GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2026 Spring 2025
Lab #5: FIR Filtering and Frequency Responses

Date: 10 – 28 March 2025

Special Lab Instructions in Spring 2025: The labs will be held in person (or remotely through BlueJeans when needed). Students registered in a particular lab session are required to be present at the designated times. Attendances will be taken before each class. A total of six labs will be conducted. Each lab will typically last two weeks. The first week is devoted to students' **Q&As** and the second is for students' **demos** to the instructors for codes and lab results. Students will be grouped into teams of 4 to 5 by instructors before starting Lab 1. For each lab session, there will be two instructors administrating all activities. Each team will be given a breakout period of 15 minutes in each session for Q&As and verifications. Students are encouraged to discuss lab contents in teamwork. At the end of each lab, each student are required to turn in an individual lab report in a single pdf fill, containing answers to all lab questions, including codes and plots. Georgia Tech's **Honor Code** will be strictly enforced. See CANVAS Assignments for submission instructions.

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, but you cannot give or receive any written material or electronic files. In addition, you are not allowed to use or copy material from old lab reports from previous semesters. Your submitted work must be your own original work.

1 Introduction

The first objective in this lab to learn showing how interpolation is used to create color images for a digital camera. For the interpolation, you will learn how to implement FIR filters in MATLAB, and then use these FIR filters to perform the interpolation of images.

The second objective of this lab is to study the frequency response. For FIR filters this is the response to inputs such as complex exponentials and sinusoids. Although you can use `firfilt()`, or `conv()`, to implement filters in the *time domain*, the function `freqz()` is used to characterize the filter in the *frequency domain* via its frequency response.[1] As a result, you will learn how to obtain the output when the input is a sum of sinusoids because the frequency response characterizes a filter by telling how the filter responds to different frequency components in the input.

1.1 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 system for which the output values are given by

$$\begin{aligned} y[n] &= \tfrac{1}{3}x[n] + \tfrac{1}{3}x[n-1] + \tfrac{1}{3}x[n-2] \qquad (2)\\ &= \tfrac{1}{3}\left\{x[n] + x[n-1] + x[n-2]\right\} \end{aligned}$$

[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 find it in the *SP-First Toolbox*.

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 $b_2 = \frac{1}{3}$.

CD-ROM
`firfilt.m`

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):

```
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

```
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.

1.2 Frequency Response of FIR Filters

The output or *response* of a filter for a complex sinusoid input, $e^{j\hat{\omega}n}$, is a complex exponential at the same frequency. The magnitude and phase of the output will be different, and that change depends on the frequency, $\hat{\omega}$. The dependence of these magnitude and phase changes versus frequency is called the *frequency response*. In effect, the filter is described by how it affects different input frequencies. For example, the frequency response of the two-point averaging filter

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

can be found by using a general complex exponential as an input $x[n]$ and observing the output or *response*.

$$x[n] = Ae^{j(\hat{\omega}n + \varphi)} \tag{3}$$

$$y[n] = \tfrac{1}{2}Ae^{j(\hat{\omega}n + \varphi)} + \tfrac{1}{2}Ae^{j(\hat{\omega}(n-1) + \varphi)} \tag{4}$$

$$= Ae^{j(\hat{\omega}n + \varphi)}\left\{\tfrac{1}{2} + \tfrac{1}{2}e^{-j\hat{\omega}}\right\} \tag{5}$$

In (5) there are two terms, the original input, and a term that is a function of $\hat{\omega}$. This second term is the frequency response and it is commonly denoted[2] by $H(e^{j\hat{\omega}})$.

$$H(e^{j\hat{\omega}}) = \mathcal{H}(\hat{\omega}) = \tfrac{1}{2}\left\{1 + e^{-j\hat{\omega}}\right\} \tag{6}$$

[2]The notation $H(e^{j\hat{\omega}})$ is used in place of $\mathcal{H}(\hat{\omega})$ for the frequency response because we will eventually connect this notation with the `z-transform`, `Hz`, in Chapter 7.

The general form of the frequency response for an M-th order FIR linear time-invariant system with filter coefficients $\{b_k\}$ is

$$H(e^{j\hat{\omega}}) = \mathcal{H}(\hat{\omega}) = \sum_{k=0}^{M} b_k e^{-j\hat{\omega}k} \tag{7}$$

Once the frequency response, $H(e^{j\hat{\omega}})$, has been determined, the effect of the filter on any complex exponential input may be determined by evaluating $H(e^{j\hat{\omega}})$ at the corresponding input frequency. The output signal, $y[n]$, will be a complex exponential whose complex amplitude has a constant magnitude and phase. The phase of $H(e^{j\hat{\omega}})$ describes the phase change of the complex sinusoid and the magnitude of $H(e^{j\hat{\omega}})$ describes the "gain" applied to the complex sinusoid.

2 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.

2.1 Discrete-time Convolution GUI

This lab involves the use of a MATLAB GUI for convolution of discrete-time signals, `dconvdemo`. This is exactly the same as the MATLAB functions `conv()` and `firfilt()` used to implement FIR filters. This demo is part of the *SP-First* Toolbox.

2.2 Discrete-Time Convolution Demo

The first objective of this lab is to demonstrate usage of the `dconvdemo` GUI. If you have installed the *SP-First* Toolbox, you will already have this demo on the `matlabpath`. 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, where one of the signals must be *flipped and shifted* along the axis when convolution is computed. Figure 1 shows the interface for the `dconvdemo` GUI.
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 in the bottom panel.

(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 to the left or right; or you can use the left and right arrow keys. 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."

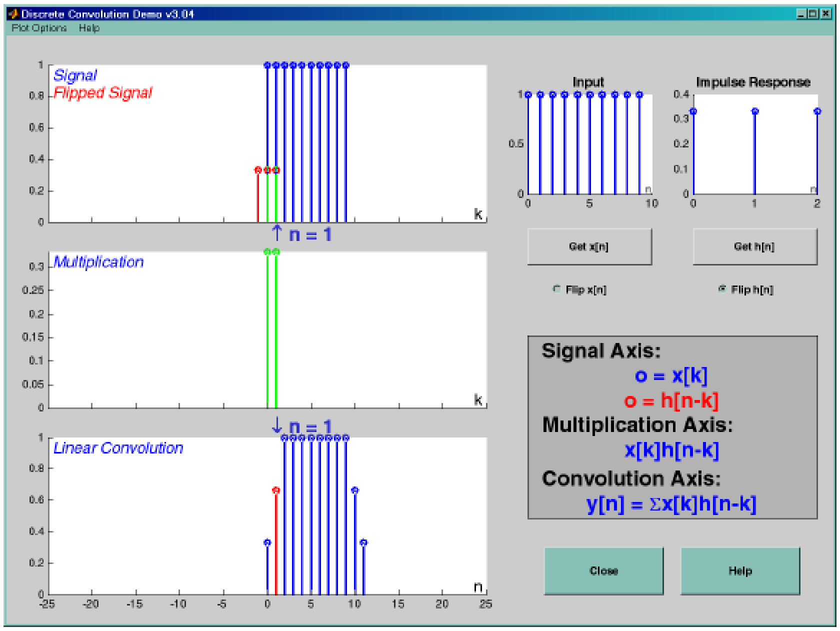

Figure 1: Interface for discrete-time convolution GUI called `dconvdemo`. This is the convolution of a three-point averager with a ten-point rectangular pulse.

2.3 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-2026, 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."

2.4 FIR Nulling Filters

A simple second-order FIR filter can be used to remove a sinusoid from an input signal. The general form for the filter coefficients of an FIR nulling filter is

$$b_0 = 1 \qquad b_1 = -2\cos(\hat{\omega}_{\text{NULL}}) \qquad b_2 = 1 \tag{8}$$

Nulling means that the frequency response will be zero at $\hat{\omega} = \hat{\omega}_{\text{NULL}}$. With $\hat{\omega}_{\text{NULL}} = 0.75\pi$, enter the filter coefficients in the `dltidemo` GUI to see the frequency response; verify that it is zero at $\hat{\omega} = 0.75\pi$. In addition, define the input to be $x[n] = 0.4 + 1.5\cos(0.75\pi n)$ and observe that the sinusoidal component is not present in the output.

2.5 MATLAB Function for Frequency Response

The second goal of this lab is to learn about frequency response of FIR filters in MATLAB, and then study the response of FIR filters to various signals. MATLAB has a built-in function for computing the frequency response of a discrete-time LTI system. The following MATLAB statements show how to use `freqz` to compute and plot both the magnitude (absolute value) and the phase of the frequency response of a two-point averaging system as a function of $\hat{\omega}$ in the range $-\pi \le \hat{\omega} \le \pi$:

```
bb = [0.5, 0.5];        %-- Filter Coefficients
ww = -pi:(pi/100):pi;   %-- omega hat frequency vector
H = freqz(bb, 1, ww);   %<--freekz(bb,1,ww) is an alternative
subplot(2,1,1);
plot(ww, abs(H)), grid on
subplot(2,1,2);
plot(ww, angle(H)), grid on
xlabel('Normalized Radian Frequency')
```

For FIR filters, the second argument of `freqz( _, 1, _ )` must always be equal to `1`. The frequency vector `ww` should cover an interval of length 2π for $\hat{\omega}$, and its spacing must be fine enough to give a smooth curve for $H(e^{j\hat{\omega}})$. Note: we will always use capital `H` for the frequency response.[3]

2.6 Periodicity of the Frequency Response

The frequency responses of discrete-time filters are *always* periodic with period equal to 2π. Explain why this is the case by using the definition of the frequency response (7) and then considering two input sinusoids whose frequencies are $\hat{\omega}$ and $\hat{\omega} + 2\pi$.

$$x_1[n] = e^{j\hat{\omega}n} \qquad \text{versus} \qquad x_2[n] = e^{j(\hat{\omega} + 2\pi)n}$$

It should be easy to prove that $x_2[n] = x_1[n]$. Consult Chapter 6 for a mathematical proof that the outputs from each of these signals will be identical. **The implication of periodicity is that a plot of $H(e^{j\hat{\omega}})$ only has to be made over the interval $-\pi \le \hat{\omega} \le \pi$.**

[3] If the output of the `freqz` function is not assigned, then plots are generated automatically; however, the magnitude is given in decibels which is a logarithmic scale. For linear magnitude plots a separate call to `plot` is necessary.

2.7 Frequency Response of the Four-Point Averager

In Chapter 6 we examined filters that compute the average of input samples over an interval. These filters are called "running average" filters or "averagers" and they have the following form for the L-point averager:

$$y[n] = \frac{1}{L}\sum_{k=0}^{L-1} x[n-k] \tag{9}$$

(a) Use Euler's formula and complex number manipulations to show that the frequency response for the 4-point running average operator can be written as:

$$H(e^{j\hat{\omega}}) = \mathcal{H}(\hat{\omega}) = \left(\frac{2\cos(0.5\hat{\omega}) + 2\cos(1.5\hat{\omega})}{4}\right) e^{-j1.5\hat{\omega}} = C(\hat{\omega})e^{j\psi(\hat{\omega})} \tag{10}$$

(b) Implement (10) directly in MATLAB. Use a vector that includes 400 samples covering the interval $[-\pi, \pi)$ for $\hat{\omega}$. Make plots of $C(\hat{\omega})$ and $\psi(\hat{\omega})$ versus $\hat{\omega}$. It is tempting to think that $C(\hat{\omega})$ is the magnitude of the frequency response, but $C(\hat{\omega})$ can go negative, so these are not plots of the magnitude and phase. You would have to use `abs()` and `angle()` to extract the magnitude $|H(e^{j\hat{\omega}})|$ and phase $\angle H(e^{j\hat{\omega}})$ of the frequency response for plotting.

(c) In this part, use `freqz.m` or `freekz.m` in MATLAB to compute $H(e^{j\hat{\omega}})$ numerically (from the filter coefficients) and plot its magnitude and phase versus $\hat{\omega}$. Write the appropriate MATLAB code to plot both the magnitude and phase of $H(e^{j\hat{\omega}})$. Follow the example in Section 2.5. The filter coefficient vector for the 4-point averager is defined via:

```
bb = 1/4*ones(1,4);
```

Recall that the function `freqz(bb,1,ww)` evaluates the frequency response for all frequencies in the vector `ww`. It uses the summation in (7), not the formula in (10). The filter coefficients are defined in the assignment to vector `bb`. How do your results compare with part (b)?
Note: the plots should not be identical, but you should be able to explain why they are equivalent by converting the minus sign in the negative values of $C(\hat{\omega})$ to a phase of π radians, which then modifies the phase plot with jumps of $\pm\pi$ radians.

3 In-Lab Exercises

3.1 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) The convolution of two impulses, $\delta[n-3] * \delta[n-5]$.

(b) Filter the input signal $x[n] = (-3)\{u[n-2] - u[n-8]\}$ with a first-difference filter. Make $x[n]$ by selecting the "Pulse" signal type from the drop-down menu within `Get x[n]`, and also use the text box "Delay."
Next, set the impulse response to match the filter coefficients of the first-difference. Enter the impulse response values by selecting "User Signal" from the drop-down menu within `Get h[n]`. Illustrate the output signal $y[n]$ and write a simple formula for $y[n]$ which should use only two impulses.

(c) Explain why $y[n]$ from the previous part is zero for almost all n.

(d) Convolve two rectangular pulses: one with an amplitude of 2 and a length of 7, the other with an amplitude of 3 and a length of 4. Make a sketch of the output signal, showing its starting and ending points, as well as its maximum amplitude.

(e) State the *length* and *maximum amplitude* of the convolved rectangles.

(f) The first-difference filter can be used to find the *edges* in a signal or in an image. This behavior can be exhibited with the `dconvdemo` GUI. Set the impulse response $h[n] = \delta[n] - \delta[n-1]$. In order to set the input signal $x[n]$, use the *User Input* option to define $x[n]$ via the MATLAB statement `double((sin(0.5*(0:50))+0.2)<0)`, which is a signal that has *runs* of zero and ones.
The output from the convolution $y[n]$ will have only a few nonzero values. Record the locations of the nonzero values, and explain how these are related to the *transitions* in the input signal. Also, explain why some values of $y[n]$ are positive, and others are negative.

Completion of Lab Results (on separate Report page)

3.2 Filtering Images via 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 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, e.g., to filter the m_0-th row:

$$y_1[m_0, n] = x[m_0, n] - x[m_0, n-1]$$

(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 one row of the image by applying the `conv()` function to one row extracted from the image, `echart(m,:)`.

```
bdiffh = [1, -1];
yy1 = conv(echart(m,:), bdiffh);
```

Pick a row where there are several black-white-black transitions, e.g., choose row number 65, 147, or 221. Display the row of the input image `echart` and the filtered row `yy1` on the screen in the same figure window (with `subplot`. Compare the two stem plots and give a qualitative description of what you see. Note that the polarity (positive/negative) of the impulses will denote whether the transition is from white to black, or black to white. Then explain how to calculate the width of the "E" from the impulses in the stem plot of the filtered row.
Note: Use the MATLAB function `find` to get the locations of the impulses in the filtered row `yy1`.

Completion of Lab Results (on separate Report page)

3.3 FIR Filtering of Images

FIR filters can produce many types of special effects, including:

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. *Deconvolution:* one FIR filter can (approximately) undo the effects of another—we will investigate a cascade of two FIR filters that distort and then restore an image. This process is called *deconvolution.*

In the following sections, we will study how an FIR filter can perform *Edge Detection* as a pre-processing step for measuring the widths of black bars found in the UPC bar codes.

3.3.1 Finding Edges: 1-D Filter Cascaded with a Nonlinear Operators

More complicated systems are often made up from simple building blocks. In the system of Fig. 2, a 1-D FIR filter processes one or more rows; then a second system does detection by using a threshold on the absolute value of the filtered output. If the input row $x[m_0, n]$ is very "blocky" with transitions between two levels, then the output signal, $d[m_0, n]$, should be very *sparse*—mostly zero with only a few nonzero values. In other words, $d[m_0, n]$ can be written as the sum of a small number of shifted deltas (impulses). The *locations of the impulses* correspond to transitions in the input signal from one level to another. In MATLAB the `find` function can extract the locations and produce an output signal $\ell[m_0, n]$ that is dense, i.e., no zeros, because a value like $\ell[m_0, 5]$ is the location of the fifth impulse in $d[m_0, n]$ which is a positive integer.

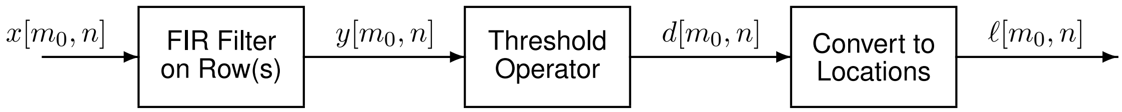

Figure 2: Using an FIR system plus a threshold operator to detect and locate edges, i.e., transitions.

3.3.2 Edge Detection and Location via 1-D Filters

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

```
xx = 255*(rem(1:159,30)>19);
```

Doing the first-difference filter in MATLAB requires that you define the vector of filter coefficients `bb` needed for `firfilt`.

(a) Plot both the input and output waveforms $x[n]$ and $y[n]$ on the same figure, using `subplot`. Make the discrete-time signal plots with MATLAB's `stem` function.

(b) Explain why the output appears the way it does by writing an explicit mathematical formula for the output signal. In other words, justify the effect of the first-difference operator on this input signal.

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

(d) The *edges* in a 1-D signal such as `xx` are the transitions. If you need 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. For example,

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

Determine an appropriate value of the threshold τ to get the edges. In MATLAB use the `abs` function along with a logical operator (such as $>$ or $<$) to define this thresholding system that gives a "TRUE" binary output for the edges, with $y[n]$ as the input to this thresholding system.

(e) Use MATLAB's `find` function to produce a shorter signal that contains the edge locations; make a stem plot of this "signal," and determine its length.
NOTE: you will mark only one side of the transition as true when you threshold the output of the first-difference filter. Is it located before or after the transition?

3.3.3 Bar Code Detection and Decoding

A 12-digit bar code consists of alternating black and white bars; the white bars appear to be spaces. The UPC (Universal Product Code) uses widths of the bars to encode numbers. There are four widths which are integer multiples of the thinnest black bar, or thinnest white space. We will define a 3-wide black bar as three times as wide as the thinnest black bar; likewise, for 2-wide and 4-wide bars—whether black or white. Look at any bar code, and you should be able to identify the four widths.

Each number from 0 to 9 is encoded with a quadruplet. Here is the encoding of the digits 0–9:

```
0 = 3-2-1-1          5 = 1-2-3-1
1 = 2-2-2-1          6 = 1-1-1-4
2 = 2-1-2-2          7 = 1-3-1-2
3 = 1-4-1-1          8 = 1-2-1-3
4 = 1-1-3-2          9 = 3-1-1-2
```

For example, the code for the number "5" is 1-2-3-1, meaning it could be a one-unit wide white space, followed by a 2-wide black bar, followed by a 3-wide white space, and finally a 1-wide black bar (or inverted: 1-wide black, 2-wide white, 3-wide black, and 1-wide white).

The UPC (Universal Product Code) consists of twelve digits delimited on each end by 1-1-1 (black-white-black), and separated in the middle (between the sixth and seventh digit) by white-black-white-black-white (1-1-1-1-1). Thus the entire UPC must have 30 black bars and 29 white bars for a total of 59. Furthermore, note that the encoding for each digit always adds up to seven so the total width of the bar code is always the same. In terms of the unit width where the thinnest bars have width one, it should be easy to determine that the total width of the UPC bar code is 95 units.[4]

3.3.4 Decode the UPC from a Scanned Image

Follow the steps below to develop the processing needed to decode a typical bar code from a scanned image. A decoder for the final step is provided as a MATLAB p-code file. The data files and the decoder for the lab are available from a posted ZIP file.

(a) Read the image `HP110v3.png` into MATLAB with the `imread` function. Extract one row (in the middle) to define a 1-D signal $x[n]$ in the MATLAB vector `xn` for processing.

(b) Filter the signal $x[n]$ with a first-difference FIR filter; call the output $y[n]$. Make a stem plot of the input and output signals, using a subplot to show both in the same figure window.

(c) Create a sparse detected signal $d[n]$ by comparing the magnitude $|y[n]|$ to a threshold. Then convert the sparse signal $d[n]$ into a location signal $\ell[n]$ by using the `find` function to extract locations. Make a stem plot of the location signal, $\ell[n]$.
Note: The length of the location signal must be greater than or equal to 60, if the rest of the processing is going to succeed.

(d) Apply a first-difference filter to the location signal; call the output $\Delta[n]$; these should be the widths of the bars. Make a stem plot of $\Delta[n]$, and put this plot and the previous one of $\ell[n]$ in the same figure window by using `subplot`. Explain how the plot of $\Delta[n]$ conveys the idea that there are (approximately) four different widths in the bar code.

(e) One problem with the idea of having four widths is that the width of the thinnest bar may not be an integer number of pixels. For example, when the basic width is 3.5, we would expect the plot of $\Delta[n]$ to jump between 3 and 4 for 1-wide bars. Such variation will complicate the final decoding, so it is important to estimate the *basic width* (θ_1) of the thinnest bar, and use that to fix $\Delta[n]$.

First of all, prove that the total width of a valid 12-digit bar code is equal to $95\theta_1$. Write a logical argument to justify the total width.

[4] For an example, see `http://electronics.howstuffworks.com/gadgets/high-tech-gadgets/upc3.htm`.

(f) Next, use the fact that a valid bar code has 59 bars to derive a simple method to estimate θ_1 from the signal $\Delta[n]$. Since the length of $\Delta[n]$ will generally be greater than 59, it will be necessary to perform this estimate for every subset of length 59.
Note: The method for estimating θ_1 could also be based on the signal $\ell[n]$.

(g) Using your estimate of θ_1 from the previous part, convert the values of $\Delta[n]$ into relative sizes by dividing by θ_1 and rounding. The result should be integers that are equal to 1, 2, 3 or 4, assuming you are analyzing a valid bar code.

(h) Now you are ready to perform the decoding to digits. A p-code function `decodeUPC` is provided for that purpose. It takes one input vector which has to be a length-59 vector of integers, i.e., the output of the previous part. The function `decodeUPC` has one output which should be a vector of twelve single-digit numbers, if the decoder does not detect an error. When there is an error the output may be partially correct, but the incorrect decodes will be indicated with a `-1`.

(i) For the test image `HP110v3.png` the correct answer is known because it is included at the bottom of the barcode. Check your result.

(j) Another image must also be processed; `OFFv3.png`. In this case, the scan is a bit skewed and the answer is not known. Process this image to extract its UPC from the bar code. The estimate of θ_1 will be different for this case.

3.4 LTI Frequency Response Demo

The second objective of this lab is to use a MATLAB GUI to demonstrate the frequency response, as well as the *sinusoid-in gives sinusoid-out* property of LTI systems. If you are working in the ECE lab it is **NOT** necessary to install the GUI; otherwise, you should install the *SP-First* toolbox. The frequency response demo, `dltidemo`, is part of the *SP-First Toolbox*. The `dltidemo` GUI illustrates the "sinusoid-IN gives sinusoid-OUT" property of LTI systems. In this demo, you can change the amplitude, phase and frequency of an input sinusoid, $x[n]$, and you can change the digital filter that processes the signal. Then the GUI will show the output signal, $y[n]$, which is also a sinusoid (at the same frequency). Figure 3 shows the interface for the `dltidemo` GUI. It is possible to see the formula for the output signal; just click on the `Theoretical Answer` button located at the bottom-middle part of the window. The digital filter can be changed by choosing different `Filter Choice` options in the `Filter Specifications` box in the lower right-hand corner.
Perform the following steps with the `dltidemo` GUI:

(a) Set the input to $x[n] = 1.8\cos(0.2\pi(n-3))$; note that this sinusoid has a positive peak at $n = 3$.

(b) Set the digital filter to be a 7-point averager. Using the middle panels that show the frequency response, measure the magnitude and phase of the frequency response at the input frequency. Right-click to get values from the frequency response plot.

(c) Give an equation that explains how the time delay is related to the phase of $H(e^{j\hat{\omega}})$ at $\hat{\omega} = 0.2\pi$. Remember that phases differing by integer multiples of 2π are the same.

(d) Convert the phase to time-index delay. Then you can determine a formula for the output signal that can be written in the form: $y[n] = A\cos(\hat{\omega}_0(n - n_7))$, where n_7 is an integer. Using n_7 from $y[n]$ and the fact that the input signal had a peak at $n = 3$, it should be easy verify how much the peak of the cosine wave has been shifted. This is called the *time delay* through the filter.

Completion of Lab Results (on separate Report page)

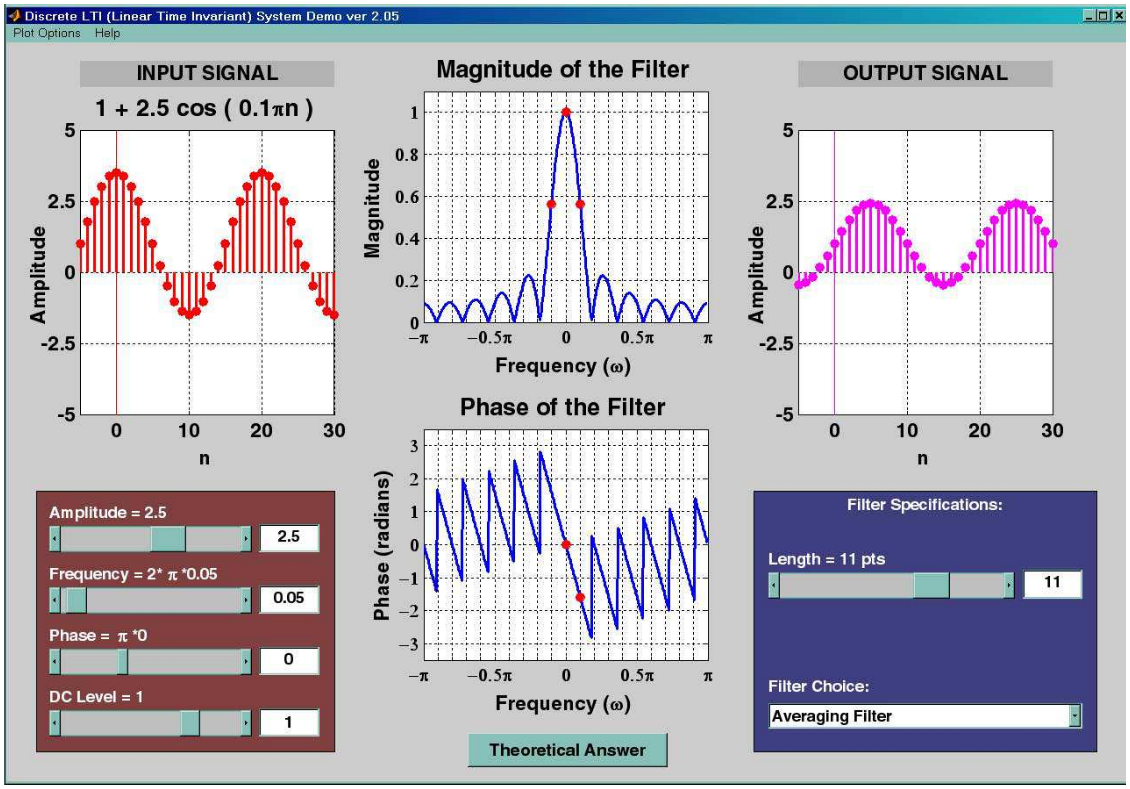

Figure 3: DLTI demo interface. The frequency label should be $\hat{\omega}$, but MATLAB won't display the hat in $\hat{\omega}$.

(e) Now, change the frequency of the input signal so that the output will be exactly zero. Sinusoidal components at other frequencies $\hat{\omega}$ will also be nulled—make a list of these nulled frequencies in the range $0 \leq \hat{\omega} \leq \pi$. There are many choices for this frequency; list them all.
Hint: Recall the Dirichlet form for the frequency response of the averaging filter, and where it has regularly spaced zeros versus $\hat{\omega}$.

(f) When the output of an FIR filter is zero, the FIR filter acts as a *Nulling Filter* for a certain input frequency. Design a second-order nulling filter that will remove the cosine component from the following signal: $x[n] = 1 + \cos(0.4\pi n)$. List the three filter coefficients of the nulling filter. Also, the DC component will not be removed, so determine the DC level of the output signal.

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

Application: Iris Recognition *Biometrics* refers to the use of innate human features for identification, e.g., fingerprints. Often a biometric feature requires significant signal processing to be reduced to a simple code. One new biometric is *Iris Recognition*[a] in which the patterns in the colored part of the eye are encoded and used to verify a person's identity. The first step in this type of system is isolating the iris by finding its boundaries. The figure shows a typical image that would have to be processed. A small amount of glare from the lighting of the photo evident on both sides of the pupil makes the processing harder.

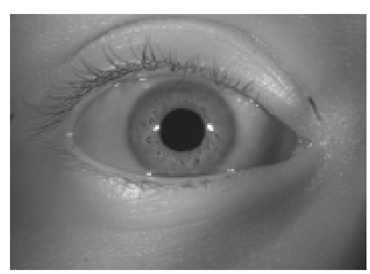

Image of an eye showing that the iris region is bounded by the pupil, eyelids and the white region of the eye.

[a] J.Daugman, Statistical Richness of Visual Phase Information: Update on Recognizing Persons by Iris Patterns, *International Journal of Computer Vision,* 2001.

Lab #5
ECE-2026 Spring-2025
LAB COMPLETION REPORT

Name: ________________________ gtLoginUserName: ____________ Date: ______

Part	Observations (Write down answers for each part)
3.1(a)	Convolve impulses: $\delta[n-3] * \delta[n-5] =$ write formula
3.1(b)	Rectangular Pulse through a First-Difference filter: $y[n] =$ write formula
3.1(c)	Explain why $y[n]$ is zero for most values of n.
3.1(d)	Convolve two rectangles, sketch result; make sure you have the correct beginning and end!
3.1(e)	Maximum Amplitude and Length of the convolved-rectangles output.
3.1(f)	List the locations of the transitions in the output signal, $y[n]$.
3.1(f)	Explain polarity (positive/negative) of the transitions in the output signal, $y[n]$.

Part 3.2(a) Process one row of the input image `echart` with a 1-D first-difference filter. Explain how the output from the filter makes it easy to measure the width of black regions. Use MATLAB's `find` function to help in determining the width of the black "E" from the impulses in the first-difference output.

Part	Observations
3.4(b)	Magnitude and phase of the frequency response of the length-7 averager, at the input frequency.
3.4(c)	Give an equation that explains how the filter's *delay* is related to the phase of $H(e^{j\hat{\omega}})$ at $\hat{\omega} = 0.2\pi$.
3.4(d)	Formula for output from a length-7 averager written in the form $y[n] = A\cos(\hat{\omega}_0(n - n_7))$

Part 3.3.4 is optional: (1) Explain your approach or results; and (2) Turn in your code if required in obtaining your results.

Part 3.3.4 (a): *Explain*.

Part 3.3.4 (b): *Explain*.

Part 3.3.4 (c): *Explain*.

Part 3.3.4 (d): *Explain*.

Part 3.3.4 (e): *Explain*.

Part 3.3.4 (f): *Explain*.

Part 3.3.4 (g): *Explain*.

Part 3.3.4 (h): *Explain*.

Part 3.3.4 (i): did you match the decoded sequence?

Part 3.3.4 (j): what is the decoded sequence?