ENGN1610&2605 Image Understanding
Lab03: Edges

The goal of this lab is to

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

Part I: Edge Detection

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

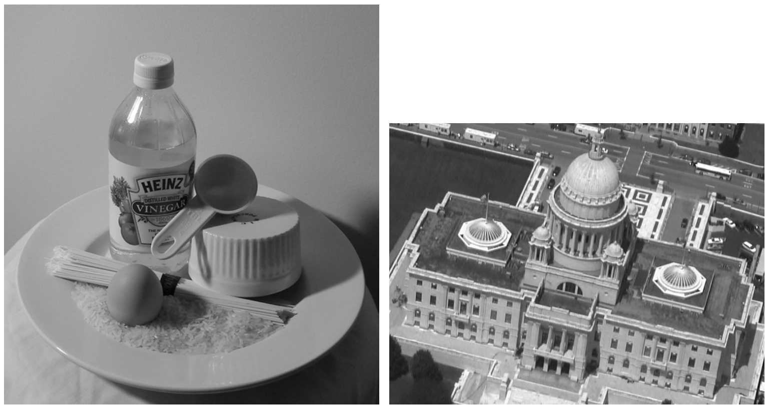


Figure 1

Problem 2. Intensity Based Edge Detection In this problem, you will explore a very simple edge detector running on six images in Figure 2. Create a function for intensity based edge detector:

```
function edge_img = intensity_edge_detector(image,threshold)
```

The necessary steps to follow are:

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(image);`) to compute the first derivatives in the x and y direction.

c) Create a magnitude map M of the gradient image, *i.e.*, `M=sqrt(dx^2+dy^2)`. You can observe how M looks like using `imshow`.

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

e) Perform **non-max supression**:

   (a) For each pixel p in the image, look at its gradient direction (dy, dx) and see which of the 8 directions it belongs to.

   (b) Look at the gradient magnitude of the pixel q located at the gradient direction of p and also look at the pixel $\bar{q}$ located at the opposite gradient direction of p.

   (c) If the gradient magnitude of p is less than either the magnitude of q or $\bar{q}$, set the gradient magnitude of p to 0.

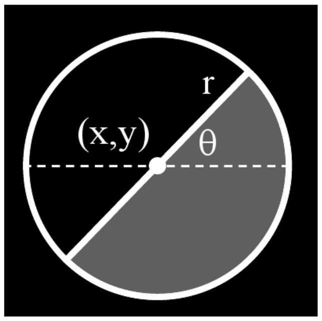
f) Create a binary edge image by thresholding. Pick a value that gives you only a few gaps in edges.

In your report, show the final binary edge image and the threshold you pick.

Problem 3: Canny Edge Detector Run Matlab's Canny edge detection (you can type `help edge` in the MATLAB command window for instructions) over the images in Figure 2.

- Describe 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 edge detector, *i.e.*, the two thresholds and the Gaussian sigma.
- Which of the edge types above are most easily detected? 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 then 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



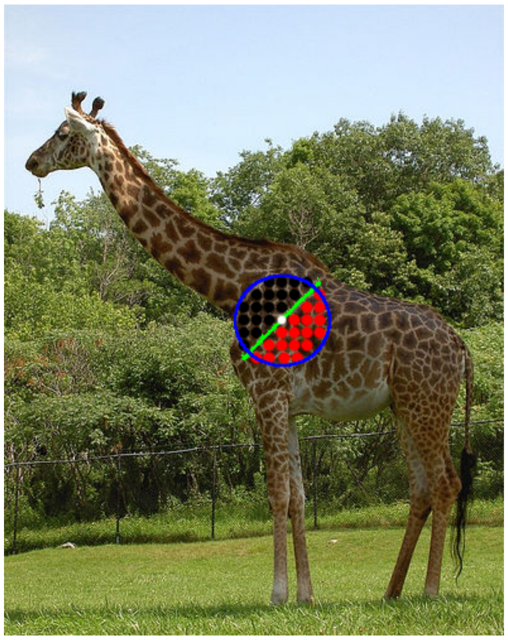


Figure 3: Histogram based edge detection window.

Create a function for the histogram based edge detection:

```
function [edge_map,orient_map]=hist_edge_detector(img,rad,num_orient,num_bins)
```

The inputs `rad` is the radius of a circle, `num_orient` is the number of orientations, and `num_bins` is the number of histogram bins. The outputs `edge_map` is an edge map, and `orient_map` is an orientation map. The necessary steps to follow for implementing a histogram based edge detector are:

a) Convert the input image from color to grayscale. Also make sure to convert the class of the image from `uint8` to `double`.

b) Use `im2double` to rescale the image data to a range between 0 to 1.

c) Loop over all numbers of orientations θ, *i.e.*, $0, \frac{\pi}{4}, \frac{\pi}{2}, ..., \pi$. You can set the number of orientations to 8. (You are also allowed to use `linspace` to help you create the orientations, *i.e.*, `linspace(0,pi,8)`).

   (1) For each pixel, define a circle based on the radius *rad*, and use θ to determine the left and right half's. Think about what has to be done if only a portion of a pixel is within the circle. (Hint: Essentially you can find a mask defined by `meshgrid` and the circle equation to determine which pixels to be considered on the left and right of the disk.)

   (2) For each half of the circle, create a histogram using `hist`. Remember to normalize your histograms.

   (3) To compare histograms of the left and right halves, *i.e.*, g and h, use the Chi-Square distance χ^2, Equation 1, where n is the number of bins. You have to experiment it with the histogram bin sizes.

$$\chi^2_\theta(g,h) = \frac{1}{2}\sum_{i=1}^{n}\frac{(g_i - h_i)^2}{g_i + h_i} \tag{1}$$

   (4) Repeat the process for every θ.

d) Now for each pixel, the final orientation is defined by $\hat{\theta} = \arg\max_\theta \chi^2_\theta$ and the *strength* (magnitude) of the orientation is $\chi^2_{\hat{\theta}}$. The map that stores the final orientation $\hat{\theta}$ of every pixel is the output `orient_map` of your function `hist_edge_detector`.

e) Do non-max suppression **orthogonal to the final edge orientation** $\hat{\theta}$. The result of the non-max suppression is the output `edge_map` of your function `hist_edge_detector`.

f) Outside the function `hist_edge_detector`, create a binary edge map by thresholding. Keep in mind that you are thresholding the $\chi^2_{\hat{\theta}}$ distances which may not be in the range of 0 to 1. Show your binary edge map in your report.

There are many parameters to play around with, *e.g.*, radius, number of bins, etc. The radius could be 3, 5, 10, 20, and the number of bins could be 16 and 32. Try different settings for each of the images in Figure 2 and report what you think is the best.

Problem 5. Texture Based Edges Create a function of a texture based edge detector:

```
function edge_img=text_edge_detector(image,threshold)
```

The necessary steps to follow are:

a) Create a filter bank consisting of:

(1) **18 oriented odd-symmetric filters:** $f_\theta(x, y)$,

$$f_\theta(x, y) = G'_{\sigma 1}(y)G_{\sigma 2}(x) \tag{2}$$

$$G_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{x^2}{2\sigma^2}} \tag{3}$$

$$G'_\sigma(x) = \frac{dG_\sigma(x)}{dx} = -\frac{1}{\sigma^3\sqrt{2\pi}}xe^{-\frac{x^2}{2\sigma^2}} \tag{4}$$

where θ represents the orientation of the filter, and σ_1, σ_2 are the scales needed for the two Gaussians, Equations 3 and 6. We will be using 3 combinations of scales σ_1, σ_2 and 6 orientations θ, which in total defines 18 filters for our filter bank.

- For the three combinations of scales σ_1 and σ_2, use $\sigma_1 = \sqrt{2}^k$ and $\sigma_2 = 3\sqrt{2}^k$, $k = 1, 2, 3$ so that the ratio $\frac{\sigma_2}{\sigma_1}$ can be kept to be 3.
- For the 6 orientations, use `imrotate` to rotate your filter with θ sampled from 0 to π.

(2) **18 oriented even-symmetric filters:** $g_\theta(x, y)$,

$$g_\theta(x, y) = G''_{\sigma 1}(y)G_{\sigma 2}(x) \tag{5}$$

$$G''_\sigma(x) = \frac{d^2G_\sigma(x)}{dx^2} = \frac{1}{\sigma^5\sqrt{2\pi}}x^2e^{-\frac{x^2}{2\sigma^2}} \tag{6}$$

Repeat the same procedure as in $f_\theta(x, y)$ to create 18 filters. Keep in mind you are taking the second derivatives of the Gaussian this time.

(3) **8 Laplacian of Gaussian (LOG) filters:** Four of them use $\sigma = \sqrt{2}^k$ while the other four use $\sigma = 3\sqrt{2}^k$, $k = 1, 2, 3$. You are allowed to use MATLAB's `fspecial` function with appropriate filter arguments.

(4) **4 Gaussian filters:** Use four different scales $\sigma = \sqrt{2}^k$, $k = 1, 2, 3, 4$ for the Gaussians.

A picture of all 48 filters in the filter bank is shown in Figure 4. Show your filter bank. The coloring does not matter.

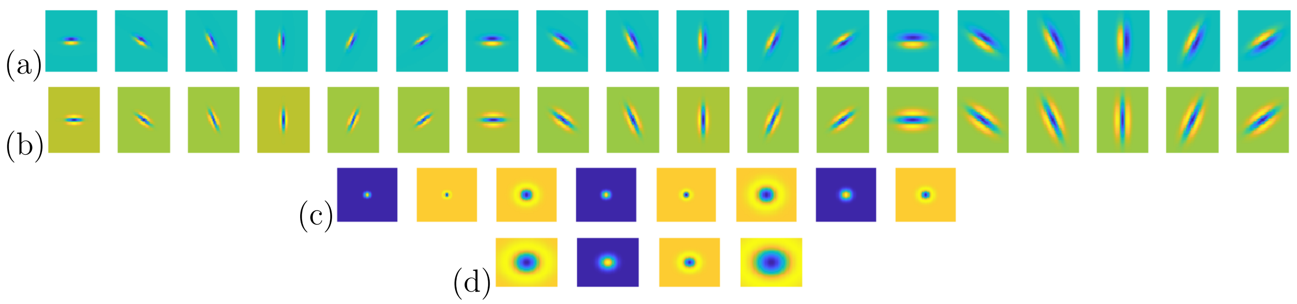


Figure 4: Filters in the filter bank. (a) 18 oriented odd-symmetric filters. (b) 18 oriented even-symmetric filters. (c) 8 Laplacian of Gaussian filter. (d) 4 Gaussian filters.

b) Convert the input image from color to grayscale. Also make sure to convert the class of the image from `uint8` to `double`.

c) Use `im2double` to rescale the image data to a range between 0 to 1.

d) Run all the filters of the filter bank using either MATLAB's function `imfilter` or your `my_conv` implemented in lab02, on images of Figure 2. Since there are 48 filters, we will have 48 filter responses for an image. In terms of implementation, a 2D filtered image is stacked into a 3D matrix with dimension $h \times w \times f$ where h and w are the height and width of the image, respectively, and f is the number of filters.

e) Loop over each filter response as an input to your histogram based edge detector implemented in problem 4, *i.e.*, `hist_edge_detector`. Store the output edge map and orientation map in two separate 3D matrices.

f) For 3D matrix of the edge map, do max over all strengths (magnitude), *i.e.*, the Chi-Square distances, and store in a map called `max_strength_map`. The corresponding orientation is also stored in a map called `angle_map`.

g) Finally, do non-max suppression as you did in the histogram based edge detector using `max_strength_map` and `angle_map`. The binary edge map is then obtained by thresholding. Show your binary edge map in the report.

Part II: Edge Linking

This part is mandatory for ENGN2605 students but optional for ENGN1610 students.

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

a) After doing non-maximal suppression, create two images by thresholding: I_h which is an edge map thresholded by T_h, and I_l which is an edge map thresholded by T_l.

b) For each edge point in I_h,

(1) Determine the edge orientation (Hint: an edge orientation is normal to the gradient direction) and round it to the closest of the 8 directions.

(2) Look at a pixel in its orientation. If the pixel has a gradient magnitude between T_l and T_h, add that point to I_h. Then, move to that pixel and repeat the same process. If the pixel has a gradient magnitude lower than T_l or higher than T_h (which is already in I_h), then stop the process.

Show the final binary edge map in your report.