Homework #2: Image Processing in MATLAB

Assigned: Friday, September 16
Due: Tuesday, September 27

For this assignment you are to write several small programs in MATLAB as an introduction to both MATLAB and image processing. For each problem, run your code on the provided test images given on the homework web page and, optionally, other images of your choice. When submitting your homework, *create a folder for* ***each*** *problem, calling them P1, P2 and P3*. 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 auxiliary functions should conform to the same naming convention. 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, and then at the end convert the result image back to integer values (using `uint8`) before saving. *All output images should be saved as* `.jpg` *files.* Put all three folders together in a *single* zip file called `<NetID username>-HW2.zip` and submit this *one* file to Moodle.

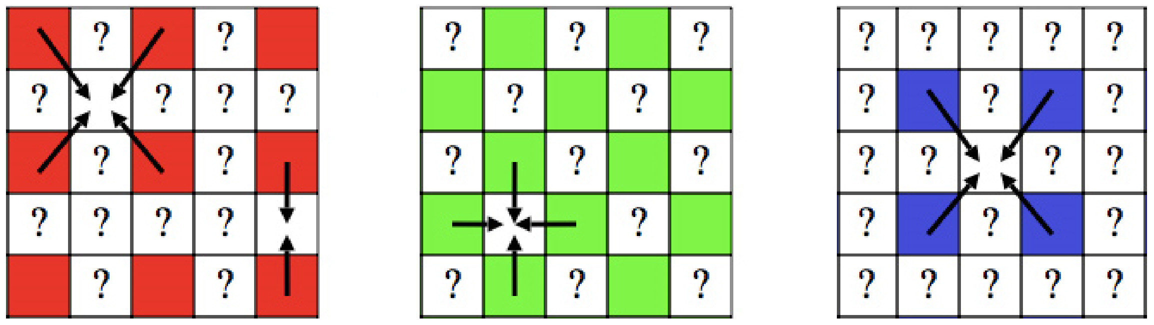
1. Histogram Equalization

Histogram equalization is a commonly used image operator for 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` which creates a `double` image), (2) computing the histogram and cumulative histogram of the *V* (luminance) image only, (3) transforming the intensity values in *V* to occupy the full range 0..255 in a new image *W* so that the histogram of *W* is roughly "flat," and (4) combining the original H and S channels with the *W* image to create a new color image, which is then converted to an RGB color output image (using `hsv2rgb`). Information about HSV color space is given 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. Careful with image types: `uint8` arrays contain integer values between 0 and 255, whereas `double` arrays contain floating point values between 0 and 1. Converting between types automatically rescales their values.

Your `main_P1.m` script file should read an input image file, call `myhisteq`, and write the output image as a `.jpg` file. So, it should look like: `clear; img = imread('P1-bridge.jpg'); out = myhisteq(img); imwrite(out, 'P-bridge-out.jpg')` You can hard-code the names of each input image and output image file pair you use. Also create images of the histograms of *V* and *W* (using `imhist` applied to *V* and *W,* not the histogram arrays you generate). Do *not* use `histeq` or `hist`. You *may* use `imhist` and `cumsum`. Turn in `jpg` images for each output image, and its two histograms. Step 3 should be implemented as $W(i,j) = \max(0, ((256/\text{total number of pixels in } V) * c(V(i,j))) - 1)$ where cumulative histogram *c* is computed by $c(z) = \sum_{j=0..z} h(j)$, and histogram *h* is computed by $h(z)$ = number of pixels in *V* with intensity *z,* where *z* is an integer in the range 0 to 255. Note: To compute *c* and *h, V must* contain discrete, integer values in the range 0… 255; use the function `uint8` to convert *V*. To do step 4, you can simply do: `img(:,:,3) = W; J = hsv2rgb(img)` Submit your output RGB images and histogram images, named `P1-bridge-out.jpg`, `P1-snow-out.jpg`, `P1-bridge-Vhist.jpg, P1-bridge-Whist.jpg, P1-snow-Vhist.jpg` and `P1-snow-Whist.jpg`

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] starting from the upper-left corner of the image. So, the upper-left corner pixel in an image at coordinates (1,1) is assumed to be a red pixel. 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 channels instead of one. Implement this as `function J = mydemosaic(I)` where `I` is an input mosaic image (input as a .bmp image file) and `J` is an RGB output image. Avoid using loops if possible. Instead use `imfilter`. (You may find it easiest to use several "filters," one, for example, for averaging the two horizontally-adjacent pixels in the G channel image at each missing pixel's coordinates, and another one for averaging the two vertically-adjacent pixels in the G channel image at each missing pixel's coordinates; then combine the two results.) You may *not* use `interp2 or demosaic`. Your `main_P2.m` file should read an image file, call `mydemosaic`, and write the output file as a `.jpg` image file, preferably without any user interaction by hardcoding the file names into `main_P2.m` For example, do something like: `img1 = imread('P2-union-raw.bmp'); J1 = mydemosaic(img1); imwrite(J1, 'P2-union-demosaic.jpg', 'jpg');`

We will provide each test image in both JPEG and Raw formats. The Raw image is stored as a `.bmp` format image file (containing a single 2D array) so that you can read the original 2D data using `imread`. The provided JPEG image contains the RGB image produced in the camera by the vendor's demosaicing software so that you can compare your results with theirs.

To evaluate each result image, create an "error" image by computing at *each* pixel the squared difference between the provided `.jpg` image and the demosaiced image you produced for *each* color channel separately, and then adding the three numbers together to obtain a value for that pixel in the "error" image (stored as a grayscale, not color image). Display it using `imshow` and scale the output values so that the maximum value is 255 (i.e., white). Most of this image will be black meaning there is little or no error at those pixels. Find a region in *one of your test images* where artifacts of demosaicing are visible, and crop that region out manually into a new "artifact" image. Save that small image and give a brief explanation (in a `README.txt` file) of the likely cause of this artifact. Hand in the result images in `.jpg` files called `P2-crayons-demosaic.jpg and P2-union-`

demosaic.jpg, error images called P2-crayons-error.jpg and P2-union-error.jpg, and *one* artifact image called P2- artifact.jpg (corresponding to a region in *either one* of the two test images).

Note: 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.

EXTRA CREDIT: 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.

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 (focus on the section “Statistics and color correction”). 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 and std2. Convert images between RGB and L*a*b* color spaces using the MATLAB functions rgb2lab and lab2rgb. Write a function K = mycolortransfer(I, J) to implement the algorithm, where I is the RGB input source image (i.e., the one you want to change), J is the palette (target) input image (i.e., the image you want to steal the colors from), 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 two input image files, call mycolortransfer, and write the output image as an RGB .jpg file. To compute the ratio of standard deviations, use something like: L_out = (std-dev_L_target / std-dev_L_source)(L_source – mean_L_source) + mean_L_target where L_source is the L channel after converting the source input image to Lab color space, std-dev_L_source is the standard deviation of all the values in the 2D matrix L_source, and L_out is the L channel for the output image. Similarly, compute A_out and B_out. Then combine these three into a 3D matrix and convert back to RGB. Run your code on the provided test image pair called P3-source.jpg and P3-target.jpg to create an output image called P3-out.jpg As a second test, *find your own pair of input images*, call them P3-mysource.jpg and P3-mytarget.jpg and create their output image called P3-myout.jpg Hand in these three images as well as the image P3-out.jpg