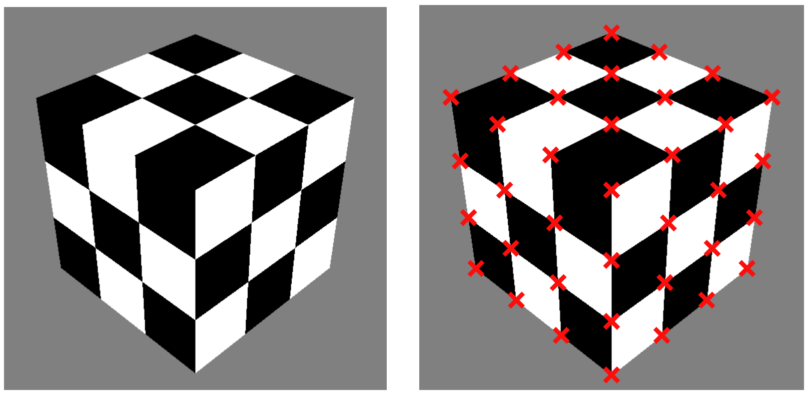
Problem Set 1

Assigned: 02/19/2004
Due: 03/02/2004

Problem 1 *Camera Calibration*



(a) Original image. (b) Detected features.

Figure 1: Input image and detected features.

The goal of this problem is to implement a linear calibration algorithm in MATLAB based on the method described in Section 3.2 (Forsyth and Ponce). Provided an input image, we want to extract the intrinsic (focal length and center of image) and extrinsic (rotation and translation) parameters of the camera used to grab this image. We assume no radial distortion.

A typical way to calibrate a camera is to take a picture of a calibration object, find 2D features in the picture and derive the calibration from the 2D features and their corresponding positions in 3D. In our case, we use a 2m-wide cube as calibration object textured with a checkerboard pattern (see Figure 1(a)). We search in the image (of size 600x600) for the 2D features corresponding to corners of the checkerboard. Figure 1(b) shows these features.

Since we know its exact size (2m), we can find the exact 3D position of each 2D feature relative to the center of the cube. This process of finding correspondences is simple but time consuming. We did this part of the work for you. `Features2D.mat` and `Features3D.mat` contain the 2D corner features and the corresponding 3D positions.

(a) Your first task is to write a MATLAB function which takes these two lists as input and returns the calibration parameters as output. Your function should have the following syntax:

$$\texttt{function}\ [\alpha, \beta, \theta, \texttt{u0}, \texttt{v0}, \texttt{R}, \texttt{t}] = \texttt{calibrate}(\texttt{f2D}, \texttt{f3D})$$

where α and β are the horizontal and vertical scale factors of the camera CCD (in pixels), θ is the camera skew (in radians), `u0` and `v0` are the center of the image (in pixels), and `[R t]` is the relative rigid transformation between the center of the cube and the camera. Also compute the camera focal length, f, in meters. The size of the camera CCD is (1 x 1) square inches and its pixels are square (i.e. $\alpha = \beta$).

(b) To check your solution create the function,

$$\texttt{function f2D} = \texttt{pcamview}(\texttt{f3D}, \alpha, \beta, \theta, \texttt{u0}, \texttt{v0}, \texttt{R}, \texttt{t})$$

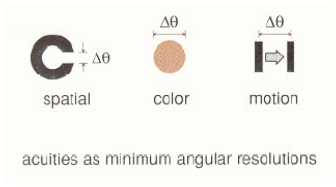
where `f2D` are the 2D projections of the points `f3D`, and the camera intrinsic and extrinsic parameters are as defined above. Using this function project the provided 3D features onto the camera focal plane. Check your solution from part (a) by superimposing the projected points onto the original image `checkercube.bmp`.

Please submit the files `calibration-[last_name].m` *and* `pcamview-[last_name].m` *by email to* `6.891-submit@ai.mit.edu`*. In your problem set solution, write the values* (α, β, θ, `f`, `u0`, `v0`, `R` *and* `t`) *you found using the cube data set. Also include the result image of part (b) and a printout of your code.*

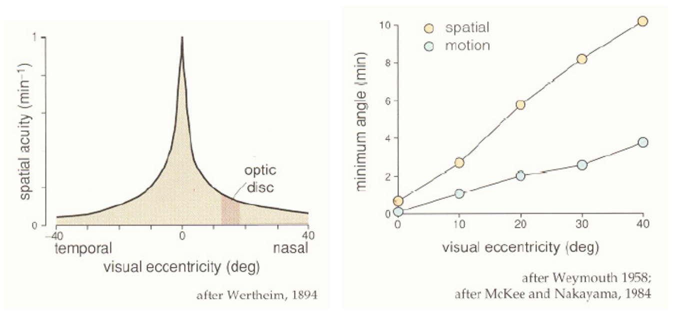
Problem 2 *Image Pyramids*

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 this problems, you may always use the default decomposition filters. For this problem submit your MATLAB code and include a printout.*

Subjectively, our visual world appears to us to be high resolution everywhere. However, we have much higher spatial resolution in the center of our



(a)



(b)

Figure 2: (a) Measures of acuity. (b) Plots of eccentricity versus accuity.

field of view than in the periphery. In this problem, we will synthesize an image approximating our visual resolution as a function of eccentricity.

Figure 2 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,

$$a = 0.23e + 0.7. \tag{1}$$

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 $\tan(\theta) \approx \theta$.

(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? Assume that the highest resolution image point lies at half the maximum acuity, where maximum acuity is as specified by (1).

(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, 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).*

Problem 3 *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:

$$\texttt{synthIm} = \texttt{SynthTexture}(\texttt{sample}, \texttt{w}, \texttt{s})$$

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:

$$[\texttt{bestMatches}, \texttt{errors}] = \texttt{FindMatches}(\texttt{template}, \texttt{sample}, \texttt{G})$$

where `bestMatches` is the list of possible candidate matches with corresponding SSD errors specified by `errors`. `template` is the $w \times w$ 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 $(1+\epsilon)$. 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 $\epsilon = 0.1$ 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:

$$\texttt{Mask} = \texttt{G} \, .* \, \texttt{validMask}$$

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 $w \times w$ 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* findmatches-[last_name].m *by email to* 6.891-submit@ai.mit.edu.

Problem 4 *Color*

The Matlab data file CIE.mat contains the spectra which will be needed for this problem. The vectors c_x, c_y, and c_z give the CIE color matching functions for the X, Y, and Z color coordinates, respectively (these are also plotted in Figure 6.7 of Forsyth and Ponce). These functions are sampled at 5 nm intervals for wavelengths from 400 through 700 nm.

(a) What are the CIE X, Y, and Z coordinates of the light corresponding to the power spectrum in the vector s_A? What are the normalized CIE coordinates, x and y?

(b) Suppose you have a test color specified as a linear combination of primaries at 450, 550, and 650 nm, given by $(a, b, c)^T$, respectively. What linear combination of light sources at 460, 510, and 590 nm is need to match the test color?

(c) We want to specify one possible spectral power distribution of the non-physical primary lights of the CIE color coordinate system, corresponding to the color matching functions c_x, c_y, c_z. Suppose you insist that the spectral power of the CIE primaries be zero for all wavelengths except at 460, 510, and 640 nm. Specify the linear combination of lights at 460, 510, and 640 nm that corresponds to the primary lights of the CIE color coordinate system.