Fall 2023 EE 313 Linear Systems and Signals UT Austin

Mini-Project #2: Image Restoration

Prof. Brian L. Evans

November 5, 2023, *Version 1.0*

1.0 Introduction (*9 points*)

Humans can interpret a wide of sounds and images. Our ears and eyes sense stimuli and perceive frequencies in sound and images, and pass along their responses as neural firings. The neural firings are interpreted by higher-level functions in the brain known as *cognition*.

In the human auditory system, for example, the cochlea in the inner ear has thousands of hairs that convert vibrations over audible frequencies from 20 Hz to 20 kHz into neural firings. Each hair is tuned to a particular frequency range or band, which is a *bandpass filter* because it responds to frequencies in a particular frequency band but does not respond much to other frequencies. For reference, each key on a piano corresponds to a principal frequency (pitch) that increases from left to right on the keyboard, e.g. from 27.5 to 4186 Hz on an 88-key piano [1].

Similarly, in the human visual system, the retina and Lateral Geniculate Nucleus (LGN) for each eye respond to frequencies in time and space [2]. In space, the LGN detects lines, edges, and corners at horizontal, vertical, and other orientations. These features represented a sudden change from a minimum to maximum value and maximum to minimum value in nearby pixels, which corresponds to high frequencies. The LGN applies a wide range of 2-D lowpass, bandpass, and highpass filters in analyzing visual features in an image.

We can automate audio and image analysis by applying filters in either analog continuous-time circuits or digital discrete-time hardware/software. For example, we could design three bandpass filters to split an audio signal over the audible frequencies into subwoofer, woofer, and tweeter frequency bands, which are approximately 20-200 Hz, 200-2,000 Hz, and 2,000-20,000 Hz. For images, we could design lowpass, bandpass, and highpass filters to split an image into three visible frequency bands.

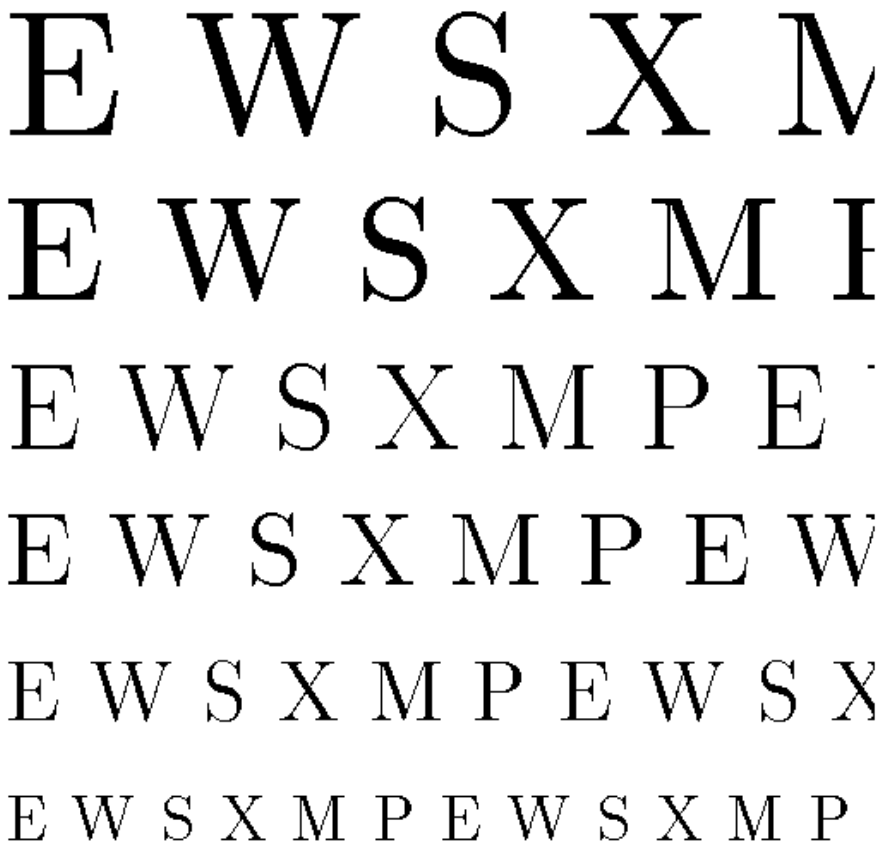
To design, implement, and apply a filter, we can choose a finite impulse response (FIR) or infinite impulse response (IIR) filter [4]. A FIR filter depends on a finite number of input values to compute an output value, whereas an IIR filter depends on a finite number of input and output values to compute an output value. In audio systems, both FIR and IIR are filters are commonly used, whereas in image processing, FIR filters are widely used and IIR filters are rarely used, in part because an image is finite in length in the horizontal and vertical directions.

2.0 Image Filtering (*15 points*)

An image is acquired by photon detectors arranged on a rectangular grid. Periodically, the photon counts in each photon detector is read and converted into an integer value representing intensity. Hence, images

are sampled in the horizontal and vertical dimensions. When acquiring color images, the photon detectors are placed in an alternating pattern of red, green, and blue sensors. [3] A color image can be converted into a grayscale image (a.k.a a black-white intensity image), or a three-channel representation of black-white intensity channel and two color channels (e.g. blue-yellow and red-green).

In this project, we will process grayscale intensity images represented as matrices. Each matrix element represents a pixel with an intensity value stored as a 64-bit IEEE floating-point number. The echart reference image [4] will be represented as a 512 x 512 matrix with each element value initially being an 8-bit unsigned integer but stored as a 64 bit IEEE floating-point number. The reference image is below.



We can apply a 1-D filter along each row of an image to produce a new row, or down each to produce a new column. For a 1-D FIR filter with filter coefficients $h[n]$ for $n = 0, 1, \ldots, N-1$, the relationship between output $y[n]$ and input $x[n]$ can be written as a convolution $y[n] = h[n] * x[n]$

$$y[n] = \sum_{k=0}^{N-1} h[k]\, x[n-k] = h[0]\, x[n] + h[1]\, x[n-1] + \cdots + h[N-1]\, x[n-(N-1)]$$

where the FIR filter is assumed to be causal [5]. The output of a causal system only depends on current and previous input values (i.e. a causal system). The filter order is $N-1$.

If we start observing signals $x[n]$ and $y[n]$ for $n \geq 0$, then the first output sample $y[0]$ is

$$y[0] = h[0]\,x[0] + h[1]\,x[-1] + \cdots + h[N-1]\,x[-(N-1)]$$

As a necessary condition for the filter to have the system properties of linearity and time invariance (LTI), the system must be at rest, which means that all the initial conditions must be zero: [5]

$$x[-1] = 0 \text{ and } x[-2] = 0 \text{ and } \cdots\ x[-(N-1)] = 0$$

Linearity means scaling the input by a constant always leads to a scaling of the output by the same amount (homogeneity) and the output of two signals added together the input is always the sum of their individual responses. Time invariance means if the input it shifted in time by a constant, then the output is always shifted in time by the same constant. An LTI system is uniquely represented by its impulse response. In these cases, the constant means constant with respect to time. [5]

For a 1-D IIR filter with feedforward coefficients b_k for $k = 0, 1, \ldots, N-1$, and feedback coefficients a_m for $m = 1, 2, \ldots, M-1$, the relationship between output $y[n]$ and input $x[n]$ can be written as

$$y[n] = \sum_{k=0}^{N-1} b_k\,x[n-k] + \sum_{m=1}^{M-1} a_m\,y[n-m]$$

assuming the IIR filter only depends on current and previous input values as well as previous output values (i.e. a causal system). Like an FIR filter, a necessary condition for the IIR filter to be LTI is that the system must be at rest, which means that all initial conditions must be zero. [5]

For an LTI filter, we can analyze how each discrete-time frequency $\hat{\omega}$ in the input signal is transferred to the output signal [5]

$$Y\left(e^{j\hat{\omega}}\right) = H\left(e^{j\hat{\omega}}\right) X\left(e^{j\hat{\omega}}\right)$$

where $Y\left(e^{j\hat{\omega}}\right), H\left(e^{j\hat{\omega}}\right),$ and $X\left(e^{j\hat{\omega}}\right)$ are the Discrete-Time Fourier Transforms of $y[n]$, $h[n]$ and $x[n]$, respectively. The definition of the Discrete-Time Fourier Transform of signal $h[n]$ is

$$H\left(e^{j\hat{\omega}}\right) = \sum_{n=-\infty}^{\infty} h[n]\,e^{-j\,\hat{\omega}\,n}$$

By taking the magnitude of both sides of the equation for $Y\left(e^{j\hat{\omega}}\right)$

$$\left|Y\left(e^{j\hat{\omega}}\right)\right| = \left|H\left(e^{j\hat{\omega}}\right) X\left(e^{j\hat{\omega}}\right)\right| = \left|H\left(e^{j\hat{\omega}}\right)\right| \left|X\left(e^{j\hat{\omega}}\right)\right|$$

we can determine whether a frequency component will be passed through, amplified, attenuated, or eliminated, which will correspond to $\left|H\left(e^{j\hat{\omega}}\right)\right|$ being equal to 1, greater than 1, between 0 and 1, and equal to 1. [5] We will see examples of this in the following sections.

Another signal analysis tool is the *z*-transform where *z* is a complex variable:

$$H(z) = \sum_{n=-\infty}^{\infty} h[n]\,z^{-n}$$

When the summation converges for values of *z* on the unit circle, we can convert the *z*-transform to the discrete-time Fourier transform by substituting $z = \exp(j\,\hat{\omega})$.

3.0 Filtering Images (*24 points*)

This section will implement 2-D filtering of an image by applying a 1-D filter along each row to produce an intermediate image and then down each column of the intermediate image.

In Matlab, applying a 1-D FIR filter with coefficients b along each row of image x to produce image y2:

```
y2 = filter(b, 1, x, [], 2);
```

Alternately, if b is a row vector, then we could use

```
y2 = conv2(x, b);
```

Applying a 1-D filter with coefficients b down each column of an image x to produce a new image y1:

```
y1 = filter(b, 1, x, [], 1);
```

Alternately, if b is a column vector, then we could use

```
y1 = conv2(x, b);
```

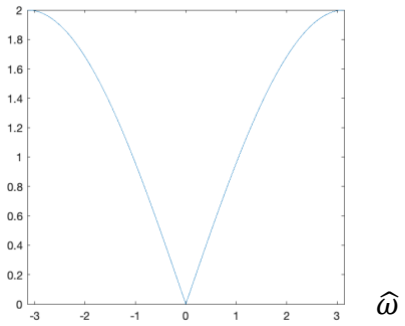
Convert a row vector into a column vector by using the transpose operator: bcolvec= browvec';

First-order LTI difference filter. We will apply this filter

$$y[n] = x[n] - x[n-1]$$

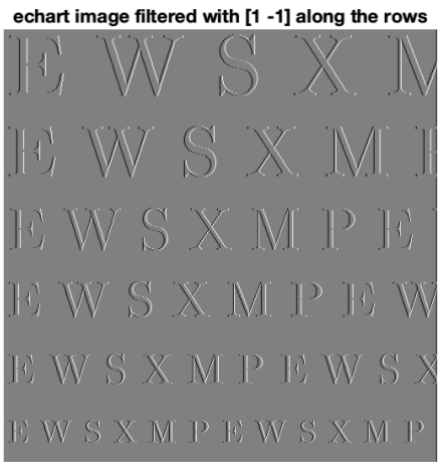
along the rows and columns of the reference image `echart`.

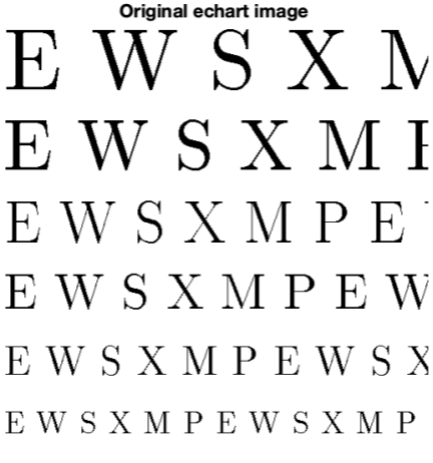
In 1-D, the first-order LTI difference filter is a highpass filter; that is, it passes high frequencies, attenuates low frequencies, and eliminates zero frequency, as shown by the magnitude of the frequency response of the filter is shown on the right.

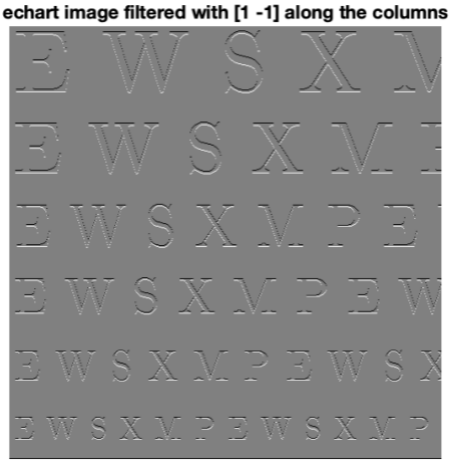


The `echart` image is a binary image of values 0 and 255. Since we are displaying the image on a monitor, 0 is background intensity (black) and 255 is the foreground intensity (white). The image has six rows of the same text "E W S X M P E W S X M P" with the font size (different resolutions) getting smaller from top to bottom. The `echart` image is stored as a double-precision floating-point matrix.

The first-order LTI difference filter applied along each row or column of the binary `echart` image extracts the lines and edges in the image as shown on the left and the right:







- In the spatial domain (image pixel domain), each output sample of the 1-D first-order difference filter will be the current input value minus the previous input value. Along one row of the echart image, because the echart image only has values of 0 and 255, will be one of these outcomes:

  -255 due to 0 - 255 = -255 when a white pixel (255) is followed by black pixel (0)

  0 due to either 0 - 0 or 255 - 255 when a black pixel (0) is followed by a black pixel (0) or a white pixel is followed by a white pixel

  255 due to 255 - 0 when a black pixel (0) is followed by a white pixel (255)

  Here, -255 would correspond to black and 255 to white in the grayscale image.

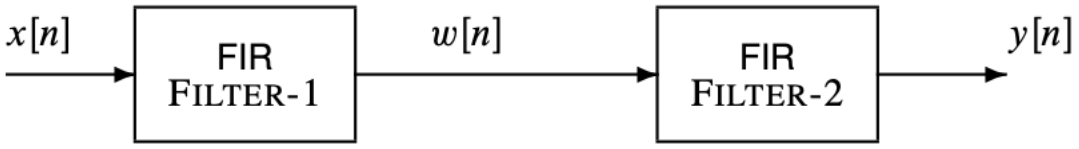
- In the frequency domain, the first-order difference filter is a highpass filter. When applied in either direction in a grayscale image, the filter will enhance the places where high frequencies occur, e.g. a sudden change from black to white or white to black. In the output image, the visual effect will be to extract lines, edges, texture, and other high frequency components.

Here's the Matlab code to apply the first-order LTI difference filter along the rows and the columns. We use `imshow(echart, [])` so the full range of pixel values are displayed and not clipped:

```
% The load command will define a Matlab matrix echart.
load echar512.mat

% Display the image.
hAxes = axes(figure);
hImage = imshow(echart, [], 'Parent', hAxes);
title(hAxes, 'Original echart image');

% Apply a first-order difference filter along the rows.
% Display the resulting image.
bdiffh = [1, -1];
yy1 = conv2(echart, bdiffh);
hAxes1 = axes(figure);
hImage1 = imshow(yy1, [], 'Parent', hAxes1);
title(hAxes1, 'echart image filtered with [1 -1] along the rows');

% Apply a first-order difference filter along the columns.
% Display the resulting image.
bdiffh = [1, -1];
yy2 = conv2(echart, bdiffh');
hAxes2 = axes(figure);
hImage2 = imshow(yy2, [], 'Parent', hAxes2);
title(hAxes2, 'echart image filtered with [1 -1] along the columns');
```

4.0 Image Restoration

We will valuate three different FIR filters to restore an image after the image has been distorted by an FIR filter. In our scenario, we will have a cascade of the distortion filter (FIR filter #1) followed by the restoration filter (FIR filter #2): [4]



Here are the difference equations for the filters [4]:

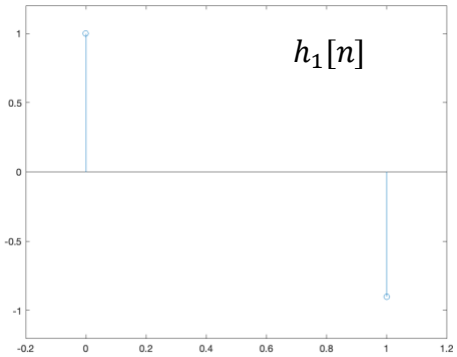
$$w[n] = x[n] - q\,x[n-1] \qquad \text{(FIR FILTER-1)}$$

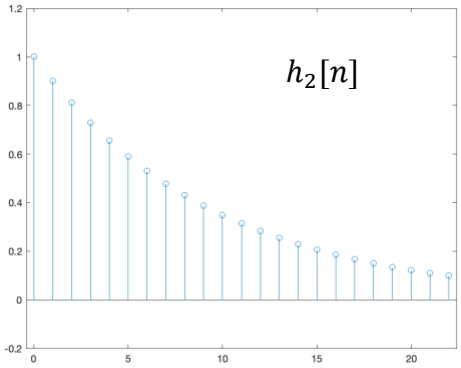
$$y[n] = \sum_{\ell=0}^{M} r^{\ell} w[n-\ell] \qquad \text{(FIR FILTER-2)}$$

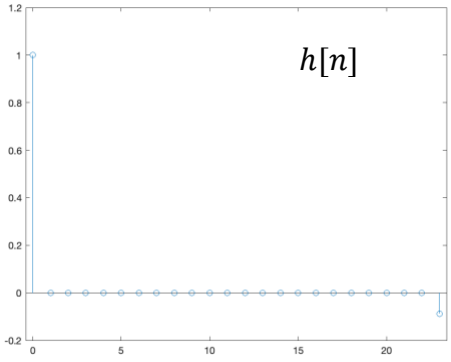
4.1 Overall Impulse Response (*15 points*)

(a) Here is the Matlab code to define the two impulse responses $h_1[n]$ and $h_2[n]$ and the overall impulse response $h[n]$ given $q = 0.9, r = 0.9$ and $M = 22$ and their plots are shown on the right:

```
% h1[n] has non-zero coefficients of 1 and -q:
q = 0.9;
h1 = [ 1 -q ];

% h2[n] = r^n for 0 <= n <= M and 0 elsewhere:
r = 0.9;
M = 22;
n = 0 : M;
h2 = r .^ n;

% Overall impulse response
h = conv(h1, h2);

% Plots
n = 0 : 1;
figure; stem(n, h1);
xlim( [-0.2 1.2] ); ylim( [-1.2 1.2] );
n = 0 : M;
figure; stem(n, h2);
xlim( [-0.4 (M+0.4)] ); ylim( [-0.2 1.2] );
n = 0 : M+1;
figure; stem(n, h);
xlim( [-0.4 (M+1.4)] ); ylim( [-0.2 1.2] );
```







(b) Here is the derivation of the overall response for general values of $q, r,$ and M:

$$h[n] = h_1[n] * h_2[n]$$

$$h[n] = \sum_{k=-\infty}^{\infty} h_1[k]\, h_2[n-k] = \sum_{k=0}^{1} h_1[k]\, h_2[n-k]$$

$$h[n] = h_2[n] - q\, h_2[n-1]$$

$h[0] = h_2[0] - q\, h_2[-1] = 1$

$h[1] = h_2[1] - q\, h_2[0] = r - q$

For $2 \le n \le M$,

$$h[n] = r^n - q\, r^{n-1} = r^n - q\, r^n\, r^{-1} = r^n - \frac{q}{r}\, r^n = \left(1 - \frac{q}{r}\right) r^n$$

For $n = M + 1$,

$$h[M+1] = h_2[M+1] - q\, h_2[M] = 0 - q\, r^M = -q\, r^M$$

For $n > M + 1$,

$$h[n] = 0$$

When $q = r$, the derivation matches the numerical calculations in part (a).

(c) If we design FIR FILTER-2 to undo all distortion caused by FIR FILTER-1, then FIR FILTER-2 is called a deconvolution filter or an equalizer. In this case, we would like to have an identity system $y[n] = x[n]$ which has an impulse response of $h[n] = \delta[n]$. That is,

$$h[n] = h_1[n] * h_2[n] = \delta[n]$$

In the *z*-domain,

$$H_1(z)\, H_2(z) = 1$$

$$H_2(z) = \frac{1}{H_1(z)} = \frac{1}{1 - q\, z^{-1}} \text{ for } |z| > |q|$$

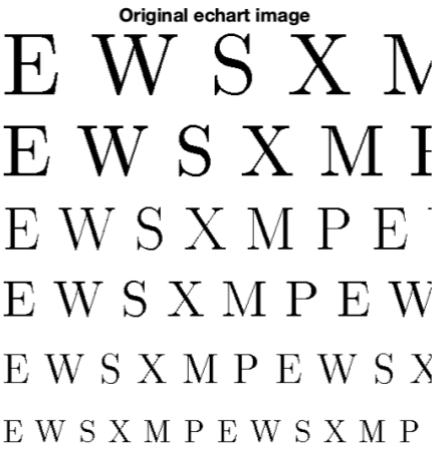
$$h_2[n] = q^n\, u[n]$$

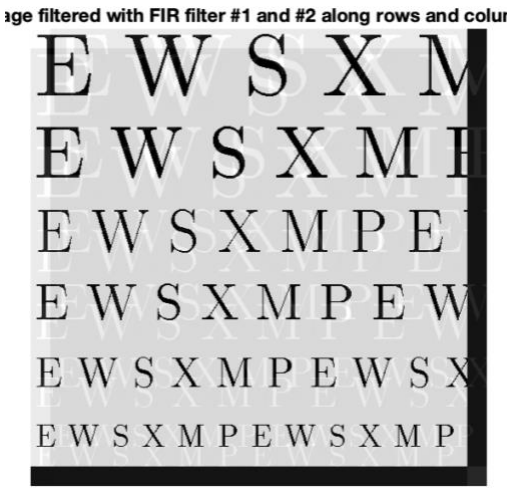
This is an IIR filter. Since FILTER-2 is FIR, FIR FILTER-2 cannot perform perfect deconvolution.

4.2 Distorting and Restoring Images (*15 points*)

We will design FIR FILTER-2 to undo a significant amount of distortion introduced by FIR FILTER-1.

We apply FIR FILTER-1 with $q = 0.9$ along the rows of `echart` to create an intermediate image and then apply FIR FILTER-1 along the columns of the intermediate image. For the restoration filter, we





use $r = q = 0.9$ and $M = 22$. The distorted, original, and restored images are shown below:

For the impulse response of the restoration filter, we are using a truncated version of the ideal IIR impulse response derived in the previous section:

$$h_2^{IIR}[n] = q^n\, u[n]$$

For FIR FILTER-2, we are keeping the first $M + 1$ values of $h_2^{IIR}[n]$ from $n = 0, 1, \ldots, M$.

The restored image recovers the text quite well, but we see two artifacts:

- ***Ghosting***. There is a "ghost" copy of each letter $M + 1$ pixels directly to the right of the letter and $M + 1$ pixels directly below the letter. Each ghost copy appears as a white letter. The ghosting is due to the overall impulse response $h[n]$ shown in Section 4.1(b) on page 5 which has value of 1 at $n = 0$, r^{M+1} when $n = M + 1$, and 0 otherwise. The value of the impulse response at $n = M + 1$ generates the ghost letter in each dimension due to convolution.
- ***Border***. Due to convolution, the distorted image has one more row and one more column than the original `echart` image, and the restored image has M additional rows and M additional columns than the distorted image. Visually, we see these extra pixels in the restored image as the black border on the bottom and the right.

Worst case error between the restored image and the original `echart` image is 45.2 gray levels out of 303.2 gray levels in the restored image. After normalizing the restored image so that the range of gray levels is the same as that for `echart`, the worst case error was 39.8 gray levels out of 256 grade levels. This worst-case error of roughly 15% should be visible under our viewing conditions (lighting, illumination, pixel size, viewing distance, etc.). [6][7]

Matlab code to create the distorted and restored images on the previous page appears below. The black border in the restored image is a result of convolution, which we implement using the `conv2` Matlab command. If we had used the `filter` command instead, then the distorted and restored images would have been the same size as the original image, and we would not have seen the black border in the distorted image. The `filter` command produces as many output samples as there are input samples.

```
% Section 4.2 Deconvolution
load echar512.mat
hAxes = axes(figure);
hImage = imshow(echart, [], 'Parent', hAxes);
title(hAxes, 'Original echart image');

% Apply two 2-D FIR filters
% h1[n] has non-zero coefficients of 1 and -q:
q = 0.9;
h1 = [ 1 -q ];
yy5 = conv2(echart, h1);
ech90 = conv2(yy5, h1');
hAxes6 = axes(figure);
hImage6 = imshow(ech90, [], 'Parent', hAxes6);
title(hAxes6, 'Image filtered with FIR filter #1 along rows and columns');

% h2[n] = r^n for 0 <= n <= M and zero elsewhere:
r = 0.9;
M = 22;
l = 0 : M;
h2 = r .^ l;
```

```
yy7 = conv2(ech90, h2);
yy8 = conv2(yy7, h2');
hAxes8 = axes(figure);
hImage8 = imshow(yy8, [], 'Parent', hAxes8);
title(hAxes8, 'Image filtered with FIR filter #1 and #2 along rows and columns');

% Worst case error in the restored image vs. the echart image
% on a pixel-by-pixel basis.  We have to remove the black border
% in the restored image so the restored images are same size
% and then normalize the number gray levels.
[numrows numcols] = size(echart);
yy8cropped = yy8(1:numrows, 1:numcols);
yy8min = min(min(yy8cropped));
yy8max = max(max(yy8cropped));
echartmin = min(min(echart));
echartmax = max(max(echart));
yy8normalized = (yy8cropped - yy8min) * (echartmax / (yy8max - yy8min));
worstCaseError = max(max(abs(yy8normalized - echart)));
```

4.3 Evaluating Three Candidate Restoration Filters (*15 points*)

Next, we will evaluate three different FIR image restoration filters using the image restoration approach in Section 4.2. For the image restoration FIR FILTER-2, we will consider

a) $r = q = 0.9$ and $M = 11$
b) $r = q = 0.9$ and $M = 22$
c) $r = q = 0.9$ and $M = 33$

For FIR FILTER-2, we are keeping the first $M + 1$ values of $h_2^{IIR}[n]$ from $n = 0, 1, \ldots, M$ where

$$h_2^{IIR}[n] = q^n \, u[n]$$

Increasing the number of coefficients in FIR FILTER-2 will better approximate the ideal IIR filter.

Each restored image recovers the text, but we see ghosting and border artifacts mentioned in Section 4.2. There are two ghost letters for every letter in the original image. As M increases,

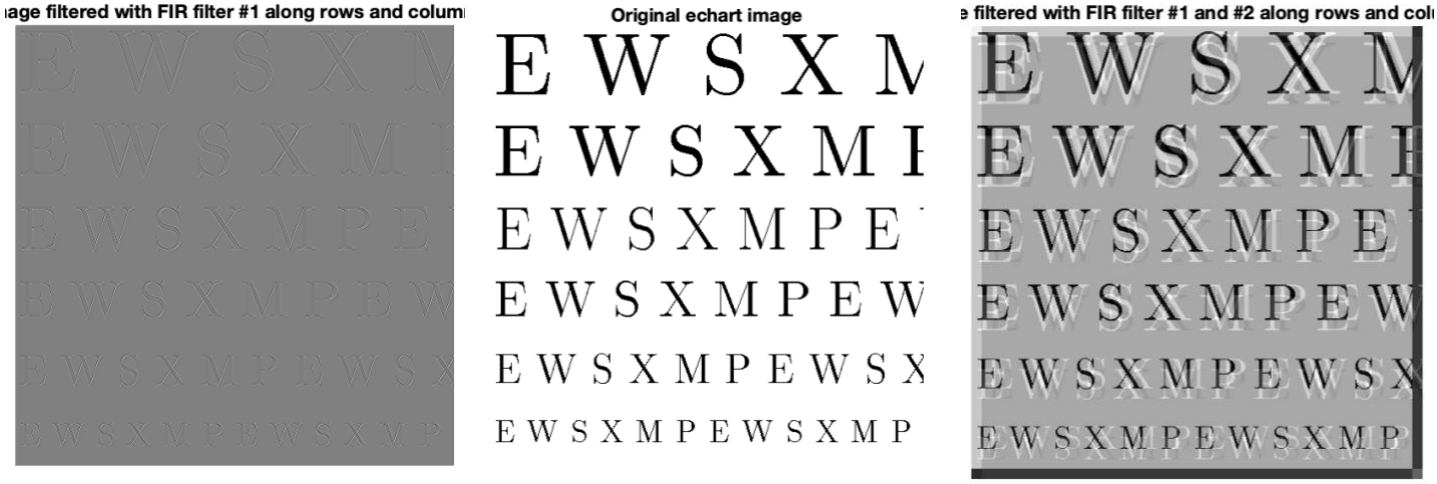
- Each ghost letter will appear farther away from the letter in the original image because the distance is $M + 1$ pixels to the right for the horizontal ghost letter and $M + 1$ pixels below for the vertical ghost letter.
- Each ghost character will have reduced intensity due to the filter coefficient in overall impulse response $h[n]$ at index $n = M + 1$ of value $-r^{M+1}$ and $r = 0.9$.
- The border on the right and bottom of the restored image will grow larger, but we will later crop out the border region to make the restored image the same size as the original.

We will use image quality measures to compare the original and they restored image for each of the three restoration filters. Matlab has several standard Image Quality Metrics [8]:

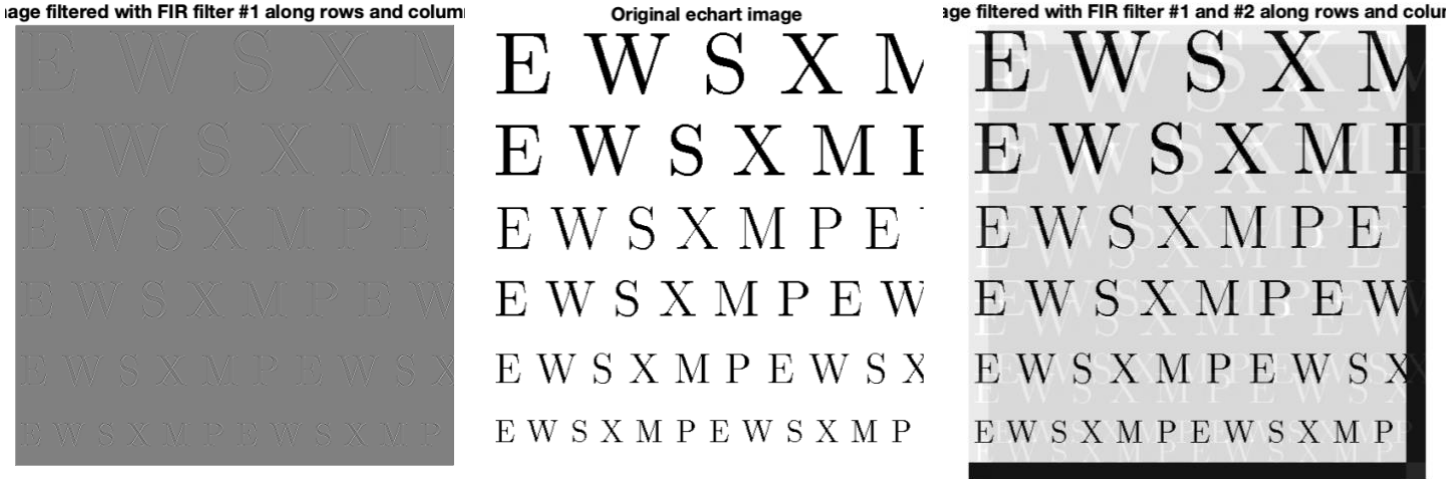
- `immse` to compute the mean squared error (MSE) where a lower score is better, but the scores might not align well with human perception of quality.
- `psnr` to compute the peak signal-to-noise ratio (PSNR) in dB where a higher score is better, but the scores might not align well with human perception of quality.

- `ssim` to compute the structural similarity (SSIM) index on a scale of 0 to 1 where a higher score is better, and the scores align well with human perception of quality.

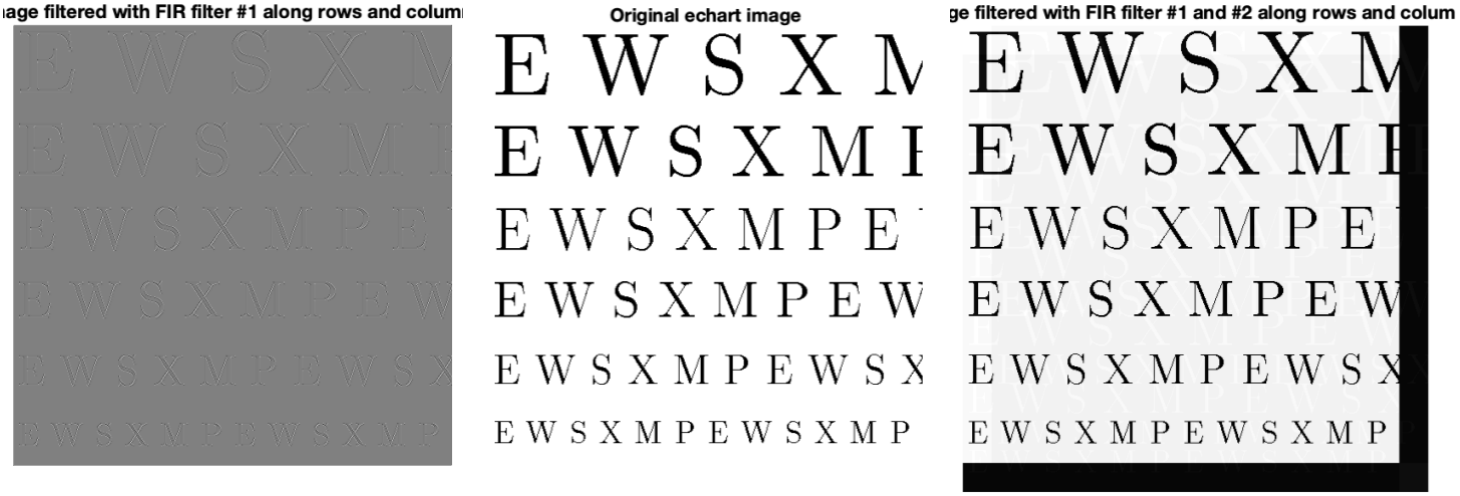
The restored images are shown on the next page for restoration filter orders of $M = 11, 22, 33$.



Restoration filter with $M = 11$



Restoration filter with $M = 22$



Restoration filter with $M = 33$

Image quality measures. Here are the numerical measures of the difference between the distorted and original image as well as between the restored and original image for three different restoration filter orders $M = 11, 22, 33$. When computing these measures, we crop the distorted and restored images to be the same size as the original image. The restoration filter only affects the restored image, and that is why the same numerical measure comparing the distorted and original image do not change across the three tables. As the restoration filter order M increases, all image quality measures improve for the restored image. The Matlab code for these calculations appears after the tables.

Restoration filter $M = 11$	**Distorted vs. original image**	**Restored vs. original image**
IMMSE	58180	13032
PSNR	-47.6 dB	-41.2 dB
SSIM	0.0077	0.4234

Restoration filter $M = 22$	**Distorted vs. original image**	**Restored vs. original image**
IMMSE	58180	1524
PSNR	-47.6 dB	-31.8 dB
SSIM	0.0077	0.4779

Restoration filter $M = 33$	**Distorted vs. original image**	**Restored vs. original image**
IMMSE	58180	153
PSNR	-47.6 dB	-21.9 dB
SSIM	0.0077	0.5173

Matlab code appears below to measure the difference between the distorted and original images as well as the difference between the restored and original images.

```
% Comparing two images
% Mean squared error
% We have to make sure the two images have the same dimensions
% and have the same
% The order of the arguments do not matter
ech90cropped = ech90(1:512, 1:512);
yy8cropped = yy8(1:512, 1:512);

ech90immse = immse(echart, ech90cropped)
yy8immse = immse(echart, yy8cropped)

ech90psnr = psnr(echart, ech90cropped)
yy8psnr = psnr(echart, yy8cropped)

ech90ssim = ssim(echart, ech90cropped)
yy8pssim = ssim(echart, yy8cropped)
```

5.0 Image Restoration (*7 points*)

In this mini project, we investigated image distortion and restoration. Distortion was modeled as finite impulse response (FIR) filter that extracted high-frequency features such as lines, edges, and corners. The ideal image restoration filter to compensate fully for the distortion is an infinite impulse response (IIR) filter. To find FIR filter approximations to the ideal IIR filter, we truncated the impulse response of the IIR filter. We used FIR filter orders of 11, 22, and 33, and evaluated the resulting quality of the restored image vs. the original image using mean squared error, peak signal-to-noise ratio, and structural similarity index measures. For all three measures, the restored image had improved scores as the FIR filter order increased from 11 to 22 to 33.

References

[1] https://en.wikipedia.org/wiki/Piano_key_frequencies, accessed on Nov. 4, 2023.

[2] https://en.wikipedia.org/wiki/Lateral_geniculate_nucleus, accessed on Nov. 4, 2023.

[3] https://en.wikipedia.org/wiki/Bayer_filter, accessed on Nov. 4, 2023.

[4] B. L. Evans, "Mini-Project #2: Image Filtering", ECE 313 Linear Systems & Signals, UT Austin, Fall 2023.

[5] J. H. McClellan, R. W. Schafer & M. A. Yoder, *Signal Processing First*, 2003. Errata. On-line Companion.

[6] https://pixelcraft.photo.blog/tag/human-eye/

[7] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3043920/

[8] https://www.mathworks.com/help/images/image-quality-metrics.html