Lab 10: Filtering and Edge Detection of Images

In this lab we re-introduce digital images as a signal type for the application of filtering. The lab has a project on lowpass and highpass filtering of images with an investigation of the frequency content of an image. In addition, the problem of edge detection is introduced, where a combination of linear and nonlinear filters are used to solve the problem.

1 Overview

1.1 Digital Images

An image can be represented as a function $x(t_1, t_2)$ of two continuous variables representing the horizontal (t_2) and vertical (t_1) coordinates of a point in space. For monochrome images, the $x(t_1, t_2)$ would be a scalar function of the two spatial variables, but for color images the function would be a vector function of the two variables.[1] Moving images (TV) would add a time variable to the two spatial variables. Monochrome images are displayed using black and white and shades of gray, so they are called *grayscale* images. In this lab we will consider only sampled grayscale still images.

A sampled grayscale still image would be represented as a two-dimensional array of numbers of the form

$$x[m, n] = x(mT_1, nT_2) \qquad 1 \leq m \leq M, \text{ and } 1 \leq n \leq N$$

Typical values of M and N are on the order of 256 or 512; e.g., a 512×512 image has nearly the same resolution as a standard TV image. In MATLAB we can represent an image as a matrix consisting of M rows and N columns. The matrix entry at (m, n) is the sample value $x[m, n]$—called a *pixel* (short for picture element).

An important property of light images such as photographs and TV pictures is that their values are always non-negative and finite in magnitude; i.e.,

$$0 \leq x[m, n] \leq X_{\max}$$

This is because light images are formed by measuring the intensity of reflected or emitted light which must always be a positive finite quantity. When stored in a computer or displayed on a monitor, the values of $x[m, n]$ are usually scaled so that an eight-bit integer representation can be used. With 8-bit integers, the maximum value can be $X_{\max} = 2^8 - 1 = 255$, and there are 256 gray levels for the display.

1.2 Displaying Images

As you will discover, the correct display of an image on a gray-scale monitor can be tricky, especially after some processing has been performed on the image. We have provided the function `show_img.m` in the *DSP First Toolbox* to handle most of these problems, but it will be helpful if the following points are noted:

1. All image values must be non-negative for the purposes of display. Filtering may introduce negative values, especially if differencing is used e.g., a high-pass filter).

[1] For example, an RGB color system needs three values for red, green and blue at each spatial location.

2. The default format for most gray-scale displays is eight bits, so the pixel values $x[m,n]$ in the image are integers in the range $0 \le x[m,n] \le 255 = 2^8 - 1$.

3. The MATLAB functions `max` and `min` can be used to find the largest and smallest values in the image.

4. The functions `round`, `fix` and `floor` can be used to quantize pixel values to integers.

5. The actual display on the monitor is created with the `show_img` function. The appearance of the image can be altered by running the pixel values through a "color map." In our case, all three primary colors (red, green and blue, or RGB) are used equally, so we get a "gray map." In MATLAB the `gray` color map is created via

   ```
   colormap(gray(256))
   ```

   which gives a 256×3 matrix where all 3 columns are equal. `colormap(gray(256))` creates a linear mapping, so that each input pixel amplitude is rendered with a screen intensity proportional to its value (assuming the monitor is calibrated). For our experiments, non-linear color mappings would introduce an extra level of complication, so we won't use them.

6. When the image values lie outside the range [0,255], or when the image is scaled so that it only occupies a small portion of the range [0,255], the display may have poor quality. We can analyze this condition by using the MATLAB function `hist` to plot how often each pixel value occurs (called a histogram). This will indicate if some values at the extremes are seldom used and, therefore, can be "thrown away."

   Based on the histogram, we can adjust the pixel values prior to display. This can be done in two different ways:

7. *Clipping the image:* When some of the pixel values lie outside the range [0,255], but the scaling needs to be preserved, the pixel values should be clipped. All negative values would be set to zero, and anything above 255 would be set equal to 255. The function `clip` is provided to do this job.

8. *Automatically rescaling the image:* This requires a linear mapping of the pixel values:[2]

   $$x_s[m,n] = \alpha x[m,n] + \beta$$

   The scaling constants α and β can be derived from the min and max values of the image, so that all pixel values are recomputed via:

   $$x_s[m,n] = \left\lfloor 256 \, \frac{x[m,n] - x_{\min}}{x_{\max} - x_{\min}} \right\rfloor$$

   where $\lfloor x \rfloor$ is the floor function, i.e., the greatest integer less than or equal to x.

1.3 Image filtering

It is possible to filter image signals just as it is possible to filter one-dimensional signals. One method of filtering two-dimensional signals (images) is to filter each row with a one-dimensional filter and then filter each of the resulting columns with a one-dimensional filter. This is the approach taken in this lab.

[2] The MATLAB function `show_img` will perform this scaling while making the image display.

2 Warm-up: Display of Images

You can load the images needed for this lab from `*.mat` files. Any file with the extension `*.mat` is in MATLAB format and can be loaded via the `load` command. To find some of these files, look for `*.mat` in the *DSP First* toolbox or in the MATLAB directory called `toolbox/matlab/demos`. Some of the image files are named `lenna.mat`, `echart.mat` and `zone.mat`, but there are others within MATLAB's demos. The default size is 256×256, but alternate versions are available at 512×512 under names such as `lenna512.mat`. After loading, use the `whos` to determine the name of the variable that holds the image and its size.

Although MATLAB has several functions for displaying images on the CRT of the computer, we have written a special function `show_img()` for this lab. It is the visual equivalent of `sound()`, which we used when listening to speech and tones; i.e., `show_img()` is the "D-to-C" converter for images. This function handles the scaling of the image values and allows you to open up multiple image display windows. Here is the `help` on `show_img`:

```
function [ph] = show_img(img, figno, scaled, map)
%SHOW_IMG    display an image with possible scaling
% usage:  ph = show_img(img, figno, scaled, map)
%    img = input image
%    figno = figure number to use for the plot
%              if 0, re-use the same figure
%              if omitted a new figure will be opened
% optional args:
%    scaled = 1 (TRUE) to do auto-scale (DEFAULT)
%           not equal to 1 (FALSE) to inhibit scaling
%    map = user-specified color map
%     ph = figure handle returned to caller
%----
```

Notice that unless the input parameter `figno` is specified, a new figure will be opened. This is extremely important if you want to display several images together in `subplot`. Using `subplot` is the preferred method of *printing* multiple images (e.g., for comparisons in lab reports) because one subplot can easily hold four images.

2.1 Display Test

In order to probe your understanding of image display, generate a simple test image in which all of the rows are identical.

(a) Create a test image that is 256×256, by making each horizontal line a discrete-time sinusoid with a period of 50 samples. Note that the row index is the vertical axis and the column index is the horizontal axis.

(b) Make a plot of one row of the image to verify contents of the image.

(c) Use `show_img` to display the test image, and then explain the gray scale pattern that you observe. If the sinusoid was generated with values between $+1$ and -1 explain the scaling used to make the 8-bit image for screen display. In other words, what is the (integer) gray level for zero? for -0.3?

Instructor Verification (separate page)

(d) Now load and display the `lenna` image from `lenna.mat`. The command `load lenna` will put the sampled image into the array `xx`.

(e) Make a plot of the 200$^{\text{th}}$ row of the `lenna` image. Observe the maximum and minimum values.

(f) Consider a way in which you might display the negative of the `lenna` image by rescaling the pixels. In other words derive a linear mapping to remap the pixel values so that white and black are interchanged.

Instructor Verification (separate page)

3 Lab: Filtering Images

Previously you have experimented with one-dimensional filters, such as running averagers and first-difference filters, applied to one-dimensional signals. You have also applied some filters to images by regarding each row (or column) of the image as a one-dimensional signal.

3.1 One-Dimensional Filtering

For example, the 50$^{\text{th}}$ row of an image is the N-point sequence `x[50,n]` for $1 \leq n \leq N$. We can filter this sequence with a 1-D filter using the `conv` operator, but how can we filter the entire image? That is the objective of this Lab.

(a) Load in the image `echart.mat` with the `load` command. Extract the 33$^{\text{rd}}$ row from the bottom of the image using the statement

```
x1 = echart(256-33,:);
```

Filter this one-dimensional signal with a 7-point averager and plot both the input and the output in the same figure using a two-panel `subplot`. Observe whether or not the filtered waveform is "smoother" or "rougher" than the input. Explain.

(b) Now extract the 99$^{\text{th}}$ column of the image, and filter it with a first difference. Plot the input and output together for comparison. Once again, observe whether or not the filtered waveform is "smoother" or "rougher" than the input. Explain.

3.2 Blurring an Image

From parts (a) and (b) above, perhaps you can see how you could use a `for` loop to write an M-file that would filter all the rows. This would create a new image made up of the filtered rows:

$$y_1[m,n] = \frac{1}{7}\sum_{\ell=0}^{6} x[m,n-\ell] \qquad \text{for } 1 \leq m \leq M$$

However, this image $y_1[m,n]$ would only be filtered in the horizontal direction. Filtering the columns would require another `for` loop, and finally you would have the completely filtered image:

$$y_2[m,n] = \frac{1}{7}\sum_{k=0}^{6} y_1[m-k,n] \qquad \text{for } 1 \leq n \leq N$$

In this case, the image $y_2[m,n]$ has been filtered in both directions by a 7-point averager.

These filtering operations involve a lot of `conv` calculations, so the process can be slow. Fortunately, MATLAB has a built-in function `conv2( )` that will do this with a single call. It performs a more general filtering operation than row/column filtering, but since it can do these simple 1-D operations it will be very helpful in this lab.

(a) To filter the image in the horizontal direction using a 7-point averager, we form a *row* vector of filter coefficients and use the following statement:

```
bh = ones(1,7)/7;
y1 = conv2(xx, bh);
```

In other words, the filter coefficients `bh` for the 7-point averager stored in a *row* vector will cause `conv2( )` to filter all rows in the *horizontal* direction. Display the input image `xx` and the output image `y1` on the screen at the same time. Compare the two images and give a qualitative description of what you see. Extract row #33 (from the bottom again) from the output image (`y1(256-33,:)`) and compare it to the output obtained in part (a) of Section 3.1.

(b) Now filter the image `y1` in the vertical direction with a 7-point running averager to produce the image `y2`. This is done by calling `conv2` with a column vector of filter coefficients. Plot all three of the images `xx`, `y1`, and `y2` on the screen at the same time. Now describe what you see. Where or when have you seen this effect before?

(c) What do you think will happen if you repeat parts (a) and (b) for a 21-point moving averager? Try it, and compare the results of the 21-point and 7-point averaging filters. Which one causes a more severe degradation of the original image?

3.3 More Image Filters

(a) Load the `lenna` image into MATLAB and display the image using the `show_img` command.

(b) Filter the `lenna` image with each of the following filters. Remember to filter both the rows and columns unless otherwise directed.

(i) `a1 = [1 1 1]/3;`

(ii) `a2 = ones(1,7)/7;`

(iii) `a3 = [1 -1];` For this filter, filter only the rows. Note that this filter will yield negative values. Before displaying the resulting image it must be scaled to fit back into the allowable range of [0,255]. This will be done automatically by the `show_img()` command.

(iv) `a4 = [-1 3 -1];` For this filter only filter the rows.

(v) `a5 = [-1 1 1 1 -1];`

Comment on the effect of each filter, pay special attention to regions of the image with lots of detail such as the feathers. Answer the following questions by showing representative images that exhibit the characteristics of highpass or lowpass filtering.

- Which filters appear to be lowpass filters?
- Describe the general effect of a lowpass filter on an image.
- Which ones are highpass filters?

3.4 Frequency Content of an Image

Filters can be used to investigate the frequency content of an image. From what we have seen so far, lowpass filtering of an image causes blurring. In this section, we would like to prove that high-pass (or band-pass) filtering will "sharpen" an image. In the next section, we will examine a system for edge detection that also uses high-pass filters. For this exercise you should use `baboon.mat` as the test image.[3]

(a) To make this demonstration we need some filters that give a smooth frequency response. For this purpose we will use "Gaussian" shaped functions for the FIR filter coefficients $\{b_k\}$. First, of all we need coefficients for a low-pass filter:

$$b_k = \begin{cases} 1.1 & \text{for } k = 10 \\ e^{-0.075(k-10)^2} & \text{for } k = 0, 1, 2, \ldots 20 \text{ and } k \neq 10 \\ 0 & \text{elsewhere} \end{cases}$$

For this filter, plot the impulse response as a `stem` plot, and then plot the magnitude of the frequency response. Put them on the same page with a two-panel subplot. Hint: remember that MATLAB starts indexing at one, the formula for b_k will have to be adjusted by one.

(b) Produce a bandpass filter by the following definition:

$$\tilde{b}_k = \begin{cases} \cos(0.4\pi(k-10))e^{-0.13(k-10)^2} & \text{for } k = 0, 1, 2, \ldots 20 \\ 0 & \text{elsewhere} \end{cases}$$

In this case the filter coefficients $\{\tilde{b}_k\}$ are a Gaussian multiplied by a cosine. Once again, plot the impulse response as a `stem` plot, and also plot the magnitude of the frequency response. Put them on the same page with a two-panel subplot. Explain why this filter is called a bandpass filter (BPF).

(c) Load the test image from `baboon.mat`. Filter this image along both its rows and columns with the LPF from part (a), and save the result as $y[m, n]$ for part (e). Display the image to see the effect of the filter. Determine whether the frequency content of $y[m, n]$ is mostly low frequency or high frequency.

(d) Filter the test image along both its rows and columns with the BPF $\{\tilde{b}_k\}$ and save the result as $v[m, n]$ for part (e). Once again, determine whether the frequency content of $v[m, n]$ is mostly low frequency or high frequency.
NOTE: if you try to display this image you will experience problems because it has negative values which must be rescaled to get a positive image for the CRT. In addition, the resulting display looks mostly gray because its contrast has been compressed significantly due to a small number of very large or very small values. Use `hist` to plot the distribution of pixel values, and then clip the image to eliminate a few of the very large and very small values.

(e) The blurred image in $y[m, n]$ from part (c) can be improved by adding in some of the band-pass filtered image $v[m, n]$ from part (d). The system in Fig. 1 implements the following combination:

$$(1 - \alpha)y[m, n] + \alpha v[m, n]$$

[3]The image `baboon.mat` is one of the MATLAB demos, so it always on the MATLAB path and can be loaded with the `load` command.

where α is a fraction, i.e., $0 \leq \alpha \leq 1$. In effect, α controls how much of the final result will come from the "band-pass filtered" image. Try to find the *best* value of α so that the combination looks as close as possible to the original. Explain why the result looks sharper.

For display, you can put four images together on one page with a four-panel subplot, i.e., `subplot(2,2,*)`. When using `show_img` in subplots, set the second argument to 0 to stay in the same figure (see `help show_img`).

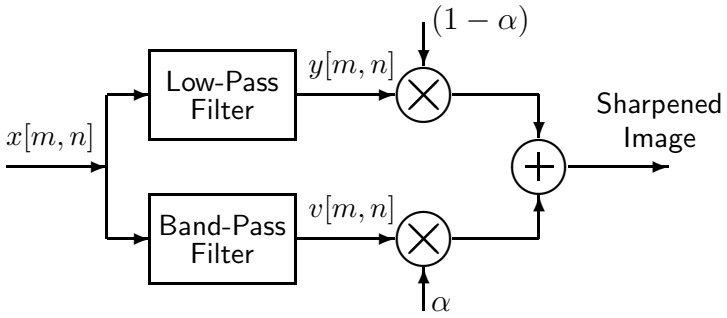

Figure 1: Test set-up for varying low and high frequency content of an image. The parameter α should be between 0 and 1.

3.5 Nonlinear Filters

As you have seen in a previous section of this lab, linear FIR filters are useful in reconstructing images. They can also blur or sharpen images via lowpass or highpass filters. However, there are also other types of filters which are useful in image processing. Nonlinear filters are useful because the frequency content of the output image can be quite different from that of the input image, yet the two are still mathematically and visually related. For example, when an artist touches up a photograph, the output image is "visually related" to the input. A common problem in image processing is to find a sequence of linear and non-linear filters that corresponds to a certain visual relation. The problem posed in this section is, how do we get a computer to trace the outlines of objects in an image? A good solution turns out to require a highpass linear filter followed by a nonlinear filter.

3.6 Edges in an Image

The outline of an object is usually marked by a sudden change in color or gray level. The outline of a black object against a white background would be the set of points where neighbors have a different color. A black point in the image which has a white point to its right would thus be on the edge of the object. Most images don't have solid black and white regions, though.

Since the gray scale variations within each region are usually gradual but the edges are supposed to be rapid variations in color, we suspect that a simple highpass filter could detect edges. Try the first difference filter on an image such as `echart` which has only black and white regions. Display the result as a grayscale image. How well does the filter work on vertical and horizontal edges? what about edges at an angle? Explain.

3.7 The Slope-Threshold Function

If we accept the assumption that color (or grayscale) should change rapidly at an edge, all we have to do is mark the points whose neighbors have greatly different values. For a single point in the image, this means comparing the point's value to the values of its neighbors. If the absolute value of the difference between one point and the next in any direction is greater than some threshold

value, the point is probably on an edge. Figure 2 shows a binary (black/white) image where the pixels greater than the threshold are black.

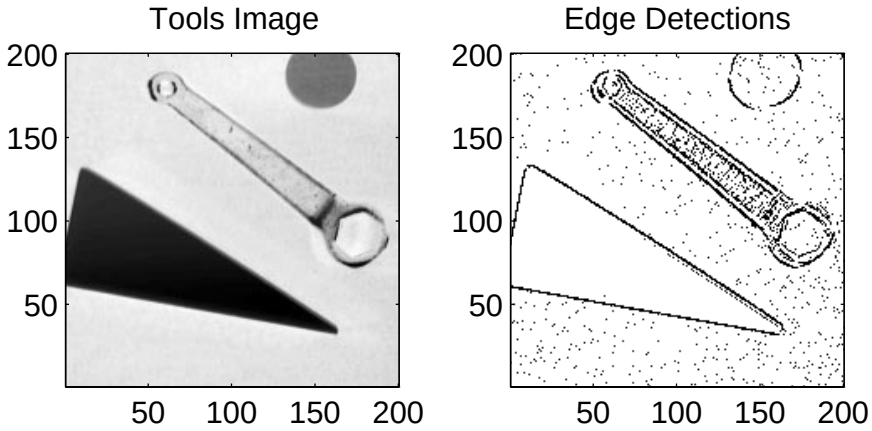

Figure 2: Binary image (right) of edge detections on the 200×200 "Tools" grayscale image (left).

So far this method is only applied to a single point in an image. In MATLAB it is much easier to apply an operation to an entire matrix, though. So, we need to create matrices full of all the differences between points and their nearest neighbors, and then compare each entire matrix to the threshold value.

The easiest way to do this is with the first difference filter. Applying the horizontal first difference filter to the image produces a matrix full of the difference between each point and the one to its right. However, the filtering operation inserts an extra column (see `help filter2`). Which column is extra? Can you remove it? You should be able to produce a matrix `yh` of exactly the same size as the original image where each point in the matrix is exactly the difference between the corresponding point and the one to its right in the original image. We also need the vertical first difference for the edge detector. If we examine the difference to the right and the difference downward, that will cover all the points so that the difference between each point and the next will be evaluated exactly once. In an approximation of the edges, we can disregard the difference to the left and the difference upward because those differences will be evaluated for the points which lie to the left and up. What this means is that for a sudden transition between two points, only one of the two points will be marked as an edge.

Now you should have the horizontal (`yh`) and vertical (`yv`) difference matrices trimmed so that they line up with the original image, i.e. `xx(1,1)-xx(1,2)` equals `yh(1,1)` and so on. Now you have to compare their absolute values to the threshold value. In MATLAB, logical operations such as `a>b`, `a<b`, etc. produce result values equal to 1 for true and 0 for false. The logical operation `a|b` means "a OR b" and it will be useful here. The statement

```
yy = (abs(yh)>thr)|(abs(yv)>thr);
```

where `thr` is the threshold value, should produce a matrix `yy` filled with zeros where points in the original image do not lie on edges and ones where they do. For visibility, you should scale the output so that the minimum and maximum values are at least 200 apart. For printing, you should invert the color so that most of the output is white.

If the threshold is zero, any point which is not surrounded by points of the same color is marked. This is too sensitive for an image with a good variety of color, i.e. a photograph. If the threshold value is too high, the edge of a gray object on a black background might not be marked. The

threshold value should be carefully chosen for this operation. For `lenna` or `echart`, a good value might be between 8 and 18. Experiment with the threshold value to get a good looking result—it is possible to make the output look like a line drawing that outlines the head, hat and feathers in `lenna.mat`.

3.8 What's Nonlinear about Edge Detection?

Although we used the horizontal and vertical first difference filters which are LTI systems, the logical operation is not linear. The threshold comparison converts each line of the image into a type of "square wave" function. Plot row 94 of the output image `y` along with the corresponding row of the input image `xx`. Is edge detection sensitive to low frequency or high frequency content, or neither? Explain.

Lab 10
Instructor Verification Sheet

Staple this page to the end of your Lab Report.

Name: _____ Date: _____

Part 2.1 Create and display sinusoidal test image:

Verified: _____

Part 2.1 Display "lenna" image and image negative:

Verified: _____