CHAPTER 12

IMAGE RESTORATION

WHAT WILL WE LEARN?

- What is noise (in the context of image processing) and how can it be modeled?
- What are the main types of noise that may affect an image?
- What is blurring (in the context of image processing) and how can it be modeled?
- Which noise removal techniques are typically used in image processing?
- Which deblurring techniques are typically used in image processing?

12.1 MODELING OF THE IMAGE DEGRADATION AND RESTORATION PROBLEM

This chapter presents techniques used to improve the appearance of an image that has been subject to degradation and noise. Figure 12.1 shows a diagram of the degradation and restoration processes. In this diagram, it is assumed that an original image $f(x, y)$ has been subject to some sort of quality degradation (e.g., blurring caused by lack of focus or camera motion, atmospheric disturbances, or geometric distortions caused by imperfect lenses) that can be modeled by a function $h(x, y)$. The image may (also) have been contaminated by additive noise ($n(x, y)$). The resulting degraded image, $g(x, y)$, is the input for the image restoration algorithms described later. These algorithms are usually implemented as restoration filters that are able to undo—to some extent—the

Practical Image and Video Processing Using MATLAB®. By Oge Marques.

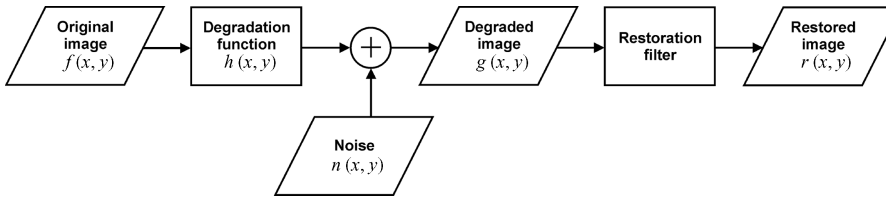

FIGURE 12.1 Image degradation and restoration.

degradation process, resulting in a restored image $r(x, y)$ (which should be interpreted as an estimate of the original image $f(x, y)$ and is often referred to as $\hat{f}(x, y)$). In other words, the goal of restoration techniques is to obtain an image that is as close to the original image as possible.

Mathematically, the degradation and restoration problem can be described as

$$g(x, y) = f(x, y) * h(x, y) + n(x, y) \tag{12.1}$$

where $*$ denotes convolution.

From the convolution theorem, the following frequency-domain relation among the Fourier transform of $f(x, y)$, $g(x, y)$, $h(x, y)$, and $n(x, y)$ holds:

$$G(u, v) = F(u, v)H(u, v) + N(u, v) \tag{12.2}$$

The *restoration filter* block in Figure 12.1 is typically designed following these steps:

1. Collect knowledge about the degradation process (usually through examples of degraded images and knowledge of the image acquisition process).
2. Use that knowledge to develop a degradation model.
3. Develop the *inverse* degradation process and model it as a filter.

Restoration filters are, therefore, specifically designed to solve one particular type of degradation. They can work in the spatial domain or in the frequency domain.

Note that the goal of image *restoration* techniques is clearly distinct from the goal of image *enhancement* techniques—such as the ones described in Chapters 8–11—where no mathematical modeling of (the inverse of) a degradation process is needed. This chapter presents representative examples of restoration filters, particularly for noise reduction and deblurring, and their application to digital images.

12.2 NOISE AND NOISE MODELS

Noise can be defined as any undesired artifact that contaminates an image. The presence of noise in an image can be due to several sources, resulting in different types of noise, from thermal noise in acquisition devices to periodic noise in the

communication channel used to transmit an image from a remote sensing location to a base station, among many others.

In this section, we present an overview of the main types of noise that may be found in degraded digital images. Our main goals are threefold: (1) to describe the main noise models in a mathematical and graphical way, (2) to remind the reader that different types of noise will require different noise reduction techniques, and (3) to introduce the problem of noise estimation.

We treat noise as a random variable whose probability density function (PDF), or histogram, describes its shape and distribution across the range of gray levels. In addition to that spatial-domain representation, noise patterns can sometimes also be represented in the frequency domain (through their frequency spectrum).

12.2.1 Selected Noise Probability Density Functions

Gaussian Noise The PDF of a Gaussian random variable z is given by

$$p_{\mathrm{g}}(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(z-\bar{z})^2/2\sigma^2} \tag{12.3}$$

where z represents the gray level, $\bar{z}$ is its mean, and σ is its standard deviation (σ^2 is called the *variance* of z). A plot of this function is shown in Figure 12.2a.

Impulse (Salt and Pepper) Noise The PDF of (bipolar) impulse noise is given by

$$p_{\mathrm{sp}}(z) = \begin{cases} P_p & \text{for } z = p \\ P_s & \text{for } z = s \\ 0 & \text{otherwise} \end{cases} \tag{12.4}$$

where P_p and P_s are the probability of occurrence of pixel whose values are equal to p (pepper) or s (salt), respectively. A plot of this function is shown in Figure 12.2b.

Uniform Noise The histogram of uniform noise is given by

$$p_{\mathrm{u}}(z) = \begin{cases} \frac{1}{b-a} & \text{if } a \le z \le b \\ 0 & \text{otherwise} \end{cases} \tag{12.5}$$

where $a \ge 0$ and $0 < a < b$.

The mean of the uniform noise is given by

$$\bar{z} = \frac{a+b}{2} \tag{12.6}$$

whereas the variance is given by

$$\sigma^2 = \frac{(b-a)^2}{12} \tag{12.7}$$

A plot of this function is shown in Figure 12.2c.

Rayleigh Noise The PDF of Rayleigh noise is given by

$$p_{\mathrm{r}}(z) = \begin{cases} \frac{2}{b}(z-a)e^{-(z-a)^2/b} & \text{for } z \geq a \\ 0 & \text{for } z < a \end{cases} \tag{12.8}$$

where $a \geq 0$ and $0 < a < b$.

The mean and the variance of the Rayleigh PDF are given by

$$\bar{z} = a + \sqrt{\pi b/4} \tag{12.9}$$

and

$$\sigma^2 = \frac{b(4-\pi)}{4} \tag{12.10}$$

A plot of this function is shown in Figure 12.2d.

Gamma (Erlang) Noise The histogram of gamma (Erlang) noise is given by

$$p_{\mathrm{E}}(z) = \begin{cases} \frac{a^b z^{b-1}}{(b-1)!}e^{-az} & \text{for } z \geq 0 \\ 0 & \text{for } z < 0 \end{cases} \tag{12.11}$$

where $a > 0$, b is a positive integer, and "!" indicates factorial.

The mean and the variance of the gamma PDF are given by

$$\bar{z} = \frac{b}{a} \tag{12.12}$$

and

$$\sigma^2 = \frac{b}{a^2} \tag{12.13}$$

A plot of this function is shown in Figure 12.2e.

Exponential Noise The PDF of exponential noise (a special case of the Erlang PDF, with $b = 1$) is given by

$$p_{\exp}(z) = \begin{cases} ae^{-az} & \text{for } z \geq 0 \\ 0 & \text{for } z < 0 \end{cases} \tag{12.14}$$

The mean and the variance of the exponential PDF are given by

$$\bar{z} = \frac{1}{a} \tag{12.15}$$

and

$$\sigma^2 = \frac{1}{a^2} \tag{12.16}$$

A plot of this function is shown in Figure 12.2f.

■ **EXAMPLE 12.1**

Figures 12.3 and 12.4 show a test image contaminated by different types of noise and the resulting histograms. Since the test image contains only two gray levels, it is possible to map the histograms to the plots in Figure 12.2.[1]

12.2.2 Noise Estimation

It is often necessary to estimate the type of noise (and the main parameters of its PDF) before designing a solution for noise reduction. In cases where the acquisition device (sensor) is the primary source of noise, it is common to generate test images consisting of large patches of homogeneous gray-level values and observe the resulting histograms. If creating such test images is not possible, a widely used alternative is to crop a relatively large homogeneous region from an image and inspect its histogram: even though the histogram of the entire image may not provide an adequate hint as to the type of noise present in the image, the histogram of the cropped portion will.

■ **EXAMPLE 12.2**

Figure 12.5 shows the process of noise estimation. The addition of noise to the original image (a) causes dramatic changes to its histogram (from (c) to (d)), without, however, providing a reliable hint as to the type of noise present in the image (b). Inspecting the histogram (e) of the cropped portion of the image (indicated by a rectangle in (b)) allows us to conclude that the noise is of Gaussian type.

In MATLAB

MATLAB's IPT has a built-in function to add noise to an image: `imnoise`. It allows the generation of noisy versions of input images. It supports several types of noise, including Gaussian and salt and pepper additive noise and speckle multiplicative noise. It does not, however, support other types of noise described in this section (e.g., Rayleigh, uniform, Erlang, and exponential) (see Problem 12.1). You will learn how to use this function in Tutorial 12.1.

12.3 NOISE REDUCTION USING SPATIAL-DOMAIN TECHNIQUES

Noise reduction filters work under the assumption that the only degradation present in an image is additive noise.[2] Mathematically, equations (12.1) and (12.2) become

$$g(x, y) = f(x, y) + n(x, y) \tag{12.17}$$

and

$$G(u, v) = F(u, v) + N(u, v) \tag{12.18}$$

[1] Some of the histograms show a peak at the rightmost value because the corresponding noise PDF has a "long tail" and, consequently, many values get truncated at 1.

[2] From the perspective of Figure 12.1, you can think of the degradation function as an *all-pass* filter.

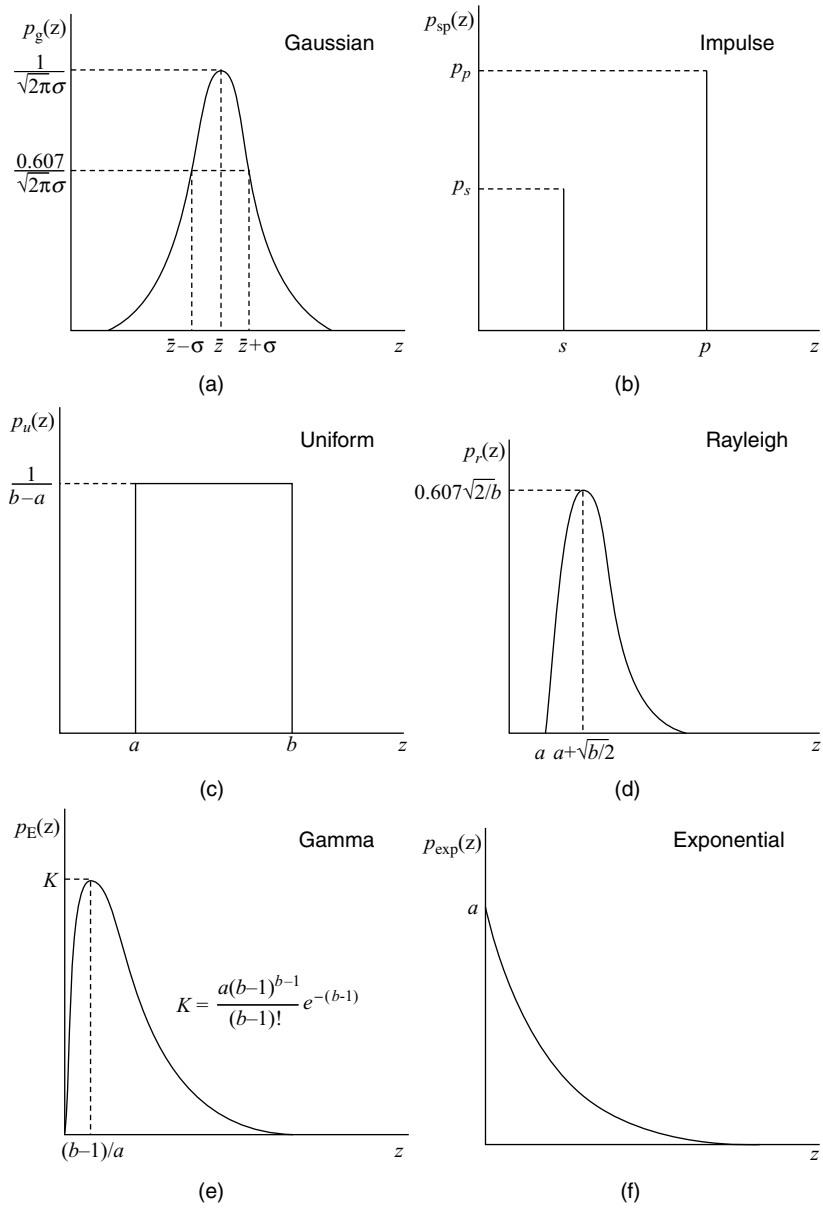

FIGURE 12.2 Histograms of representative noise types: (a) Gaussian, (b) impulse (salt and pepper), (c) uniform, (d) Rayleigh, (e) gamma (Erlang), and (e) exponential. Redrawn from [Pra07].

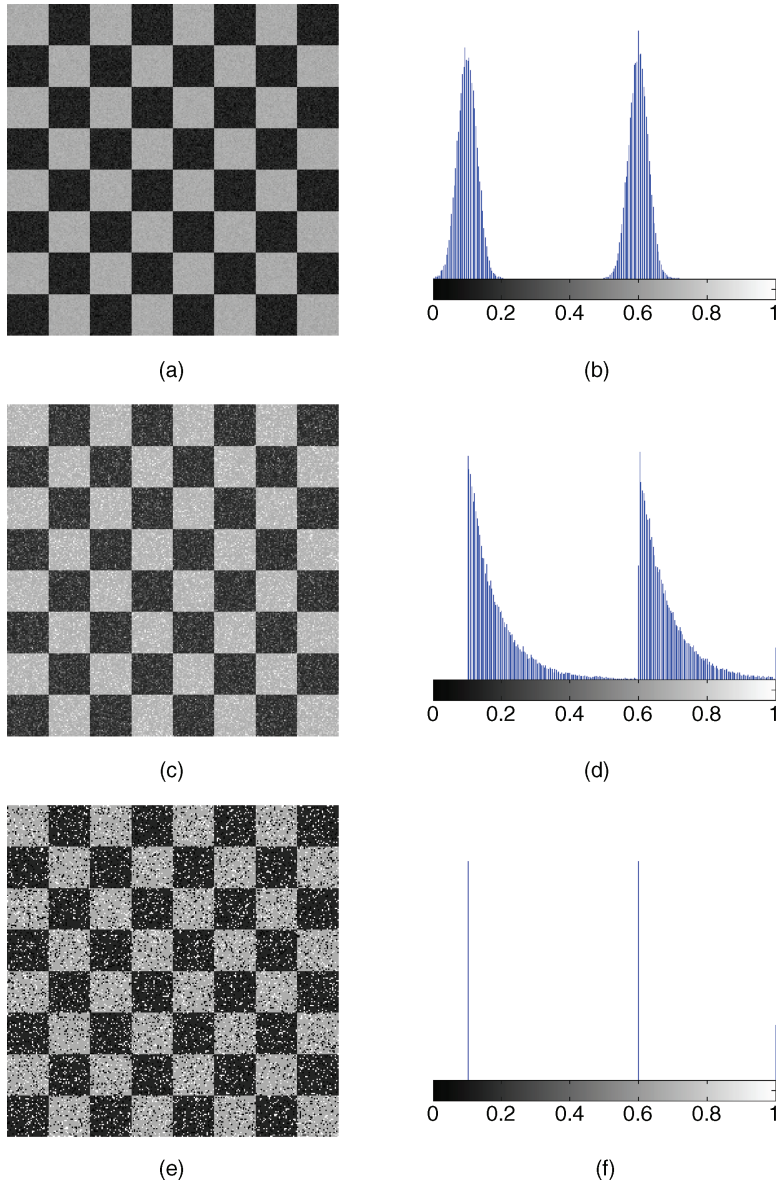

FIGURE 12.3 Test images and corresponding histograms for different types of noise: (a and b) Gaussian; (c and d) exponential; (e and f) salt and pepper.

This section presents the most popular spatial-domain techniques for noise reduction. Although some of them have been introduced previously (Chapter 10), they are being discussed again in this section. There are two main groups of spatial-domain noise reduction techniques: mean filters (Section 12.3.1) and order statistic (or *rank*)

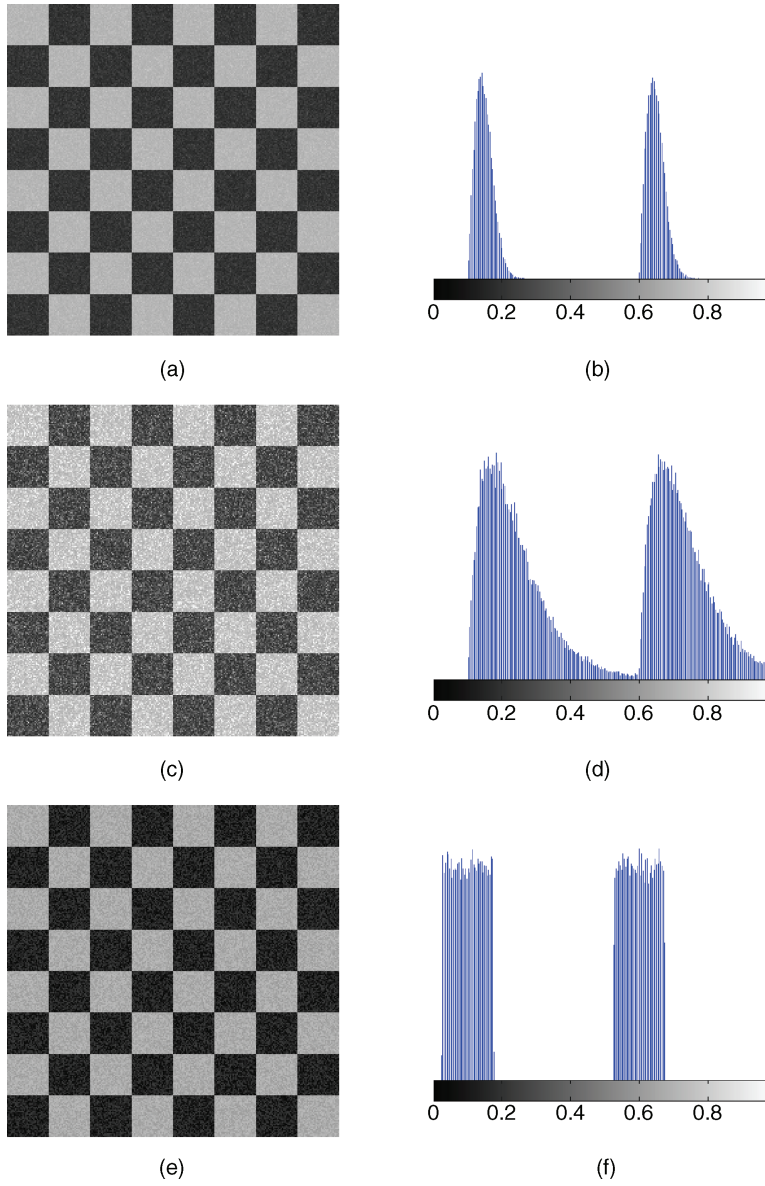

FIGURE 12.4 Test images and corresponding histograms for different types of noise: (a and b) Rayleigh; (c and d) Gamma; (e and f) uniform.

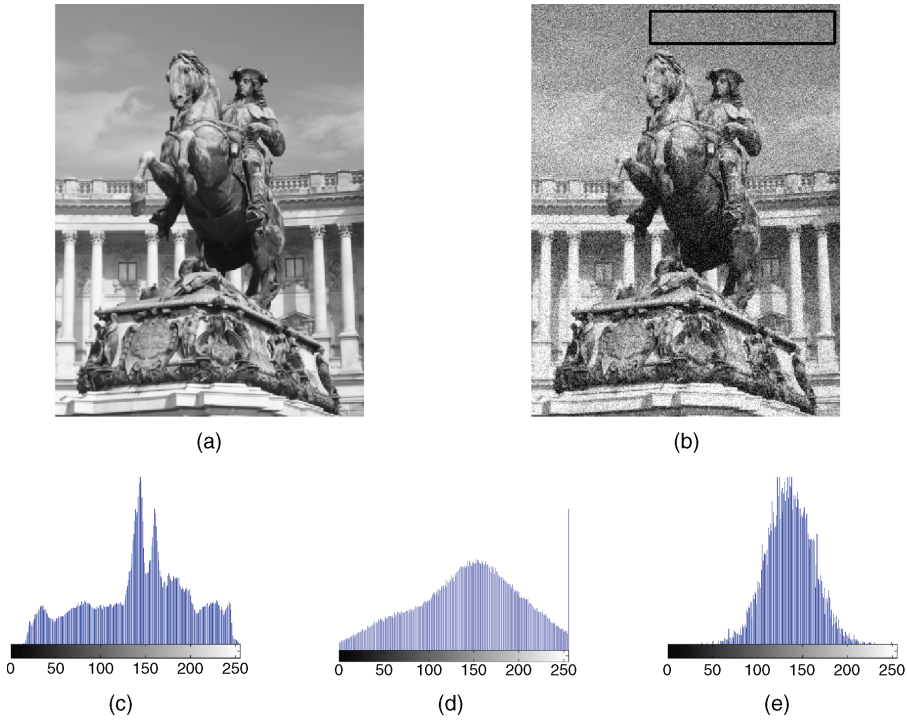

FIGURE 12.5 Estimating noise type from a homogeneous patch within an image: (a) original image; (b) noisy image (where the rectangle indicates a manually selected patch); (c) histogram of the original image; (d) histogram of the noisy image; (e) histogram of selected patch showing clearly that the noise is of Gaussian type in this case.

filters (Section 12.3.2). At the end of this section, we will briefly introduce the third category of spatial domain noise reduction techniques: adaptive filters.

12.3.1 Mean Filters

In this section, we revisit the arithmetic mean filter introduced in Section 10.3.1—from the perspective of its noise reduction capabilities—and expand the discussion to include other related filters and discuss their performance.

Arithmetic Mean Filter The arithmetic mean filter, also known as averaging filter, operates on an $m \times n$ sliding window by calculating the average of all pixel values within the window and replacing the center pixel value in the destination image with the result. Its mathematical formulation is given as follows:

$$\hat{f}(x, y) = \frac{1}{mn} \sum_{(r,c)\in W} g(r, c) \tag{12.19}$$

where g is the noisy image, $\hat{f}$ is the restored image, and r and c are the row and column coordinates, respectively, within a window W of size $m \times n$ where the operation takes place.

The arithmetic mean filter causes a certain amount of blurring (proportional to the window size) to the image, thereby reducing the effects of noise. It can be used to reduce noise of different types, but works best for Gaussian, uniform, or Erlang noise.

Geometric Mean Filter The geometric mean filter is a variation of the arithmetic mean filter and is primarily used on images with Gaussian noise. This filter is known to retain image detail better than the arithmetic mean filter. Its mathematical formulation is as follows:

$$\hat{f}(x, y) = \left[\prod_{(r,c)\in W} g(r, c) \right]^{1/mn} \tag{12.20}$$

Harmonic Mean Filter The harmonic mean filter is yet another variation of the arithmetic mean filter and is useful for images with Gaussian or salt noise. Black pixels (pepper noise) are not filtered. The filter's mathematical formulation is as follows:

$$\hat{f}(x, y) = \frac{mn}{\sum_{(r,c)\in W}(1/g(r, c))} \tag{12.21}$$

Contraharmonic Mean Filter The contra-harmonic mean filter is another variation of the arithmetic mean filter and is primarily used for filtering salt *or* pepper noise (but not both). Images with salt noise can be filtered using negative values of R, whereas those with pepper noise can be filtered using positive values of R. The filter's mathematical formulation is

$$\hat{f}(x, y) = \frac{\sum_{(r,c)\in W} g(r, c)^{R+1}}{\sum_{(r,c)\in W} g(r, c)^{R}} \tag{12.22}$$

where R is called the *order* of the filter.

■ EXAMPLE 12.3

Figure 12.6 shows the effects of applying different filters to an image corrupted by Gaussian noise of zero mean and variance 0.01. The result obtained with any of the four filters contains less noise than that with the image in Figure 12.6b. Moreover, you may also have noticed that the geometric and harmonic mean filters generate more dark pixels in the resulting image. Finally, the figure also helps to confirm the trade-off involved when choosing the mask size for a mean averaging filter: the result for Figure 12.6d (5×5 window) contains less noise—but is significantly more blurry—than the one for Figure 12.6c (3×3 window).

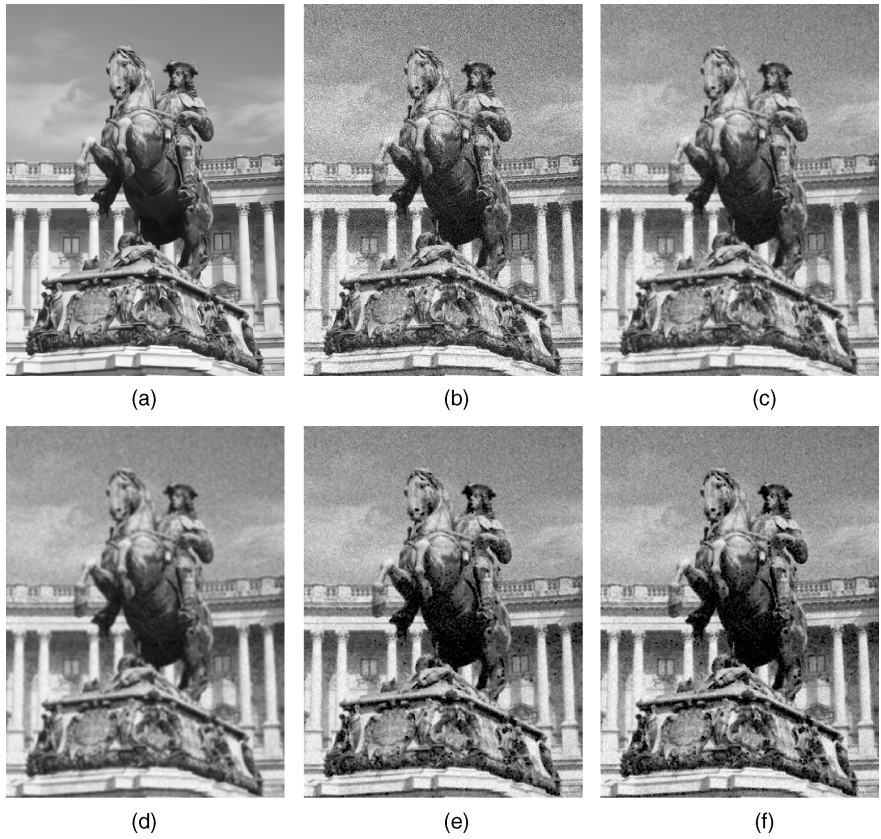

FIGURE 12.6 (a) Original image; (b) image with Gaussian noise; (c) result of 3×3 arithmetic mean filtering; (d) result of 5×5 arithmetic mean filtering; (e) result of 3×3 geometric mean filtering; (f) result of 3×3 harmonic mean filtering.

■ EXAMPLE 12.4

Figure 12.7 shows the effects of applying different filters to an image corrupted by salt and pepper noise. The result obtained with the mean averaging filter is the only acceptable one. The geometric and the harmonic mean filters perform very poorly for the pepper portion of the noise, whereas the performance of the contraharmonic filter confirms what we had stated earlier: depending on the choice of R, it will reduce the amount of salt *or* pepper noise, but not both.

12.3.2 Order Statistic Filters

Order statistic filters (also known as *rank filters*, or simply *order filters*) operate on a neighborhood around a reference pixel by ordering (i.e., ranking) the pixel values and then performing an operation on those ordered values to obtain the new value for

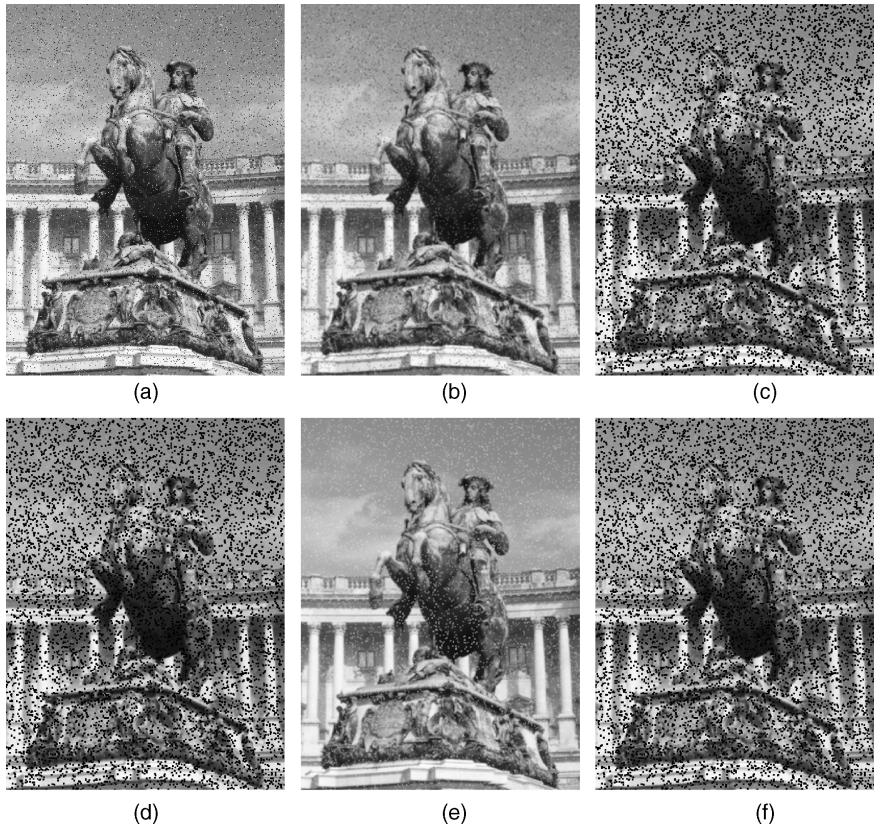

FIGURE 12.7 (a) Image with salt and pepper noise; (b) result of 3×3 arithmetic mean filtering; (c) result of 3×3 geometric mean filtering; (d) result of 3×3 harmonic mean filtering; (e) result of 3×3 contraharmonic mean filtering with $R = 0.5$; (f) result of 3×3 contraharmonic mean filtering with $R = -0.5$.

the reference pixel. Order statistic filters perform very well in the presence of salt and pepper noise but are computationally more expensive than mean filters.

In this section, we revisit the median filter introduced in Section 10.3.4—from the perspective of its noise reduction capabilities—and expand the discussion to include other related filters (the min filter, the max filter, the midpoint filter, and the alpha-trimmed mean filter) and discuss their performance.

The Median Filter The most popular and useful of the rank filters is the median filter. It works by selecting the middle pixel value from the ordered set of values within the $m \times n$ neighborhood (W) around the reference pixel. If mn is an even number (which is not common), the arithmetic average of the two values closest to the middle of the ordered set is used instead.

Mathematically,

$$\hat{f}(x, y) = \text{median}\{g(r, c)|(r, c) \in W\} \tag{12.23}$$

There have been many variants, extensions, and optimized implementations of the median filter proposed in the literature. Please refer to the "Learn More About It" section at the end of the chapter for useful pointers.

The Min and Max Filters The min and max filters also work on a ranked set of pixel values. Contrary to the median filter—which replaces the reference pixel with the median of the ordered set—the *min* filter, also known as the zeroth percentile filter, replaces it with the lowest value instead.

Mathematically,

$$\hat{f}(x, y) = \min\{g(r, c)|(r, c) \in W\} \tag{12.24}$$

Similarly, the *max* filter, also known as the 100th percentile filter, replaces the reference pixel within the window with the highest value, that is,

$$\hat{f}(x, y) = \max\{g(r, c)|(r, c) \in W\} \tag{12.25}$$

The min filter is useful for reduction of salt noise, whereas the max filter can help remove pepper noise.

The Midpoint Filter The midpoint filter calculates the average of the highest and lowest pixel values within a window, thereby combining order statistics and averaging into one filter. It is used to reduce Gaussian and uniform noise in images.

Mathematically,

$$\hat{f}(x, y) = \frac{1}{2}\left[\max\{g(r, c)|(r, c) \in W\} + \min\{g(r, c)|(r, c) \in W\}\right] \tag{12.26}$$

The Alpha-Trimmed Mean Filter The alpha-trimmed filter uses another combination of order statistics and averaging, in this case an average of the pixel values closest to the median, after the D lowest and D highest values in an ordered set have been excluded. The rationale behind this filter is to allow its user to control its behavior by specifying the parameter D: for $D = 0$, the filter behaves as a regular arithmetic mean filter; for $D = (mn - 1)/2$, it is equivalent to the median filter. It is used in cases where the image is corrupted by more than one type of noise, for example, salt and pepper (where the median filter performs well) and Gaussian (where the arithmetic mean filter shows satisfactory performance).

The mathematical description of the alpha-trimmed filter is as follows:

$$\hat{f}(x, y) = \frac{1}{mn - 2D} \sum_{(r,c)\in W} g(r, c) \tag{12.27}$$

where D is the number of pixel values excluded at each end of the ordered set, which can range from 0 to $(mn - 1)/2$.

■ **EXAMPLE 12.5**

Figure 12.8 shows the effects of applying different rank filters to an image corrupted by salt and pepper noise. The results obtained with the median filter (part (c)) are clearly better than the ones obtained with the average filter of same window size (part (b)) as expected. The midpoint filter not only is ineffective in removing this type of noise, but also makes it worse by replacing the noisy pixels (and their immediate neighbors) with the average between the minimum—pepper—and maximum—salt—values.

In MATLAB

Sliding window neighborhood operations are implemented in the IPT using one of these two functions: `nlfilter` or `colfilt`. Both functions accept a (user-defined) function as a parameter. Such function can perform linear (e.g., averaging) or nonlinear (e.g., median) operations on the pixels within a window. Both functions might take long to process results on an image with hundreds (or thousands) of pixels in each dimension; they both provide a progress bar indicator to inform to the user that the processing is taking place, but `colfilt` is considerably faster than `nlfilter`. You will have a chance to use `nlfilter` in Tutorial 12.1 (page 289).

For rank filters, the IPT function `ordfilt2` makes it very easy to create the min, max, and median filters, as demonstrated in Tutorial 12.1. Since the median filter is by far the most popular rank filter, the IPT has another function with a more familiar name, `medfilt2`, that implements it.

12.3.3 Adaptive Filters

The basic idea behind adaptive filters is to design the filter in such a way that its behavior changes depending on the pixel values of the neighborhood currently being processed. A classical use of adaptive filters is found under the category of *edge-preserving smoothing filters*: in this case, the goal is to apply a low-pass filter to an image in a selective way, minimizing the edge blurring effect that would be present if a standard LPF had been applied to the image. There are many variants of adaptive filters for noise reduction and image restoration in the literature. Refer to the "Learn More About It" section at the end of the chapter for useful pointers.

12.4 NOISE REDUCTION USING FREQUENCY-DOMAIN TECHNIQUES

In Chapter 11, we studied frequency-domain filters commonly used in image enhancement tasks, notably high-pass and low-pass filters. In this section, we introduce three additional types of frequency-domain filters—bandpass, bandreject,

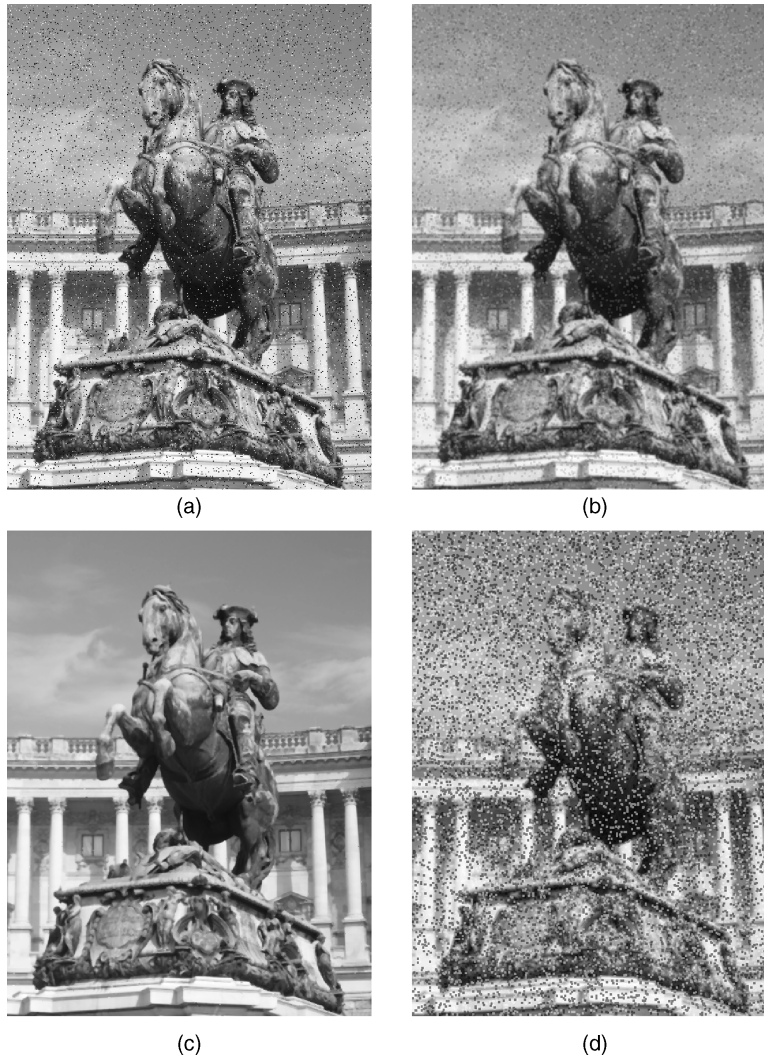

FIGURE 12.8 (a) Image with salt and pepper noise; (b) result of 3×3 arithmetic mean filtering (for comparison); (c) result of 3×3 median filtering; (d) result of 3×3 midpoint filtering.

and notch—whose main application is in the reduction or removal of periodic noise.

12.4.1 Periodic Noise

This is a type of noise that usually arises as a result of electrical or electromechanical interference during the image acquisition process.

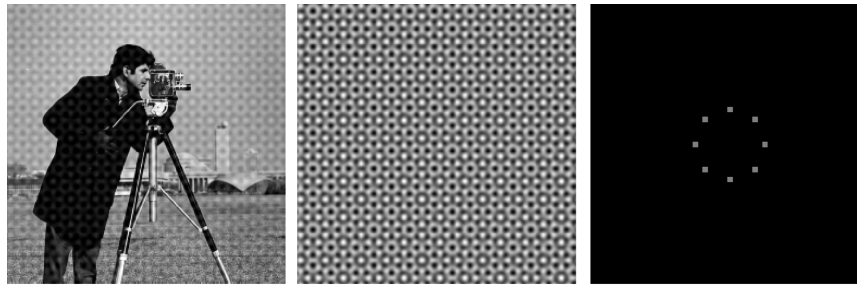

FIGURE 12.9 Example of an image corrupted by periodic noise: (a) noisy image; (b) periodic noise component; (c) the Fourier spectrum of the noise component (bright dots were enlarged for viewing purposes).

■ EXAMPLE 12.6

Figure 12.9 shows an example of an image corrupted by periodic noise, the noise component, and its Fourier spectrum. The bright dots in the spectrum indicate four pairs of impulse functions, each pair corresponding to a sinusoidal noise source.

12.4.2 Bandreject Filter

A bandreject filter, as its name suggests, attenuates frequency components within a certain range (the *stopband* of the filter), while leaving all other frequency components untouched (or amplifying them by a certain gain). The mathematical formulation for an *ideal* bandreject filter is as follows:

$$H_{\text{br}}^{\text{i}}(u,v) = \begin{cases} 1 & \text{if } D(u,v) < D_0 - \frac{W}{2} \\ 0 & \text{if } D_0 - \frac{W}{2} \leq D(u,v) \leq D_0 + \frac{W}{2} \\ 1 & \text{if } D(u,v) > D_0 + \frac{W}{2} \end{cases} \tag{12.28}$$

where $D(u, v)$ is the distance from the origin of the frequency spectrum, W is the width of the band, and D_0 is the radius of the circle-shaped band.

A *Butterworth* bandreject filter of order n is described mathematically as

$$H_{\text{br}}^{\text{b}}(u,v) = \frac{1}{1 + \left[\frac{D(u,v)W}{D^2(u,v) - D_0^2}\right]^{2n}} \tag{12.29}$$

whereas the transfer function for a *Gaussian* bandreject filter is given by the following equation:

$$H_{\text{br}}^{\text{g}}(u,v) = 1 - e^{-\frac{1}{2}\left[\frac{D^2(u,v) - D_0^2}{D(u,v)W}\right]^2} \tag{12.30}$$

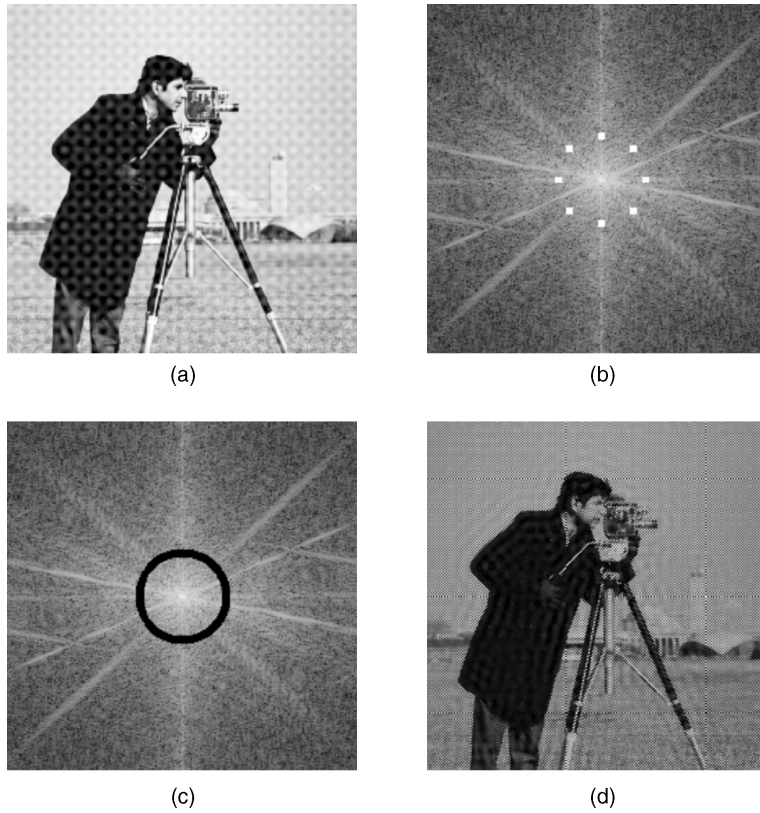

FIGURE 12.10 Example of using a bandreject filter to reduce periodic noise: (a) noisy image; (b) noisy image spectrum (the eight spots corresponding to the noise have been made brighter and bigger for visualization purposes); (c) the Fourier spectrum of the image after applying the bandreject filter; (d) resulting image.

■ EXAMPLE 12.7

Figure 12.10 shows an example of an ideal bandreject filter (of radius $D_0 = 32$ and width $W = 6$) used to reduce periodic noise. Although the periodic noise has been successfully removed, the resulting image shows an undesirable ringing effect as a result of the sharp transition between passband and stopband in the bandreject filter.

12.4.3 Bandpass Filter

A bandpass filter allows certain frequencies (within its passband) to be preserved while attenuating all others. It is, in effect, the opposite of a bandreject filter (Section 12.4.2), and its mathematical formulation can be simply stated as follows:

$$H_{\text{bp}}(u, v) = 1 - H_{\text{br}}(u, v) \tag{12.31}$$

where $H_{\text{br}}(u, v)$ is the transfer function of a bandreject filter.

Applying equation (12.31) to equations (12.28)–(12.30) leads to the mathematical formulations for the different types of bandpass filters:

Ideal BPF

$$H_{\text{bp}}^{i}(u,v) = \begin{cases} 0 & \text{if } D(u,v) < D_0 - \frac{W}{2} \\ 1 & \text{if } D_0 - \frac{W}{2} \leq D(u,v) \leq D_0 + \frac{W}{2} \\ 0 & \text{if } D(u,v) > D_0 + \frac{W}{2} \end{cases} \tag{12.32}$$

Butterworth BPF

$$H_{\text{bp}}^{\text{b}}(u,v) = \frac{\left[\frac{D(u,v)W}{D^2(u,v)-D_0^2}\right]^{2n}}{1+\left[\frac{D(u,v)W}{D^2(u,v)-D_0^2}\right]^{2n}} \tag{12.33}$$

Gaussian BPF

$$H_{\text{bp}}^{\text{g}}(u,v) = e^{-\frac{1}{2}\left[\frac{D^2(u,v)-D_0^2}{D(u,v)W}\right]^2} \tag{12.34}$$

12.4.4 Notch Filter

The *notch* filter is a special kind of frequency-domain filter that attenuates (or allows) frequencies within a neighborhood around a center frequency. Owing to the symmetry property of the Fourier transform (Section 11.2.3), the spectrum of notch filters shows symmetric pairs around the origin (except for a notch filter located at the origin, of course).

The mathematical formulation for an ideal notch filter of radius D_0 with center at (u_0, v_0) (and by symmetry at $(-u_0, -v_0)$) that rejects frequencies within a predefined neighborhood is

$$H_{\text{nr}}^{\text{i}}(u,v) = \begin{cases} 0 & \text{if } D_1(u,v) < D_0 \text{ or } D_2(u,v) < D_0 \\ 1 & \text{otherwise} \end{cases} \tag{12.35}$$

where

$$D_1(u,v) = \left[(u - M/2 - u_0)^2 + (v - N/2 - v_0)^2\right]^{1/2} \tag{12.36}$$

and

$$D_2(u,v) = \left[(u - M/2 + u_0)^2 + (v - N/2 + v_0)^2\right]^{1/2} \tag{12.37}$$

It is assumed that the spectrum has been shifted by $(M/2, N/2)$[3] and the values of (u_0, v_0) are relative to the shifted center.

A Butterworth notch filter of order n can be mathematically described as

$$H_{\text{nr}}^{\text{b}}(u, v) = \frac{1}{1 + \left[\frac{D_0^2}{D_1(u,v)D_2(u,v)}\right]^n} \tag{12.38}$$

where $D_1(u, v)$ and $D_2(u, v)$ are given by equations (12.36) and (12.37), respectively.

A Gaussian notch filter is mathematically described by the following equation:

$$H_{\text{nr}}^{\text{g}}(u, v) = 1 - e^{-\frac{1}{2}\left[\frac{D_1(u,v)D_2(u,v)}{D_0^2}\right]} \tag{12.39}$$

To convert a notch filter that rejects certain frequencies into one that allows those same frequencies to pass, one has to simply compute

$$H_{\text{np}}(u, v) = 1 - H_{\text{nr}}(u, v) \tag{12.40}$$

where $H_{\text{np}}(u, v)$ is the transfer function of the notch pass filter that corresponds to the notch reject filter whose transfer function is $H_{\text{nr}}(u, v)$.

12.5 IMAGE DEBLURRING TECHNIQUES

The goal of image deblurring techniques is to process an image that has been subject to blurring caused, for example, by camera motion during image capture or poor focusing of the lenses. The simplest image deblurring filtering technique, *inverse filtering*, operates in the frequency domain, according to the model in Figure 12.1, and assuming that there is no significant noise in the degraded image,

$$G(u, v) = F(u, v)H(u, v) + 0 \tag{12.41}$$

which leads to

$$F(u, v) = \frac{G(u, v)}{H(u, v)} = G(u, v)\frac{1}{H(u, v)} \tag{12.42}$$

where the term $1/H(u, v)$ is the FT of the restoration filter, which will be denoted by $R_{\text{inv}}(u, v)$.

The simplicity of this formulation hides some of its pitfalls. If there are any points in $H(u, v)$ that are zero, a divide by zero exception will be generated.[4] Even worse, if the assumption of no additive noise is correct, the degraded image $H(u, v)$ will

[3]The shift can be performed using MATLAB function *fftshift*, similarly to what we did in the tutorials for Chapter 11.

[4]Even a milder version of this problem—where the values of $H(u, v)$ are close to zero—will lead to unacceptable results, as the following example demonstrates.

exhibit zeros at the same points, which will lead to a 0/0 indeterminate form. On the other hand, if the image is contaminated by noise—although we assumed it had not—the zeros will not coincide but the result of the inverse filter calculations will be heavily biased by the noise term. The latter problem will be handled by Wiener filters (Section 12.5.1).

There are two common solutions to the former problem:

1. Apply a low-pass filter with transfer function $L(u, v)$ to the division, thus limiting the restoration to a range of frequencies below the *restoration cutoff frequency*, that is,

$$F(u, v) = \frac{G(u, v)}{H(u, v)} L(u, v) \tag{12.43}$$

   For frequencies within the filter's passband, the filter's gain is set to any desired positive value (usually the gain is set to 1); for frequencies within the filter's stopband, the gain should be zero. If the transition between the passband and stopband is too sharp (as in the case of the ideal LPF), however, ringing artifacts may appear in the restored image; other filter types (e.g., Butterworth with high order) may be preferred in this case.
2. Use *constrained division* where a threshold value T is chosen such that if $|H(u, v)| < T$, the division does not take place and the original value is kept untouched:

$$F(u, v) = \begin{cases} \frac{G(u,v)}{H(u,v)} & \text{if } |H(u, v)| \geq T \\ G(u, v) & \text{otherwise} \end{cases} \tag{12.44}$$

■ EXAMPLE 12.8

Figure 12.11 shows an example of image restoration using inverse filtering. Part (a) shows the input (blurry) image. Part (b) shows the result of naively applying inverse filtering (equation (12.42)), which is completely unacceptable due to the division by very small values of $H(u, v)$. Parts (c) and (d) show the results of applying a 10th-order Butterworth low-pass filter to the division, with different cutoff frequencies. Parts (e) and (f) show the results of using constrained division, with different values for the threshold T.

Motion deblurring can be considered a special case of inverse filtering. Figure 12.12 shows the results of applying inverse filtering with constrained division to a blurry image (generated using `fspecial('motion',10,0)` to simulate a horizontal displacement equivalent to 10 pixels).

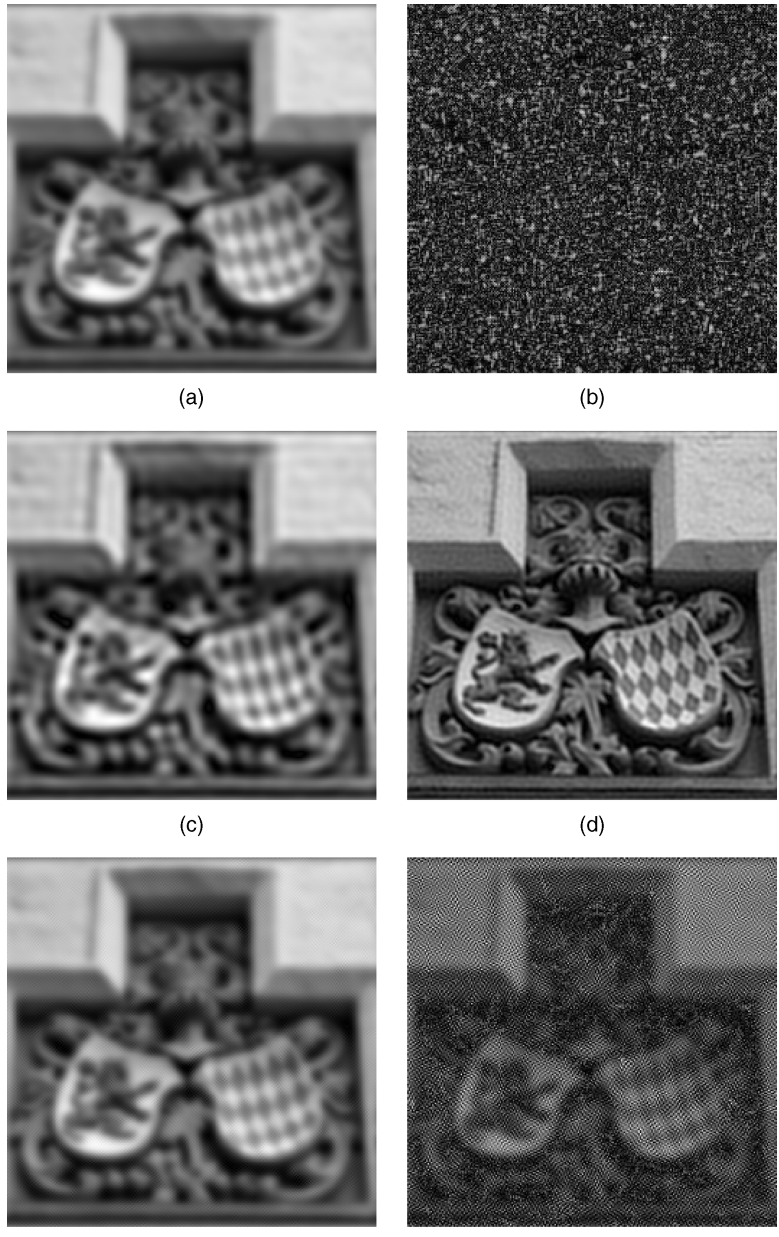

FIGURE 12.11 Example of image restoration using inverse filtering: (a) input (blurry) image; (b) result of naive inverse filtering; (c) applying a 10th-order Butterworth low-pass filter with cutoff frequency of 20 to the division; (d) same as (c), but with cutoff frequency of 50; (e) results of using constrained division, with threshold $T = 0.01$; (f) same as (e), but with threshold $T = 0.001$.

(a) (b)

FIGURE 12.12 Example of motion deblurring using inverse filtering: (a) input image; (b) result of applying inverse filtering with constrained division and threshold $T = 0.05$: the motion blurred has been removed at the expense of the appearance of vertical artifacts.

12.5.1 Wiener Filtering

The Wiener filter (developed by Norbert Wiener in 1942) is an image restoration solution that can be applied to images that have been subject to a degradation function *and* also contain noise, which corresponds to the worst-case scenario for the degraded image $g(x, y)$ in Figure 12.1.

The design of the Wiener filter is guided by an attempt to model the error in the restored image through statistical methods, particularly the *minimum mean square estimator*: once the error is modeled, the average error is mathematically minimized. Assuming a degraded version $g(x, y)$ of some original image $f(x, y)$, and a restored version $r(x, y)$, if we compute the sum of the squared differences between each and every pixel in $f(x, y)$ and the corresponding pixel in $r(x, y)$, we have a figure of merit that captures how well the restoration algorithm worked: smaller values mean better results.

The transfer function of a Wiener filter is given by

$$R(u, v) = \left[\frac{1}{H(u, v)} \frac{|H(u, v)|^2}{|H(u, v)|^2 + K} \right] G(u, v) \tag{12.45}$$

where $H(u, v)$ is the degradation function and K is a constant used to approximate the amount of noise. When $K = 0$, equation (12.45) reduces to equation (12.42).

In MATLAB

The IPT has a function that implements image deblurring using Wiener filter: `deconvwnr`.

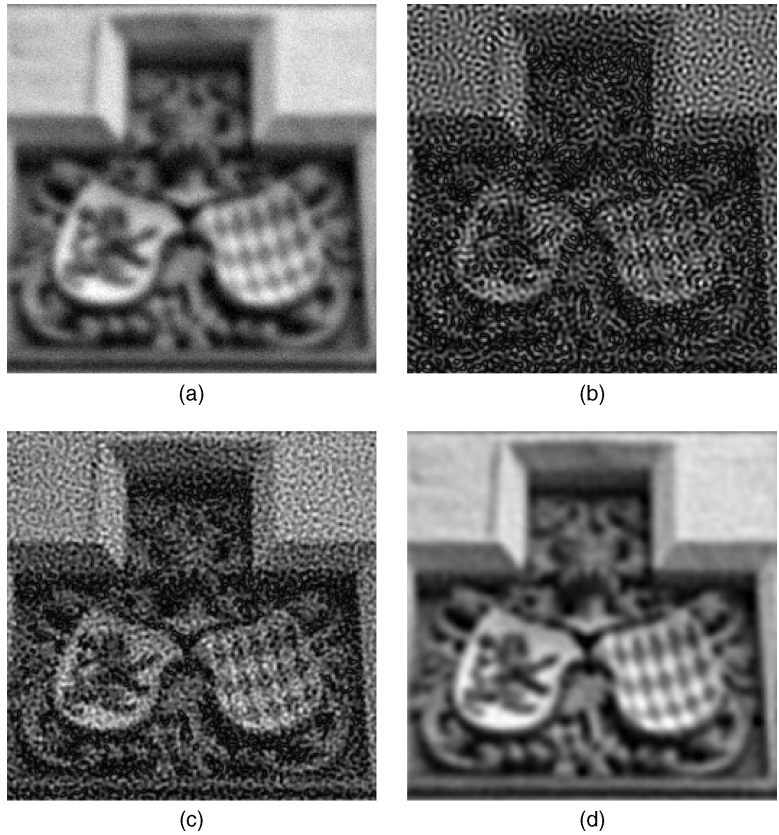

FIGURE 12.13 Example of image restoration using Wiener filtering: (a) input image (blurry and noisy); (b) result of inverse filtering, applying a 10th-order Butterworth low-pass filter with cutoff frequency of 50 to the division; (c) results of Wiener filter, with $K = 10^{-3}$; (d) same as (c), but with $K = 0.1$.

■ EXAMPLE 12.9

Figure 12.13 shows an example of image restoration using Wiener filtering. Part (a) shows the input image, which has been degraded by blur and noise. Part (b) shows the results of applying inverse filtering, using a 10th-order Butterworth low-pass filter and limiting the restoration cutoff frequency to 50. Part (c) shows the results of using Wiener filter with different values of K, to illustrate the trade-off between noise reduction (which improves for higher values of K) and deblurring (which is best for lower values of K).

The Wiener filter can also be used to restore images in the presence of blurring only (i.e., without noise), as shown in Figure 12.14. In such cases, the best results are obtained for lower values of K (Figure 12.14c), but the resulting image is still not as crisp as the one obtained with the inverse filter (Figure 12.14b).

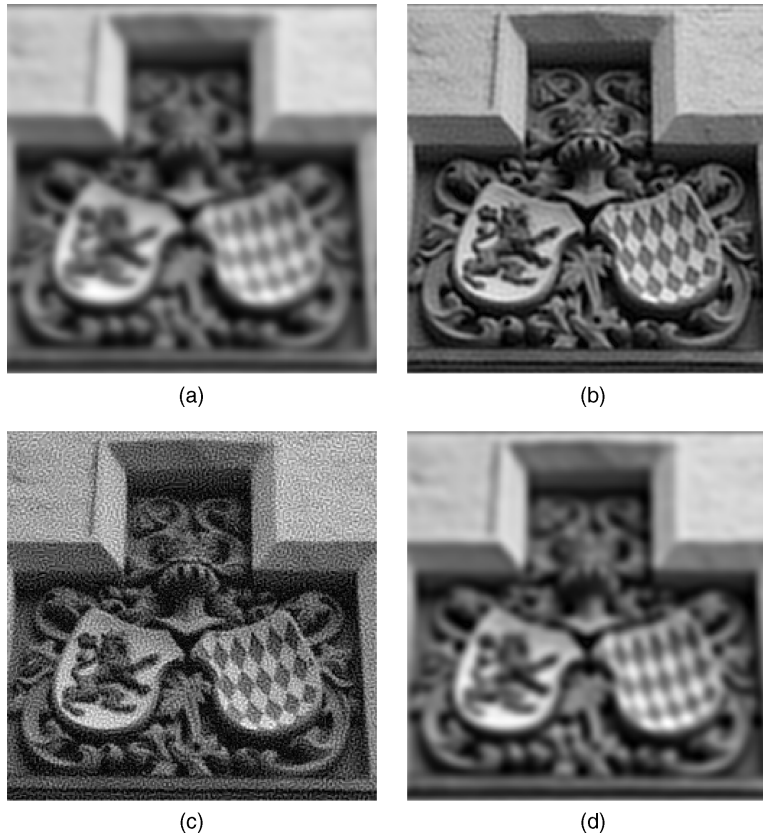

FIGURE 12.14 Example of image restoration using Wiener filtering: (a) input (blurry) image; (b) result of inverse filtering, applying a 10th-order Butterworth low-pass filter with cutoff frequency of 50 to the division; (c) results of Wiener filter, with $K = 10^{-5}$; (d) same as (c), but with $K = 0.1$.

In MATLAB

The IPT has several other built-in functions for image deblurring, namely,

- `deconvreg`: deblur image using regularized filter
- `deconvlucy`: deblur image using Lucy–Richardson method
- `deconvblind`: deblur image using blind deconvolution

The complexity of these functions (in terms of both their mathematical formulation and the number of parameters that need to be adjusted for better performance) is beyond the scope of this book. The reader should refer to the corresponding MATLAB documentation and demos for additional information.

12.6 TUTORIAL 12.1: NOISE REDUCTION USING SPATIAL-DOMAIN TECHNIQUES

Goal

The goal of this tutorial is to learn how to perform noise reduction using spatial-domain techniques.

Objectives

- Learn how to implement the arithmetic mean filter, as well as some of its variations, such as the contraharmonic mean, the harmonic mean, and the geometric mean filters.
- Learn how to perform order statistic filtering, including median, min, max, midpoint, and alpha-trimmed mean filters.

What You Will Need

- `atmean.m`
- `geometric.m`
- `harmonic.m`
- `c_harmonic.m`

Procedure

Arithmetic Mean Filter

The *arithmetic mean filter*, also known as an *averaging* or *low-pass* (from its frequency-domain equivalent) filter, is a simple process of replacing each pixel value with the average of an $N \times N$ window surrounding the pixel. The averaging filter can be implemented as a convolution mask. As in previous tutorials, we will use the function `fspecial` to generate the averaging convolution mask.

1. Load the `eight` image, add (Gaussian) noise to it, and display the image before and after adding noise.

```
I = imread('eight.tif');
In = imnoise(I,'gaussian',0,0.001);
figure, subplot(2,2,1), imshow(I), title('Original Image');
subplot(2,2,2), imshow(In), title('Noisy Image');
```

2. Apply an averaging filter to the image using the default kernel size (3×3).

```
f1 = fspecial('average');
I_blur1 = imfilter(In,f1);
```

```
subplot(2,2,3), imshow(I_blur1), ...
    title('Averaging with default kernel size');
```

Question 1 What is the general effect of the arithmetic mean filter?

3. Implement an averaging kernel with a 5×5 mask.

```
f2 = fspecial('average',[5 5]);
I_blur2 = imfilter(In,f2);
subplot(2,2,4), imshow(I_blur2), ...
    title('Averaging with 5x5 kernel');
```

Question 2 How does the size of the kernel affect the resulting image?

Contraharmonic Mean Filter

The *contraharmonic mean filter* is used for filtering an image with either salt or pepper noise (but not both). When choosing a value for `r`, it is important to remember that negative values are used for salt noise and positive values are used for pepper noise. As we will see, using the wrong sign will give undesired results. This filter does not have a convolution mask equivalent, so we must implement it as a sliding neighborhood operation using the `nlfilter` function. This function allows us to define how we want to operate on the window, which can be specified within a function of our own.

4. Close any open figures.
5. Load two noisy versions of the eight image: one with salt noise and the other with pepper. Also, display the original image along with the two affected images.

```
I_salt = im2double(imread('eight_salt.tif'));
I_pepper = im2double(imread('eight_pepper.tif'));
figure
subplot(2,3,1), imshow(I), title('Original Image');
subplot(2,3,2), imshow(I_salt), title('Salt Noise');
subplot(2,3,3), imshow(I_pepper), title('Pepper Noise');
```

The contraharmonic function requires that images be of class `double`. This is why we convert the image when loading it.

6. Filter the salt noise affected image using -1 for the value of `r`.

```
I_fix1 = nlfilter(I_salt,[3 3],@c_harmonic,-1);
subplot(2,3,5), imshow(I_fix1), title('Salt Removed, r = -1');
```

Our function `c_harmonic` takes two parameters: the current window matrix and a value for `r`. The window matrix, which gets stored into variable `x`, is passed implicitly by the `nlfilter` function. Note how we specified the `c_harmonic` function as the third parameter of the `nlfilter` function call. When using the `nlfilter` function, if you want to pass any additional parameters to your function, you can specify those parameters after the function handle (labeled with a '`@`' in front of it). Notice above how we specified the value of `r` directly after the function handle.

7. Filter the pepper noise affected image using 1 for the value of `r`.

```
I_fix2 = nlfilter(I_pepper,[3 3],@c_harmonic,1);
subplot(2,3,6), imshow(I_fix2), title('Pepper Removed, r = 1');
```

As mentioned previously, using the wrong sign for the value of `r` can lead to unwanted results.

8. Filter the pepper noise image using the wrong sign for `r`.

```
I_bad = nlfilter(I_pepper,[3 3],@c_harmonic,-1);
subplot(2,3,4), imshow(I_bad), title('Using wrong sign for r');
```

Question 3 What is the effect of using the wrong sign when filtering with the contraharmonic mean filter?

Harmonic Mean Filter

The *harmonic mean filter* is another variation of the mean filter and is good for salt and Gaussian noise. It fails, however, when used on pepper noise.

9. Close any open figures.
10. Filter the salt noise affected image with the harmonic filter.

```
I_fix4 = nlfilter(I_salt,[3 3],@harmonic);
figure
subplot(2,3,1), imshow(I), title('Original Image');
subplot(2,3,2), imshow(I_salt), title('Salt Noise');
subplot(2,3,3), imshow(I_pepper), title('Pepper Noise');
subplot(2,3,5), imshow(I_fix4), title('Harmonic Filtered (salt)');
```

11. Filter the pepper noise image and display the result.

```
I_bad2 = nlfilter(I_pepper,[3 3],@harmonic);
subplot(2,3,6), imshow(I_bad2), title('Harmonic Filtered (pepper)');
```

Question 4 Why does the harmonic mean filter fail for images with pepper noise?

12. Try to filter the `In` image (`I` with additive Gaussian noise) with the harmonic mean filter. The image must be converted to `double` first.

```
In_d = im2double(In);
I_fix5 = nlfilter(In_d,[3 3],@harmonic);
figure
subplot(1,3,1), imshow(I), title('Original Image');
subplot(1,3,2), imshow(In_d), title('Image w/ Gaussian Noise');
subplot(1,3,3), imshow(I_fix5), title('Filtered w/ Harmonic Mean');
```

Question 5 How does the size of the window affect the output image?

Geometric Mean Filter

The last variation of the mean filters we will look at is the *geometric mean filter*. This filter is known to preserve image detail better than the arithmetic mean filter and works best on Gaussian noise.

13. Close any open figures.
14. Perform a geometric mean filter on the `eight` image with Gaussian noise (currently loaded in the variable `In_d`).

```
I_fix6 = nlfilter(In_d,[3 3],@geometric);
figure
subplot(1,3,1), imshow(I), title('Original Image');
subplot(1,3,2), imshow(In_d), title('Gaussian Noise');
subplot(1,3,3), imshow(I_fix6), title('Geometric Mean Filtered');
```

Question 6 Filter the salt and pepper noise images with the geometric mean filter. How does the filter perform?

Order Statistic Filters

The *median filter* is the most popular example of an *order statistic filter*. This filter simply sorts all values within a window, finds the median value, and replaces the original pixel value with the median value. It is commonly used for salt and pepper noise. Because of its popularity, the median filter has its own function (`medfilt2`) provided by the IPT.

15. Close any open figures and clear all workspace variables.
16. Load the `coins` image and apply salt and pepper noise.

```
I = imread('coins.png');
I_snp = imnoise(I,'salt & pepper');
figure
subplot(1,3,1), imshow(I), title('Original Image');
subplot(1,3,2), imshow(I_snp), title('Salt & Pepper Noise');
```

17. Filter the image using the `medfilt2` function.

```
I_filt = medfilt2(I_snp,[3 3]);
subplot(1,3,3), imshow(I_filt), title('Filtered Image');
```

Question 7 How does the size of the window affect the output image?

18. Apply the filter to an image with Gaussian noise.

```
I_g = imnoise(I,'gaussian');
I_filt2 = medfilt2(I_g,[3 3]);
figure
subplot(1,3,1), imshow(I), title('Original Image');
subplot(1,3,2), imshow(I_g), title('Gaussian Noise');
subplot(1,3,3), imshow(I_filt2), title('Filtered');
```

Question 8 Why do you think the median filter works on salt and pepper noise but not Gaussian noise?

A quick way to get rid of salt noise in an image is to use the *min filter*, which simply takes the minimum value of a window when the values are ordered. Recall that we previously used the `imfilter` function when dealing with convolution masks and `nlfilter` for sliding neighborhood operations that could not be implemented as a convolution mask. Similarly, the `ordfilt2` function is used for order statistic operations.

19. Close any open figures and clear all workspace variables.
20. Use the `ordfilt2` function to implement a min filter on an image with salt noise.

```
I_s = imread('eight_salt.tif');
I2 = ordfilt2(I_s, 1, ones(3,3));
figure
subplot(1,2,1), imshow(I_s), title('Salt Noise');
subplot(1,2,2), imshow(I2), title('Min Filter');
```

Question 9 Why would this filter not work on pepper noise?

The first parameter specified in the `ordfilt2` function is the image we wish to filter. The second parameter specifies the index of the value to be useed after all

values in the window have been ordered. Here, we specified this parameter as 1, which means we want the first value after reordering, that is, the minimum value. The last parameter defines the size of the window as well as which values in that window will be used in the ordering. A 3×3 matrix of 1's would indicate a 3×3 window and to use all values when ordering. If we instead specified a 3×3 matrix where only the first row was 1's and the last two rows were zeros, then the sliding window would consist of a 3×3 matrix, but only the top three values would be considered when ordering. In addition, keep in mind that even though we used a special function to implement the median filter, it is still an order statistic filter, which means we could have implemented it using the `ordfilt2` function.

Question 10 Implement the median filter using the `ordfilt2` function.

The *max filter* is used for filtering pepper noise, similar to the technique of the min filter.

21. Filter a pepper noise affected image with the max filter.

```
I_p = imread('eight_pepper.tif');
I3 = ordfilt2(I_p, 9, ones(3,3));
figure
subplot(1,2,1), imshow(I_p), title('Pepper Noise');
subplot(1,2,2), imshow(I3), title('Max Filter');
```

Although the *midpoint filter* is considered an order statistic filter, it cannot be directly implemented using the `ordfilt2` function because we are not selecting a particular element from the window, but instead performing a calculation on its values—namely, the minimum and maximum values. Rather, we will implement it using the familiar `nlfilter` function. This noise removal technique is best used on Gaussian or uniform noise.

22. Filter an image contaminated with Gaussian noise using the midpoint filter.

```
I = imread('coins.png');
I_g = imnoise(I,'gaussian',0,0.001);
midpoint = inline('0.5 * (max(x(:)) + min(x(:)))');
I_filt = nlfilter(I_g,[3 3],midpoint);
figure
subplot(1,2,1), imshow(I_g), title('Gaussian Noise');
subplot(1,2,2), imshow(I_filt), title('Midpoint Filter');
```

You may have noticed that we have used an inline function instead of creating a separate function, as we did in previous steps. Inline functions are good for quick tests, but—as you may have realized—they are much slower than regular functions.

Alpha-Trimmed Mean Filters

The *alpha-trimmed mean filter* is basically an averaging filter whose outlying values are removed before averaging. To do this, we sort the values in the window, discard elements on both ends, and then take the average of the remaining values. This has been defined in the function `atmean`.

23. Close any open figures and clear all workspace variables.
24. Generate a noisy image with Gaussian noise and salt and pepper noise.

```
I = imread('cameraman.tif');
Id = im2double(I);
In = imnoise(Id,'salt & pepper');
In2 = imnoise(In,'gaussian');
```

25. Filter the image using the alpha-trimmed mean filter.

```
I_filt = nlfilter(In2,[5 5],@atmean,6);
figure
subplot(1,3,1), imshow(I), title('Original Image');
subplot(1,3,2), imshow(In2), title('S&P and Gaussian Noise');
subplot(1,3,3), imshow(I_filt), title('Alpha Trimmed Mean');
```

Question 11 When filtering an image with both types of noise, how does the alpha-trimmed mean filter compare to the arithmetic mean filter?

WHAT HAVE WE LEARNED?

- In the context of image processing, *noise* is a general term used to express deviations from a pixel's expected (or *true*) value. When these deviations are offsets from the true value, the noise is said to be of *additive* type. When the true value is rescaled as a result of noise, such noise is said to be of *multiplicative* type.
- The statistical properties of noise are usually modeled in a way that is independent of the actual causes of the noise. Common probability distribution functions associated with noise are Gaussian, exponential, uniform, and gamma (Erlang).
- Two of the most common types of noise in image processing are the Gaussian noise and the *salt and pepper* noise. Both are forms of additive noise. The Gaussian noise follows a zero-mean normal distribution. The salt and pepper noise is a type of impulsive noise that appears as black and white specks on the image.

- The most common noise removal techniques in the spatial domain are the mean filter, the median filter, and variants and combinations of them.
- The most common noise removal techniques in the frequency domain are low-pass, bandpass, bandreject, and notch filters.
- *Blurring* is the loss of sharpness in an image. It can be caused by poor focusing, relative motion between sensor and scene, and noise, among other factors.
- Deblurring techniques typically consist of applying inverse filtering techniques with the goal of "undoing" the degradation. The best-known approach to image deblurring (even in the presence of noise) is the Wiener filter, which is implemented in MATLAB by the `deconvwnr` function.

LEARN MORE ABOUT IT

- Chapter 4.5 of [Bov00a] provides additional information on noise sources and noise models.
- Chapters 11 and 12 of [Pra07] discuss image restoration models and techniques in more detail.
- Section 5.2 of [GWE04] discusses the generation of spatial random noise with specified distributions and extends the functionality of the IPT function `imnoise`.
- There have been many variants, extensions, and optimized implementations of the median filter proposed in the literature, such as
  - the pseudomedian [PCK85], also described in Section 10.3 of [Pra07];
  - weighted median filters, described in Chapter 3.2 of [Bov00a];
  - faster implementations, such as the one proposed in [HYT79].
- Nonlinear filters have been the subject of entire chapters, for example, [Dou94], and book-length treatment, for example, [PV90].
- To learn more about adaptive filters, we recommend Section 5.3.3 of [GW08] and Chapter 11 of [MW93].
- Sections 5.7–5.10 of [GWE04] discuss the main IPT functions for image deblurring.
- Chapter 4.2 of [SOS00] presents a technique for noise removal in binary images, the *k*Fill filter.
- Chapter 3.5 of [Bov00a] provides additional information on image restoration, image deblurring, and blur identification techniques.

12.7 PROBLEMS

12.1 Write a modified and expanded version of the IPT function `imnoise`. Your function should allow the specification of other types of noise, currently not supported by `imnoise`, for example, Rayleigh, Erlang, uniform, and exponential.

12.2 Write a MATLAB function to generate periodic noise (and its spectrum), given a set of coordinate pairs corresponding to different values of frequencies (u, v) in the 2D spatial frequency domain. Use your function to generate the same periodic noise as displayed in Figure 12.9 (where the value of D_0 is 32).

12.3 What is the effect on an image if we apply a contraharmonic mean filter with the following values of R:

(a) $R = 0$

(b) $R = -1$

12.4 Write a MATLAB function to implement an ideal bandreject filter of radius D_0 and width W and use it to reduce periodic noise on an input noisy image.

12.5 Write a MATLAB function to implement a Butterworth bandreject filter of order n, radius D_0, and width W and use it to reduce periodic noise on an input noisy image.

12.6 Write a MATLAB function to implement a Gaussian bandreject filter of radius D_0 and width W and use it to reduce periodic noise on an input noisy image.

12.7 Test your solutions to Problems 12.4–12.6 using the image from Figure 12.9a (available at the book web site) as an input.

12.8 Write a MATLAB function to implement an ideal bandpass filter of radius D_0 and width W.

12.9 Write a MATLAB function to implement a Butterworth bandpass filter of order n, radius D_0, and width W.

12.10 Write a MATLAB function to implement a Gaussian bandpass filter of radius D_0 and width W.

12.11 Test your solutions to Problems 12.8–12.10 using the image from Figure 12.9a (available at the book web site) as an input and comparing the result produced by your function with the image in Figure 12.9b.

12.12 Write a MATLAB function to implement an ideal notch reject filter of radius D_0, centered at (u_0, v_0), and use it to reduce periodic noise on an input noisy image.

12.13 Write a MATLAB function to implement a Butterworth notch reject filter of order n and radius D_0, centered at (u_0, v_0), and use it to reduce periodic noise on an input noisy image.

12.14 Write a MATLAB function to implement a Gaussian notch reject filter of radius D_0, centered at (u_0, v_0), and use it to reduce periodic noise on an input noisy image.

12.15 Test your solutions to Problems 12.12–12.14 using a test image with one sinusoidal noise component created with your solution to Problem 12.2.

12.16 Write a MATLAB function to implement a *range filter* whose output is the difference between the maximum and the minimum gray levels in a neighborhood centered on a pixel [Eff00], test it, and answer the following questions:

(a) Is the range filter a linear or nonlinear one? Explain.

(b) What can it be used for?