ECE438 - Laboratory 10: Image Processing (Week 1)

October 6, 2010

1 Introduction

This is the first part of a two week experiment in image processing. During this week, we will cover the fundamentals of digital monochrome images, intensity histograms, pointwise transformations, gamma correction, and image enhancement based on filtering.

In the **second week** , we will cover some fundamental concepts of *color* images. This will include a brief description on how humans perceive color, followed by descriptions of two standard *color spaces*. The second week will also discuss an application known as image *halftoning*.

2 Introduction to Monochrome Images

An *image* is the optical representation of objects illuminated by a light source. Since we want to process images using a computer, we represent them as functions of discrete spatial variables. For *monochrome* (black-and-white) images, a scalar function $f(i,j)$ can be used to represent the light *intensity* at each spatial coordinate (i,j). Figure 1 illustrates the convention we will use for spatial coordinates to represent images.

If we assume the coordinates to be a set of positive integers, for example $i = 1, \ldots, M$ and $j = 1, \ldots, N$, then an image can be conveniently represented by a matrix.

$$f(i,j) = \begin{bmatrix} f(1,1) & f(1,2) & \cdots & f(1,N) \\ f(2,1) & f(2,2) & \cdots & f(2,N) \\ \vdots & \vdots & & \vdots \\ f(M,1) & f(M,2) & \cdots & f(M,N) \end{bmatrix} \tag{1}$$

We call this an $M \times N$ image, and the elements of the matrix are known as *pixels.*

The pixels in digital images usually take on integer values in the finite range,

$$0 \leq f(i,j) \leq L_{max} \tag{2}$$

Questions or comments concerning this laboratory should be directed to Prof. Charles A. Bouman, School of Electrical and Computer Engineering, Purdue University, West Lafayette IN 47907; (765) 494-0340; bouman@ecn.purdue.edu

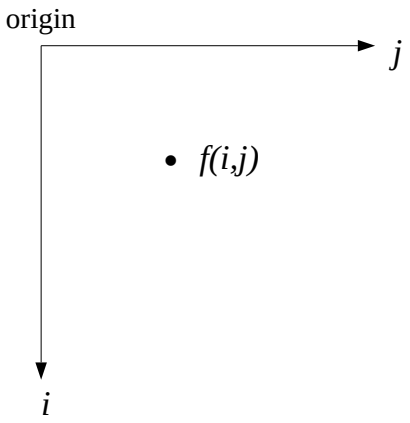


Figure 1: Spatial coordinates used in digital image representation.

where 0 represents the minimum intensity level (black), and L_{max} is the maximum intensity level (white) that the digital image can take on. The interval $[0, L_{max}]$ is known as a *gray scale*.

In this lab, we will concentrate on 8-bit images, meaning that each pixel is represented by a single byte. Since a byte can take on 256 distinct values, L_{max} is 255 for an 8-bit image.

2.1 Exercise

Download **yacht.tif**
Help on **image command**

In order to process images within Matlab, we need to first understand their numerical representation. Download the image file **yacht.tif** . This is an 8-bit monochrome image. Read it into a matrix using `A = imread('yacht.tif');`.

Type `whos` to display your variables. Notice under the "Class" column that the A matrix elements are of type *uint8* (unsigned integer, 8 bits). This means that Matlab is using a single byte to represent each pixel. Matlab cannot perform numerical computation on numbers of type *uint8*, so we usually need to convert the matrix to a floating point representation. Create a double precision representation of the image using `B = double(A);`. Again, type `whos` and notice the difference in the number of bytes between A and B. In future sections, we will be performing computations on our images, so we need to remember to convert them to type *double* before processing them.

Display *yacht.tif* using the following sequence of commands:

```
image(B);
colormap(gray(256));
axis('image');
```

The *image* command works for both type *uint8* and *double* images. The *colormap* command

specifies the range of displayed gray levels, assigning black to 0 and white to 255. It is important to note that if any pixel values are outside the range 0 to 255 (after processing), they will be clipped to 0 or 255 respectively in the displayed image. It is also important to note that a floating point pixel value will be rounded down ("floored") to an integer before it is displayed. Therefore the maximum number of gray levels that will be displayed on the monitor is 255, even if the image values take on a continuous range.

Now we will practice some simple operations on the *yacht.tif* image. Make a horizontally flipped version of the image by reversing the order of each column. Similarly, create a vertically flipped image. Print your results.

Now, create a "negative" of the image by subtracting each pixel from 255 (here's an example of where conversion to *double* is necessary.) Print the result.

Finally, multiply each pixel of the original image by 1.5, and print the result.

INLAB REPORT:

1. Hand in two flipped images.
2. Hand in the negative image.
3. Hand in the image multiplied by factor of 1.5. What effect did this have?

3 Pixel Distributions

Download **house.tif**
Download **narrow.tif**

3.1 Histogram of an Image

The *histogram* of a digital image shows how its pixel intensities are distributed. The pixel intensities vary along the horizontal axis, and the number of pixels at each intensity is plotted vertically, usually as a bar graph. A typical histogram of an 8-bit image is shown in Fig. 2.

Write a simple Matlab function `Hist(A)` which will plot the histogram of image matrix A. You may use Matlab's *hist* function, however that function requires a vector as input. An example of using *hist* to plot a histogram of a matrix would be

```
x=reshape(A,1,M*N);
hist(x,0:255);
```

where A is an image, and M and N are the number of rows and columns in A. The *reshape* command is creating a row vector out of the image matrix, and the *hist* command plots a histogram with bins centered at $[0:255]$.

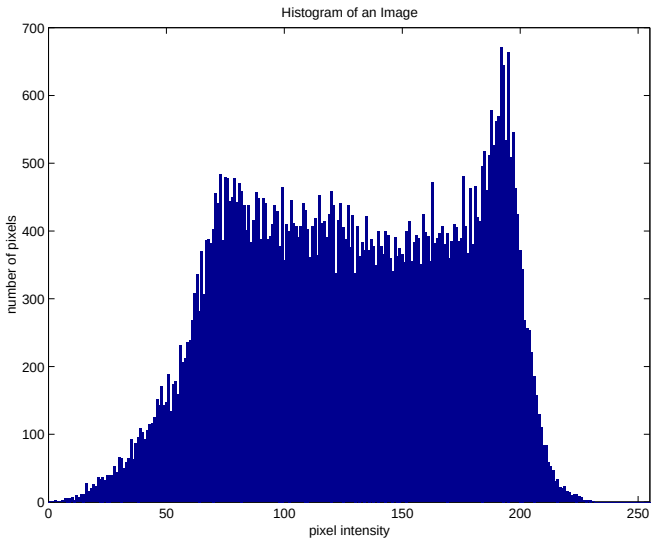


Figure 2: Histogram of an 8-bit image

Download the image file **house.tif** , and read it into Matlab. Test your *Hist* function on the image. Label the axes of the histogram and give it a title.

INLAB REPORT:
Hand in your labeled histogram. Comment on the distribution of the pixel intensities.

3.2 Pointwise Transformations

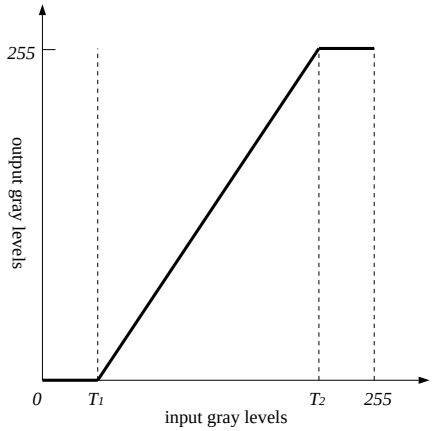


Figure 3: Pointwise transformation of image

A pointwise transformation is a function that maps pixels from one intensity to another. An example is shown in Fig. 3. The horizontal axis shows all possible intensities of the original image, and the vertical axis shows the intensities of the transformed image. This particular transformation maps the "darker" pixels in the range $[0, T_1]$ to a level of zero

(black), and similarly maps the "lighter" pixels in $[T_2, 255]$ to white. Then the pixels in the range $[T_1, T_2]$ are "stretched out" to use the full scale of $[0, 255]$. This can have the effect of increasing the contrast in an image.

Pointwise transformations will obviously affect the pixel distribution, hence they will change the shape of the histogram. If a pixel transformation can be described by a one-to-one function, $y = f(x)$, then it can be shown that the input and output histograms are approximately related by the following:

$$H_{out}(y) \approx \left. \frac{H_{in}(x)}{|f'(x)|} \right|_{x=f^{-1}(y)} . \tag{3}$$

Since x and y need to be integers in (3), the evaluation of $x = f^{-1}(y)$ needs to be rounded to the nearest integer.

The pixel transformation shown in Fig. 3 is not a one-to-one function. However, equation (3) still may be used to give insight into the effect of the transformation. Since the regions $[0, T_1]$ and $[T_2, 255]$ map to the single points 0 and 255, we might expect "spikes" at the points 0 and 255 in the output histogram. The region $[1, 254]$ of the output histogram will be directly related to the input histogram through equation (3).

First, notice from $x = f^{-1}(y)$ that the region $[1, 254]$ of the output is being mapped from the region $[T_1, T_2]$ of the input. Then notice that $f'(x)$ will be a constant scaling factor throughout the entire region of interest. Therefore, the output histogram should approximately be a stretched and rescaled version of the input histogram, with possible spikes at the endpoints.

Write a Matlab function that will perform the pixel transformation shown in Fig. 3. It should have the syntax

```
output = pointTrans(input, T1, T2) .
```

Hints:

- Determine an equation for the graph in Fig. 3, and use this in your function. Notice you have three input regions to consider. You may want to create a separate function to apply this equation.
- If your function performs the transformation one pixel at a time, be sure to allocate the space for the output image at the beginning to speed things up.

Download the image file **narrow.tif** and read it into Matlab. Display the image, and compute its histogram. The reason the image appears "washed out" is that it has a narrow histogram. Print out this picture and its histogram.

Now use your *pointTrans* function to spread out the histogram using $T1 = 70$ and $T2 = 180$. Display the new image and its histogram. (You can open another figure window using the *figure* command.) Do you notice a difference in the "quality" of the picture?

INLAB REPORT:

1. Hand in your code for *pointTrans*.
2. Hand in the original image and its histogram.
3. Hand in the transformed image and its histogram.
4. What qualitative effect did the transformation have on the original image? Do you observe any negative effects of the transformation?
5. Compare the histograms of the original and transformed images. Why are there zeros in the output histogram?

4 Gamma(γ) Correction

Download **dark.tif**

The light intensity generated by a physical device is usually a nonlinear function of the original signal. For example, a pixel that has a gray level of 200 will not be twice as bright as a pixel with a level of 100. Almost all computer monitors have a *power law* response to their applied voltage. For a typical cathode ray tube (CRT), the brightness of the illuminated phosphors is approximately equal to the applied voltage raised to a power of 2.5. The numerical value of this exponent is known as the *gamma* (γ) of the CRT. Therefore the power law is expressed as

$$I = V^{\gamma} \tag{4}$$

where I is the pixel intensity and V is the voltage applied to the device.

If we relate equation (4) to the pixel values for an 8-bit image, we get the following relationship,

$$y = 255\left(\frac{x}{255}\right)^{\gamma} \tag{5}$$

where x is the original pixel value, and y is the pixel intensity as it appears on the display. This relationship is illustrated in Figure 4.

In order to achieve the correct reproduction of intensity, this nonlinearity must be compensated by a process known as γ *correction*. Images that are not properly corrected usually appear too light or too dark. If the value of γ is available, then the correction process consists of applying the inverse of equation (5). This is a straightforward pixel transformation, as we discussed in Section 3.2.

Write a Matlab function that will γ correct an image by applying the inverse of equation (5). The syntax should be

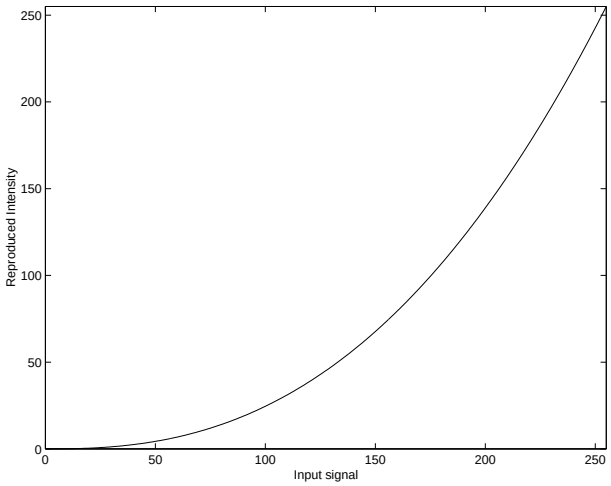


Figure 4: Nonlinear behavior of a display device having a γ of 2.2.

```
B = gammCorr(A,gamma)
```

where A is the uncorrected image, *gamma* is the γ of the device, and B is the corrected image. (See the hints in Section 3.2.)

The file **dark.tif** is an image that has not been γ corrected for your monitor. Download this image, and read it into Matlab. Display it and observe the quality of the image.

Assume that the γ for your monitor is 2.2. Use your *gammCorr* function to correct the image for your monitor, and display the resultant image. Did it improve the quality of the picture?

INLAB REPORT:

1. Hand in your code for *gammCorr*.
2. Hand in the γ corrected image.
3. How did the correction affect the image? Does this appear to be the correct value for γ ?

5 Image Enhancement Based on Filtering

Sometimes, we need to process images to improve their appearance. In this section, we will discuss two fundamental image enhancement techniques: image *smoothing* and *sharpening*.

5.1 Image Smoothing

Smoothing operations are used primarily for diminishing spurious effects that may be present in a digital image, possibly as a result of a poor sampling system or a noisy trans-

mission channel. Lowpass filtering is a popular technique of image smoothing.

Some filters can be represented as a 2-D convolution of an image $f(i,j)$ with the filter's impulse response $h(i,j)$.

$$\begin{aligned} g(i,j) &= f(i,j) ** h(i,j) \\ &= \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f(k,l)h(i-k,j-l) \end{aligned} \tag{6}$$

Some typical lowpass filter impulse responses are shown in Fig. 5, where the center element corresponds to $h(0,0)$. Notice that the terms of each filter sum to one. This prevents amplification of the DC component of the original image. The frequency response of each of these filters is shown in Fig. 6.

$\frac{1}{9}\bullet$

1	1	1
1	1	1
1	1	1

(a)

$\frac{1}{10}\bullet$

1	1	1
1	2	1
1	1	1

(b)

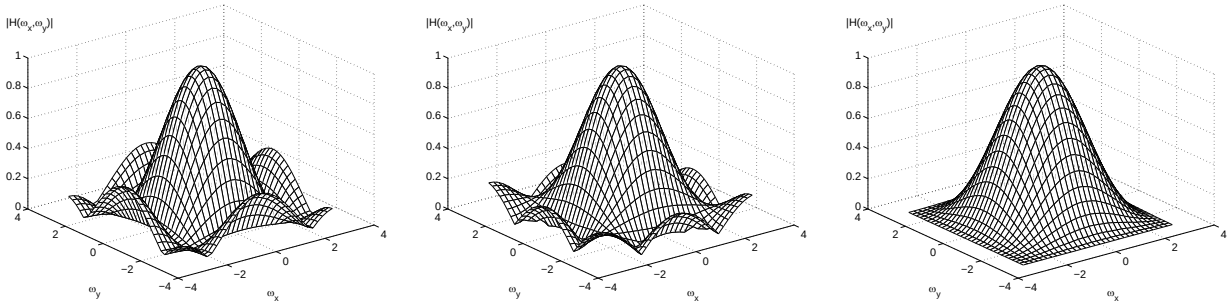
$\frac{1}{16}\bullet$

1	2	1
2	4	2
1	2	1

(c)

Figure 5: Impulse responses of lowpass filters useful for image smoothing.

An example of image smoothing is shown in Fig. 7, where the degraded image is processed by the filter shown in Fig. 5(c). It can be seen that lowpass filtering clearly reduces the additive noise, but at the same time it *blurs* the image. Hence, blurring is a major limitation of lowpass filtering.



(a) (b) (c)

Figure 6: Frequency responses of the lowpass filters shown in Fig. 5

In addition to the above linear filtering techniques, images can be smoothed by *nonlinear* filtering, such as mathematical morphological processing. *Median* filtering is one of the simplest morphological techniques, and is useful in the reduction of impulsive noise. The main advantage of this type of filter is that it can reduce noise while preserving the detail of the original image. In a median filter, each input pixel is replaced by the median of the pixels contained in a surrounding window. This can be expressed by

$$g(i,j) = median\{f(i-k,j-l)\},\ (k,l) \in W \tag{7}$$

where W is a suitably chosen window. Figure 8 shows the performance of the median filter in reducing so-called "salt and pepper" noise.

(a)

(b)

(c)

Figure 7: (a) Original gray scale image. (b) Original image degraded by additive white Gaussian noise, $N(0, 0.01)$. (c) Result of processing the degraded image with a lowpass filter.

5.2 Smoothing Exercise

Download **race.tif**
Download **noise1.tif**
Download **noise2.tif**
Help on **mesh command**

Among the many spatial lowpass filters, the Gaussian filter is of particular importance. This is because it results in very good spatial and spectral localization characteristics. The Gaussian filter has the form

$$h(i,j) = C \ \exp(-\frac{i^2+j^2}{2\sigma^2}) \tag{8}$$

where σ^2, known as the *variance*, determines the size of passband area. Usually the Gaussian filter is normalized by a scaling constant C such that the sum of the filter coefficient

(a) (b)

Figure 8: (a) Image degraded by "salt and pepper" noise with 0.05 noise density. (b) Result of 3×3 median filtering.

magnitudes is one, allowing the average intensity of the image to be preserved.

$$\sum_{i,j} h(i,j) = 1$$

Write a Matlab function that will create a *normalized* Gaussian filter that is centered around the origin (the center element of your matrix should be $h(0,0)$). Note that this filter is both *separable* and *symmetric*, meaning $h(i,j) = h(i)h(j)$ and $h(i) = h(-i)$. Use the syntax

```
h=gaussFilter(N, var)
```

where `N` determines the size of filter, `var` is the variance, and `h` is the $N \times N$ filter. Notice that for this filter to be symmetrically centered around zero, N will need to be an odd number.

Use Matlab to compute the frequency response of a 7×7 Gaussian filter with $\sigma^2 = 1$. Use the command

```
H = fftshift(fft2(h,32,32));
```

to get a 32×32 DFT. Plot the magnitude of the frequency response of the Gaussian filter, $|H_{Gauss}(\omega_1, \omega_2)|$, using the *mesh* command. Plot it over the region $[-\pi, \pi] \times [-\pi, \pi]$, and label the axes.

Filter the image contained in the file **race.tif** with a 7×7 Gaussian filter, with $\sigma^2 = 1$. **Hint:** You can filter the signal by using the Matlab command

```
Y = filter2(h, X);
```

where `X` is the matrix containing the input image and `h` is the impulse response of the filter. Display the original and the filtered images, and notice the blurring that the filter has caused.

Now write a Matlab function to implement a 3×3 median filter (without using the *medfilt2* command). Use the syntax

```
Y = medianFilter(X);
```

where `X` and `Y` are the input and output image matrices, respectively. For convenience, you do not have to alter the pixels on the border of X.

Hint: Use the Matlab command *median* to find the median value of a subarea of the image,

i.e. a 3×3 window surrounding each pixel.

Download the image files **noise1.tif** and **noise2.tif** . These images are versions of *race.tif* that have been degraded by additive white Gaussian noise and "salt and pepper" noise, respectively. Read them into Matlab, and display them using *image*. Filter each of the noisy images with both the 7×7 Gaussian filter ($\sigma^2 = 1$) and the 3×3 median filter. Display the results of the filtering, and place a title on each figure. (You can open several figure windows using the *figure* command.) Compare the filtered images with the original noisy images. Print out the four filtered pictures.

INLAB REPORT:

1. Hand in your code for *gaussFilter* and *medianFilter*.
2. Hand in the plot of $|H_{Gauss}(\omega_1, \omega_2)|$.
3. Hand in the results of filtering the noisy images (4 pictures).
4. Discuss the effectiveness of each filter for the case of additive white Gaussian noise. Discuss both positive and negative effects that you observe for each filter.
5. Discuss the effectiveness of each filter for the case of "salt & pepper" noise. Again, discuss both positive and negative effects that you observe for each filter.

5.3 Image Sharpening

Image sharpening techniques are used primarily to enhance an image by highlighting details. Since fine details of an image are the main contributors to its high frequency content, highpass filtering often increases the local contrast and sharpens the image. Some typical highpass filter impulse responses used for contrast enhancement are shown in Fig. 9. The frequency response of each of these filters is shown in Figure 10.

0	1	0
1	-4	1
0	1	0

(a)

1	1	1
1	-8	1
1	1	1

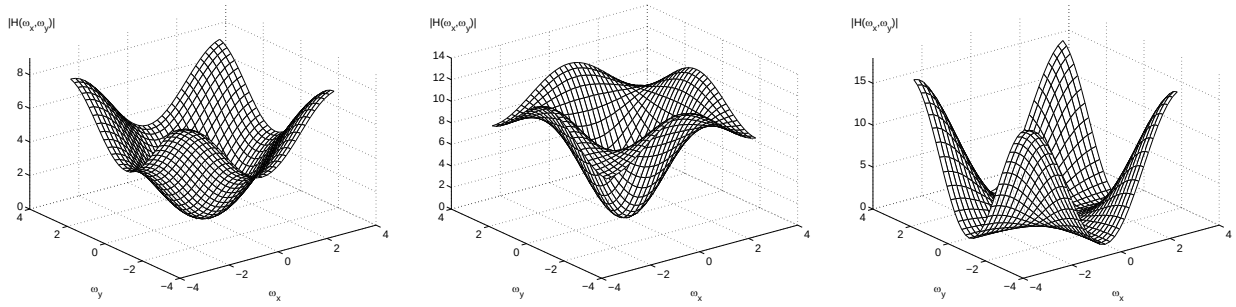
(b)

-1	2	-1
2	-4	2
-1	2	-1

(c)

Figure 9: Impulse responses of highpass filters useful for image sharpening.

An example of highpass filtering is illustrated in Fig. 11. It should be noted from this example that the processed image has enhanced contrast, however it appears more noisy than the original image. Since noise will usually contribute to the high frequency content of an image, highpass filtering has the undesirable effect of accentuating the noise.



(a) (b) (c)

Figure 10: Frequency responses of the highpass filters shown in Fig. 9.

(a) (b)

Figure 11: (a) Original gray scale image. (b) Highpass filtered image.

5.4 Sharpening Exercise

Download **blur.tif**

In this section, we will introduce a sharpening filter known as an *unsharp mask*. This type of filter subtracts out the "unsharp" (low frequency) components of the image, and consequently produces an image with a sharper appearance. Thus, the unsharp mask is closely related to highpass filtering. The process of unsharp masking an image $f(i,j)$ can be expressed by

$$g(i,j) = \alpha f(i,j) - \beta[f(i,j) ** h(i,j)] \tag{9}$$

where $h(i,j)$ is a *lowpass* filter, and α and β are positive constants such that $\alpha - \beta = 1$.

Analytically calculate the frequency response of the unsharp mask filter in terms of α, β, and $h(i,j)$ by finding an expression for

$$\frac{G(\omega_1,\omega_2)}{F(\omega_1,\omega_2)} \ . \tag{10}$$

Using your *gaussFilter* function from Section 5.2, create a 5×5 Gaussian filter with $\sigma^2 = 1$. Use Matlab to compute the frequency response of an unsharp mask filter (use your expression for equation (10)), using the Gaussian filter as $h(i, j)$, $\alpha = 5$ and $\beta = 4$. The size of the calculated frequency response should be 32×32. Plot the magnitude of this response in the range $[-\pi, \pi] \times [-\pi, \pi]$ using *mesh*, and label the axes. You can change the viewing angle of the mesh plot with the *view* command. Print out this response.

Download the image file **blur.tif** and read it into Matlab. Apply the unsharp mask filter with the parameters specified above to this image, using equation (9). Use *image* to view the original and processed images. What effect did the filtering have on the image? Label the processed image and print it out.

Now try applying the filter to *blur.tif*, using $\alpha = 10$ and $\beta = 9$. Compare this result to the previous one. Label the processed image and print it out.

INLAB REPORT:

1. Hand in your derivation for the frequency response of the unsharp mask.
2. Hand in the labeled plot of the magnitude response. Compare this plot to the highpass responses of Fig. 10. In what ways is it similar to these frequency responses?
3. Hand in the two processed images.
4. Describe any positive and negative effects of the filtering that you observe. Discuss the influence of the α and β parameters.