Project 4: Image Signal Processing and ADC & DAC

You will write up a formal lab report in IEEE double-column format with figures integrated with the text. The exercises should written up in this week's lab report. You should **label** the axes of your plots, have a caption, and Figure number for every plot. Every plot should be referenced by Figure number in your text discussion. Include each plot *inlined* within your report.

Forgeries and plagiarism are a violation of the honor code and will be referred to the Dean of Students for disciplinary action. You are allowed to discuss lab exercises with other students and you are allowed to consult old lab reports but the submitted work should be original and it should be your own work.

The objective in this lab is to introduce digital images as a second useful signal type. We will discuss these concepts as image processing or image signal processing. We will show how the A-to-D sampling and the D-to-A reconstruction processes are carried out for digital images. In particular, we will show that pixel repetition, a commonly used method of image zooming (reconstruction), gives "poor" results, but other interpolators will do a better job.

1 Introduction to Digital Image Processing

1.1 Digital Images

In this lab we introduce digital images as a signal type for studying the effect of sampling, aliasing and reconstruction. 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.[1] For monochrome images, the signal $x(t_1, t_2)$ would be a scalar function of the two spatial variables, but for color images the function $x(\cdot, \cdot)$ would have to be a vector-valued function of the two variables.[2] Moving images (such as 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

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

where T_1 and T_2 are the sample spacings in the horizontal and vertical directions. Typical values of M and N are 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, so it would consist 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).

[1]The variables t_1 and t_2 *do not denote time, they represent spatial dimensions.* Thus, their units would be inches or some other unit of length.

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

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.,

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

This is because light images are formed by measuring the intensity of reflected or emitted light, and intensity 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 a maximum value $X_{\max}$. Usually an eight-bit integer representation is used. With 8-bit integers, the maximum value (in the computer) would be $X_{\max} = 2^8 - 1 = 255$, and there would be $2^8 = 256$ gray levels for the display, from 0 to 255.

1.2 Displaying Images

As you will discover, the correct display of an image on a computer monitor can be tricky, especially if the processing performed on the image generates negative values. We have provided the function `show_img.m` in the *SP-First* toolbox to handle most of these problems,[3] but it will be helpful if the following points are noted:

- All image values must be non-negative for the purposes of display. Filtering may introduce negative values, especially when a first-difference is used (e.g., a high-pass filter).
- 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 $0 \le x[m,n] \le 255 = 2^8 - 1$.
- The actual display on the monitor created with the `show_img` function[4] 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, we want a "grayscale display" where all three primary colors (red, green and blue, or RGB) are used equally, creating what is called a "gray map." In MATLAB the `gray` color map is set up via

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

  which gives a 256×3 matrix where all 3 columns are equal. The 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 lab experiments, non-linear color mappings would introduce an extra level of complication, so we won't use them.
- 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. In this lab, we use `show_img.m` to *automatically rescale the image:* This does a linear mapping of the pixel values:[5]

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

  The scaling constants μ 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.

[3]If you have the MATLAB Image Processing Toolbox, then the function `imshow.m` can be used instead.

[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.

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

Below is the help on show_img; notice that unless the input parameter figno is specified, a new figure window will be opened.

```
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
%----
```

1.3 Overview of Filtering

For this lab, we will define an FIR *filter* as a discrete-time system that converts an input signal $x[n]$ into an output signal $y[n]$ by means of the weighted summation:

$$y[n] = \sum_{k=0}^{M} b_k \, x[n-k] \tag{1}$$

Equation (1) gives a rule for computing the n^{th} value of the output sequence from certain values of the input sequence. The filter coefficients $\{b_k\}$ are constants that define the filter's behavior. As an example, consider the 3-point averaging system for which the output values are given by

$$y[n] \quad = \quad \tfrac{1}{3}x[n] + \tfrac{1}{3}x[n-1] + \tfrac{1}{3}x[n-2]$$

This equation states that the n^{th} value of the output signal is the average of the n^{th} value of the input signal $x[n]$ and the two preceding values, $x[n-1]$ and $x[n-2]$. The b_k's are $b_0 = \frac{1}{3}$, $b_1 = \frac{1}{3}$, and $b_2 = \frac{1}{3}$.

2 Using MATALB for Images

2.1 MATLAB Function to Display Images

You can load the images needed for this lab from *.mat files, or from *.png files. Image files with the extension *.png can be read into MATLAB with the imread function. Any file with the extension *.mat is in MATLAB format and can be loaded via the load command. After loading, use the command 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 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.

2.2 Get Test Images

In order to probe your understanding of image display, do the following simple displays:

- Load and display the 428×642 "lighthouse" image[6] from `lighthouse.png`. This image can be downloaded from Web-CT. The MATLAB command `ww = imread('lighthouse.png')` will put the sampled image into the array `ww`, Use `whos` to check the size and type of `ww` after loading. Notice that the array type for `ww` is `uint8`, so it might be necessary to convert `ww` to double precision floating-point with the MATLAB command `double`. When you display the image it might be necessary to set the colormap via `colormap(gray(256))`.

- Use the colon operator to extract the 440th row of the "lighthouse" image, and make a plot of that row as a 1-D discrete-time signal.

```
ww440 = ww(440,:);
```

  Observe that the range of signal values is between 0 and 255. Which values represent white and which ones black? Can you identify the region where the 440th row crosses the fence? Can you match up a black region between the image and the 1-D plot of the 440th row?

2.3 FIR Filtering in MATLAB

MATLAB has built-in functions, `conv( )` and `filter( )`, for implementing the operation in (1), but we have also supplied another M-file `firfilt( )` for the special case of FIR filtering. The function `filter` implements a wider class of filters than just the FIR case. Technically speaking, both `conv` and `firfilt` implement the operation called *convolution.* The following MATLAB statements implement the three-point averaging system of (2):

```
nn = 0:99;                      %<--Time indices
xx = cos( 0.08*pi*nn );       %<--Input signal
bb = [1/3 1/3 1/3];           %<--Filter coefficients
yy = firfilt(bb, xx);         %<--Compute the output
```

In this case, the input signal `xx` is a vector containing a cosine function. In general, the vector `bb` contains the filter coefficients $\{b_k\}$ needed in (1). These are loaded into the `bb` vector in the following way:

```
bb = [b0, b1, b2, ...  , bM].
```

In MATLAB, all sequences have finite length because they are stored in vectors. If the input signal has, for example, N samples, we would normally only store the N samples in a vector, and would assume that $x[n] = 0$ for n outside the interval of N samples; i.e., we do not have to store any zero samples unless it suits our purposes. If we process a finite-length signal through (1), then the output sequence $y[n]$ will be longer than $x[n]$ by M samples. Whenever `firfilt( )` implements (1), the relationship is:

```
length(yy) = length(xx)+length(bb)-1
```

2.4 Debugging

One of the most useful modes of the debugger causes the program to jump into "debug mode" whenever an error occurs. This mode can be invoked by typing:

```
dbstop if error
```

[6]The image size of 428×642 is the horizontal by vertical dimensions. When stored in a MATLAB matrix the `size` command will give the matrix dimensions, i.e., number of rows by number of columns, which is `[642 428]` for the lighthouse image.

With this mode active, you can snoop around inside a function and examine local variables that probably caused the error. You can also choose this option from the `Breakpoints` menu in the MATLAB editor. It's sort of like an automatic call to 911 when you've gotten into an accident.

When unsure about a command, use `help`.

Download the file `coscos.m` and use the debugger to find the error(s) in the function. Call the function with the test case: `[xn,tn] = coscos(2,3,20,1)`. Use the debugger to:

- Set a breakpoint to stop execution when an error occurs and jump into "Keyboard" mode,
- display the contents of important vectors while stopped,
- determine the size of all vectors by using either the `size()` function or the `whos` command.
- and, lastly, modify variables while in the "Keyboard" mode of the debugger.

```
function [xx,tt] = coscos( f1, f2, fs, dur )
% COSCOS   multiply two sinusoids
%
t1 = 0:(1/fs):dur;
t2 = 0:(1/f2):dur;
cos1 = cos(2*pi*f1*t1);
cos2 = cos(2*pi*f2*t2);
xx = cos1 .* cos2;
tt = t1;
```

2.5 Filtering a Signal

You will now use the signal vector defined via `x1(1:3:30) = 0:9;` as the input to an FIR filter whose filter coefficients are

```
bb = [0,1,2,3,2,1,0]/3;
```

- For this warm-up, process the signal `x1` with the FIR filter defined by `bb`. How long are the input and output signals?
- To illustrate the filtering action, you must make a plot of the input signal and output signal together using subplot. Since $x_1[n]$ and $y_1[n]$ are discrete-time signals, a `stem` plot is needed. One way to put the plots together is to use `subplot(2,1,*)` to make a two-panel display:

```
nn = first:last;
subplot(2,1,1);
stem(nn,x1(nn))
subplot(2,1,2);
stem(nn,y1(nn),'filled')   %--Make black dots
xlabel('Time Index (n)')
```

  This code assumes that the output from `firfilt` is called `y1`. Try the plot with `first` equal to the beginning index of the input signal, and `last` chosen to be the *last index of the input.* The plotting range for both signals should be set equal to the length of the input signal, even though the output signal is longer.

2.6 Filtering Images: 2-D Convolution

One-dimensional FIR filters can be applied to images if we regard each row (or column) of the image as a one-dimensional signal. For example, the 50th row of an image is the N-point sequence `xx[50,n]` for $1 \le n \le N$, so we can filter this sequence with a 1-D filter using the `conv` or `firfilt` operator. It is possible to use a `for` loop to write an M-file that would filter all the rows. For a *first-difference filter*, this would create a new image made up of the filtered rows:

$$y_1[m,n] = x[m,n] - x[m,n-1]$$

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:

$$y_2[m,n] = y_1[m,n] - y_1[m-1,n]$$

In this case, the image $y_2[m,n]$ has been filtered in both directions by a first-difference filter

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 could be helpful in this lab.

- Load in the lighthouse image creating the variable `ww`. 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 first-difference filter, we form a *row* vector of filter coefficients and use the following MATLAB statements:

  ```
  bdiffh = [1, -1];
  yy1 = conv2(ww, bdiffh);
  ```

  In other words, the filter coefficients `bdiffh` for the first-difference filter are stored in a *row* vector and will cause `conv2( )` to filter all rows in the *horizontal* direction.

- To filter in the *vertical* direction with a first-difference filter, use `yy2 = conv2(ww,bdiffh’)`, i.e., put the filter coefficient into a column vector.

2.7 Printing Multiple Images on One Page

The phrase “what you see is what you get” can be elusive when dealing with images. It is ***very tricky*** to print images so that the hard copy matches exactly what is on the screen, because there is usually some interpolation being done by the printer or by the program that is handling the images. One way to think about this in signal processing terms is to think of the screen as one kind of D-to-A and the printer as another kind; each one uses a different D-to-A reconstruction method to get the continuous-domain (analog) output image that you see.

Another problem occurs when you try to put two images of different sizes into subplots of the same MATLAB figure. It doesn’t work because MATLAB wants to force them to be the same size. Therefore, you should display your images in separate MATLAB figure windows.

2.8 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 (i.e., a matrix). 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).[7] 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 (in our example, the 300 dpi image would become a 150 dpi image). Usually this is called *sub-sampling* or *down-sampling*.[8]

[7]For this example, the sampling periods would be $T_1 = T_2 = 1/300$ inches.

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

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

```
wp = ww(1:p:end,1:p:end);
```

when we are downsampling by a factor of `p`.

- One potential problem with down-sampling is that aliasing might occur because f_s is being changed—it's getting smaller. This can be illustrated in a dramatic fashion with the `lighthouse` image.

  Read in the `lighthouse.png` file with the MATLAB function `imread`. When you check the size of the image, you'll find that it is not square. Now down-sample the `lighthouse` image by a factor of 2. What is the size of the down-sampled image? 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.[9]

  Which parts of the image show the aliasing effects most dramatically? Explain why the aliasing is happening by thinking about high frequencies in the image, i.e., look for features in the images that are periodic and can be described as having a frequency.

  Explain why the aliasing happens in the `lighthouse` image by using a "frequency domain" explanation. In other words, estimate the frequency of the features that are being aliased. Give this frequency as a number in cycles per pixel. (Note that the fence provides a sort of "spatial chirp" where the spatial frequency increases from left to right.) Can you relate your frequency estimate to the Sampling Theorem?

3 Sampling, Aliasing and Reconstruction

3.1 3 Synthesizing a Test Image

3.2 Aliasing in a Test Image

In order to probe your understanding of the relationship between MATLAB matrices and image display, you can generate a synthetic image from a mathematical formula. Then you can use the theory of sampling and aliasing to explain how downsampling the cosine formula will provide surprising results.

(a) Generate a simple test image in which all of the columns are identical by using the following outer product of vectors:

```
xpix = ones(256,1)*cos(2*pi*(0:255)/32);
```

Display the image and explain the gray-scale pattern that you see. Count the number of black stripes across the image. Explain how you can predict that number from the period of the formula for xpix?

(b) In the previous part, which data value in xpix is represented by white? which one by black? Keep in mind that the cosine has values between 1. Instructor Verification (separate page)

[9]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 *SP-First* toolbox, an equivalent function called `trusize.m` is provided.

(c) Explain how you would produce an image with bands that are horizontal. Give the formula that would create a 400 ? 400 image with five horizontal black bands separated by white bands. Write the MATLAB code to make this image and display it.

The banding structure in the test images is controlled by the frequency of the cosine. In other words, we can rewrite the formula for the test image (above) as

```
wd = 2*pi*1/32; xpix = ones(256,1)*cos(wd*(0:255));
```

(a) Generate two test images with different frequencies, one with wd = 2*pi*4/32 and the other with wd = 2*pi*12/32. Call these images xpix4 and xpix12. Display the images, and explain why the image made from the higher frequency cosine has a shorter horizontal period.

(b) Nowweapplydownsamplingbytwo,i.e.,xpix4(1:2:end,1:2:end)andxpix12(1:2:end,1:2:end), to both images from the previous part. Use subplot(2,2,n) to make a four-panel display; put xpix4 and xpix12 in the top row, and put the two down-sampled images in the bottom row of the 2 ? 2 sub- plot. Explain why the two down-sampled images look the same.

3.3 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 for the reconstruction.

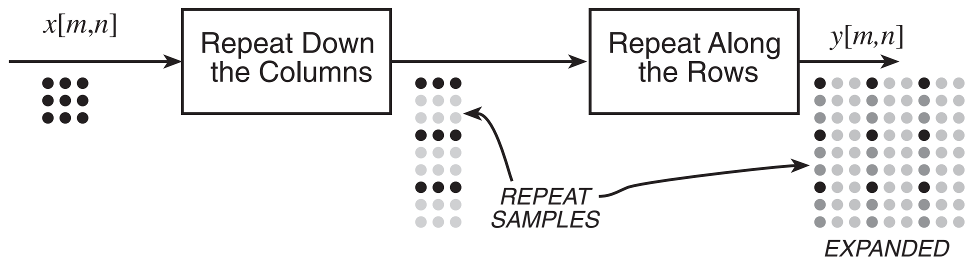

Figure 1: 2-D Interpolation broken down into row and column operations: the gray dots indicate repeated data values created by a zero-order hold; or, in the case of linear interpolation, they are the interpolated values.

For these reconstruction experiments, use the `pccat.png` image, down-sampled by a factor of p. Use $p = 5$, but write your code so that it would work for any integer value of p. Downsampling should be done in the same manner as in Section 2.8. You will have to read in the image from `pccat.png` using the `imread` function, and you should store the image into an array named `pccat`. Two down-sampled `pccat` images should then be created and stored in the variables `pccat5` and `pccat10`. The objective will be to reconstruct an approximation to the original `pccat` image, which is 892×592, from the smaller down-sampled images.

- The simplest interpolation would be reconstruction with a square pulse which produces a *zero-order hold.* The MATLAB code below is a method that uses an FIR filter and works for a one-dimensional signal (i.e., one row or one column of the image). If we start with a row vector `pc1`,then the result is the row vector `pc1hold`.

```
pc1 = cos(2*pi*(0:7)/8);
L = length(pc1);
pcp(1:8:8*L) = pc1;
pc1hold = firfilt(ones(1,8),pcp);
```

Plot the vector `pc1hold` to verify that it is a *zero-order hold* version derived from `pc1`. Plot the vector `pcp` and describe in words how it is derived from `pc1`. If we define the *interpolation ratio* as the relative size of the interpolated signal to the original, determine the *interpolation ratio* when going from `pc1` to `pc1hold`. Your lab report should include an explanation for this part; use plots to simplify the explanation.

- Now return to the down-sampled `pccat` image, and process all the columns of `pccat5` to fill in the missing points. Use the zero-order hold idea from part (a), but do it for an interpolation ratio of 5. Call the result `pc5holdcols`. Display `pc5holdcols` as an image, and compare it to the downsampled image `pccat5`; compare the sizes of the images as well as their content.
- Now process all the rows of `pc5holdcols` to fill in the missing points in each row and and call the result `pc5hold`. Compare the result (`pc5hold`) to the original image `pccat`. Include your code for parts (b) and (c) in the lab report.[10]
- *Linear interpolation* can be done with an FIR filter that has a triangular shaped impulse response. Here is an example on a 1-D signal:

```
n1 = 0:7;
pc1 = cos(2*pi*n1/8);
pcp(1:8:8*length(pc1)) = pc1;
pc1linear = firfilt(1-abs(0.125*(-8:8)),pcp);
tti = 0.125*(0:length(pc1linear)-1) - 1.0;
stem(tti,pc1linear),
hold on, plot(n1,pc1,'pr'), hold off, grid on
```

  For the example above, determine the interpolation ratio when converting `pc1` to `pc1linear`. Determine the impulse response $h[n]$ of the FIR filter and make a plot of $h[n]$.
- Discuss how *causality* comes into play in the example given in the previous part. Specifically, how do you get alignment between the original signal `pc1` and the interpolated result `pc1linear`? Why is the time vector `tti` defined with a -1.0? Note: `firfilt` always implements a causal filter.
- In the case of the `pccat` image, you need to carry out a linear interpolation operation first on the columns and then on the rows of the down-sampled image `pccat5`. Name the interpolated output image `pc5linear`. Include images as well as your code for this part in the lab report.
- Demonstrate that your code is general by processing `pccat10`, the downsampled by 10 image. Explain how you wrote the code so that the interpolation ratio is a parameter, i.e., only one line, or one function call has to be changed when going from $p = 5$ to $p = 10$. For this processing, you need only show the final interpolated images.

3.4 Quality of the Interpolation

Comment on the processed images that you created:

- For both cases ($p = 5$ and 10), compare the linear interpolated result to the original, `pccat`. Comment on the visual appearance of the "reconstructed" images versus the original; point

[10]The MATLAB function `conv2` will perform both the row and column filter with one call, but for this lab you should write the loops for processing all the rows and columns so that you can display the intermediate results.

out differences and similarities. Describe the degradation that the interpolator causes for $p = 10$. Can the reconstruction (i.e., zooming) process remove the aliasing effects from the down-sampled `pccat` image?

- For both cases ($p = 5$ and $p = 10$), compare the quality of the linear interpolation result to the zero-order hold result. Point out regions where they differ and try to justify this difference by estimating the local frequency content. In other words, look for regions of "low-frequency" content and "high-frequency" content and see how the interpolation quality is dependent on this factor.

- Display the `pccat5` image (`pccat` downsampled by 5) and be sure that it is "truesize" by using:

  ```
  show_img(pccat5)
  trusize
  ```

  Now expand the display window to full size so that it fills all of the screen. What type of interpolation is MATLAB doing when it expands a window that displays an image?

Here are a couple of questions to think about: Are edges low frequency or high frequency features? Are the cat's whiskers low frequency or high frequency features? Is the background a low frequency or high frequency feature?

Comment: You might use MATLAB's zooming feature to show details in small patches of the output image. However, be careful because zooming does its own interpolation.

3.5 More about Images in MATLAB (

This section relates these MATLAB operations to previous experience with software such as *Photoshop.* There are many image processing functions in MATLAB. For example, try the help command:

```
help images
```

for more information.

3.5.1 Zooming in Software

If you have used an image editing program such as the GIMP or Adobe's *Photoshop,* you might have observed how well or how poorly image zooming (i.e., interpolation) 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');
```

`imread( )` is part of the the MATLAB core toolbox.

3.5.2 Warnings

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

- 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);
```

You can convert back to 8-bit values with the function `uint8()`.

- 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
```