Introduction to Digital Image Processing with MATLAB® Asia Edition

McAndrew·Wang·Tseng

Chapter 8:

Image Restoration

CENGAGE Learning

8.1 Introduction

- **Image restoration** concerns the removal or reduction of degradations that have occurred during the acquisition of the image
- Some restoration techniques can be performed very successfully using neighborhood operations, while others require the use of frequency domain processes

8.1.1 A Model of Image Degradation

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

$f(x, y)$: image $h(x, y)$: spatial filter

✓ Where the symbol $*$ represents **convolution**

- In practice, the noise $n(x, y)$ must be considered

$$g(x,y) = f(x,y) * h(x,y) + \boxed{n(x,y)}$$

Ch8-p.191

CENGAGE Learning

8.1.1 A Model of Image Degradation

- We can perform the same operations in the frequency domain, where **convolution** is replaced by **multiplication**

$$G(i,j) = F(i,j)H(i,j) + N(i,j)$$

- If we knew the values of H and N, we could recover F by writing the above equation as

$$F(i,j) = (G(i,j) - N(i,j))/H(i,j)$$

this approach may not be practical

8.2 Noise

- **Noise—**any degradation in the image signal caused by external disturbance
- These errors will appear on the image output in different ways depending on the **type of disturbance** in the signal
- Usually we **know what type of errors** to expect and the type of noise on the image; hence, we can choose the most **appropriate method** for reducing the effects

Ch8-p.192

CENGAGE Learning

8.2.1 Salt and Pepper Noise

- 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');
```

FIGURE 8.1

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

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

(a) (b)

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

8.2.2 Gaussian Noise

- **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');
```


Ch8-p.193

8.2.3 Speckle Noise

- **Speckle** noise (or more simply just speckle) can be modeled by random values multiplied by pixel values

- It is also called **multiplicative noise**

$$I(1+N)$$

- `imnoise` can produce speckle

```
>> t_spk=imnoise(t,'speckle');
```


Ch8-p.194

CENGAGE Learning

FIGURE 8.2

(a) (b)

FIGURE 8.2 *The twins image corrupted by Gaussian and speckle noise. (a) Gaussian noise. (b) Speckle noise.*

8.2.4 Periodic Noise

```
>> 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;
```

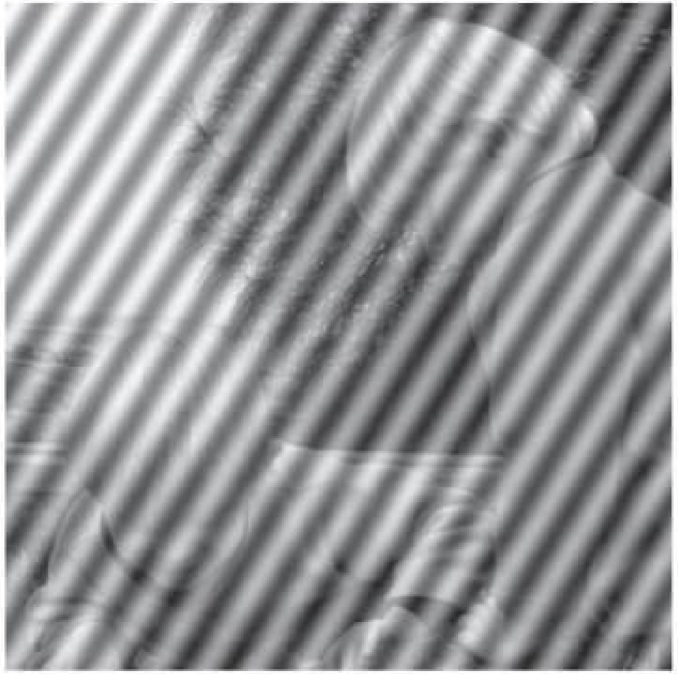


FIGURE 8.3 *The twins image corrupted by periodic noise.*

8.3 Cleaning Salt and Pepper Noise

- **Low-Pass Filtering**

```
>> 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.*

8.3.2 Median Filtering

50	65	52
63	255	58
61	60	57

$\longrightarrow$ 50 52 57 58 $\boxed{60}$ 61 63 65 255 $\longrightarrow$ 60

```
>> t_sp_m3=medfilt2(t_sp);
```

FIGURE 8.5 *Cleaning salt and pepper noise with a median filter.*

Ch8-p.197

FIGURE 8.6

```
>> 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.*

FIGURE 8.7

```
>> 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.*

8.3.3 Rank-Order Filtering

- Median filtering is a special case of a more general process called **rank-order filtering**
- A mask as 3×3 cross shape

0	1	0
1	1	1
0	1	0

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


Ch8-p.199

8.3.4 An Outlier Method

- Applying the median filter can in general be a slow operation: each pixel requires the sorting of at least nine values

- **Outlier Method**
  - ✓ 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 as noisy, otherwise not
  - ✓ If the pixel is noisy, replace its value with m; otherwise leave its value unchanged

FIGURE 8.8

```
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.*

Ch8-p.200

CENGAGE Learning

FIGURE 8.9

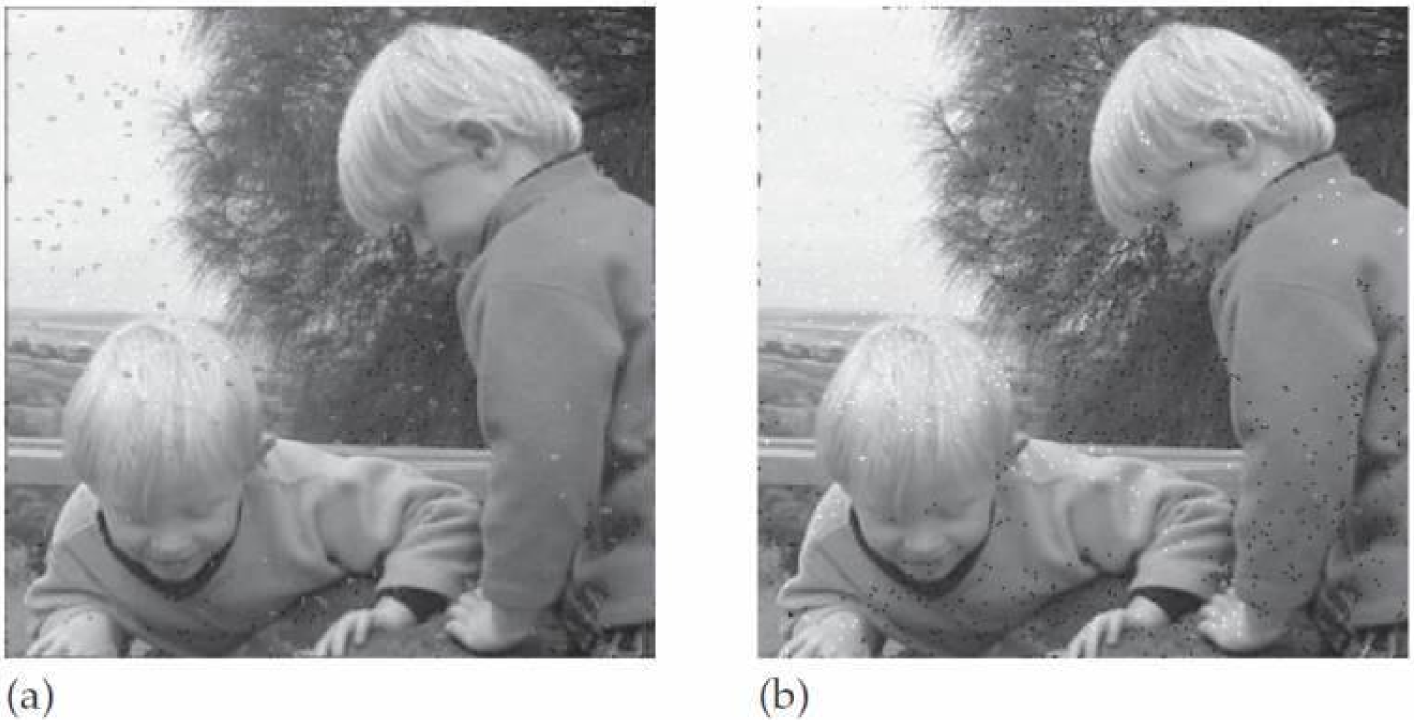


FIGURE 8.9 *Applying the outlier method to 10% salt and pepper noise. (a) $D = 0.2$. (b) $D = 0.4$.*

Ch8-p.201

8.4 Cleaning Gaussian Noise

- **Image Averaging**
  - ✓ suppose we have 100 copies of our image, each with noise

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

  - ✓ 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

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

Ch8-p.202

CENGAGE Learning

FIGURE 8.10

```
>> 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);
```

(a)

(b)

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

Ch8-p.203

8.4.2 Average Filtering

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

(a)

(b)

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

Ch8-p.203

8.4.3 Adaptive Filtering

- **Adaptive filters** are a class of filters that change their characteristics according to the values of the grayscales under the mask

  ✓ **Minimum mean-square error filter**

$$m_f + \frac{\sigma_f^2}{\sigma_f^2 + \sigma_g^2}(g - m_f)$$

The noise may not be normally distributed with mean 0

8.4.3 Adaptive Filtering

- **Wiener filters (`wiener2`)**

$$m_f + \frac{\max\{0, \sigma_f^2 - n\}}{\max\{\sigma_f^2, n\}}(g - m_f)$$

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

FIGURE 8.12

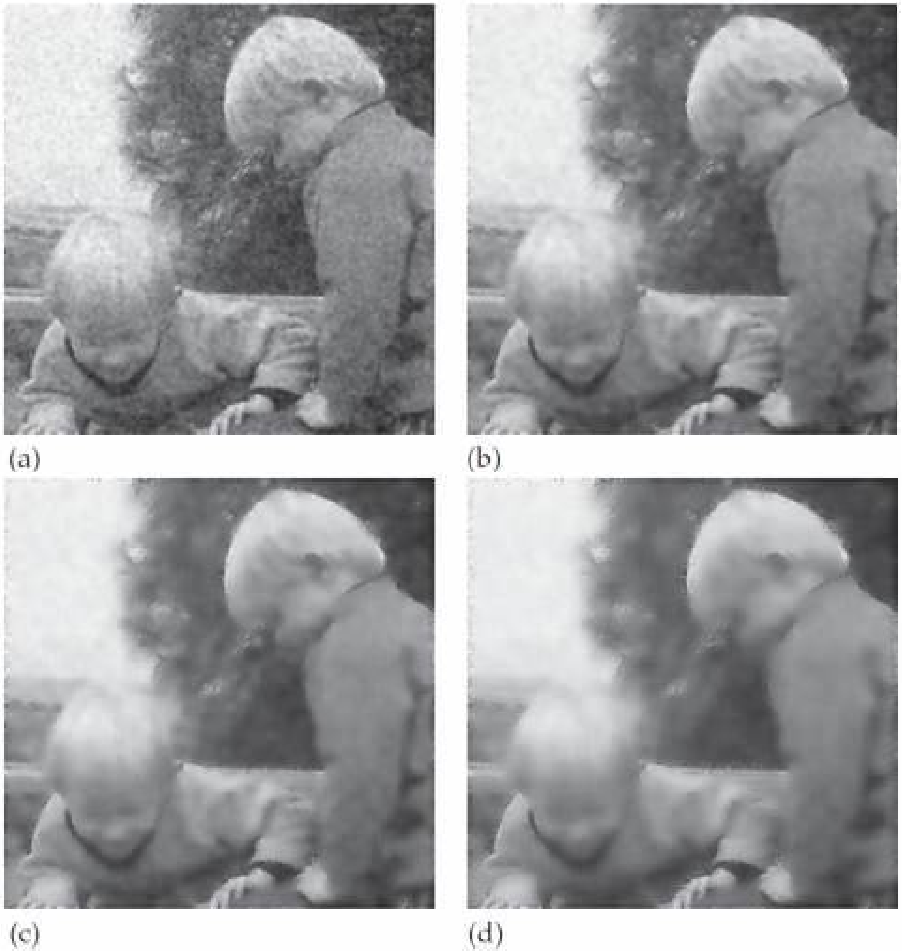


FIGURE 8.12 *Examples of adaptive filtering to remove Gaussian noise. (a) 3 × 3 filtering. (b) 5 × 5 filtering. (c) 7 × 7 filtering. (d) 9 × 9 filtering.*

Ch8-p.206

FIGURE 8.13

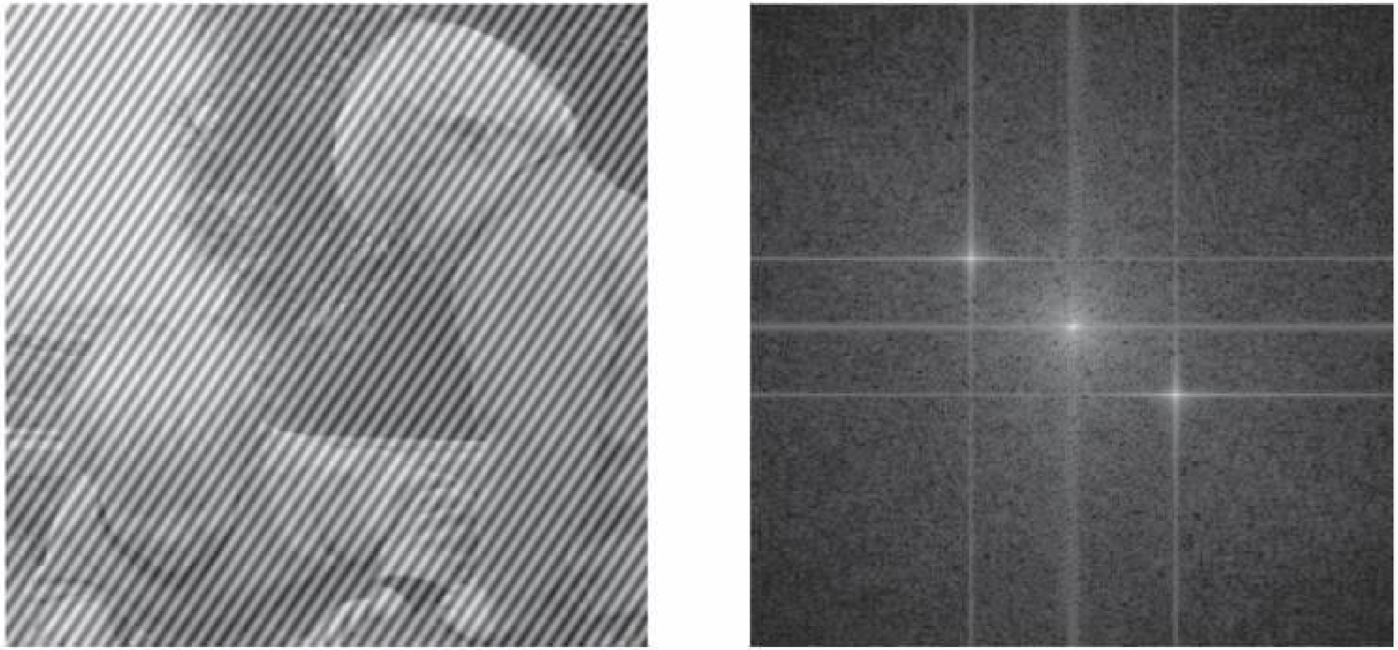
```
>> t2=imnoise(t,'gaussian',0,0.005);
>> imshow(t2)
>> t2w=wiener2(t2,[7,7]);
>> figure,imshow(t2w)
```

FIGURE 8.13 *Using adaptive filtering to remove Gaussian noise with low variance.*

Ch8-p.206

8.5 Removal of Periodic Noise

```
>> [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)

FIGURE 8.14 *The twins image (a) with periodic noise, and (b) its transform.*

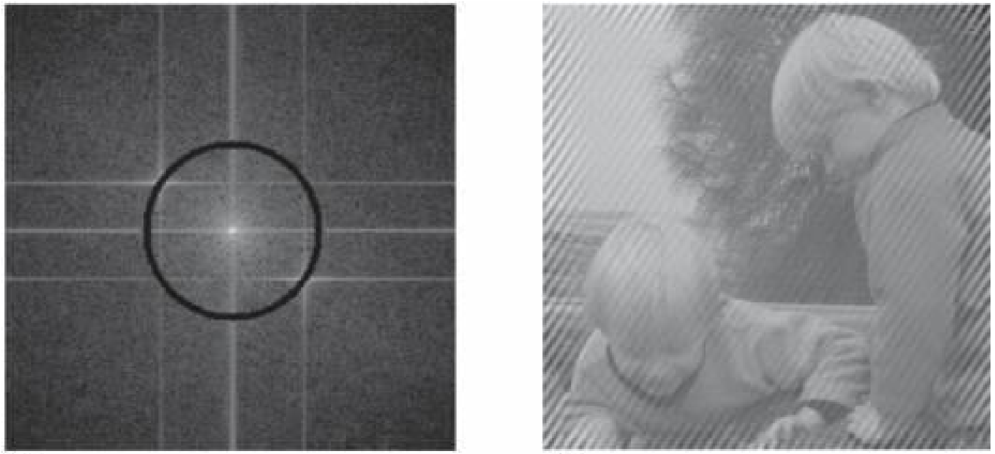
FIGURE 8.15

- **BAND REJECT FILTERING**

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

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

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



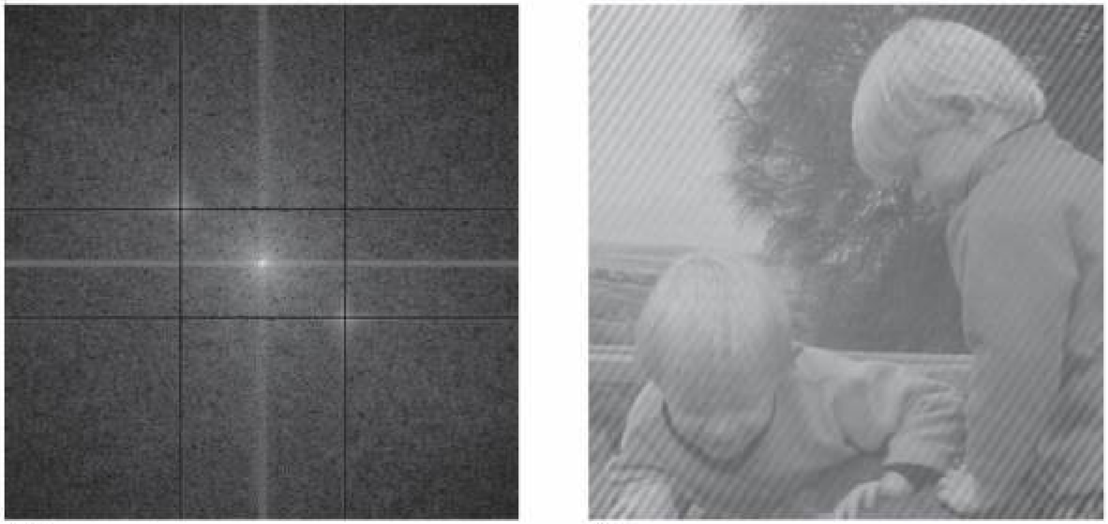
(a) (b)

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

FIGURE 8.16

- **NOTCH FILTERING**

```
>> 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.*

Ch8-p.209

8.6 Inverse Filtering

```
>> 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))
```

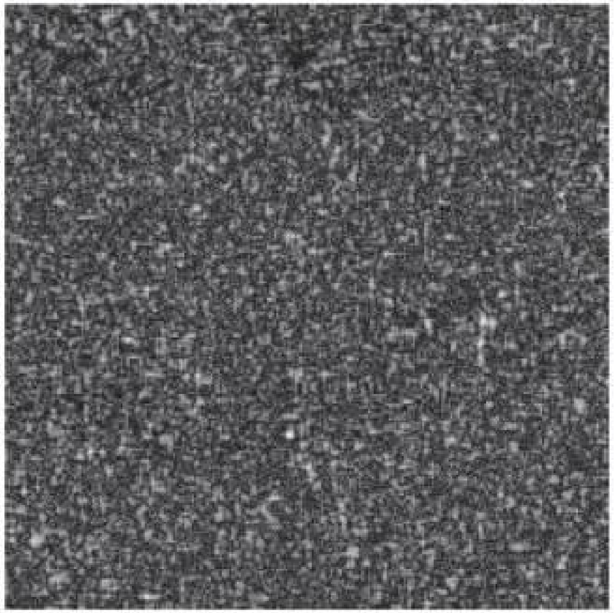


FIGURE 8.17 *An attempt at inverse filtering.*

Ch8-p.210

CENGAGE Learning

FIGURE 8.18

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

cutoff radius: 60

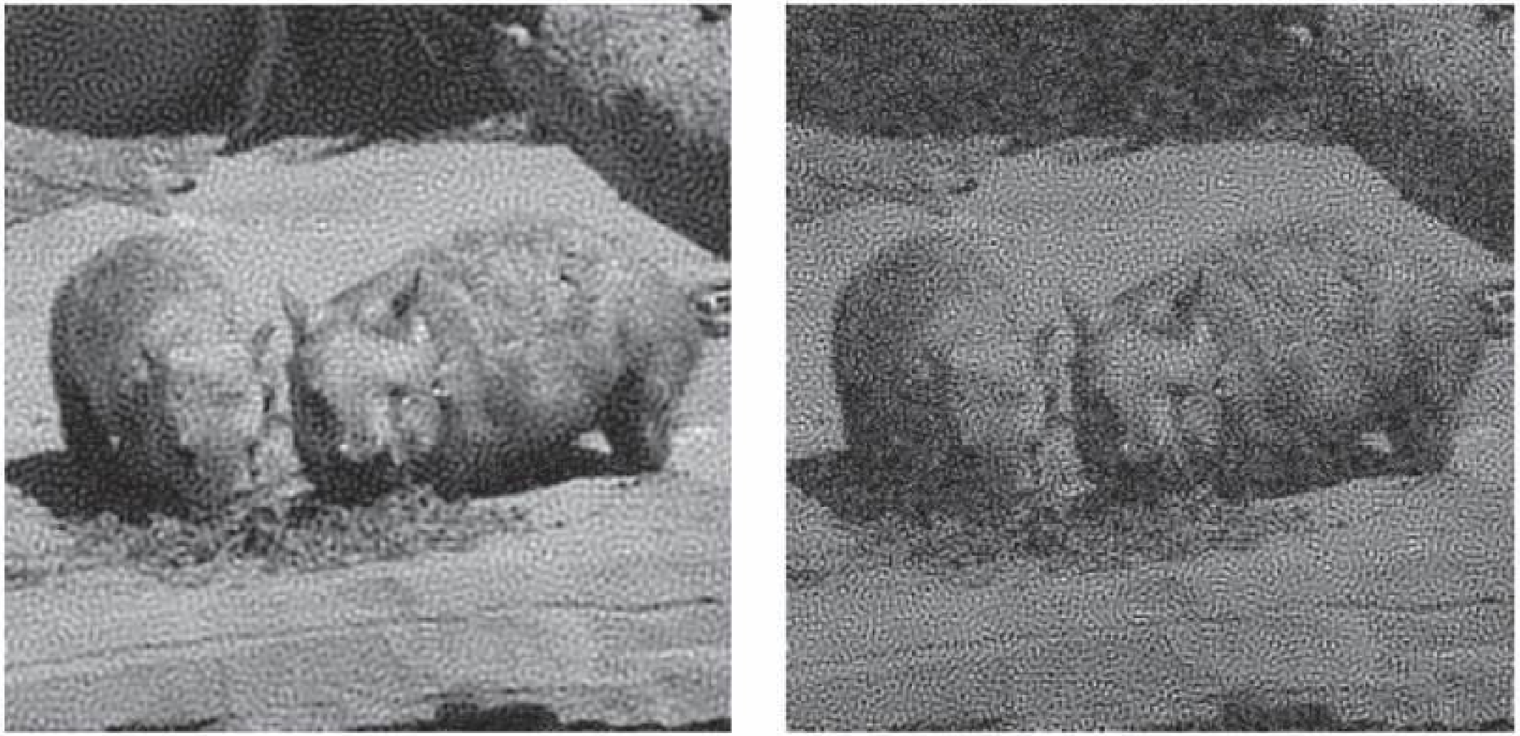
(a)

(b)

Ch8-p.211

FIGURE 8.18

cutoff radius: 80 cutoff radius: 100



(c) (d)

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

FIGURE 8.19

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

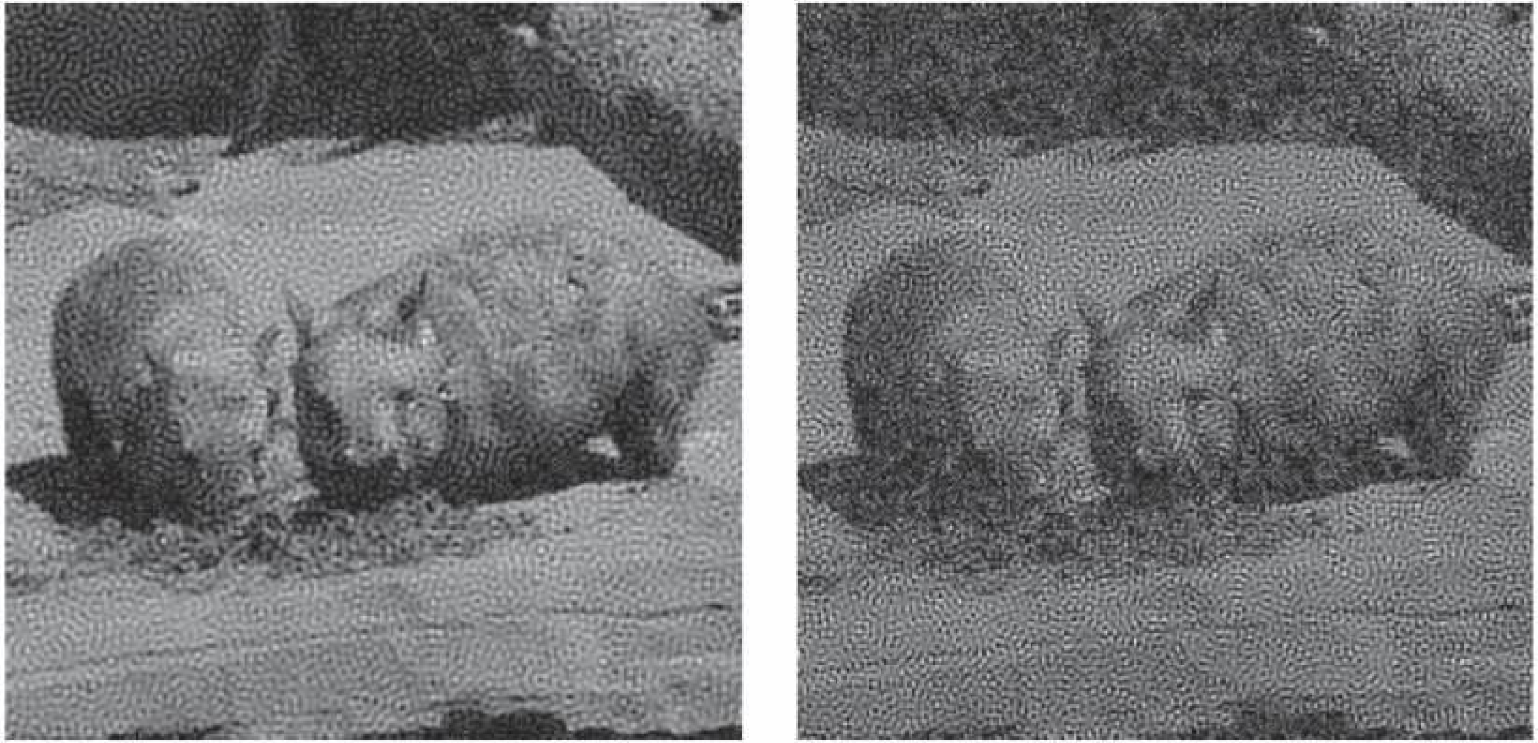
$d = 0.005$

(a)

(b)

FIGURE 8.19

d = 0.002 d = 0.001



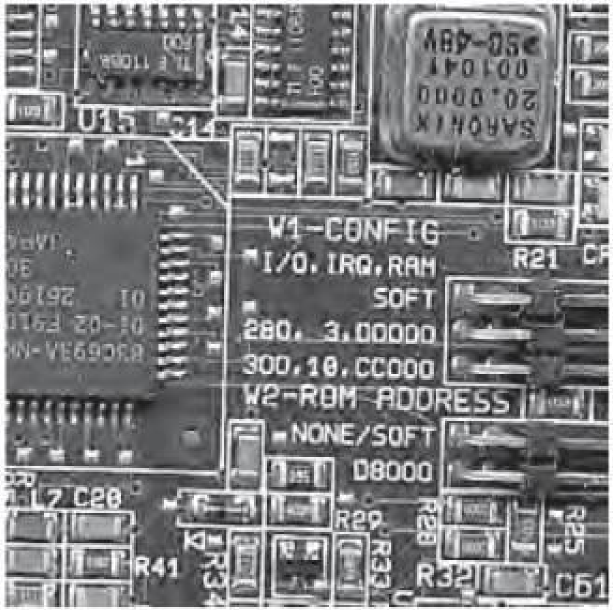
(c) (d)

FIGURE 8.19 *Inverse filtering using constrained division.*

8.6.1 Motion Deblurring

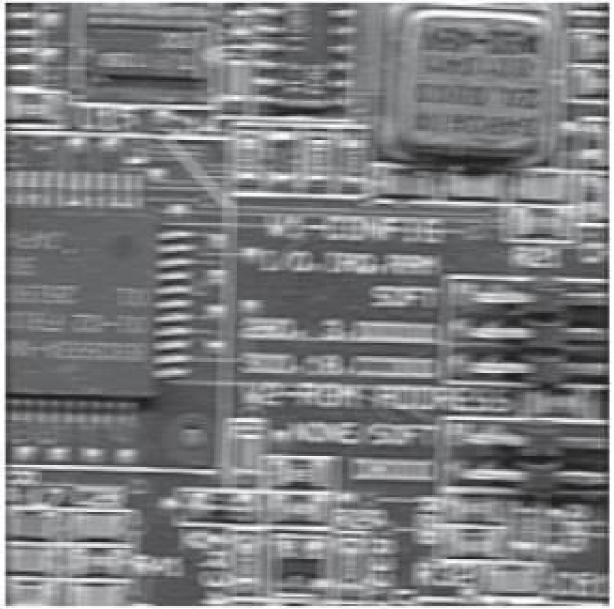
```
>> bc=imread('board.tif');
>> bg=im2uint8(rgb2gray(bc));
>> b=bg(100:355,50:305);
>> imshow(b)
```

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



Mathworks

(a)



(b)

FIGURE 8.20 *The result of motion blur.*

8.6.1 Motion Deblurring

- 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

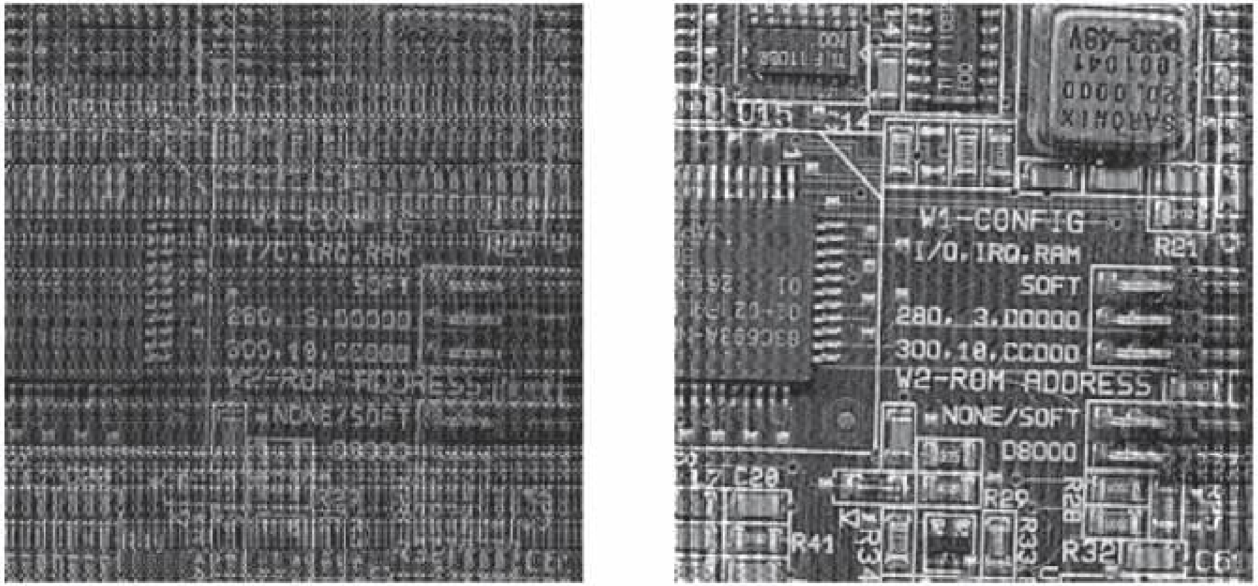
```
>> m2=zeros(256,256);
>> m2(1,1:7)=m;
>> mf=fft2(m2);
```


Ch8-p.215

FIGURE 8.21

```
>> bmi=ifft2(fft2(bm)./mf);
>> fftshow(bmi,'abs')
```

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



Mathworks

(a) (b)

FIGURE 8.21 *Attempts at removing motion blur. (a) Straight division. (b) Constrained division.*

CENGAGE Learning

8.7 Wiener Filtering

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

```
>> w=imread('wombats.tif');
>> wf=fftshift(fft2(w));
>> b=lbutter(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))
```


Ch8-p.216

CENGAGE Learning

FIGURE 8.22

$K = 0.001$

(a) (b)

Ch8-p.217

CENGAGE Learning

FIGURE 8.22

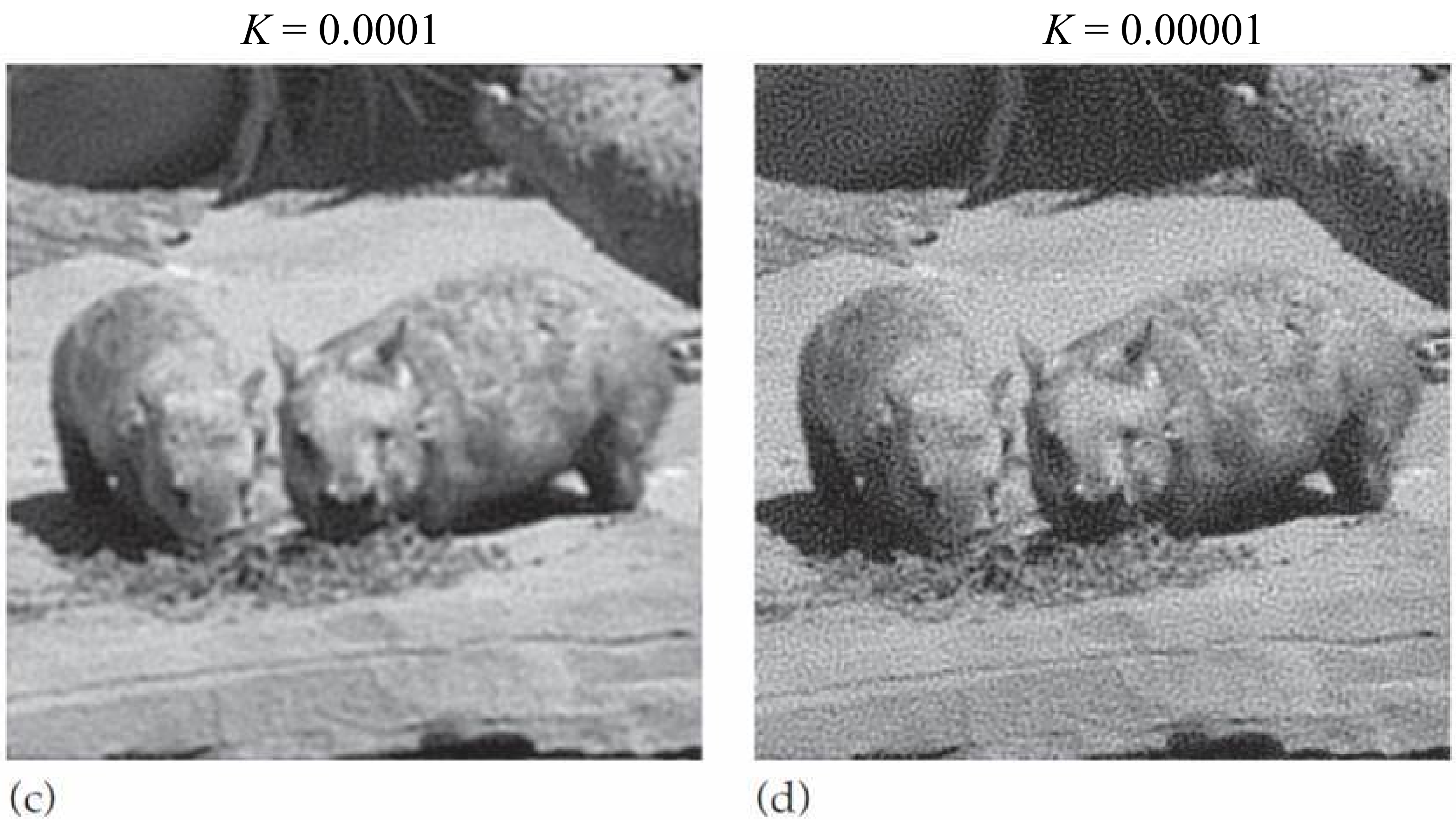


FIGURE 8.22 *Wiener filtering.*