GEORGIA INSTITUTE OF TECHNOLOGY

SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2025 Fall 2009

Lab #7: FIR Filtering of Digital Images

Date: 13–19 Oct. 2008

You should read the Pre-Lab section of the lab and do all the exercises in the Pre-Lab section before your assigned lab time.

The Warm-up section of each lab must be completed **during your assigned Lab time** and the steps marked *Instructor Verification* must also be signed off **during the lab time** by one of the laboratory instructors. After completing the warm-up section, turn in the verification sheet to your TA.

It is only necessary to turn in Section 4 as the lab report for this lab. More information on the lab report format can be found on `t-square` under the **INFO** link. Please ***label*** the axes of your plots and include a title and Figure number for every plot. In order to reduce *orphan plots*, include each plot as a figure *embedded* within your report. This can be done easily with MATLAB's `notebook` capability. For more information on how to include figures and plots from MATLAB in your report file, consult the **INFO** link on `t-square`, or ask your TA for details.

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 you cannot give or receive written material or electronic files. Your submitted work should be original and it should be your own work.

The lab report for this week will be an **Informal Lab Report.** The report will be **due during the period 20–26 Oct. at the start of your lab.**

1 Introduction

One objective in this lab is to introduce digital images as a second useful signal type. The goal of this lab is to learn how to implement FIR filters in MATLAB, and then study the response of FIR filters to various signals, especially images. As a result, you should learn how filters can create interesting effects such as blurring and echoes. In addition, we will use FIR filters to study the convolution operation and properties such as linearity and time-invariance.

In the experiments of this lab, you will use `firfilt()`, or `conv()`, to implement 1-D filters and `conv2()` to implement two-dimensional (2-D) filters. The 2-D filtering operation actually consists of 1-D filters applied to all the rows of the image and then all the columns.

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. For example, an RGB color system needs three values at each spatial location: one for red, one for green and one for blue. Video or TV which consists of a sequence of images to show motion would add a time variable to the two spatial variables.

[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.

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. which can 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 frame. 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).

An important property of light images such as photographs and TV pictures is that their values are always non-negative and are also 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,[2] but it will be helpful if the following points are noted:

CD-ROM
show_img.m

1. 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).

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 $0 \le x[m,n] \le 255 = 2^8 - 1$.

3. The actual display on the monitor created with the `show_img` function[3] 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, which we will avoid.

4. 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

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

[3]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.

show_img.m to *automatically rescale the image to use the full range of pixel value:* We can do this by applying a linear mapping of the pixel values:[4]

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

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.

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

```
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 formula:

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

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

$$\begin{aligned} y[n] &= \tfrac{1}{3}x[n] + \tfrac{1}{3}x[n-1] + \tfrac{1}{3}x[n-2] \\ &= \tfrac{1}{3}\{x[n] + x[n-1] + x[n-2]\} \end{aligned} \tag{2}$$

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

MATLAB has two built-in functions, conv() and filter(), for implementing the operation in (1), and the *SP-First* toolbox supplies another M-file, called 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 the conv and the firfilt function implement the operation called *convolution*. The following MATLAB statements implement the three-point averaging system of (2):

CD-ROM
firfilt.m

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

```
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 contained in a vector defined by the cosine function. In general, the vector `bb` contains the filter coefficients $\{b_k\}$ needed in (1). The `bb` vector is defined 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 L nonzero samples, we would normally store only the L nonzero samples in a vector, and would assume that $x[n] = 0$ for n outside the interval of L samples, i.e., don't 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), we will find that the output length is:

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

In the experiments of this lab, you will use `firfilt()` to implement FIR filters and begin to understand how the filter coefficients define a digital filtering algorithm. In addition, this lab will introduce examples to show how a filter reacts to different frequency components in the input.

2 Pre-Lab

2.1 MATLAB Function to Display Images

CD-ROM
IMAGE DATA FILES

You can load the images needed for this lab from `*.mat` files, or from `*.png` files. Image files with the extension `.png`, as well as other common formats like JPEG, can be read into MATLAB with the `imread` function. Any file with the extension `*.mat` is in MATLAB's binary format and must 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.

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.

2.2 Get Test Images

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

(a) Load and display the 428×642 "lighthouse" image[5] 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 would be necessary to convert `ww` to double precision floating-point with the MATLAB command `double` if calculations such as filtering are going to be done on `ww`. When you display the image it might be necessary to set the colormap via `colormap(gray(256))`.

(b) 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,:);
```

[5]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.

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 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).[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—in our example, the 300 dpi image would become a 150 dpi image. Usually this is called *sub-sampling* or *down-sampling*.[7]

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 by a factor of `p`. This can be illustrated in a dramatic fashion with the `lighthouse` image; see Section 3.1 in the Warm-up.

2.4 Printing Multiple Images on One Page

The phrase "what you see is what you get" can be elusive when displaying and printing 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 these different size images in separate MATLAB figure windows. In order to get a printout with multiple images on one page, use the following procedure:

1. In MATLAB, use `show_img` and `trusize` to put your images into separate figure windows at the correct pixel resolution.
2. Use a Windows program such as `PAINT` to assemble the different images onto one page. This program can be found under `Accessories.`
3. For each MATLAB figure window, do `ALT-PRINT-SCREEN` which will copy the active window contents to the clipboard.
4. After each "window capture" in step 3, paste the clipboard contents into `PAINT.`[8]

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

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

[8]An alternative is to use the free program called IRFANVIEW, which can do image editing and also has screen capture capability. It can be obtained from `www.irfanview.com`. Other alternatives are Photoshop, or "The GIMP" at `www.gimp.org/windows`.

5. Arrange the images so that you can make a comparison for your lab report.
6. Print the assembled images from `PAINT` to a printer.

2.5 Filtering via Convolution

You can perform the same convolution as done by the **`dconvdemo`** GUI by using the MATLAB function **`firfilt`**, or **`conv`**. For ECE-2025, the preferred function is **`firfilt`**.

(a) For the Pre-Lab, you should do some filtering with a three-point averager. The filter coefficient vector for the three-point averager is defined via:

```
bb = 1/3*ones(1,3);
```

Use **`firfilt`** to process an input signal that is a length-10 pulse:

$$x[n] = \begin{cases} 1 & \text{for } n = 0,1,2,3,4,5,6,7,8,9 \\ 0 & \text{elsewhere} \end{cases}$$

Note: in MATLAB indexing can be confusing. Our mathematical signal definitions start at $n = 0$, but MATLAB starts its indexing at "1". Nevertheless, we can ignore the difference and pretend that MATLAB is indexing from zero, as long as we don't try to write `x[0]` in MATLAB. For this experiment, generate the length-10 pulse and put it inside of a longer vector with the statement `xx = [ones(1,10),zeros(1,5)]`. This produces a vector of length 15, which has 5 extra zero samples appended.

(b) To illustrate the filtering action of the three-point averager, it is informative to make a plot of the input signal and output signal together. Since $x[n]$ and $y[n]$ 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;     %--- use first=1 and last=length(xx)
subplot(2,1,1);
stem(nn-1,xx(nn))
subplot(2,1,2);
stem(nn-1,yy(nn),'filled')   %--Make black dots
xlabel('Time Index (n)')
```

This code assumes that the output from `firfilt` is called `yy`. 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. In other words, the plotting range for both signals will be equal to the length of the input signal, even though the output signal is longer. Notice that using `nn-1` in the two calls to `stem()` causes the x-axis to start at zero in the plot.

(c) Explain the filtering action of the three-point averager by comparing the plots in the previous part. This averaging filter might be called a "smoothing" filter, especially when we see how the transitions in $x[n]$ from zero to one, and from one back to zero, have been "smoothed."

3 Warm-up

The instructor verification sheet may be found at the end of this lab.

3.1 Sample an Image and Observe Aliasing

Perform this operation on the `lighthouse.png` image.

(a) 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.

Instructor Verification (separate page)

3.2 Filtering Images: 2-D Convolution

One-dimensional FIR filters, such as running averagers and first-difference filters, can be used to process one-dimensional signals such as speech or music. 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 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.

One objective of this lab is to show how simple 2-D filtering can be accomplished with 1-D row filters and/or column filters. It might be tempting to use a `for` loop when writing 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:

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

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) Load in the image `echart.mat` (from the *SP-First* Toolbox) with the `load` command. This will create the variable `echart` whose size is 257×256. 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(echart, bdiffh);
```

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

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.

3.3 Synthesize a Test Image (Optional)

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.

(a) Generate a simple test image in which all of the columns are identical by using the following *outer product* to generate a 256×256 matrix from a column vector times a row vector:

```
xpix = ones(256,1)*cos(2*pi*(0:255)/16);
```

Display the image and explain the gray-scale pattern that you see. How wide are the bands in number of pixels? How can you predict that width from the formula for `xpix`?

(b) In the previous part, which data value in `xpix` is represented by white? which one by black?

(c) *Optional:* 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.

4 Lab: 2D FIR Filtering of Images

In the following sections we will study how an FIR filter can perform *Edge Detection* or edge enhancement. A first-difference FIR filter will have zero output when the input signal is constant, but a nonzero output when the input changes, so we can use such a filter to find edges in an image.

4.1 Edge Detection via 1-D Filters

Use the function `firfilt()` to implement the "first-difference" FIR filter: $w[n] = x[n] - x[n-1]$ on the input signal $x[n]$ defined via the MATLAB statement:

```
xx = 255*(floor( rem( cumsum( 0.3*rand(1,99) ), 3) ) );
```

In order to do the first-difference filter in MATLAB, you must define the vector of filter coefficients `bb` needed for `firfilt`.

(a) Plot both the input and output waveforms $x[n]$ and $w[n]$ on the same figure, using `subplot`. Make the discrete-time signal plots with MATLAB's `stem` function. Explain why the output appears the way it does by figuring out (mathematically) the effect of the first-difference operator on this signal.

(b) Note that $w[n]$ and $x[n]$ are not the same length. Determine the length of the filtered signal $w[n]$, and explain how its length is related to the length of $x[n]$ and the length of the FIR filter.

(c) The *edges* in a 1-D signal are the transitions. If you only want an indicator for the edges, then you must define an additional system whose output is 1 (true) at the exact edge location, and 0 (false) otherwise. *However, an edge really lies between two samples, so you should mark both sides as true.* With $w[n]$ as the input to this new system, use the `abs` function along with a logical operator (such as $>$ or $<$) to define this new system that gives a "TRUE" binary output for both sides of the edges.
Hint: If you have a MATLAB vector that is mostly zeros with a few isolated ones, e.g., `ww=[0,0,1,0,0,0,0,1,0,0,1]`, then you can repeat the ones, i.e., `yy=[0,1,1,0,0,0,1,1,0,1,1]` by using MATLAB's logical-OR function, `yy=or(ww,[ww(2:end),0])`.

(d) Is the *edge-marking* system defined in the previous part a linear or nonlinear system? Explain.

4.2 2-D Edge Detection using Parallel 1-D Filters Cascaded with a Nonlinear Detector

More complicated systems are often made up from simple building blocks. In the system of Fig. 1, two 1-D FIR filters process the columns and rows separately. Then a second system does detection by using a threshold on the absolute value of the filtered output.

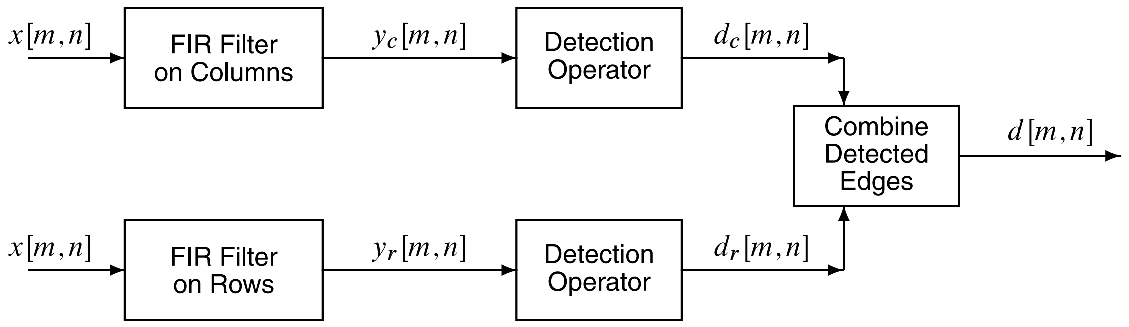

Figure 1: Using two FIR systems (plus detectors) in parallel to perform edge detection.

4.3 Edge Detection for Images

For this section, the `SuperK.png` image should be read in with the `imread` command and used for processing. You can get the `SuperK` image from the `lab07f09dat.zip` file.

(a) *Scaling:* Scale the image so that its maximum value is one. This can be accomplished by dividing all pixels by the largest amplitude pixel. Use `max` to find the largest.

(b) Use 1-D *First-Difference* filters to process the scaled `SuperK` image. Implement each 1-D filter as a filter along either the rows or the columns of the image. Call the results `skFIRrow` and `skFIRcol`. Display the absolute value[10] of *both* of these images, making sure to call `show_img` so that it rescales the image back to the range ($0 \rightarrow 255$) for display.

(c) Convert both filtered images, `skFIRrow` and `skFIRcol`, into binary images (consisting of zeros and ones) by using the absolute value and logical operations as was done in Section 4.1. For each pixel, the logical operator must compare the pixel value to a fixed threshold to determine whether or not there is an edge at that pixel location. For example,

$$d[m,n] = \begin{cases} \text{Edge true if} & |y[m,n]| \geq \tau \\ \text{Edge false if} & |y[m,n]| < \tau \end{cases}$$

Determine an appropriate value of the threshold τ to get the correct edges.

(d) Display the results $d_r[m,n]$ and $d_c[m,n]$ so that an edge location displays as black, and everything else is white. This requires that TRUE be black, and FALSE white. Furthermore, call the image display function `show_img` so that it rescales the binary (0-1) image to occupy eight bits ($0 \rightarrow 255$).

(e) Combine the two separate images into one. Since $d_r[m,n]$ and $d_c[m,n]$ are binary images, the combination should be done with a logical operator (e.g., OR, AND, etc.). Display the result $d[m,n]$ so that an edge location displays as black, and everything else is white.

(f) Comment on why this processing works almost perfectly in finding edges in the `SuperK` image.

[10]In order to save toner in the your printer, use `colormap(1-gray(256))` to display a "negative image." Then the edges will display as dark and the background as white.

4.3.1 Edge Detection for a Gray-Scale Image

In this section, a different image will be used: the `cardioAngio` image in `cardioAngio.jpg` which can be read into MATLAB with the `imread` function. This image shows the major arteries of a human heart obtained during an angiogram which uses x-rays to acquire and form the image.

(a) Use the `max` function to determine the largest value in the `cardioAngio` image. Then, scale the `cardioAngio` image to have a maximum value of one prior to the filtering.

(b) Apply the edge detection system developed in Section 4.3 to the `cardioAngio` image, and show the result. Adjust the threshold τ to get the best possible result.

(c) Display $d[m,n]$ which is the output of the edge detector. Make sure to call `show_img` so that it rescales $|y[m,n]|$ back to the range $(0 \rightarrow 255)$ when doing the display.
Note: if your edge detection system uses the *edge-marking* operator form Section 4.1(d), then turn it off and see whether you can get a better result; if so, use it it for the next part.

(d) Comment on the good and bad aspects of the edge detection system for this gray-scale image. Explain why it is not possible to get just the desired edges without some extra detections of other small features in the final image.

Final Comment: Include all images and plots for the previous parts to support your discussions in the lab report.

4.4 More about Images in MATLAB (Information Only)

This section is included for those students who might want to relate MATLAB's capabilities 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, but keep in mind that the Image Processing Toolbox, which is available in the ECE labs, may not be on your computer.

Images obtained from JPEG files that use color are 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( )`. Finally, prior to doing any processing the datatype must converted from `uint8` to `double`, which can be done with `xx=double(xx)`.

Lab #7
ECE-2025 Fall-2009
WORKSHEET & VERIFICATION PAGE

For each verification, be prepared to explain your answer and respond to other related questions that the lab TA's or professors might ask. Turn this page in at the end of your lab period.

Name: ______________________________ Date of Lab: __________

Part 3.1(a) Downsample the `lighthouse` image to see aliasing. Describe the aliasing, point out where it occurs in the image, and explain why it occurs.

Verified:____________________ Date/Time:__________

Part 3.2(a) Process the input image `echart` with a 2-D filter that filters along the horizontal direction with a first difference filter. Explain how the filter changes the "image signal."

Verified:____________________ Date/Time:__________

$\Longrightarrow$ Answered ITS questions:

Verified:____________________ Date/Time:__________ Completed Peer Evlauation?