GEORGIA INSTITUTE OF TECHNOLOGY

SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2025 Spring 2004
Lab #6: Digital Camera: Image Interpolation

Date: 18 – 24 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 25 Feb. – 2 Mar. at the start of your lab.**

1 Introduction

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

2 Pre-Lab

In the experiments of this lab, you will use `firfilt( )`, or `conv()`, to implement 1-D filters and `conv2()` to implement two-dimensional (2-D) filters. The 2-D filtering operation actually consists of 1-D filters applied to all the rows of the image and then all the columns.

2.1 Two GUIs

This lab also involves the use of two MATLAB GUIs: one for sampling and aliasing and one for convolution.

1. **`con2dis:`** GUI for sampling and aliasing. An input sinusoid and its spectrum is tracked through A/D and D/A converters.
2. **`dconvdemo:`** GUI for discrete-time convolution. This is exactly the same as the MATLAB functions `conv()` and `firfilt()` used to implement FIR filters.

Both of these demos are part of the *SP-First Toolbox*, which can be downloaded from WebCT for ECE-2025 via a link on the "Lab Assignments" page; this link points to the most recent version of the toolbox.

2.2 Overview of Filtering

For this lab, we will define an FIR *filter* as a discrete-time system that converts an input signal $x[n]$ into an output signal $y[n]$ by means of the weighted summation:

$$y[n] = \sum_{k=0}^{M} b_k\, x[n-k] \tag{1}$$

Equation (1) gives a rule for computing the n^{th} value of the output sequence from certain values of the input sequence. The filter coefficients $\{b_k\}$ are constants that define the filter's behavior. As an example, consider the system for which the output values are given by

$$\begin{aligned} y[n] &= \tfrac{1}{3}x[n] + \tfrac{1}{3}x[n-1] + \tfrac{1}{3}x[n-2] \\ &= \tfrac{1}{3}\{x[n] + x[n-1] + x[n-2]\} \end{aligned} \tag{2}$$

This equation states that the n^{th} value of the output sequence is the average of the n^{th} value of the input sequence $x[n]$ and the two preceding values, $x[n-1]$ and $x[n-2]$. For this example the b_k's are $b_0 = \frac{1}{3}$, $b_1 = \frac{1}{3}$, and $b_2 = \frac{1}{3}$.

CD-ROM
`firfilt.m`

MATLAB has built-in functions, `conv( )` and `filter( )`, for implementing the operation in (1), but we have also supplied another M-file `firfilt( )` for the special case of FIR filtering. The function `filter` implements a wider class of filters than just the FIR case. Technically speaking, the `conv` and `firfilt` functions both implement the operation called *convolution*. The following MATLAB statements implement the three-point averaging system of (2):

```
nn = 0:99;                %<--Time indices
xx = cos( 0.08*pi*nn );   %<--Input signal
bb = [1/3 1/3 1/3];       %<--Filter coefficients
yy = firfilt(bb, xx);     %<--Compute the output
```

In this case, the input signal `xx` is a vector containing a cosine function. In general, the vector `bb` contains the filter coefficients $\{b_k\}$ needed in (1). These are loaded into the `bb` vector in the following way:

```
bb = [b0, b1, b2, ...  , bM].
```

In MATLAB, all sequences have finite length because they are stored in vectors. If the input signal has, for example, L samples, we would normally only store the L samples in a vector, and would assume that $x[n] = 0$ for n outside the interval of L samples; i.e., we do not have to store any zero samples unless it suits our purposes. If we process a finite-length signal through (1), then the output sequence $y[n]$ will be longer than the input $x[n]$ by M samples. Whenever `firfilt( )` implements (1), we will find that

```
length(yy) = length(xx)+length(bb)-1
```

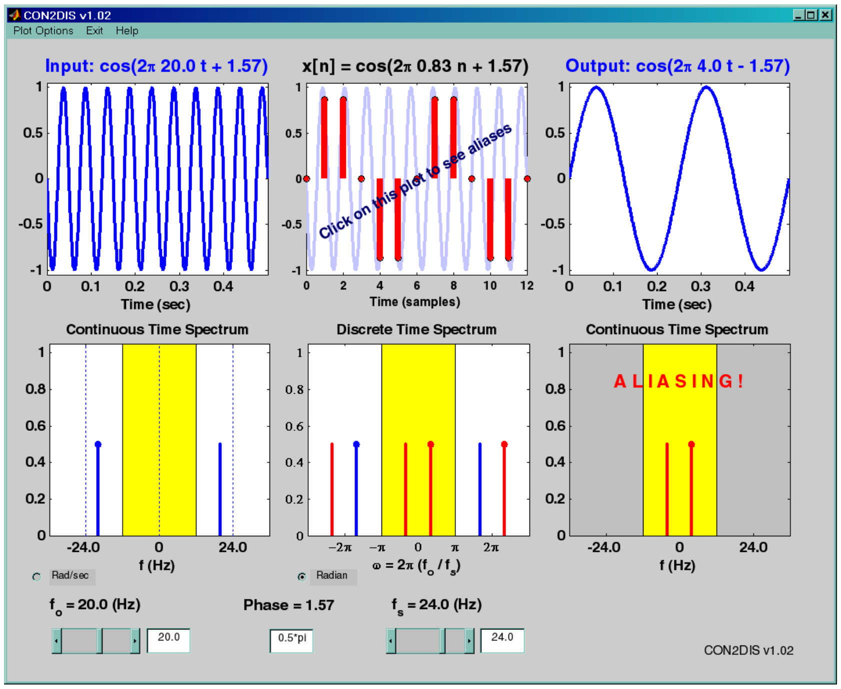
In the experiments of this lab, you will use `firfilt( )` to implement FIR filters and begin to understand how the filter coefficients define a digital filtering algorithm. In addition, this lab will introduce examples to show how a filter reacts to different frequency components in the input.

2.3 Pre-Lab: Run the GUIs

The first objective of this lab is to demonstrate usage of the two GUIs. Thus, you must install the *SP-First Toolbox*, or download the ZIP files for each and install them. Each GUI installs as a directory containing a number of files. You should put the GUI directories on the `matlabpath` so they can be run from the command line.

2.4 Sampling and Aliasing Demo

In this demo, you can change the frequency of an input signal that is a sinusoid, and you can change the sampling frequency. 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. In order to see the entire GUI, you must select **`Show All Plots`** under the **`Plot Options`** menu.



f_o = 20.0 (Hz) 20.0
Phase = 1.57 0.5*pi
f_s = 24.0 (Hz) 24.0

Figure 1: The **`con2dis`** MATLAB GUI interface.

In the pre-Lab, you should perform the following steps with the **`con2dis`** GUI:

(a) Set the input to $x(t) = \cos(40\pi t + 0.5\pi)$

(b) Set the sampling rate to $f_s = 24$ samples/sec.

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

2.5 Discrete-Time Convolution Demo

In this demo, you can select an input signal $x[n]$, as well as the impulse response of the filter $h[n]$. Then the demo shows the ***sliding window*** view of FIR filtering. In this view, one of the signals must be *flipped and shifted* along the axis when convolution is computed. Figure 2 shows the interface for **`dconvdemo`**.

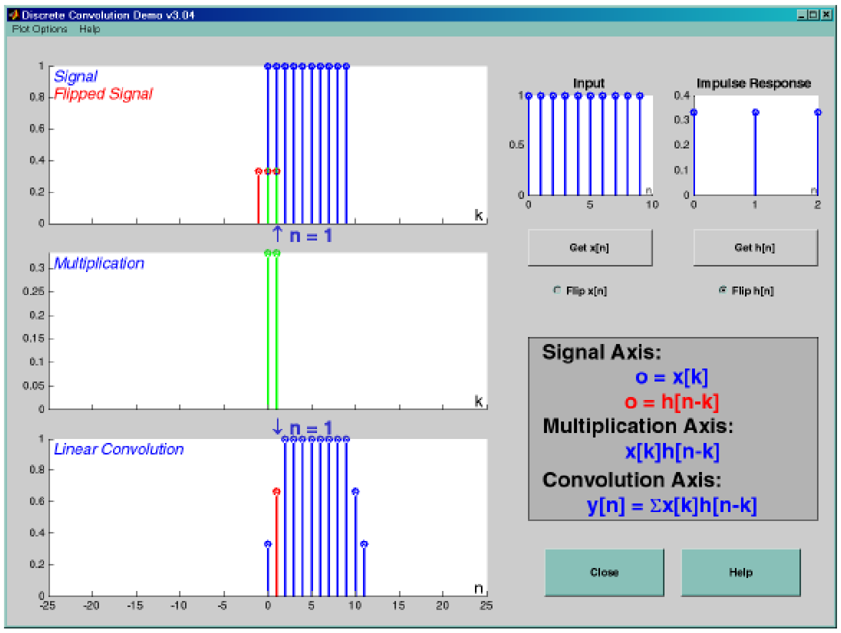


Figure 2: Interface for discrete-time convolution GUI called **dconvdemo**.

In the pre-lab, you should perform the following steps with the **dconvdemo** GUI.

(a) Click on the **Get x[n]** button and set the input to a finite-length pulse: $x[n] = (u[n] - u[n-10])$. Note the length of this pulse.

(b) Set the filter to a three-point averager by using the **Get h[n]** button to create the correct impulse response for the three-point averager. Remember that the impulse response is identical to the b_k's for an FIR filter. Also, the GUI allows you to modify the length and values of the pulse.

(c) Observe that the GUI produces the output signal.

(d) When you move the mouse pointer over the index "n" below the signal plot and do a click-hold, you will get a *hand tool* that allows you to move the "n"-pointer. By moving the pointer horizontally you can observe the sliding window action of convolution. You can even move the index beyond the limits of the window and the plot will scroll over to align with "n."

2.6 Filtering via Convolution

You can perform the same convolution as done by the **dconvdemo** GUI by using the MATLAB function **firfilt**, or **conv**. For ECE-2025, the preferred function is **firfilt**.

(a) For the Pre-Lab, you should do the filtering with a 3-point averager. The filter coefficient vector for the 3-point averager is defined via:

```
bb = 1/3*ones(1,3);
```

Use **firfilt** to process an input signal that is a length-10 pulse:

$$x[n] = \begin{cases} 1 & \text{for } n = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \\ 0 & \text{elsewhere} \end{cases}$$

NOTE: in MATLAB indexing can be confusing. Our mathematical signal definitions start at $n = 0$, but MATLAB starts its indexing at "1". Nevertheless, we can ignore the difference and pretend that MATLAB is indexing from zero, as long as we don't try to write `x[0]` in MATLAB. For this experiment, generate the length-10 pulse and put it inside of a longer vector by using the statement `xx = [ones(1,10),zeros(1,5)].` This produces a vector of length 15, which has 5 extra zero samples appended.

(b) To illustrate the filtering action of the 3-point averager, it is informative to make a plot of the input signal and output signals together. Since $x[n]$ and $y[n]$ are discrete-time signals, a `stem` plot is needed. One way to put the plots together is to use `subplot(2,1,*)` to make a two-panel display:

```
nn = first:last;         %--- use first=1 and last=length(xx)
subplot(2,1,1);
stem(nn-1,xx(nn))
subplot(2,1,2);
stem(nn-1,yy(nn),'filled') %--Make black dots
xlabel('Time Index  (n)')
```

This code assumes that the output from `firfilt` is called `yy`. Try the plot with `first` equal to the beginning index of the input signal, and `last` chosen to be the last index of the input. In other words, the plotting range for both signals will be equal to the length of the input signal, even though the output signal is longer. Notice that using `nn-1` in the call to `stem( )` causes the x-axis to start at zero in the plot.

(c) Explain the filtering action of the 3-point averager by comparing the plots in the previous part. This filter might be called a "smoothing" filter. Note how the transitions in $x[n]$ from zero to one, and from one back to zero, have been "smoothed."

2.7 Digital Camera Color Imaging

Digital cameras are now so popular that more people are choosing to take their pictures with digital cameras than with traditional film 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.[1] This lab investigates some of that processing.

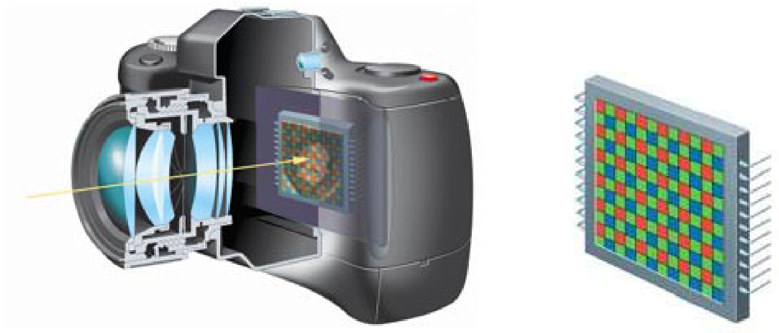


Figure 3: CFA in the digital camera (left), CFA module (right).

A color image requires at least three color values at each pixel location, usually red, green and blue for a computer image. A camera would need 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

[1]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`

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.*

2.7.1 Bayer CFA

Although several patterns exist for the CFA that can be used, the Bayer pattern is the one that is most commonly used.[2] As shown in Fig. 4, the Bayer CFA measures the green components of the image on a

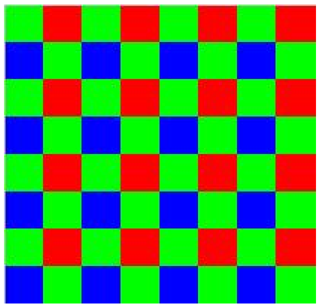


Figure 4: Bayer CFA pattern.

quincunx (checkerboard pattern) grid, while the red and blue components are measured on rectangular grids.

2.7.2 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 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.

Note: The 2D array of these digitized intensities is the starting point for the demosaicking process. In the context of Fig. 5, it is the output of the imaging array.

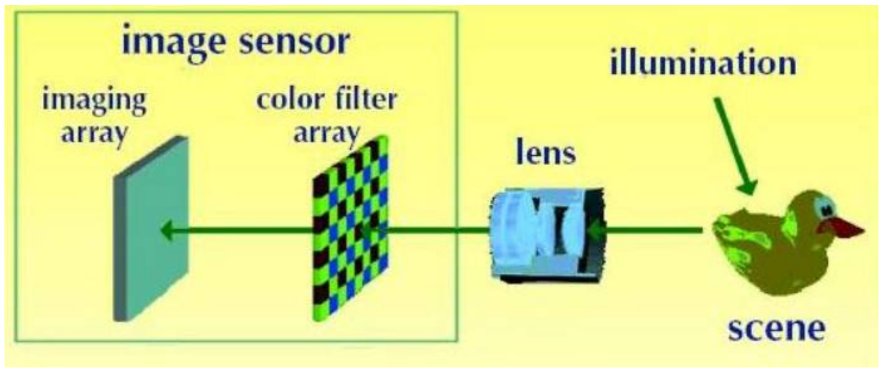


Figure 5: Digital still camera: image capture principle.

A brief summary of the problem statement is as follows:

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

- 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 camera must estimate the two missing color values at each pixel; this is what must be implemented for the lab project in Section 4.

3 Warm-up

3.1 Sampling and Aliasing

Use the `con2dis` GUI to do the following problem:

(a) Input frequency is 12 Hz; input phase is $\phi = -\pi/3$.

(b) Sampling frequency is 14 Hz.

(c) Determine the frequency and phase of the reconstructed output signal

(d) Determine the locations in $\hat{\omega}$ of the lines in the spectrum of the discrete-time signal. Give precise numerical values.

(e) Change the sampling frequency to 10 Hz, and explain the appearance of the output signal.

Instructor Verification (separate page)

3.2 Discrete-Time Convolution

In this section, you will generate filtering techniques similar to those needed in a later section. Use the discrete-time convolution GUI, **`dconvdemo`**, to do the following:[3]

(a) Set the input signal to be $x[n] = \sin(0.2\pi n)(\text{mod}(n, 2))\,(u[n] - u[n-20])$. where $\text{mod}(n, 2)$ means "n-modulo-2", i.e., the remainder when n is divided by 2. MATLAB has a `mod` function. Use the **`User Signal`** signal type to define $x[n]$ within the **`Get x[n]`** window.

(b) Set the impulse response to be $h[n] = \frac{1}{2}\delta[n] + \delta[n-1] + \frac{1}{2}\delta[n-2]$. Once again, choose **`User Signal`** on the menu within **`Get h[n]`**.

(c) Illustrate the output signal $y[n]$ and explain why it is an interpolated version of $x[n]$ for almost all points. Compute the relative time shift between the input and output signals.

3.3 Interpolation via Filtering

It is possible to use FIR filters to perform interpolation, e.g., linear interpolation.

(a) For this warm-up, we need a test signal in which *every other* signal value is zero, e.g.,

$$x[n] = \begin{cases} 0 & \text{for } n = 0, 2, 4, 6, \ldots \\ 3\cos(0.1\pi n) & \text{for } n = 1, 3, 5, 7, \ldots \end{cases}$$

[3]You must install version 3.06 of **`dconvdemo`** in order to have the **`User Signal`** capability; look for the version number displayed in the title bar of the GUI window.

In MATLAB there is a trick that can be used to make a signal with zeros in the even indices

```
nn = 0:(L-1);
xx = ?????;  %<-- use the definition of x[n]
xx = xx .* mod(nn,2); %- zero out every other element
```

For the filtering in the next part, make a signal with 15 nonzero values, i.e., a total length of 30 points.

(b) For this warm-up, you should do the filtering with the following FIR filter:

$$y[n] = \tfrac{1}{2}x[n] + x[n-1] + \tfrac{1}{2}x[n-2]$$

Define the filter coefficient vector `bb` for this FIR filter and then use it in `firfilt` to process `xx` created in part (a). How long are the input and output signals?

When unsure about a command, use `help`.

(c) To illustrate the filtering action of the FIR filter, you must make a plot of the input signal and output signal together using `subplot` as shown in Section 2.6. The plotting range for both signals should be set equal to the length of the input signal, even though the output signal is longer.

(d) Explain the filtering action of the FIR interpolation filter by comparing the plots in part (b). Two features should be noted: the output is shifted with respect to the input; and the output signal has filled in "interpolated" values in between the original input values.

Instructor Verification (separate page)

3.4 Filtering Images: 2-D Convolution

One-dimensional FIR filters, such as running averagers and first-difference filters, can be applied to one-dimensional signals such as speech or music. These same filters can be applied to images if we regard each row (or column) of the image as a one-dimensional signal. For example, the 50th row of an image is the N-point sequence `xx[50,n]` for $1 \le n \le N$, so we can filter this sequence with a 1-D filter using the `conv` or `firfilt` operator.

One objective of this lab is to show how simple 2-D filtering can be accomplished with 1-D row and column filters. It might be tempting to use a `for` loop to write an M-file that would filter all the rows. For a *first-difference filter*, this would create a new image made up of the filtered rows:

$$y_1[m,n] = x[m,n] - x[m,n-1]$$

However, this image $y_1[m,n]$ would only be filtered in the horizontal direction. Filtering the columns would require another `for` loop, and finally you would have the completely filtered image:

$$y_2[m,n] = y_1[m,n] - y_1[m-1,n]$$

In this case, the image $y_2[m,n]$ has been filtered in both directions by a first-difference filter

These filtering operations involve a lot of `conv` calculations, so the process can be slow. Fortunately, MATLAB has a built-in function `conv2( )` that will do this with a single call. It performs a more general filtering operation than row/column filtering, but since it can do these simple 1-D operations it will be very helpful in this lab.

(a) Load in the image `echart.mat` with the `load` command (it will create the variable `echart` whose size is 257×256). We can filter all the rows of the image at once with the `conv2( )` function. To filter the image in the horizontal direction using a first-difference filter, we form a *row* vector of filter coefficients and use the following MATLAB statements:

```
bdiffh = [1, -1];
yy1 = conv2(echart, bdiffh);
```

In other words, the filter coefficients `bdiffh` for the first-difference filter are stored in a *row* vector and will cause `conv2( )` to filter all rows in the *horizontal* direction. Display the input image `echart` and the output image `yy1` on the screen at the same time. Compare the two images and give a qualitative description of what you see.

(b) Now filter the “eye-chart” image `echart` in the *vertical* direction with a first-difference filter to produce the image `yy2`. This is done by calling `yy2 = conv2(echart,bdiffh’)` with a column vector of filter coefficients. Display the image `yy2` on the screen and describe in words how the output image compares to the input.

3.5 Reading and Displaying CFA Images

Each element of the recorded matrix (the image) corresponds only to the intensity of the light that was incident on a particular photosite; it contains no explicit color information. Since we know the exact CFA that was used, we can figure out the image that was ”seen” by the imaging array.

The CFA image acquired by a digital camera can be contained in a single array. Complete the MATLAB code below to load and view the CFA output as separate colors.[4] Refer to Fig. 4 for the arrangement of the red, green and blue pixels. Once the three color planes have been separated, they can be used as the input signals to the demosaicking program of Section 4.

```
% Read in the Recorded image (that was sent through the Bayer CFA)
CFA_out = imread(’peppers_CFA.png’);
figure(1234), subplot(111), imshow(CFA_out), title(’CFA data’), pause

R_image = zeros(size(CFA_out));
G_image = zeros(size(CFA_out));
B_image = zeros(size(CFA_out));

R_image(1:2:end, 2:2:end) = CFA_out(1:2:end, 2:2:end);
imshow(uint8(R_image)),shg, pause %<- show_img would also display the image

G_image(???, ???) = CFA_out(???, ???);  %<---- Fill in code
G_image(???, ???) = CFA_out(???, ???);  %<---- Fill in code
imshow(uint8(G_image)), shg, pause %<- show_img would also display the image

B_image(???, ???) = CFA_out(???, ???);  %<---- Fill in code
imshow(uint8(B_image)),shg, pause %<- show_img would also display the image

Bayer_RGB(:,:,1) = R_image;  %-- load RGB info into 3 color planes
Bayer_RGB(:,:,2) = G_image;
Bayer_RGB(:,:,3) = B_image;
imshow(uint8(Bayer_RGB))     %<--  show_img( ) would also display the image
```

The result of the code above should be a color image as in right hand side of Fig. 6. If you zoom in to a small region of this image you will see the Bayer CFA pattern. If your display is not showing the true-size image, displayed images will most likely show some aliasing, which will manifest in the form of Moiré patterns. When you zoom in on a region of the image you will see the true samples, but if you try to zoom in further, you will most likely see a few more interesting Moiré patterns.

You will also notice that `CFA_out` will be an $M \times N$ array of `uint8` numbers. This means that a pixel will take on integer values that are in the range 0 to 255. Numbers of type `uint8` are not capable of accommodating simple mathematical operations. Therefore, we need to ”cast” the input image to a double

[4]Image display can be done with `imshow` from MATLAB’s Image Processing Toolbox, or with `show_img` in the *SP-First Toolbox* if you install the most recent version (Feb 2004) that works with color images.

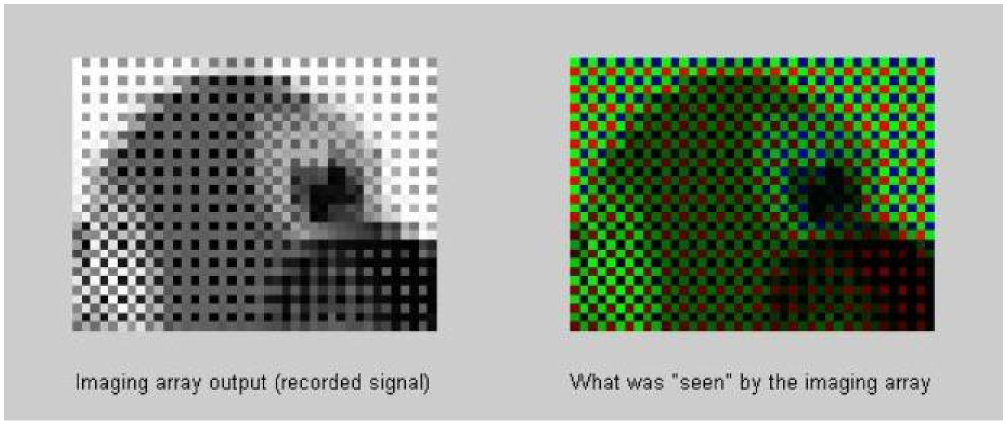

Imaging array output (recorded signal)
What was "seen" by the imaging array

Figure 6: Images at two different stages in the image processing chain: (left) recorded CFA intensities, (right) color image after the RGB pixels have been separated.

array (floating-point) in order to do any signal processing on it. After processing, the final result must be rounded and cast back to `uint8` so that it can be saved in a standard image format and/or compared with the true color image. It is also good practice to round and cast back to `uint8` any time we may want to display/analyze intermediate results.

Instructor Verification (separate page)

4 Lab Exercise: Forming Color Images for a Digital Camera

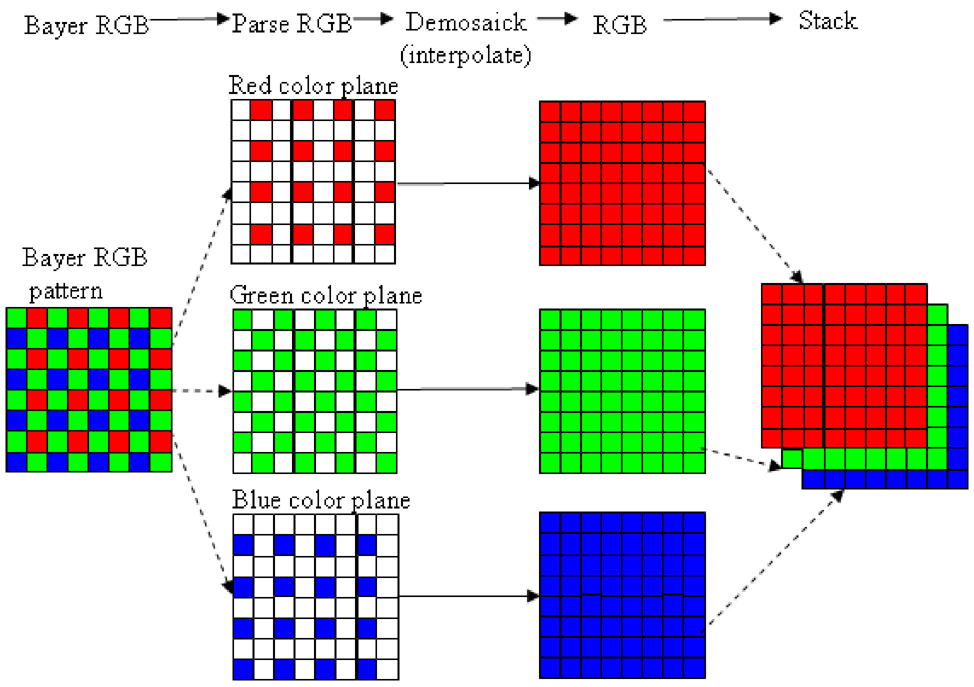


Figure 7: A demosaicking procedure; refer to it as you complete the lab exercises.

The demosaicking process is illustrated in Fig. 7; the steps in the procedure are as follows:

1. Read in the recorded image and parse it into three color planes, RGB.

2. Perform different interpolations on each of the color planes.
3. Stack the RGB color planes to form a 3D array that can be displayed via `show_img` or `imshow`.

First, download an image sensor output image such as `(motorcycles_CFA.png)` and the corresponding "true" color image `(motorcycles.png)`. Note: The "true" color image should only be used to measure the success of your demosaicking implementation, and should not be used during any part of the actual reconstruction effort.

Some useful MATLAB commands for this lab: `imread, imwrite, imshow, round, uint8, double, median.` If you have MATLAB's Image Processing Toolbox, then you can learn more via: `help images`, but the IP Toolbox is not necessary for completing this lab.

4.1 Interpolate the Color Planes

We are now ready to consider the most important part of the demosaicking procedure: interpolation. This is the part of the interpolation where we make our best effort to estimate the two missing color values at each pixel location.

4.1.1 Interpolate the Red channel

First, we will interpolate (estimate) the missing values in the red image using the values of the surrounding captured pixels. 2-D interpolation will be carried out by interpolating the rows (with a 1-D FIR interpolation filter) and then the columns. By inspection of the parsed red image in Fig. 7, it is evident that the odd-indexed columns and even-indexed rows are all zeros.

The 2-D interpolation will fill in the missing red values by first performing linear interpolation along the rows where it is feasible (the odd-indexed rows). Once row-wise interpolation is complete, you will see that the resultant image has every other row populated. It is now possible to use the same concept along the columns of the row-interpolated image.

Linear interpolation along a row or column can be carried out with an FIR filter:

$$y[n] = \tfrac{1}{2}x[n] + x[n-1] + \tfrac{1}{2}x[n-2]$$

which can be computer in MATLAB with `firfilt` (one row/column at a time) or with `conv2` which can do all columns with one call. Since the output of an FIR filter is longer than the input, and the FIR filter introduces a shift, it will be necessary to align the rows (or columns) so that the original nonzero values remain in the location.[5]

Interpolation of the red channel is now complete. If you have not already done so, view the fully interpolated `R_image` just to make sure that everything is as expected.

4.1.2 Interpolate the Blue channel

Looking at the parsed blue color plane in Fig. 7, it is evident that the interpolation procedure that should be carried out is nearly the same as that employed to interpolate the red channel. In fact, we can use the exact same code used to interpolate the red channel, but the alignment of pixels would seem to be slightly different. However, the red-channel function could be re-used to interpolate the blue channel by noticing that we can transpose the parsed blue color plane to get it into the same form as the parsed red color plane.

Give the command(s) used to interpolate the `B_image` using transposes along with the red channel interpolation function. Then view the interpolated `B_image` to check that the result is reasonable.

[5] We would like to implement the non-causal FIR operation: $y[n] = \frac{1}{2}x[n+1] + x[n] + \frac{1}{2}x[n-1]$ to make the interpolation symmetric, but `firfilt` or `conv2` in MATLAB cannot do that, so we implement the causal version and then shift the result.

4.1.3 Interpolate the Green channel

Next, we must interpolate the missing values in the green image which are on a quincunx grid as shown in Fig. 7. In the previous lab a method for quincunx interpolation was implemented. Now, that quincunx interpolation method should be rewritten using the FIR interpolation filter. As a recap, the quincunx interpolation is the average of two interpolated images: a row-interpolated image and a column-interpolated image. As in the red and blue images, care must be taken to align the pixels so that the original nonzero values remain in the same position. View the fully interpolated `G_image` and compare it to the interpolated `R_image` and `B_image` images.

4.2 Stack and View the Channels

Interpolation of all three color planes is now complete. The final color image is constructed by stacking the three color planes into a single 3D array. Use the same procedure used to construct the image seen by the imaging array before. Create a MATLAB figure with four subplots showing each interpolated color plane and the (stacked) interpolated full-color image.

4.2.1 Recap of Demosaicking

The interpolation procedure that we have just completed on each of the color planes is referred to as bilinear interpolation. It is an implementation of bilinear interpolation that is specific to this situation (simpler than the general implementation). This is one of the simplest demosaicking techniques, and can be grouped into the class of demosaicking methods that applies well-known interpolation techniques to each color channel separately. There is another class of demosaicking algorithms that exploit the inter-channel correlation and has significantly better performance; however, they are beyond the scope of this lab.

4.3 Performance Evaluation

At this point in time, it is prudent to obtain a measure of the performance of our demosaicking procedure. This will give us an idea of how well we estimated the actual image, and more importantly, will give us an idea of the extent of opportunity available for us to improve our estimates. It is difficult to assess how well the image interpolation was performed unless a quantitative measure is adopted. Visual comparisons are too subjective. One commonly accepted measure of performance is *normalized root mean-square error* which can be defined as follows:

$$\text{NRMSE} = \frac{\left(\sum_m \sum_n |i[m,n] - r[m,n]|^2\right)^{\frac{1}{2}}}{\left(\sum_m \sum_n |i[m,n]|^2\right)^{\frac{1}{2}}} \tag{3}$$

where $i[m,n]$ is the "true" image and $r[m,n]$ is the reconstructed (or interpolated) image.[6] The value for the NRMSE should be between 0 and 1, so it can be expressed as a percentage.

Write a simple function that will return the NRMSE for each of the three color channels. Your function should take two color images as its input, and return a three-element NRMSE vector as its output. Include the NRMSE percentages in your lab report. Discuss the NRMSE numbers. What information is conveyed by the NRMSE numbers and what information is not? Which color channel(s) have the best NRMSE, and why?

In addition to the NRMSE numbers, provide a visual evaluation of your final result. Look for details in the "motorcycle" image that are not well represented in the interpolated image. Relate your visual observations to the concept of aliasing.

[6] Since the left and right columns and the top and bottom rows contain large interpolation errors, the sums in the NRMSE definition should omit those rows and columns. These errors are caused by the "edge" effects of the FIR filter.

Lab #6
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: Demonstrate that you can run the `con2dis` GUI. Calculate the locations of the spectrum lines for the discrete-time signal. Write the precise values of the $\hat{\omega}$ and the output phase below for both cases: $f_s = 14$ Hz and $f_s = 10$ Hz.

Verified:____________________ Date/Time:__________

Part 3.3 Interpolate an input signal `xx` with an FIR filter by using `firfilt.m`. Display 30 points of the input and output signals. Determine the shift (delay) between the input and output.

Verified:____________________ Date/Time:__________

Part 3.5 Read the CFA intensity image and parse into 3 color planes; display the RGB image.

Verified:____________________ Date/Time:__________