Assignment 3 Report

고려대학교 컴퓨터학과
정경륜

1. assign_3_skeleton.m 코드 설명

$$g = \frac{1}{1 + |\nabla \hat{I}|^p}$$

먼저 g를 계산하기 위해서 $\hat{I}$을 구해주어야 한다. $\hat{I}$은 input image인 I에 gaussian filter를 적용시킨 것이다. 이를 구하기 위해선 아래와 같이 matlab 내장 함수인 imfilter를 사용하여 구할 수도 있고 혹은 직접 구현을 할 수도 있다.

```matlab
% ToDO ------------------------
% --------------------------------------------------
% Get smoothed version of the input Image
% --------------------------------------------------
% Use matlab function
 h = fspecial('gaussian',5,1.0);
 gaussian_blurred = imfilter(Img,h,'symmetric');
 I = gaussian_blurred;
```

직접 구현하게 된다면 assignment2에서 수행했던 LPF를 가져와서 아래와 같이 사용하면 된다. 해당 과제에서 수행했던 filter의 종류는 butter worth filter이기 때문에 gaussian filter로 활용하고 싶다면 order를 1에 가깝게 설정해야 한다.

```matlab
% ------------------------------------------------
% Use LPF from assignment2
% Get size
% dimX = size(Img,1);
% dimY = size(Img,2);
% % Padding
% PQ = size(Img)*2;
% F = fft2(Img,PQ(1),PQ(2));
% F = fftshift(F);
% % figure,imshow(log(1+abs((F))), []);
% G = F;
% D = zeros(PQ(1), PQ(2));
% H = zeros(PQ(1), PQ(2));
% centerP = PQ(1)/2;
% centerQ = PQ(2)/2;
%
% D0=30;
% n=1;
%
% for x=1:PQ(1)
%     for y=1:PQ(2)
%         D(x,y)= sqrt((x-centerP)^2 + (y-centerQ)^2);
%         H(x,y)=1/(1+(D(x,y)/D0)^(2*n));
%     end
% end
% % Low pass filtering
% G= G.*H;
 % % figure,imshow(log(1+abs((G))), []);
 % G = ifftshift(G);
 % I = ifft2(G);
 % I = I(1:dimX, 1:dimY);
 % I = real(I);
 % figure,imshow(I, []);
 % ------------------------------------------------
```

$$g = \frac{1}{1 + |\nabla \hat{I}|^p}$$

이제 앞에서 구한 $\hat{I}$의 미분을 구해야한다. 미분은 forward, backward, prewitt, sober filter 등 여러가지 방법으로 구할 수 있으나 이번 과제에선 sobel filter를 사용하였다. Sobel filter를 사용한 이유는 가로, 세로 뿐만 아니라 대각선 방향의 edge도 잘 탐색하기 위해서 이다. 이는 다음과 같이 구현하였다.

```
% Derivative code
% Get size
dimX = size(Img,1);
dimY = size(Img,2);
% Initalize
dx=zeros(dimX,dimY);
dy=zeros(dimX,dimY);
% Using sobel filter
for x=2:dimX-1
    for y=2:dimY-1
        dx(x,y) = (I(x+1, y-1) + 2*I(x+1, y) + I(x+1, y+1) - (I(x-1, y-1) + 2*I(x-1, y) + I(x-1, y+1)))./9;
        dy(x,y) = ((I(x-1, y+1) + 2*I(x, y+1) + I(x+1, y+1)) - (I(x-1, y-1) + 2*I(x, y-1) + I(x+1, y-1)))./9;
    end
end
```

이번 구현에선 sobel filter를 따로 두고 convolution을 진행하는 것이 아닌, 이중 for 문을 돌면서 각 (x, y) 좌표값 별로 sobel filter를 적용했을 때 사용하는 index를 가져와 해당하는 가중치를 곱해 값을 구하였다.

```
% Calculate magnitude
magnitude = sqrt(dx.^2 + dy.^2);
p=2;
g = 1./(1+magnitude.^p);
```

그 다음으론 magnitude 인 $\left|\nabla \hat{I}\right|$를 다음과 같이 구하였다. 이후 $g = \dfrac{1}{1 + \left|\nabla \hat{I}\right|^p}$ 수식에 맞춰서 g를 구하였다.

2. levelset_update.m 코드 설명 (간단한 구현)

$$\frac{\partial u}{\partial t} = g(I)|\nabla u|\text{div}\left(\frac{\nabla u}{|\nabla u|}\right) + cg(I)|\nabla u| = g(I)(c+\kappa)|\nabla u|,$$

Level set update를 위해선 왼쪽의 수식을 적용해야 한다. Expanding term인 c는 사전에 미리 설정하기 때문에 해당 함수에선 $\left|\nabla u\right|$와 $div(\dfrac{\nabla u}{\left|\nabla u\right|})$ 인 k를 먼저 구해야 한다. 각 term을 구하기에 앞서 미분 함수가 반복적으로 나오기 때문에 아래와 같이 미분 함수인 sobel_filter를 작성해주었다. 해당 함수는 위에서 사용했던 sobel filter와 같다.

```
% Sobel filter for derivative
function [dx, dy] = sobel_filter(input)
% Get size
dimX = size(input,1);
dimY = size(input,2);
% Initalize
dx = zeros(dimX,dimY);
dy = zeros(dimX,dimY);
% Calculate
for x=2:dimX-1
    for y=2:dimY-1
        dx(x,y) = (input(x+1, y-1) + 2*input(x+1, y) + input(x+1, y+1) - (input(x-1, y-1) + 2*input(x-1, y) + input(x-1, y+1)))./9;
        dy(x,y) = ((input(x-1, y+1) + 2*input(x, y+1) + input(x+1, y+1)) - (input(x-1, y-1) + 2*input(x, y-1) + input(x+1, y-1)))./9;
    end
end
```

```
% Gradient of phi
[phi_dx, phi_dy] = sobel_filter(phi_in);
dPhi = sqrt(phi_dx.^2 + phi_dy.^2); % mag(grad(phi))
```

먼저 $\left|\nabla u\right|$를 왼쪽의 코드를 통해서 구현하였다. 해당 코드에서 u는 phi_in과 같다.

다음으론 k를 구하기 위해서 divergence 안에 들어갈 normalize 값을 구해야 한다. 이를 위해서 위에서 구한 ∇u 를 $|\nabla u|$ 로 나누어야 하는데 이때 $|\nabla u|$ 가 0일 경우 분모가 0이 되어 오류가 발생한다. 따라서 epsilon인 eps 변수를 설정해 분모에 더해주어 오류를 방지하였다. Eps는 주로 $1e-4$에서 $1e-8$를 사용하기에 이번 과제에선 $1e-8$ 로 설정하였다. 이제 divergence를 구해야 한다. 앞에서 구한 normalize 값을 미분하고, 미분 후 나온 dx, dy 값을 더하여 divergence인 k (kappa)를 구하였다.

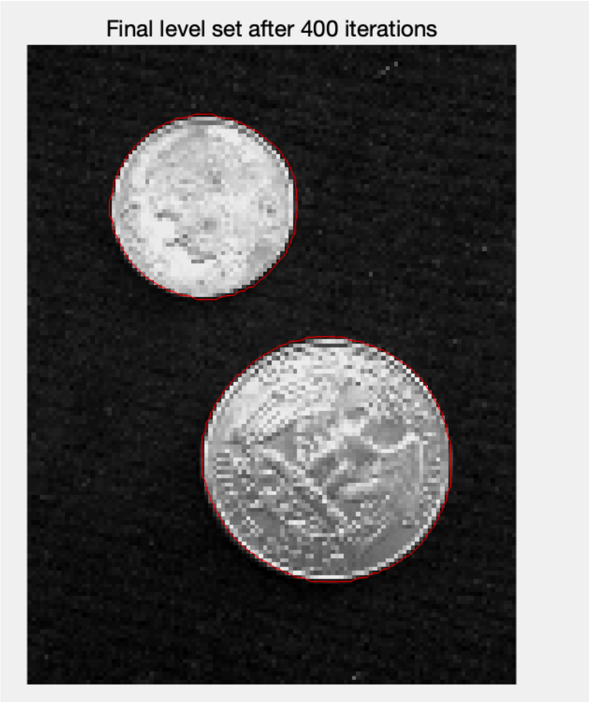
```
% Use eps(epsilon) to prevent division by zero
% Generally use 1e-4 ~ 1e-8 at deep learning. So I use 1e-8 in this assignment.
eps = 1e-8;

% -----------------------------------------------------------------
% 간단한 구현
% -----------------------------------------------------------------

normalize_phi = (phi_dx + phi_dy)./(dPhi+eps);
[divergence_x, divergence_y] = sobel_filter(normalize_phi);
kappa =  divergence_x + divergence_y; % curvature
```

```
smoothness = g.*kappa.*dPhi;
expand = c*g.*dPhi;

phi_out = phi_out + timestep*(expand + smoothness);
```

Skeleton code에서 제공한 code를 통해 왼쪽의 수식과 같이 update를 진행해 준다.

3. 구현 결과 (간단한 구현)

먼저 skeleton code의 default 값인 dt = 0.8, c = 1.0를 사용하여 기본 제공 이미지에 적용한 결과이다. Edge를 거의 잘 탐색하긴 했지만 그림의 아랫 동전에서 아랫 부분 edge가 살짝 잘못 탐지한 것을 볼 수 있다.



c = 0.8로 낮춘 결과 왼쪽 그림에서 볼 수 있듯이 edge를 잘 탐색한 것을 볼 수 있다.

Test 이미지로 4개의 달 사진을 가져와서 실험해보았다. 해당 사진에선 dt = 0.8, c = 0.6을 사용했을 때 edge를 가장 잘 탐색하였다.

4. 추가적인 levelset_update.m 코드 설명 (정확한 구현)

$$
\begin{aligned}
\frac{\partial u}{\partial t} &= g(I)|\nabla u| \boxed{\operatorname{div}\left(\frac{\nabla u}{|\nabla u|}\right)} + c g(I)|\nabla u| \\
&= g(I)(c+\kappa)|\nabla u|,
\end{aligned}
$$

The curvature of a level set is given by $\kappa = (u_{xx}u_y^2 - 2u_x u_y u_{xy} + u_{yy}u_x^2)/|\nabla u|^3$.

기존에 구현했던 divergence는 간단하게 구현 한 것으로 div term 안의 값을 x와 y에 대해서 미분한 후 더해주어 값을 구했다. 그러나 위의 수식에서 볼 수 있듯이 divergence를 좀 더 정확하게 구할려면 위의 수식을 적용해야 한다. 좀 더 정확한 second-order 미분 값을 구하기 위해 함수를 따로 작성하였고 그 코드는 아래와 같다.

```
function [dxx, dyy] = second_derivative(input)
% Get size
dimX = size(input,1);
dimY = size(input,2);
% Initalize
dxx = zeros(dimX,dimY);
dyy = zeros(dimX,dimY);
% Calculate
for x=2:dimX-1
    for y=2:dimY-1
        dxx(x,y) = (input(x+1, y) - 2*input(x, y) + input(x-1, y))/4;
        dyy(x,y) = (input(x, y+1) - 2*input(x, y) + input(x, y-1))/4;
    end
end

function [dxy] = cross_second_derivative(input)
% Get size
dimX = size(input,1);
dimY = size(input,2);
% Initalize
dxy = zeros(dimX,dimY);
% Calculate
for x=2:dimX-1
    for y=2:dimY-1
        dxy(x,y) = (input(x+1, y-1) - input(x+1, y+1) - input(x-1, y-1) + input(x-1, y+1))/4;
    end
```

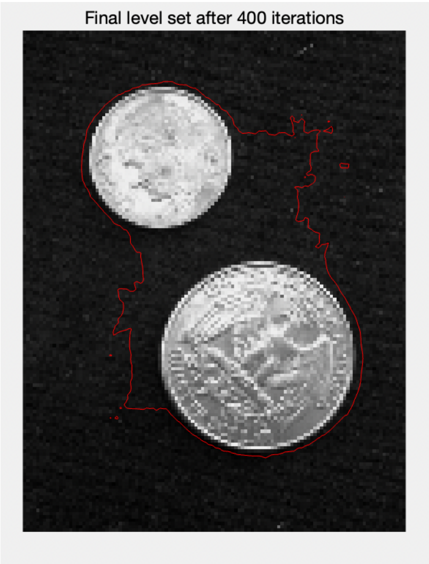
추가로 구현한 함수는 총 두 가지로 dxx, dyy를 구하는 second_derivative와 cross partial derivative인 dxy를 구하는 cross_second_derivative 함수 이다. 미분 시 border 부분처리는 가장자리 값과 같은 값을 padding 처리하였다고 가정한다면 변화량이 0이기에 따로 계산해주지 않았다. (초기값이 0이기 때문이다.) 위의 함수를 가지고

```
% Gradient of phi
[phi_dx, phi_dy] = sobel_filter(phi_in);
dPhi = sqrt(phi_dx.^2 + phi_dy.^2); % mag(grad(phi))

% Use eps(epsilon) to prevent division by zero
% Generally use 1e-4 ~ 1e-8 at deep learning. So I use 1e-8 in this assignment.
eps = 1e-8;

% ---------------------------------------------------------------------
% 좀 더 정확한 구현
% ---------------------------------------------------------------------

[phi_dxx, phi_dyy] = second_derivative(phi_in);
phi_dxy = cross_second_derivative(phi_in);
div = (phi_dxx.*(phi_dy.^2) - phi_dx.*phi_dy.*phi_dxy + phi_dyy.*(phi_dy.^2)) ./ (dPhi.^3 + eps);
kappa = div; % curvature

```

divergence의 정확한 값을 구하게 되면 다음과 같다.
위에 나왔던 수식을 그대로 옮겨 작성하였다. 일차 미분은 앞서 작성한 sobel filter를 그대로 활용하였고 이차 미분 시에만 새로 작성한 함수를 사용하였다.

5. Divergence의 정확한 구현 결과

C = 0.2

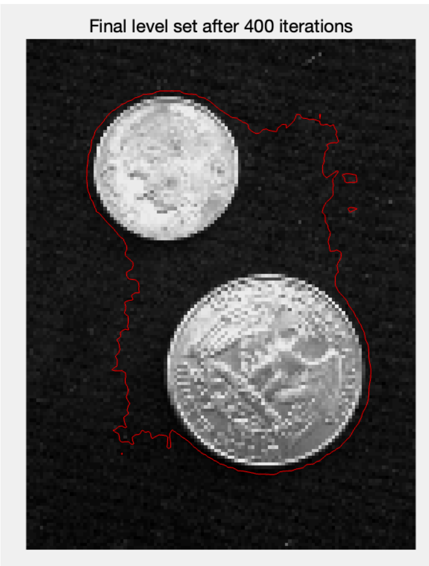


c = 0.5

c = 1

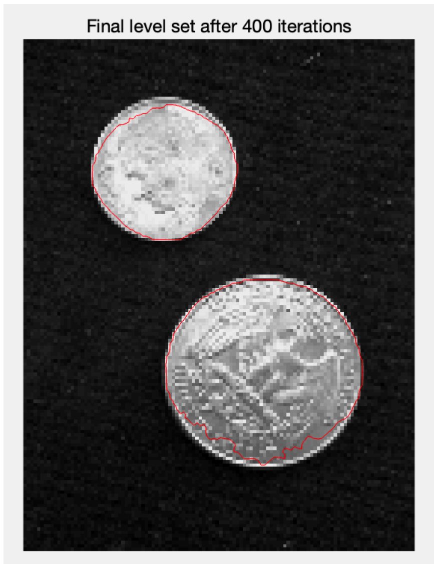
우선 기본 제공 이미지를 dt = 0.55로 고정하고, c의 값만 각각 0.2, 0.5, 1로 설정한 실험 결과이다. c의 값이 커질 수록 반복 당 expand의 스케일이 더 커지기 때문에 수렴도 더 빨리한다. 그래서 c가 너무 작으면 수렴하지 못 하고 반면에 c가 너무 크면 edge를 뚫고 들어간다. 따라서 적절한 c의 값을 설정하는 것이 중요하다.

dt = 0.2



dt = 0.55

dt = 0.8



c를 0.5로 고정하고, dt의 값만 각각 0.2, 0.55, 0.8로 설정한 실험 결과이다. 마찬가지로 dt의 값이 커질 수록 수렴도 더 빨리한다. 그래서 dt가 너무 작으면 수렴하지 못 하고 반면에 dt가 너무 크면 edge를 뚫고 들어간다. 따라서 적절한 dt의 값을 설정하는 것이 중요하다.

위의 실험 결과에서 볼 수 있듯이 이미지 별로 적절한 dt와 c의 값을 설정하는 것이 중요하다. 테스트 용 이미지에서 실험 결과가 가장 잘 나왔던 parameter의 값은 dt = 0.6, c = 0.5이다. 왼쪽 그림에서 볼 수 있듯이 4개의 달의 edge 를 잘 탐색한 것을 볼 수 있다.

해당 이미지에서 dt = 0.7, c = 0.7로 높이게 되면 예상한 바와 같이 실제 달의 edge를 뚫고 들어간 모습을 볼 수 있다. 따라서 적절한 값의 설정이 중요함을 다시 한번 확인할 수 있었다.