GEORGIA INSTITUTE OF TECHNOLOGY

SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2026 Fall 2013

Lab #6: Digital Images: A/D and D/A

Date: 7-10 October, 2013

Each Lab assignment in ECE2026 consists of three parts: Pre-Lab, In-lab Tasks, and Take-home Questions. It requires you to come into lab prepared. Be sure to read the entire lab carefully before arriving.

Pre-Lab: You should read the Pre-Lab section of the lab and go over all exercises in this section before going to your assigned lab session. Although you do not need to turn in results from the Pre-Lab, doing the exercises therein will help make your in-lab experience more rewarding and go more smoothly with less frustration and panicking.

In-lab Tasks and Verification: There are a number of designated tasks for each student to accomplish during the lab session. Students are encouraged to read, prepare for and even attempt at these tasks beforehand. These tasks 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 put a plastic cup on top of your PC and demonstrate the step to one of the TAs or the professor. (You can also use the plastic cups to indicate if you have a more general question, i.e. you can use it to get our attention even if you don't have an Instructor Verification ready.)

Take-home Questions: At the end of each lab sheet below all the verification steps, several questions are to be answered by the student, who can choose to complete the answers while in the lab or after leaving the lab.

The lab sheet with all verification signatures and answers to the questions needs to be turned in to the Lab-grading TA at the beginning of the next lab session.

---

1. Introduction

One objective in this lab is to introduce digital images as a second useful signal type. A second objective is to study sampling and aliasing with simple signals: sinusoids and chirps. For the second objective, we will use a MATLAB GUI for sampling and aliasing, called **`con2dis`**, which tracks an input sinusoid and its spectrum through A/D and D/A converters. This demo is part of the *SP-First* Toolbox.

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. For example, an RGB color system needs three values at each spatial location: one for red, one for green and one for blue. Video or TV which consists of a sequence of images to show motion 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, which can be represented as a two-dimensional array of numbers of the form

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

where T_1 and T_2 are the sample spacing in the horizontal and vertical direction, respectively. 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

[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.

frame. 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 *pic*ture *el*ement).

An important property of light images such as photographs and TV pictures is that their values are always non-negative and are also finite in magnitude; i.e.,

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

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.

For this lab, two images are provided in the zip file **`Lab06-Images.zip`**, available for download on **`t-square`**.

1.2 Displaying Images


CD-ROM
`show_img.m`

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[2], 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 8-bit, and so the pixel values $x[m, n]$ in the image must be converted to integers in the range $0 \le x[m, n] \le 255 = 2^8 - 1$.
3. The actual display on the monitor created with the **`show_img`** function[3] 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, which we will avoid.
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 to use the full range of pixel value:* We can do this by applying a linear mapping of the pixel values[4]:

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

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:

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

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

---

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

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

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

```
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. (Recall the **wavread** function for reading **.wav** files.) Any file with the extension ***.mat** is in MATLAB's binary format and must 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_img

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. If you have a newer version of MATLAB, you can also use **imshow()** to display an image. Use **help** to find detailed instructions about **imshow()**.

2.2 Get Test Images

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

(a) Load and display the "lighthouse" image[5] from **lighthouse.png** in **Lab06-Images.zip**. 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**, and so it would be necessary to **convert ww to double precision floating-point** with the MATLAB command double if calculations such as filtering are going to be done on **ww**. When you display the image it might be necessary to set the **colormap** via **colormap(gray(256)).** If you use **imshow()**, it can automatically adjust itself to allow display of grayscale images.

(b) Use the colon operator to extract the 439th row of the "lighthouse" image, and make a plot of that row as a 1-D discrete-time signal.

```
ww439 = ww(439,:);
```

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 439th row crosses the fence? Can you match up a

[5]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 (y) by number of columns (x), which is [642 428] for the lighthouse image.

black region between the image and the 1-D plot of the 439th row?

2.3 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 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)[6]. 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[7]. 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**.

One potential problem with down-sampling is that aliasing might occur because f_s is being changed and the image size will be reduced accordingly. This can be illustrated in a dramatic fashion with the **lighthouse** image; see Section 3.3 in the Exercise Section.

2.4 Sampling and Aliasing GUI

A primary objective of this lab is to study sampling and aliasing by using the **con2dis** GUI. If you have installed the *SP-First* Toolbox, you will already have this demo on the **matlabpath**.

In this MATLAB GUI, you can change the frequency of an input signal that is a sinusoid, and you can change the sampling frequency. Then the GUI will show the sampled signal, $x[n]$, its spectrum, and also the reconstructed output signal, $y(t)$ with its spectrum. Figure 1 shows the interface for the **con2dis** GUI. The top row shows plots in the time domain; the bottom row has the corresponding spectrum plots.

In this Pre-Lab, you should perform the following steps with the **con2dis** GUI:

(a) Set the input to $x(t) = \cos(36\pi t + 0.4\pi)$. Determine the Nyquist rate for sampling this signal.

(b) Set the sampling rate to f_s = 24 samples/sec. Notice that this rate is too low to satisfy the Nyquist condition. Thus the output signal is not equal to the input.

(c) Determine the locations of the spectrum lines for the discrete-time signal, $x[n]$, found in the middle panels. Make sure that the **Radian** button is active so that the frequency axis for the discrete-time signal is $\hat{\omega}$.

(d) Determine the complex amplitudes for the spectral lines found in the previous part. Notice that the * on top of the spectral line indicates a line that was originally a negative frequency component in the input signal.

(e) Determine the formula for the output signal, $y(t)$, shown in the rightmost panels. What is the output frequency in Hz?

---

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

[7]Strictly speaking, the down-sampled (factor of 2) image is not a 150 dpi image because the pixel size/resolution remains the same without alteration. A more accurate description in this case is that the image is shrunk to quarter size and the spatial frequency (both horizontal and vertical) in the image is being doubled (i.e., lines become thinner), while the spatial sampling rate remains the same. The potential effect of aliasing is similar, nevertheless, except for the issue of "spatial resolution". See Section 3.4 below.

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

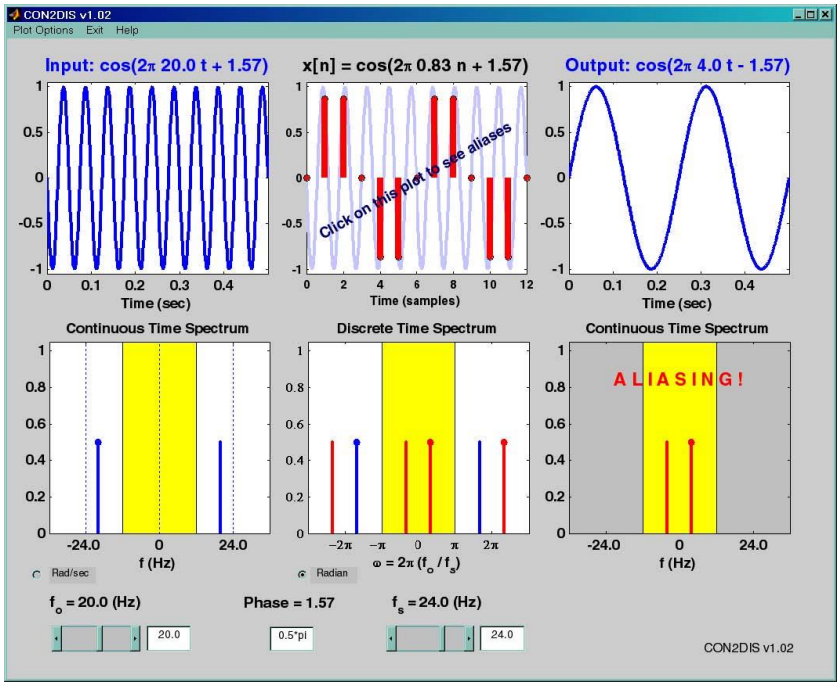

Figure 1: The con2dis MATLAB GUI interface.

3. Lab Exercise

The instructor verification sheet may be found at the end of this lab. ***For this lab, the verification requires that you write down your observations on the Lab sheet when using the GUI. These written observations will be graded.***

3.1 Relationships between the Frequency Domains: ω and $\hat{\omega}$

As you work through the following exercises, keep track of your observations by filling in the worksheet at the end of this assignment. Here are some issues to keep in mind:

- Is the question asking about a continuous-time signal $x(t)$, or a discrete-time signal, $x[n]$?
- The frequency axis (ω) for the spectrum of a continuous-time signal is different from the frequency axis ($\hat{\omega}$) of a discrete-time signal. The range of frequencies is also different: the $\hat{\omega}$-axis for the spectrum of a discrete-time signal has a primary section that goes from $\hat{\omega} = -\pi$ to $\hat{\omega} = \pi$; then the spectrum repeats every 2π.
- Thus the spectrum of a discrete-time signal will have many spectral lines separated by 2π.

Note: read Sections 4-1 and 4-2 in Chapter 4 for more information about the spectra of discrete-time signals, and for information about aliasing.

3.2 Sampling and Aliasing

Use the **`con2dis`** GUI to do the following exercises. The parameters of the input signal are its frequency f_0 in Hz, and its phase ϕ in rads. The sampling rate of the A/D converter and the D/A converter is f_s in samples/sec.

Write your answers in the Lab Sheet. In all cases, write a brief explanation of your answer. "Trial and error" is a very weak justification, so try to write something better than that.

(a) Set the input frequency to f_0 = 14 Hz. Determine the minimum sampling rate f_s so that no aliasing occurs. The units of f_s are samples per second. Justify your answer.

(b) Set the input frequency to f_0 = 14 Hz and the input phase to $\phi = \pi/6$. Determine the locations of the

spectral lines in the spectrum of the discrete-time signal when the sampling rate is f_s = 29 Hz. Do this for the two spectral lines that lie in the range $-\pi \le \hat{\omega} \le \pi$. Explain how you calculated the two values for $\hat{\omega}$.

(c) For the same parameters as the previous part, determine the complex amplitudes for the two spectral lines that lie in the range $-\pi \le \hat{\omega} \le \pi$. Give the complex amplitudes in polar form.

(d) Set the input frequency to f_0 = 18 Hz and the input phase to $\phi = 0.35\pi$. Determine the formula for the output signal when the sampling rate is f_s = 30 Hz. (Note: Be mindful of the sampling rate range implemented in **con2dis** GUI. What if the sampling rate is specified as 36 Hz?)

(e) Set the input frequency to f_0 = 15 Hz and the input phase is $\phi = \pi/3$. Determine the sampling rate f_s so that the output signal is DC, i.e., a constant. Also, determine the value of the constant.

(f) Set the input frequency to f_0 = 12 Hz and the input phase is $\phi = 0.35\pi$. Determine the sampling rate f_s so that the output signal has a frequency of 6 Hz, and a phase of -0.35π.

3.3 Sample an Image and Observe Aliasing

Insufficient spatial sampling (possibly caused by the limited resolution of the CCD sensors) may result in visual artifacts like what you observe in Figure 2 (from **http://support.svi.nl/wiki/AntiAliasing**). The brick structure as well as the roof has a fine texture that is beyond the resolution of the optical sensors, and the resultant image shows the aliasing patterns that are not in the actual scene.

Figure 2: Spatial aliasing resulting in visual artifacts.

Now perform this operation on the **lighthouse.png** image in **Lab06-Images.zip**. 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 (see Pre-Lab 2.3) the **lighthouse** image by a factor of 2, re-display the result, and save the processed image by writing it to an image file using **imwrite()**. Then, answer the following questions:

(a) 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.

(b) Describe how the aliasing appears visually.[9] Which parts of the image show the aliasing effects most

[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.

dramatically? Show these parts to the verifying instructor.

(c) Why is the aliasing happening? Try to think about high spatial frequencies in the image, i.e., look for areas in the image that display alternating patterns in brightness (here for a grayscale image) along a certain direction as if having a frequency. Develop your appreciation and intuition around "frequency" in the spatial instead of temporal sense.

Instructor Verification (separate page)

3.4 Aliasing, Spatial Resolution and Visual Display

In Section 3.3, you down-sample an image by a factor of p (where $p = 2$), which is equivalent to changing the sampling rate by the corresponding factor, and observe spatial aliasing caused by this change. The size of the "pixel" nevertheless stays intact and since there are fewer pixels along both horizontal and vertical axes, the image also shrinks accordingly. The shrinkage sometimes makes viewing comparisons with the original image a bit more difficult because objects in the image become smaller. You can use the **`size`** command to verify the dimensions of the image matrix before and after down-sampling.

Now one can scale up the size of the image by duplicating the pixel values p times along both axes, to return the image to the original size. Figure 3 shows how pixels are being duplicated by a factor of 2 along both dimensions.

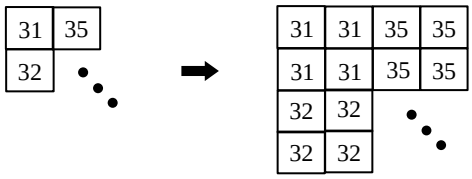

Figure 3 Image matrix expanded by a factor of 2 in row and in column

The following MATLAB code allows you to flexibly expand the image matrix:

```
function Mout = mxpd( A, irup, icup )
% expand matrix
% irup = row expansion factor
% icup = column expansion factor
[m n] = size(A);
vr = ceil((1:m*irup)/irup);
vc = ceil((1:n*icup)/icup);
Mout = A(vr,vc);
end
```

This time we'll use another image in **`Lab06-Images.zip`**, **`silo.png`**. Like in 3.2, read in the image and down-sample it by a factor of p (p = 2, 3, and 4). Then, use the above code to expand your down-sampled image matrices back to the original size (make sure to set the right values for **`irup`** and **`icup`** in the code) and show the three images along with the original in separate figures. (If the original image is a 300 dpi image and $p = 2$, the expanded image from a down-sampled one will be the equivalent of a 150 dpi image. See footnote #7 above.) Demonstrate the images and point out areas of aliasing to the instructor for verification, and describe qualitatively on your lab sheet what you observe.

Instructor Verification (separate page)

3.5 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.

3.5.1 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-2026 Fall-2013
Lab Sheet

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 beginning of your next lab period.

Name: _____ Date of Lab: _____

Part	Observations
3.2(a)	The minimum sampling rate for no aliasing is $f_s >$
3.2(b)	
3.2(c)	
3.2(d)	Output signal: $y(t)$ =
3.2(e)	Sampling rate f_s =
3.2(f)	Sampling rate f_s =

Verified: _____ Date/Time: _____

Part 3.3 Down-sample the downloaded image to see aliasing. Describe the aliasing, point out where it occurs in the image, and try to explain why it occurs – i.e., answer questions (a) and (b). Show the saved images, resulted from various down-sampling factors, to the TA and point to the aliasing areas.

Verified: _____ Date/Time: _____

Part 3.4 Compare the original image and images of lower spatial resolution by a factor of 2 and 4. Obtain verification and describe qualitatively your observations.

Verified: _____ Date/Time: _____

Question:

Question 6.1: A chirp signal $x(t) = \cos(1500\cos(2\pi t) + 3000\pi t + 2)$ is sampled at 8000 samples/s and stored in a MATLAB data array **`xx`** which can then be played back with **`soundsc(xx,8000)`**. Use Matlab to generate such a chirp signal for as long as you want. When you listen to the output, you hear sweeping tones, up and down, with some pauses in-between. This is due to two related reasons: human cannot hear any sound with frequency lower than typically 20Hz, and as such audio devices normally do not bother to reproduce signal components with frequency lower than 20Hz. For this chirp, what is the duration of the pauses between the oscillating tones during which you hear no sound at all? Explain how you arrive at your answer. (Hint: First generate the signal and confirm the existence of pauses in what you hear. Then, derive the instantaneous frequency as a function of time. You can use Matlab to store this function of instantaneous frequency. Then find the duration over which the instantaneous frequency falls below 20Hz.)

Answer: _____ seconds. Explanation: