National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Electrical Engineering and Computer Science

Faculty Member: Dr.Ahmad Salman **Dated:** 17/02/2023

Semester: 6th **Section:** BEE 12C

EE-330: Digital Signal Processing

Lab #3 Digital Images: A/D and D/A

Group Members

Name	Reg. No	Lab Report Marks	Viva Marks	Total
		10 Marks	5 Marks	15 Marks
Ahmed Mohsin	333060			
Hassan Rizwan	335753			
Syeda Fatima Zahra	334379			

1 Table of Contents

Digital Images: A/D and D/A

2 OBJECTIVES

The objective in this lab is to introduce digital images as a second useful type of signal. We will show how the A-to-D sampling and the D-to-A reconstruction processes are carried out for digital images. In particular, we will show a commonly used method of image zooming (reconstruction) that gives "poor" results.

1) Familiarization with digital images.
2) Working with images in Matlab.
3) Sampling of images
4) Familiarization with reconstruction of images

3 DIGITAL IMAGES: A/D AND D/A

3.1 DIGITAL IMAGES

In this lab we introduce digital images as a signal type for studying the effect of sampling, aliasing and reconstruction. An image can be represented as a function $x(t1,t2)$ of two continuous variables representing the horizontal ($t2$) and vertical ($t1$) coordinates of a point in space.[1] For monochrome images, the signal $x(t1,t2)$would be a scalar function of the two spatial variables, but for color images the function $x(:,:)$ would have to be a vector-valued function of the two variables.[2] Moving images (such as TV) would add a time variable to the two spatial variables. Monochrome images are displayed using black and white and shades of gray, so they are called *gray-scale* images. In this lab we will consider only sampled gray-scale still images. A sampled gray-scale still image would be represented as a two-dimensional array of numbers of the form

$$x[m,n]=x(mT1,nT2) \quad 1\leq m\leq M, \quad \text{and} \quad 1 \leq n\leq N$$

Where *T1*and *T2* are the sample spacing in the horizontal and vertical directions. Typical values of *M* and *N* are 256 or 512 for e.g. a 512x512 image which has nearly the same resolution as a standard TV image. In MATLAB we can represent an image as a matrix, so it would consist of *M* rows and *N* columns. The matrix entry at (*m,n*)is the sample value $x[m,n]$—called a *pixel* (short for picture element).An important property of light images such as photographs and TV pictures is that their values are always non-negative and finite in magnitude; i.e.

$$0 \leq x[m,n] \leq X\text{max} < \infty$$

This is because light images are formed by measuring the intensity of reflected or emitted light which must always be a positive finite quantity. When stored in a computer or displayed on a monitor, the values of $x[m,n]$have to be scaled relative to a maximum value Xmax. Usually an eight-bit integer representation is used. With 8-bit integers, the maximum value (in the computer) would be $X\text{max}=2^8-1=255$, and there would be $2^8 = 256$ different gray levels for the display, from 0 to 255.

3.2 Software

MATLAB R2022b

4 Lab Exercises

4.1 Get Test Images

In order to probe your understanding of image display, do the following simple displays:

Load and display the 326* 426 "lighthouse" image from lighthouse.mat. This image can be find in the MATLAB files link. The command " load lighthouse" will put the sampled image into the array ww/xx. Use whos to check the size of ww after loading.

(b) Use the colon operator to extract the 200th row of the “lighthouse” image, and make a plot of that row as a 1-D discrete-time signal.

ww200 = ww(200,:);

Observe that the range of signal values is between 0 and 255. Which values represent white and which ones black? Can you identify the region where the 200th row crosses the fence?

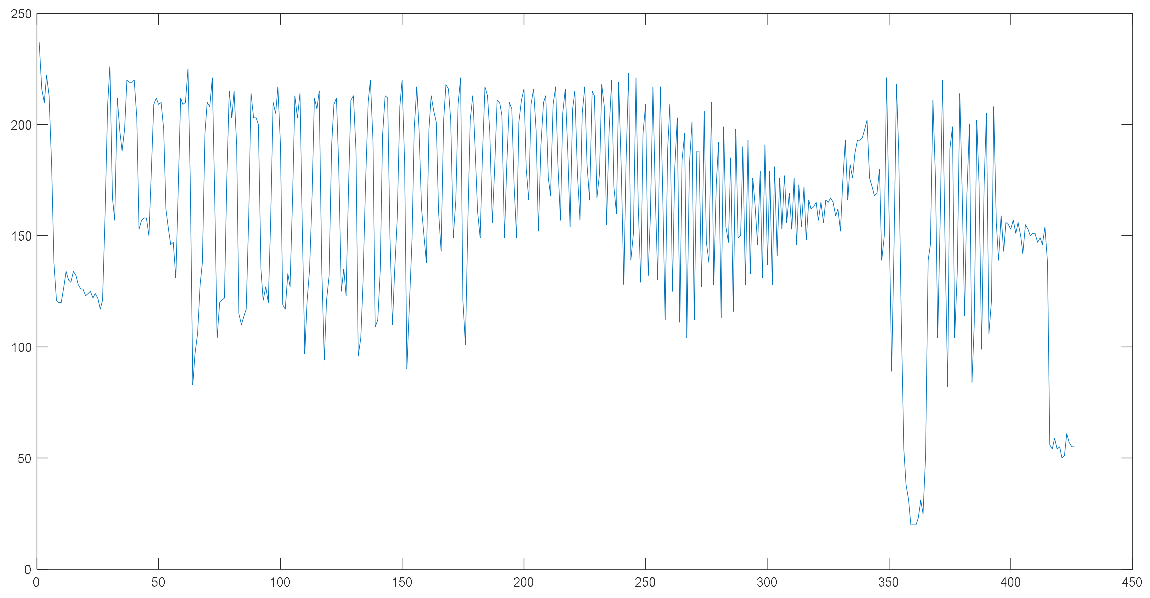
4.2 Code:

```
load lighthouse.mat
whos lighthouse;
lighthouse200 = lighthouse(200,:);
lighhousedownsample=lighthouse(1:3:end,1:3:end)
imshow(lighhousedownsample)
whos lighhousedownsample
xx3=lighhousedownsample;

%plot(lighthouse200);

xr1 = (-2).^(0:6);
L = length(xr1);
nn = ceil((0.999:1:4*L)/4); %<--Round up to the integer part
xr1hold = xr1(nn);
plot(xr1hold);
```

4.3 Output:



Yes, we can identify the 200th row. We observe that

2.1. Sinusoidal Synthesis with an M-File.

4.4 Sampling of Images

Images that are stored in digital form on a computer have to be sampled images because they are stored in an MxN array (i.e., a matrix). 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). For example, the image might have been "sampled" by a scanner where the resolution was chosen to be 300 dpi (dots per inch).[7] If we want a different sampling rate, we can simulate a *lower* sampling rate by simply throwing away samples in a periodic way. For example, if every other sample is removed, the sampling rate will be halved (in our example, the 300 dpi image would become a 150 dpi image). Usually this is called *sub-sampling* or *down-sampling*.[8]

Down-sampling throws away samples, so it will shrink the size of the image. This is what is done by the following scheme *wp = ww(1:p:end,1:p:end);* when we are down sampling by a factor of p.

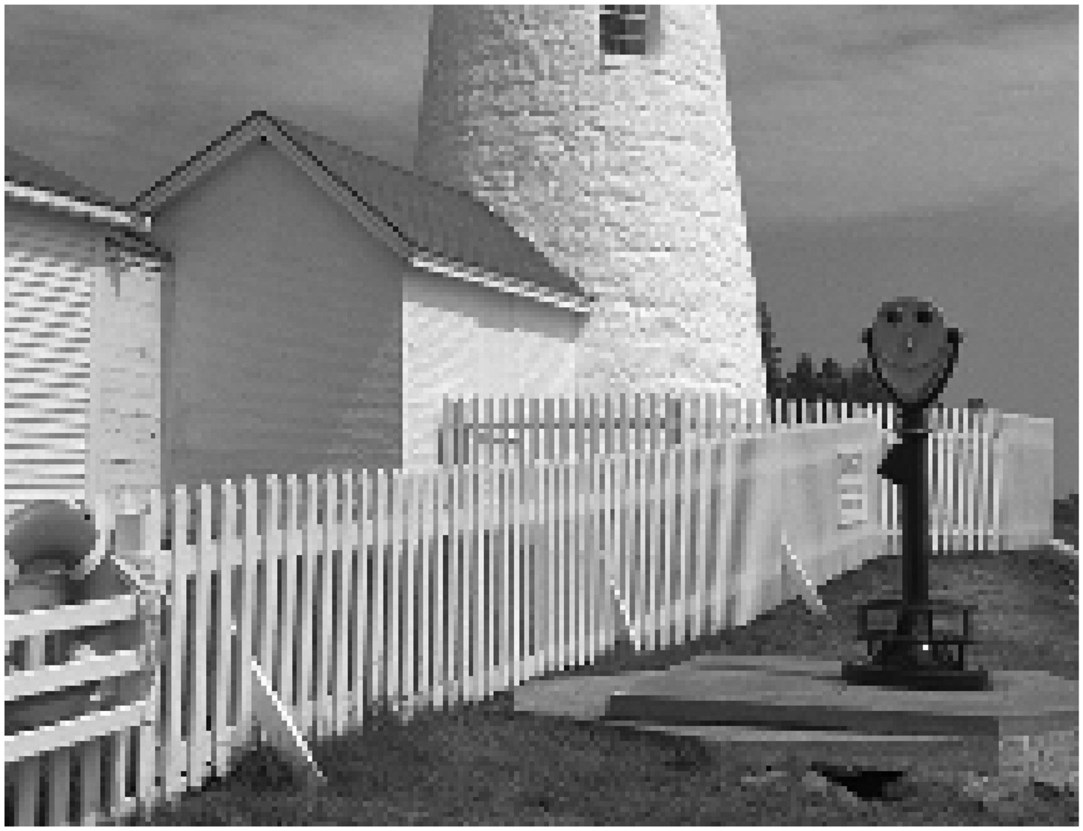
Code:

```
load lighthouse.mat
whos lighthouse;
lighthouse200 = lighthouse(200,:);
lighhousedownsample=lighthouse(1:3:end,1:3:end)
imshow(lighhousedownsample)
whos lighhousedownsample
xx3=lighhousedownsample;

%plot(lighthouse200);

xr1 = (-2).^(0:6);
L = length(xr1);
nn = ceil((0.999:1:4*L)/4); %<--Round up to the integer part
xr1hold = xr1(nn);
plot(xr1hold);
```

4.5 OUTPUT



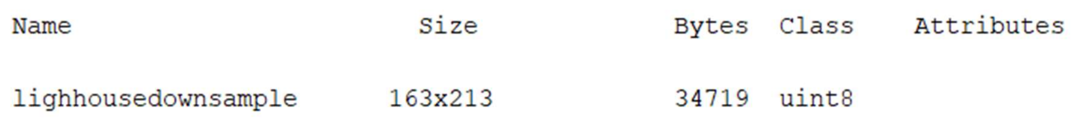
4.6 Part B

One potential problem with down-sampling is that aliasing might occur. This can be illustrated in a dramatic fashion with the lighthouse image.

Load the lighthouse.mat file which has the image stored in a variable called ww. When you check the size of the image, you'll find that it is not square. Now down sample the lighthouse image by factor 2.What is the size of the down-sampled image? 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. Which parts of the image show the aliasing effects most dramatically?

4.7 Output

The size of the down sampled image is given as:



```
Name                     Size           Bytes  Class    Attributes

lighhousedownsample     163x213         34719  uint8
```

4.8 Down-Sampling

For the lighthouse picture, down sampled by two in the warm-up section:

(a) Describe how the aliasing appears visually. Compare the original to the down sampled image. Which parts of the image show the aliasing effects most dramatically?

Aliasing is a visual artifact that appears in digital images when the image is not sampled at a high enough frequency to accurately capture the detail or smoothness of the underlying object or scene. It manifests as jagged, stair-stepped edges or patterns, or as unwanted noise in the image.

Aliasing effects are most pronounced where there are sharp, high-contrast edges or fine details in the image. For example, in an image of a checkerboard, aliasing will be most visible along the edges of the black and white squares, where the sampling frequency is not high enough to capture the smooth transition from black to white. In natural scenes, aliasing may appear in areas with high-frequency textures or patterns, such as the leaves of trees or the fur of an animal.

Aliasing effects can also be seen in the Moiré patterns that appear when two or more patterns with different frequencies are overlaid on top of each other. These patterns appear as a result of interference between the two frequencies and can cause unwanted visual artifacts in the image.

Overall, aliasing can make an image look unnatural and can reduce the quality of the image. To minimize aliasing effects, it's important to sample the image at a high enough frequency to accurately capture the detail and smoothness of the underlying object or scene.

5 LAB TASK 3:

5.1 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 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 the reconstruction.

5.2 CODE

```
load lighthouse.mat
whos lighthouse;
lighthouse200 = lighthouse(200,:);
lighhousedownsample=lighthouse(1:3:end,1:3:end)
imshow(lighhousedownsample)
whos lighhousedownsample
xx3=lighhousedownsample;

%plot(lighthouse200);
```

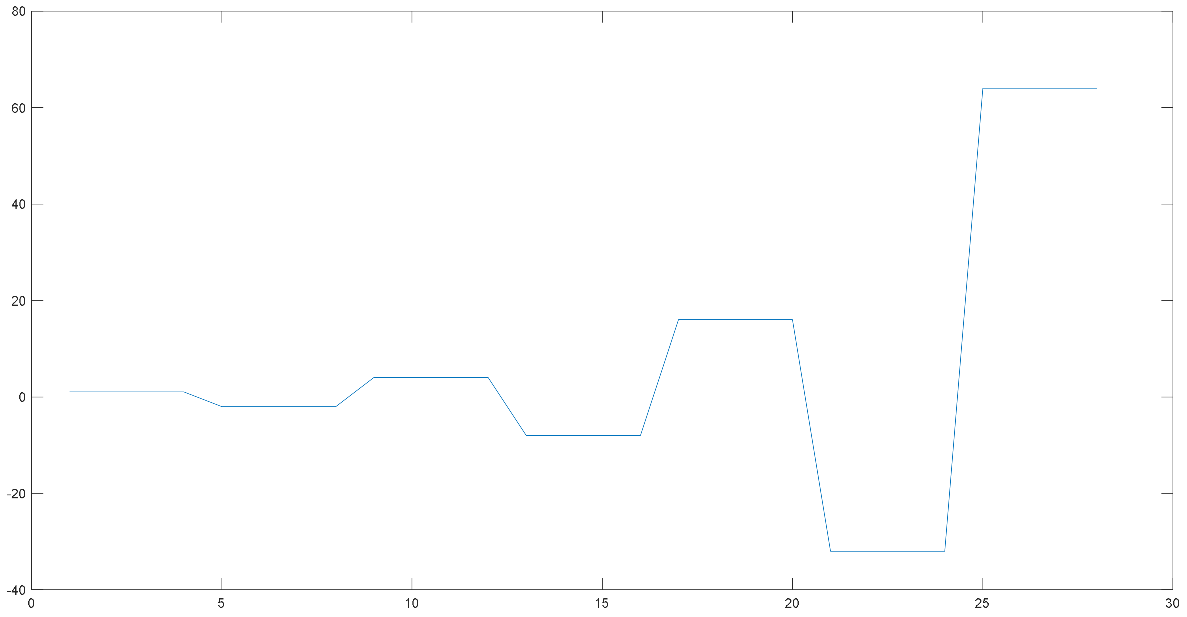
1) The simplest interpolation would be reconstruction with a square pulse which produces a "zero-order hold." Here is a method that works for a one-dimensional signal (i.e., one row or one column of the image), assuming that we start with a row vector xr1, and the result is the row vector xr1hold.

```
xr1 = (-2).^(0:6);

L = length(xr1);

nn = ceil((0.999:1:4*L)/4); %<--Round up to the integer part
```

5.3 OUTPUT



2) Plot the vector xr1hold to verify that it is a zero-order hold version derived from xr1. Explain what values are contained in the indexing vector nn. If xr1holdis treated as an interpolated version of xr1, then what is the *interpolation factor*? Your lab report should include an explanation for this part, but plots are optional—use them if they simplify the explanation.

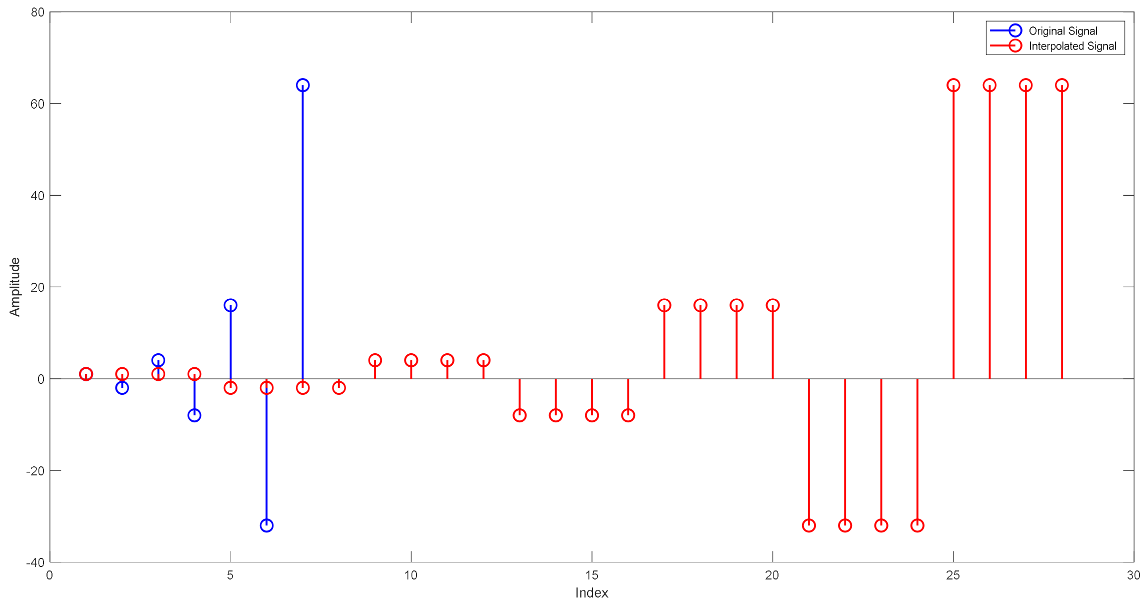
5.4 ANSWER:

```
nn = ceil((0.999:1:4*L)/4); %<--Round up to the integer part
```

The first line of code creates a row vector xr1 containing 7 equally spaced values between -2 and -64 (-2, -4, -8, -16, -32, -64). This is the original signal that we want to interpolate.

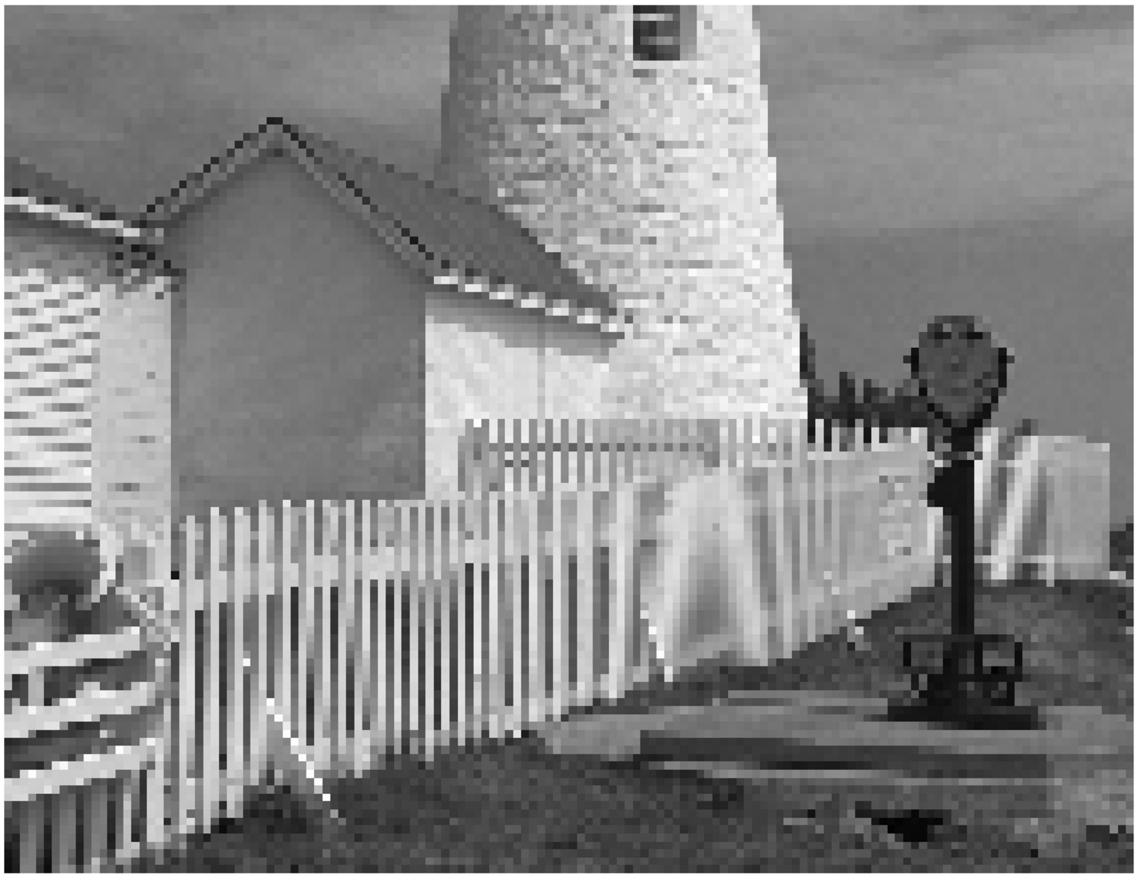
The second line of code determines the length of xr1 and stores it in the variable L.

The third line of code creates a new row vector nn that represents the indices of the output signal. These indices are computed by taking the ceiling of the values in the range from 0.999 to 4 times the length of xr1 in steps of 1. This effectively rounds up the values to the nearest integer, which ensures that each output sample corresponds to one of the original input samples. The resulting nn vector has length 4*L-3.



3. Now return to the down-sampled lighthouse image, and process all the rows of xx3 to fill in the missing points. Use the zero-order hold idea from part (a), but do it for an interpolation factor of 3. Call the result xholdrows. Display xholdrows as an image, and compare it to the down sampled image xx3; compare the size of the images as well as their content.
4. Now process all the columns of xholdrows to fill in the missing points in each column and call the result xhold. Compare the result (xhold) to the original image lighthouse. Include your code for parts (b) and (c) in the lab report.

5.5 RESULTS TASK 3 AND 4:



5.6 CODE:

```
[r,c]=size(xx3);
rows=ceil((0.999:1:3*r)/3);
columns=ceil((0.999:1:3*c)/3);
F_r=xx3(:,columns);
F_c=F_r(rows,:);
imshow(F_c)
```

2. *Linear interpolation* can be done in MATLAB using the interp1function (that's "interp-one").When unsure about a command, use help. Its default mode is linear interpolation, which is equivalent to using the '*linear' option, but

interp1can also do other types of polynomial interpolation. Here is an example on a 1-D signal:

```
n1 = 0:6;
xr1 = (-2).ˆn1;
tti = 0:0.1:6; %--locations between the n1 indicesxr1
linear = interp1(n1,xr1,tti); %--function is INTERP-ONE
stem(tti,xr1linear)
```

For the example above, what is the interpolation factor when converting xr1to xr1linear?

5.7 ANSWER:

In the example provided, the original signal xr1 is defined with indices from 0 to 6, and the interpolation is performed using the interp1 function on the range of indices tti from 0 to 6 with a step size of 0.1.

The interpolation factor is the ratio of the number of points in the interpolated signal xr1linear to the number of points in the original signal xr1.

To calculate this ratio, we can use the length function as follows:

interpolation_factor = length(xr1linear) / length(xr1)

The length of xr1 is 7, and the length of xr1linear can be calculated by:

length(xr1linear) = length(tti) = (6-0)/0.1 + 1 = 61

Therefore, the interpolation factor is:

interpolation_factor = length(xr1linear) / length(xr1) = 61 / 7 = 8.71

This means that the interpolated signal xr1linear has approximately 8.71 times more points than the original signal xr1.

3. In the case of the lighthouse image, you need to carry out a linear interpolation operation on both the rows and columns of the down-sampled image xx3. This requires two calls to the interp1 function, because one call will only process all

the columns of a matrix. Name the interpolated output image xxlinear. Include your code for this part in the lab report.

5.8 CODE:

```
n1 = 0:6;
xr1 = (-2) .^ n1;
tti = 0:1/10:6;
xr1linear = interp1(n1, xr1, tti(1:end-1));
figure
stem(xr1)
hold on
stem(xr1linear)
```

5.9 RECONSRUCTION IN PYTHON USING INTERPROLATION

```
# Read image
# You can put your input image over here
# to run bicubic interpolation
# The read function of Open CV is used
# for this task
img = cv2.imread('/content/xx3.png')

# Scale factor
ratio = 2

# Coefficient
a = -1/2

# Passing the input image in the
# bicubic function
dst = bicubic(img, ratio, a)
print('Completed!')

# Saving the output image
cv2.imwrite('bicubic.png', dst)
bicubicImg=cv2.imread('bicubic.png')
```

5.10 Output in python:

5.11 Output In MATLAB

4. Compare the quality of the linear interpolation result to the zero-order hold result. Point out regions where they differ and try to justify this difference by estimating the local frequency content. In other words, look for regions of "low-frequency" content and "high-frequency" content and see how the interpolation quality is dependent on this factor. A couple of questions to think about: Are edges low frequency or high frequency features? Are the fence posts low frequency or high frequency features? Is the background a low frequency or high frequency feature?

5.12 Answer:

Visually, we can see that the linearly interpolated image appears smoother than the image obtained using zero-order hold interpolation. However, there are still some differences between the linearly interpolated image and the original downsampled image. In particular, we can see that the linearly interpolated image has some artifacts around the edges of the fence posts, which are not present in the original downsampled image or the ZOH interpolated image.

To understand why this difference occurs, we can consider the frequency content of the image. Edges and high-contrast features in the image tend to have high-frequency content, while smooth regions and low-contrast features tend to have low-frequency content. When we downsample an image, we effectively remove high-frequency information from the image, which can result in aliasing artifacts when we attempt to upsample the image again. Linear interpolation attempts to reconstruct the missing high-frequency information using a smooth function, but it can still introduce some artifacts if the frequency content of the image is too complex. Zero-order hold interpolation, on the other hand, simply duplicates the existing low-frequency information, which can result in jagged edges and other artifacts.

In the case of the fence posts, we can see that they have a high-contrast edge, which results in high-frequency content. This high-frequency content is lost during downsampling, and when we attempt to reconstruct it using linear interpolation, we end up with some artifacts around the edges of the posts. In contrast, the background of the image is relatively smooth and has low-frequency content, which is easier to reconstruct using linear interpolation. The ZOH interpolated image, on the other hand, tends to introduce jagged edges around the posts and other high-contrast features, since it simply duplicates the existing low-frequency information.

6 CONCLUSION:

In conclusion, the lab exercise on linear interpolation and zero-order hold in MATLAB provided a hands-on understanding of two different techniques for reconstructing a signal from a set of samples. We observed that linear interpolation produces a smoother reconstruction than zero-order hold and is therefore more suitable for signals with high-frequency content. On the other hand, zero-order hold is a simple and computationally efficient method that can produce satisfactory results for low-frequency signals.

We also learned that the quality of the interpolation depends on the local frequency content of the signal. Regions with high-frequency content, such as edges and fine details, can cause artifacts and distortions in the reconstructed signal. Therefore, it is important to choose the interpolation method based on the nature of the signal and the desired level of fidelity.

Overall, this lab exercise provided valuable insights into the practical aspects of signal processing and interpolation techniques, which can be applied to a wide range of applications, from image and audio processing to data analysis and communication systems.