Assignment 5

CS283, Computer Vision
Harvard University

Due Fri, Oct 13 at 5:00pm

Edges, corners, and multi-resolution representations are the topics covered in this assignment. As usual, the assignment will be submitted via canvas and formatted according to the guidelines.

1. *(10 points)* In class we derived a finite-difference approximation to the derivative of the univariate function $f(x)$ by considering the Taylor polynomial approximations of $f(x+h)$ and $f(x-h)$. We showed that
$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2),$$
so that the derivative can be approximated by convolving a discrete version of $f(x)$—a vector of values $(\ldots, f(x_o - \Delta),\ f(x_o),\ f(x_o + \Delta), \ldots)$—with kernel $(1/2,\ 0,\ -1/2)$. This is termed a *central difference* because its interval is symmetric about a sample point.

   (a) Derive a higher order central-difference approximation to $f'(x)$ such that the truncation error tends to zero as h^4 instead of h^2. *Hint*: consider Taylor polynomial approximations of $f(x \pm 2h)$ in addition to $f(x \pm h)$.

   (b) What is the corresponding convolution (not correlation!) kernel?

2. *(20 points)* In this question you will build your own edge detectors with sub-pixel accuracy, borrowing heavily from Chapter Four of Szeliski. Your detector `[X,G]=edgels(im,sigma)` takes an image `im` and a scale parameter `sigma` (σ) at which derivatives are computed. The output `X` is an $N \times 2$ array where each row stores the (x, y) pixel coordinates of an "edgel", and the output `G` is another $N \times 2$ array with each row storing the corresponding gradient vector (I_x, I_y). Substantial skeleton code for this problem can be found in the `edgels()`. To complete this function, you will need to write three parts:

   (a) A helper function `[Dx,Dy]=deriv(sigma)` that creates derivative-of-Gaussian kernels for horizontal and vertical derivatives. (Enormous hint: check out the code for the existing helper function `lapgauss`)

   (b) For each quadruple of pixels, $\{(i,j), (i+1,j), (i,j+1), (i+1,j+1)\}$, code that counts the number of zero crossings along the four edges. When there are exactly two zero-crossings, the code must compute the two sub-pixel zero-crossing locations using Eq. 4.25 in Szeliski (make sure to check the code comments for a typo in this equation), and then the midpoint of these two zero-crossing locations as the final edgel location.

   (c) For each edgel, code that samples the image gradient at the sub-pixel location.

   Validate your code by running it on the image `calib_right.bmp` contained in the data folder of this assignment. Superimpose your detected edgels `{X,G}` on the image using the `quiver` command and submit your code and results.

3. *(20 points)* In class we discussed a measure of "cornerness" based on the second moment matrix (also called the auto-correlation matrix),
$$\mathcal{M}_W = \sum_{(x,y) \in W} (\nabla I(x,y))(\nabla I(x,y))^\top. \tag{1}$$
   An alternative approach to corner detection is based on another matrix, the *Hessian*:
$$\mathcal{H} = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}. \tag{2}$$

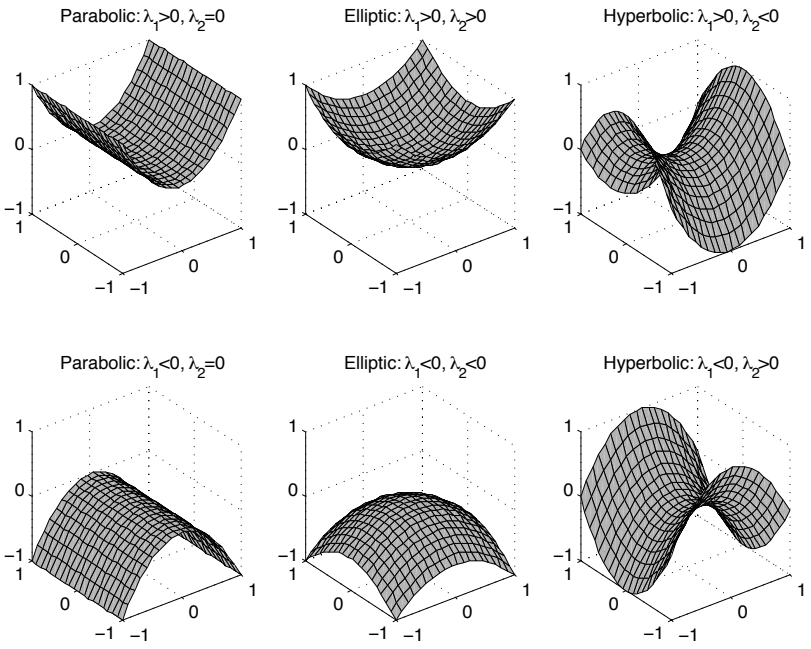

Figure 1: Eigenvalues (λ_1, λ_2) of the Hessian correspond to the principal curvatures of a surface and can be used to classify different surface types.

Like the second moment matrix, the Hessian is real and symmetric and therefore has two real eigenvalues. The eigenvalues and eigenvectors of the Hessian have a different geometric interpretation, however, and this interpretation is depicted in Fig. 1. If we interpret the intensity function $I(x, y)$ as a surface, then the eigenvalues of $\mathcal{H}(x, y)$ correspond to the *principal curvatures* of the surface at the point (x, y) , meaning the values of maximum and minimum curvature at that point. (The eigenvectors are orthogonal and correspond to the directions of maximum and minimum curvature.)

The product of principal curvatures (equal to $\det(\mathcal{H})$) is termed the *Gaussian curvature* and their average (equal to $\frac{1}{2}\text{trace}(\mathcal{H})$) is the *mean curvature*.

(a) Use the image `calib_right.bmp` from problem 2. Convert it to grayscale and display it. Assuming the mean and Gaussian curvature (denoted $H(x, y)$ and $K(x, y)$, respectively) have been computed at each point on the image plane, write an expression for 'cornerness' $C(x, y)$ in terms of H and/or K that will yield a strong response at the corners of the checker pattern and a small response elsewhere.

(b) Write a Matlab function `C=cornerness(I,sigma)` that computes your function $C(x, y)$ from part (a) given a grayscale image and a smoothing parameter σ used to estimate the second derivatives of I. This function should contain no loops. (Also, see Hints and Information below regarding discrete approximations to second derivatives.) Run this function on the grayscale version of `calib_right.bmp` using $\sigma = 2$, display the function $C(x, y)$ as an image and submit it.

(c) Write a function `Co=nonmax_suppression(C)` that returns a 2D array the same size as `C` that satisfies the following rule: $C_o(x, y) = C(x, y)$ if $C(x, y)$ is maximal in its local 3×3 neighborhood, and $C_o(x, y) = 0$ otherwise (see Hints and Information below about morpholotical operators, especially dilation and the corresponding command `imdilate`). Display the image `calib_right.bmp` and superimpose a plot of the points (x, y) that satisfy $C_o(x, y) > \mathit{threshold}$, where *threshold* is manually selected to provide a reasonable balance of false detections and missed detections.

4. *(10 points)* The continuous convolution of two functions $f(x)$ and $g(x)$ is given by

$$(f * g)(x) = \int_{-\infty}^{+\infty} f(y)\, g(x - y)\, \mathrm{d}y. \tag{3}$$

The Gaussian function at scale s is defined as

$$G_s(x) = \frac{1}{\sqrt{2\pi s}} \exp\left(-\frac{x^2}{2s}\right), \tag{4}$$

and has the property that

$$\int_{-\infty}^{+\infty} G_s(x)\, \mathrm{d}x = 1. \tag{5}$$

Prove that this class of functions satisfies the *semigroup property*: the convolution of one Gaussian with another produces a third Gaussian with scale equal to their sum, or

$$(G_{s_1} * G_{s_2})(x) = G_{s_1+s_2}(x). \tag{6}$$

Hints and Information

- Robustly estimating the second derivatives of a function on a two dimensional domain is achieved by pre-filtering the function with a Gaussian kernel. Since convolution is associative, the smoothing and differentiation can be accomplished in a single step using filters that are second derivatives of a Gaussian. For example, the filter for computing the second derivative in the horizontal direction I_{xx} is

$$\frac{d^2}{dx^2}G(x, y; \sigma) = \left(\frac{x^2 - \sigma^2}{\sigma^4}\right) G(x, y; \sigma),$$

  where, as usual, the parameter σ defines the "scale" of the derivative, and a discrete window width of about 5σ is sufficient to accurately represent this continuous filter function. You should be able to write corresponding expressions for I_{yy} and $I_{xy} = I_{yx}$.

- One way to think of convolution is as computing sums of weighted intensity values in a neighborhood around each pixel, with the shape of the neighborhood determined by the shape of the kernel, and the weights by its values. We can then define generalizations of convolution by replacing the sum operation with other functions.

  One such class of (non-linear) generalizations is *morphological operators*, where the sum is replaced by an ordinal relationship: after weighting the intensity values, compute their max, min, median, or their k-th largest value for any integer k. This last case is called *k-ordinal filtering*. The case of max, which is equivalent to 1-ordinal filtering, is known as *dilation*. Finally, the case of min, equivalent to setting k equal to the neighborhood size, is known as *erosion*. For more on morphological operators, you can read Section 3.3.2 of Szeliski.

  As with linear convolution, there are several efficient algorithms for computing morphological operators, many of which are implemented in Matlab. Two-dimensional dilation in particular is provided by the function `imdilate`. You can find out more information in Matlab's very detailed morphological operator tutorial[1], especially the page on dilation[2].

[1] http://www.mathworks.com/help/images/morphological-filtering.html
[2] http://www.mathworks.com/help/images/morphology-fundamentals-dilation-and-erosio n.html