



UNIVERSIDADE DO MINHO
DEPARTAMENTO DE INFORMÁTICA
TECNOLOGIA DE SEGURANÇA

Tutorial 1

Grupo 12

Autores:

Joel Gama (A82202)



Tiago Pinheiro (A82491)



29 de Novembro de 2019

Conteúdo

1	Introdução	2
2	<i>Smooth images</i>	3
2.1	<i>Input</i>	3
2.2	Algoritmo	7
2.3	<i>Output</i>	10
2.3.1	<i>Noisy image</i>	10
2.3.2	<i>Smoothed image</i>	10
2.4	Análise	11
2.4.1	<i>Salt and Pepper Noise</i>	11
2.4.2	<i>Gaussian Noise</i>	11
2.4.3	<i>Spatial domain com average</i>	12
2.4.4	<i>Spatial domain com gaussian</i>	12
2.4.5	<i>Spatial domain com median</i>	14
2.4.6	<i>Frequency domain com gaussian</i>	14
2.4.7	<i>Frequency domain com butterworth</i>	15
3	<i>Detect edges with Canny Detector</i>	16
3.1	<i>Input</i>	16
3.2	Algoritmo	17
3.3	<i>Output</i>	19
3.3.1	<i>Edge strength image before nonmax suppression</i>	19
3.3.2	<i>Edge strength image after nonmax suppression</i>	20
3.3.3	<i>Edge strength image after hysteresis thresholding</i>	20
4	Conclusão	21

1 Introdução

No âmbito da unidade curricular de Visão por Computador, englobada no perfil de Computação Gráfica, foi-nos proposto fazer um tutorial. Este tutorial, que é o primeiro de dois que irão ser realizados este semestre, é constituído por duas partes. Sendo que, ambas as partes, são constituídas por um programa.

O primeiro é um programa de *smooth images* que aplica vários conceitos abordados nas aulas teóricas e práticas como, por exemplo, *noise*, filtros e *smoothing*. O segundo é um programa para *edge detection* usando o *Canny detector*, este programa irá ter vários *outputs* depois de serem aplicados vários métodos a uma imagem com *Gaussian noise* que é dada como *input*.

2 Smooth images

2.1 Input

```
image_path = input('Give the image name:\n');

path = strcat('Imagens\',image_path);
format = extractAfter(image_path,'.');

if (strcmp(format,'png'))
    image = imread(path, 'png');
else
    image = imread(path, 'jpg');
end

imageGreyScale = rgb2gray(image);
```

Inicialmente, o utilizador deve fornecer o nome da imagem sobre a qual irá ser aplicado o algoritmo. O código prevê que o utilizador escreva não só o nome como também a extensão do ficheiro (*png* ou *jpg*). O programa apenas funciona se a imagem estiver dentro de uma pasta chamada **Imagens**. Para além disto, o programa, nesta primeira fase, transforma a imagem numa *grey scale image*.

```
typeOfNoise = input('Select the type of noise:\n
(1) Salt-an-pepper\n      (2) Gaussian Noise:\n');

if (typeOfNoise==1)
    typeOfNoise = "salt & pepper";
    noiseArg = input('What is the percentage of occurrence
                     desired?\n Type "-1" for default values\n:');
elseif (typeOfNoise==2)
    typeOfNoise = "gaussian noise";
    noiseArg = input('What is the variance desired?\n
                     Type "-1" for default values\n:');
else
    fprintf('Error');
    exit;
end

if(noiseArg == (-1))
    noiseArg = 0.05;
end
```

Neste programa, o utilizador tem a possibilidade de escolher o tipo de *noise* a utilizar, para isso basta introduzir o valor 1, caso queira utilizar o *salt-an-pepper*, ou o valor 2, caso queira utilizar o *gaussian noise*, se o utilizador inserir outro valor, o programa vai fechar com uma mensagem de erro.

Conforme o tipo de *noise* selecionado o programa vai pedir ao utilizador para inserir um valor para o argumento do respetivo *noise*. Caso, o utilizador insira o valor -1 o programa vai assumir valores *default* para os dois tipos de *noise*.

```

filteringDomain = input('Select the filtering domain:\n
                        (1) Spatial domain\n      (2) Frequency domain:\n');

if (filteringDomain==1)
    filteringDomain="spatial";
    % ... (Código omitido para melhor visualização)
elseif (filteringDomain==2)
    filteringDomain="frequency";
    % ... (Código omitido para melhor visualização)
else
    fprintf('Error');
    exit();
end

```

O programa permite ao utilizador escolher um de dois tipos de processamento de imagem, se o utilizador o valor 1 é escolhido o *Spatial domain*, por outro lado, se o valor escolhido for o 2 o utilizador seleciona o *Frequency domain*, em qualquer outro cenário, o programa termina com uma mensagem de erro.

```

% ...
if (filteringDomain==1)
    typeOfSmoothing = input('Select the type of smoothing:\n
                            (1) Average\n      (2) Gaussian\n
                            (3) median filters\n');

    % ...
elseif (filteringDomain==2)
    typeOfSmoothing = input('Select the type of smoothing:\n
                            (1) Gaussian\n      (2) Butterworth filters\n');

    % ...
% ...

```

De seguida, o programa pede o tipo de *smoothing* que o utilizador deseja, tendo em atenção a escolha feita anteriormente.

```

if (typeOfSmoothing==1)
    typeOfSmoothing = "average";
    a = input('What is the width desired?\n
              Type "-1" for default values\n:');
    if (a==(-1))
        a = 3
    end
    b = 0;
elseif (typeOfSmoothing==2)
    typeOfSmoothing = "gaussian";
    a = input('What is the sigma desired ?\n
              Type "-1" for default values\n:');
    if (a==(-1))
        a = 10
    end
    b = input('What is the size of your kernel?\n
              Type "-1" for default values\n:');
    if (b==(-1))
        b = 5
    end
elseif (typeOfSmoothing==3)
    typeOfSmoothing = "median";
    a = input('What is the width desired ?\n
              Type "-1" for default values\n:');
    if (a==(-1))
        a = 3
    end
    b = 0;
else
    fprintf('Error');
    exit();
end

```

Para o *spatial domain* o programa fornece três hipóteses de *smoothing*: *average*, *gaussian* e *median*; Que podem ser selecionados introduzindo os valores 1,2 ou 3, respetivamente. Caso não seja introduzido um dos três valores alternativos o programa encerra com uma mensagem de erro.

```

if (typeOfSmoothing==1)
    typeOfSmoothing = 'gaussian';
    a = input('What is the sigma desired?\n
              Type "-1" for default values\n:');
    if (a==(-1))
        a = 10
    end
    b = 0;
elseif (typeOfSmoothing==2)
    typeOfSmoothing = 'butter';
    a = input('What is the filter order desired?\n
              Type "-1" for default values\n:');
    if (a==(-1))
        a = 20
    end
    b = input('What is the cutoff desired?\n
              Type "-1" for default values\n:');
    if (b==(-1))
        b = 5
    end
else
    fprintf('Error');
    exit();
end

```

Para o *frequency domain* o programa tem o mesmo funcionamento, mas apenas duas alternativas de *smoothing*: *gaussian* ou *butterworth filters*.

Por fim, são pedidos argumentos para acompanhar os tipos de *smoothing*. São pedidos os seguintes valores:

- o valor da *width* na *average do spatial domain*.
- o valor do *sigma* e do *kernel* no *gaussian do spatial domain*.
- o valor da *width* na *median do spatial domain*.
- o valor do *sigma* no *gaussian do frequency domain*.
- o valor da *order* e do *cutoff* no *butterworth do frequency domain*.

Caso o utilizador não pretenda introduzir valores, basta introduzir o valor -1 e o programa assume os valores *default*.

2.2 Algoritmo

Noise

Para a introdução do ruído *salt and pepper* é criada uma matriz com o mesmo tamanho da imagem passada como argumento. Em seguida é utilizada metade a percentagem do ruído para obter os pixels brancos (*salt*, com valor 0 de saturação) e a outra metade para os pixels pretos(*pepper* com valor 255 de saturação). Desta forma a distribuição de *salt* e *pepper* é a mesma.

```
x = rand(size(imageGreyScale));  
  
% Pixeis menores que a metade da percentagem  
b = find(x < noiseArg/2);  
imageGreyScale(b) = 0;  
  
% Pixeis entre a metade e o valor total da percentagem  
p = find(x >= noiseArg/2 & x < noiseArg);  
imageGreyScale(p) = 255;
```

Já no ruído gaussiano o algoritmo utilizado foi o seguinte.

```
p2 = 0;  
imageGreyScale = im2double(imageGreyScale);  
imageGreyScale = imageGreyScale + sqrt(noiseArg)  
    *randn(size(imageGreyScale)) + p2;
```

Os valores da imagem são convertidos para *double* apenas para que o algoritmo funcione por questões dos tipos do *matlab*. Em seguida é utilizada a formula $b = a + \sqrt{p4} * \text{randn}(\text{sizeA}) + p3$ para aplicar o ruído à imagem.

Smoothing (Spacial)

Neste tutorial é possível aplicar três tipos de *smoothing*: *Average*, *Gaussian* e *Median*.

O tipo *Average* começa por preencher uma matriz de tamanho *width* com valores $1/\text{width}^2$ em cada elemento. Em seguida é feita a convulsão entre a imagem com ruído e essa matriz.

```
matrix = ones(width);  
for i = 1:width  
    for j = 1:width  
        matrix(i, j) = 1/(width*width);  
    end  
end  
imageGreyScale = conv2(imageGreyScale, matrix);
```

No tipo *Gaussian* é utilizado uma função para fazer o filtro gaussiano com os parâmetros passados pelo utilizador. Depois é aplicada a função *imfilter* à imagem com ruído e com o filtro criado antes.

```
h = fspecial('gaussian', kernelSize, width);  
imageGreyScale = imfilter(imageGreyScale, h);
```

No último tipo, *Median*, é utilizada a função *medfilter2* com a imagem com ruído e uma matriz com tamanho *width*. A matriz é utilizada para fazer a média da imagem na vizinhança do tamanho da matriz, isto é, se $[p \ q] = \text{size}(\text{matrix})$ então a vizinhança será p por q.

```
matrix = ones(width);
[p, q] = size(matrix);
imageGreyScale = medfilt2(imageGreyScale, [p q]);
```

Cada tipo de *smoothing* funciona melhor dependendo do tipo de ruído presente na imagem. Então, depois de analisar todos os tipos, concluímos que:

- *Average Smoothing* é indicado para imagens com ruído gaussiano.
- *Gaussian Smoothing* é indicado para imagens com ruído gaussiano.
- *Median Smoothing* é indicado para imagens com ruído *salt and pepper*.

Foi também testado o smoothing para diferentes tamanhos de *kernel*. Com tamanho 5 a imagem fica nítida, mas o ruído continua visível. Quando se aumenta o *kernel* o ruído presente na imagem diminui, mas a imagem perde nitidez, ficando desfocada.

Discrete Fast Fourier Transform

A partir das transformadas de *Fourier* podemos observar que as imagens estavam mais claras no centro porque correspondem às frequências baixas da imagem original. O ruído torna a *dft* mais uniforme porque a imagem fica mais cinzenta por toda ela. O *smoothing* resolve esse problema, voltando a focar a claridade no centro da imagem.

Smoothing (Frequency)

Neste tutorial é possível aplicar dois tipos de *smoothing*: *Gaussian* e *Butterworth*. Para este processo utilizamos como base os slides das aulas teóricas.

O início do processo de filtragem em frequência é o mesmo nos dois métodos.

```
f = double(imageGreyScale);
[M N] = size(f);
P = 2*M;
Q = 2*N;
f = padarray(f, [M N], 'post');

for i = 1:M
    for j = 1:N
        f(i, j) = (-1)^(i+j) * f(i, j);
    end
end
G = fft2(f);
```

Começa por ser convertida a imagem para *double* e são retirados os valores das linhas e colunas da imagem. Em seguida são calculados dois outros valores para serem usados no filtro. É então feito o *padding* da imagem e feito o cálculo de $(-1)^{(i+j)} * f(i,j)$ para cada valor da imagem. No final desta primeira parte é calculada a transformada de *Fourier* da imagem.

Depois deste passo o processo é tratado de forma diferente em função do tipo de filtro a usar. No caso do filtro *Gaussiano*:

```

H = fspecial('gaussian', [P Q], a);
G = H .* G;
smoothed_image = ifft2(G);
smoothed_image = real(smoothed_image);

for i=1:M
    for j=1:N
        smoothed_image(i, j) = (-1)^(i+j) * smoothed_image(i, j);
    end
end

smoothImage = uint8(smoothed_image(1:M, 1:N));
smoothImage = double(smoothImage);

```

É feito o *fspecial* para criar o filtro gaussiano para aplicar na imagem. Em seguida é feita a multiplicação do filtro pela imagem. Por fim é feito a inversa da transformada de *Fourier*, é retirada a parte imaginária da imagem e os cálculos inversos que foram feitos no processo inicial.

Já no filtro *Butterworth* o processo é ligeiramente diferente. O filtro (neste caso *high-pass*) é calculado manualmente através da formula $H(i,j) = 1 / (1 + (\text{cutoff}/\text{dist})^{2*\text{filter_order}})$. Os passos seguintes são comuns aos dois tipos de filtros, uma vez que é sempre necessários fazer os cálculos do inverso do que foi feito inicialmente.

```

H = double(zeros(P, Q));
for i=1:P
    for j=1:Q
        dist = ((i-P/2)^2 + (j-Q/2)^2)^0.5;
        H(i, j) = 1 / (1 + (cutoff/dist)^(2*filter_order));
    end
end
G = H .* G;
smoothed_image = ifft2(G);
smoothed_image = real(smoothed_image);

for i=1:M
    for j=1:N
        smoothed_image(i, j) = (-1)^(i+j) * smoothed_image(i, j);
    end
end

smoothImage = uint8(smoothed_image(1:M, 1:N));
smoothImage = double(smoothImage);

```

2.3 Output

2.3.1 Noisy image



Figura 1: *Output* do programa usando o *Gaussian* como o tipo de *noise* com média 0 e variância de 0.05

2.3.2 Smoothed image



Figura 2: *Output* do programa usando o *Smoothing Gaussian* e o filtro *Spatial domain* com os valores 10 e de 5 para o *sigma* e *kernel*, respectivamente

2.4 Análise

2.4.1 Salt and Pepper Noise



Figura 3: As imagens 3a, 3b e 3c representam o uso do tipo de ruído *Salt and Pepper* com os valores de ocorrência, 0.05, 0.25 e 0.50, respectivamente.

A ocorrência é um parâmetro que indica a percentagem dos pixels que vão ser brancos ou pretos. Se o valor da ocorrência for 0 significa que vão ser alterados 0 pixels, ou seja, a imagem vai ser igual à original. Se o valor da ocorrência for 1 significa que 100 por cento do pixels irão ser alterados, assim a imagem vai passar a ser apenas um conjunto de "pontos" pretos e brancos. Então, quanto maior a ocorrência, maior o ruído presente na imagem, como é possível observar na figura 3.

2.4.2 Gaussian Noise

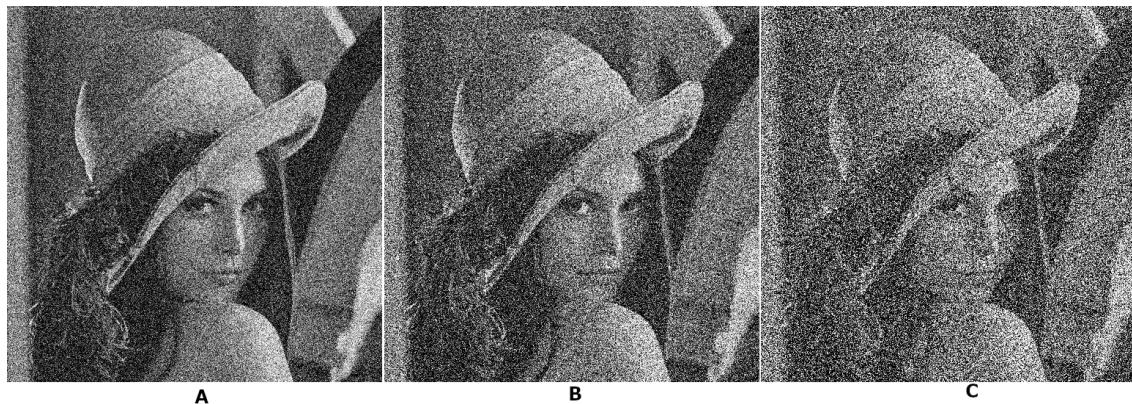


Figura 4: As imagens 4a, 4b e 4c representam o uso do tipo de ruído *Gaussian* com o valor de 0 para a média e de 0.05, 0.10 e 0.25, respectivamente, para a variância.

A variância é um parâmetro que indica a quantidade de ruído presente na imagem. Então, quanto maior a variância, maior o ruído presente na imagem, como é possível observar na figura 4.

2.4.3 Spatial domain com average



Figura 5: As imagens 5a, 5b e 5c representam o uso do ruído *salt and pepper* e com a *average* a ser o tipo de *smoothing*, os valores para o parâmetro *width* são 3,7 e 50, respectivamente.

Como foi explicado em cima, no filtro *average* é feita a multiplicação de uma matriz com a imagem. Os valores dessa matriz são calculado com base no parâmetro *width*. Desta forma, quanto maior o *width* menor o ruído presente na imagem, mas também será maior o desfoque.

2.4.4 Spatial domain com gaussian

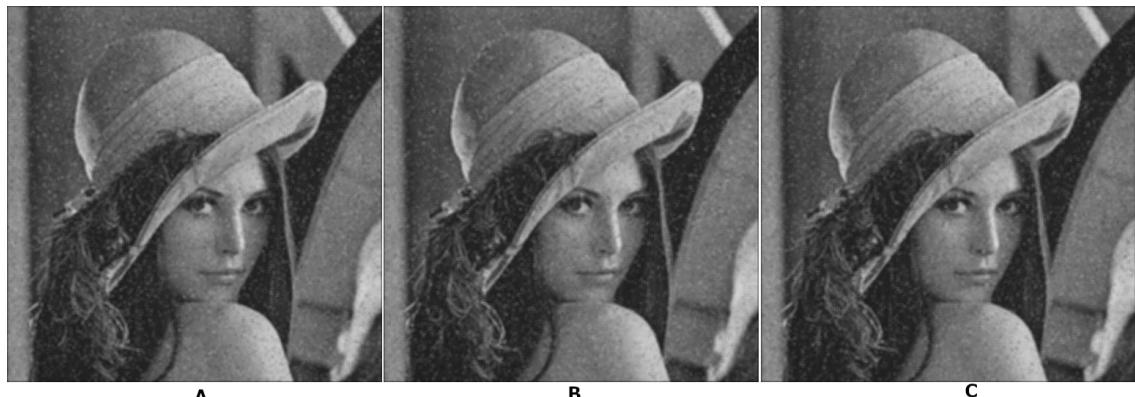


Figura 6: As imagens 6a, 6b e 6c representam o uso de ruído *salt and pepper* juntamente com a utilização do *smooth* gaussiano do *spatial domain*. As três imagens tem o mesmo *kernel*, 5, mas o valor do *sigma* é 5,10 e 15, respectivamente.

No filtro gaussiano existem dois parâmetros, tamanho do *kernel* e *sigma*. O aumento destes parâmetros causa uma diminuição no ruído da imagem, porém aumenta o desfoque.



Figura 7: As imagens 7a, 7b e 7c representam o uso de ruído *salt and pepper* juntamente com a utilização do *smooth* gaussiano do *spatial domain*. As três imagens tem o mesmo *kernel*, 10, mas o valor do sigma é 3,5 e 15, respectivamente.



Figura 8: As imagens 8a, 8b e 8c representam o uso de ruído *salt and pepper* juntamente com a utilização do *smooth* gaussiano do *spatial domain*. As três imagens tem o mesmo *kernel*, 30, mas o valor do sigma é 1,5 e 15, respectivamente.

2.4.5 Spatial domain com median



Figura 9: As imagens 9a, 9b e 9c representam o uso de ruído *salt and pepper* juntamente com a utilização do *smooth median (moving median)* do *spatial domain*. Os valores usados na *width* são: 3,5 e 20, respectivamente

No filtro *median* o parâmetro alterado é o *width*. Quanto maior for este parâmetro menos ruído estará presente na imagem. Mas a imagem vai ficando cada vez menos nítida.

2.4.6 Frequency domain com gaussian



Figura 10: As imagens 10a, 10b e 10c representam o uso de ruído *salt and pepper* juntamente com a utilização do *smooth gaussiano* do *frequency domain*. O parâmetro *sigma* assume os valores 5, 20 e 50 nas imagens a, b e c, respectivamente

Ao contrário do que acontece no *smooth gaussiano* espacial, aqui o aumento do *sigma* resulta num aumento do foco da imagem, apesar de aumentar também o ruído.

2.4.7 Frequency domain com butterworth



Figura 11: As imagens 11a, 11b e 11c representam o uso de ruído *salt and pepper* juntamente com a utilização dos filtros de *butterworth* do *frequency domain*. Nas três imagens a *order* assume o valor de 2, os valores de *cutoff* (valor da frequência) usados são 5, 20 e 80, respectivamente.

O valor do *cutoff* representa a frequência de corte. Sabendo isso e observando as imagens podemos ver que com valores de *cutoff* menores obtemos uma imagem com muito menos ruído. Mas imperceptível. Se aumentarmos esse valor a imagem fica mais perceptível, mas com mais ruído.



Figura 12: As imagens 12a, 12b e 12c representam o uso de ruído *salt and pepper* juntamente com a utilização dos filtros de *butterworth* do *frequency domain*. Nas três imagens o valor de *cutoff* é 80, já a *order* assume valores de 2, 6 e 20, respectivamente.

O aumento da *order* no filtro *butterworth* faz com que a imagem fique mais "pixelizada".

Em suma, no domínio da frequência a tarefa de filtrar torna-se mais fácil quanto maior a janela. Já a filtragem no domínio espacial normalmente têm pequenas máscaras para tentar captar a essência da função de filtro completo, para que, desta forma, seja mais rápido e menos complexo.

3 Detect edges with Canny Detector

3.1 Input



Figura 13: *Output* da parte 1 do programa usando o ruído *Gaussian* com variância de 0.05

```
image = imread('Input2.png','png');
```

Como é pedido no enunciado, o *input* do segundo programa é uma imagem com ruído gaussiano, neste caso, nós decidimos usar como input a imagem gerar no primeiro programa.

3.2 Algoritmo

Gaussian smoothing

O processo utilizado para *Gaussian smoothing* é o mesmo utilizado no exercício de *Smoothing images*.

Gradient

O calculo do gradiente está dividido em 3 partes. Calcular o x e y do gradiente, a magnitude e a direção. Para o calculo do Gx e Gy é usada a função *imgradientxy* com a imagem como argumento. Em seguida é calculada a magnitude.

```
for i = 1:size(Gx)
    for j = 1:size(Gy)
        % raiz quadrada da soma dos quadrados de Gx e Gy
        Gmag(i, j) = sqrt(double(Gx(i, j)^2 + Gy(i, j)^2));
    end
end
```

Por fim, a direção é calculada utilizando a função *atan2* com os Gx e Gy como argumentos.

Nonmax

Para calcular a *Non Maximum Suppression* são necessárias as direção e a magnitude da imagem calculadas anteriormente. São testadas as várias direções e comparada a intensidade do pixel atual com a intensidade da *edge* dos pixels vizinhos nos dois lados opostos ao longo da direção. Se a intensidade do pixel atual for maior que a dos pixels vizinhos na direção do gradiente, o valor da resistência da *edge* é mantido. Se não, retira-se o valor definindo-o como zero.

Double threshold

No *Double threshold* as *edges* que estiverem acima ou abaixo dos valores passados são retiradas, ficando apenas os valores intermédios.

Hysteresis thresholding

```
I = imageGreyScale; % Imagem
v = zeros(M,N); % Visitados

for i = 1:M
    for j = 1:N
        if v(i,j) ~= 1
            if I(i,j) >= high
                v(i,j)=1;
            elseif I(i,j) <= low
                I(i,j)=0;
                v(i,j)=1;
            else
                [I,v]=ht(I,v,high,low,i,j);
            end
        end
    end
end
```

Para calcular o *Hysteresis thresholding* são utilizados dois valores de *threshold* e uma matriz com as informações dos pixels já visitados para reduzir a computação necessária.

Percorrendo a imagem é verificado se o valor é superior ou igual ao maior valor de *threshold*. Caso seja o valor mantém-se. No caso de o valor ser menor que o menor valor de *threshold*, o valor do pixel passa a 0 (zero). Por fim, é chamada uma função recursiva que verifica se as *edges* estão ligados a *edges* com valores altos (superiores ao *threshold* maior). Se for esse o caso a *edge* é mantida. Se a *edge* estiver separada é retirada.

3.3 Output

3.3.1 Edge strength image before nonmax suppression



Figura 14: *output* da imagem antes da *nonmax suppression*

Apesar do algoritmo estar correto, o output não foi o desejado e o grupo não conseguiu encontrar a falha.

3.3.2 Edge strength image after nonmax suppression

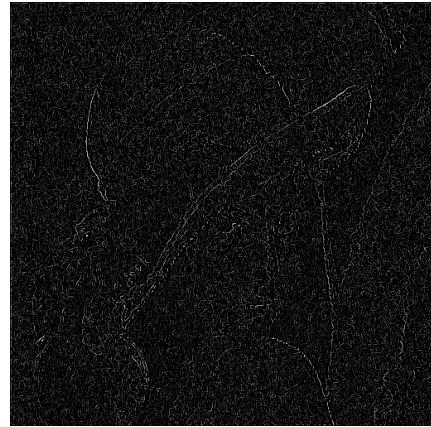


Figura 15

Na imagem acima é possível observar que apenas são visíveis algumas das *edges*. Pois o *Nonmax* utiliza a direção e o gradiente para retirar os pixeis que não se encontram na mesma direção das *edges*.

3.3.3 Edge strength image after hysteresis thresholding

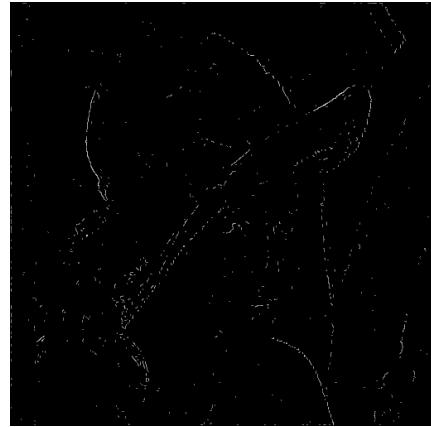


Figura 16

Nesta última imagem observamos que houve uma redução nas *edges*. Esta redução deve-se ao *hysteresis thresholding* uma vez que este retira as *edges* com valor baixo e as que não estão ligadas às *edges* fortes.

4 Conclusão

A realização deste trabalho permitiu ao grupo estabelecer e alargar o conhecimento sobre *Image smoothing*, *Edge Detection* e a forma como estes funcionam, ajudando a interiorizar os conhecimentos obtidos nas aulas práticas leccionadas até ao momento.

Neste tutorial o enunciado fornecido foi bastante exigente, isto fez com que o grupo se tivesse que organizar e trabalhar de forma bastante eficaz. Felizmente, o *matlab* dispõe de uma documentação que ajuda os utilizadores a entender o modo de funcionamento. Na fase de tratamento de informação, grupo sentiu dificuldades, devido ao pouco conhecimento obtido na área até ao momento.

Em suma, este trabalho prático ajudou a evoluir o seu conhecimento sobre *Image smoothing e Edge Detection*. O grupo espera também poder aplicar os conhecimentos obtidos em trabalhos futuros.