GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

# ECE 2025 Spring 2004
# Lab #5: Digital Images: A/D and D/A

Date: 11 – 17 Feb 2004

---

**You should read the Pre-Lab section of the lab and do all the exercises in the Pre-Lab section before your assigned lab time.** You **MUST** complete the online Pre-Post-Lab exercise on Web-CT at the beginning of your scheduled lab session. You can use MATLAB and also consult your lab report or any notes you might have, but you cannot discuss the exercises with any other students. You will have approximately 20 minutes at the beginning of your lab session to complete the online Pre-Post-Lab exercise. The Pre-Post-Lab exercise for this lab includes some questions about concepts from the previous Lab report as well as questions on the Pre-Lab section of this lab.

The Warm-up section of each lab must be completed **during your assigned Lab time** and the steps marked *Instructor Verification* must also be signed off **during the lab time**. One of the laboratory instructors must verify the appropriate steps by signing on the **Instructor Verification** line. When you have completed a step that requires verification, simply raise your hand and demonstrate the step to the TA or instructor. After completing the warm-up section, turn in the verification sheet to your TA.

It is only necessary to turn in Section 4 as this week's lab report. More information on the lab report format can be found on Web-CT under the "Information" link. You should **label** the axes of your plots and include a title and Figure number for every plot. Every plot should be referenced by Figure number in your text discussion. In order to make it easy to find all the plots, include each plot *inlined* within your report. For more information on how to include figures and plots from MATLAB to your report file, consult the "Information" link on Web-CT. If you still do not know how to do so, ask your TA.

***Forgeries and plagiarism are a violation of the honor code and will be referred to the Dean of Students for disciplinary action. You are allowed to discuss lab exercises with other students and you are allowed to consult old lab reports but the submitted work should be original and it should be your own work.***

The lab report for this week will be an **Informal Lab Report.**

The report will be **due during the period 18–24 Feb. at the start of your lab.**

---

## 1 Introduction

The objective in this lab is to introduce digital images as a second useful signal type. 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—a later lab will revisit this issue and do a better job.

## 2 Pre-Lab

### 2.1 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 $x(t_1, t_2)$ of two continuous variables representing the horizontal ($t_2$) and vertical ($t_1$) coordinates of a point in space.[1] For monochrome images, the signal

---

[1] The variables $t_1$ and $t_2$ *do not denote time, they represent spatial dimensions.* Thus, their units would be inches or some other unit of length.

$x(t_1, t_2)$ would be a scalar function of the two spatial variables, but for color images the function $x(\cdot, \cdot)$ 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

$$x[m, n] = x(mT_1, nT_2) \qquad 1 \leq m \leq M, \text{ and } 1 \leq n \leq N$$

where $T_1$ and $T_2$ are the sample spacings in the horizontal and vertical directions. Typical values of $M$ and $N$ are 256 or 512; e.g., a $512 \times 512$ 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.,

$$0 \leq x[m, n] \leq X_{\max}$$

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 $X_{\max}$. Usually an eight-bit integer representation is used. With 8-bit integers, the maximum value (in the computer) would be $X_{\max} = 2^8 - 1 = 255$, and there would be $2^8 = 256$ gray levels for the display, from 0 to 255.

## 2.2 Displaying Images

As you will discover, the correct display of an image on a computer monitor can be tricky, especially after some processing has been performed on the image. We have provided the function `show_img.m` in the *SP-First* toolbox to handle most of these problems,[3] but it will be helpful if the following points are noted:

CD-ROM
`show_img.m`

1. All image values must be non-negative for the purposes of display. Filtering may introduce negative values, especially if differencing is used (e.g., a high-pass filter).

2. The default format for most gray-scale displays is eight bits, so the pixel values $x[m, n]$ in the image must be converted to integers in the range $0 \leq x[m, n] \leq 255 = 2^8 - 1$.

3. The actual display on the monitor is created with the `show_img` function.[4] The `show_img` function will handle the color map and the "true" size of the image. The appearance of the image can be altered by running the pixel values through a "color map." In our case, we want "grayscale display" where all three primary colors (red, green and blue, or RGB) are used equally, creating what is called a "gray map." In MATLAB the `gray` color map is set up via

   ```
   colormap(gray(256))
   ```

   which gives a $256 \times 3$ matrix where all 3 columns are equal. The function `colormap(gray(256))` creates a linear mapping, so that each input pixel amplitude is rendered with a screen intensity proportional to its value (assuming the monitor is calibrated). For our lab experiments, non-linear color mappings would introduce an extra level of complication, so we won't use them.

---

[2] For example, an RGB color system needs three values at each spatial location: one for red, one for green and one for blue.

[3] If you have the MATLAB Image Processing Toolbox, then the function `imshow.m` can be used instead.

[4] If the MATLAB function `imagesc.m` is used to display the image, two features will be missing: (1) the color map may be incorrect because it will not default to gray, and (2) the size of the image will not be a true pixel-for-pixel rendition of the image on the computer screen.

4. When the image values lie outside the range [0,255], or when the image is scaled so that it only occupies a small portion of the range [0,255], the display may have poor quality. In this lab, we will use `show_img.m` to *automatically rescale the image:* This requires a linear mapping of the pixel values:[5]

$$x_s[m,n] = \mu x[m,n] + \beta$$

The scaling constants $\mu$ and $\beta$ can be derived from the min and max values of the image, so that all pixel values are recomputed via:

$$x_s[m,n] = \left\lfloor 255.999 \left( \frac{x[m,n] - x_{\min}}{x_{\max} - x_{\min}} \right) \right\rfloor$$

where $\lfloor x \rfloor$ is the floor function, i.e., the greatest integer less than or equal to $x$.

## 2.3 MATLAB Function to Display Images

CD-ROM

*IMAGE DATA FILES*

You can load the images needed for this lab from `*.mat` files, or from `*.png` files. Image files with the extension `*.png` can be read into MATLAB with the `imread` function. Any file with the extension `*.mat` is in MATLAB format and can be loaded via the `load` command. To find some of these files, look for `*.mat` in the *SP-First* toolbox or in the MATLAB directory called `toolbox/matlab/demos`. Some of the image files are named `lenna.mat`, `echart.mat` and `zone.mat`, but there are others within MATLAB's demos. The default size is 256 × 256, but alternate versions are available as 512 × 512 images under names such as `lenna512.mat` and `zone512.mat`. After loading, use the command `whos` to determine the name of the variable that holds the image and its size.

CD-ROM

`show_im.m`

Although MATLAB has several functions for displaying images on the CRT of the computer, we have written a special function `show_img()` for this lab. It is the visual equivalent of `soundsc()`, which we used when listening to speech and tones; i.e., `show_img()` is the "D-to-C" converter for images. This function handles the scaling of the image values and allows you to open up multiple image display windows. Here is the `help` on `show_img`:

```
function [ph] = show_img(img, figno, scaled, map)
%SHOW_IMG    display an image with possible scaling
% usage:  ph = show_img(img, figno, scaled, map)
%    img = 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 returned to caller
%----
```

Notice that unless the input parameter `figno` is specified, a new figure window will be opened.

## 2.4 Get Test Images

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

(a) Load and display the 326 × 426 "lighthouse" image from `lighthouse.png`. This image can be downloaded from Web-CT. The MATLAB command `ww = imread('lighthouse.png')` will put the sampled image into the array `ww`, Use `whos` to check the size and type of `ww` after loading.

---

[5] The MATLAB function `show_img` has an option to perform this scaling while making the image display.

Notice that the array type for `ww` is `uint8`, so it is necessary to convert `ww` to double precision floating-point with the MATLAB command `double(ww);`. When you display the image it might be necessary to set the colormap via `colormap(gray(256))`.

(b) Use the colon operator to extract the 440th row of the "lighthouse" image, and make a plot of that row as a 1-D discrete-time signal.

```
ww440 = ww(440,:);
```

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 440th row crosses the fence? Can you match up a black region between the image and the 1-D plot of the 440th row?

# 3 Warm-up

The instructor verification sheet may be found at the end of this lab.

## 3.1 Synthesize a Test Image

In order to probe your understanding of the relationship between MATLAB matrices and image display, you can generate a synthetic image from a mathematical formula.

(a) Generate a simple test image in which all of the columns are identical by using the following *outer product*:

```
xpix = cos(2*pi/32*(1:256)')*cos(2*pi/128*(1:256));
```

Display the image and explain the gray-scale pattern that you see. Why does one direction have more bright "dots" than the other? How can you predict the number of dots along vertical direction from the formula for `xpix`?

(b) In the previous part, which data value in `xpix` is represented by white? which one by black?

(c) Explain how you would produce an image with white areas at the four corners and in the middle. Give the formula that would create such a 512 × 512 image. Write the MATLAB code to make this image and display it.

**Instructor Verification** (separate page)

## 3.2 Printing Multiple Images on One Page

The phrase "what you see is what you get" can be elusive when dealing with images. It is ***very tricky*** to print images so that the hard copy matches exactly what is on the screen, because there is usually some interpolation being done by the printer or by the program that is handling the images. One way to think about this in signal processing terms is to think of the screen as one kind of D-to-A and the printer as another kind; each one uses a different kind of (D-to-A) reconstruction method to get the continuous-domain (analog) output image that you see.

Furthermore, if you try to put two images of different sizes into subplots of the same MATLAB figure, it won't work because MATLAB wants to force them to be the same size. Therefore, you should display your images in separate MATLAB Figure windows. In order to get a printout with multiple images on one page, use one of the following procedures:

1. Use MATLAB's `notebook` feature so that the images are sent directly to MS-Word and manipulated within the MS-Word document.

2. In MATLAB, use `show_img` and `trusize` to put your images into separate figure windows at the correct pixel resolution. Then you must also do the following:
    (a) Use the Windows program called `PAINT` to assemble the different images onto one page. This program can be found under `Accessories.`
    (b) For each MATLAB figure window, do `ALT-PRINT-SCREEN` which will copy the active window contents to the clipboard.
    (c) After each "window capture" in step 3, paste the clipboard contents into `PAINT.`[6]
    (d) Arrange the images so that you can make a comparison for your lab report.
    (e) Print the assembled images from `PAINT` to a printer.

### 3.3 Sampling of Images

Images that are stored in digital form on a computer have to be sampled images because they are stored in an $M \times N$ array (i.e., a matrix). The sampling rate in the two spatial dimensions was chosen when 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 downsampling by a factor of `p`.

(a) One potential problem with down-sampling is that aliasing might occur. This can be illustrated in a dramatic fashion with the `lighthouse` image. Note that aliasing and blurring are different phenomena in images, so when you examine the image below try to isolate the parts of the image where the distortion can be blamed on aliasing.

Read in the `lighthouse.png` file with the MATLAB function `imread`. When you check the size of the image, you'll find that it is not square. Now down-sample the `lighthouse` image by a factor of 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.[9] Which parts of the image show the aliasing effects most dramatically? Explain why the aliasing is happening by thinking about high frequencies in the image.

**Instructor Verification** (separate page)

---

[6] An alternative is to use the free program called IRFANVIEW, which can do image editing and also has screen capture capability. It can be obtained from `www.irfanview.com`. Other alternatives are Photoshop, or the GIMP at `www.gimp.org/win32`.

[7] For this example, the sampling periods would be $T_1 = T_2 = 1/300$ inches.

[8] The Sampling Theorem applies to digital images, so there is a *Nyquist Rate* that depends on the maximum *spatial* frequency in the image.

[9] 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. In the *SP-First* toolbox, an equivalent function called `trusize.m` is provided.

# 4 Lab Exercises: Sampling, Aliasing and Reconstruction

The objective of this lab exercise is to reconstruct an image from samples taken on a *quincunx* grid, i.e., a checkerboard pattern.

## 4.1 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 the reconstruction.

For these reconstruction experiments, use the `stinger04q` image, which has been sampled on a quincunx grid (see Fig. 1). When you display this image you will see the checkerboard pattern because each of the missing pixels has been replaced by a 0. The original `stinger04` image[10] prior to sampling is also available.

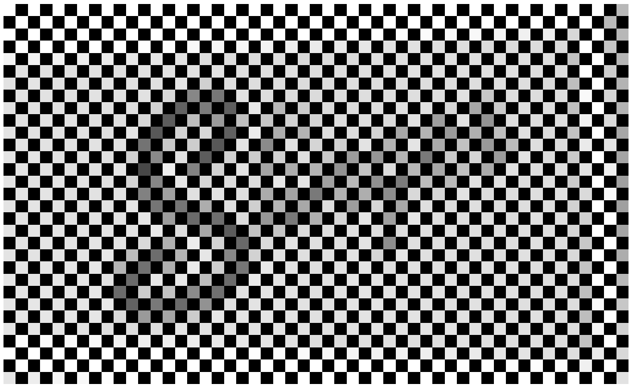

Figure 1: Checkerboard pattern of quincunx sampling. Black indicates missing pixels.

The reconstruction process involves three steps: (1) interpolate the rows of `stinger04q`, (2) interpolate the columns of of `stinger04q`, and then (3) average the two interpolated images together. Furthermore, when doing the interpolations, take into account that the even rows and odd rows must be treated a bit differently.

(a) 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 `zz`.

```
xr1 = (-2).^(0:6);
L = length(xr1);
nn = ceil((0.999:1:5*L)/5);    %<-- Round up to the integer part
xr1hold = xr1(nn);
```

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 `xr1hold` is 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.

(b) Now return to the quincunx-sampled `stinger04q` image, and process all the rows of `stinger04q` to fill in the missing points. Use the zero-order hold idea from part (a), but do it for an interpolation factor of 2. Call the result `stholdrows`. When writing the MATLAB code for this, you will need

---

[10] The "stinger" image was taken by Peter Jensen in 2001, and is used in ECE-2025 with his permission.

two separate cases for the even and odd rows. Likewise, process all the columns of `stinger04q` to fill in the missing points, and call the result `stholdcols.`

Display the `stholdrows` and `stholdcols` images, and compare them to each other and to the original. Identify regions of the image that could be considered "high frequency" in order to find places where the interpolated images differ from the original.

Include your code for this part in the lab report.

(c) It is likely that the row-interpolated and column-interpolated images will exhibit certain biases for horizontal or vertical features within the image. Therefore, it is possible to create a compromise by averaging those two results. Call the averaged image `stholdavg`.

```
stholdavg = 0.5*stholdrows + 0.5*stholdcols
```

Display this image and compare it to the original.

(d) *Linear interpolation* can be done in MATLAB using the `interp1` function (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 `interp1` can also do other types of polynomial interpolation. Here is an example on a 1-D signal:

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

For the example above, what is the interpolation factor when converting `xr1` to `xr1linear`?

(e) In the case of the `stinger04q` image, you need to carry out a linear interpolation operation on both the rows and columns of the quincunx-sampled image `stinger04q` and average the results. This requires multiple calls to the `interp1` function, because one call will only process all the even (or odd) columns of a matrix.[11] Name the interpolated output image `stlinear`. Include your code for this part in the lab report.

(f) Compare `stlinear` to the original image `stinger04`. Comment on the visual appearance of the "reconstructed" image versus the original; point out differences and similarities. Can the reconstruction (i.e., zooming) process remove the aliasing effects from the down-sampled `stinger04` image?

(g) 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 tree branches low frequency or high frequency features? Are the panels on the side of the bus low frequency or high frequency features? How about the windows in the building? Or, the cracked glass on the bus?

*Comment:* You might use MATLAB's zooming feature to show details in a small patches of the output image. However, be careful because zooming does its own interpolation, probably a zero-order hold.

---

[11] You can do the rows by using a matrix transpose to turn rows into columns.

## 4.2 More about Images in MATLAB (Optional)

This section[12] is included for those students who might want to relate these MATLAB operations to previous experience with software such as *Photoshop*. There are many image processing functions in MATLAB. For example, try the help command:

```
help images
```

for more information, but keep in mind that the Image Processing Toolbox, which is available in the ECE computer labs, may not be on your computer.

### 4.2.1 Zooming in Software

If you have used an image editing program such as Adobe's *Photoshop*, you might have observed how well or how poorly image zooming (i.e., interpolation) is done. For example, if you try to blow up a JPEG file that you've downloaded from the web, the result is usually disappointing. Since MATLAB has the capability to read lots of different formats, you can apply the image zooming via interpolation to any photograph that you can acquire. The MATLAB function for reading JPEG images is `imread( )` which would be invoked as follows:

```
xx = imread('foo.jpg','jpeg');
```

Since `imread( )` is part of the image processing toolbox, this test can be done in the ECE computer labs, but may not be possible on your home computer.

### 4.2.2 Warnings

Images obtained from JPEG files might come in many different formats. Two precautions are necessary:

1. If MATLAB loads the image and stores it as 8-bit integers, then MATLAB will use an internal data type called `uint8`. The function `show_img( )` cannot handle this format, but there is a conversion function called `double( )` that will convert the 8-bit integers to double-precision floating-point for use with filtering and processing programs.

   ```
   yy = double(xx);
   ```

   You can convert back to 8-bit values with the function `uint8().`

2. If the image is a color photograph, then it is actually composed of three "image planes" and MATLAB will store it as a 3-D array. For example, the result of `whos` for a 545 × 668 color image would give:

   ```
   Name       Size         Bytes  Class
   xx      545x668x3    1092180  uint8 array
   ```

   In this case, you should use MATLAB's image display functions such as `imshow( )` to see the color image. Or you can convert the color image to gray-scale with the function `rgb2gray( )`. For more information on the image processing functions in MATLAB, try help:

   ```
   help images
   ```

---

[12] Optional mean that you shouldn't include this in a lab report because it wouldn't be counted for a grade. This section is provided in case you are *curious* and want to learn more on your own.

# Lab #5

# ECE-2025

# Spring-2004

# INSTRUCTOR VERIFICATION PAGE

*For each verification, be prepared to explain your answer and respond to other related questions that the lab TA's or professors might ask. Turn this page in at the end of your lab period.*

Name: ____________________________________________ Date of Lab: ___________

Part 3.1(c) Create a 512 × 512 image with white regions in the four corners and in the middle. Write the MATLAB code to make this image and display it.

Verified:______________________ Date/Time:___________

Part 3.3(a) Downsample the `lighthouse` image to see aliasing. Describe the aliasing, where it occurs in the image, and why it occurs.

Verified:______________________ Date/Time:___________