CSCI-631-02 Computer Vision HW-04

You can talk with each other, but submit your own independent work. You are responsible for understanding everything you submit.

How to submit your homework:

A. Create a directory named HW04_Lastname_Firstname.
B. Put everything in that directory: code, and PDF of your write-up.
C. Zip up the entire directory, and submit it to the dropbox.

Remember, submit your programs, and a PDF write-up of what you did, all combined into one single ZIP file describing the results of your experiments. The grader and/or professor might run your code, or might just read your write-up. Be sure that both your code and your write-ups show good documentation practices and correct English.

In your write-up, provide evidence of learning and evidence of understanding.

You will receive an email separately indicating which image to process. Use that image for all parts of the homework.

1. **Block Smoothing**: (3 pts)
   The goal here is to be sure you know how to work on a single channel of an image, and smooth pixels. The intent is to give you a chance to practice your Matlab coding, but talk with others if needed.

   Write a routine called local_block_smear(input_image). This implements block averaging over a fixed 5x5 neighborhood. It is given an input image, and computes the local average of every single pixel in the image. Inside your code there should be something like:

```
function output_image = local_block_smear ( input_image )
dimensions = size( input_image )

if length( dimensions ) > 2
      % add your comments here.
    input_image = rgb2gray( input_image );
end

% Default return values …
output_image = input_image;

% Add comments about all of this, and everthing that is happening.
% What are the 3's for ??
for row = 3 : (dimensions(2) - 3)
   for col = 3 : (dimensions(1) - 3 )
      sum = 0;
       for ii = -2 : 2
          for kk  = -2 : 2
               sum = sum + input_image( col + ii, row + kk );          % Add comments.
          end
       end
       output_image( row, col ) = sum / 25;
   end
end
```

   In your write-up include a picture of your original image in grayscale.

   Then include a picture of the image after it has been run through the smearing routine.

   Do you notice any particular differences? Do you notice anything odd about the resulting image? How would you modify this to use an arbitrary [n by m] sized filter?

   In your write-up, provide evidence of learning and evidence of understanding.

2. **Canny Edge Detection – non maximal suppression only:** (6)
Write a routine named maximal_edges(input_image)
This implements the part of the Canny edge detection that only finds the maximum edge in each direction.

You will need many copies of the image to use in the process of computing this.

a. Filter the image using a Gaussian filter to remove small edges:
```
fltr    = fspecial('Gauss', 43, 3 );   % Play with the values here to remove the unimportant edges.
im2     = imfilter( input_image, fltr, 'same', 'repl' );            % Takes a second or two…
```

b. For each pixel in im2, use a Sobel filter as specified in the lecture, to estimate the df/dx and df/dy of the gradient at each pixel, in each plane of the image. (For all colors given.) Use imfilter. Do not use edge( ).

c. Compute the edge magnitude at each pixel.
d. Compute the edge direction at each pixel (the angle).
e. Convert the edge angle to a multiple of 45 degrees. This means using the round( ) or floor( ) functions. The angle 0 should round to 0. The angle -20 should round to zero. This should give you a few angles to handle: 0, 45, 90, and 135.

f. For every pixel in the intermediate image, only keep edges if they have a larger edge magnitude than the pixel infront of them and behind them. So, if the edge is at 45 degrees, only keep that edge if it has a stronger edge gradient than the pixel up to the right and down to the left of it.

g. Edges that are not maximum are set to zero in the output.

h. The output image will be only those edges that are the maximum edges. However, they are not binary. They have the actual edge strength.

In your write-up, show the input image, using imagesc( ), and the output image generated. This means you will need to run the code, and save the output using imwrite( ) to another file.

Discuss the results that you get with different sized blurring amounts that were applied in part a. What impact does larger standard deviations cause? What do smaller standard deviations cause?

3. **Bonus (1)**
As a bonus point, implement hysteresis matching. This means that the surrounding edges must be at least a certain edge strength to be kept as starting edges. AND then those edges are continued as long as the image edge strength has at least a second (lower) edge strength. Do a separate write-up for this part. Show your results.