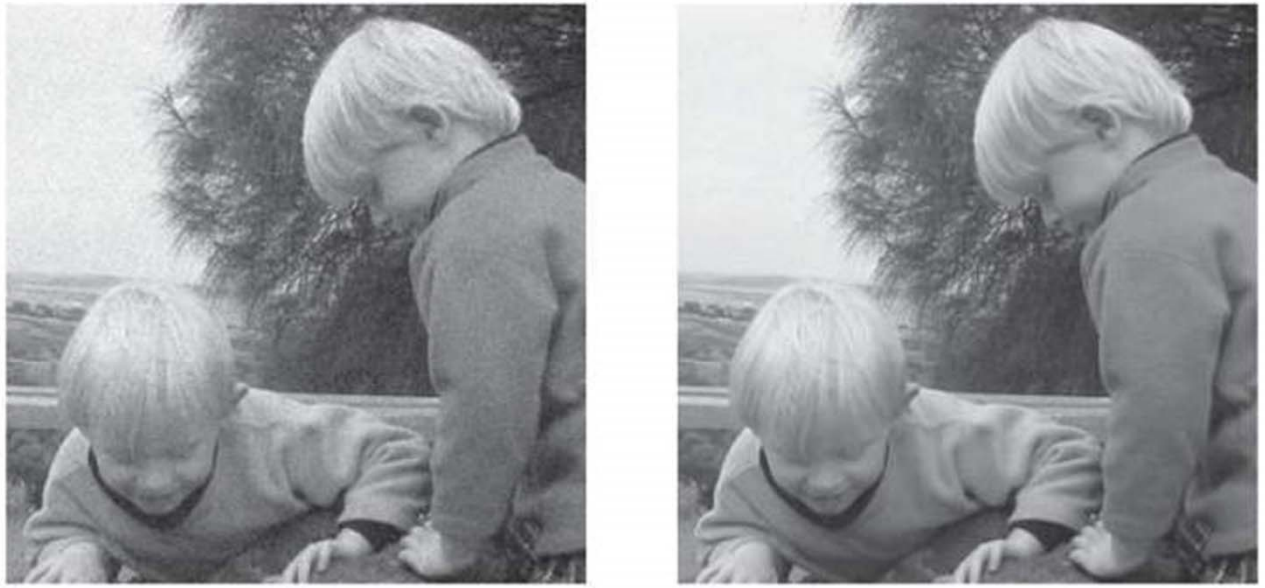
The greater the number of N_i's; the closer to zero

- Because N_i is normally distributed with mean 0, it can be readily shown that the mean of all the N_i's will be close to zero.

Image Averaging with 10 and 100 Images

```
>> s=size(t);
>> t_ga10=zeros(s(1),s(2),10);
>> for i=1:10 t_ga10(:,:,i)=imnoise(t,'gaussian'); end
```

```
>> t_ga10_av=mean(t_ga10,3); % mean value in z direction
```



(a) (b)

FIGURE 8.10 *Image averaging to remove Gaussian noise. (a) 10 images. (b) 100 images.*

Restoration from Gaussian Noise: Average Filter

```
>> a3=fspecial('average');
>> a5=fspecial('average',[5,5]);
>> tg3=filter2(a3,t_ga);
>> tg5=filter2(a5,t_ga);
```

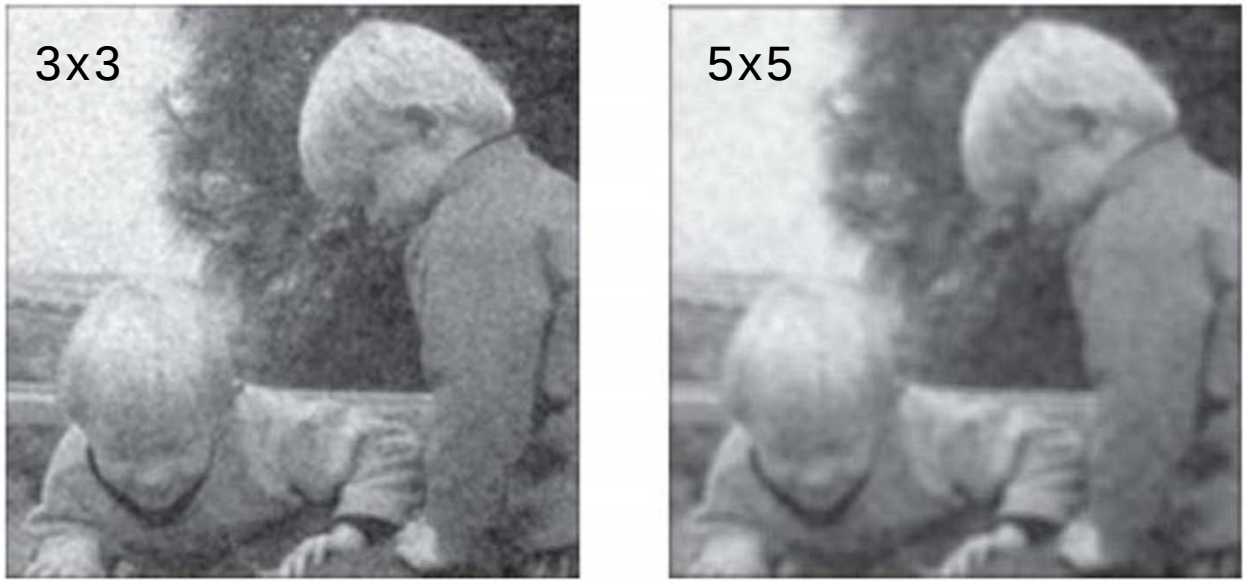


FIGURE 8.11 *Using averaging filtering to remove Gaussian noise. (a) 3 × 3 averaging. (b) 5 × 5 averaging.*

Removal of Gaussian Noise: Adaptive Filtering

- **Adaptive filters** are a class of filters that change their characteristics according to the values of the grayscales under the mask.
- Uses local statistical properties within the mask.

1. Minimum mean-square error filter

- *M'= M + N: assume that* the noise may not be normally distributed with mean 0.
- output value = $m_f + \frac{\sigma_f^2}{\sigma_f^2 + \sigma_g^2}(g - m_f)$

  mean in the mask

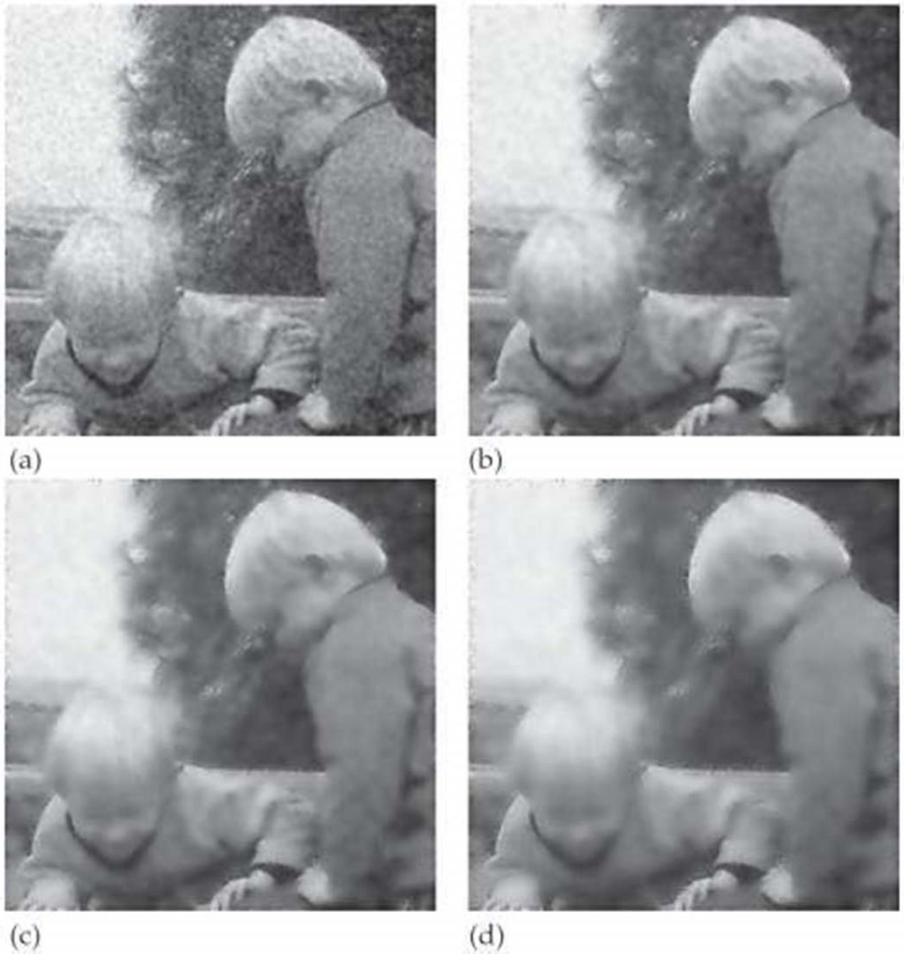
  current pixel value

  variance of noise over entire image

- If local variance σ_f^2 is high enough, output is close to g. -> If var in mask(σ_f^2) » var in image(σ_g^2), edge is preserved(g).
- If local variance σ_f^2 is low, output is close to m_f. -> If var in mask(σ_f^2) « var in image(σ_g^2), background is averaged(m_f).

Restoration from Gaussian Noise : Adaptive Filter

Gaussian noised added.



3x3 adaptive filtered image

5x5 adaptive filtered image

7x7 adaptive filtered image

- tends to blur edges and high freq. components
- better than general LPFs.

Restoration from Gaussian Noise: Adaptive Filtering

2. Wiener filters

- operates in a sense that it minimizes the square of the difference between input and output images.
- preferred to be used in frequency domain
- output value = $m_f + \boxed{\frac{\max\{0, \sigma_f^2 - n\}}{\max\{\sigma_f^2, n\}}}(g - m_f)$
  - noise variance **n** = mean of all σ_f^2 over the entire image

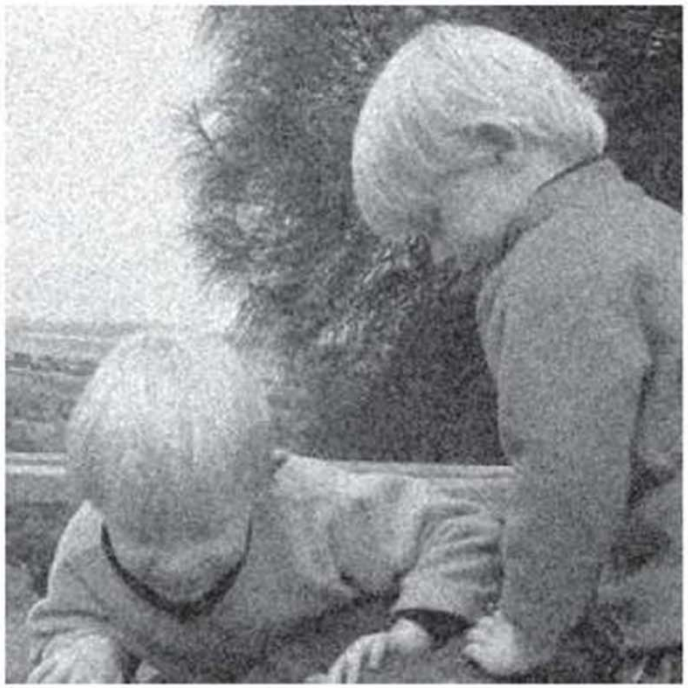
If var in mask(σ_f^2) » var in image(n), edge is preserved (g).
If var in mask(σ_f^2) « var in image(n), background is averaged(m_f).

```
>> t1=wiener2(t_ga);
>> t2=wiener2(t_ga,[5,5]);
>> t3=wiener2(t_ga,[7,7]);
>> t4=wiener2(t_ga,[9,9]);
```

Results are shown in the next slide.

Restoration from Gaussian Noise : Wiener Filter

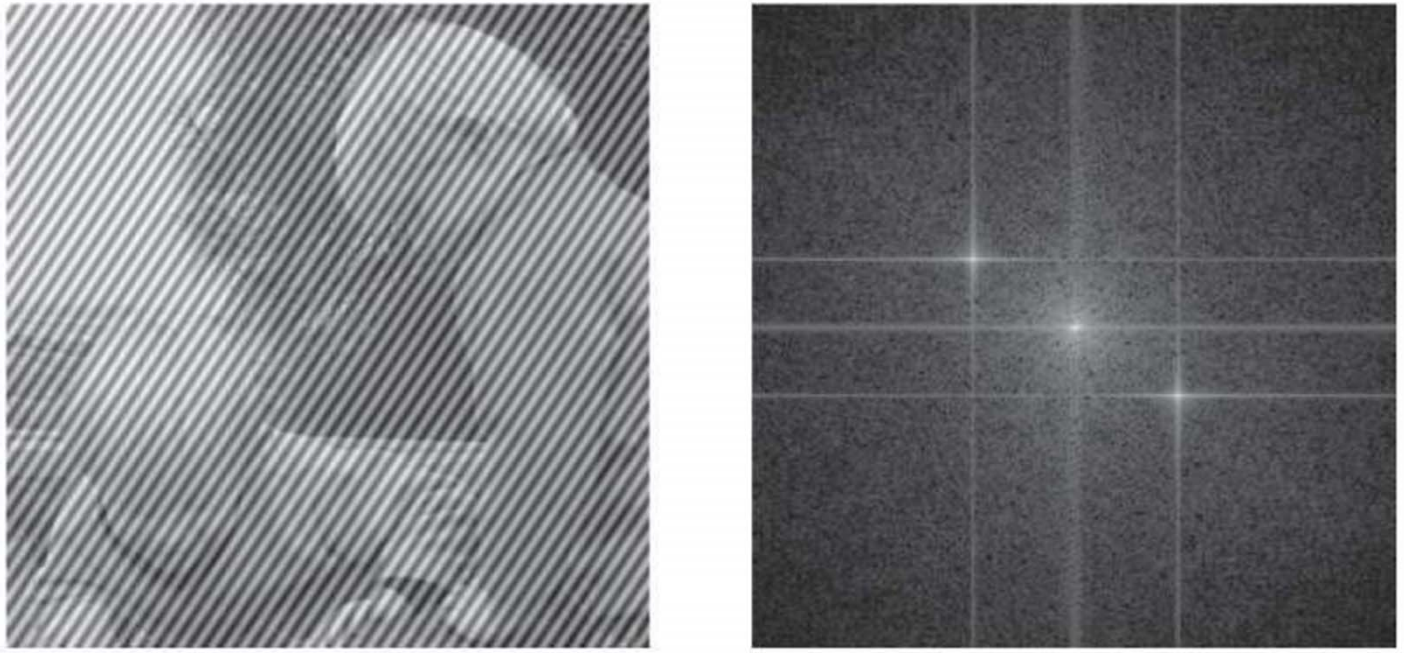
```
>> t2=imnoise(t,'gaussian',0,0.005);
>> imshow(t2)
>> t2w=wiener2(t2,[7,7]);
>> figure,imshow(t2w)
```



- Better for low variance images.
- Blurring background while preserving edges.

Restoration from Periodic Noise in Freq. Domain

```
>> [x,y]=meshgrid(1:256,1:256);
>> p=1+sin(x+y/1.5);
>> tp=(double(t)/128+p)/4;
>> tf=fftshift(fft2(tp));
```



(a) (b)

Quiz: How does the spectrum look like if there is a periodic noise with higher frequency?

Restoration from Periodic Noise: Band Reject Filter

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

- Used when the general location of the noise components in the frequency domain is approximately known.

1. Bandreject filter

a b c

FIGURE 5.15 From left to right, perspective plots of ideal, Butterworth (of order 1), and Gaussian bandreject filters.

Quiz: What do you call the plot above?

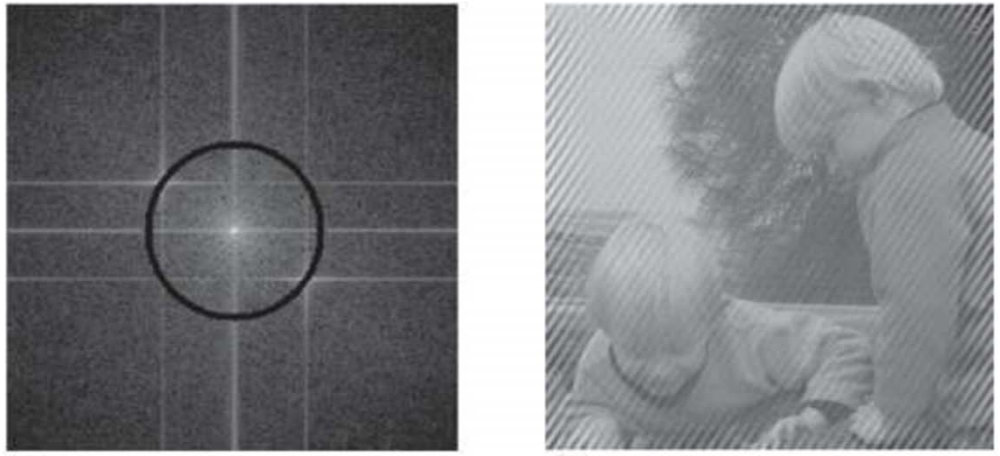
Quiz: What do those plots mean?

Restoration from Periodic Noise: Band Reject Filter

```
>> z=sqrt((x-129).^2+(y-129).^2);
```

```
>> br=(z < 47 | z > 51);
```

```
>> tbr=tf.*br;
```



(a) (b)

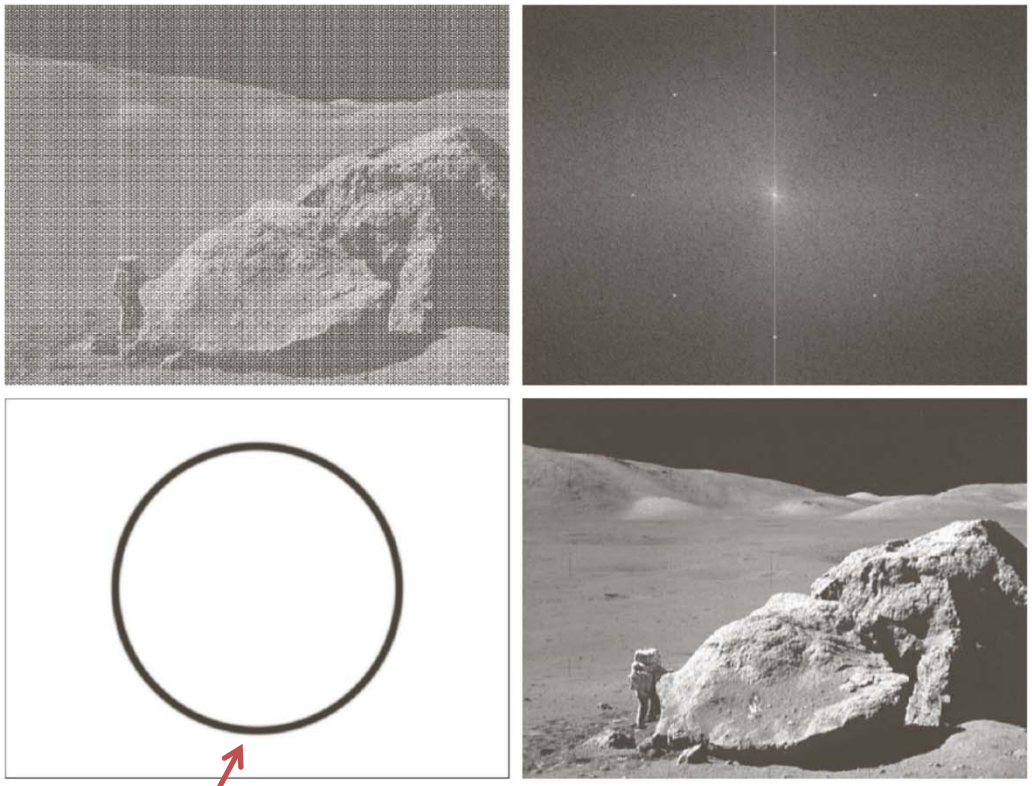
FIGURE 8.15 *Removing periodic noise with a band-reject filter. (a) A band-reject filter. (b) After inversion.*

Restoration from Periodic Noise

1. Bandreject filter (contd.)

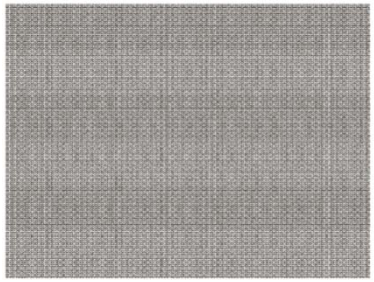
a b
c d

FIGURE 5.16
(a) Image corrupted by sinusoidal noise. (b) Spectrum of (a). (c) Butterworth bandreject filter (white represents 1). (d) Result of filtering. (Original image courtesy of NASA.)

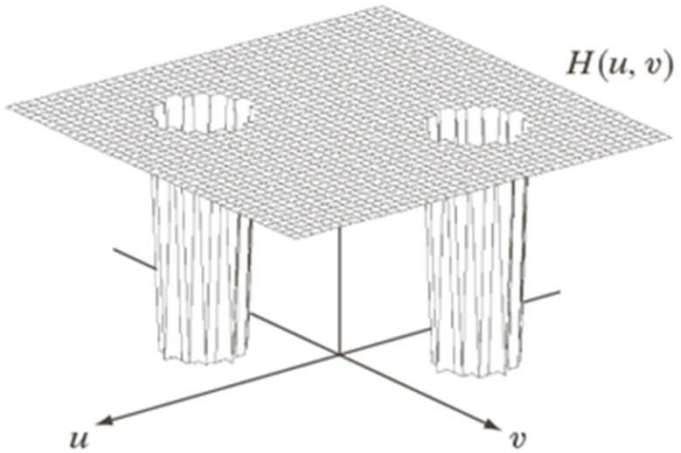


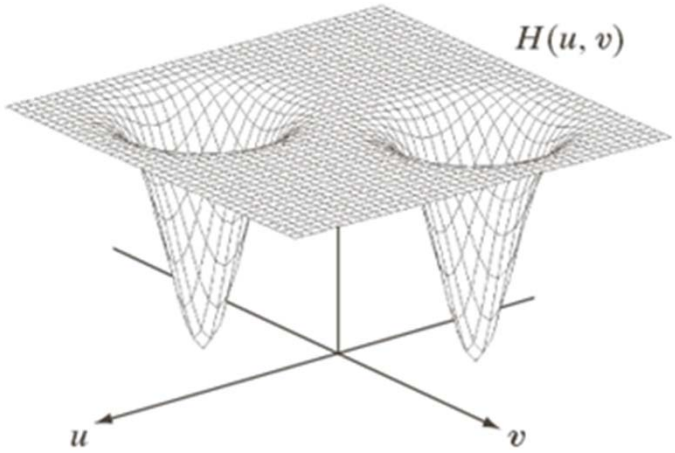
2. Bandpass filter

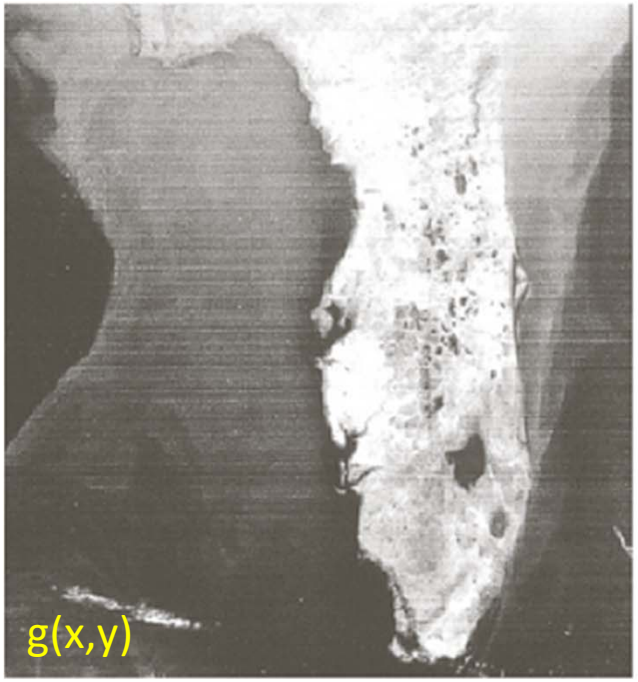
$$H_{BP}(u,v) = 1 - H_{BR}(u,v)$$

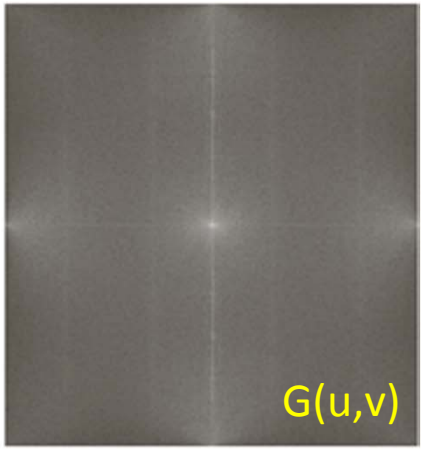


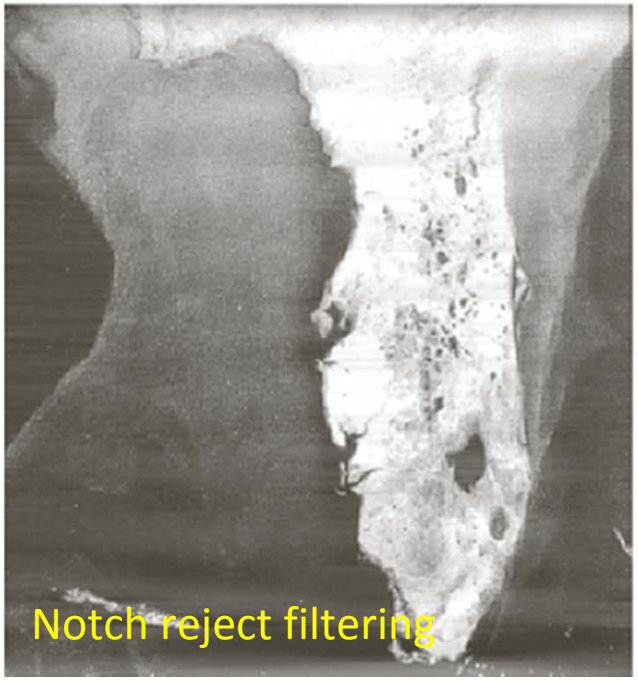
Restoration from Periodic Noise: Notch (reject) filter

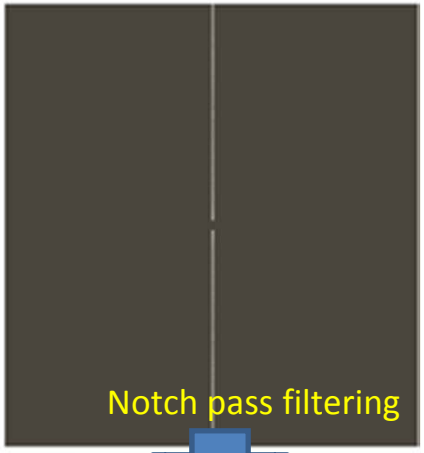


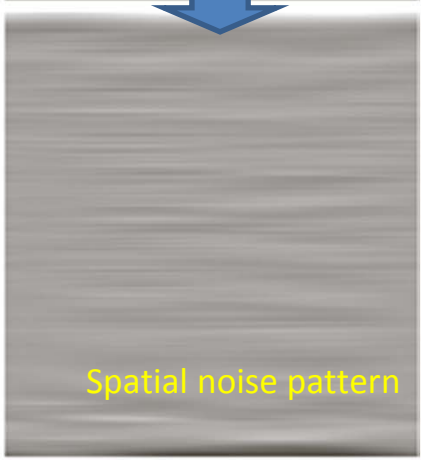






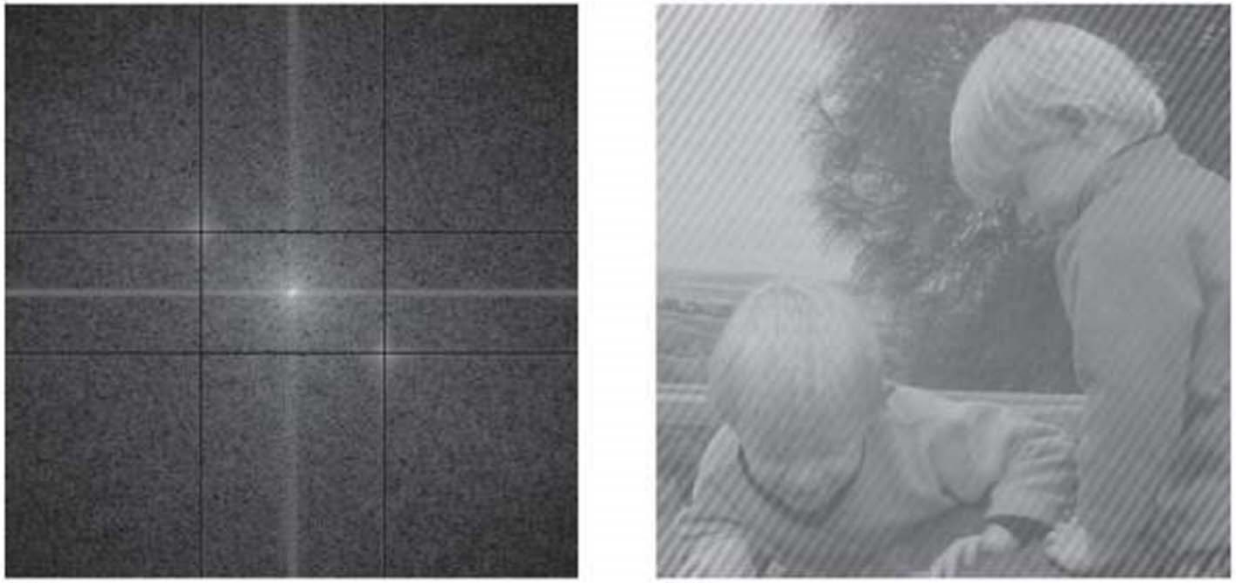






Restoration from Periodic Noise: Notch Filter

```
>> tf(156,:)=0;
>> tf(102,:)=0;
>> tf(:,170)=0;
>> tf(:,88)=0;
```



(a) (b)

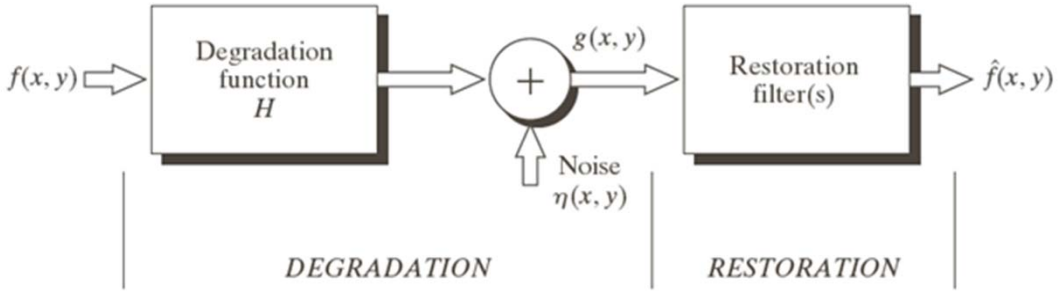
FIGURE 8.16 *Removing periodic noise with a notch filter. (a) A notch filter. (b) After inversion.*

Restoration of Image Corrupted by Linear Shift Invariant Model and Additive Noise

Linear Position-Invariant Degradation Model

- The degraded image $g(x,y)$ is obtained by passing an input image $f(x,y)$ through the degradation operator, H, together with an additive noise, $\eta(x,y)$

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



- Linearity
  - additivity: $H[f_1(x,y) + f_2(x,y)] = H[f_1(x,y)] + H[f_2(x,y)]$
  - homogeneity: $H[af_1(x,y)] = aH[f_1(x,y)]$
- Position-invariant (=shift-invariant):

  When $g(x,y) = H[f(x,y)]$, $H[f(x-\alpha, y-\beta)] = g(x-\alpha, y-\beta)$.

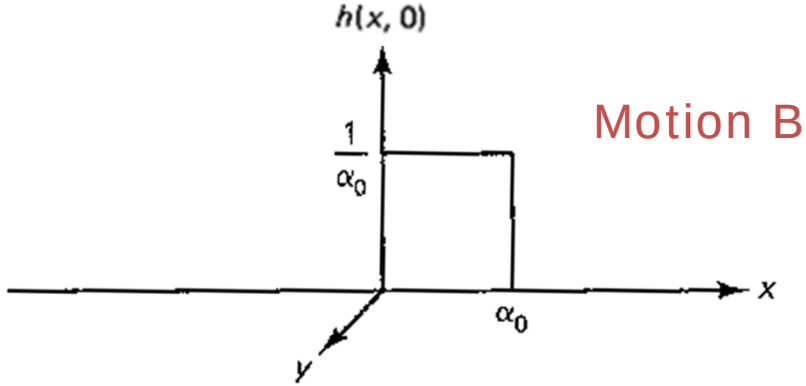
Linear Position-Invariant Degradation Model

- Normally H is assumed to be a linear position invariant (LPI) system, which can then be expressed as:

$$y(x,y) = x(x,y) * h(x,y) + \eta(x,y)$$
$$= \sum_{\alpha=0}^{M-1} \sum_{\beta=0}^{N-1} x(\alpha,\beta) h(x-\alpha, y-\beta) + \eta(x,y)$$

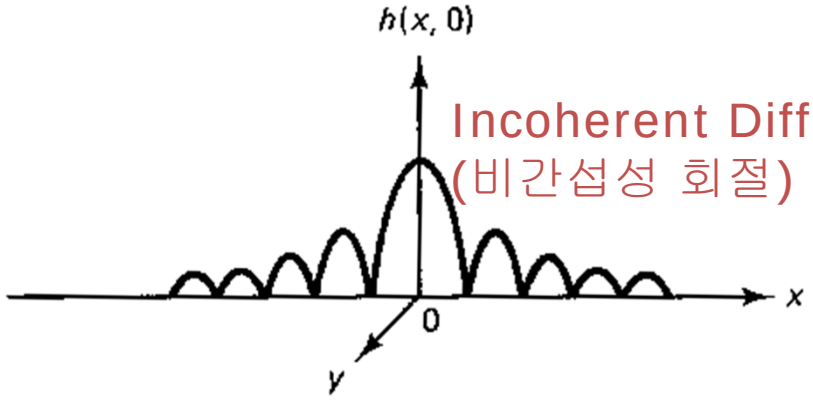
$$g(x,y) = H[f(x,y)] + \eta(x,y)$$
$$G(u,v) = H(u,v)F(u,v) + N(u,v)$$

Typical LPI Degradation Model : Point Spread Functions (PSFs)



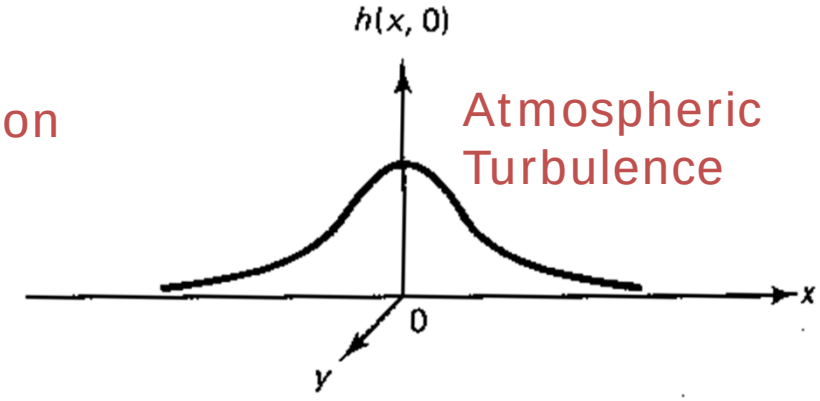
Motion Blur

(a) One dimensional motion blur



Incoherent Diffraction
(비간섭성 회절)

(b) Incoherent diffraction limited system (lens cutoff)



Atmospheric Turbulence

(c) Average atmospheric turbulence

Typical LPI Degradation Model:

Atmospheric turbulence

$$H(u,v) = e^{-k(u^2+v^2)^{5/6}}$$

strength of turbulence by nature

severe turbulence k = 0.0025

Typical LPI Degradation Model : Motion blurring

- 2D model in optical process while opening diaphragm

$$g(x,y) = \int_0^T f[x - x_0(t),\ y - y_0(t)]\,dt$$

blurred image

duration of exposure

$$G(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g(x,y)e^{-j2\pi(ux+uy)}dxdy$$

$$F[f(x-x_0(t))] \Leftrightarrow e^{-j2\pi u x_0(t)}F(u)$$

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}[\int_0^T f[x-x_0(t), y-y_0(t)]dt]e^{-j2\pi(ux+uy)}dxdy$$

$$= F(u,v)\int_0^T e^{-j2\pi(ux_0(t)+uy_0(t))}dt$$

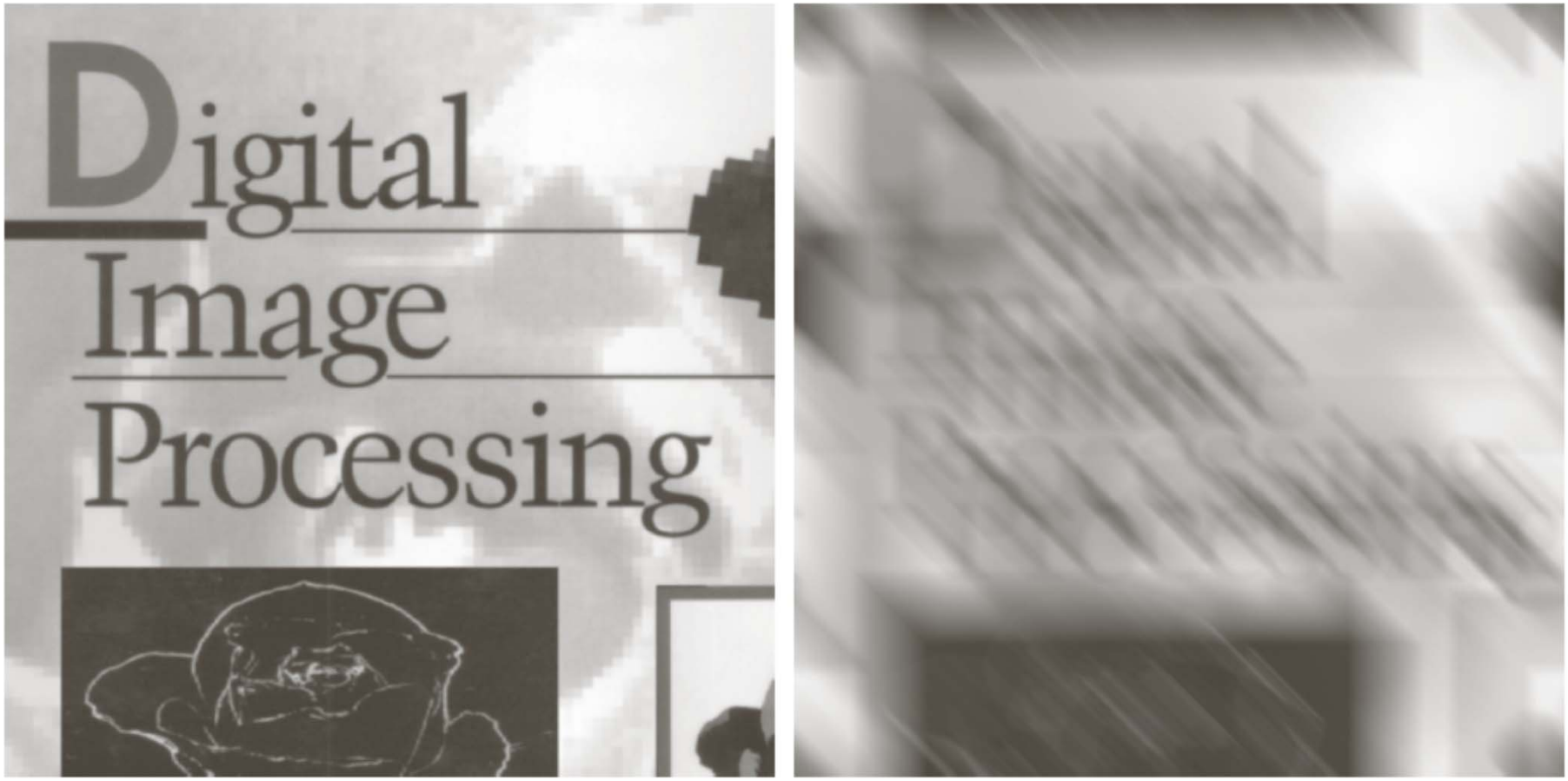
H(u,v)

$$H(u,v) = \frac{T}{\pi(ua+vb)}\sin[\pi(ua+vb)]e^{-j\pi(ua+vb)}$$

$$x_0(t) = at/T$$

$$y_0(t) = bt/T$$

Image Blurred by Uniform Motion



a=b=0.1, T=1

Restoration of Blurred Image: Inverse Filtering

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

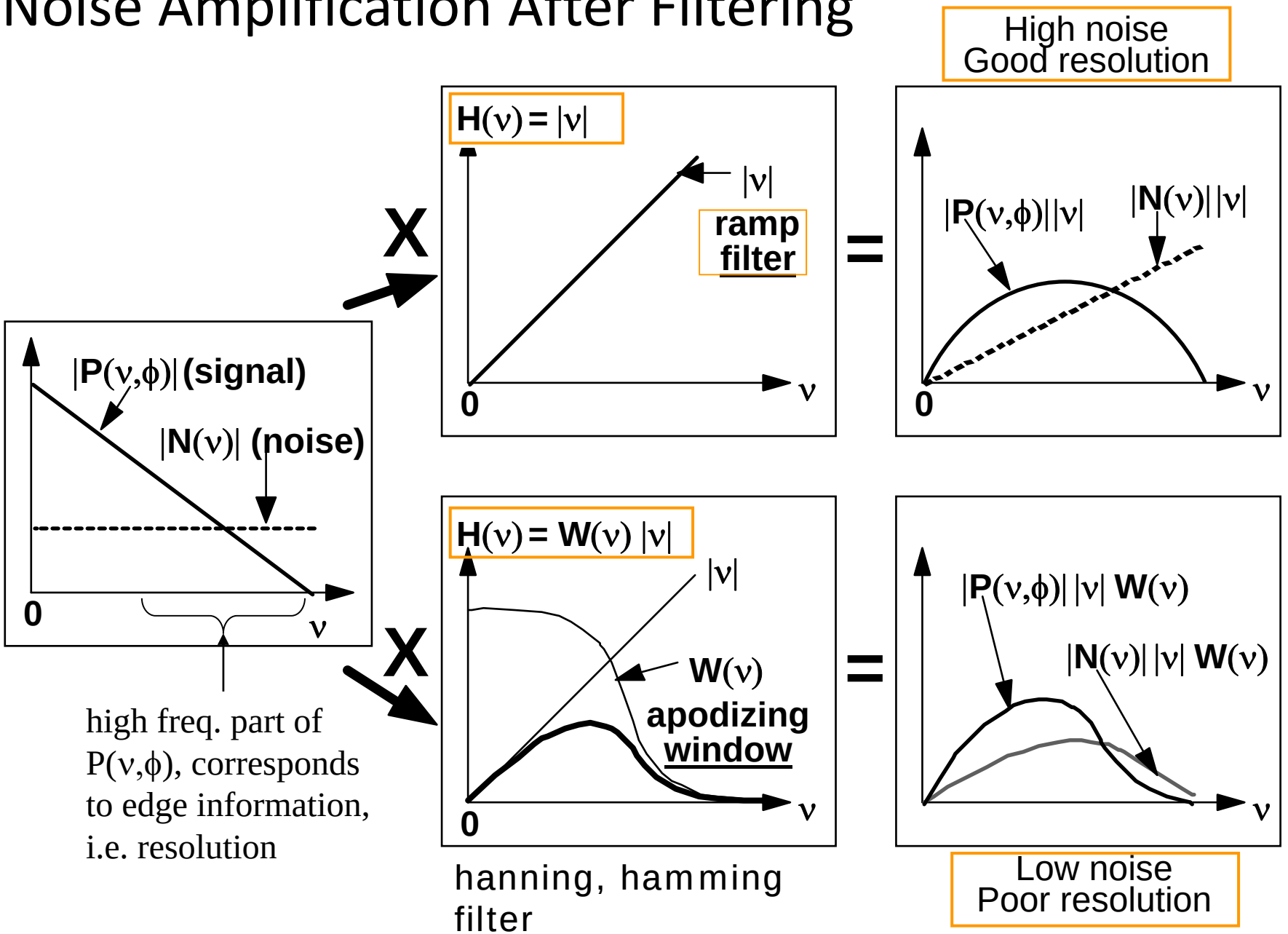
$$G(u,v) = H(u,v)F(u,v) + N(u,v)$$

- Inverse Filtering:

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

- Even if we know the degradation function we cannot recover the orginal image exactly because N(u,v) is usually unknown.
- When H(u,v) is zero or very small, the error N(u,v)/H(u,v) could easily dominate the estimate F_hat(u,v).

Noise Amplification After Filtering



Simple Attempt of Inverse Filtering

$$Y(i,j) = X(i,j)F(i,j) \Rightarrow X(i,j) = \frac{Y(i,j)}{F(i,j)}$$

```
>> w=imread('wombats.tif');
>> wf=fftshift(fft2(w));
>> b=lbutter(w,15,2);
>> wb=wf.*b;
>> wba=abs(ifft2(wb));
>> wba=uint8(255*mat2gray(wba));
>> imshow(wba)
```

```
>> w1=fftshift(fft2(wba))./b;
>> w1a=abs(ifft2(w1));
>> imshow(mat2gray(w1a))
```

Dividing by very small values of filter coefficients amplified noise into very large values!!

Solution 1 for Inverse Filtering

Appling LPF L(i,j) after inverse filtering.

$$X(i,j) = \frac{Y(i,j)}{F(i,j)} L(i,j)$$

```
>> wbf=fftshift(fft2(wba));
>> w1=(wbf./b).*lbutter(w,40,10);
>> w1a=abs(ifft2(w1));
>> imshow(mat2gray(w1a))
```

cutoff radius: 60

(a)

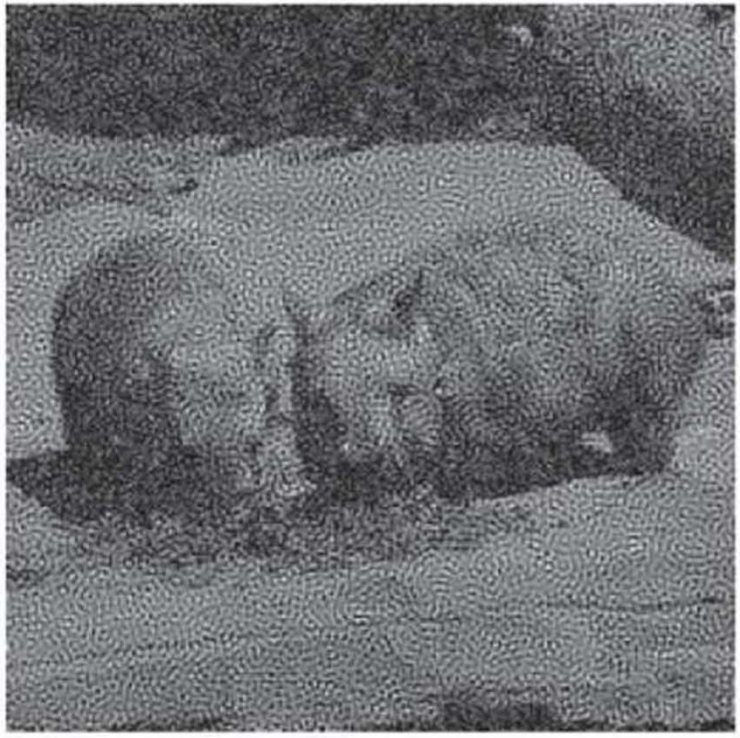
(b)

Solution 1 for Inverse Filtering

cutoff radius: 80

cutoff radius: 100

(c)



(d)

FIGURE 8.18 *Inverse filtering using low-pass filtering to eliminate zeros.*

Solution 2 for Inverse Filtering

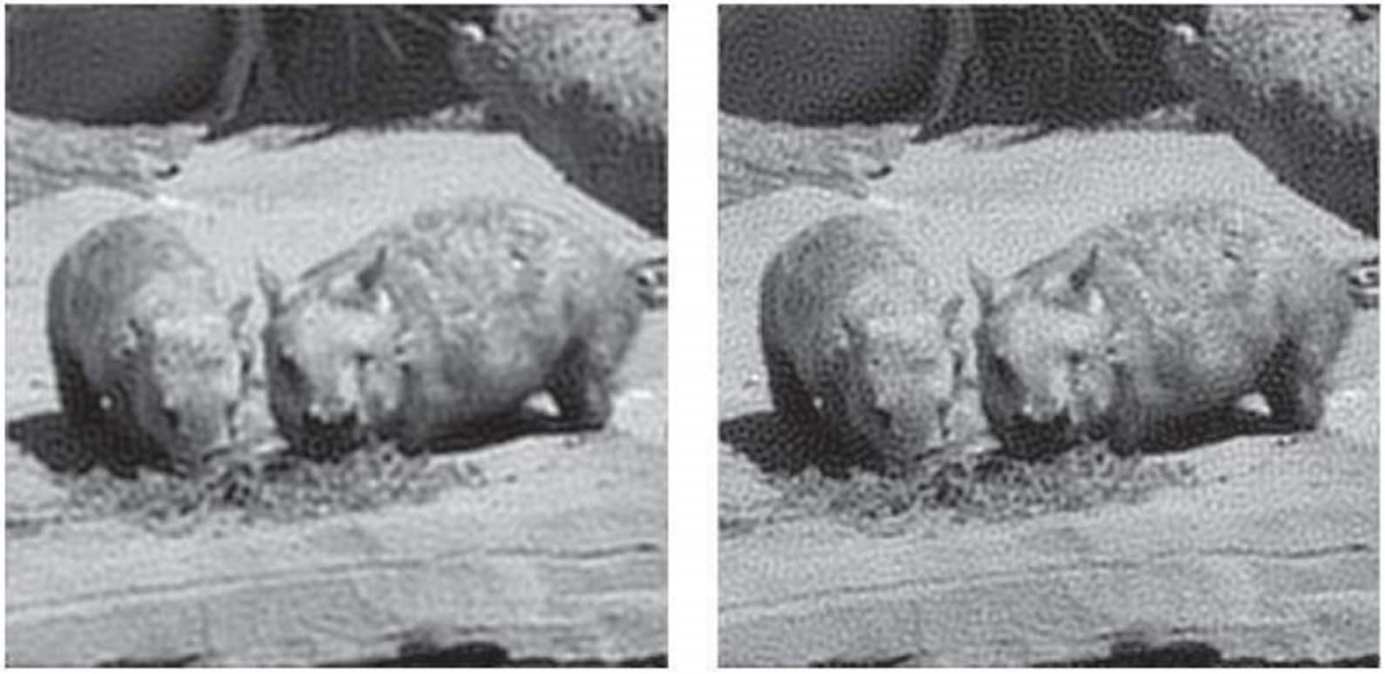
Use constraint for division.

=> Pseudo inverse filter

$$X(i,j) = \begin{cases} \dfrac{Y(i,j)}{F(i,j)} & if \ |F(i,j)| \geq d, \\ Y(i,j) & if \ |F(i,j)| < d. \end{cases}$$

```
>> d=0.01;
>> b=1butter(w,15,2);b(find(b<d))=1;
>> w1=fftshift(fft2(wba))./b;
>> w1a=abs(ifft2(w1));
>> imshow(mat2gray(w1a))
```

d = 0.005

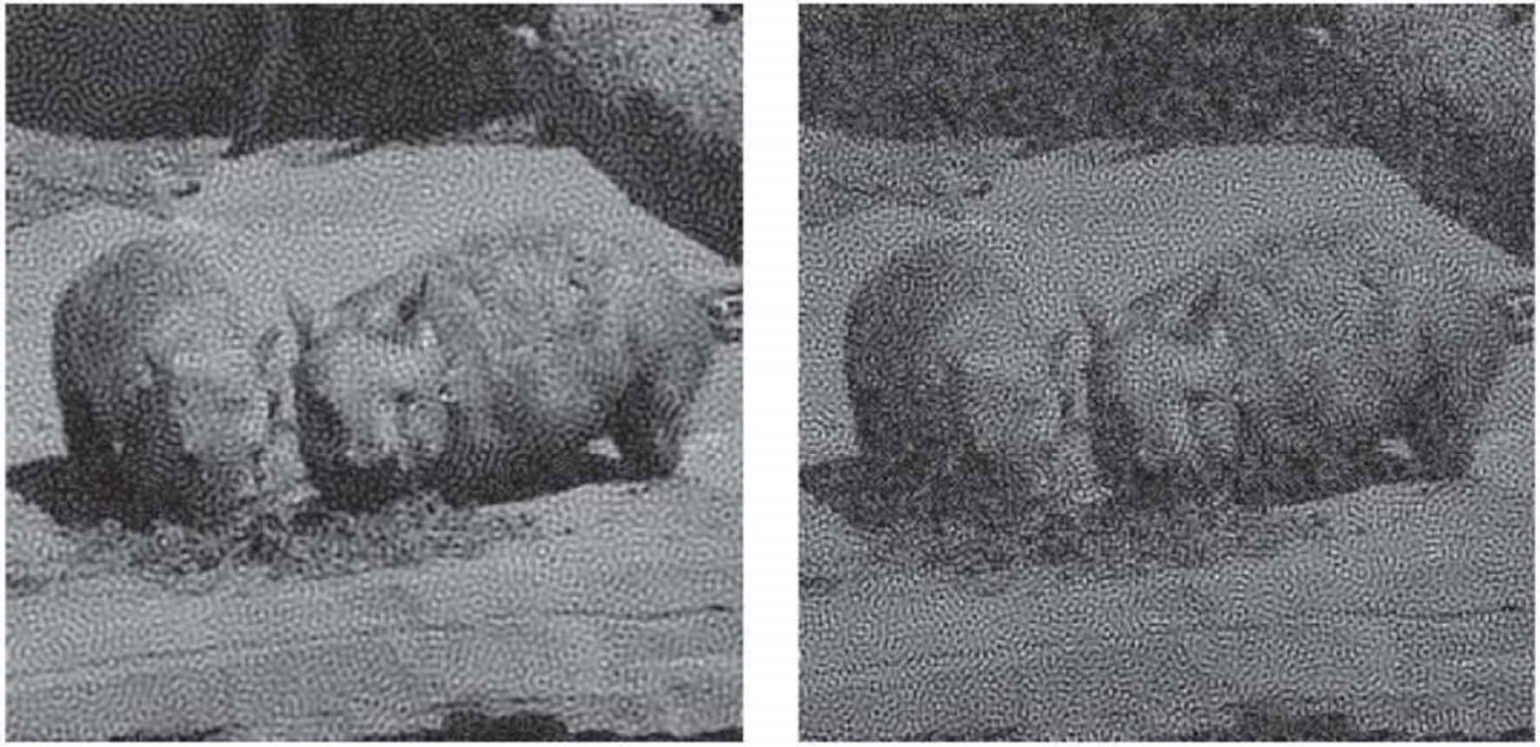


(a) (b)

Solution 2 for Inverse Filtering

d = 0.002

d = 0.001



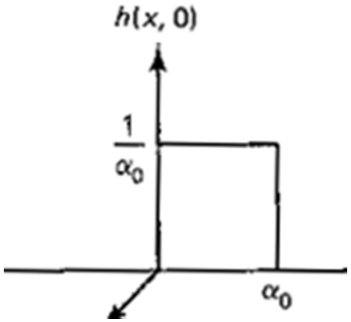
(c)

(d)

FIGURE 8.19 *Inverse filtering using constrained division.*

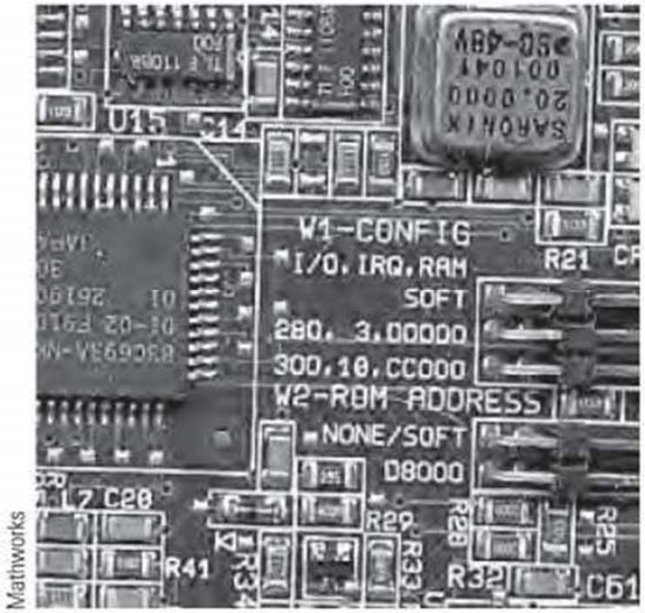
Generation of Motion Blur in Matlab

```
>> fspecial('motion', 7, 0)
0.1429    0.1429    0.1429    0.1429    0.1429    0.1429    0.1429
```

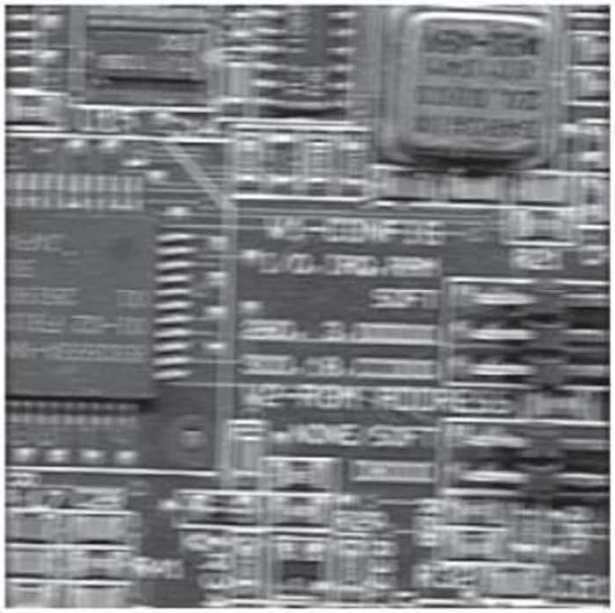



```
>> bc=imread('board.tif');
>> bg=im2uint8(rgb2gray(bc));
>> b=bg(100:355,50:305);
>> imshow(b)
```

```
>> m=fspecial('motion',7,0);
>> bm=imfilter(b,m);
>> imshow(bm)
```



(a)



(b)

FIGURE 8.20 *The result of motion blur.*

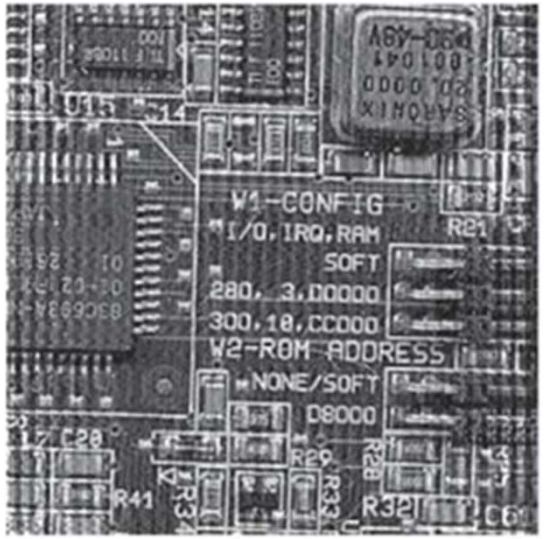
Motion Deblurring By Inverse Filtering

- To deblur the image, we need to divide its transform by the transform corresponding to the blur filter.
- This means that we first must create a matrix corresponding to the transform of the blur. => simple inverse filtering
- For more decent results, we can use pseudo inverse filter.

```
>> m2=zeros(256,256);
>> m2(1,1:7)=m;
>> mf=fft2(m2);
>> bmi=ifft2(fft2(bm)./mf);
>> fftshow(bmi,'abs')
```

Simple inverse filtering

```
>> d=0.02;
>> mf=fft2(m2);mf(find(abs(mf)<d))=1;
>> bmi=ifft2(fft2(bm)./mf);
>> imshow(mat2gray(abs(bmi))*2)
```



Pseudo inverse filtering

Restoration Using Minimum Mean Square Error (Wiener) Filter

- also called *least square error* filter
- Inverse filter makes no explicit provision for handling noise.
- Thus we need to incorporate both the degradation function and statistical characteristics of noise into the restoration process. => Wiener filter
- Objective: to find an estimate f_hat of the true image f such that MSE is minimized.

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

expectation value

Wiener Filter

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

$H^*(u,v)$: complex conjugate of $H(u,v)$

$|H(u,v)|^2 = H^*(u,v)H(u,v)$

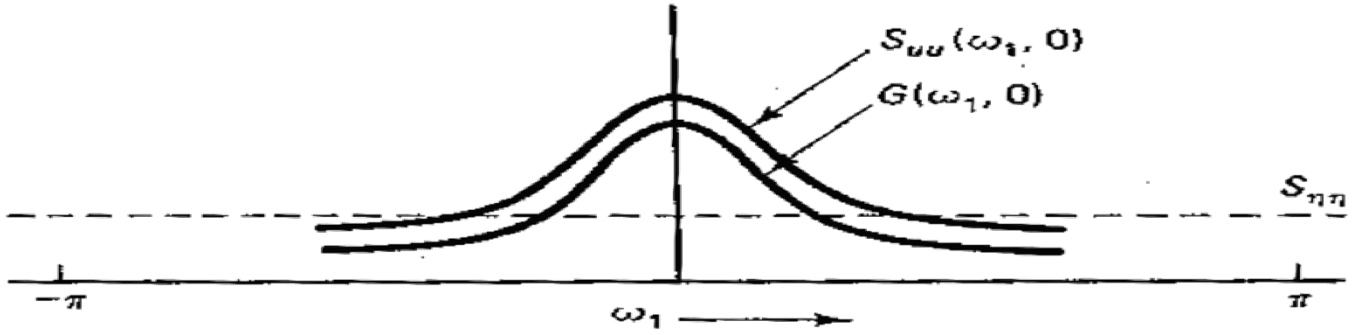
$S_\eta(u,v) = |N(u,v)|^2$: power spectrum of noise

$S_f(u,v) = |F(u,v)|^2$: power spectrum of true image : usually unknown

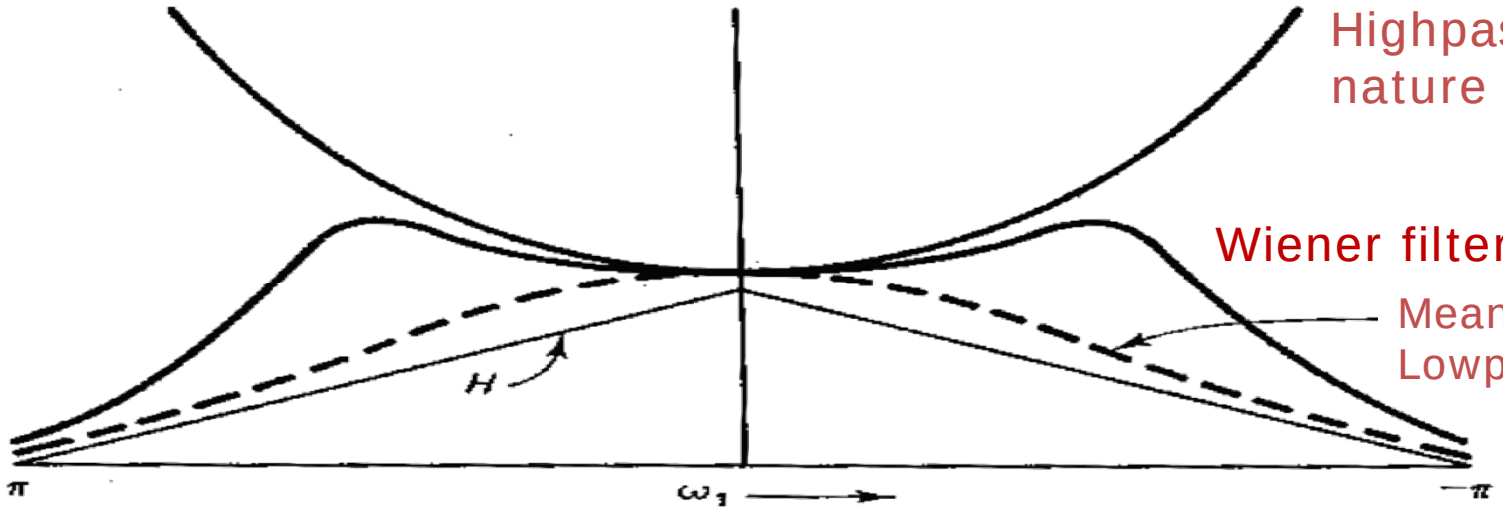
$$\hat{F}(u,v) = \left[\frac{1}{H(u,v)} \frac{|H(u,v)|^2}{|H(u,v)|^2 + K} \right] G(u,v)$$

approximated constant

Wiener Filtering: Bandpass Nature



(a) Noise smoothing ($H = 1$)



(b) Deblurring

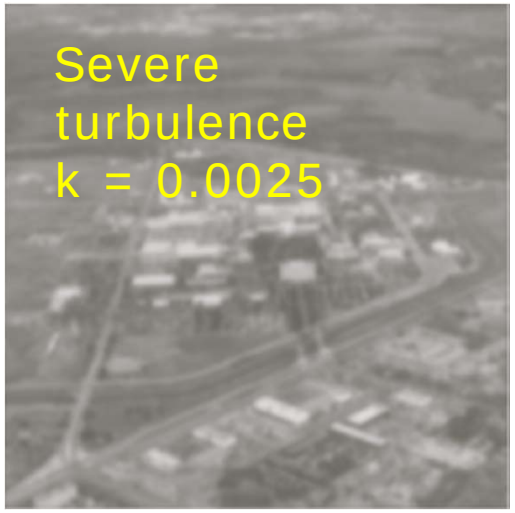
Inverse filter: Highpass nature

Wiener filter

Mean filter: Lowpass nature

Figure 8.11 Wiener filter characteristics.

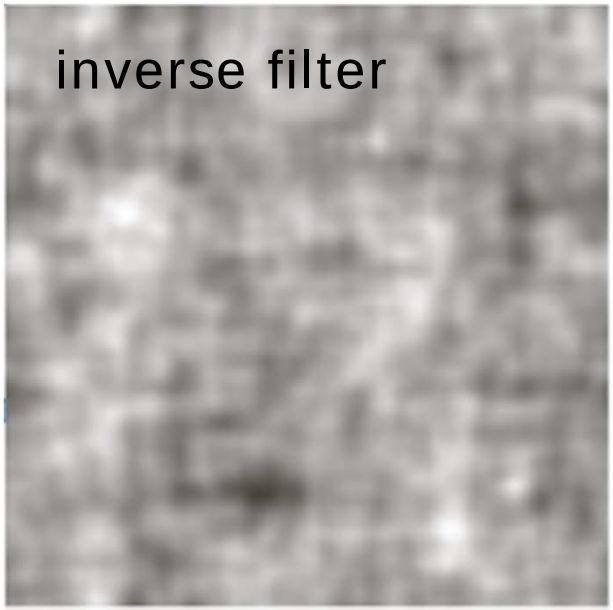
Wiener Filter



+

Gaussian noise
mean: 0
variance: 650

K was chosen empirically to produce best results.



Wiener Filtering in Matlab

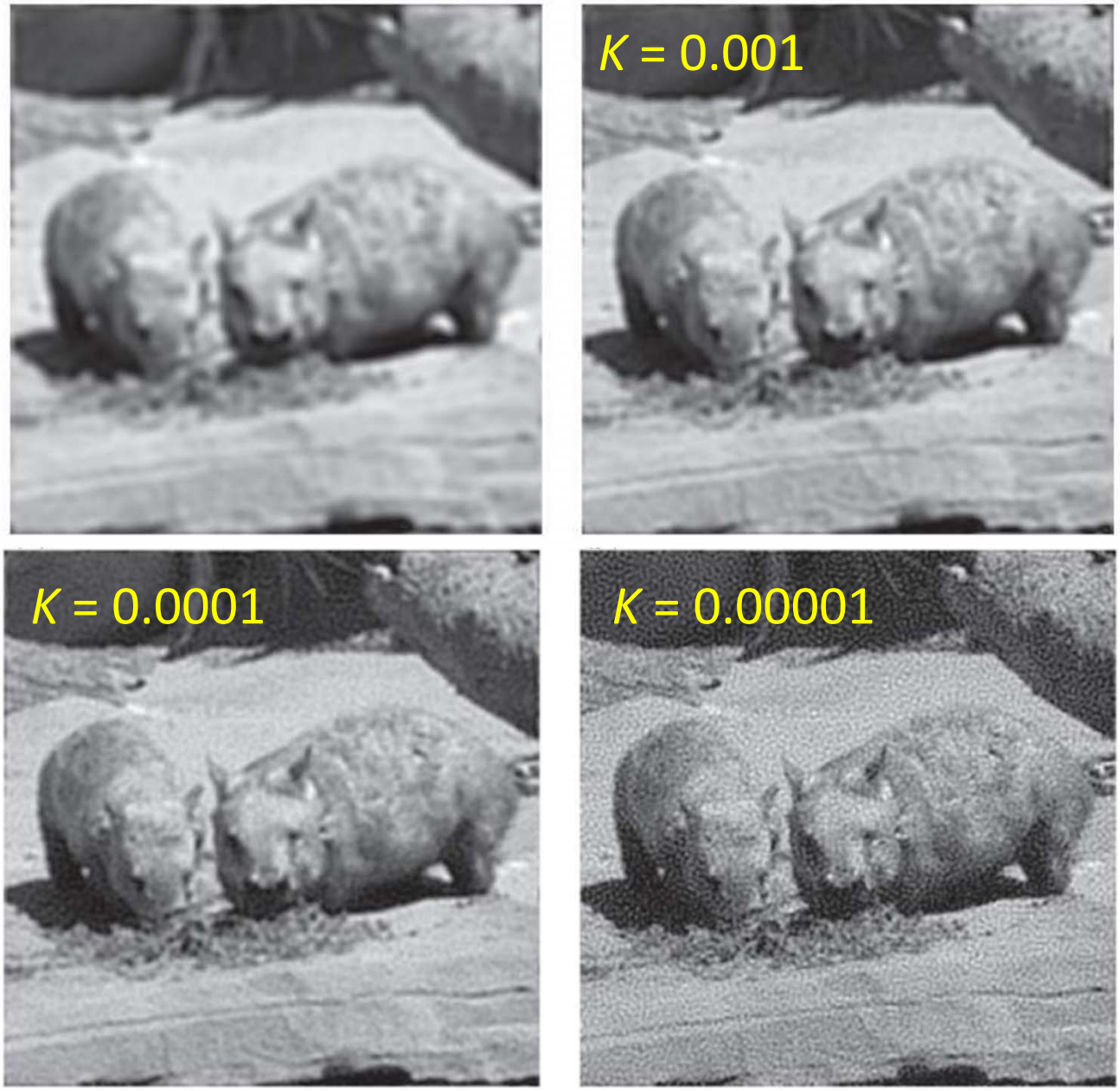
$$Y(i,j) = X(i,j)F(i,j) + N(i,j) \Rightarrow X(i,j) = \frac{Y(i,j) - N(i,j)}{F(i,j)}$$

$$X(i,j) \approx \left[\frac{1}{F(i,j)} \frac{|F(i,j)|^2}{|F(i,j)|^2 + K} \right] Y(i,j)$$

```
>> w=imread('wombats.tif');
>> wf=fftshift(fft2(w));
>> b=1butter(w,15,2);
>> wb=wf.*b;
>> wba=abs(ifft2(wb));
>> k=0.01;
>> wbf=fftshift(fft2(wba));
>> w1=wbf.*(abs(b).^2./(abs(b).^2+k)./b);% This is the equation
>> w1a=abs(ifft2(w1));
>> imshow(mat2gray(w1a))
```

Wiener Filter

As K becomes very small, noise starts to dominate the image



Summary

- **Chap 8. Image Restoration**
  - ✓ Noise Model: S&P, Gaussian, Periodic
  - ✓ Restoring in Spatial Domain
    - ✓ Restoration from S&P: LPF, Rank-Order Filter, Outlier Method
    - ✓ Restoration from Gaussian Noise: Image Averaging, LPF, Adaptive Filter
  - ✓ Restoring in Frequency Domain
    - ✓ Restoration from Periodic Noise: Band Reject Filter, Notch Filter
    - ✓ Restoration from motion blur: Inverse Filter, Pseudo-Inverse Filter, Wiener Filter