EEL 7825 - Projeto Nível II em Cont e Proc de Sinais I
Daniel Augusto Figueiredo Collier
Exercícios - Aula 5

1. Carregue a imagem *I = imread('thumb.pgm');* e segmente a região de interesse do fundo da imagem. Apresente seu programa e resultados obtidos.

Para segmentar a imagem utilizei o algoritmo de otsu. O valor de limiar obtido foi de 125.

```
//exercicio 1: segmentar a região de interesse do fundo
I = imread('dip\thumb.pgm');
limiar = otsu_im(I);
R = 1*(I>limiar); // imagem segmentada
```

2. Segmente as imagens *plate1.pgm* e *plate2.pgm* e isole os números (ou letras) para posterior processamento.

Segmentei a imagem de duas maneiras: na primeira utilizei o limiar calculado pelo algoritmo de otsu; e, na segunda passei a imagem por um detector de bordas e em seguida limiarizei com o algoritmo de otsu. Achei interessante fazer da segunda forma porque na segunda imagem, que tem letras brancas e pretas, o resultado com o limiar sem detecção de bordas perde qualidade.


```
//exercicio 2: isolar números e letras para posterior
processamento
// imagem 1
I1 = imread('dip\plate1.pgm');
// otsu
limiar1 = otsu_im(I1);
R1 = 1*(I1>limiar1);
// sobel
Gv=[-1 -2 -1; 0 0 0; 1 2 1];
Gh=[-1 0 1; -2 0 2; -1 0 1];
Dv = imconv(I1,Gv);
Dh = imconv(I1,Gh);
Dt = sqrt(Dv.^2 + Dh.^2);
Dt = ajeita(Dt);
limiar1 = otsu_im(Dt);
Dt = 1*(Dt>limiar1);
```

3. Aplique a detecção de descontinuidade para a imagem *pollen_d.pgm*. Observe que não só as bordas de objetos, mas também as texturas produzem descontinuidades. Compare os resultados do Laplaciano e Sobel.

O resultado obtido pelo método de Sobel destacou bem mais as continuidades do que o Laplaciano. Com o Laplaciano o resultado ficou muito borrado devido à textura dos polens.



```
//exercício 3: detecção de descontinuidades
// comparar com o laplaciano e sobel
I = imread('dip\pollen_d.pgm');
// laplaciano (pré-pocessar a imagem: filtrar e equalização de
histograma)
// equalização e Laplaciano
E = mogrify((I-1)/255,['-equalize']);
E = round(E*255+1);
L = mkfilter('laplacian');
D = imconv(E,L);
D = abs(D);
limiar = mean(D);
D = 1*(D>limiar);
// Sobel
Gv=[-1 -2 -1; 0 0 0; 1 2 1];
Gh=[-1 0 1; -2 0 2; -1 0 1];
Dv = imconv(I,Gv);
Dh = imconv(I,Gh);
Dt = sqrt(Dv.^2 + Dh.^2);
```

4. Carregue a imagem *casa.pgm*, e faça a detecção das bordas *principais* da imagem (ou seja, não estamos interessados nos pequenos tijolos e telhas da imagem!) Como fazer para evitar o efeito das telhas e tijolos?

Para detectar as bordas principais utilizei o método de Sobel e para evitar o efeito dos tijolos e telhas apliquei uma filtragem com o filtro mediana (5x5). Percebe-se que não ficam tijolos nem telhas na imagem.

```
//exercício 4: detecção das bordas prinicipais
// evitar o efeito das telhas e tijolos
I = imread('dip\casa.pgm');
// sobel (enfatiza muito mais a detecção de bordas)
Gv=[-1 -2 -1; 0 0 0; 1 2 1];
Gh=[-1 0 1; -2 0 2; -1 0 1];
Dv = imconv(I,Gv);
Dh = imconv(I,Gh);
Dt = sqrt(Dv.^2 + Dh.^2);
Dt = ajeita(Dt); // range 0 a 1
//
Dt = mediana(Dt,5);
```

5. Segmente a imagem *celulas1.pgm*. Esta imagem está contaminada com ruído. Agora, tente implementar um algoritmo para contar o número de células encontradas (Dica: procure lembrar de conceitos abordados nas primeiras aulas)

Para segmentar a imagem uma filtragem passa-baixa é feita e depois a segmentação com resultado em binário. Para tentar contar as céculas utilizei uma função de dar *labels* a elementos conectados por vizinhança 4. Como o resultado apresentou *labels* para regiões em que não estava interessado, usei o histograma da imagem para contar o número de pixels para cada *label* dado e usei um valor que observei durante as tentativas como limiar para dizer se a região é célula ou não. O resultado dessa contagem foi de 14 células.

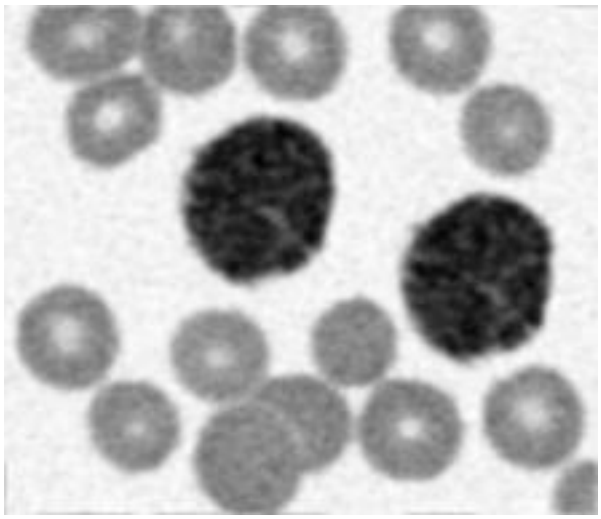

Figura 1: Resultado da filtragem do ruído.

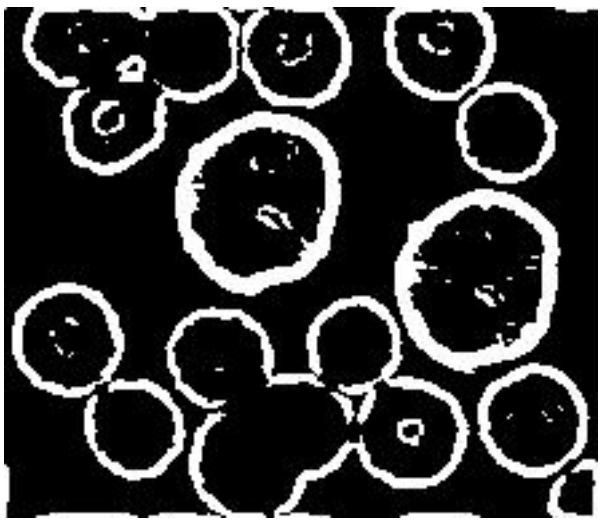

Figura 2: Resultado da detecção de bordas.

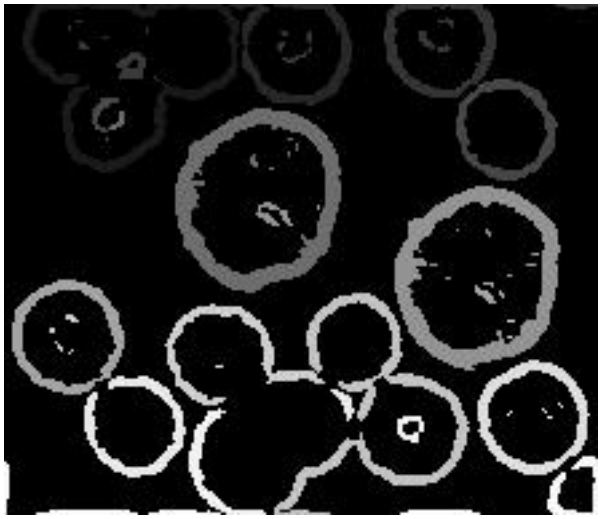

Figura 3: Imagem com diferentes labels.

```
//exercício 5: imagem com ruído, contar o número de células
encontradas // usar conceitos abordados nas primeiras aulas
I = imread('dip\celulas1.pgm');
// filtrar o ruído
h = mkfftfilter(I,'butterworth2',40);
F = fft(I,-1);
F = F.*fftshift(h);
f = real(fft(F,1));
clear F
// detectar bordas
f = edge(f,'fftderiv',0.25);
// label aos objetos
f = bwlabel(f,4);
f = ajeita(f);
// contar as células
f = mogrify((f-1)/255,['-equalize']);
f = ajeita(f);
f = (1*(f > 10)).*f;
h = histograma(256,ajeita(f));
h = h * prod(size(f));
h = (1*(h > 90)).*h;
// número de células
n = length(unique (sort(h')))-2;
```

Listagem da implementação do algoritmo de otsu:

```
function O = otsu_im(I)
// OTSU_IM_
// O: o valor de limiar
// I: imagem em tons de cinza
//
// Uso:
// I = imread('figura.jpg');
// O = otsu(I);
// imshow(1*(I>O),[])
//
// histograma normalizado da imagem
h = histograma(256,I);
// zeroth-order cumulative moment of the histogram
w = cumsum(h);
// first-order cumulative moment of the histogram
mi = cumsum((h.*[1:256]));
// mean value of histogram
M = sum((h.*[1:256]));
// ajustes dos valores para não ocorrer divisão por zero
[m, ka] = max(w);
[m, ki] = min(w(256:-1:1));
ki = 256 - ki;
w = w((2+ki):1:(ka-1));
mi = mi((2+ki):1:(ka-1));
// between class-variance of histogram
E = ((M*w-mi).^2)./(w.*(1-w));
// valor de limiar
[m, O] = max(E);
O = O + ki;

endfunction
```