1st Assignment 776: “Computer Vision”, Fall 2016

Due date: September 7th, 11:59 pm (by e-mail to jmf@cs.unc.edu)
Data: http://www.cs.unc.edu/~jmf/teaching/fall2016/Assignment_1.zip
What you should turn in: A single pdf file with all solutions and a discussion of your solution. Along with the pdf file you should provide a link for your results and code. The pdf file name should be Assignment_1_<your last name>.pdf.

1) Bayer Pattern (adopted from S. Lazebnik)

The goal of the assignment is to get started with image processing in MATLAB or Python by implementing a very simple demosaicing algorithm.

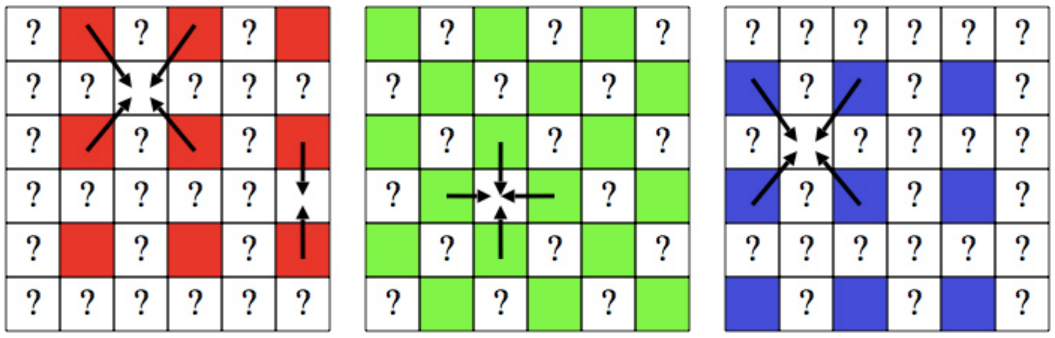
The "mosaic" image (crayons_mosaic.bmp) was created by taking the original color image and keeping only one color component for each pixel, according to the standard Bayer pattern:

R G . . .

G B
.
.
.
So, once you read the "mosaic" image into a matrix, entry (1,1) will be the value of the "red" component for pixel (1,1), entry (1,2) will be "green", etc.

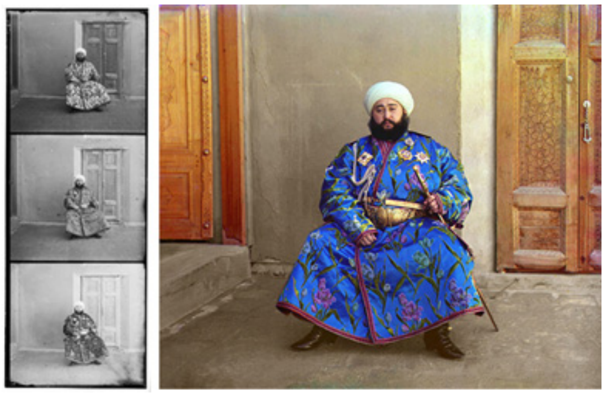
Part 1: Linear Interpolation

Implement a very simple linear interpolation approach for demosaicing: for each pixel, fill in the two missing channels by averaging either the four or the two neighboring known channel values:



The above method, being very simple, does not work perfectly. You can see where it makes mistakes by computing a map of squared differences (summed over the three color components) between the original and reconstructed color value for each pixel. Compute such a map and display it using the imshow or imagesc commands (see the documentation for those commands to figure out how to scale the values for good visibility). In addition, compute the average and maximum per-pixel errors for the image. Finally, show a close-up of a patch of the reconstructed image where the artifacts are particularly apparent and explain the cause of these artifacts.

2) **Image Alignment** (adopted from A. Efros)



Background: Sergei Mikhailovich Prokudin-Gorskii (1863-1944) was a photographer who, between the years 1909-1915, traveled the Russian empire and took thousands of photos of everything he saw. He used an early color technology that involved recording three exposures of every scene onto a glass plate using a red, green, and blue filter. Back then, there was no way to print such photos, and they had to be displayed using a special projector. Prokudin-Gorskii left Russia in 1918. His glass plate negatives survived and were purchased by the Library of Congress in 1948. Today, a digitized version of the Prokudin-Gorskii collection is available online.

Overview: The goal of this assignment is to learn to work with images in MATLAB by taking the digitized Prokudin-Gorskii glass plate images and automatically producing a color image with as few visual artifacts as possible. In order to do this, you will need to extract the three-color channel images, place them on top of each other, and align them so that they form a single RGB color image.

Data: In the data directory you find six input images. Note that the filter order from top to bottom is BGR, not RGB!

Detailed instructions: Your program should divide the image into three equal parts (channels) and align two of the channels to the third (you should try different orders of aligning the channels and figure out which one works the best). For each input image, you will need to display the colorized output and report the (x,y) displacement vector that was used to align the channels.
The easiest way to align the parts is to exhaustively search over a window of possible displacements (say [-15,15] pixels), score each one using some image matching metric like SSD or NCC from class, and take the displacement with the best score. Note that in our case, the images to be matched do not actually have the same

brightness values (they are different color channels), so normalized cross-correlation (NCC normxcorr2) is expected to work better.

For Bonus Points

Multiscale alignment. The data_hires directory contains several high-resolution glass plate scans. For these images, exhaustive search over all possible displacements will become prohibitively expensive. To deal with this case, implement a faster search procedure such as an image pyramid. An image pyramid represents the image at multiple scales (usually scaled by a factor of 2) and the processing is done sequentially starting from the coarsest scale (smallest image) and going down the pyramid, updating your estimate as you go. It is very easy to implement by adding recursive calls to your original single-scale implementation. Alternatively, if you have other ideas for speeding up alignment of high-resolution images, feel free to implement and test those.
Other improvements. Implement and test any additional ideas you may have for improving the quality of the colorized images. For example, the borders of the photograph will have strange colors since the three channels won't exactly align. See if you can devise an automatic way of cropping the border to get rid of the bad stuff. One possible idea is that the information in the good parts of the image generally agrees across the color channels, whereas at borders it does not.