GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2025 Spring 2011
Lab #6: Digital Images: A/D and D/A

Date: 1–7 March 2011

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.

Verification: 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**. 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 *before leaving the lab.*

It is only necessary to turn in Section 4 as the lab report for this lab. More information on the lab report format can be found on **`t-square`** under the **INFO** link. Please ***label*** the axes of your plots and include a title and Figure number for every plot. In order to reduce *orphan plots*, include each plot as a figure *embedded* within your report. This can be done easily with MATLAB's `notebook` capability. For more information on how to include figures and plots from MATLAB in your report file, consult the **INFO** link on **`t-square`**, or ask your TA for details.

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, but you cannot give or receive any written material or electronic files. In addition, you are not allowed to use or copy material from old lab reports from previous semesters. Your submitted work must be your own original work.
The lab report for this week will be a ***very brief Informal Lab Report.*** The report will be **due during the period 8–14 Mar. at the start of your lab.**

---

1 Introduction

One 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 addition, this lab and the next one will have the objective of learn showing how interpolation is used to create color images for a digital camera. For the interpolation, you will learn how to implement FIR filters in MATLAB, and then use these FIR filters to perform the interpolation of images.

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

[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.

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 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 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 picture element).

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.

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,[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 eight bits, 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]

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

[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.

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:

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

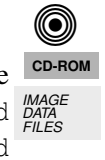
where $\lfloor x \rfloor$ is the floor function, i.e., the greatest integer less than or equal to x.

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



You can load the images needed for this lab from `*.mat` files, or from `*.png` files. Image files with the extension `*.png`, as well as other common formats like JPEG, can be read into MATLAB with the `imread` function. 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.

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[5] 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 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))`.

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

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

[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 by number of columns, which is `[642 428]` for the lighthouse image.

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 440$^{\text{th}}$ row crosses the fence? Can you match up a black region between the image and the 1-D plot of the 440$^{\text{th}}$ 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. Usually this is called *sub-sampling* or *down-sampling*.[7]

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`.

One potential problem with down-sampling is that aliasing might occur because f_s is being changed—it's getting smaller by a factor of `p`. This can be illustrated in a dramatic fashion with the `lighthouse` image; see Section 3.3 in the Warm-up.

2.4 Printing Multiple Images on One Page

The phrase "what you see is what you get" can be elusive when displaying and printing 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 these different size 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`.[8]

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

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

[8] 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`.

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.

2.5 Digital Camera Color Imaging

Digital cameras are now ubiquitous because most cell phones come equipped with cameras pictures with digital cameras. When a digital image is recorded, the camera needs to perform a significant amount of processing in order to provide the user with a viewable image.[9] With a little knowledge of DSP, we can investigate some of that processing.

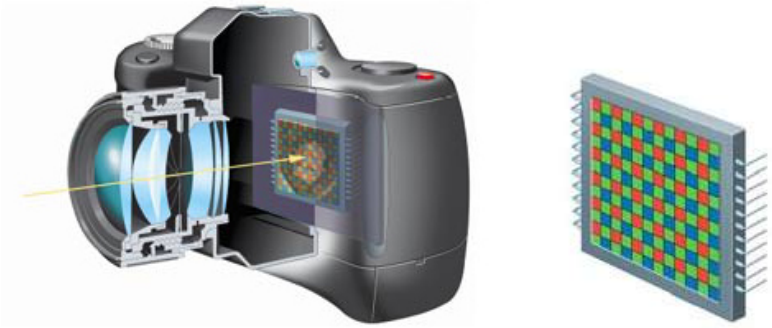


Figure 1: CFA in the digital camera (left), CFA module (right).

A color image requires at least three color values at each pixel location, usually red, green and blue for a computer image. Ideally, a camera would use three separate sensors in order to measure the red, green and blue intensities at every single pixel location. To reduce size and cost, many cameras use a single sensor array in conjunction with a color filter array. The color filter array (CFA) will pass the red, the green or the blue component of light to a given pixel location on the sensor array. This means that the initially captured image matrix does not contain full color information at any pixel location; rather, any one pixel contains only red, or green, or blue intensity information for that pixel. Therefore, the camera must estimate the two missing color values at each pixel, and this estimation process is known as *demosaicking.*

2.5.1 Bayer CFA

Although several possible patterns exist for the CFA, the Bayer pattern is the one that is most commonly used.[10] As shown in Fig. 2, the Bayer CFA measures the green components of the image on a quincunx (checkerboard pattern) grid, while the red and blue components are measured on rectangular grids. The density of green pixels is twice that of either the red or blue pixels.

2.5.2 Image Sensors

Two technologies exist to manufacture imaging sensors: CCD (Charge Coupled Device) and CMOS (Complementary Metal Oxide Semiconductor). Most digital cameras use CCD sensors. The CCD is a collection of tiny light-sensitive diodes that convert photons (light) into electrons (electrical charge). These diodes are called photosites. As their name suggests, photosites are sensitive to light—the brighter the light that is incident on the photosite, the greater the electrical charge that will accumulate at that site. The amount of

[9] Ref: B. K. Gunturk, J. Glotzbach, Y. Altunbasak, R. W. Schafer, R. M. Mersereau, "Demosaicking: Color Filter Array Interpolation in Single-Chip Digital Cameras," Center of Signal and Image Processing, Georgia Institute of Technology, Available at `http://users.ece.gatech.edu/ rmm/fall2003/ece6258/Demosaicking_SPM.pdf`

[10] Note: The Bayer pattern is the exact CFA that should be referred to for the remainder of this lab.

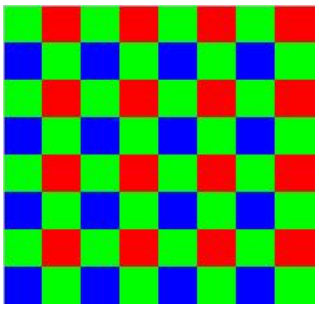


Figure 2: Bayer CFA pattern for an 8×8 image.

electrical charge that accumulated at each photosite (pixel) is read, and an ADC (analog-to-digital converter) turns each pixel's value into a digital value that is recorded. In the block diagram of the imaging system shown in Fig. 3, the 2D array of these digitized intensities is the output of the imaging array.

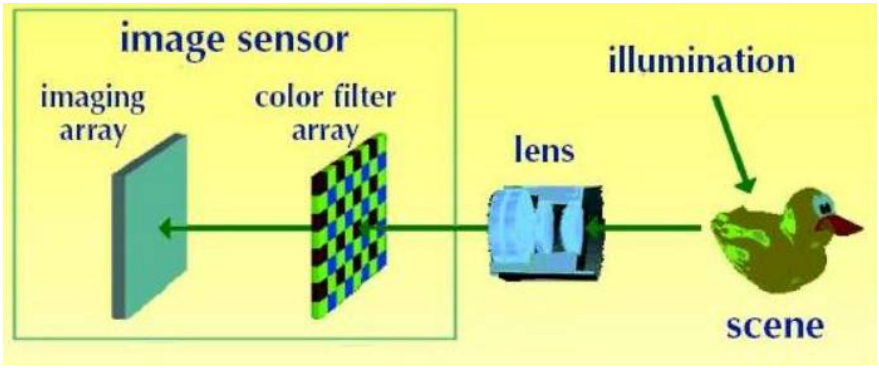


Figure 3: Digital still camera: image capture principle.

To summarize:

- A color image requires at least three color samples at each location. This would require three separate sensors at each location.
- For economic and size reasons, cameras are built with a single sensor in each location and a color filter array (CFA).
- At each pixel location, only one of the three color components is recorded.
- The 2D array of digitized intensities is the starting point for the demosaicking process.
- The DSP algorithm in the camera must estimate the two missing color values at each pixel; this is what will be implemented for the lab project.

2.6 Demosaiking Process

The demosaicking process is illustrated in Fig. 4; the steps in the procedure are as follows:

1. Read in the recorded Bayer CFA array.
2. Then parse the CFA array into three color planes, RGB.
3. Perform different interpolations (with FIR filters) on each of the color planes.
4. Stack the RGB color planes to form a 3D array that can be displayed via `show_img` or `imshow`.

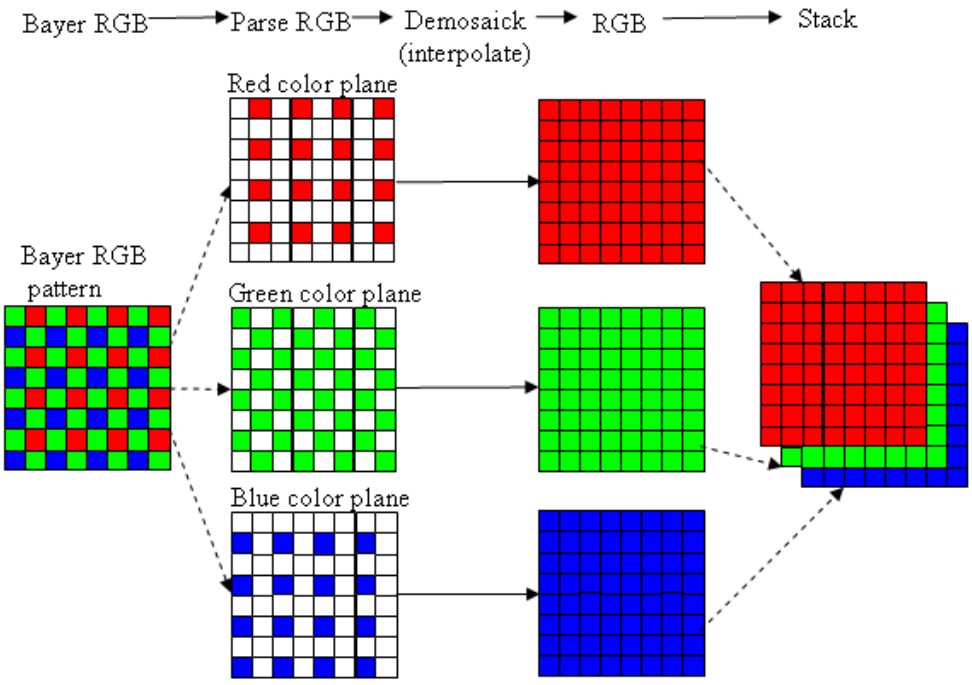


Figure 4: A demosaicking procedure that requires interpolation.

Figure 5 shows the first step of taking the recorded Bayer CFA array (left image) and identifying the RGB color information which is shown in the right image.

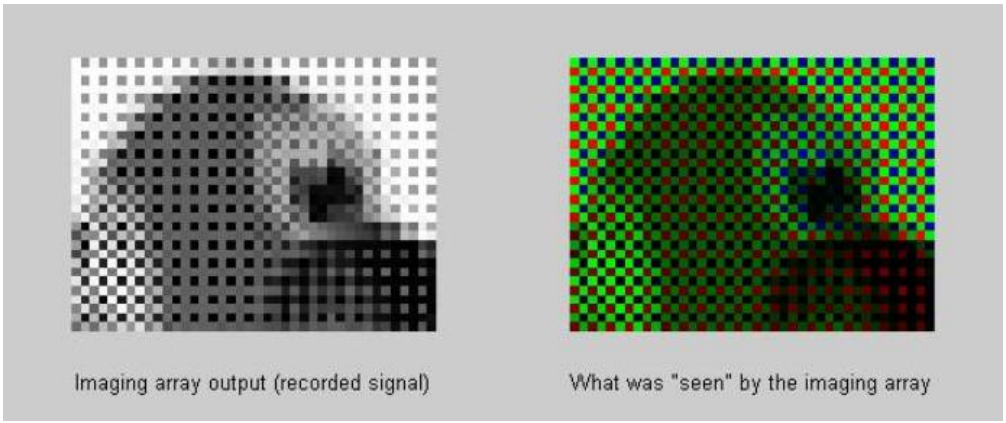


Figure 5: Images at two different stages in the image processing chain: (left) recorded CFA intensities, (right) color image after the RGB pixels have been separated.

3 Warm-up

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. Then you can use the theory of sampling and aliasing to explain how downsampling the cosine formula will provide surprising results.

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

```
xpix = ones(256,1)*cos(2*pi*(0:255)/32);
```

Display the image and explain the gray-scale pattern that you see. Count the number of black stripes across the image. Explain how you can predict that number from the period of the formula for `xpix`?

(b) In the previous part, which data value in `xpix` is represented by white? which one by black? Keep in mind that the cosine has values between ± 1.

Instructor Verification (separate page)

(c) *Optional:* Explain how you would produce an image with bands that are horizontal. Give the formula that would create a 400×400 image with five horizontal black bands separated by white bands. Write the MATLAB code to make this image and display it.

3.2 Aliasing in a Test Image

The banding structure in the test images is controlled by the frequency of the cosine. In other words, we can rewrite the formula for the test image (above) as

```
wd = 2*pi*1/32; xpix = ones(256,1)*cos(wd*(0:255));
```

(a) Generate two test images with different frequencies, one with `wd = 2*pi*2/32` and the other with `wd = 2*pi*14/32`. Call these images `xpix2` and `xpix14`. Display the images, and explain why the image made from the higher frequency cosine has a shorter horizontal period.

(b) Now we apply downsampling by two, i.e., `xpix2(1:2:end,1:2:end)` and `xpix14(1:2:end,1:2:end)`, to both images from the previous part. Use `subplot(2,2,n)` to make a four-panel display: put `xpix2` and `xpix14` in the top row, and put the two down-sampled images in tthe bottom row of the 2×2 subplot. Explain why the two downsampled images look the same.

Instructor Verification (separate page)

3.3 Sample a Photographic Image and Observe Aliasing

Perform this operation on the `lighthouse.png` image.

(a) 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, and display the down-sampled image. What is the size of the down-sampled image?

(b) Notice the aliasing in the down-sampled image, which is surprising since no new values are being created by the down-sampling process. Identify specific parts of the image where you can see the effects of aliasing effects most dramatically. Describe how the aliasing appears visually.[11] Explain why the aliasing is happening by thinking about high frequencies in the image, i.e., look for features in the images that are *quasi-periodic* and can be described as having a frequency.

Instructor Verification (separate page)

[11] 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 Exercise: Exhibit 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) images for this lab report: one for each team member. There are a couple of possible sources:

1. Get a high-resolution image from a digital camera: determine the kind of scene needed and then photograph it, e.g, your lab partner in a striped shirt.
2. Scan in an existing photograph to an image file.

No matter how you create the image, make sure that you work with a high-resolution 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 1024×768. Crop the image if necessary.

4.2 Downsample the Image

Apply down-sampling by a factor of three 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 others 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 three is applied. More than likely you will have to do a bit of trial and error with more than one image.

Your lab report can be concise: two images (before and after) along with a carefully written explanation that points out all regions where you see aliasing, a description of visual phenomenon that you are calling aliasing, and an explanation that connects the cause of the aliasing to the frequency content of the image in that region.

4.2.1 Information About File Formats

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
   ```

Lab #6
ECE-2025 Spring-2011
WORKSHEET & 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: _____

$\Longrightarrow$ Completed Peer Evaluation?
Uploaded ZIP file with .m and .doc files?

Part 3.1(a,b) Generate a test image with bands. Explain how to control the number of bands. Explain grayscale, i.e., which values correspond to black, white, or gray.

Verified:_____ Date/Time:_____

Part 3.2(b) Generate two test images with bands. Explain how downsampling two different images can give the same result.

Verified:_____ Date/Time:_____

Part 3.3(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:_____