# Lab 11: Sampling and Zooming of Images

In this lab we study the application of FIR filtering to the image zooming problem, where lowpass filters are used to do the interpolation needed for high quality zooming. The zooming problem is basically the same as the D-to-A reconstruction problem for analog signals.

## 1 Overview

Review the image display techniques from Lab 10 (**`show_img`**, etc.)

## 2 Warm-up: Linear Interpolation

In Section 3.2, we will be interest in image zooming which is an interpolation problem. Before processing images, however, we consider the following one-dimensional interpolation problem.

(a) Generate a sequence of data samples with two zeros between each non-zero sample:

```
xss = zeros(1,19);
samp = [1 3 -2 4 2 -1 -3];
xss(1:3:19) = samp;
```

(b) Now process this sequence through an FIR filter with “triangular” coefficients.

```
coeffs = [1/3, 2/3, 1, 2/3, 1/3];
output = firfilt(xss,coeffs);
```

What observations can you make regarding the similarities between the sequence `output` and the nonzero samples of the original sequence (`xss`)? Do you notice a relative shift between these two sequences? Measure the length of this time shift in samples. Explain why the output of the triangle FIR filter is a *linearly interpolated* version of the input sequence.

(c) Write the difference equation for the “triangular” filter defined by `coeffs` in part (b). Now calculate, *by hand*, the output of the filter to the input sequence

$$x[n] = \delta[n] + 0.5\delta[n-3]. \tag{1}$$

$$= \begin{cases} 1, & \text{when } n = 0 \\ 0.5, & \text{when } n = 3 \\ 0, & \text{else} \end{cases} \tag{2}$$

Do not use MATLAB to complete this exercise. The purpose is to understand exactly how the linearly interpolated output is generated when you implement the difference equation.

**Instructor Verification** (separate page)

In summary, we notice that linear interpolation involves two steps. The first step is zero filling, where the number of zeros inserted between each sample determines the degree of interpolation. The second step is FIR filtering with the appropriate triangle coefficients.

## 3 Lab: Sampling of Images

The images that are stored on the computer have to be sampled images because they are stored in an $M \times N$ array. The sampling rate in the two spatial dimensions was chosen at the time the image was digitized (in units of samples per inch if the original was a photograph). However, we can simulate a *lower* sampling rate by simply throwing away samples in a periodic way. If every other sample is removed, the sampling rate will be halved. Sometimes this is called *sub-sampling* or *down-sampling*. So, for example, if you have a vector `x1` representing a signal such as a row of an image, we can reduce the sampling rate by a factor of 4 by simply taking every 4th sample. In MATLAB this is easy with the colon operator, i.e., `xs = x1(1:4:length(x1))`. The vector `xs` is one fourth the length of `x1`.

An alternative sampling strategy is to take every 4th sample, but place zeros in between. The M-file below called `imsample( )` will perform this type of sampling on an image, e.g., `xs = imsample(xx,4):`.

(a) Explain how the function `imsample.m` works:

```
function yy = imsample(xx, P)
%IMSAMPLE    Function for sub-sampling an image
% usage:   yy = imsample(xx, P)
% xx = input image to be sampled
%   P = sub-sampling period (an integer like 2,3,etc)
%  yy = output image
%
[M,N] = size(xx);
S = zeros(M,N);
S(1:P:M,1:P:N) = ones(length(1:P:M), length(1:P:N));
yy = xx .* S;
```

(b) Execute the statement `xs = imsample(xx,4);` and use `show_img( )` to plot the images `xs` and `xx`. From the plot you should see that `imsample( )` throws away samples by setting them to zero. The samples that it keeps remain in their original spatial locations. With the zero samples included, the "sampled image" has the same spatial dimensions as the original when displayed on the screen.

(c) *Down-sampling* throws away samples, so it will shrink the size of the image. This is what is done by the following scheme:

```
xp = xx(1:p:M,1:p:N);
```

One potential problem with down-sampling is that aliasing might occur. This can be illustrated in a dramatic fashion with the **`zone`** image, or the **`lenna`** image.

Now down-sample the **`zone`** image by a factor of 2. Notice the aliasing in the down-sampled image, which is surprising since no new values are being created by the down-sampling process. Describe how the aliasing appears visually.[1]

[1]One difficulty with showing aliasing is that we must display the pixels of the image exactly. This almost never happens because most monitors and printers will perform some sort of interpolation to adjust the size of the image to match the resolution of the device. In MATLAB we can override these size changes by using the function `truesize` which is part of the Image Processing Toolbox.

Down-sample the **lenna** image by a factor of 2. Notice that this image seems to be relatively unaffected by the down-sampling by 2 process, what can you say about the frequency content of the `lenna` image as opposed to the `zone_plate` image?

## 3.1 Reconstruction of Images

When an image has been sampled, we can fill in the missing samples by doing interpolation. For images, this would be analogous to the examples shown in Chapter 4 for sine-wave interpolation which is part of the reconstruction process in a D-to-A converter. We could use a "square pulse" or a "triangular pulse" or other pulse shapes.

For these reconstruction experiments, use either the **tools** image or the **lenna** image. It would be wise to put the original image and all the filtered reconstructions on the screen in different figure windows for easy comparison.

(a) The simplest interpolation would be the square pulse which produces a "zero-order hold." We could fill in the gaps between samples in each row with the following statement:

```
xs = imsample(xx,4);
bs = ones(1,4);        %--- Length-4 square pulse
yhold = conv2(xs,bs);  %--- 2-D FIR filtering (horizontally)
```

Try this and plot the result `yhold`.

(b) Now filter the columns of `yhold` to fill in the missing points in each column and compare the result to the original image `xx`.

(c) *Linear interpolation:* Now use what you learned about linear interpolation in the warm-up section to determine an FIR filter that will perform linear interpolation on the rows and columns of the sub-sampled (by 4) signal. This filter must have coefficients $\{b_k\}$ that follow a triangle shape, and the order must be $M = 6$.

(d) Carry out the linear interpolation operations using MATLAB's `conv2` function. Call the interpolated output `ylin`. Compare `ylin` to the original image `xx` and to the square pulse interpolated image. Comment on the visual appearance of the two "reconstructed" images.

(e) Compute the frequency response of your linear interpolator used in the previous part. Only the 1-D FIR filter that acts on the rows must be analyzed. Plot its magnitude (frequency) response for $-\pi \leq \hat{\omega} \leq \pi$.

(f) *Smoothing via Lowpass Filtering:* At this point, you should be thinking that interpolation is very similar to lowpass filtering. To test this hypothesis, create a lowpass filtered version `xs_filt` of the sampled image by filtering the rows and columns with a 23-point FIR filter, whose coefficients are given by a modified "sinc" formula:

$$b_k = \frac{\sin(\pi(k-11)/4)}{\pi(k-11)/4} w_k \qquad k = 0, 1, 2, \ldots, 22$$

where $w_k$ is given by:

$$0.54 - 0.46 \cos\left(\frac{2\pi k}{22}\right) \qquad k = 0, 1, 2, \ldots, 22$$

Remember that MATLAB will have problems evaluating $b_{11}$ because the sinc function is an indeterminate form when $k = 11$. You will have to compute $b_{11}$ yourself and include it at the proper location in the vector $b_k$. Compare `xs_filt` to the original image. In addition, plot the magnitude of the frequency response for this FIR filter.

## 3.2 Zooming for an Image

Zooming in on a section of an image is very similar to the D-to-A reconstruction process because it also requires interpolation. The three interpolation systems (zero-order hold, linear interpolation, and lowpass filtering) developed in the previous section, can be applied to do zooming by a factor of four.

(a) Take a small patch of an image (about $50 \times 50$) where there is some interesting detail (e.g., the eye or feather of `lenna`). There are several ways to produce a larger image that appears zoomed. One way is to simply repeat pixel values. So in order to zoom a $50 \times 50$ section up to $200 \times 200$, you would repeat each pixel four times in each direction. Display a zoomed portion of an image by repeating pixel values.

(b) Another way to produce a larger image is to insert zeros between the existing samples. In other words, you must produce an image that is $200 \times 200$ with data values only in rows or columns whose index is a multiple of four. Consider the following code that does this for a 1-D vector.

```
L = length(xx);
yy = zeros(1,4*L);
yy(4:4:4*L) = xx;
```

Generalize this idea to write a function that will insert zeros in the rows and columns of a 2-D image.

(c) Now filter the image from part (b) to do the interpolation. Use both the linear and the "sinc" function interpolators.

(d) Comment on the ability of all three zooming operators to preserve detail and edges in the image while expanding the size of details. Try to explain your observations by considering the frequency content of the "zoomed" image.

(e) MATLAB has a zoom command in the Image Processing Toolbox. The function is called `imzoom`. If this is available on your system, try to determine what sort of interpolation scheme it is using.

# Lab 11
# Instructor Verification Sheet

Staple this page to the end of your Lab Memo Report.

Name: ______________________________ Date: __________

Part 2 Perform linear interpolation by hand:

Verified: ______________