6.869 Advances in Computer Vision

Spring 2010

Problem set 2

Due date: Wednesday, March 10, 2010

You need to submit a report with descriptions of what you did. Insert images showing your results as well as Matlab plots that you find useful. You should also include pieces of your Matlab code.

Each part of the problem set is independent.

Each person should submit a separate report (pdf). For the filename use: *familyname.pdf*
Write in the report the names of the people you discussed with.

This problem uses pyramid image processing. Download and install the matlabPyrTools from http://www.cns.nyu.edu/~eero/software.html*. When forming pyramid decompositions for these problems, you may always use the default decomposition filters. For this problem submit your MATLAB code and include a printout.*

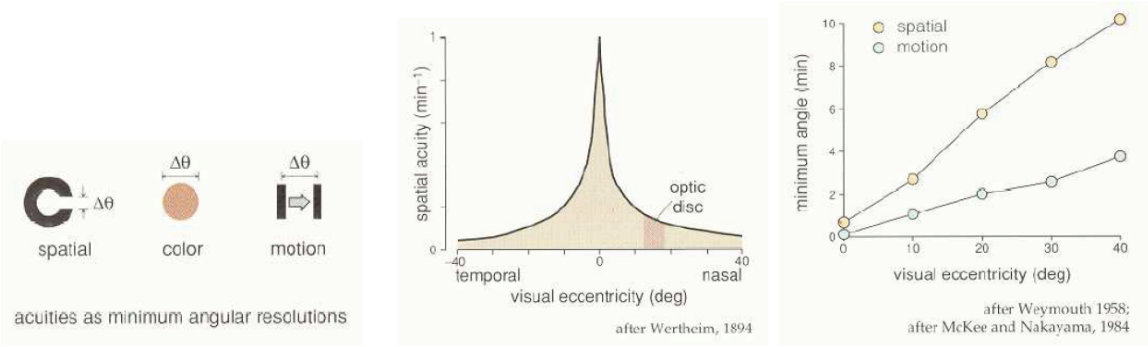
1 – Image blending

1.1- Build a Laplacian pyramid of one image and show you can reconstruct back the original image. Code for the Laplacian pyramid is available in the pyramid image processing toolbox.

1.2 – Blend two images of your favorite pets, friends or objects using the Laplacian pyramid and the method described in class. Include in your report the original images, their Laplacian pyramids, the blending mask, and the resulting blended image.

2 – Image pyramids

Subjectively, our visual world appears to us to be high resolution everywhere. However, we have much higher spatial resolution in the center of our field of view than in the periphery. In this problem, we will synthesize an image approximating our visual resolution as a function of eccentricity.



(a) (b)

Figure 1: (a) Measures of acuity. (b) Plots of eccentricity versus accuity.

Figure 1 shows a plot of the minimum angle resolvable as a function of the visual eccentricity. The visual eccentricity is measured in degrees away from the center of fixation. (From Rodieck, "The First Steps in Seeing", Sinauer, 1998).

Approximate acuity, *a*, in minutes of arc (60 minutes to a degree) as a function of eccentricity, *e*, in degrees, by the expression,

$$a = 0.23e + 0.7 \qquad (1)$$

We will create an image with the effective spacing of the pixels equal to the angular size of the acuity limit. In the figure, that limit is defined as the white space between two ends of a circle. Adjacent black, white, black pixels could approximately represent that circle opening if the pixel spacing were equal to the angular size of the acuity limit.

Assume that the image (or monitor) is square, and that you view it from a distance of two times the length of one side of the image. Where convenient, you may assume angles are small enough so that $\tan(\theta) \sim \theta$.

(a) How many evenly spaced pixels per side does the image need to have in order that the highest resolution part of the image has one pixel per length of finest acuity? CORRECTION: forget this hint: ~~Assume that the highest resolution image point lies at half the maximum acuity, where maximum acuity is as specified by (1).~~ The correct clarification is: acuity is defined at the left of figure 2(a), as the space of that opening in the circle that's visible. So you should equated the acuity value with the spacing between pixels, because a black/white/black pixel sequence could approximately represent that gap in the circle. So one pixel would be equivalent to that gap in the circle that defines the acuity.

(b) Let the upper left corner of the image be (0,0), and the right and bottom edges of the picture be at a distance 1 from this corner. Assume that the upper

left corner is the center of fixation. What effective pixel spacing, as a function of these units, causes the pixel spacing to equal the spatial acuity for the corresponding eccentricity?

(c) We can approximate images of this resolution by using a Gaussian pyramid, which generates images at different numbers of pixel samples, dividing the number of pixels by two at each level of the Gaussian pyramid. Start from an image at the full resolution of part (a). Each pyramid level increases the effective size of its pixels by a factor of two in each dimension. As a function of the coordinate system used in (b), by how many factors of two should the resolution of the original image be reduced as a function of position in the image in order to simulate the human visual acuity, assuming the viewer stares at the upper left corner of the image?

(d) The expression in (c) involves fractional pyramid levels. We can visually approximate images at those intermediate resolution levels by linearly interpolating between our Gaussian pyramid levels. On the class web site is a 2000x2000 image (prob2.jpg), which should be more than enough pixels for you. Crop that image to the desired resolution such that the upper left corner will be at half the maximum visual acuity, when viewed from 2 picture lengths away. Use the Gaussian pyramid to create an image that simulates the fall-off in visual acuity, assume the fixation point is at the upper left corner. At any given pixel, determine the coefficients for interpolating between images by linearly interpolating the corresponding pixel dimensions.

Hint: You will want to use the upBlur *function to transform the Gaussian pyramid levels to all have the same number of pixels. Assume that a pyramid level after* upBlur *has effectively the same number of pixels (in terms of picture content) as the original pyramid band before the* upBlur *operation. That is a reasonable approximation (take 6.341 for the details that we're glossing over here).*

3. Steerable filters

Let $G^0(x,y)$ be some 2-d filter, a function of the cartesian coordinates x and y. Let $G^\theta(x,y)$ be a rotation of $G^0(x,y)$ by q radians about the origin in the counterclockwise direction, i.e.,

$$G^\theta(x,y) = G^\theta(r\cos\Phi, r\sin\Phi) = G^0(r\cos\Phi - \theta, r\sin\Phi - \theta)$$

where $r = \sqrt{x^2 + y^2}$ and $\tan\theta = y/x$. In this problem, you may give your answers in cartesian or polar coordinates, whichever is more convenient.

(a) Suppose $G^0(x,y) = -2xe^{-(x^2+y^2)}$. Find $G^\theta(x,y)$

(b) Show that:

$$G^{\theta}(x,y) = \cos(\theta)G^{0}(x,y) + \sin(\theta)G^{\pi/2}(x,y)$$

and that the output image $F(x,y) = I(x,y) * G^{\theta}(x,y)$ is equal to:

$$\cos(\theta)\left\{I(x,y) * G^{0}(x,y)\right\} + \sin(\theta)\left\{I(x,y) * G^{\pi/2}(x,y)\right\}$$

where * denotes convolution.

(c) Find the direction and magnitude of maximum response at a point (x, y) of the image I(x, y) to the steerable filter $G^{\theta}(x,y)$. The direction of maximum response is the q, such that $F^{\theta}(x,y)$ has the biggest magnitude. Give your answer in terms of the image I(x, y) and the responses $F^{0}(x,y),\ F^{\pi/2}(x,y)$ to the two steerable basis filters $G^{0}(x,y),\ G^{\pi/2}(x,y)$. Note that other steerable filters could require more basis filters than two.

4. *Texture Synthesis*

In this problem you will implement the Efros and Leung algorithm for texture synthesis discussed in Section 9.3 of Forsyth and Ponce. In addition to reading the textbook you may also find it helpful to visit Efros' texture synthesis website: http://www.cs.berkeley.edu/~efros/research/synthesis.html, from which many of the implementation details described below can be found.

As discussed in class, the Efros and Leung algorithm synthesizes a new texture by performing an exhaustive search of a source texture for each synthesized pixel in the target image, in which sum-of-squared differences (SSD) is used to associate similar image patches in the source image with that of the target. The algorithm is initialized by randomly selecting a 3x3 patch from the source texture and placing it in the center of the target texture. The boundaries of this patch are then recursively filled until all pixels in the target image have been considered.

Implement the Efros and Leung algorithm as the following MATLAB function:

synthIm = SynthTexture(sample, w, s)

where sample is the source texture image, w is the width of the search window, and s=[ht wt] specifies the height and width of the target image *synthIm*. As described above, this algorithm will create a new target texture image, initialized with a 3x3 patch from the source image. It will then grow this patch to fill the entire image. As discussed in the textbook, when growing the image un-filled pixels along the boundary of the block of synthesized values are considered at each iteration of the algorithm. A useful technique for recovering the location of these pixels in MATLAB is using *dilation*, a morphological operation that expands image regions (it performs the opposite function of the *erode* operation from the previous problem set). Use MATLAB's imdilate and find routines to recover the

un-filled pixel locations along the boundary of the synthesized block in the target image.

In addition to the above function we ask you to write a subroutine that for a given pixel in the target image, returns a list of possible candidate matches in the source texture along with their corresponding SSD errors. We ask this function to have the following syntax:

[bestMatches, errors] = FindMatches(template, sample, G)

where *bestMatches* is the list of possible candidate matches with corresponding SSD errors specified by errors. *template* is the *w* x *w* image template associated with a pixel of the target image, sample is the source texture image, and *G* is a 2D Gaussian mask discussed below. This routine is called by SynthTexture and a pixel value is randomly selected from bestMatches to synthesize a pixel of the target image. To form bestMatches accept all pixel locations whose SSD error values are less than the minimum SSD value times $(1+\varepsilon)$. To avoid randomly selecting a match with unusually large error, also check that the error of the randomly selected match is below a threshold δ. Efros and Leung use threshold values of $\varepsilon = 0.1$ and $\delta = 0.3$.

Note *template* can have values that have not yet been filled in by the image growing routine. Mask the template image such that these values are not considered when computing SSD. Efros and Leung suggest using the following image mask:

Mask = G .* validMask

where *validMask* is a square mask of width *w* that is 1 where *template* is filled, 0 otherwise and *G* is a 2D zero-mean Gaussian with variance $\sigma = w/6.4$ sampled on a *w* x *w* grid centered about its mean. *G* can be pre-computed using MATLAB's *fspecial* routine. The purpose of the Gaussian is to down weight pixels that are farther from the center of the template. Also, make sure to normalize the mask such that its elements sum to 1.

Test and run your implementation using the grayscale source texture image rings.jpg, with window widths of *w* = 5, 7, 13, s=[100 100] and an initial starting seed of (*x, y*) = (4, 32). Explain the algorithms performance with respect to window size. For a given window size, if you re-run the algorithm with the same starting seed do you get the same result? Why or why not? Is this true for all window sizes?

Please include the synthesized textures that correspond to each window size along with answers to the above questions and a printout of your code in your writeup. Also, submit the files synthtexture-[last name].m *and* findmatch es-[last name].m.