GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2025 Spring 2003
Lab #6: FIR Filtering of Images

Date: 18 – 24 Feb 2003

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. You **MUST** complete the online Pre-Post-Lab exercise on Web-CT at the beginning of your scheduled lab session. You can use MATLAB and also consult your lab report or any notes you might have, but you cannot discuss the exercises with any other students. You will have approximately 20 minutes at the beginning of your lab session to complete the online Pre-Post-Lab exercise. The Pre-Post-Lab exercise for this lab includes some questions about concepts from the previous Lab report as well as questions on the Pre-Lab section of this lab.

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**. After completing the warm-up section, turn in the verification sheet to your TA.

It is only necessary to turn in Section 3 as this week's lab 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 lab report for this week will be an **Informal Lab Report.**

The report will **due the next time your lab meets: 25–27 February, or 10-Mar at the start of your lab**.

---

1 Pre-Lab

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, including images or speech. 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 Two GUIs

This lab involves the use of two MATLAB GUIs: one for sampling and aliasing and one for convolution.

1. **`con2dis:`** GUI for sampling and aliasing. An input sinusoid and its spectrum is tracked through A/D and D/A converters.

2. **`dconvdemo:`** GUI for discrete-time convolution. This is exactly the same as the MATLAB functions `conv()` and `firfilt()` used to implement FIR filters.

Both of these demos can be downloaded from WebCT for ECE2025. From the Homepage, click "Resources". Then click "MATLAB GUIs". Then go to the two demos and click on the download statement.

1.2 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 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 a 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, the 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

```
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, L samples, we would normally only store the L samples in a vector, and would assume that $x[n] = 0$ for n outside the interval of L 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), we will find that

```
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.

1.3 Pre-Lab: Run the GUIs

The first objective of this lab is to demonstrate usage of the two GUIs. First of all, you must download the ZIP files for each and install them. Each one installs as a directory containing a number of files. You can put the GUIs on the `matlabpath`, or you can run the GUIs from their home directories.

1.4 Sampling and Aliasing Demo

In this demo, you can change the frequency of an input signal that is a sinusoid, and you can change the sampling frequency. The GUI will show the sampled signal, $x[n]$, its spectrum, and also the reconstructed output signal, $y(t)$ with its spectrum. Figure 1 shows the interface for the **`con2dis`** GUI. In order to see the entire GUI, you must select **`Show All Plots`** under the **`Plot Options`** menu.

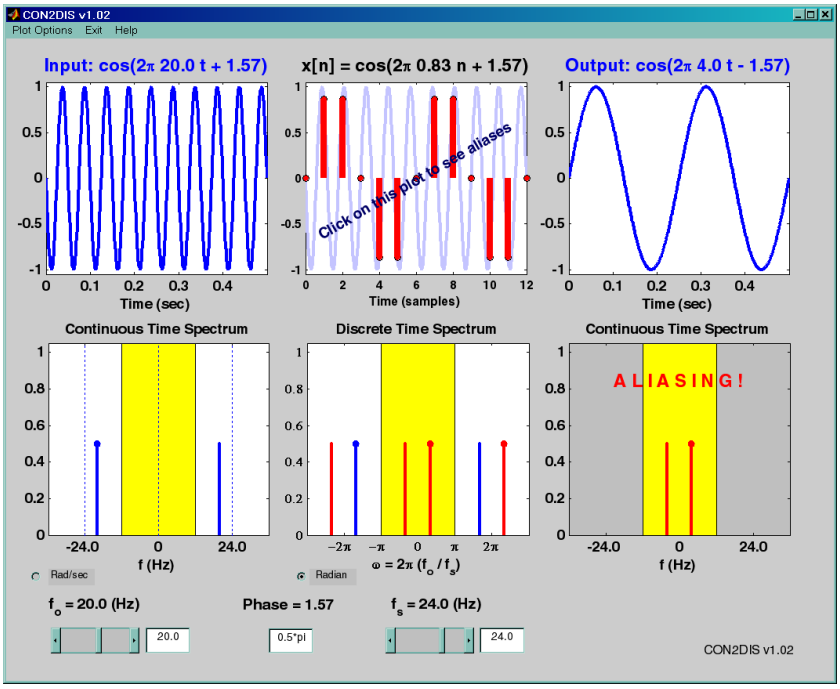

Figure 1: The **`con2dis`** MATLAB GUI interface.

In the pre-Lab, you should perform the following steps with the **`con2dis`** GUI:

(a) Set the input to $x(t) = \cos(40\pi t + 0.5\pi)$

(b) Set the sampling rate to $f_s = 24$ samples/sec.

(c) Determine the locations of the spectrum lines for the discrete-time signal, $x[n]$, found in the middle panels. Make sure that the **`Radian`** button is active so that the frequency axis for the discrete-time signal is $\hat{\omega}$.

(d) Determine the formula for the output signal, $y(t)$ shown in the rightmost panels. What is the output frequency in Hz?

1.5 Discrete-Time Convolution Demo

In this demo, you can select an input signal $x[n]$, as well as the impulse response of the filter $h[n]$. Then the demo shows the ***sliding window*** view of FIR filtering. In this view, one of the signals must be *flipped and shifted* along the axis when convolution is computer. Figure 2 shows the interface for the **`dconvdemo`** GUI.

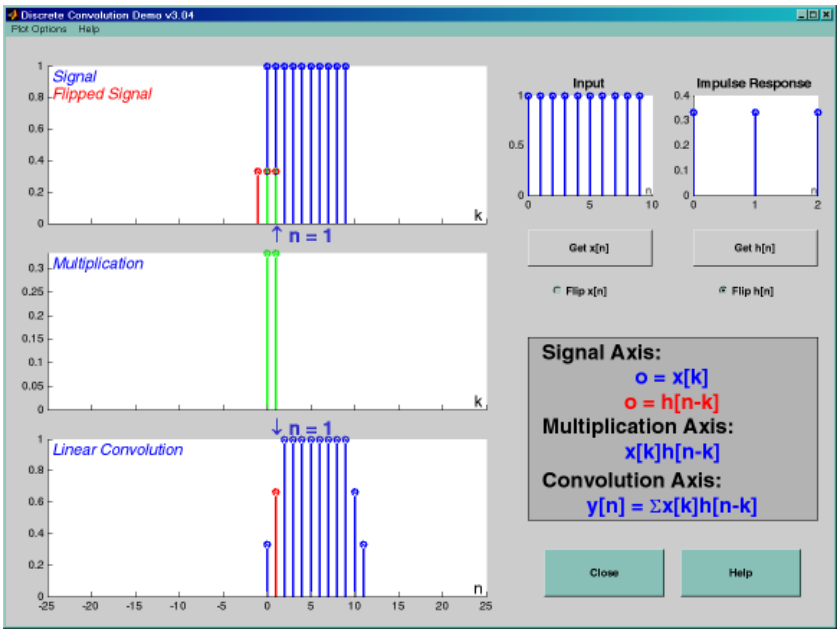

Figure 2: Interface for discrete-time convolution GUI called **`dconvdemo`**.

In the pre-lab, you should perform the following steps with the **`dconvdemo`** GUI.

(a) Click on the **`Get x[n]`** button and set the input to a finite-length pulse: $x[n] = (u[n] - u[n-10])$. Note the length of this pulse.

(b) Set the filter to a three-point averager by using the **`Get h[n]`** button to create the correct impulse response for the three-point averager. Remember that the impulse response is identical to the b_k's for an FIR filter. Also, the GUI allows you to modify the length and values of the pulse.

(c) Observe that the GUI produces the output signal.

(d) When you move the mouse pointer over the index "n" below the signal plot and do a click-hold, you will get a *hand tool* that allows you to move the "n"-pointer. By moving the pointer horizontally you can observe the sliding window action of convolution. You can even move the index beyond the limits of the window and the plot will scroll over to align with "n."

1.6 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 the filtering with a 3-point averager. The filter coefficient vector for the 3-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 3-point averager, it is informative to make a plot of the input signal and output signals 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 call to `stem( )` causes the x-axis to start at zero in the plot.

(c) Explain the filtering action of the 3-point averager by comparing the plots in the previous part. This filter might be called a "smoothing" filter. Note how the transitions in $x[n]$ from zero to one, and from one back to zero, have been "smoothed."

1.7 Vector Logicals

In MATLAB, logical operations are always vector (or array) operations. Analyze the following line of code to understand how a logical comparison returns an array of ones (true) and zeros (false).

```
xx = exp(j*pi/7*(0:14)), tau = 1/2, yy = abs(real(xx))>tau
```

2 Warm-up

2.1 Sampling and Aliasing

Use the `con2dis` GUI to do the following problem:

(a) Input frequency is 12 Hz; input phase is $\phi = -\pi/3$.

(b) Sampling frequency is 14 Hz.

(c) Determine the frequency and phase of the reconstructed output signal

(d) Determine the locations in $\hat{\omega}$ of the lines in the spectrum of the discrete-time signal. Give precise numerical values.

(e) Change the sampling frequency to 10 Hz, and explain the appearance of the output signal.

Instructor Verification (separate page)

2.2 Discrete-Time Convolution

In this section, you will generate filtering results needed in a later section. Use the discrete-time convolution GUI, **`dconvdemo`**, to do the following:

(a) Set the input signal to be $x[n] = (0.9)^{n-4}\,(u[n-12] - u[n-4])$. Use the "Exponential" signal type and the "Delay" feature within **`Get x[n]`**.

(b) Set the impulse response to be $h[n] = \delta[n-1] - 0.9\delta[n-2]$. Once again, use the "Exponential" signal type within **`Get h[n]`**.

(c) Illustrate the output signal $y[n]$ and explain why it is zero for almost all points. Compute the numerical value of the last point in $y[n]$, i.e., the one that is nonzero.

Instructor Verification (separate page)

2.3 Loading Data

In order to exercise the basic filtering function `firfilt`, we will use some "real" data. In MATLAB you can load data from a file called `lab6dat.mat` file by using the `load` command as follows:

```
load lab6dat
```

`lab6dat.mat`

The data file `lab6dat.mat` contains two filters and three signals, stored as separate MATLAB variables:

- `x1:` a stair-step signal such as one might find in one sampled scan line from a TV test pattern image.
- `xtv:` an actual scan line from a digital image.
- `x2:` a speech waveform ("oak is strong") sampled at $f_s = 8000$ samples/second.
- `h1:` the coefficients for a FIR discrete-time filter of the form of (1).
- `h2:` coefficients for a second FIR filter.

After loading the data, use the `whos` function to verify that all five vectors are in your MATLAB workspace.

2.4 Filtering a Signal

You will now use the signal vector `x1` as the input to an FIR filter.

(a) For the warm-up, you should do the filtering with a first-difference FIR filter. Define the filter coefficient vector `bb` for the first difference filter and the use it in `firfilt` to process `x1`. How long are the input and output signals?

When unsure about a command, use `help`.

(b) To illustrate the filtering action of the first difference filter, you must make a plot of the input signal and output signal together (as shown in Section 1.6). The plotting range for both signals should be set equal to the length of the input signal, even though the output signal is longer.

(c) Since the previous plot is quite crowded, it is useful to show a small part of the signals. Repeat the previous part with the plotting interval chosen to display 30 points from the middle of the signals.

(d) Explain the filtering action of the first-difference filter by comparing the plots from parts (b) and (c). Note how the transitions from one level to another have been "enhanced."

Instructor Verification (separate page)

2.5 Filtering Images: 2-D Convolution

One-dimensional FIR filters, such as running averagers and first-difference filters, can be applied to 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 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.

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. 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 will be very helpful in this lab.

(a) Load in the image `echart.mat` with the `load` command (it 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 `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.

(b) Now filter the "eye-chart" image `echart` in the *vertical* direction with a first-difference filter to produce the image `yy2`. This is done by calling `yy2 = conv2(echart,bdiffh')` with a column vector of filter coefficients. Display the image `yy2` on the screen and describe in words how the output image compares to the input.

3 Lab: FIR Filters

In the following sections we will study how a filter can produce the following special effects:

1. *Edge Detection:* a first-difference FIR filter will have zero output when the input signal is constant, but a large output when the input changes, so we can use such a filter to find edges in an image.

2. *Echo:* FIR filters can produce echoes and reverberations because the filtering formula (1) contains delay terms. In an image, such phenomena would be called "ghosts."

3.1 Edge Detection for 1-D Filters

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

```
xx = 128*((rem(0:100,30)<20)-1);
```

In order to do this in MATLAB, you must define the vector of filter coefficients `bb` needed in `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 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. (If you need a hint refer to Section 1.2.)

(c) The *edges* in a 1-D signal are the transitions. If you only want an indicator for the edges, then you would like to define an additional system whose output is either 1 (true) or 0 (false) at the exact edge location. 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 the edges. Make sure that you have perfect alignment on the n-axis.

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

3.2 An Echo Filter

The following FIR filter can be interpreted as an echo filter.

$$y_1[n] = x_1[n] + r\, x_1[n - P] \tag{3}$$

Learn why this is a valid interpretation by working out the following:

(a) If you have an audio signal sampled at $f_s = 8$ kHz, then you can add a delayed version of the signal to simulate an echo. Suppose the time delay of the echo is 300 msec, and the echo amplitude is 95% percent of the original. Determine the values of r and P in (3); make P an integer.

(b) Describe the filter coefficients of this FIR filter, and determine its length.

(c) Implement the echo filter in (3) with the values of r and P determined in part (a). Use the speech signal in the vector `x2` found in the file `lab6dat.mat`. Listen to the result to verify that you have produced an audible echo.

(d) Reduce the echo time, in steps from 300 msec down to zero, in an effort to determine the shortest echo time that can be perceived by human hearing. Give your value of the "shortest perceptible echo time" in milliseconds.

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

More complicated systems are often made up from simple building blocks. In the system of Fig. 3, two 1-D FIR filters perform a 2-D FIR filter along the columns and rows. Then a second system uses a threshold on the absolute value of the filtered output.

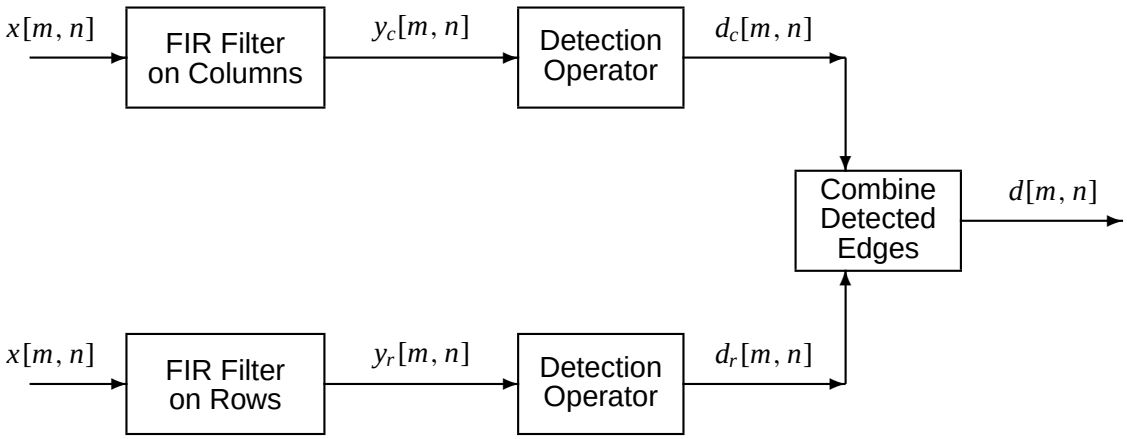

Figure 3: Using two FIR systems in parallel to perform edge detection.

3.4 Edge Detection for Images

For this section, the `echart` image should be loaded and used for processing.

(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 `echart` image. Implement each 2-D filter as a 1-D filter along either the rows or the columns of the image. Call the results `echFIRrow` and `echFIRcol`. Display the absolute value[1] 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, `echFIRrow` and `echFIRcol`, into a binary images (consisting of zeros and ones) by using the absolute value and logical operations as was done in Section 3.1. 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 the 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 well in finding edges in the `echart` image.

3.4.1 Another Edge Enhancement

In this section, a different image will be used: the `lighthouse` image, which can be read into MATLAB with the `imread` function.

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

(a) Scale the lighthouse image to the range [0, 1] prior to the filtering.

(b) Apply the edge detection system developed in Section 3.4 to the lighthouse 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.

(d) Comment on how the edge detection system works by comparing the two images that you have processed. In addition, describe the characteristics of images for which you think this edge detector would give excellent results.

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

Lab #6
ECE-2025
Spring-2003
INSTRUCTOR 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 2.1: Demonstrate that you can run the `con2dis` GUI. Calculate the locations of the spectrum lines for the discrete-time signal. Write the precise values of the $\hat{\omega}$ and the output phase below for both cases: $f_s = 14$ Hz and $f_s = 10$ Hz.

Verified:_____ Date/Time:_____

Part 2.2: Demonstrate that you can run the `dconvdemo` GUI. Explain why the output signal is zero for most values of n.

Verified:_____ Date/Time:_____

Part 2.4(b,c,d) Process the input signal `x1` with a first-difference filter by using `firfilt.m`. Display 30 points from the middle of the input and output signals.

Verified:_____ Date/Time:_____