Homework #3 - Gradients and Edges - Due Thursday June 4th, before 11:55pm

Basic Description:

In the first part of this assignment you will program three functions that manually peform operations on images. The first two functions will involve you manually differentiating an image in the x and y axis. The third function will be a much more generic function which will use cross correlation to apply a kernel to an input image (View lecture slides 2-05, 2-06 and 2-07 for a refresher on this).

In the second part of the assignment we will allow you to perform edge detection on an image using OpenCV's functions. You will then describe the different things you attempted to get the edges of an image, and include your output in a PDF report.

Part 1: Programming Gradients

The first part of the assignment requires you to program three functions. You may download the assignment files (zip file) that include the python template and a basic test file that will verify your functions output the correct type after you have written them (Note: simply because the tests succeed does not mean your function is doing the correct thing, you must verify this yourself, the test serves as a very basic check).

This zipped folder contains the following files:

assignment3.py - This is where you will write the code for the three functions we provide.

assignment3_test.py - This can be used to test if your code is outputting the correct data type.

README - Some basic instructions on how to run the unit test.

test_image.jpg - A basic test image of a butterfly.

The function documentation details exactly what you need to do, but a basic description is provided here:

imageGradientX: This function takes in a grayscale image, and you must differentiate the image in the X direction, without using openCV functions! You are manually each subtraction manually using iteration.

imageGradientY: This function takes in a grayscale image and you must differentiate the image in the Y direction (very similar to #1).

computeGradient: This function applies a 3x3 input kernel to the input image. The output is the computed gradient for the image.

As with the previous two functions, you will NOT be using the OpenCV built in functions, and instead will be manually performing a cross-correlation number by number.

Part 2: Edge Detection

For the second part of this assignment, we have an open-ended task. We want you to produce an edge detected output image from an image of your choosing (choose or take an image that has clear edges to get better results). We do not care how you produce it, but if you want to get the most out of this assignment, here are some suggestions:

Try using different kernels with your computeGradient function. After you do that, apply the convertToBlackAndWhite function you wrote in HW 1. See what results you get, and play with the threshold of 128, maybe the thresholding should be more / less depending on the image you use!

Look into OpenCV's Canny Edges for classic edge detection results to compare to if you try the above so you get an idea as to how to get an image that simply displays the edges.

After you find a result that you like, add that to a PDF, and describe what you did to accomplish these results!

Please turn in the following files to T-Square.

assignment3.py

Please put your name and email at the top of this assignment. Please do not include print statements, or statements like imread or imwrite in your final submisison. You are free to do that to test your output but when you submit it, please comment these out as it makes grading a bit more difficult for us.

assignment3.pdf

In the PDF we want you to write how you approached the first two functions (you can describe both of these together since they are very similar), and how you approached the third function. After this, we want you to include your edge-detected result image (as well as show the input image) for Part 2, and describe what you did to obtain this image.

Grading Rubric:

part 1: 40%

- imageGradientX: 10%
    - diffs image in x direction
    - no opencv functions
- imageGradientY: 10%
    - diffs image in y direction
    - no opencv functions
- computeGradient: 20%
    - no opencv functions
    - manual cross-correlation

part 2: 40%

- produce edge detection

- Approaching the quality of the Canny Edge procedure will be a 40%
- will eyeball Canny Edge function vs student's result
    - the closer to the function the more points awarded

pdf: 20%

- explain the process for differentiating an image in x and y directions: 5%
- explain computing the gradient using a kernel vs differentiating the images in separate directions: 5%
- explain your own specific method for edge detection: 10%