GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

EE 2200 Fall 1998
Lab #8: Sampling and Zooming of Images

Date: week of 16 Nov 1998

Lab Quiz will be held at the beginning of this week's lab time.

This is *the official* Lab #8 description; it is based on Labs C.8 and C.9 in Appendix C of the text.

The Warm-up section of each lab must be completed in Lab and the steps marked *Instructor Verification* must also be signed off **during the lab time**.

The lab report for this lab will be **FORMAL:** discuss your results from section 4. Staple the **Instructor Verification** sheet to the end of your lab report.

The report will due during the week of **30-Nov** at the start of your lab.

---

1 Introduction

In this lab we introduce digital images as a signal type for the application of filtering. We will study the application of FIR filtering to the image zooming problem, where lowpass filters are used to do the interpolation needed for high quality zooming. The zooming problem is basically the same as the D-to-A reconstruction problem treated in Chapter 4; the exception being that the output of the zooming operation is still a digital image.

In the experiments of this lab, you will use `conv2()`, to implement two-dimensional (2-D) filters. The 2-D filtering operation will consist of 1-D filters applied to all the rows of the image and then all the columns. Therefore, the function `freqz()` can still be used to characterize the filter's frequency response.[1]

2 Overview

2.1 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.[2] 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.[3] 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 *gray-scale* images. In this lab we will consider only sampled gray-scale still images.

A sampled gray-scale 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 which has nearly the same resolution as a standard TV image. In MATLAB we can represent an image as a matrix consisting of

[1] If you are working at home and do not have the function `freqz.m`, there is a substitute available called `freekz.m`. You can get it from the EE-2200 WebCT page.

[2] The variables t_1 and t_2 are confusing since they *do not denote time*.

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

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 $x[m, n]$ have to be scaled relative to the maximum value $X_{\max}$. Usually an eight-bit integer representation is used. With 8-bit integers, the maximum value can be $X_{\max} = 2^8 - 1 = 255$, and there are $2^8 = 256$ gray levels for the display.

2.2 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).

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

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.[4] The `show_img` function will handle the color map and the "true" size of the image. 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. The MATLAB function `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:

[4] If the MATLAB function `imagesc.m` is used to display the image, two features will be missing: (1) the color map may be incorrect because it will not default to gray, and (2) the size of the image will not be a true pixel-for-pixel rendition of the image on the computer screen.

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. In the *DSP First* toolbox, the function `clip.m` is provided to do this job.

CD-ROM
clip.m

8. *Automatically rescaling the image:* This requires a linear mapping of the pixel values:[5]

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

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:

$$x_s[m,n] = \left\lfloor 255.999 \left(\frac{x[m,n] - x_{\min}}{x_{\max} - x_{\min}} \right) \right\rfloor$$

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

2.3 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.

3 Warm-up: Display of Images

CD-ROM
IMAGE DATA FILES

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_plate.mat`, but there are others within MATLAB's demos. The default size is 256×256, but alternate versions are available as 512×512 images under names such as `lenna_512.mat`. After loading, use the command `whos` to determine the name of the variable that holds the image and its size.

CD-ROM
show_im.m

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 `soundsc()`, 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
%----
```

[5] The MATLAB function `show_img` has an option to perform this scaling while making the image display.

Notice that unless the input parameter `figno` is specified, a new figure window will be opened. This feature must be turned off 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. In addition, having several images on one printed page makes it much easier to discuss comparisons in a lab report.

3.1 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 intra-row index is the second variable in `xx(n1,n2)`; in other words, `xx(7,nn)` denotes the `nn`-th sample within the 7-th row in the image `xx()`.

(b) Recall the outer product trick for repeating rows or columns, e.g., `qq = ones(4,1)*[3,1,4]`

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

(d) 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, when the sinusoid is zero, what is the (integer) gray level? how about when the sinusoid's value is -0.2?

Instructor Verification (separate page)

(e) Now load and display the `lenna` image from `lenna.mat`. The command `load lenna` will put the sampled image into the array `xx`. Use `whos` to check the size of lenna.

(f) Use the colon operator to extract the 200^{th} row of the `lenna` image, and make a plot of the 200^{th} row of `lenna` as a 1-D discrete-time signal. Observe the maximum and minimum values.

(g) 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 of `lenna` so that white and black are interchanged.

3.2 Warm-up: Linear Interpolation

In Section 4.6, we will be interested in image zooming which is an interpolation problem. Before processing images, however, we consider the process of one-dimensional interpolation. The purpose of this warm-up is to verify that interpolation can be decomposed into two simpler systems as shown Fig. 1. In this case, two

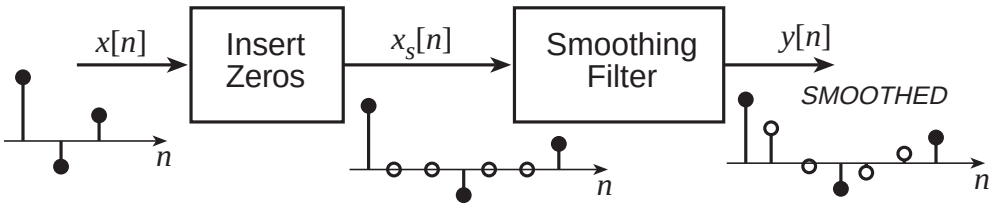

Figure 1: 1-D Digital Interpolation broken down into two systems: zero-insertion followed by lowpass filtering. In this case, the signal is being interpolated "up" by a factor of 3.

zeros are inserted between the samples in the input signal $x[n]$. The "smoothing" filter is a lowpass filter with special constraints on its filter coefficients. In the example below, the filter coefficients form an impulse response that is triangular in shape. As we know from Chapter 4, a triangular reconstruction pulse would yield linear interpolation for a D/A converter. You should be able to demonstrate that a similar result holds for the digital interpolator.

(a) Generate a sequence of data samples with two zeros between each non-zero sample:

```
xss = zeros(1,19);
xx = [1 3 -2 4 2 -1 -3];  %<-- input signal
xss(1:3:19) = xx;
xss = [xss,0,0];   %<-- needed to make the length(xss) = 3*length(xx)
```

(b) Now process this sequence through an FIR filter with "triangular" coefficients (using `conv` or `firfilt`).

```
coeffs = [1/3, 2/3, 1, 2/3, 1/3];
output = firfilt(xss,coeffs);
```

What observations can you make regarding the similarities between the sequence `output` and the nonzero samples of the original sequence (`xss`)? Do you notice a relative shift between these two sequences? Measure the length of this time shift in samples. Explain why the output of the triangle FIR filter is a *linearly interpolated* version of the input sequence. Does the frequency response of the "triangular" filter have a lowpass shape?

Instructor Verification (separate page)

(c) Write the difference equation for the "triangular" filter defined by `coeffs` in part (b). Now calculate, *by hand*, the output of the filter to the input sequence

$$x[n] = \delta[n] + 0.5\delta[n-3]. \quad (1)$$

$$= \begin{cases} 1, & \text{when } n = 0 \\ 0.5, & \text{when } n = 3 \\ 0, & \text{else} \end{cases} \quad (2)$$

Do not use MATLAB to complete this exercise. Make a table of values and generate them by hand. The purpose is to understand exactly how the linearly interpolated output is generated when you implement the difference equation.

In summary, we notice that linear interpolation involves two steps (Fig. 1) The first step is zero filling, where the number of zeros inserted between each sample determines the degree of interpolation. The second step is FIR filtering with the appropriate triangle coefficients.

4 Lab: Filtering and Sampling of Images

4.1 Filtering Images

In previous labs, you have experimented with one-dimensional FIR filters, such as running averagers and first-difference filters, applied to one-dimensional signals. These same filters can be applied to images if we regard each row (or column) of the image as a one-dimensional signal. For example, the 50^{th} row of an image is the N-point sequence `xx[50,n]` for $1 \leq n \leq N$, so we can filter this sequence with a 1-D filter using the `conv` or `firfilt` operator.

4.2 Blurring an Image

One objective of this lab is to show how simple 2-D filtering can be accomplished with 1-D row and column filters. It might be tempting to 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:

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

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:

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

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.

`echart.mat`

(a) Load in the image `echart.mat` with the `load` command. Extract the 33^{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. Use the frequency response to explain your observation.

(b) We can filter all the rows of the image at once with the `conv2( )` function. 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;
yy1 = 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 `yy1` 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 (`yy1(256-33,:)`) and compare it to the output obtained in part (a).

(c) Now filter the image `yy1` in the vertical direction with a 7-point running averager to produce the image `yy2`. This is done by calling `conv2( )` with a column vector of filter coefficients. Plot all three of the images `xx`, `yy1`, and `yy2` on the screen at the same time. Now describe in words how the output images compare to the input.

(d) What do you think will happen if you repeat parts (b) and (c) 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? Use the frequency response as part of your explanation.

4.3 Filtering a Digital Photograph (Optional)

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

(ii) `a2 = ones(1,21)/21;`

(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.

Observe the effect of each filter, paying special attention to regions of the image with lots of detail such as the feathers. Explain in words the effects of highpass and lowpass filtering of this image.

4.4 Sampling of Images

Images that are stored in digital form on a computer have to be sampled images because they are stored in an $M \times N$ array. The sampling rate in the two spatial dimensions was chosen at the time the image was digitized (in units of samples per inch if the original was a photograph). For example, the image might have been "sampled" by a scanner where the resolution was chosen to be 300 dpi (dots per inch).[6] If we want a different sampling rate, we can simulate a *lower* sampling rate by simply throwing away samples in a periodic way. For example, if every other sample is removed, the sampling rate will be halved. Sometimes this is called *sub-sampling* or *down-sampling*.

(a) If the vector `x1` represents a signal such as a row of an image, we can reduce the sampling rate by a factor of 4 by simply taking every 4^{th} sample. In MATLAB this is easy with the colon operator, i.e., `xd = x1(1:4:length(x1))`. The vector `xd` is one fourth the length of `x1`.

An alternative sampling strategy is to take every 4^{th} sample, but place zeros in between. The M-file below called `imsample( )` performs this type of sampling on an image, e.g., `xs = imsample(xx,4);`. Explain how the function `imsample.m` works by giving a mathematical description.

```
function yy = imsample(xx, P)
%IMSAMPLE    Function for sub-sampling an image
% usage:   yy = imsample(xx, P)
% xx = input image to be sampled
%   P = sub-sampling period (an integer like 2,3,etc)
%  yy = output image
%
[M,N] = size(xx);
S = zeros(M,N);
S(1:P:M,1:P:N) = ones(length(1:P:M), length(1:P:N));
yy = xx .* S;
```

(b) Execute the statement `xs = imsample(xx,4);` for the `echart.mat` image, and use `show_img( )` to plot the images `xs` and `xx` side by side. From the plot you should see that `imsample( )` throws

[6] The Sampling Theorem applies to digital images, so there is a *Nyquist Rate* that depends on the maximum *spatial* frequency in the image.

away samples by setting them to zero. The samples that it keeps remain in their original spatial locations. With the zero samples included, the "sampled image" has the same spatial dimensions as the original when displayed on the screen.

(c) *Down-sampling* throws away samples, so it will shrink the size of the image. This is what is done by the following scheme:

```
xp = xx(1:p:M,1:p:N);
```

One potential problem with down-sampling is that aliasing might occur. This can be illustrated in a dramatic fashion with `zone_plate` image, or the `lenna` image.

Now down-sample the `zone_plate` image by a factor of 2. Notice the aliasing in the down-sampled image, which is surprising since no new values are being created by the down-sampling process. Describe how the aliasing appears visually.[7]

Down-sample the `lenna` image by a factor of 2. Notice that this image seems to be relatively unaffected by the down-sampling by 2 process, what can you say about the frequency content of the `lenna` image as opposed to the `zone_plate` image?

4.5 Reconstruction of Images

When an image has been sampled, we can fill in the missing samples by doing interpolation. For images, this would be analogous to the examples shown in Chapter 4 for sine-wave interpolation which is part of the reconstruction process in a D-to-A converter. We could use a "square pulse" or a "triangular pulse" or other pulse shapes.

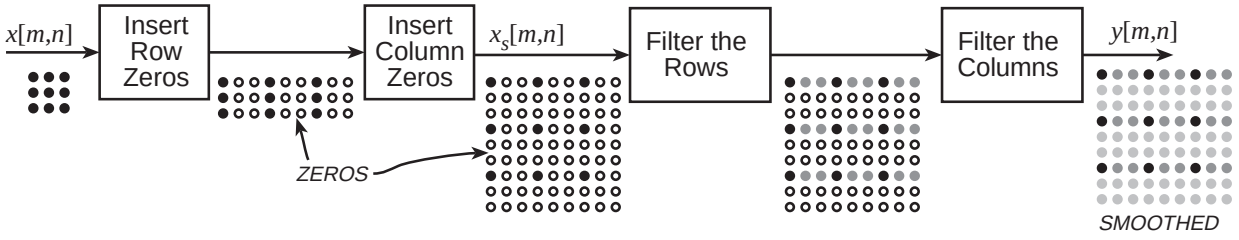

Figure 2: 2-D Interpolation broken down into four systems: zero-insertion on the rows and columns followed by 1-D lowpass filtering of the rows and columns. The open circle indicate zeros placed in the image; the gray dots indicate data values created by filtering.

For these reconstruction experiments, use the `lenna` image. It would be wise to put the original image and all the filtered reconstructions on the screen in different figure windows for easy comparison. However, when making hardcopy for your lab report, use `subplot` to print multiple images on the same page

(a) The simplest interpolation would be reconstruction with a square pulse which produces a "zero-order hold." We could fill in the gaps between samples in each row with the following statement:

```
xs = imsample(xx,4);
bs = ones(1,4);        %--- Length-4 square pulse
yhold = conv2(xs,bs);  %--- 2-D FIR filtering (horizontally)
```

Try this and plot the result `yhold`.

[7] One difficulty with showing aliasing is that we must display the pixels of the image exactly. This almost never happens because most monitors and printers will perform some sort of interpolation to adjust the size of the image to match the resolution of the device. In MATLAB we can override these size changes by using the function `truesize` which is part of the Image Processing Toolbox. In the *DSP First Toolbox,* an equivalent function called `trusize.m` is provided.

(b) Now filter the columns of `yhold` to fill in the missing points in each column and compare the result to the original image `xx`.

(c) *Linear interpolation:* Now use what you learned about linear interpolation in the warm-up section to determine an FIR filter that will perform linear interpolation on the rows and columns of the sub-sampled (by 4) signal. This filter must have coefficients $\{b_k\}$ that follow a triangle shape, and the order must be $M = 6$.

(d) Carry out the linear interpolation operations using MATLAB's `conv2` function. Call the interpolated output `ylin`. Compare `ylin` to the original image `xx` and to the square pulse interpolated image from part (b). Comment on the visual appearance of the two "reconstructed" images.

(e) Compute the frequency response of the linear interpolator used in the previous part. Only the 1-D FIR filter that acts on the rows must be analyzed. Plot its magnitude (frequency) response for $-\pi \leq \hat{\omega} \leq \pi$.

(f) *Smoothing via Lowpass Filtering:* At this point, you should be thinking that interpolation is very similar to lowpass filtering. To test this hypothesis, create a lowpass filtered version `xs_filt` of the sampled image by filtering the rows and columns with a 23-point FIR filter, whose coefficients are given by a modified "sinc" formula:

$$b_k = \frac{\sin(\pi(k-11)/4)}{\pi(k-11)/4}\, w_k \qquad k = 0, 1, 2, \ldots, 22$$

where w_k is given by (the Hamming shape):

$$0.54 - 0.46\cos\left(\frac{2\pi k}{22}\right) \qquad k = 0, 1, 2, \ldots, 22$$

Remember that MATLAB will have problems evaluating b_{11} because the sinc function is an indeterminate form when $k = 11$. You will have to compute b_{11} yourself and include it at the proper location in the vector b_k. Compare `xs_filt` to the original image and also to the linear interpolation output. In addition, plot the magnitude of the frequency response for this FIR filter.

4.6 Zooming for an Image

Zooming in on a section of an image is very similar to the D-to-A reconstruction process because it also requires interpolation. The three interpolation systems (zero-order hold, linear interpolation, and lowpass filtering) developed in the previous section, can be applied to do zooming by a factor of four.

(a) Take a small patch of an image (about 70×70) where there is some interesting detail (e.g., the eye or feathers in `lenna`). There are several ways to produce a larger image that appears zoomed. One way is to simply repeat pixel values. So in order to zoom a 70×70 section up to 280×280, you would repeat each pixel four times in each direction. Display a zoomed portion of an image by repeating pixel values.

(b) Another way to produce a larger image is to insert zeros between the existing samples. In other words, you must produce an image that is 200×200 with data values only in rows or columns whose index is a multiple of four. Consider the following code that does this for a 1-D vector.

```
L = length(xx);
yy = zeros(1,4*L);
yy(4:4:4*L) = xx;
```

Generalize this idea to write a function that will insert zeros in the rows and columns of a 2-D image.

(c) Now filter the image from part (b) to do the interpolation. Use both the linear and the "sinc" function interpolators.

(d) Comment on the ability of all three zooming operators to preserve detail and edges in the image while expanding the size of details. Try to explain your observations by considering the frequency content of the "zoomed" image.

(e) MATLAB has a zoom command in the Image Processing Toolbox. The function is called `imzoom`. If this is available on your system, try to determine what sort of interpolation scheme it is using.

4.7 Zooming in Software (Optional)

If you have used an image editing program such as Adobe's "Photoshop," you might have observed how well or how poorly image zooming is done. For example, if you try to blow up a JPEG file that you've downloaded from the web, the result is usually disappointing. Since MATLAB has the capability to read lots of different formats, you can apply the image zooming via interpolation to any photograph that you can acquire. The MATLAB function for reading JPEG images is `imread( )` which would be invoked as follows:

```
xx = imread('foo.jpg','jpeg');
```

Since `imread( )` is part of the image processing toolbox, this test can be done in the CoC computer labs, but may not be possible on your home computer.

4.7.1 Warnings

Images obtained from JPEG files might come is many different formats. Two precautions are necessary:

1. If MATLAB loads the image and stores it as 8-bit integers, then MATLAB will use an internal data type called `uint8`. The function `show_img( )` cannot handle this format, but there is a conversion function called `double( )` that will convert the 8-bit integers to double-precision floating-point for use with filtering and processing programs.

   ```
   yy = double(xx);
   ```

2. If the image is a color photograph, then it is actually composed of three "image planes and MATLAB will store it as a 3-D array. For example, the result of `whos` for a 545×668 color image would give:

   ```
   Name      Size            Bytes  Class
   xx      545x668x3    1092180  uint8 array
   ```

   In this case, you should use MATLAB's image display functions such as `imshow( )` to see the color image. Or you can convert the color image to gray-scale with the function `rgb2gray( )`. For more information on the image processing functions in MATLAB, try help:

   ```
   help images
   ```

Lab #8
EE-2200
Fall-1998
INSTRUCTOR VERIFICATION PAGE

Staple this page to the end of your Lab Report.

Name: _____ Date of Lab: _____

Part 3.1 Create and display sinusoidal test image:

Verified:_____ Date/Time:_____

Part 3.2 Perform linear interpolation with a triangle-shaped FIR Filter in MATLAB:

Verified:_____ Date/Time:_____