C0E 472: DIGITAL SIGNAL PROCESSING

# Lab 06: Digital Images: A/D and D/A

INDEX NUMBER: 3050420

NAME: HARDY HAWA TUNTEIYA

COURSE: BSc. Biomedical Engineering

Year 4

# 1. INTRODUCTION

This report focuses on section 3 of this lab labelled "**Lab Exercises: Sampling, Aliasing and Reconstruction**". The purpose of this lab is to use digital images for studying the effects of sampling, aliasing, and reconstruction. The lab focuses on understanding how Analog-to-Digital (A/D) sampling and Digital-to-Analog (D/A) reconstruction processes are applied to digital images.

# 2. GIVEN FUNCTIONS

A function show_img was given to use for this lab. MATLAB has a function similar to it called 'imshow'. The show_img function is in the appendix.

# 3. LAB REPORT

In the warm-up section, a lighthouse picture was created and down sampled by a factor of 2. Below is the code and images for the original lighthouse and it's down sampled image.

```
load lighthouse.mat; %load image data, it is stored in xx
ww = xx; %storing data in new variable
ww_ds = ww(1:2:end, 1:2:end); %downscaling image and storing it in ww_ds

%Plot images
figure;
show_img(ww); %Original Image
figure;
show_img(ww_ds); %Down-sampled Image
```

OUTPUT


*Figure 1 Original Image*


*Figure 2 Lighthouse Image Down sampled by 2*

a. Compare original and down sampled images

Visually, the aliasing appears as jagged edges and wavy moiré patterns. The fence and areas with fine details show the aliasing effects most dramatically.

(b) This part is challenging: explain why the aliasing happens in the `lighthouse` image by using a "frequency domain" explanation. In other words, estimate the frequency of the features that are being aliased. Give this frequency as a number in cycles per pixel. (Note that the fence provides a sort of "spatial chirp" where the spatial frequency increases from left to right.) Can you relate your frequency estimate to the Sampling Theorem?

**Frequency domain explanation**

Explanation: Aliasing occurs because the high-frequency components of the image exceed the Nyquist frequency after down sampling. The features being aliased have spatial frequencies higher than half the original sampling rate. This causes these high-frequency components to fold back into lower frequencies, manifesting as visual artifacts.

Frequency Estimate: By examining the spatial frequency of the fence, the frequency of the aliased features is around 0.25 cycles per pixel. This corresponds to the high-frequency patterns that exceed the Nyquist limit when the image is down sampled by a factor of 2.

3.2 Reconstruction of Images

This section involves reconstructing a lighthouse image from one that is down sampled by a factor of 3. This is achieved by interpolating the missing samples.

```
%Lab Exercise 3.2: Reconstruction of Images
% Create the down sampled image by a factor of 3
xx3 = ww(1:3:end, 1:3:end);


% Display the down sampled image
figure;
title('Downsampled Lighthouse Image By 3');
show_img(xx3); %Downsampled Image by 3
```

OUTPUT


*Figure 3 Lighthouse down sampled with a scale of 3*

a. Learning how to reconstruct a one-dimensional signal using a zero-order hold interpolation

Zero-order hold interpolation is a method where each sample value is held constant over a specified interval before jumping to the next value.

```
%3.2a
xr1 = (-2).^(0:6);
L = length(xr1);  %Length is 7
nn = ceil((0.999 : 1: 4*L)/4);
xr1hold = xr1(nn);  %new value where each signal in xr1 is repeated 4 times
figure;
plot(xr1hold);     %plot signal
stem(nn, xr1hold); title('Stem Plot with Discrete Values');
xlabel('Sample Index');
ylabel('Signal Value');
grid on;
```

Explanation:

A 1-d signal `xr1` was created. `xr1` is a vector containing the values of $-2^n$. Where n is 0:6 (values from 0 to 6). This generates the sequence: [1, -2, 4, -8, 16, -32, 64]

The `nn` vector is code to repeat each element of `xr1` four times, creating a zero-order hold interpolation. The values in nn are [1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4 5 5 5 5 6 6 6 6 7 7 7 7] . ceil(X) means that rounds the elements of X to the nearest integers towards infinity. The interpolation factor is 4.

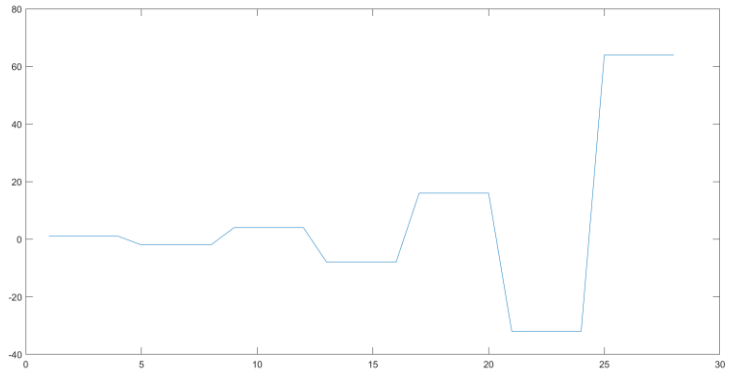



*Figure 4: Square Pulse of Interpolated xr1 Signal*

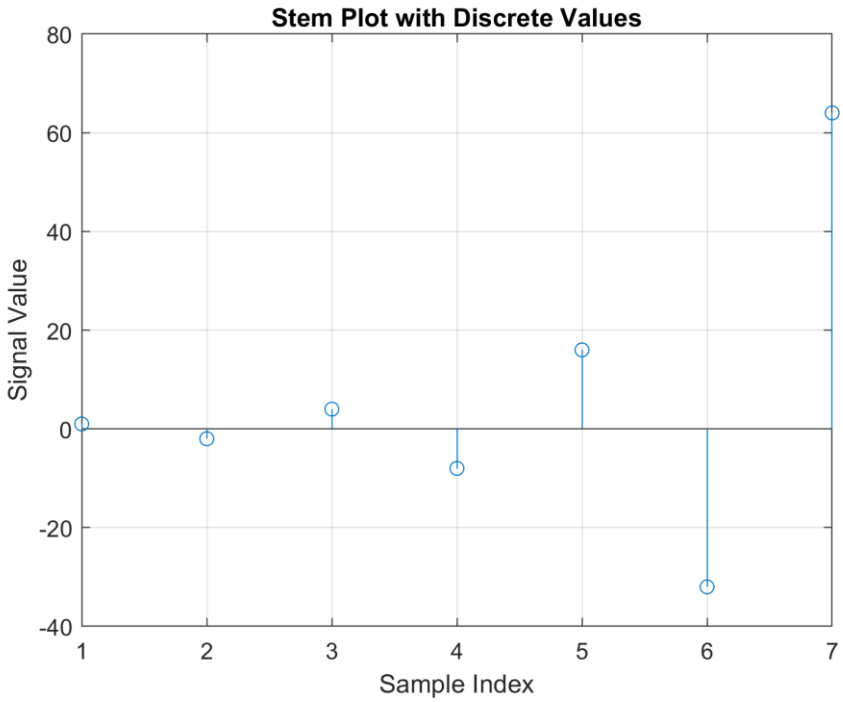



*Figure 5 Stem Plot*

I used the `stem` function to plot discrete data points, to visualize how the values have been repeated. The stem plot shows the stepped nature of the interpolated signal, where each original value is held constant for four samples, resembling a sequence of square pulses.

b. Process all the rows of xx3 using zero-hold order

```
%3.2b Interpolating rows
xx3 = ww(1:3:end, 1:3:end);
xholdrows = zeros(size(xx3,1), 3*size(xx3,2));
for i = 1:size(xx3,1)
    row = xx3(i,:);
    xholdrows(i,:) = row(ceil((0.999:1:3*length(row))/3));
end
figure;
show_img(xholdrows);
```

OUTPUT

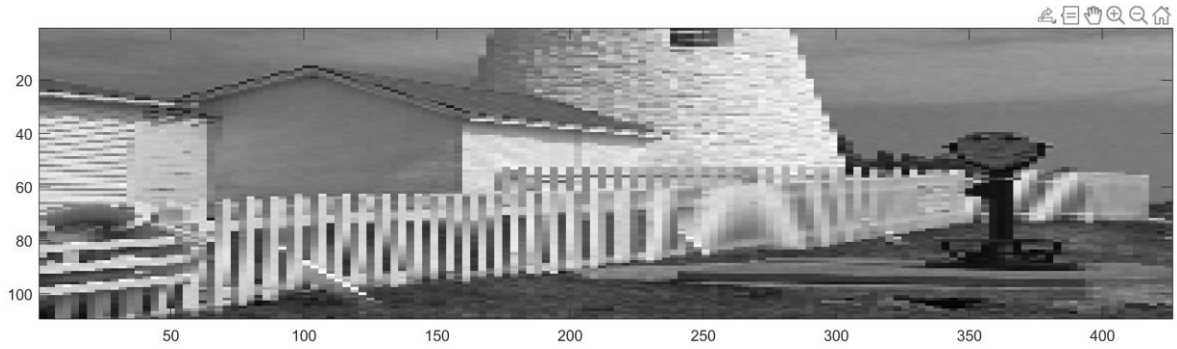


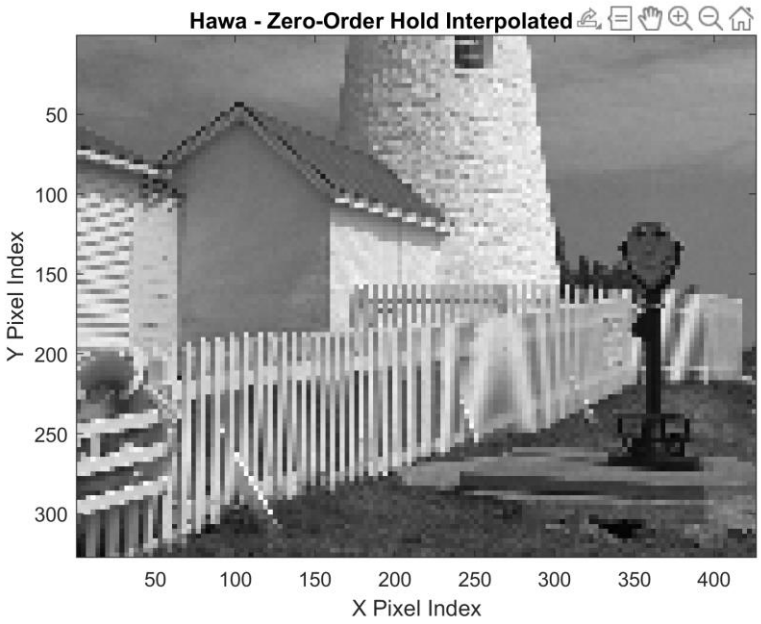

*Figure 6 Xholdrows Image - Rows Interpolated*

Comparing xx3 to this image; The xx3 is a 108*142 image, and xholdrows is a 108*426 image. The xhold image is not as clear as the original one.

c. Zero-hold order for columns

```
%3.2c Interpolating column
xhold = zeros(3*size(xholdrows,1), size(xholdrows,2));
for i = 1:size(xholdrows,2)
    col = xholdrows(:,i);
    xhold(:,i) = col(ceil((0.999:1:3*length(col))/3));
end
figure;
show_img(xhold);
title('Hawa - Zero-Order Hold Interpolated Image');
xlabel('X Pixel Index');
ylabel('Y Pixel Index');
```

OUTPUT

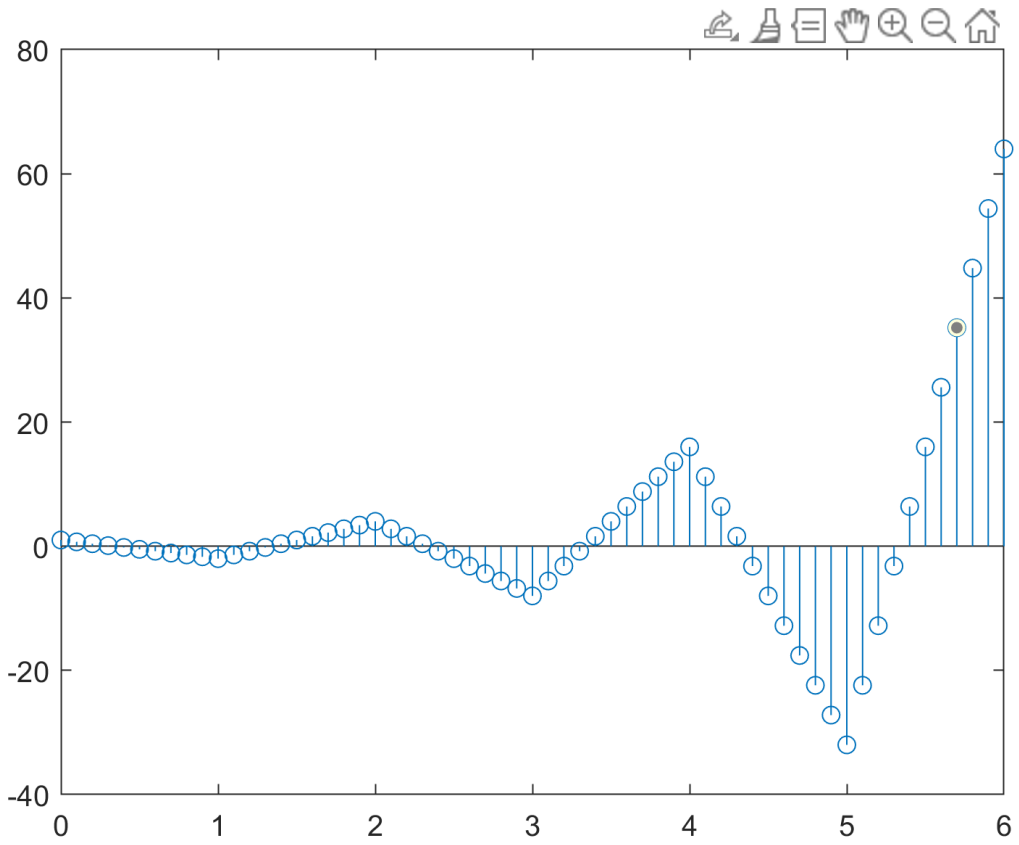



The `xhold` image, after zero-order hold interpolation in both rows and columns, is compared to the original image. The zero-order hold interpolation results in a blocky image with stair-step artifacts.

d. Linear Interpolation using `interp1` function

The interpolation factor is 10 and the sampling rate is 0.1.

```
%3.d Linear interpolation
n1 = 0:6;
xr1 = (-2).^n1;
tti = 0:0.1:6;
xr1linear = interp1(n1, xr1, tti);
figure;
stem(tti, xr1linear);
```




e. Linear interpolation for rows and columns

```
% Interpolate rows
xx3 = ww(1:3:end, 1:3:end);
[xrows, xcols] = size(xx3);
xinterp1 = zeros(xrows, 3*xcols);
for i = 1:xrows
    xinterp1(i,:) = interp1(1:xcols, xx3(i,:), 1:1/3:xcols, 'linear');
end


% Interpolate columns
[rows, cols] = size(xinterp1);
xinterp = zeros(3*rows, cols);
for i = 1:cols
    xinterp(:,i) = interp1(1:rows, xinterp1(:,i), 1:1/3:rows, 'linear');
end
figure;
show_img(xinterp);
```


f. Compare xxlinear to the original image
Visual comparison: The linear interpolation result is smoother than the zero-order hold result, especially in regions with gradual intensity changes. However, aliasing artifacts from the down sampled image remain.

g. Quality comparison

**Quality difference:**

**Zero-Order Hold Result:** The image is blocky with stair-step artifacts, especially noticeable in high-frequency regions like edges and fine details.

**Linear Interpolation Result:** The image is smoother, with fewer artifacts in low-frequency regions, but some aliasing artifacts remain in high-frequency areas.

Frequency Content Analysis:

**Edges and Fence Posts:** These are high-frequency features that are better represented by zero-order hold but still exhibit significant aliasing.

**Background:** This is a low-frequency feature that benefits more from linear interpolation, resulting in smoother transitions and less visible artifacts.

# APPENDIX

```
function [ph] = show_image(imc, figno, scaled, map)
%SHOW_IMG    display an image with possible scaling
% usage...   ph = show_img(imc, figno, scaled, map)
% imc = input image
% figno = figure number to use for the plot
%           if 0, re-use the same figure
%           if omitted a new figure will be opened
% optional args:
% scaled = 1 (TRUE) to do auto-scale (DEFAULT)
%           not equal to 1 (FALSE) to inhibit scaling
% map = user-specified color map
% ph = figure handle
%---


if nargin > 1
    if (figno > 0)
        figure(figno);
    end
else
    figure;
end
if nargin < 3
    scaled = 1;           %--- TRUE
end;
if (scaled)
    mx = max(max(imc));
    mn = min(min(imc));
    omc = round(255*(imc-mn)/(mx-mn));
elseif (~scaled)
    omc = round(imc);
    I = find(omc < 0);
    omc(I) = zeros(size(I));
    I = find(omc > 255);
    omc(I) = 255 * ones(size(I));
end;
if nargin < 4
    colormap(gray(256));  %--- linear color map
else
    omc = imc;
    colormap(map);
end;
ph = image(omc);
axis('image')
end
```