GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2025 Spring 2005
Lab #6: Digital Images: A/D and D/A

Date: 23-Feb. – 1-Mar. 2005

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.

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 you cannot give or receive written material or electronic files. Your 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 2-Mar. to 8-Mar. at the start of your lab.**

---

1 Introduction

The objective in this lab is to introduce digital images as a second useful signal type.

1.1 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 $x(t_1, t_2)$ 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

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

[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.

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

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 512×512 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, and intensity 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.

1.2 Displaying Images

As you will discover, the correct display of an image on a computer monitor can be tricky, especially if the processing performed on the image generates negative values. 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:

1. All image values must be non-negative for the purposes of display. Filtering may introduce negative values, especially when a first-difference 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 created with the `show_img` function[4] 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 a "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 256×3 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.

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 use `show_img.m` to *automatically rescale the image:* This does a linear mapping of the pixel values:[5]

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

   The scaling constants μ and β can be derived from the min and max values of the image, so that all pixel values are recomputed via:

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

[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.

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

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

Below is the `help` on `show_img`; notice that unless the input parameter `figno` is specified, a new figure window will be opened.

```
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
%----
```

2 Pre-Lab

2.1 MATLAB Function to Display Images

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. After loading, use the command `whos` to determine the name of the variable that holds the image and its size.

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.

2.2 Get Test Images

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

(a) Load and display the 428 × 642 "lighthouse" image[6] 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. Notice that the array type for `ww` is `uint8`, so it might be necessary to convert `ww` to double precision floating-point with the MATLAB command `double`. 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?

[6] The image size of 428 × 642 is the horizontal by vertical dimensions. When stored in a MATLAB matrix the `size` command will give the matrix dimensions, i.e., number of rows by number of columns, which is `[642 428]` for the lighthouse image.

3 Warm-up

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

3.1 Debugging

One of the most useful modes of the debugger causes the program to jump into "debug mode" whenever an error occurs. This mode can be invoked by typing:

```
dbstop if error
```

With this mode active, you can snoop around inside a function and examine local variables that probably caused the error. You can also choose this option from the `Breakpoints` menu in the MATLAB editor. It's sort of like an automatic call to 911 when you've gotten into an accident.

When unsure about a command, use `help`.

Download the file `coscos.m` and use the debugger to find the error(s) in the function. Call the function with the test case: `[xn,tn] = coscos(2,3,20,1)`. Use the debugger to:

1. Set a breakpoint to stop execution when an error occurs and jump into "Keyboard" mode,
2. display the contents of important vectors while stopped,
3. determine the size of all vectors by using either the `size()` function or the `whos` command.
4. and, lastly, modify variables while in the "Keyboard" mode of the debugger.

```
function [xx,tt] = coscos( f1, f2, fs, dur )
% COSCOS   multiply two sinusoids
%
t1 = 0:(1/fs):dur;
t2 = 0:(1/f2):dur;
cos1 = cos(2*pi*f1*t1);
cos2 = cos(2*pi*f2*t2);
xx = cos1 .* cos2;
tt = t1;
```

Instructor Verification (separate page)

3.2 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 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]

[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.

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 because f_s is being changed—it's getting smaller. This can be illustrated in a dramatic fashion with the `lighthouse` image.

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, i.e., look for features in the images that are periodic and can be described as having a frequency.

Instructor Verification (separate page)

3.3 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 D-to-A reconstruction method to get the continuous-domain (analog) output image that you see.

Another problem occurs when you try to put two images of different sizes into subplots of the same MATLAB figure. It doesn'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 the following procedure:

1. In MATLAB, use `show_img` and `trusize` to put your images into separate figure windows at the correct pixel resolution.
2. Use a Windows program such as `PAINT` to assemble the different images onto one page. This program can be found under `Accessories`.
3. For each MATLAB figure window, do `ALT-PRINT-SCREEN` which will copy the active window contents to the clipboard.
4. After each "window capture" in step 3, paste the clipboard contents into `PAINT`.[10]
5. Arrange the images so that you can make a comparison for your lab report.
6. Print the assembled images from `PAINT` to a printer.

[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.

[10] 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/windows`.

4 Lab Exercise: Explain Sampling and Aliasing

Suppose that you wanted to illustrate the idea that sampling can cause aliasing. Images provide one way to show aliasing as a visual phenomenon.

4.1 Provide Your Own Image

You must provide the (grayscale) image for this lab report. There are several possible sources:

1. Get an image from the Web.
2. Get an image from a digital camera: determine the kind of scene needed and then photograph it.
3. Scan in a photograph to an image file.

No matter how you create the image, make sure that you work with an image that is ***unique—no other student should be using the same image.*** Furthermore, obtain an image that (more or less) fills your screen, e.g., with dimensions no smaller than 800×600. Crop the image if necessary.

4.2 Downsample the Image

Apply down-sampling by a factor of two to your image. Then compare the original image to the down-sampled version and point out all regions where aliasing has occurred. In your lab report write a justification that uses theory to explain why some regions of the image aliased, and other did not.

In order to have a good example, you will have to choose the original image carefully. The original should have no aliasing, but must have features that will alias when the down-sample by two is applied. More than likely you will have to do a bit of trial and error.

Your lab report can be concise: two images (before and after) along with a carefully written explanation about where you see aliasing and what is causing the aliasing.

4.3 More about Images in MATLAB (Information Only)

This section is included for those students who might want to relate MATLAB's capabilities 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 labs, may not be on your computer.

4.3.1 Color Images

Images obtained from JPEG files that use color are 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( )`.

Lab #6
ECE-2025
Spring-2005
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 Demonstrate your debugging skills with `coscos.m`

Verified:_____ Date/Time:_____

Part 3.2(a) Downsample the `lighthouse` image to see aliasing. Describe the aliasing, point out where it occurs in the image, and explain why it occurs.

Verified:_____ Date/Time:_____