EN1610 Image Understanding

Lab # 3: Edges

The goal of this fourth lab is to

- Understanding what are edges, and different ways to detect them
- Understand different types of edge detectors - intensity, histogram, and texture

All the images you will need for the lab can be downloaded off the class website, `http://vision.lems.brown.edu/engn161/fall2011/Labs`.

Edges

Problem 1. Types of Edges Give a few examples of each of these types of edges in the Images in Figure 1

- Reflectance Edges
- Texture Edges
- Highlight Edges
- Shadow Edge

Figure 1: Various Edge Images

Figure 2: Images for Lab

Problem 2. Intensity Based Edge Detection In this problem you will explore a very simple edge detector. You will run your detector on the five images in Figure 2

a) Load the Images and smooth it with a Gaussian filter to eliminate noise. Remember to convert to grayscale first

b) Use the gradient function (`[dy,dx]=gradient(A);`) to compute the 1st derivatives in the x and y direction

c) Look at the magnitude of the gradient image using imshow, $M = sqrt(dx^2 + dy^2)$

d) Now use the command `quiver(dy,dx)` to view the actual gradient vectors; you may want to zoom in. (The image will be upside down, don't worry about it). It is very important that you understand what is going on here for this and the next lab.

e) Create a binary edge image by thresholding the gradient magnitude image. Pick a value that gives you few gaps in the edges.

f) Finally thin the edges using the following process:

   (a) Visit every edge pixel in the binary image. For each one, look at the gradient direction at that point (dy,dx) and round it to the nearest of the 8 pixel directions.

   (b) Look at the gradient magnitude of the pixel in that direction and also at the pixel in the opposite direction.

   (c) If the pixel in question has a gradient magnitude less than either of those two neighbors, set its value in the binary image to 0.

Submit the thinned and unthinned images.

Problem 3: Canny Edge Detector Run Matlab's Canny edge detection ("help edge" in Matlab) over the images in Figure 2

- Determine the strengths and weaknesses of Canny relative to the edge detector you implemented in Problem 2. Specifically look for the success of each algorithm with respect to corners, curved edges, noise inclusion/exclusion, etc.
- Play with the parameters of Canny - the two thresholds and the Gaussian sigma.
- Which of the edge types above are most easily detected; which are more difficult. Provide specific examples and images to illustrate your findings.

Problem 4. Histogram Based Edge Detection Similar to the intensity based edge detector, histogram based edge detectors look for local discontinuities in brightness at each pixel. Just as you used the gradient for intensity based edge detector, a similar function is defined to look for local changes in intensity. At a location (x, y) in the image, draw a circle of radius r, and divide it along the diameter at orientation θ. The gradient function $G(x, y, \theta, r)$ compares the contents of the two resulting disc halves. A large difference between the disc halves indicates a discontinuity in the image, along the disc's diameter. A picture of this can be seen in Figure 3. Run this on the images in Figure 2

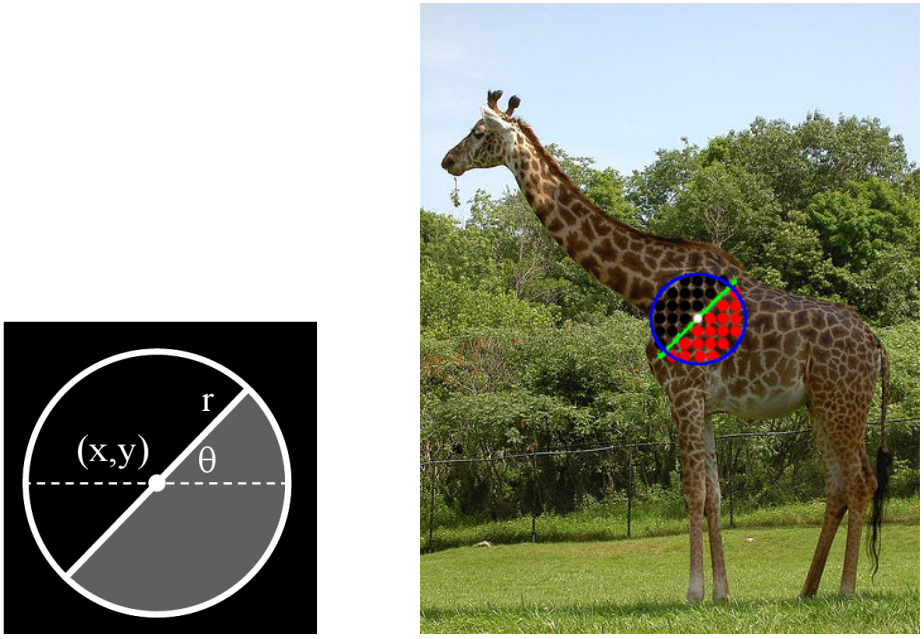

Figure 3: Histogram Based edge detection window

```
function output=histogram_edges(image,radius,numb_orientations,threshold)
```

a Use `im2double` and convert the image pixels from 0 to 1

b You will have three loops for this code, two loops for the raster scan of the image, and one loop for the number of orientations, i,e. $0, \frac{\pi}{4}, \frac{\pi}{2}, ..., \pi$. You should store your intermediate results in a 3d matrix, where each plane corresponds to an orientation.

c For each pixel you visit, you will define a circle based on the radius r, and use the theta to determine the left and right half's, Think about what you want to do if a portion of a pixel is within the circle. Essentially you need to find a mask to determine which pixels to consider on the left and right of the edge

d For each half of the circle , you will create a histogram , use `hist`. Remember to normalize your histograms

e To compare each histogram use the χ^2 distance, Equation 1, you will have to experiment with bin size

$$\chi^2(g,h) = \frac{1}{2}\sum \frac{(g_i - h_i)^2}{g_i + h_i} \tag{1}$$

f Now for every pixel of the image we have $G(x, y, \theta, r)$, and we will define the *orientation* at the center point of the circle, as $\hat{\theta} = \arg\max_\theta G(x, y, \theta, r)$ and the *strength* to be $G(x, y, \hat{\theta}, r)$, this will now reduce your storage to just , [w x h]

g After finding the maximum response along orientation, we need to do non-max suppression as before

h Finally, make the image a binary map, by thresholding the resulting edge map, keep in mind, that you are thresholding the χ^2 distances which may not be in the range of 0 to 1

i Parameters, You have many parameters to play around with, I am giving some examples, Try different settings for each of the images in Figure 2, and report what you think is best,

bins	16,32
norient	6,8
radius	5,10,20

Problem 5. Texture Based Edges We will now be looking at texture edges on the images in Figure 2

a First run a filter bank on the image, Now each pixel has a 48 dimensional vector response

(a) Oriented odd-symmetric filters, at 3 scales and 6 orientations, modeled as rotated copies of the horizontal filter, $f(x, y) = G'_{\sigma 1}(y)G_{\sigma 2}(x)$ where $G(x)$ is given by Equation 2. Use a ratio of 3 for $\sigma_2 : \sigma_1$. Set the 3 scales to be a "half-octave" apart, i.e. $\sigma_1^{i+1} = \sqrt{2}^{\sigma_1^i}$ where $\sigma = 1, 2, 3$

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \tag{2}$$

(b) Oriented even-symmetric filters at 3 scales and 6 orientations, again rotated copies of a horizontal filter, this time $f(x, y) = G''_{\sigma 1}(y)G_{\sigma 2}(x)$

(c) Generate 8 Laplacian of Gaussian (LOG) filters, and 4 Gaussians. Use 4 scales with that are again half octave apart. Use matlab's `fspecial` command and use the appropriate filter arguments. Set the 4 scales to be a "half-octave" apart, i.e. $\sigma_1^{i+1} = \sqrt{2}^{\sigma_1^i}$ where $\sigma = 1, 2, 3, 4$

(d) A picture of the full filter bank can be seen in Figure 4

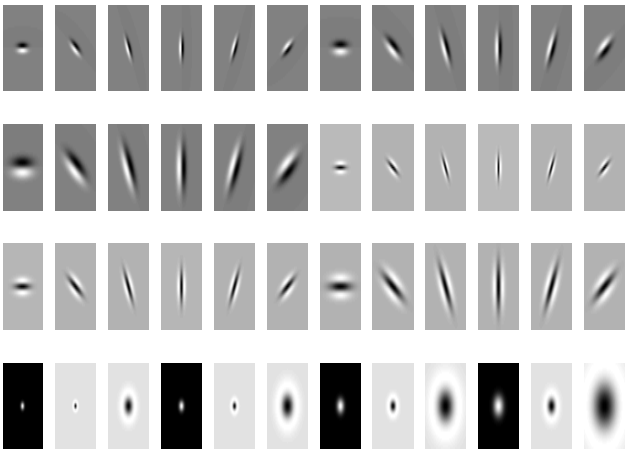

Figure 4: 48 Filter Bank

b Now visit each pixel location (x,y) and consider a patch around it, now look at a patch some distance from (x,y) to the left, right, up and down, For each pixel in the adjacent patch compute the Euclian distance with the patch surrounding the pixel (x,y) under consideration, Now each patch has a distribution of distances. Figure 5 shows the pixel under consideration with the red box, surrounded by its patches

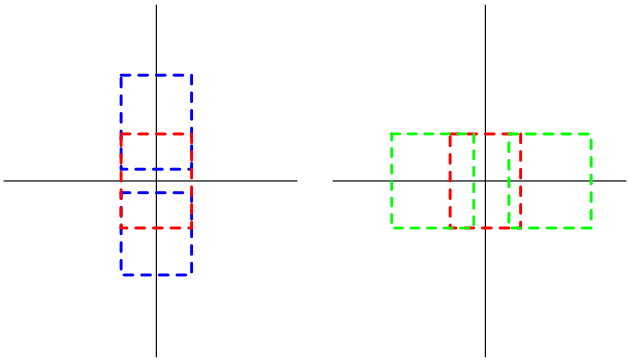

Figure 5: a) Up Down Patches b) Left/Right Patches

c Now do a histogram for each of the four patches, and look at the χ^2 distance between the up/down patches, and the left/right patches, record the minimum distance

d Threshold , the χ^2 distance, and that is the location of your texture edges

Edge Linking

Problem 6. Edge Linking Perform dual threshold edge linking using your gradient based edge detector from Problem 2. Again, run on the images in Figure 2

a) At the stage in which you threshold the gradient magnitude image, create two images, one using a high threshold Th and another using a low threshold Tl.

b) Visit every edge point in the image created from Th. For this point, determine the edge direction (hint: it's normal to the gradient direction) and round to the closest of the 8 pixel directions.

c) Look at the pixel in that direction. If the pixel has a gradient magnitude $T_l < m < T_h$, then add that point to the binary edge map (the one created from T_h), move to that pixel and repeat. If the pixel has a gradient magnitude $< T_l$, or $> T_h$ (already in the edge map), then stop and visit the next pixel.