Homework #2: Image Processing in MATLAB

Assigned: Monday, September 19
Due: Friday, September 30

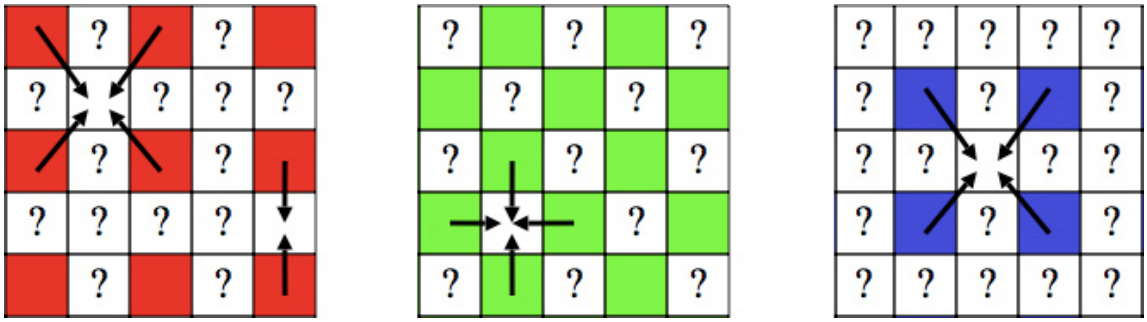
For this assignment you will write several small programs in MATLAB as an introduction to both MATLAB and some simple image processing operations. For each problem, run your code on the provided test images given on the homework web page and, optionally, other images of your own choice. When submitting your homework, please create a folder for each problem. In each folder include a 'main_P#.m' script file (e.g., `main_P1.m` for the first problem) and several function M-files. The naming of any auxilliary functions should conform to the same naming convention. Also, note that it is often a good idea to initially convert integer pixel values after reading in an input image to floating point (using `im2double`) before performing any image operations on it, and then at the end convert the result back to integer values (using `uint8`) before saving. All output images should be saved as `.jpg` files.

1. Histogram Equalization

Histogram equalization is a commonly used image operator for, among other things, enhancing the contrast in an image. To learn about it, read Section 3.1.4 in the Szeliski book and Wikipedia at http://en.wikipedia.org/wiki/Histogram_equalization Next, implement in MATLAB a function that performs histogram equalization by (1) converting an input color image from RGB to HSV color space (using `rgb2hsv`), (2) computing the histogram and cumulative histogram of the V (luminance) image, (3) using the cumulative histogram to create a new luminance image, V′, that has a roughly "flat" histogram, and (4) converting the HSV′ image to RGB color space (using `hsv2rgb`). Information about HSV color space is in Section 2.3.2 in the Szeliski book. The calling form should be `function J = myhisteq(I)` where the function takes a color image array `I` as input, and outputs a new color image array `J` after equalization. Your `main_P1.m` script file should read an input image file, call `myhisteq`, and write the output image as a `.jpg` file. Also create images of the histograms of V and V′ (using `imhist`). Do not use `histeq`. Hand in jpg images of the input, output, and two histograms.

2. Demosaicing

Digital cameras that contain a single image sensor capture a color image by overlaying a color filter array in front of the image sensor's pixels. The color filter array is known as a Bayer pattern that contains red, green and blue filters arranged in 2 x 2 blocks as [R G; G B]. (In order to get the Bayer pattern of an image, you will need a digital camera that can save the image in "Raw" format.) The process of converting a raw image into a full color image consisting of three channels, one for red, green and blue, is called **demosaicing**. Read Section 10.3.1 in Szeliski for a brief description. Implement a simple *linear interpolation* method for demosaicing, defined as follows: for each pixel in each color channel, fill in the missing values by averaging either the four or the two nearest neighbors' values:



The output image should be the same size as the input image but with three bands instead of one. Implement this as `function J = demosaic(I)` where `I` is an input mosaic image and `J` is an RGB output image. Avoid using loops if possible. Instead use `imfilter`. You may *not* use `interp2`. Your `main_P2.m` file should read an image file, call `demosaic`, and write the output file as a `.jpg` image file.

We will provide each test image in both JPEG and Raw formats. The Raw image is stored as a `.bmp` or `.pgm` format image file so that you can easily read its original Bayer Pattern data using `imread`.

To evaluate the result image, compute the squared difference between the original and reconstructed color values for each pixel in each color channel separately, and then add the three color components together to obtain an output grayscale image. Display it using `imshow` and scale the output values so that the maximum value is 255 (i.e., white). Create a second image that shows a portion where artifacts of demosaicing are visible, and give a brief explanation (in a `README.txt` file) of the likely cause of this artifact. Hand in the result image, difference image, and artifact image as .jpg files.

OPTIONAL: Bill Freeman proposed an improvement to the simple bilinear interpolation approach. Since the G channel is sampled at a higher rate than the R and B channels, one would expect interpolation to work better for G values. Then it would make sense to use the interpolated G channel to modify the interpolated R and B channels. The improved algorithm begins with linear interpolation applied separately to each channel, just as you have already done above. The estimated G channel is not changed, but R and B channels are modified as follows. First, compute the difference images R-G and B-G between the respective interpolated channels. Mosaicing artifacts tend to show up as small "splotches" in these images. To eliminate the "splotches", apply *median filtering* (use the `medfilt2` command in MATLAB) to the R-G and B-G images. Finally, create the modified R and B channels by adding the G channel to the respective difference images. Implement the demosaicing part of this algorithm using `function J = FreemanDemosaic(I)`, where `I` is the Bayer Pattern image, `J` is an RGB image.

ATTENTION: Your demosaicing function should accept images with different pixel value scales. The output image should either be a [0,1] `double` image or a [0,255] `uint8` image.

3. Color Transfer

Color correction is a common image processing operation. One form of this is to modify the colors of one image based on the colors in a second image. This “color transfer” process is described in the paper “Color Transfer between Images” by E. Reinhard *et al*., which is available in the course Readings. Implement the basic algorithm described there (i.e., just use the mean and standard deviation of all pixels in the image, and don’t do gamma

correction). Useful MATLAB functions include `mean2`, `std2`, `makecform`, and `applycform`. Write a `function K = mycrtransfer(I, J)` to implement the algorithm, where `I` is the RGB input source image, J is the reference (target) input image, and `K` is the output RGB image. `K` should either be a [0,1] `double` image or a [0,255] `uint8` image. Write `main_P3.m` to read the two input image files, call `mycrtranster`, and write the output image file in `.jpg` format. Hand in the three images as .jpg files.