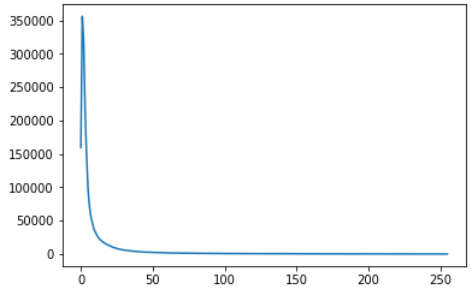
Digital Image Processing HW2 Report

- ## Probelm1

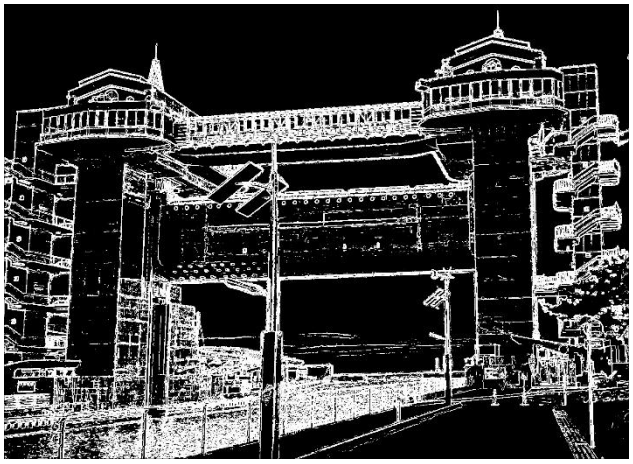
- (a)

  The input image(**sample1.jpg**) is the following:

(1) For first order edge detection, after trying several masks I chose Prewitt Mask. The threshold was chosen by observing the histogram as below:



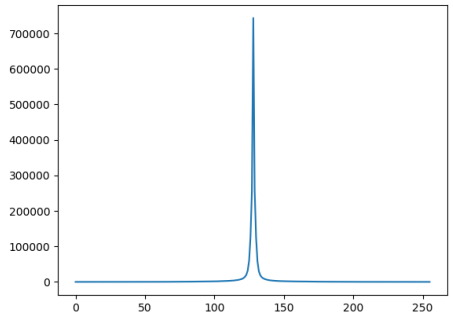
It seemed to have a significant fall on $x = 10$, so I chose threshold $= 10$. In other words, for the output function $G(j,k)$, which is the result by performing Prewitt Mask on input image, I set pixel (j,k) to be an edge if $G(j,k) > 10$, and not an edge otherwise. The following is the output result(**result1.jpg**)



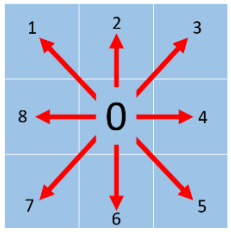
(2) I performed second order edge detection by the following steps:

Step1*:* Perform on input image the eight-neighbor mask H as Laplacian impulse response, where $\mathrm{H} = \frac{1}{8}\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$. Say the result $\mathrm{G}(\mathrm{j},\mathrm{k})$.

Step2*:* Observe the histogram and choose threshold = *5*, if $\mathrm{G}(\mathrm{j},\mathrm{k}) < 5$, set another function $\mathrm{G}'(\mathrm{j},\mathrm{k}) = 0$; otherwise, $\mathrm{G}'(\mathrm{j},\mathrm{k}) = \mathrm{G}(\mathrm{j},\mathrm{k})$. The following is the histogram.

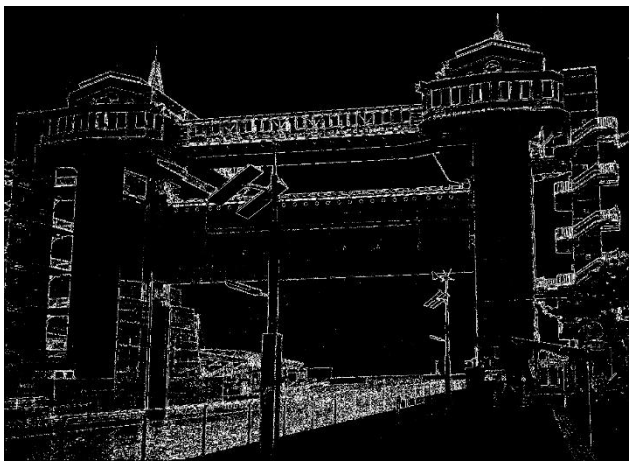


Step3*:* For each point, check the 8 directions whether there is a zero-crossing. If there exists a zero-crossing, set that point to be an edge.



8 directions around a zero point

The following is the result(**result2.jpg**).



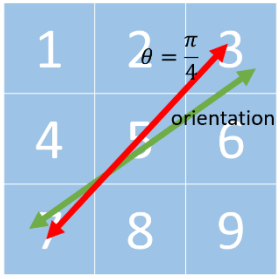
(3) The Canny edge detection was performed by the following 5 steps.

Step1: Reduce the noise. I performed a low-pass filter with mask H.

$$\mathrm{H} = \frac{1}{159}\begin{pmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{pmatrix}$$

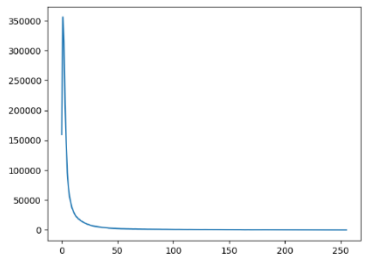
Step2: Compute the gradient magnitude. I used the method as in (1). Put the output function $\mathrm{G}(\mathrm{j},\mathrm{k})$

Step3: For each point (j,k), perform non-maximal suppression. Consider the orientation $(\cos\theta, \sin\theta)$, I selected from the 8 direction (i.e. $\theta = 0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi, \frac{5\pi}{4}, \frac{3\pi}{2}, \frac{7\pi}{4}$) the closest to the orientation, and checked whether $\mathrm{G}(\mathrm{j},\mathrm{k})$ is a local maximum. If do, set function $\mathrm{G_N}(\mathrm{j},\mathrm{k}) = \mathrm{G}(\mathrm{j},\mathrm{k})$; otherwise, $\mathrm{G_N}(\mathrm{j},\mathrm{k}) = 0$

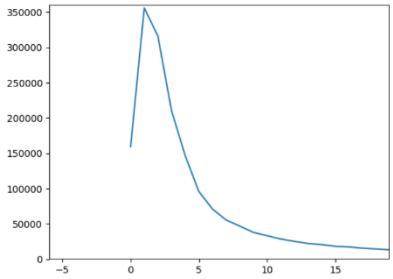


Example: for orientation like above green arrow, choose direction $\theta = \pi/4$
One then check if value in the center(position 5) > value at position 3 and 7

Step4: Set threshold by observing the histogram on $\mathrm{G_N}$. The histogram is shown below. I set high-threshold $\mathrm{T_H} = 15$, and low-threshold $\mathrm{T_L} = 3$. For pixel (i,j), set it to be an edge if $\mathrm{G_N}(\mathrm{i},\mathrm{j}) > \mathrm{T_H}$, not an edge if $\mathrm{G_N}(\mathrm{i},\mathrm{j}) < \mathrm{T_L}$, and an candidate if $\mathrm{T_L} < \mathrm{G_N}(\mathrm{i},\mathrm{j}) < \mathrm{T_H}$.



Histogram of $\mathrm{G_N}$



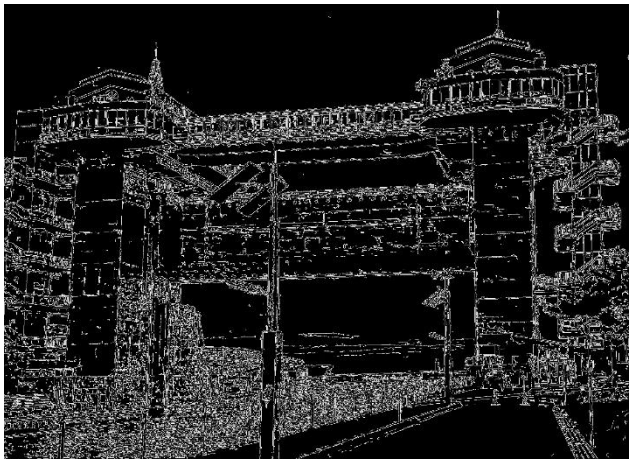
Histogram of $\mathrm{G_N}$ (Zoom in)

Step5*:* If a candidate pixel is connected to an edge, or connected to a candidate pixel which is connected to and edge, set it to be an edge.
The following is my result(**result3.jpg**).

(4) I used the unsharp masking to do edge crispening. The low-pass filter mask I used is the following:

$$\mathrm{H} = \frac{1}{159}\begin{pmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{pmatrix}$$

Name the output for applying this mask $\mathrm{F_L}(\mathrm{i},\mathrm{j})$. Let the original image be $\mathrm{F}(\mathrm{i},\mathrm{j})$. The unsharp masking method was applied by defining the output image $\mathrm{G}(\mathrm{j},\mathrm{k})$ as $\mathrm{G}(\mathrm{j},\mathrm{k}) = \frac{\mathrm{c}}{2\mathrm{c}-1}\mathrm{F}(\mathrm{j},\mathrm{k}) - \frac{1-\mathrm{c}}{2\mathrm{c}-1}\mathrm{F_L}(\mathrm{j},\mathrm{k})$, where c is some constant parameter.
By testing some samples, I found $\mathrm{c} \leq 0.7$ may add too much noise to image, but it helps little when using too large c. Hence I chose $\mathrm{c} = 0.7$.
Then the edge map generated by Canny edge detection(all parameters are set like (3)) is as the follows(**result5.jpg**).



(5)

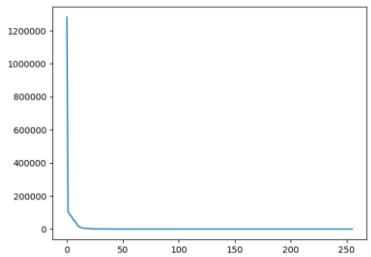
result1.jpg shows the result of first-order edge detection. It leads to an edge map with thick lines. Second-order edge detection(**result2.jpg**) result in an edge map with lines much thinner. However, second order edge detection cannot detect some lines that are too thin, since there is no zero-crossing there. As a result, there are many discontinuities on **result2.jpg**. Canny edge detection, as in **result3.jpg,** behaves as a balance between the previous two. It contains thin and clear edges, while most of them are connected.

result5.jpg is the result of performing Canny edge detection on the edge-crispening image. It contains much more details than **result3.jpg,** the version of Canny edge detection on original image. Some shadow and texture are shown, and more edges are connected together. However, part of them is not much worth-doing. For instance, the shadow on the holes on the bridge should not have to appear edges inside(or it may be noise, I'm not sure). In addition, it is to be observed that some noise occur after edge-crispening. This is the result that edge-crispening actually enhance high-frequency part of an image, while edges and noise are both of high-frequency.

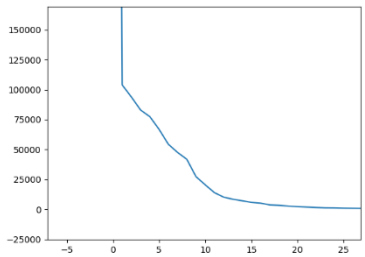
- (b)

➢ **First Try**:

At first I noticed that the image has a problem of little contrast. Hence I performed a histogram equalization to uniformly extend its intensity distribution. Next I perform Canny edge detection as discussed in (a), where the only difference is that in ***Step4,*** by observing the histogram and the output of $G_N(i,j)$, I set the high-threshold $T_H = 30$ and low-threshold $T_L = 3$.



Histogram of G_N



Histogram of G_N (Zoom in)

The result is as follows:

Original Image:

Output Image:

The first problem of the result is obvious –the noise(that's also why I set a high T_H). This is due to the histogram equalization. Although I had tried doing Canny edge rection directly without any preprocessing, the result is worse, as many edges are not shown out.

The next problem of the result is that several edges did not appear, while some are disconnection of appeared edges, and some totally disappear. This is because of the noice – I had to set a high threshold, and because of the low contrast of original image. Even though histogram equalization enhances the contrast, some edges are still difficult to detect. One obvious example is the line on the middle of the image(in front of the roof), it totally disappears.

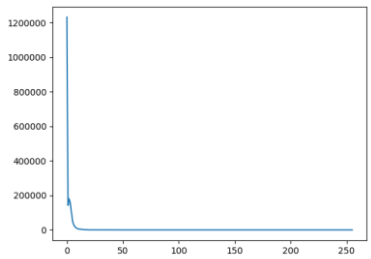
- **Second Try:** (Referenced from classmate 賴昭蓉 B07502165)

  Next I tried a four-powered transfer function to enhance the image instead of using histogram equalization.

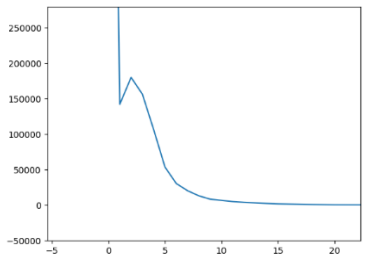
$$f(x) = 255\left(\frac{x}{255}\right)^4$$

The result image of transfer function is as follows:

In the next I also performed the Canny edge detection. In ***Step4,*** by observing the histogram of $G_N(i,j)$, I chose the threshold $T_H = 10,\ T_L = 2$



Histogram of G_N

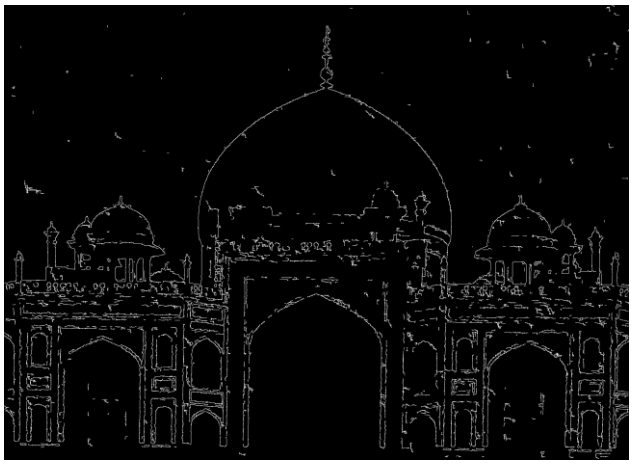


Histogram of G_N (Zoom in)

The result is as follows:

Original Image:

Output Image:



One can see this much better than the previous **First-Try** result. There are still some noise-caused edges in this result, and some edges are still not shown, but many of these problems are improved. Looking at the histogram of G_N in these two tries, one sees a four-powered transfer function help split the light area and the dark area. While a histogram-equalizationed image has a more average-weighted distribution of intensity, it may lead a difficulty in detecting edges, e.g. the sky over the building is considered an edge.

- Problem2
- (a)

  At first glance I noticed that there exists a rotation on **sample4.jpg.** Therefore I pergormed a rotation on **sample3.jpg** with rotating axis located on the center of the image. After trying several possible angle θ, I found $\theta = 1.3$ best fits **sample4.jpg.**

  Next I start processing the translation part. The result of only rotating **sample3.jpg** looks like that the man is a little higher and more right than **sample4.jpg.** Thus I try translate the image with translation vector $\mathbf{t} = (t_x, t_Y)$. After some trials and fixes, I got that $\mathbf{t} = (-200, -100)$ fits well.

  Finally the scaling part is done by scaling the image with respect to its center with magnitude $\mathbf{m}$**.** As before, several trials are done, and $\mathbf{m} = 1.4$ behaves very well. As the following result, **result6.jpg,** it looks very close to **sample4.jpg.**

  Original Image(**sample3.jpg**):

  Output Image(**result6.jpg**):

- (b)

  First I use twirl-method I found on the Internet, as the following transformation.

  ※reference:

  https://web.cs.wpi.edu/~emmanuel/courses/cs545/S14/slides/lecture11.pdf

  $$dx = x - x_c, dy = y - y_c$$

  $$r = \sqrt{(dx^2 + dy^2)}, r_{max} = \sqrt{(height - y_c)^2 + (width - x_c)^2}$$

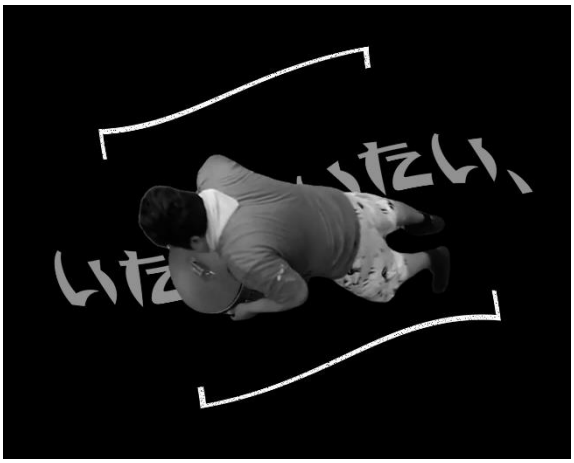
  $$\beta = \arctan(d_y/d_x) + \alpha \times (1 - r/r_{max})$$

  $$u = x_c + r \times \cos(\beta)$$

  $$v = y_c + r \times \sin(\beta)$$

  - (x_c, y_c) is the center coordinate of the image(Cartesian).
  - $(height, width)$ is the shape of the image.
  - $(x, y)/(u, v)$ is the output/input image coordinate
  - α is the angle parameter of rotating, where I set -0.7.

  The result seem to need some improvement. Like below:

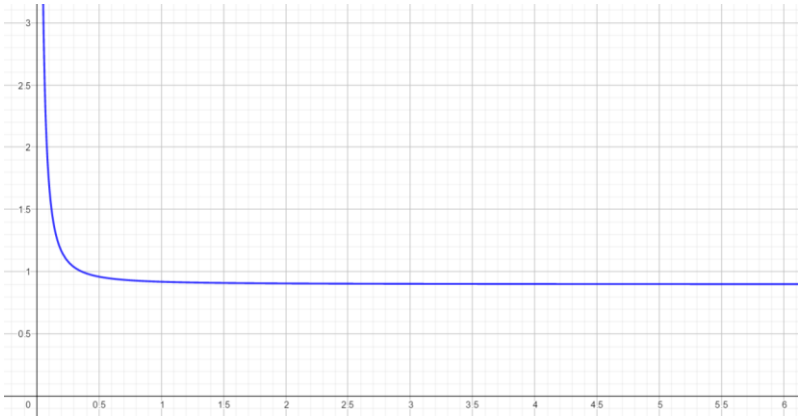
  

  Next I tried to scale the image with a magnitude proportional to the distance of the pixel to the center.

  Given a pixel, define ρ to be the ratio of the distance of the pixel to the center and the distance of the corner to the center (as $\frac{r}{r_{max}}$ defined in above). Notice that $0 \leq \rho \leq 1$, and the closer a pixel is positioned to the center, the smaller ρ is.

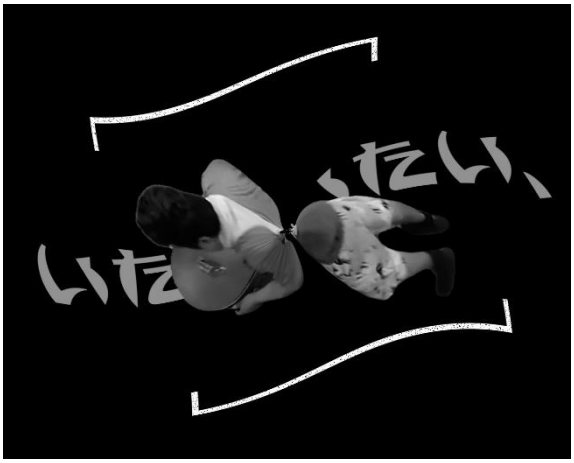
  My goal is to find a function of ρ, representing the magnitude of scaling, such that it is decreasing(so that the pixel closer to the center is scaled more), nonnegative, and for $\rho \to 0$, there is severe difference when ρ varies. To speak specifically, such function is positive and has negative first-order and positive second-order derivative.

I chose the following one:

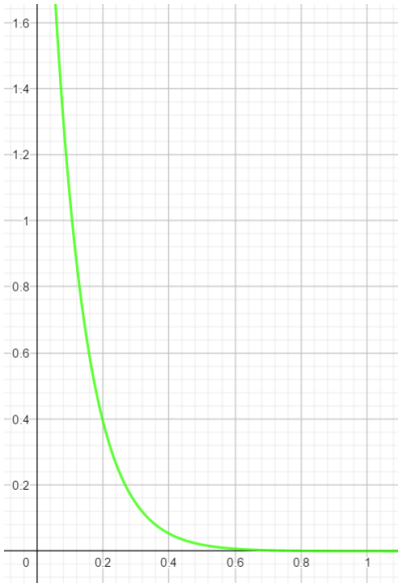


$$f(\rho) = 0.02\frac{1}{\rho^{1.6}} + 0.9, \rho \leq 1$$

The power(1.6) and the coefficient(0.02) was chosen to make the variance of the scaling magnitude as ρ varies fit the sample. The constant(0.9) was chosen to make the whole image in an appropriate scale, not too large or too small. After using such $f(\rho)$ to scale the image, the following is the result.



In the last I gave some rotation. As above, I was searching a function $g(\rho)$, representing the rotating angle, such that it is increasing(as I found the rotation in the sample is larger as ρ grows), nonnegative, and has little difference when ρ varies around a large value(there is small difference of rotation as ρ grows). I chose the following one:



$$g(\rho) = 2.9e^{-10\rho}$$

※This function is referenced by classmate 賴昭蓉, B07502165

The coefficient is chosen to make the function fit the sample.

After applying rotation with angle $g(\rho)$, the following is my result(**result7.jpg**):

To my regret, though the result fit well, there is still some room for improvement. Many parameters are chosen without rigorous reasoning or calculation. However, the process in altering function, parameters and algorithms helped me learn much in the field of digital image processing.