CSE327 Fall 2023 Homework 6 (10pts)

Due Nov 1 2023, 11:59PM, submitted via Brightspace.

For this homework, you will find a set of corner point features in an image. Future homeworks will build upon this one, for example, by matching these corners across multiple images to register camera views. This current homework emphasizes concepts of linear operators, convolution, gradient estimation, and corner feature detection.

Write a function [corners, R] = detectHarrisCorners(Image, Sigma, N, D, M) with input parameters:

Image = a grayscale image
Sigma = Gaussian smoothing kernel sigma,
N = size of the local NxN neighborhood for accumulating sums for Harris corner detection,
D = radius of neighborhood for suppression multiple corner responses,
M = number of corners we want to detect.

and output parameters:

Corners = M*2 matrix where each row consists of [x, y] for each corner point,
R = R scores for all the pixels

The function should perform the following steps:

1) Smooth the image using a Gaussian kernel with a given sigma Sigma. Sigma is given as an input parameter, so based on its value you must decide on the size of your smoothing kernel and fill in its values according to the Gaussian function. You will convolve the image with the smoothing kernel to produce a smoothed image.

2) Compute gradient images Gx and Gy from the smoothed image from step 1 by convolving with row and column finite difference kernel operators for computing partial derivatives in x and y.

3) Compute Harris corner "R" values over local neighborhoods of each pixel, using the gradient images Gx and Gy from step 2. Harris corner detection is summarized here for convenience: For each pixel, and a given neighborhood size N,

a) compute the products of derivatives Gx^2, Gx Gy, and Gy^2 at every pixel.

b) compute sums of these products over the local NxN neighborhood at each pixel e.g. Sx2 = sum over Gx^2, Sxy = sum over GxGy, Sy2 = sum over Gy^2. Hint: you can compute neighborhood sums quickly using a box filter!

c) define at each pixel (x,y) the matrix

$$H(x,y) = \begin{bmatrix} S_{x2}(x,y) & S_{xy}(x,y) \\ S_{xy}(x,y) & S_{y2}(x,y) \end{bmatrix}$$

d) the Harris corner value R(x,y) at pixel (x,y) is then defined as

R(x,y)=det(H(x,y))-kTrace(H(x,y))^2

Or, R(x,y) = [Sx2(x,y)*Sy2(x,y)-Sxy(x,y)*Sxy(x,y)] – k [Sx2(x,y)+Sy2(x,y)]^2

where det is the determinant of the 2x2 matrix H(x,y), and k is a predefined constant, let's say 0.05.

4) Extract a sparse set of the M "best" corner features. Unfortunately, whenever there is a strong R response, there are often several strong responses nearby. Basically, these represent

multiple detections of the same corner in the image. We need to filter these out to get a sparse set of corners. One way to do this is to extract your corners in decreasing order of R value, i.e. starting at the pixel with maximum R value, then setting all pixels within some distance D of that pixel to a very small value, effectively suppressing them. After that, you then find the pixel with the next highest R value, and so on. Note: there are other strategies for performing this “nonmaximum suppression”, but this is one of the simplest to implement.

Read in the given image (or images from your nice smartphone) and convert to a grayscale, double float image. Test the above function.
Draw the detected corner points as rectangular boxes on the images.
Visualize your R-score image.

Turn in a running version of your code. Put all codes, functions, and input images in a single directory, then make a zip file of that directory for submission via Brightspace.

Bonus (2 points)
Apply the eigen-decomposition on the H matrix of each pixel and use the smallest eigen-values as the metric to detect corners. Compare this method with the Harris corner.