https://piazza.com/class/j6vr5jgnh9d366?cid=55

HW1: Histograms, Filters, Deconvolution, Blending

The goal in this assignment is to get you acquainted with filtering in the spatial domain as well as in the frequency domain.

Laplacian Blending using Image Pyramids is a very good intro to working and thinking in frequencies, and Deconvolution is a neat trick.

You tasks for this assignment are:

* Perform Histogram Equalization on the given input image.
* Perform Low-Pass, High-Pass and Deconvolution on the given input image.
* Perform Laplacian Blending on the two input images (blend them together).

Histogram Equalization

Refer to the readings on [@43](https://piazza.com/class/j6vr5jgnh9d366?cid=43), particularly to Szeliski's section 3.4.1, and within it to eqn 3.9.

Getting the histogram of a grayscale image is incredibly easy (Python):

h = np.histogram(im, 256)

# or

h = cv2.calcHist(...)

Your image is color, so split it to it's channels with cv2.split(), and work on each channel.

A histogram is a vector of numbers. If you wish to visualize it use either

pyplot.hist(im, bins=256) #this will calculate the histogram and visualize it

# or if you pre calculated your histogram:

pyplot.bars(...) # https://stackoverflow.com/questions/5328556/histogram-matplotlib

Once you have the histogram, you can get the cumulative distribution function (CDF) from it

cdf = np.cumsum(h)

Then all you have left is to find the mapping from each value [0,255] to its adjusted value (just using the CDF basically).

**Validate** your results vs. OpenCV's equalizeHist() function.

Do not use cv2.equalizeHist() directly to solve the exercise!

We will expect to see in your code that you get the PDF and CDF, and that you manipulate the pixels directly (avoid a for loop, though).

Low-Pass, High-Pass and Deconvolution in the Frequency Domain

This is an easy one, just follow the tutorials and you should be fine:

<http://docs.opencv.org/master/de/dbc/tutorial_py_fourier_transform.html>

<http://docs.opencv.org/2.4/doc/tutorials/core/discrete_fourier_transform/discrete_fourier_transform.html>

For your LPF, mask a **20x20** window of the center of the FT image (the low frequencies).

For the HPF - just reverse the mask.

For deconvolution, all you are required to do is apply a gaussian kernel (gk) to your input image (in the FD/FFT):

gk = cv2.getGaussianKernel(21,5)

gk = gk \* gk.T

def ft(im, newsize=None):

dft = np.fft.fft2(np.float32(im),newsize)

return np.fft.fftshift(dft)

def ift(shift):

f\_ishift = np.fft.ifftshift(shift)

img\_back = np.fft.ifft2(f\_ishift)

return np.abs(img\_back)

imf = ft(im, (im.shape[1],im.shape[1])) # make sure sizes match

gkf = ft(gk, (im.shape[1],im.shape[1])) # so we can multiple easily

imconvf = imf \* gkf

# now for example we can reconstruct the blurred image from its FT

blurred = ift(imconvf)

Using these simple helper functions I provided you can probably now see how to go the other way around, given an already convolved image, to do deconvolution (use division instead of multiplication).

Note: please use the following image instead of `blurred2.png`: [blurred2.exr](https://d1b10bmlvqabco.cloudfront.net/attach/j6vr5jgnh9d366/iy330hg1n7z1s9/j7pttu0i547v/blurred2.exr)

To load it (as float32 single-channel):

a = cv2.imread("blurred2.exr", cv2.IMREAD\_ANYCOLOR | cv2.IMREAD\_ANYDEPTH)

Laplacian Pyramid Blending

This tutorial will tell you everything you need to know about LPB:

<http://docs.opencv.org/3.2.0/dc/dff/tutorial_py_pyramids.html>

It translates very literally to C++.

Make sure you make images rectangular and equal size:

# make images rectangular

A = A[:,:A.shape[0]]

B = B[:A.shape[0],:A.shape[0]]