Due on Wednesday, February 21st, 2018 at 11:59 pm
10% grade reduction per late day submission
Answer all problems following the instructions. Please do NOT email the instructors your solutions, but please submit the written portion typset with LaTeX or Microsoft Word with Math Editor. Scanned work will be accepted, but must be of high-quality. If we can't read your work, you get no credit. Please include the written and programming portions in a zipped folder titled "Lastname_firstname_hw0.zip"

NOTE: Since you will be dealing with images with large resolutions, I recommend always working on smaller resolution (just simple subsampling using row/column skipping) to prototype algorithms. Only run the full resolutions when you are sure that your algorithm is up and running.

Problem 1. *Demosaicing*
In this problem, we are going to implement different versions of demosaicing from RAW file (for a Nikon camera, this is a .nef file).

1. Install the Python package **rawpy**, instructions here: `https://pypi.python.org/pypi/rawpy`.
2. Read in the file "tetons.nef" using the rawpy package. Using the package and opencv, figure out how to save the original image as "tetons_original.png". Next, implement subsampling and extract the red,green, and blue channels, and save a lower resolution image "tetons_subsample.png".
3. Implement nearest neighbor demosaicing (to get back to the original resolution). Save this image as "tetons_nn.png".
4. Implement bilinear interpolation demosaicing. Save this image as "tetons_bl.png".
5. Read the paper by Gunturk et al., *Demosaicing: Color Filter Interpolation*, IEEE Signal Processing Magazine, 2005, `http://www.ece.lsu.edu/ipl/papers/IEEE_SPM2005.pdf`. Implement the algorithm of edge-based demosaicing for the green channel in Figure 4. For the R/B channels, feel free to just use bilinear interpolation as before. Save this image as "tetons_dm.png".
6. Create a side by side comparison of the three methods (NN, bilinear, and Gunturk et al.). Also include a zoom in of 2-3 regions (30x30 pixel windows or so) to illustrate the advantages of the methods closeup.

Problem 2. *HDR Imaging*
The point of this problem is to implement the basic algorithm for high dynamic range imaging. In the folder is an exposure stack of .nef files. Each filename is an exposure taken at the time $t_k = \frac{1}{2048}2^{k-1}$ where k varies from 1 to 16.

1. First we need to process the RAW files to do color balancing, and all ISP operations **except** gamma compression and auto-brightness. The relevant RawPy command is

```
rgb = raw.postprocess(gamma=(1,1),no_auto_bright=True,output_bps=16)
```

   Output/save the processed images as "processed_exposurek.tiff" where $k = 1, \cdots, 16$. For the rest of the problem, you will just work on these processed images.

2. Calculate the HDR image using the following formula:

$$I_{HDR}[i,j,c] = \frac{\sum_k w(I^k_{\text{LDR}}[i,j,c]) I^k_{\text{LDR}}[i,j,c]/t^k}{\sum_k w(I^k_{\text{LDR}}[i,j,c])} \tag{1}$$

   where the weighting between exposure frames is:

$$w(z) = e^{-4\frac{(z-0.5)^2}{0.5^2}}, \text{ for Zmin} \leq \text{z} \leq \text{Zmax}.$$

   NOTE: This formula assumes all pixels are between 0 and 1.

3. To display the image, use the tonemapping $I = I_{\text{HDR}}/(1 + I_{\text{HDR}})$.. Display and save this image as "HDR_phototonemap.png".

4. Use a built-in tone-mapping from OpenCV. Be careful to cast your images into float32 format for some of the built-in mappings. Play around with the parameters until you find a tonemapping you like. Display and save this image.

Problem 3. *Bilateral Filter*
The point of this problem is to implement and play with the bilateral filter.

1. Read in the elephant image in the directory. Using Gaussian blur in OpenCV, blur the image.

2. Implement the bilateral filter as described in the lecture slides. Display and save three images side by side: original image, Gaussian blurred image, and bilateral filtered image. You might have to do some parameter searching to find a good bilateral filter. Make sure you show some blow-ups/close details (small pixel windows) to show the effect of the bilateral filter in preserving edges while smoothing noise.

1 Handing in the Assignment

Create a written report, where you print out your code, embed the output images you generate after every *cv2.imwrite* for what you got, and answer any bolded question in the document. Feel free to ask me in class or office hours how to present your code in a legible and straightforward way.

Grading: Your written report will be graded 80% on correctness, and 20% on presentation (lots of comments, easy to read code, written report quality). The easier you make my life to grade, the easier for you! Submit your report, as well as your python files and output images in a zipped folder through Blackboard.