

Homework 1

M1522.001800 Computer Vision (2019 Spring)

2014-10469 InSeoung Han

Date: March 18 Monday

[Important] Reference to work of others. You can refer to another person's key idea in writing the source code, but in this case you must leave a reference in the writeup. If you take someone else's idea and leave a reference, you will only get 0 points in the "Implementation" part of that problem in the scoring process. However, please note that if you do not mention the reference even though you have brought in someone else's idea, you will get 0 points for all the problems in that homework.

1 Composing Filters [5 pts]

\mathcal{G} and \mathcal{E} are linear kernels, that is, they can be implemented with convolution and we can use the properties of convolutions(commutative, associative). Let the image be denoted by \mathcal{I} , then we can denote applying \mathcal{G} followed by \mathcal{E} on \mathcal{I} by using convolution, $\mathcal{G} * \mathcal{E} * \mathcal{I}$. And by below procedures,
(assume that each kernel size are the same)

$$\mathcal{G} * (\mathcal{E} * \mathcal{I}) = (\mathcal{G} * \mathcal{E}) * \mathcal{I} \text{ (by associative rule)} \quad (1)$$

$$= (\mathcal{E} * \mathcal{G}) * \mathcal{I} \text{ (by commutative rule)} \quad (2)$$

$$= \mathcal{E} * (\mathcal{G} * \mathcal{I}) \text{ (by associative rule)} \quad (3)$$

applying order between \mathcal{G} and \mathcal{E} doesn't matter. they will produce the same result. On the other hand, \mathcal{M} is a non-linear filter, so that we can't implement the filter with convolution, and the result will depend on applying order between \mathcal{M} and \mathcal{E} .

2 Identity for Convolution [10 pts]

Dirac Delta function δ is a function that satisfies below equation.

$$\delta(x) = \begin{cases} \infty & x = 0 \\ 0 & x \neq 0 \end{cases} \quad \int_{-\infty}^{\infty} \delta(x) dx = 1$$

Then, convolution between δ and some function f over t is denoted by

$$(f * \delta)(t) = \int_{-\infty}^{\infty} f(\tau)\delta(t - \tau)d\tau \quad (4)$$

$$= \int_{-\infty}^{\infty} f(t - \tau)\delta(\tau)d\tau \quad (5)$$

$$= \int_{-\infty}^{\infty} f(t)\delta(\tau)d\tau \quad (6)$$

$$= f(t) \int_{-\infty}^{\infty} \delta(\tau)d\tau \quad (7)$$

$$= f(t) \quad (8)$$

(5) hold because of the commutative property of convolution. By definition of dirac delta function $f(t - \tau)\delta(\tau)$ term would have meaning only when $\tau = 0$. Therefore, we can change $f(t - \tau)$ to $f(t)$. Finally, by definition of dirac delta function again, the term after $f(t)$ is going to be 1.

Now we can conclude that dirac delta function is the identity of convolution.

3 Decomposing a Steerable Filter [10 pts]

Definition of gaussian kernel for one-dimension is below.

$$\mathcal{G}_t = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{t^2}{2\sigma^2})$$

Then, we can decompose 2-d gaussian kernel into two successive convolution of 1-d gaussian kernel. For some function f

$$(\mathcal{G} * f)(t, s) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(t - x, s - y)\mathcal{G}(x, y)dx dy \quad (9)$$

$$= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(t - x, s - y) \frac{1}{2\pi\sigma^2} \exp(-\frac{x^2 + y^2}{2\sigma^2}) dx dy \quad (10)$$

$$= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(t - x, s - y) \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{x^2}{2\sigma^2}) \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{y^2}{2\sigma^2}) dx dy \quad (11)$$

$$= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(t - x, s - y) \mathcal{G}_x(x) \mathcal{G}_y(y) dx dy \quad (12)$$

$$= \int_{-\infty}^{\infty} \mathcal{G}_y(y) \left(\int_{-\infty}^{\infty} f(t - x, s - y) \mathcal{G}_x(x) dx \right) dy \quad (13)$$

$$= (\mathcal{G}_y * (\mathcal{G}_x * f(t, s))) \quad (14)$$

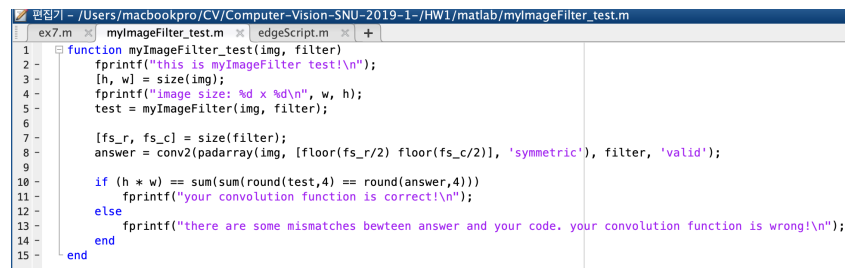
Let length of range of t, s be n , so that filter size is $n \times n$. Then, convolving with \mathcal{G} in a single step takes $O(n^2)$ computational steps. On the other hand, convolving with \mathcal{G}_x and \mathcal{G}_y in two steps takes only $O(2n)$ computational steps, because 1-dimensional gaussian kernel has n filter size. In conclusion, the latter is better in terms of computational efficiency

4 Convolution theorem [10 pts]

$$\begin{aligned} g(x) &= \int_{-\infty}^{\infty} G(w) \exp(2\pi x w) dw \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(\tau) H(w - \tau) \exp(2\pi x w) d\tau dw && \text{Definition of Inverse Fourier Transform} \\ &= \int_{-\infty}^{\infty} F(\tau) \int_{-\infty}^{\infty} H(w - \tau) \exp(2\pi x w) dw d\tau && \text{Changing the order of integration} \\ &= \int_{-\infty}^{\infty} F(\tau) \int_{-\infty}^{\infty} H(v) \exp(2\pi x (v + \tau)) dv d\tau && \text{By } v = w - \tau \\ &= \int_{-\infty}^{\infty} F(\tau) \exp(2\pi x \tau) d\tau \int_{-\infty}^{\infty} H(v) \exp(2\pi x v) dv && \text{Separating terms for } \tau \text{ and } v \\ &= f(x) h(x) \end{aligned}$$

5 Edge Detection [65 pts]

1. Convolution [10 pts]



```
1 function myImageFilter_test(img, filter)
2     fprintf('this is myImageFilter test!\n');
3     [h, w] = size(img);
4     fprintf('image size: %d x %d\n', w, h);
5     test = myImageFilter(img, filter);
6
7     [fs_r, fs_c] = size(filter);
8     answer = conv2(padarray(img, [floor(fs_r/2) floor(fs_c/2)]), 'symmetric'), filter, 'valid');
9
10    if (h * w) == sum(sum(round(test,4) == round(answer,4)))
11        fprintf('your convolution function is correct!\n');
12    else
13        fprintf('there are some mismatches bewteen answer and your code. your convolution function is wrong!\n');
14    end
15 end
```

Figure 1: Convolution test code

```
gaufilter = [1 4 7 4 1; 4 16 26 16 4; 7 26 41 26 7; 4 16 26 16 4; 1 4 7 4 1] / 273;
myImageFilter_test(img, gaufilter);
```

Figure 2: Convolution test input

```
this is myImageFilter test!
image size: 720 x 349
your convolution function is correct!
>>
```

Figure 3: Convolution test result

2. Laplacian Filter [10 pts]

I applied (5x5) gaussian kernel after convolving a image with Laplacian Filter. I used gaussian kernel which is same as what I used in previous convolution test(Figure2).

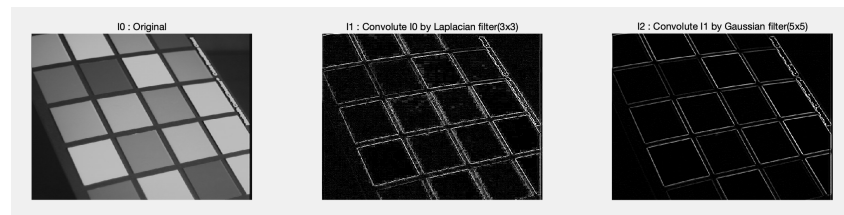


Figure 4: denoising test using function imshow()

3. Differential of Gaussian [10 pts]

4. Sobel Filter [15 pts]

I assumed that the type of elements of I_0 is degree, but not radian. (I calculated I_0 by using atan2)

5. The Non-maximum Suppression [15 pts]

To suppress the magnitude of a pixel(i, j), I got two neighbor pixels on the direction of I_0 value($I_0(i, j)$) of the pixel, dividing range from -90 degree to 90 degree into the five regions. each I_0 value maps one of the five regions and is translated to vector. To get the

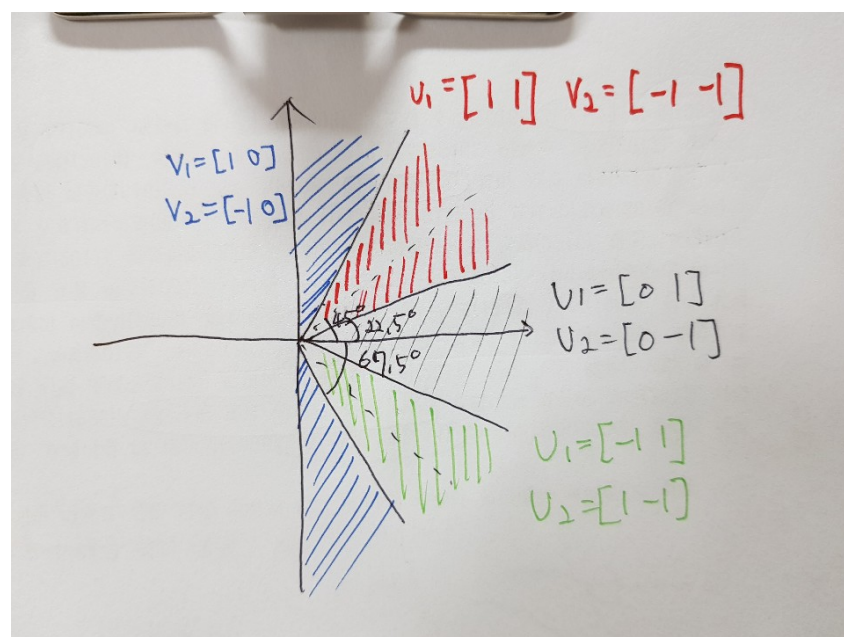


Figure 5: concept used in function orient vector()

two pixels, I simply added the vector with (i, j) and used max, min functions to handle border problem. Finally, I compared magnitude $I_m(i, j)$ with the magnitude of the two pixels

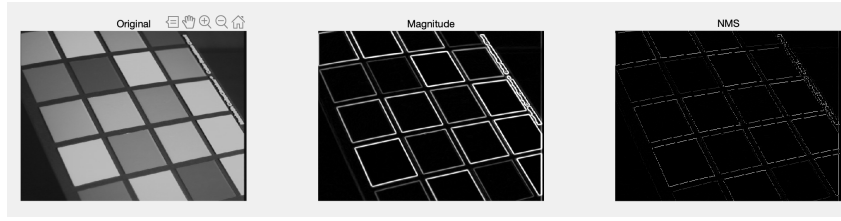


Figure 6: NMS test

6. The Double Thresholding [5 pts]

To implement double thresholding, firstly I defined zero matrix(\mathbf{I}_{edge} which size is same as \mathbf{I}_n , and traversed each pixel($\mathbf{I}_n(i, j)$), setting $\mathbf{I}_{\text{edge}}(i, j)$ value to 1 if the pixel value is larger than **lowThreshold** and one of the values of its neighbor pixels including itself is bigger than **highThreshold**

I tested this implementation with handling **lowThreshold** and **highThreshold** in 2 ways.

1. fix **lowThreshold** and change **highThreshold**

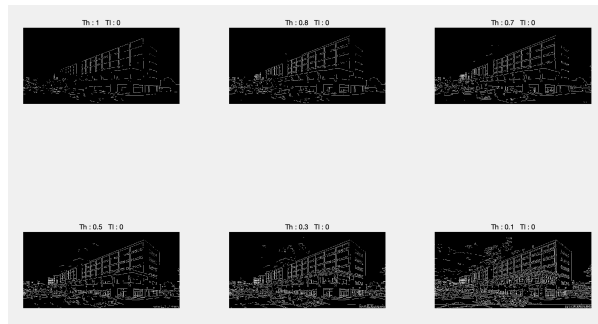


Figure 7: Threshold Test1

2. fix **highThreshold** and change **lowThreshold**



Figure 8: Threshold Test2

Through the above method, I could find that I_{edge} is much more sensitive to **highThreshold** than **lowThreshold** and **lowThreshold** doesn't make any difference in the result. To make sure that **lowThreshold** has no effect on result, I draw 3d-graph by using below code. [Figure 9, 10]

```
Th = linspace(0, 1, 100);
TL = linspace(0, 1, 100);
[X, Y] = meshgrid(Th, TL);
[h, w] = size(X);
F = zeros(h, w);
for i=1:h
    for j=1:w
        if(X(i,j) > Y(i,j))
            F(i,j) = sum(sum(myDoubleThresholding(In, X(i,j), Y(i,j))));
        end
    end
end
surf(X,Y,F);
xlabel('Th');ylabel('TL');zlabel('sum of img values');
```

Figure 9: Threshold Test3 : Code

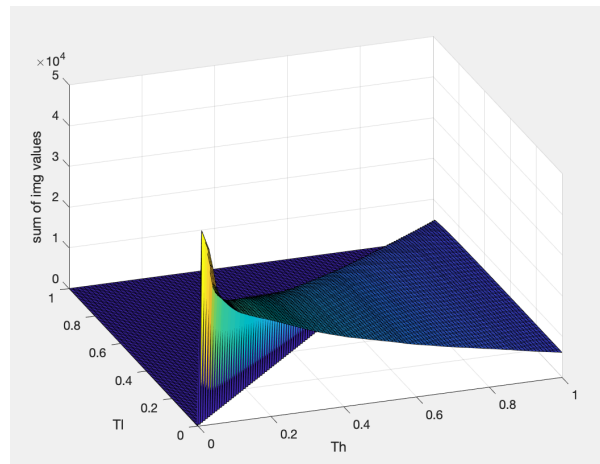


Figure 10: Threshold Test4, Z-axis means $\text{sum}(\text{sum}(I_{\text{edge}}))$

Again, I could find that **lowThreshold** doesn't make any difference. I wondered why this happen, and figured out why it is by setting the value of weak edge to 0.5. [Figure 11]



Figure 11: Threshold Test5

In conclusion, I found that the reason why weak edges do not make any big difference is that mostly they are not connected to strong edges.