# Lab 2: Echoes and Images

# Introduction

The goal of this lab is to learn how to implement downsampling, convolution, and FIR filters, and then study the response of FIR filters to various signals, including images, speech, complex exponentials, and sinusoids. You will learn about 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.

# 1. Lab Part 1

## 1.1 Implement the 3-point Averager

Before you begin this section, make sure that you have watched and understand the lecture videos on FIR filters.

1. There are 2 different functions to implement a FIR filter:

   firfilt() (see help firfilt )
   conv() (see help conv )

   **Example Code:**

   ```
   xx =[ones(1,10),zeros(1,5)]; %<--Input signal
   nn = 1:length(xx); %<--Time indices
   bk = [1/3 1/3 1/3]; %<--Filter coefficients
   yy = firfilt(bk, xx); %<--Compute the output
   yy = conv(bk, xx); %<--Equivalent method to compute output
   ```

   To illustrate the filtering action of the 3-point averager, *run the above code* and then *make a stem plot of the input signal and output signals in the same figure* using the following code:

   ```
   figure;
   clf
   subplot(2,1,1);
   stem(nn, xx(nn))
   subplot(2,1,2);
   stem(nn, yy(nn), 'filled')
   xlabel('Time Index (n)')
   ```

*Question 1: What characteristics of the input signal are most affected by the averager? Least affected?*

*Question 2: What is the relationship between the lengths of input vector, output vector, and coefficient vector?*

## 1.2 Echo Filter

The data file labdat.mat contains two filters and three signals, stored as separate 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 fs = 8000 samples per second
- h1: the coefficients for a FIR discrete-time filter
- h2: coefficients for a second FIR filter.

Load the data by using the following commands:

```
load labdat.mat
```

**Echo:** a phenomenon where the output is the sum of the input and a delayed version of the input. Echo effect can be implemented as an FIR filter called an echo filter:

$$y[n] = x_1[n] + rx_1[n - P]$$

1. Suppose you have an audio signal sampled at fs = 8000 and you want to simulate an echo.

   *Question 3: What values of r and P will give an echo with strength 85% of the original, with time delay 0.22s?*

   Using your answer to the question above, *implement the echo filter* and use it on the signal in vector x2 obtained from labdat.mat.

2. Multiple echos can be accomplished by cascading several echo filters of the above form. Using the parameters determined in the previous section, *derive **(by hand)** the impulse response of a reverb system produced by cascading three "single echo" systems.* Recall that two filters are said to be "in cascade" if the output of the first filter is used as the input to the second filter, and the output of the second filter is defined to be the output of the overall cascade system. This can be repeated for as many filters as are needed in the cascade system.

   *Filter the x2 signal with this multi-echo filter. Play it for a TA for checkoff.* If you can't finish in time, submit a .wav file on Canvas.

# 1.3 Image Processing

## 1. Show a Test Image

You can load the images needed for this lab from the included the echart.mat file.

You'll find two variables containing image data. Although MATLAB has several functions for displaying images on the monitor of the computer, we will use a function from SPFirst toolbox called show_img() (see help show_img ).

**Note**: if you want the image to be shown in grayscale, the keyword argument cmap needs to be set to 'gray' .

*Use show_img (don't forget to use figure when necessary) to display the image in echart.mat .* (Hint: It should look like an eye chart seen in an eye doctor's office).

## 2. The Lighthouse Image

*Load and display* the 326 x 426 lighthouse image from the lighthouse.mat.

Use the colon operator to *extract the 225th row* of the "lighthouse" image, and *plot it* as a discrete-time one-dimensional signal. Do **NOT** use stem.

Observe that the range of signal values is between 0 and 1.

*Question 4: Which values represent white? Black?*

*Question 5: Where does the 225th row cross the fence?*

*Question 6: What features of the image correlate with the periodic-like portion of the plot?*

*Submit an annotated version of your plot to support your answers.*

## 3. Synthesize a Test Image

To probe your understanding of the relationship between matrices and image display, generate a synthetic image from a mathematical formula.

*Display this synthetic image* in which all of the columns are identical by using the following outer product:

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

*Question 7: How wide are the bands in number of pixels? How is this width related to the formula for xpix?*

*Question 8: How would you produce an image with horizontal bands?*

*Create (and display/submit) a 450x450 image with 4 horizontal black bands separated by white bands.*

### 4. Sampling of Images

Images on a computer (digital storage) are sampled images stored in an MxN matrix. The sampling rate in the two spatial dimensions was chosen at the time the image was digitized. Just as a sinusoid is measured in dots per unit of time, a photograph is measured in dots per inch in each direction.

For example, the image might have been "sampled" by a scanner where the resolution was chosen to be 300 (dots per inch). Lowering the sampling rate (down-sampling), which compresses the image for the sake of transmission or storage, throws away samples in a periodic way. If every other sample is removed, the sampling rate will be halved. A 300 dpi image would become a 150 dpi image.

wp = ww(1:p:end) down-samples a 1D vector x by a factor of p . wp = ww(1:p:end,1:p:end) down-samples a 2D matrix x horizontally by a factor of p and vertically by a factor of q (see https://en.wikipedia.org/wiki/Downsampling_(signal_processing) or https://www.youtube.com/watch?v=TFEf8yOlXyY if necessary to understand downsampling).

However, just as downsampling a sinusoid can result in aliasing, downsampling an image can also result in aliasing.

*Load the lighthouse image from the lighthouse.mat* (if it isn't loaded already).

*Down-sample the lighthouse image in both dimensions by a factor of 2*, and *show the image*. Note that the image is not square.

*Question 9: What is the size of the down-sampled image?*

Notice the difference in appearance of the image, despite there not being any added points.

When estimating spatial frequency, consider recurring features of the images as "peaks". From this you can obtain a period (how many pixels until the image "peaks" again), and from there a frequency.

*Question 10: Describe how the aliasing appears visually.*

*Question 11: Which parts of the image most dramatically show the effects of aliasing? Why does the aliasing manifest itself in these places?*

*Question 12:* From your row plot and from zooming in on the image*, estimate the frequency of the aliased features in cycles per pixel.* When estimating spatial frequency, consider recurring features of the images as "peaks". From this you can obtain a period (how many pixels until the image "peaks" again), and from there a frequency.

*Question 13: How does your estimation of aliased features fit into the Sampling Theorem?*

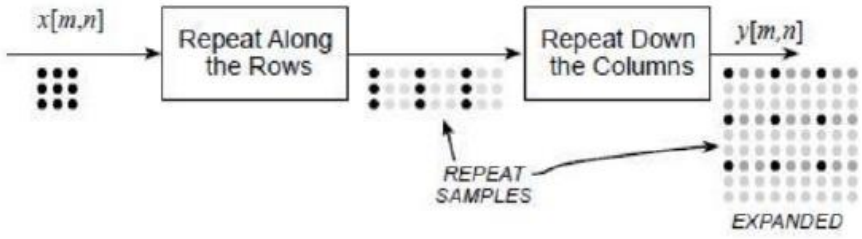
# 2. Lab Part 2

## 2.1 Image Reconstruction from Downsampling

## Note: Do NOT hard-code values!!!

### 1. Introduction to Interpolation

When an image has been sampled, we can fill in the missing samples through interpolation. For images, this would be analogous to the examples shown in the textbook for sine-wave interpolation, which is part of the reconstruction process in a digital-to analog converter. We could use a "square pulse", "triangular pulse", or other pulse shape for the reconstruction.




A "square pulse" reconstruction is also known as a zero-order hold, and is the simplest form of interpolation for this case. Every point in the decimated array is duplicated in the row and column directions without change. Moving through the reconstituted array shows copies of each value before reaching a new value; hence this reconstruction is with a "square pulse".

The following code shows two ways to perform a zero order hold for a one-dimensional signal (one row or one column of the image), assuming that we start with row vector row and the result is a row vectors hold1 :

```
row = (-2).^(0:6);
L = length(row);
nn= ceil((0.999:1:4*L)/4); %<---Round up to the integer
hold1 = row(nn);
```

(See https://en.wikipedia.org/wiki/Zero-order_hold if necessary)

*Plot the vector hold1* to verify that it is indeed a zero-order hold derived from row .

*Question 1: What values are in the indexing vector nn , and why are they what they are?*

## 2. Interpolate a Lighthouse

**a.** Load the lighthouse image data from lighthouse.mat (if you haven't already).

**b.** *Down-sample* the lighthouse image by a factor of 3 (similar to what you did in section 1.3). Let's call the new array x3 .

**c.** *Perform a zero-order hold on x3* to fill in the missing points:
For an interpolation factor of 3, process all rows of x3 to fill in missing points in that direction. Call the result xrows.

*Question 2: What are the dimensions of the rows and columns of xrows?*

**d.** Now process all the columns of xrows likewise (interpolation factor 3, now processing in the other direction) to fill in the missing points in each column. Call this result xhold , and *show* the xhold and original lighthouse images on the same plot.

*Question 3: Compare them and explain* any differences that you can see. (Note that a zero-order hold will not produce a high quality reconstruction)

**e.** Linear interpolation is sometimes more accurate. It assumes that if you have two values at points next to each other, you can take a weighted average of the two points to come up with a value

for a point in between those two points. It is thus a "triangular pulse", interpolating on a sloped line. More full explanations are found at http://www.eng.fsu.edu/~dommelen/courses/eml3100/aids/intpol/ and https://en.wikipedia.org/wiki/Linear_interpolation.

Linear interpolation can be done using interp1() (see help interp1 ).

Here is an example on the same row signal that we used earlier in this section:

```
n = 0:6;
row = (-2).^(0:6);
tt = 0:0.1:6; % <---locations between the n indices
hold_linear = interp1(n, row, tt);
stem(tt, hold_linear);
```

*Carry out linear interpolation operations* on both the rows and the columns of the down-sampled lighthouse image x3 . *Show the output*.

**f.** Compare the output images:

*Show* the original image, the down-sampled image, the zero-order-hold reconstructed image, and the linearly-interpolated reconstructed image. *Point out their differences and similarities.*

*Question 4: Can the linear interpolation reconstruction process remove the aliasing effects from the down-sampled lighthouse image?*

*Question 5: Can the zero-order hold reconstruction process remove the aliasing effects from the down-sampled lighthouse image?*

*Question 6: Point out regions* where the linear and zero-order reconstruction result images differ and try to *justify* this difference in terms of the frequency content in that area of the image. In other words, look for regions of 'low-frequency" and "high-frequency" content in the image and explain how the interpolation quality is dependent on this factor.

*Question 7: Are edges low-frequency or high-frequency features? Is the series of fence posts a low-frequency or high-frequency feature? Is the background a low-frequency or high-frequency feature?*

# 2.2 Filtering Images

## 1. 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. We can filter this signal with a 1-D filter using conv() function.

**a.** *Load in the echart image from the echart.m file.*

*Filter all the rows of the image with the conv2() function (see help conv2 )*. To filter the image in the horizontal direction using a first-difference filter, we form a row vector of filter coefficients using the following code:

```
b = [1, -1];
```

**b.** *Filter the image in the vertical direction* with this first-difference filter to *produce the image y* . This can be done by transposing the image then filtering the rows.

**c.** Using the show_img() function, *display the input image echart, the intermediate image row-filtered, and the output image y .*

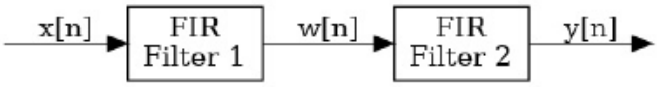
*Question 8: Compare the three images and give a qualitative description of what you see*.

## 2. Distorting and Restoring Images

More complicated systems are often made up from simple building blocks. In the system of Figure below two FIR filters are connected "in cascade." For this section, assume that the filters in Figure below are described by the two equations:

$$\omega[n] = x[n] - qx[n-1] \qquad (FIR\ 1)$$

$$y[n] = \sum_{l=0}^{M} r^l \omega[n-L] \qquad (FIR\ 2)$$




If we pick to be a little less than 1.0, then the first system (FIR 1) will cause distortion when applied to the rows and columns of an image. The objective in this section is to show that we can use the second system (FIR 2) to undo this distortion (more or less). Since FIR FILTER 2 will try to undo the convolutional effect of the first, it acts as a deconvolution operator.

**a.** Load the echart image from the echart.m module. Pick q=0.9 for the first filter (FIR 1).

Using this filter, *filter the echart image along the horizontal direction, and then filter the resulting image vertically*. Call the result echo90.

**b.** Convolve echo90 with FIR 2, choosing M=22 and r=0.9.

*Question 9: Describe the visual appearance of the output qualitatively, showing the image, and explain* its features by invoking your mathematical understanding of the cascade filtering process and why you see "ghosts" in the output image.

*Question 10:* Use some previous calculations to *determine the size and location of the "ghosts" relative to the original image.*

**c.** *Question 11: Evaluate the worst-case error* in order to say how big the ghosts are relative to "black-white" transitions which are 0 to 255. Make sure to show any code you used or plots to further your evaluation.

# 2.3 Edge Detection and Reconstruction

## 1. Image Blurring

(See https://en.wikipedia.org/wiki/Gaussian_blur)

At times, it may be useful to apply a blur to an image in order to perform transformations. One such method is applying a Gaussian blur to the image. The Gaussian blur is a type of image-blurring that creates a kernel (see https://en.wikipedia.org/wiki/Kernel_(image_processing)) with a Gaussian distribution (i.e. the center values are the highest, and the values decrease rapidly away from the center). In MATLAB, a function can be created (see blur_kernel.m) to generate a blur kernel based upon a provided standard deviation, which can then be convolved with an image to produce a blurred image. This can also be done using the imgaussfilt function (help imgaussfilt), which will filter the image with a kernel based upon the standard deviation, by default 0.5.

a. **Note: The blur_kernel code is provided for understanding of how the kernel is generated. The kernel to use is provided in tower.mat**

   Load in the Eiffel Tower image from tower.mat. Show the result.

   Convolve the original tower with the provided blur kernel. This is essentially applying a gaussian blur on the image. Show and describe the result.

   Modify the blurred image so that it has the same size as the original image, but without the black border around the edges.

## 2. Edge Detection

(See Common Uses section of Gaussian blur page or https://en.wikipedia.org/wiki/Edge_detection#A_simple_edge_model)

Blurring an image removes the high-frequency components (for example, the fence posts in the lighthouse image would be considered high-frequency, whereas the solid wall of the building would be considered low frequency). Based upon a simple manipulation of the original image and the blurred image, one can generate a new image that only contains high-frequency components. By thresholding a grayscale image (comparing all of the pixel values to a certain number, and assigning each pixel to the maximum or minimum value as necessary, https://en.wikipedia.org/wiki/Thresholding_(image_processing)) whose values are between 0 and 1, the high frequency components can be emphasized.

a. Create and show an image that only contains the high-frequency components of the image (think: the original image has both high and low-frequency components, and the blurred image only has low-frequency components).

b. Threshold the image to make high-frequency components the max value (1), and all other components 0, and show the result (hint: use a very small comparison value to get the edges to appear clearly).

## 3. Image Restoration

As you may have noticed in an earlier part of the lab, performing a zero order hold on an image, followed by interpolating the image, causes repeated high-frequency elements to become aliased. To prevent this problem (see the last paragraph of Mathematics in https://en.wikipedia.org/wiki/Gaussian_blur), we can first apply a blur to the image, then perform the aforementioned operation.

a. Downsample the tower by a factor of 3 and perform an interpolation on the tower by a factor of 3. Show both.

b. Downsample the blurred tower by a factor of 3 and perform an interpolation on the blurred tower by a factor of 3. Show both.

*Question 12:* Describe the visual differences between the interpolated tower and the interpolated blurry tower.