CMPEN/EE 454 Project 1
Harris Corner Detection

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

The algorithm should contain the following steps:

1) Read in a file of input parameters that will govern how the following program behaves. These parameters include image filename, S = Gaussian smoothing kernel sigma (standard deviation), N = size of the local NxN neighborhood for accumulating sums for Harris corner detection, D = radius of neighborhood for suppression multiple corner responses, and M = number of corners we want to detect. To make sure we are all on the same page as far as inputting these parameters, a matlab function read_corner_parameters.m has been provided, which you can call by

   [filename,S,N,D,M] = read_corner_parameters('cornerparams.dat');

   Feel free to change the names of the parameters to be something more descriptive. Look at the sample file 'cornerparams.dat' to see what the input format is.

2) Read in the given image and convert it to a grayscale, double float image.

3) Smooth the image using an isotropic Gaussian kernel with a given sigma S. S 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 from step 2 with the smoothing kernel to produce a smoothed image. Note… you may wish to use the property of separability to replace convolution via a 2D Gaussian by a more efficient pair of convolutions using two 1D Gaussians.

4) Compute gradient images Gx and Gy from the smoothed image from step 3 by convolving with row and column finite difference kernel operators for computing partial derivatives in x and y. We have discussed this in class lectures. Gx and Gy are images where each pixel contains one of the two components of the gradient vector at that pixel. Gx contains partial derivatives wrt x (across rows), and Gy contains partial derivatives wrt y (down columns).

5) Compute Harris corner "R" score values over local neighborhoods of each pixel, using the gradient images Gx and Gy from step 4. Although we covered Harris corner detection in a lecture, it is summarized here for convenience: For each pixel, and a neighborhood size N,
   a) compute the products of derivatives Gx Gx, Gx Gy, and Gy Gy at every pixel.
   b) compute sums of these products over the local NxN neighborhood at each pixel e.g. Sx2 = sum over Gx Gx, Sxy = sum over Gx Gy , Sy2 = sum over Gy Gy hint: you can compute neighborhood sums quickly using a box filter containing all 1.0 values (no normalization to sum to 1).

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

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

d) the Harris corner value R(x,y) at pixel (x,y) is then defined as
R(x,y) = det(H(x,y)) - k Tr(H(x,y))^2
where det is the determinant and Tr the trace of the 2x2 matrix H(x,y), and k is a predefined constant, let's say 0.05. [note that you are squaring the trace]

6) 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.

7) Finally output your corner features into an ASCII file. The file should have a very specific format, so that we are able to write programs that read it in for evaluation (also, future projects may also read in this format). The first line of the file contains the integer number of corners that were found. Following that is one line per corner specifying the Harris R value, column location and row location of the corner found.

%file format for corner detection project output
Line1: integer number of corners found
Line2: Rvalue1 Column_location1 Row_location1 [of the first corner]
Line3: Rvalue2 Column_location2 Row_location2 [of the second corner]
…and so on…

Any lines in the file beginning with a '%' character will be ignored by our evaluation program, thus allowing you to place comments in the file. (for example, comments to remind yourself 20 years from now what the file format means).

Half of your grade will be based on submitting a "runnable" program, which we will evaluate on new images with new parameters (more on this below), and the other half will be based on a written report discussing your program, design decisions, and experimental observations. In particular, we want you to turn in the following…

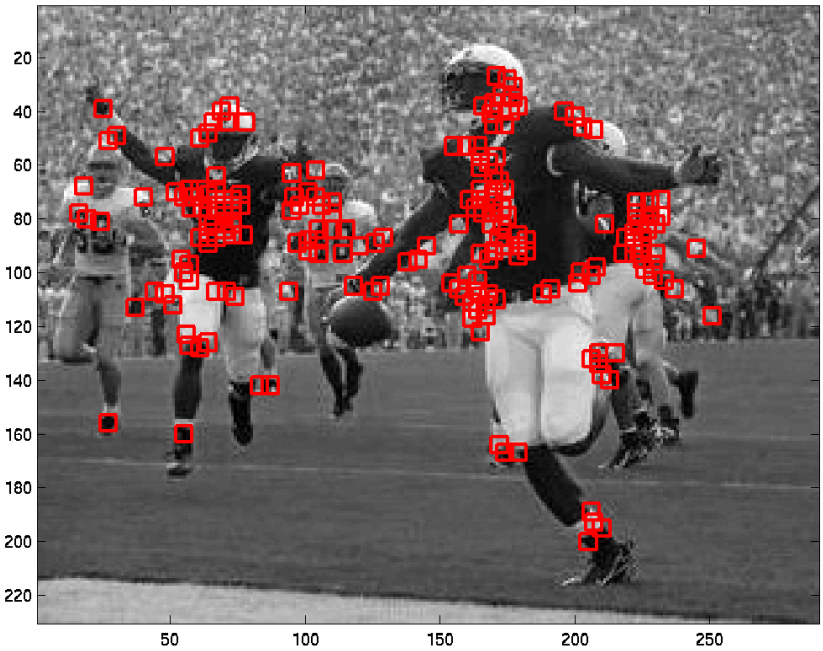
1) Submit a written report in which you discuss at least the following things:
   a) Summarize what you think the project was about (what was the task; what were you trying to achieve).
   b) Present an outline of the algorithmic approach along with a flowchart showing the flow of control and subroutine structure of your code,

c) Run your program using the supplied "cornerparams.dat" file and show pictures of intermediate and final results that convince us that the program does what you think it does. Change the file to use different sample images (e.g. psutd.jpg, checkerboard.jpg, stickfiguredance.jpg, psulion.jpg) and show the resulting Harris R image, and the top N corners found, and any other images you think are interesting (maybe the gradient Gx and Gy images produced). Certainly show a picture of the final output corners found for each image, for example as a greyscale image with boxes overlaid around all corners found. A sample picture that the instructor's version of the program produces on psutd.jpg for 200 corners is shown below.
d) Explain any design decisions you had to make. For example, how did you decide how big the Gaussian kernel had to be? Did you make it separable or not, and why? Be sure to document any deviations you made from the above project descriptions (and why), or any additional functionality you added to increase robustness or generality of the approach.
e) Experimental observations. What do you observe about the behavior of your program when you run it? Does it seem to work the way you think it should? Play around a little with parameter values to see what happens. Is the set of corners that you find affected by the amount of Gaussian smoothing you do? Are some values clearly out of the range of what would be useful? Are some parameters coupled (if you increase one you also should increase the other)?
f) Document what each team member did to contribute to the project. It is OK if you divide up the labor into different tasks (actually it is expected), and it is OK if not everyone contributes precisely equal amounts of time/effort on each project. However, if two people did all the work because a third team member could not be contacted until the day before the project was due, this is where you get to tell us about it, so the grading can reflect the inequity.

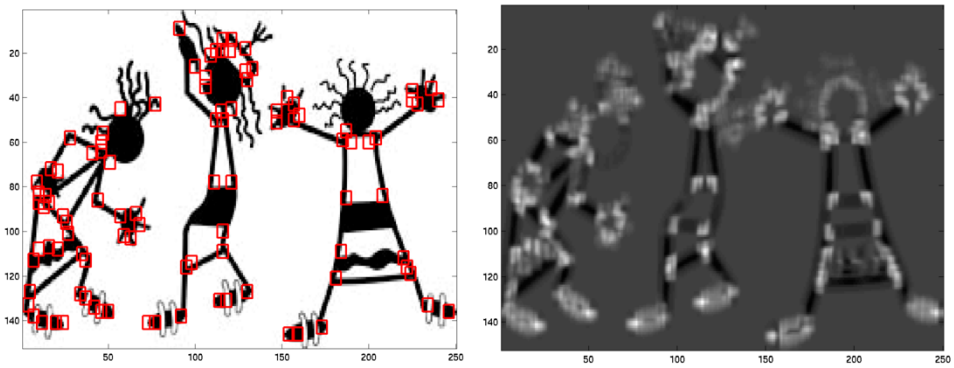
2) Turn in a running version of your code. Put all code, subroutines, data files your program needs, and also your written report, in a single directory, then make a zip file of that directory for submission on Canvas. The graders can then unzip it and go from there. Make it clear what the name of the main routine / file is that we should execute. Basically, we are going to run your code using a different cornerparams.dat file (different image and different params) to see if it works, and how well it produces the output that we think it should produce. If we have trouble running your code we will contact you. If the trouble occurred because we did something dumb, you won't lose points, but if it turns out that you were dumb, you will.

OK, that's all I can think of for now. If there are any questions, ask in class or send email to the TA or myself.

Some sample output pictures are shown below.



Output showing the 200 best corners found using the instructor's solution to this project assignment. What is shown here is a greyscale version of the input image psutd.jpg overlaid with red 7x7 boxes centered at locations where corners were found.



Another sample output with the 100 best corners found using the instructor's solution to this project assignment. Also shown, to the right, is a greyscale version of the Harris R score computed for this example. Note that the top N corners found correspond to the top N highest peaks in this Harris R score image.