Problem Set 2

Assigned: Feb 23, 2006
Due: March 9, 2006

Problem 1 *Image Histograms*

Load image `mountrainier.jpg` and use function `rgb2gray` to convert this RGB image to grayscale. Display its histogram. Perform histogram equalization. You may use MATLAB functions `imhist` and `histeq`.

Now take image `winterstorm.jpg` and transfer its pixel histogram to image `mountrainier.jpg`.

Please submit the original image, transferred image, and their histograms side by side.

Problem 2 *Steerable Pyramids*

2.1 *Warm up*

Download Simoncelli's steerable pyramid code from `http://www.cns.nyu.edu/~eero/steerpyr/`. Please make sure that you have read the readme file and add the path `../matlabPyrTools` and `../matlabPyrTools/MEX`[1] where `..` is the parent directory of the toolbox. Try it on an image with k=3 (four orientations). Load image `einstein.jpg` and set the three highest frequency bands to zero for the horizontal orientation. Reconstruct the image and observe the effect.

There are two functions you can call from the toolbox, namely `buildSpyr.m` constructed in spatial domain, and `buildSFpyr.m` constructed in frequency domain. The reconstruction of the pyramid obtained by `buildSpyr.m` is not perfect. We shall use `buildSFpyr.m` for the sake of better reconstruction. The following sample code shows pyramid construction, visualization and reconstruction:

[1] This path is not mandatory. If MATLAB reports problems about mex files just do NOT add this path. The MATLAB implementation of the functions should be good enough.

```
[pyr,pind] = buildSFpyr(im,3,3); % 3 levels, 4 orientation bands
showSpyr(pyr,pind);
res = reconSFpyr(pyr,pind);
imshow(res);
```

Please do not directly manipulate `pyr` or `pind`, but use the enclosed function `pyrBand` and `setPyrBand` to load and write each subband.

Display the pyramid of the original image. Also display the new pyramid where all the horizontal sub-bands are set to be zero, and the corresponding reconstructed image.

2.2 *Coring*

The goal is now to perform denoising using coring. Load `einstein.jpg`. Use MATLAB function `randn` or `imnoise` to synthesize AWGN (additive white Gaussian noise) to the original image. We are going to denoise it by applying a non-linearity to the high-frequency bands. That is, for each coefficient we will apply a function f that lowers the low-amplitude coefficients (that are assumed to predominantly correspond to noise) and preserves the high-amplitude coefficients. The cutoff will depend on the amount of noise, which will be an input to your function. Rather than implementing a method from the literature, we want you to experiment with coring functions. Design a non-linear functions that take the noise level as a parameter, and, given a pyramid coefficient, removes low-amplitude coefficients but keeps high-amplitude ones. That is, design a function $f(x;\sigma)$ and apply it to each coefficient x using the variance of the AWGN as σ. Note that this function should be different to the subbands at different levels. You can also decide to take the scale as a parameter. This exercise will be graded in part on your difference with the ground truth.

Use 3 levels and 3 subbands to build steerable pyramid. Compute PSNR (Peak Signal-to-Noise Ratio) using the provided function `PSNR.m` (higher PSNR indicates better denoising results). Please give the PSNR metric and denoised image for $\sigma = 0.05$, 0.10 and 0.15. Compare your algorithm with Wiener filtering, using MATLAB function `wiener2`. Hint: you should be able to outperform `wiener2` using wavelet coring, but this is not required. Display the noise image, your denoising result, and the result from `wiener2` side by side. (Optional) You may also change the number of levels and subbands to see how the performance may change.

2.3 *Multiscale blending – 6.882 only*

Use the provided `orange.jpg` and `apple.jpg` to generate two "new" fruits: *orange-apple*, where the left side is orange and the right side is apple, and *apple-orange*, where the left side is apple and the right side is orange. Load `mask.bmp`. Use Burt and Adelson's method to blend the two images on Laplacian pyramid using the appropriately smoothed and down-sampled mask. You can imagine that a Gaussian pyramid is built for the mask and the mask at the same level as the Laplacian pyramid is used for blending. You may use MATLAB function `imfilter` and `imresize` for filtering and down-sampling. Display the original and "new" fruits side by side. You may use the function `buildLpyr` and `reconLpyr` in the toolbox.

(Extra credit) Create a fun photomontage based on this principle, (e.g. eye in the hand in the original Burt and Adelson paper).

2.4 *Heeger and Bergen (Extra credit)*

Implement the pyramid texture synthesis by Heeger and Bergen [1995]. Write a MATLAB function to synthesize a new texture by matching the pixel and pyramid coefficient histograms on steerable pyramid for a given input texture. You can try the texture images in `texture.zip` (not necessarily all of them). Because they are grayscale textures you do not need to care about color. Show the input and synthesized images side by side. Show also the histograms from the input and synthesized. Do you find it easy to match histograms? Does the histogram matching depend on the input texture? Feel free to change the number of pyramid levels, number of subbands, and number of bins to see how these parameters may effect the synthesis. You only need to hand in the best synthesized results you obtain.

Problem 3 *Matting*

Given an input image, we want to extract a foreground element and the corresponding alpha matte. We are given a trimap that coarsely classifies pixels as foreground, background, and unknown. You are going to implement a natural image matting algorithm, a simplified version of Chuang et al's CVPR'01 paper (Section 3). In a nutshell, you must characterize the color distribution of the foreground and background pixels and solve for the values of α, F_g and B_g in a Bayesian framework.

The algorithm will proceed in two steps: 1. gather the color statistics from the foreground and background pixels from the trimap and model them using Gaussians in RGB space. 2. For each pixel in the unknown region, compute the foreground and background color as well as the alpha. For this, we will solve an MAP (maximum a posteriori) problem based on the above Gaussians.

Load `toy.jpg` and `trimap.png`. In the trimap, stored in `trimap.png`, you can regard intensity value $I > 0.95$ as foreground, $I < 0.05$ as background, and the rest pixels are the ambiguous area where an alpha value α, foreground color F_g and background color B_g need to be estimated per pixel. The color distributions will be extracted from all the pixels in the background and foreground regions (once and for all). Compute the mean and covariance for the foreground (resp. background) pixels.

Please see Chuang's paper for the MAP computation of the three unknowns. You need to iterate between solving α and estimating F_g and B_g.

Set $\sigma_C = 0.01$ at first, and try to vary it when the code is debugged. You may need to solve linear equation in this problem. Use MATLAB notation `x=A\b` to solve linear system $Ax = b$. You may loop around all the pixels. If the algorithm takes 10 iterations, it should not take more than one minute. Display the alpha matte.

Load another image `bookshelf.jpg`. Compose the foreground to this image with the estimated α and F_g's. Does the composition look natural to you? Can you see any artifacts?

Hand in the alpha matte and composite image.

(Extra credit) You can blur the background or foreground in composition to obtain effects of focus and field of depth. Play with blurring kernels and see if they look natural.