ECEN 642, Fall 2019

Texas A&M University

Electrical and Computer Engineering Department

Dr. Raffaella Righetti

Due: 11/15/2019 (before class)

Assignment #5

Gonzalez and Woods (4th edition), projects: 5.1, 5.2, 5.3, 5.6, 5.7, 5.8

Gonzalez and Woods (4th edition), problems: 5.20, 5.35

For project 5.2, you CAN use the MATLAB function "rand" and "randn".

For problem 5.35, you can directly start from results obtained from 4.51 without redoing it

Please clearly indicate your processing parameters (if there is any) in your solutions

For the assignments, you will need to use MATLAB. You can access it on campus through the open access lab (OAL) or remotely through the virtual open access lab (VOAL). The link below will guide you into configuring the Horizon client for VOAL remote connection step by step on your PC or mac:

https://tamu.service-now.com/tamu-selfservice/knowledge_detail.do?sysparm_document_key=kb_knowledge,6f7e0c6adbce5f84778ff5961d96199f#

After you connect to the server, you will see the MATLAB icon. If you don't, you can connect to your VOAL desktop (VOAL icon) and start MATLAB from there.

For projects from the 4th edition, photo copies of the project statements are provided. For projects from the 3rd edition, you can access them via the link: http://www.imageprocessingplace.com/DIP-3E/dip3e_student_projects.htm#02-04

5.46 Show that the Radon transform [Eq. (5-102)] of the Gaussian shape $f(x,y) = A\exp(-x^2 - y^2)$ is given by $g(\rho,\theta) = A\sqrt{\pi}\exp(-\rho^2)$. (*Hint:* Refer to Example 5.15, where we used symmetry to simplify integration.)

5.47 Do the following:

(a)* Show that the Radon transform [Eq. (5-102)] of the unit impulse $\delta(x,y)$ is a straight vertical line passing through the origin of the $\rho\theta$-plane .

(b) Show that the radon transform of the impulse $\delta(x - x_0, y - y_0)$ is a sinusoidal curve in the $\rho\theta$-plane.

5.48 Prove the validity of the following properties of the Radon transform [Eq. (5-102)]:

(a)* *Linearity:* The Radon transform is a linear operator. (See Section 2.6 regarding linearity.)

(b) *Translation property:* The radon transform of $f(x - x_0, y - y_0)$ is $g(\rho - x_0\cos\theta - y_0\sin\theta, \theta)$.

(c)* *Convolution property:* The Radon transform of the convolution of two functions is equal to the convolution of the Radon transforms of the two functions.

5.49 Provide the steps that lead from Eq. (5-113) to Eq. (5-114). [*Hint:* $G(\omega, \theta + 180°) = G(-\omega, \theta)$.]

5.50* Prove the validity of Eq. (5-125).

5.51 Prove the validity of Eq. (5-127).

Projects

MATLAB solutions to the projects marked with an asterisk (*) are in the DIP4E Student Support Package (consult the book website: www.ImageProcessingPlace.com).

5.1 Read the image **book-cover-gaussian.tif**. You are told that this image has been corrupted by additive Gaussian noise. Find estimates of the mean and standard deviation of the noise. (*Hint:* Take a look at Fig. 5.6 and review project function **centralMoments4e**.)

5.2 Obtain functions for generating 2-D arrays of uniform and Gaussian random numbers in the language you are using for your projects. If you are using MATLAB, functions **rand** and **randn** are the noise generators of choice for this purpose. Read the image **testpattern512.tif**, scale it to the range [0, 1] using the default mode of project function **intScaling4e**, and do the following.

(a)* Fix the mean at 0.25 and add three levels of Gaussian noise to the image by varying the standard deviation. The three levels should be such that the noise appears (1) *mild* (you can tell the noise is there, but it is barely perceivable; (2) *intermediate* (the noise is definitely present, but all the image features are still clearly visible); and (3) *heavy* (the noise is objectionable, causing some of the image features to be obscured by the noise); (4) *extra heavy* (the noise dominates the image; most of the smaller and light features in the image are obscured by noise). For comparisons of your results to be meaningful, you should scale the image to the full range [0, 1], using the **'full'** and **'floating'** options in function **intScaling4e**. Show all four results and list the values of standard deviation you used. Explain why image details begin to disappear as the noise level increases significantly.

(b) Repeat the four levels of noise outlined in (a) using uniform noise. Note that the parameters to specify are a and b in Eq. (5-13). Try to make your images appear as close as possible to their Gaussian counterparts. Explain why image details begin to disappear as the noise level increases significantly. Note any significant differences between corresponding images in (a) and (b).

(c) You will find that the images in (b) have higher contrast than their Gaussian counterparts in (a). Explain why.

5.3 Working with salt-and-pepper noise.

(a)* Explain how you can modify a generator of uniform random numbers to produce salt-and-pepper noise with specified probabilities P_s for salt pixels and P_p for pepper pixels.

Assume that you will be working with images of size $M \times N$ whose intensities are in the range [0,1], so that salt pixels will have value 1 and pepper pixels value 0.

(b) Use your analysis from (a) and the uniform random number generator from Problem 5.2 as the basis for writing a function **g = saltPepper4e(f,ps,pp)** to add salt-and-pepper noise to image **f**. Parameters **ps** and **pp** are the probabilities mentioned in (a).

(c)* Read the image **testpattern512.tif** and repeat the four levels of noise outlined in Problem 5.2(a), using equal values for **ps** and **pp**. Show your images and values used.

(d)* Repeat (c) using only pepper noise.

(e) Repeat (c) using only salt noise.

5.4 This project deals with extending the linear spatial filtering concepts introduced in Chapter 3. The objective is to use project function **twodConv4e** to implement the mean filters discussed in Section 5.3. Although some of those filters (e.g. the geometric mean filter) perform nonlinear operations on the pixels of a neighborhood, it is possible to convert the nonlinear operations to a form that allows linear filtering (i.e., sum-of-products operations) using spatial convolution. The solution to (b) shows how do to this. In the following, **g** is a noisy input image, **f_hat** is the filtered (estimate) image, and $m \times n$ is the size of the neighborhood that defines the filter size.

(a)* Write a function **f_hat = aMean4e(g,m,n)** that implements the arithmetic mean filter defined in Eq. (5-23).

(b)* Write a function **f_hat = geoMean4e(g,m,n)** that implements the geometric mean filter defined in Eq. (5-24).

(c) Write a function **f_hat = harMean4e(g,m,n)** that implements the harmonic mean filter defined in Eq. (5-25).

(d) Write a function **f_hat = ctharMean4e(g,m,n,q)** that implements the contraharmonic mean filter in Eq. (5-26). (*Hint:* Because you will performing a ratio computation of two filtered images, you should disable the autoscaling in project function **twodConv4e**, and do the scaling at the end using project function **intScaling4e**.)

(e)* Read the image **circuitboard-gaussian.tif** and use function **aMean4e** to duplicate the result in Fig. 5.7(c).

(f) Read the image **circuitboard-gaussian.tif** and use function **geoMean4e** to duplicate the result in Fig. 5.7(d).

(g) Read the image **circuitboard-pepper.tif** and use function **ctharMean4e** to duplicate the result in Fig. 5.8(c).

(h) Read the image **circuitboard-salt.tif** and use function **ctharMean4e** to duplicate the result in Fig. 5.8(d).

5.5 The objective of this project is to implement nonlinear (order-statistic) spatial filters. You have to implement a function in the language you are using, capable of performing sliding neighborhood operations. The concept is the same as convolution, except that, instead of performing a sum-of-products (linear) operation on every neighborhood, the function you need now must be capable of performing neighborhood operations that in general are nonlinear, and are specifiable by you. This capability is required to implement some of the filters discussed in Section 5.3. If you are using MATLAB and have the MATLAB Image Processing Toolbox installed in your system, you can use function **nlfilter**. If you do not have the Image Processing Toolbox installed, you can use utility function **mynlfilter**, which is included in the utilities folder of your *DIP4E Student Support Package*. See the solution to (a) below for details on the characteristics of the required sliding neighborhood function.

(a)* Implement a general sliding neighborhood filtering function capable of accepting user-defined operations, and performing these operations on a neighborhood of size $m \times n$.

(b)* Use your function from (a) as the basis for writing a function **f_hat = minFilter4e(g,m,n)** that implements a min filter of size $m \times n$.

(c) Use your function from (a) as the basis for writing a function **f_hat = maxFilter4e(g,m,n)** that implements a max filter of size $m \times n$.

(d) Use your function from (a) as the basis of a function **f_hat = medianFilter4e(g,m,n)** that implements a median filter of size $m \times n$. (*Hint:* If you are using MATLAB, start with function **median**, but be aware that this function only works with 1-D arrays, so your neighborhoods will have to be converted to 1-D.)

(e) Read the image **hubble.tif** and determine the minimum size of a square neighborhood needed to generate *in one pass* an image in which only part of the largest object remains. (*Hint:* Use either a max or a min filter—you have to determine which.)

(f) Read the image **circuitboard-saltandpep.tif** and use your function from (d) to duplicate the results in Figs. 5.10(b)-(d).

5.6 Experimenting with notch filters.

(a)* Write a function **H = notchReject4e(type,M,N, param,C)** for implementing an $M \times N$ notch reject filter transfer function of specified **type**: **'ideal'**, **'gaussian'**, **'butterworth'**, or **'rectangular'**. Array **param** is a list of the parameters needed to implement the chosen filter **type**, and **C** contains specified locations for the notches. Only one notch location is required. The function should generate the symmetric notch automatically. Your function need only be capable of generating one notch reject transfer function each time it is called.

(b)* Read the image **astronaut-interference.tif** and use notch filtering in the frequency domain to remove the sinusoidal interference. Do the filtering using project function **dftFiltering4e** without padding. Display the original image, the spectrum, the filter transfer function you used, the processed image, and the interference pattern. The processed image should look like Fig. 5.16(d). (*Hint:* Compute the spectrum and magnify the area near the center of the spectrum so you can see the location of the energy bursts caused by the periodic interference. Then use a very narrow Gaussian notch reject filter transfer function with centers at those locations.)

(c)* Repeat (b) using zero padding and explain the principal reason for the difference between your results. Your explanation will reveal why we did not use image padding in the image restoration techniques discussed in this chapter. (*Hint:* Consider using images as aids in your explanation.)

(d) Read the image **cassini-interference.tif** and generate results close in appearance to those shown in Figs. 4.65 and 4.66.

5.7 Blurring transfer function.

(a)* Write a function **H = motionBlurTF4e(P,Q,a,b,T)** that implements a $P \times Q$ blurring transfer function, as given in Eq. (5-77). The parameters **a**, **b**, and **T** are as defined for that equation. Transfer function **H** must be centered on the frequency rectangle.

(b) Display the spectrum of **H**.

(c) Read the image **boy.tif** and blur it with the same parameters used to generate the image in Fig. 5.26(b). Display your results.

5.8 Parametric Wiener filter.

(a)* Write a function **W = pWienerTF4e(H,K)** that implements the parametric Wiener filter transfer function given in Eq. (5-85), where **H** is a degradation transfer function, and **K** is a scalar parameter.

(b)* Read the image **boy-blurred.tif** and restore it using the degradation function from Eq. (5-77), *without* padding. (*Hint:* the image **boy-blurred.tif** was created using the degradation function in Eq. (5-77) with $T = 1.0$, but the other parameters differ by a sign from those used in Fig. 5.26. Also, the value of K needed for proper restoration is in the range 10^{-3} to 10^{-4}.)

(c) Read the image **boy.tif** (this is the unblurred image) and explain why your restored image is not quite as good as the original.

(d)* Repeat (b), but this time first pad image **boy-blurred.tif** with zero padding of size $P = 2M$ and $Q = 2N$ using the **'post'** option in project function **imPad4e**. (Note that the estimate of the blurring function will now be of size $2M \times 2N$.) You will get an unexpected result.

(e)* Read the image **boy.tif**, pad it with zeros to size $2M \times 2N$ *first,* and then blur it using project function **motionBlurtf4e** with parameters **a** = 0.1, **b** =−0.1, and $T = 1.0$. Repeat

(b) and display your result after cropping the image to its original $M \times N$ size. You will find that this approach works as expected.

(f)* Explain why (d) and (e) gave such different results. Your explanation will reveal why we did not use image padding in the image restoration techniques discussed in this chapter.

(g)* Show how you would go about blurring the original **boy.tif** image, and then restoring it so that the result is identical to the original.

(h)* In order to focus on the effects of noise on restoration, we worked with the original (complex) blurred image in Examples 5.11 and 5.12. That is the reason the results in the low-noise case were so close to the original. In order to see what the results would have been if we had started instead with a version of the low-noise blurred image that was converted to tif for archival, repeat (b) using image **book-cover-blurred.tif**. The blurring parameters in the estimate of H are $a = b = 0.1$, and $T = 1.0$. Use the same K as in (b).

5.9 Constrained least squares filter.

(a) Write a function **W = constrainedLsTF4e(H,gam)** that implements the constrained least squares filter transfer function in Eq. (5-89), where **H** is a degradation transfer function and **gam** is a scalar parameter.

(b) Read the image **boy-blurred.tif** and restore it using the degradation function from Eq. (5-77). (*Hint:* The image **boy-blurred.tif** was created using this degradation function with $T = 1.0$, but the other parameters differ by a sign from those used in Fig. 5.26. Also, the value of **gam** needed for proper restoration is in the range 10^{-3} and 10^{-4}.)

(c) Read the image **boy.tif** (this was the original before blurring) and explain why your restored image is not quite as good as the original.

5.10 Image reconstruction from projections.

(a) Write a function **g = imRecon4e(f,theta)** to implement image reconstruction from projections. Base your function on the intuitive approach explained in connection with Figs. 5.32–5.34. Here, **f** is an input image and **theta** is a 1-D array of angles. Your function must be capable of handling a single angle. Output **g** is the image reconstructed from projections of **f** at the angles specified in **theta**. Your function should scale the output using project function **intScaling4e** with the **'full'** option. (*Note:* For accuracy in reconstruction, this function requires rotation with a function capable of bilinear interpolation or better. If you are using MATLAB and the Image Processing Toolbox, the function to use is **imrotate**.)

(b)* Read the image **binary-objects.tif** and obtain its reconstruction for a single angle, **theta** = 0°.

(c) Repeat (b) for **theta** = 45°.

(d) Repeat (b) for **theta** = [0°, 45°, 90°].

(e) Repeat (b) for **theta** from 0° to 179° in increments of 1°.

5.20

A linear, space invariant system has the impulse response

$$h(x, y) = \delta(x - a, y - b)$$

where a and b are constants, and x and y are discrete quantities. Answer the following, assuming negligible noise in each case.

(a)* What is the system transfer function in the frequency domain?

(b)* What would the spatial domain system response be to a constant input, $f(x, y) = K$?

(c) What would the spatial domain system response be to an impulse input, $f(x, y) = \delta(x, y)$?

5.35 Given $p(x, y)$ in Eq. (5-90), show that

$$P(u, v) = 4 - 2\cos(2\pi u/M) - 2\cos(2\pi v/N)$$

(*Hint:* Study the solution to Problem 4.51.)

Figures & formulas

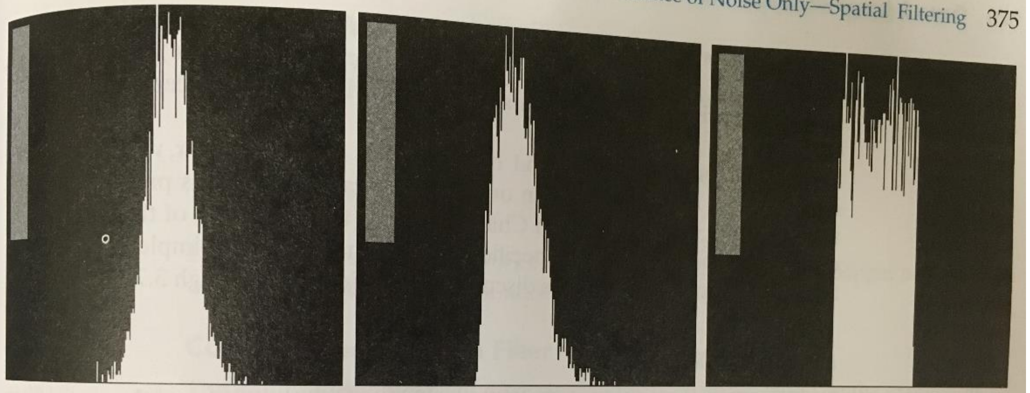

a b c

FIGURE 5.6 Histograms computed using small strips (shown as inserts) from (a) the Gaussian, (b) the Rayleigh, and (c) the uniform noisy images in Fig. 5.4.

Uniform Noise

The PDF of *uniform* noise is

$$p(z) = \begin{cases} \dfrac{1}{b-a} & a \le z \le b \\ 0 & \text{otherwise} \end{cases} \tag{5-13}$$

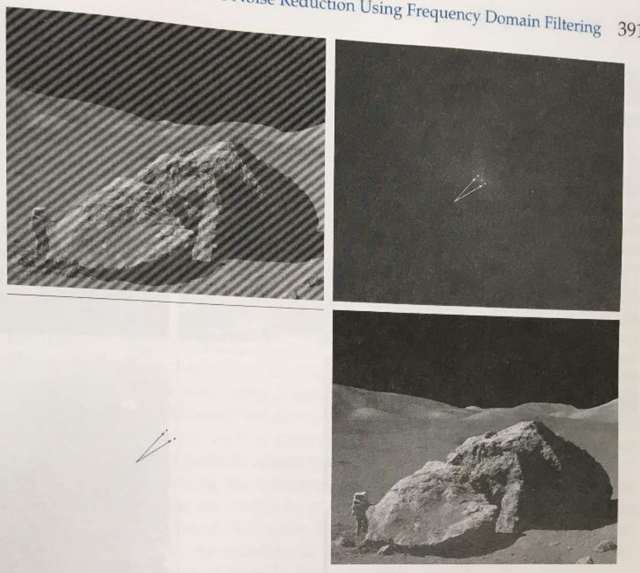

a b
c d

FIGURE 5.16 (a) Image corrupted by sinusoidal interference. (b) Spectrum showing the bursts of energy caused by the interference. (The bursts were enlarged for display purposes.) (c) Notch filter (the radius of the circles is 2 pixels) used to eliminate the energy bursts. (The thin borders are not part of the data.) (d) Result of notch reject filtering. (Original image courtesy of NASA.)

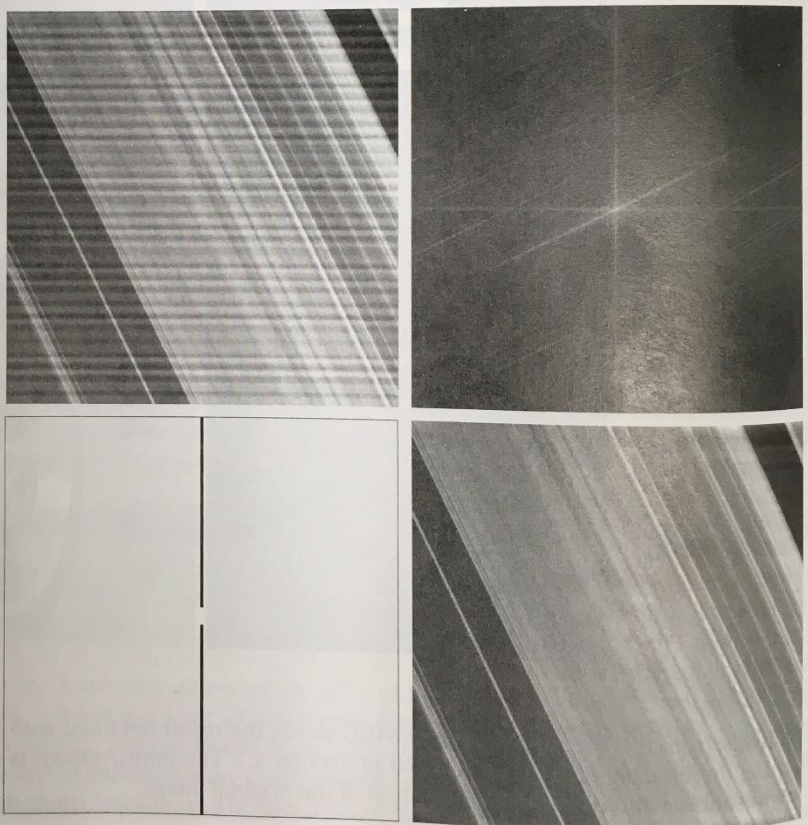
which, as you know from Chapter 4, are responsible for the intensity differences between smooth areas. Figure 5.18(c) shows the filter transfer function we used, and Fig. 5.18(d) shows the filtered result. Most of the fine scan lines were eliminated or significantly attenuated. In order to get an image of the noise pattern, we proceed as before by converting the reject filter into a pass filter, and then filtering the input image with it. Figure 5.19 shows the result.

EXAMPLE 4.25: Using notch filtering to remove periodic interference.

Figure 4.65(a) shows an image of part of the rings surrounding the planet Saturn. This image was captured by *Cassini*, the first spacecraft to enter the planet's orbit. The nearly sinusoidal pattern visible in the image was caused by an AC signal superimposed on the camera video signal just prior to digitizing the image. This was an unexpected problem that corrupted some images from the mission. Fortunately, this type of interference is fairly easy to correct by postprocessing. One approach is to use notch filtering.

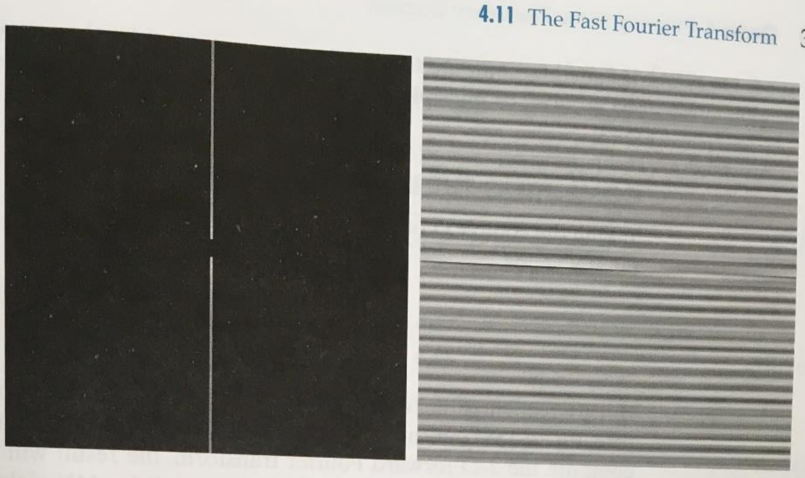
Figure 4.65(b) shows the DFT spectrum. Careful analysis of the vertical axis reveals a series of small bursts of energy near the origin which correspond to the nearly sinusoidal interference. A simple approach is to use a narrow notch rectangle filter starting with the lowest frequency burst, and extending for the remainder of the vertical axis. Figure 4.65(c) shows the transfer function of such a filter (white represents 1 and black 0). Figure 4.65(d) shows the result of processing the corrupted image with this filter. This result is a significant improvement over the original image.

To obtain and image of just the interference pattern, we isolated the frequencies in the vertical axis using a notch pass transfer function, obtained by subtracting the notch reject function from 1 [see Fig. 4.66(a)]. Then, as Fig. 4.66(b) shows, the IDFT of the filtered image is the spatial interference pattern.



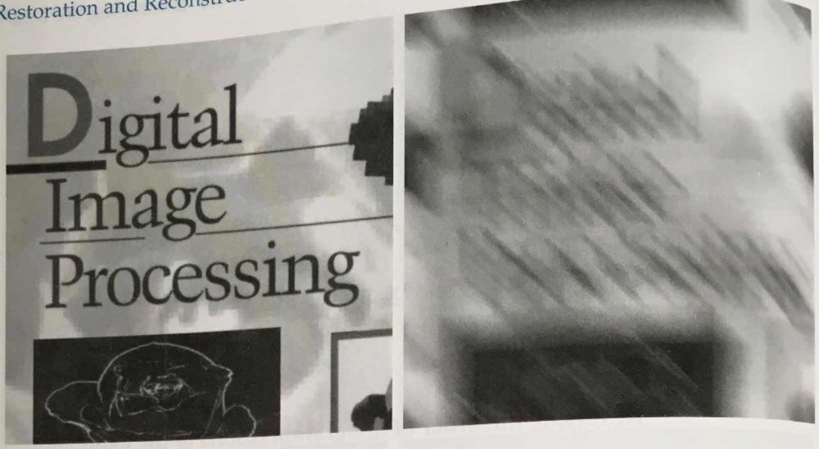
a b
c d

FIGURE 4.65 (a) Image of Saturn rings showing nearly periodic interference. (b) Spectrum. (The bursts of energy in the vertical axis near the origin correspond to the interference pattern). (c) A vertical notch reject filter transfer function. (d) Result of filtering. (The thin black border in (c) is not part of the data.) (Original image courtesy of Dr. Robert A. West, NASA/JPL.)



a b

FIGURE 4.66 (a) Notch pass filter function used to isolate the vertical axis of the DFT of Fig. 4.65(a). (b) Spatial pattern obtained by computing the IDFT of (a).



a b

FIGURE 5.26 (a) Original image. (b) Result of blurring using the function in Eq. (5-77) with $a = b = 0.1$ and $T = 1$.

If we allow the y-component to vary as well, with the motion given by $y_0(t) = bt/T$, then the degradation function becomes

$$H(u,v) = \frac{T}{\pi(ua + vb)} \sin\left[\pi(ua + vb)\right] e^{-j\pi(ua + vb)} \tag{5-77}$$

To generate a discrete filter transfer function of size $M \times N$, we sample this equation for $u = 0, 1, 2, \ldots, M-1$ and $v = 0, 1, 2, \ldots, N-1$.

A number of useful measures are based on the power spectra of noise and of the undegraded image. One of the most important is the *signal-to-noise ratio*, approximated using frequency domain quantities such as

$$\text{SNR} = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1}|F(u,v)|^2 \Big/ \sum_{u=0}^{M-1}\sum_{v=0}^{N-1}|N(u,v)|^2 \qquad (5\text{-}82)$$

This ratio gives a measure of the level of information-bearing signal power (i.e., of the original, undegraded image) to the level of noise power. An image with low noise would tend to have a high SNR and, conversely, the same image with a higher level of noise would have a lower SNR. This ratio is an important measure used in characterizing the performance of restoration algorithms.

The *mean square error* given in statistical form in Eq. (5-80) can be approximated also in terms of a summation involving the original and restored images:

$$\text{MSE} = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left[f(x,y) - \hat{f}(x,y)\right]^2 \qquad (5\text{-}83)$$

In fact, if one considers the restored image to be "signal" and the difference between this image and the original to be "noise," we can define a signal-to-noise ratio in the spatial domain as

$$\text{SNR} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\hat{f}(x,y)^2 \Big/ \sum_{x=0}^{M-1}\sum_{y=0}^{M-1}\left[f(x,y) - \hat{f}(x,y)\right]^2 \qquad (5\text{-}84)$$

The closer f and $\hat{f}$ are, the larger this ratio will be. Sometimes the square root of the preceding two measures is used instead, in which case they are referred to as the *root-mean-square-error* and the *root-mean-square-signal-to-noise ratio*, respectively. As we have mentioned before, keep in mind that quantitative measures do not necessarily relate well to perceived image quality.

When dealing with white noise, the spectrum is a constant, which simplifies things considerably. However, the power spectrum of the undegraded image seldom is known. An approach frequently used when these quantities are not known, or cannot be estimated, is to approximate Eq. (5-81) by the expression

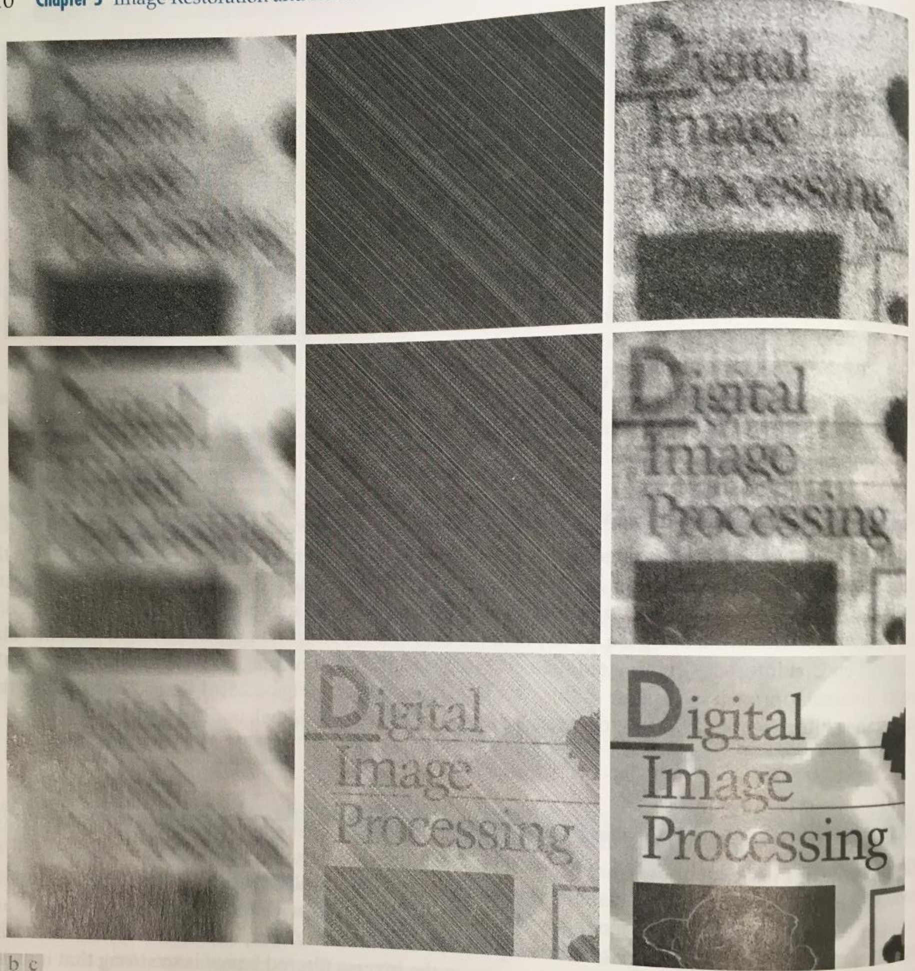
$$\hat{F}(u,v) = \left[\frac{1}{H(u,v)}\frac{|H(u,v)|^2}{|H(u,v)|^2 + K}\right]G(u,v) \qquad (5\text{-}85)$$

where K is a specified constant that is added to all terms of $|H(u,v)|^2$. The following examples illustrate the use of this expression.

EXAMPLE 5.11: More deblurring examples using Wiener filtering.

The first row of Fig. 5.29 shows, from left to right, the blurred image of Fig. 5.26(b) heavily corrupted by additive Gaussian noise of zero mean and variance of 650; the result of direct inverse filtering; and the result of Wiener filtering. The Wiener filter of Eq. (5-85) was used, with $H(u,v)$ from Example 5.8, and with K chosen interactively to give the best possible visual result. As expected, direct inverse filtering produced an unusable image. Note that the noise in the inverse filtered image is so strong that it masks completely the content of the image. The Wiener filter result is by no means perfect, but it does give us a hint as to image content. The text can be read with moderate effort.

The second row of Fig. 5.29 shows the same sequence just discussed, but with the level of the noise variance reduced by one order of magnitude. This reduction had little effect on the inverse filter, but the Wiener results are considerably improved. For example, the text is much easier to read now. In the third row of Fig. 5.29, the noise variance was reduced more than five orders of magnitude from the first row. In fact, image in Fig. 5.29(g) has no visible noise. The inverse filter result is interesting in this case. The noise is still quite visible, but the text can be seen through a "curtain" of noise (see Problem 5.32). The Wiener filter result in Fig. 5.29(i) is excellent, being quite close visually to the original image in Fig. 5.26(a). In



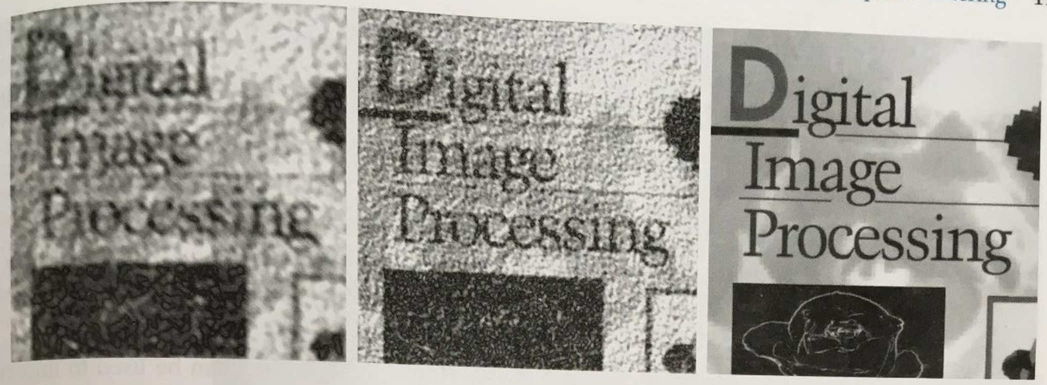
a b c
d e f
g h i

FIGURE 5.29 (a) 8-bit image corrupted by motion blur and additive noise. (b) Result of inverse filtering. (c) Result of Wiener filtering. (d)–(f) Same sequence, but with noise variance one order of magnitude less. (g)–(i) Same sequence, but noise variance reduced by five orders of magnitude from (a). Note in (h) how the deblurred image is quite visible through a "curtain" of noise.

practice, the results of restoration filtering are seldom this close to the original images, as you will learn in Projects 5.8 and 5.9. This example, and Example 5.12 in the next section, were idealized slightly to focus on the effects of noise on restoration algorithms.

EXAMPLE 5.12: Comparison of deblurring by Wiener and constrained least squares filtering.

Figure 5.30 shows the result of processing Figs. 5.29(a), (d), and (g) with constrained least squares filters, in which the values of γ were selected manually to yield the best visual results. This is the same procedure we used to generate the Wiener filter results in Fig. 5.29(c), (f), and (i). By comparing the constrained least squares and Wiener results, we see that the former yielded better results (especially in terms of noise reduction) for the high- and medium-noise cases, with both filters generating essentially



a b c

FIGURE 5.30 Results of constrained least squares filtering. Compare (a), (b), and (c) with the Wiener filtering results in Figs. 5.29(c), (f), and (i), respectively.

equal results for the low-noise case. This is not surprising because parameter γ in Eq. (5-89) is a true scalar, whereas the value of K in Eq. (5-85) is a scalar approximation to the ratio of two unknown frequency domain *functions* of size $M \times N$. Thus, it stands to reason that a result based on manually selecting γ would be a more accurate estimate of the undegraded image. As in Example 5.11, the results in this example are better than one normally finds in practice. Our focus here was on the effects of noise blurring on restoration. In Projects 5.8 and 5.9, you will encounter situations in which the restoration solutions are not quite as close to the original images as we have shown in these two examples.