



UNIVERSIDADE DO MINHO

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA

VISÃO POR COMPUTADOR

---

## Tutorial 1: Image Filtering and Edge Detection

---

*Autor:*  
João Palmeira

*Número Aluno:*  
A73864

28 de Novembro de 2018

## Conteúdo

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Desenvolvimento</b>	<b>4</b>
2.1	Smoothfilters . . . . .	4
2.1.1	Smoothfilters.m . . . . .	4
2.1.2	Main_smoothfilters . . . . .	5
2.1.3	Tutorial Smoothfilters . . . . .	8
2.2	Canny Detector . . . . .	13
2.2.1	Conversão para cinza . . . . .	14
2.2.2	Determinar a intensidade dos Gradientes . . . . .	15
2.2.3	NON MAXIMUM SUPPRESSION . . . . .	15
2.2.4	DOUBLE THRESHOLDING . . . . .	16
2.2.5	EDGE TRACKING BY HYSTERESIS . . . . .	17
2.2.6	Tutorial Canny Detector . . . . .	17
<b>3</b>	<b>Conclusão</b>	<b>21</b>

## Lista de Figuras

1	Conversão imagem original para imagem cinzenta . . . . .	4
2	Imagens com filtro <i>average</i> com diferentes <i>kernels</i> . . . . .	5
3	Imagens com filtro <i>gaussian</i> com diferentes <i>kernels</i> . . . . .	5
4	Imagens com filtro <i>median</i> com diferentes <i>kernels</i> . . . . .	6
5	Imagens SNP com filtro <i>average</i> com diferentes <i>kernels</i> . . . . .	6
6	Imagens SNP com filtro <i>gaussian</i> com diferentes <i>kernels</i> . . . . .	7
7	Imagens SNP com filtro <i>median</i> com diferentes <i>kernels</i> . . . . .	7
8	Iniciar . . . . .	8
9	Seleção da imagem . . . . .	9
10	Tipo de ruído . . . . .	9
11	Ocorrência . . . . .	10
12	Domínio de filtragem . . . . .	10
13	Tipo de filtro . . . . .	10
14	Ordem de Butterworth . . . . .	11
15	Frequência de corte . . . . .	11
16	Espetros . . . . .	12
17	Resultados obtidos . . . . .	12
18	Fluxo do algoritmo de <i>Canny Edge Detection</i> . . . . .	14
19	Conversão imagem original para imagem cinzenta . . . . .	15
20	Interpolação do gradiente . . . . .	16
21	Sem interpolação . . . . .	16
22	Iniciar . . . . .	17
23	Seleção da imagem . . . . .	18
24	Seleção variância filtro gaussiano . . . . .	18
25	Seleção tamanho Gaussian Smoothing . . . . .	18
26	Seleção variância Gaussian Smoothing . . . . .	19
27	Imagens do Gradiente . . . . .	19
28	Seleção limite inferior thresholding . . . . .	20
29	Seleção limite superior thresholding . . . . .	20
30	Resultados obtidos . . . . .	20

## 1 Introdução

Processamento digital de imagem é o uso de algoritmos computacionais para realizar o processamento de imagem em imagens digitais. A filtragem é um método amplamente utilizado no processamento de imagem.

A filtragem digital é a principal ferramenta no processamento de imagens, sendo usada para uma variedade de aplicações de processamento, como aprimoramento e deteção de bordas, redução de ruído, remoção de perda de dados e restauração de imagens. A saída de tal filtragem é uma combinação dos pixels da imagem de entrada, geralmente assumidos em uma região ou vizinhança.

No caso dos algoritmos de Edge Detection uma aresta pode ser definida como um conjunto de pixels conectados que formam um limite entre duas regiões de desarranjo. A deteção de bordas é um método de segmentar uma imagem em regiões de conclusão e desempenha um papel muito importante no processamento digital de imagens e aspectos práticos da nossa vida.

Neste relatório será dado, numa primeira fase, um foco à aplicação de filtros bem como à aplicação de suavização tanto no domínio espacial como no domínio de frequência. Numa segunda fase pretende-se dar destaque a um dos algoritmos de Edge Detection, mais concretamente o Canny Edge Detector.

Para tal serão criados dois tipos de programas como resposta para os dois tópicos referidos anteriormente.

## 2 Desenvolvimento

Como foi mencionado anteriormente este trabalho é composto pela composição de dois tipos de programas. O primeiro, *Smoothfilters*, que consiste na filtragem de imagens no domínio espacial e domínio da frequência, já o segundo, *Canny Detector*, consiste na deteção de borda nas imagens.

### 2.1 Smoothfilters

Nesta secção irei explicar em que consiste o programa *Smoothfilters*. Este programa é constituído por dois *scripts*: um para a componente gráfica e outro para a componente funcional, isto é, este *script* irá receber os dados recolhidos pela componente gráfica e, consoante o utilizador fornecer, irá utilizar as funções de modo a responder aos seus pedidos.

#### 2.1.1 Smoothfilters.m

Este *script* diz respeito à componente gráfica, isto é, a interface é criada neste *script*. Esta interface responde a todos os tópicos exigidos pelo enunciado, permitindo ao utilizador fornecer todos os *inputs* tais como:

- Imagem RGB para converter a preto e branco;
- Tipo de ruído;
- Parâmetros do ruído;
- Tipo de domínio;
- Tipo de suavização;
- Tipo de filtro relativo ao tipo de suavização;

De destacar que a conversão da imagem original é realizada neste ficheiro, tal como pedido no enunciado.

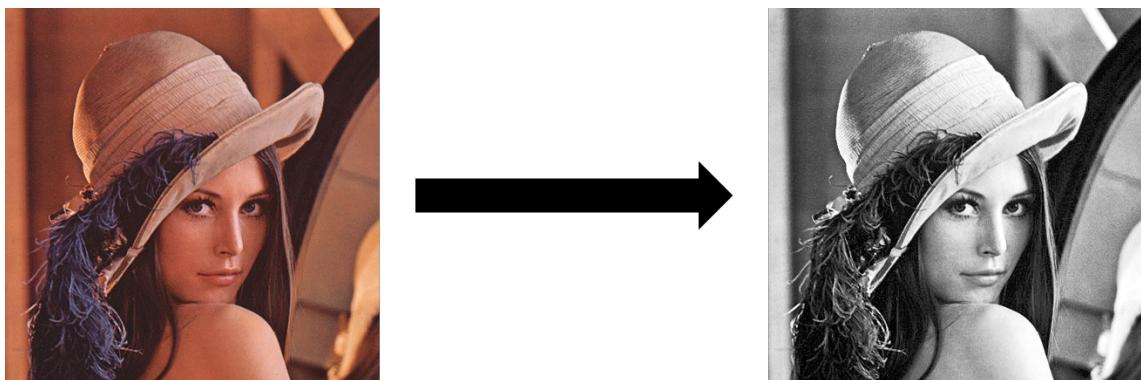


Figura 1: Conversão imagem original para imagem cinzenta

Após recolhida esta informação, ela é passada como argumento para o *script main\_smoothfilters*.

### 2.1.2 Main\_smoothfilters

Relativamente a este ficheiro, só é possível executá-lo se o utilizador fornecer toda a informação necessária. Depois de receber toda a informação, a partir da imagem convertida em tons de cinza gerada a partir da original fornecida pelo utilizador, aplica-se o filtro escolhido através da função *imnoise* tendo em atenção o tipo e os seus parâmetros correspondentes.

De seguida, aplica-se o domínio escolhido:

**Caso 1: Domínio Espacial** Se o domínio escolhido for o domínio espacial, ter-se-á de escolher entre os filtros *average*, *gaussian* ou *median* e para aplicar estes filtros à imagem com ruído utilizamos as funções *fspecial* para criar o filtro, consoante o tipo e os parâmetros, e *imfilter* para aplicar o filtro à imagem com ruído.

**Podemos ver agora alguns testes:**

**Com ruído gaussiano**



Tamanho do kernel: 5



Tamanho do kernel: 10



Tamanho do kernel: 20

Figura 2: Imagens com filtro *average* com diferentes *kernels*



Tamanho do kernel: 5  
Variância: 0.3



Tamanho do kernel: 10  
Variância: 0.3



Tamanho do kernel: 20  
Variância: 0.3

Figura 3: Imagens com filtro *gaussian* com diferentes *kernels*

Figura 4: Imagens com filtro *median* com diferentes *kernels*

Como podemos ver pelas imagens anteriores, após lhes serem aplicado o ruído gaussiano, verificamos que quando lhe atribuímos um filtro de domínio espacial é possível constatar que com um tamanho de *kernel* inferior não existe tanto desfoque da imagem em qualquer um dos três filtros (*average*, *gaussian*, *median*). Neste caso os melhores filtros são o *average* e o *median*.

#### Com ruído *Salt-and-Pepper*

Figura 5: Imagens SNP com filtro *average* com diferentes *kernels*

Figura 6: Imagens SNP com filtro *gaussian* com diferentes *kernels*Figura 7: Imagens SNP com filtro *median* com diferentes *kernels*

Tal como nas imagens anteriores, após lhes serem aplicado o ruído *Salt-and-pepper*, verificamos que quando lhe atribuímos um filtro de domínio espacial é possível constatar que com um tamanho de *kernel* inferior não existe tanto desfoque da imagem em qualquer um dos três filtros (*average*, *gaussian*, *median*). E mais uma vez o o filtro *median* é o melhor filtro a aplicar.

**Caso 2: Domínio de Frequência** Caso a escolha reverta para um domínio de Frequência, será necessário realizar o *padding* da imagem, isto é, realizar o preenchimento de imagem introduzindo novos pixel ao redor das bordas da imagem. De seguida a essa imagem é-lhe aplicado ou o filtro gaussiano ou o filtro *butterworth* através das seguintes funções, respetivamente:

**gaussianF:** Começa por criar uma distribuição de peso de acordo com as posições do filtro utilizando a função  $dist(i, j) = \sqrt{(i - (a/2))^2 + (j - (b/2))^2}$  aplicando ao mesmo tempo a função gaussiana  $h(i, j) = \exp(-((dist(i, j))^2)/(2*(d0^2)))$ . De seguida gera a matriz de imagem suavizada multiplicando o filtro pela matriz da imagem com ruído.

**butterworthF:** À semelhança da função para o filtro gaussiano, começa-se por criar uma distribuição de peso de acordo com as posições do filtro utilizando a função  $dist(i, j) = \sqrt{(i - (r/2))^2 + (j - (c/2))^2}$  aplicando ao mesmo tempo a função *butterworth*  $h(i, j) = (1 + (dist(i, j)/d0)^{2 * order})^{-1}$ . Repetindo o mesmo processo da anterior, gera a matriz da imagem suavizada multiplicando o filtro pela matriz da imagem com ruído.

Finalmente, em cada um dos casos, para se obter a imagem final suavizada aplica-se a inversa

discreta de *Fast Fourier*.

Para comparar os dois do domínios utilizo as seguintes métricas: *Signal to Noise* para a imagem com ruído e a *Signal to Noise* para a imagem com suavização, onde em ambas quanto mais alta for a relação sinal-ruído, menor é o efeito do ruído. Após alguns testes cheguei a conclusão que o SNR para a imagem suavizada é maior no domínio de frequência, já o SNR para a imagem com ruído é maior no domínio espacial.

Por último, para completar o programa é criada uma função que fornece uma comparação das 3 imagens geradas, ou seja, a imagem cinzenta, a imagem com ruído e a imagem suavizada. A estas imagens, é aplicada a função relativa à Transformada de *Fourier*. O que é a DFT? O DFT é uma técnica matemática para decompor um sinal em componentes sinusoidais de várias frequências, variando de 0 de frequência (isto é, constante) até à frequência máxima (isto é, ligeiramente abaixo da metade da taxa de amostragem). Desta feita, aplica-se a função (*fft2*) de transformada rápida de *Fourier* de maneira a gerar a transformada de *Fourier* bidimensional. De seguida, centra-se a imagem e aplica-se o módulo de modo a obter todos os valores reais da imagem. Por fim, converte-se a matriz para a imagem através da função *mat2gray*, acabando por obter um *output* idêntico ao que veremos no fim do tutorial seguinte.

### 2.1.3 Tutorial Smoothfilters

Para iniciar este tutorial seleccionamos a pasta onde estão os ficheiros (ou seja, a pasta "Problema\_1"), posteriormente seleccionamos o ficheiro *smoothfilters.m* e é esse mesmo ficheiro que vamos correr, ou seja, após seleccionarmos, clicamos em **RUN**. De salientar que consoante as escolhas do utilizador, os dados pedidos variam.

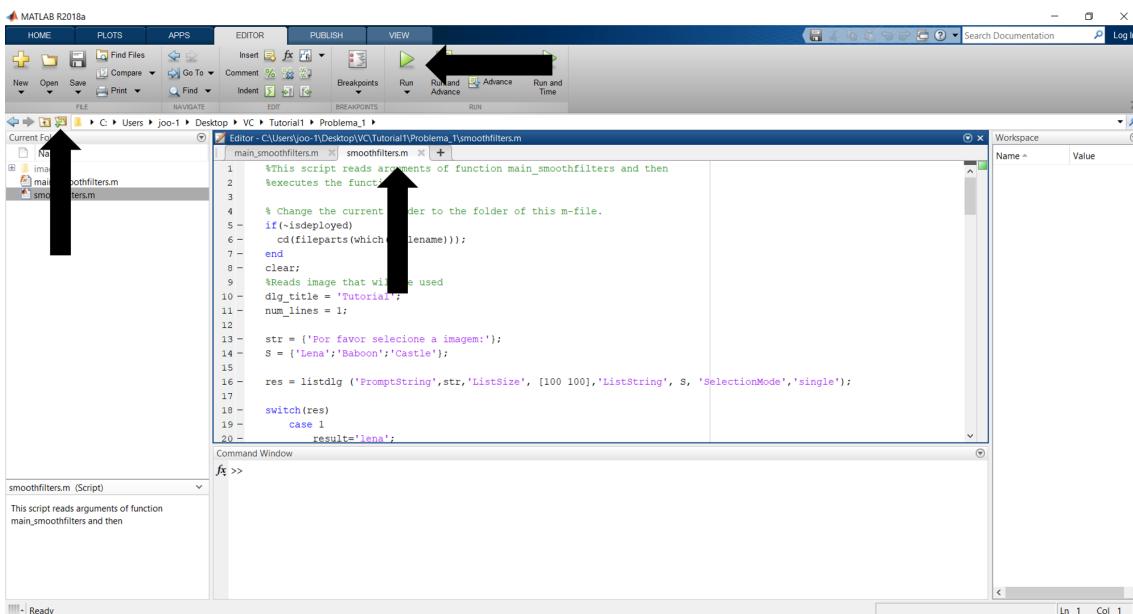


Figura 8: Iniciar

Quando o programa é iniciado será exibida uma lista com as imagens disponíveis e seleciona-se uma delas. Neste caso a imagem selecionada foi a Lena.

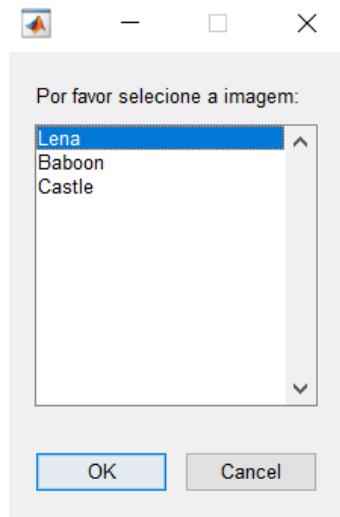


Figura 9: Seleção da imagem

Após a seleção da imagem é pedido que se escolha o tipo de ruído que pretendemos aplicar, isto é, se queremos aplicar o ruído gaussiano ou o ruído *Salt-and-pepper*. Neste caso seleccionamos o ruído *Salt-and-pepper*.

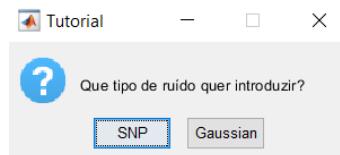


Figura 10: Tipo de ruído

Neste momento é nos pedido que indiquemos uma ocorrência e, como podemos ver pela imagem, utilizamos o valor 0.3.

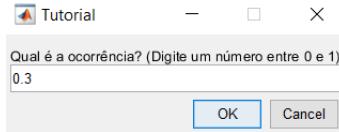


Figura 11: Ocorrência

De seguida vamos escolher o domínio filtragem onde, neste caso, selecionamos o domínio de frequência.



Figura 12: Domínio de filtragem

Desta feita, é nos pedido para digitarmos o tipo de filtro (escolhemos o *butterworth*).

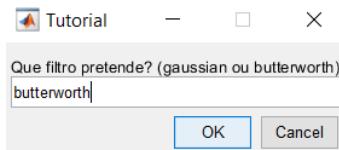


Figura 13: Tipo de filtro

Após termos escolhido o tipo de ruído, temos de fornecer as variáveis necessárias para o filtro escolhido. Neste caso teremos de fornecer duas variáveis e começamos por digitar 2 para o valor da ordem de *butterworth*.

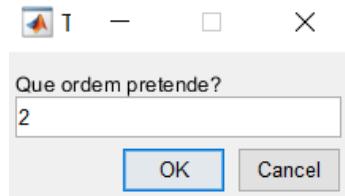


Figura 14: Ordem de Butterworth

E digitamos 4 para o valor da frequência de corte do filtro.

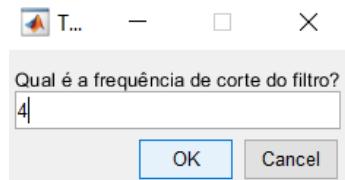


Figura 15: Frequência de corte

No fim de fornecermos todos os dados necessários é apresentado a seguinte imagem com os espetros pedidos no enunciado, isto é, o espetro da imagem original, o espetro da imagem com ruído e o espetro da imagem suavizada.

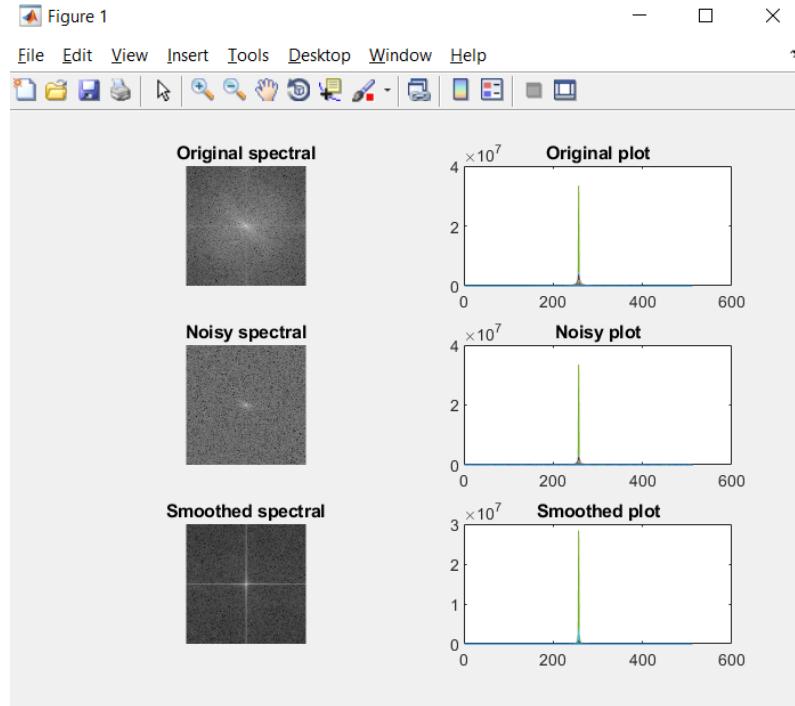


Figura 16: Espetros

Para além de se gerar a imagem anterior relativa aos espetros, também são guardadas, na pasta que selecionou no inicio, as imagens processadas, isto é, as seguintes quatro imagens.



Figura 17: Resultados obtidos

## 2.2 Canny Detector

Nesta secção irei falar sobre o programa para o algoritmo *Canny Detector*. *Canny* desenvolveu uma abordagem para criar um detetor de bordas ideal para lidar com bordas corrompidas por um ruído gaussiano. O algoritmo original de *Canny* consiste nas seguintes etapas:

- Calcular o gradiente horizontal  $\mathbf{G}_x$  e o gradiente vertical  $\mathbf{G}_y$  em cada localização do pixel, convertendo-se em máscaras de gradiente;
- Calcular a magnitude do gradiente  $\mathbf{G}$  e direção do ângulo de  $\theta$  de  $\mathbf{G}$  em cada localização de pixel;
- Aplicar o *Non Maximum Suppression* (NMS) às bordas finas. Este passo envolve calcular a direção do gradiente em cada pixel;
- Calcular os limites, alto e baixo, *thresholding* com base no histograma da magnitude do gradiente para toda a imagem;
- Executar o *hysteresis thresholding* para determinar o mapa de arestas;

Posto isto, após o cálculo dos diferentes gradientes, na aplicação do NMS, se a direção do gradiente do pixel for uma das oito possíveis direções principais, a magnitude do gradiente desse pixel será comparada com dois dos seus vizinhos imediatos ao longo da direção do gradiente e a magnitude do gradiente será definida como zero se não corresponder a um máximo local. Para as direções do gradiente que não coincidem com uma das 8 direções principais possíveis, é feita uma interpolação para calcular os gradientes vizinhos.

Desta feita, passamos para o cálculo dos limites da magnitude sabendo que o limite alto é calculado de forma que uma percentagem  $P_1$  do total de pixels da imagem seja classificada como borda forte. Por outras palavras, o limite alto corresponde ao ponto em que o valor da função de distribuição acumulativa de magnitude de gradiente (CDF) é igual a  $P_1$ . O limite baixo é calculado como uma percentagem  $P_2$  do limite alto. Os valores de  $P_1$  e  $P_2$  são tipicamente definidos como 20% e 40%, respectivamente.

Por fim, se a magnitude do gradiente de um pixel for maior que o limite alto, esse pixel será considerado como uma borda forte. Se a magnitude do gradiente de um pixel estiver entre o limite baixo e o limite alto, o pixel será rotulado como uma borda fraca. Bordas fortes podem ser incluídas imediatamente nas imagens finais da aresta. As bordas fracas são incluídas se e somente se estiverem conectadas a arestas fortes.

De seguida vemos, na imagem seguinte, um diagrama de fluxo do algoritmo de *Canny Edge Detection*.

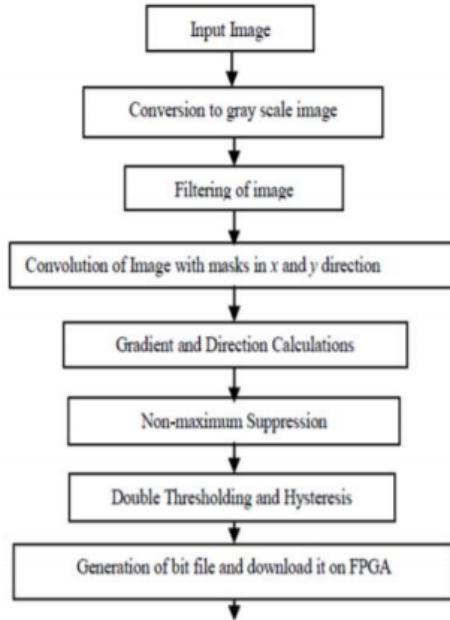


Figura 18: Fluxo do algoritmo de *Canny Edge Detection*

Neste algoritmo de deteção de bordas, o cálculo de gradiente é realizado usando máscaras de gradiente FIR (Finite-Impulse Response) projetadas para aproximar as derivadas parciais de uma função gaussiana, onde  $\sigma$  é o desvio padrão da função gaussiana.

O tamanho das máscaras de gradiente usadas pelo *Canny Edge Detection* é, geralmente, implementado em função do  $\sigma$  escolhido, com valores maiores de  $\sigma$  gerando máscaras maiores. No entanto, a melhor escolha de  $\sigma$  é dependente da imagem e pode ser selecionada pelo usuário com base no conhecimento das características atuais de ruído ou no tamanho dos objetos desejados na imagem.

### 2.2.1 Conversão para cinza

Imagem de entrada é uma imagem digital que é de diferentes formatos como .jpg, .bmp, .jpeg, .tiff, .gif, .png. Estas imagens são coloridas bem como imagens a preto e branco onde, posteriormente, são convertidas de RGB para cinza usando a função *rgb2gray*. De referir que em MATLAB, os valores de intensidade dos pixels são de 8 bits e variam de 0 a 255.

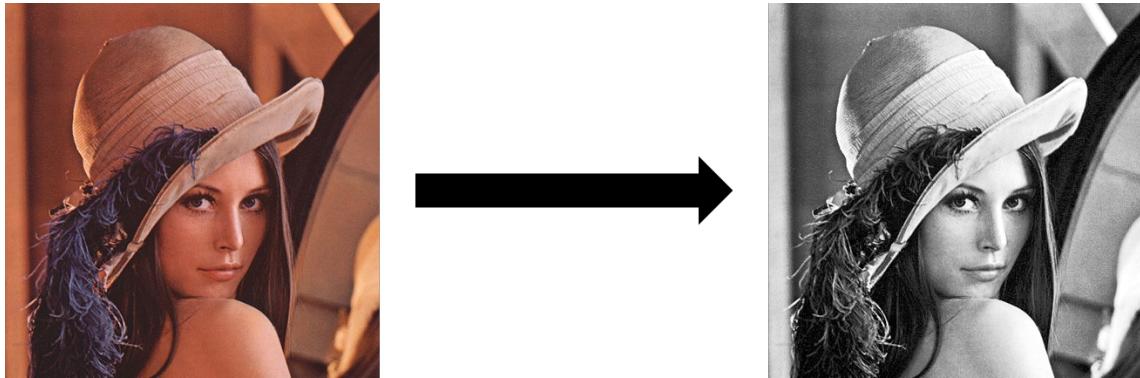


Figura 19: Conversão imagem original para imagem cinzenta

### 2.2.2 Determinar a intensidade dos Gradientes

O operador de *Sobel* executa uma medição de gradiente espacial 2D numa imagem e enfatiza regiões de alta frequência espacial que correspondem a arestas. Normalmente, ele é usado para localizar a magnitude do gradiente absoluto aproximado em cada ponto numa imagem em escala de cinza de entrada. Os gradientes podem ser determinados usando um filtro de *Sobel*, onde A é a imagem. Uma borda ocorre quando a cor de uma imagem muda, portanto, a intensidade do pixel muda também.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix} A, G_y = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix} A$$

Então, calcula-se a magnitude e o ângulo dos gradientes direcionais:

$$|G| = \sqrt{(G_x^2 + G_y^2)} \quad (1)$$

$$\angle(G) = \arctan(G_x/G_y) \quad (2)$$

### 2.2.3 NON MAXIMUM SUPPRESSION

A magnitude da imagem produzida resulta em bordas grossas. Idealmente, a imagem final deve ter bordas finas. Assim, devemos executar a *Non Maximum Suppression* para diminuir as arestas.

A *Non Maximum Suppression* funciona encontrando o pixel com o valor máximo numa borda. Na imagem acima, ocorre quando o pixel q tem uma intensidade maior que p e r, onde os pixels p e r são os pixels na direção do gradiente de q. Se esta condição é verdadeira, então nós mantemos o pixel, caso contrário, definimos o pixel para zero (tornando-o um pixel preto).

A *Non Maximum Suppression* pode ser obtida interpolando os pixels para uma maior precisão:

$$r = ab + (1 - \alpha)a \quad (3)$$

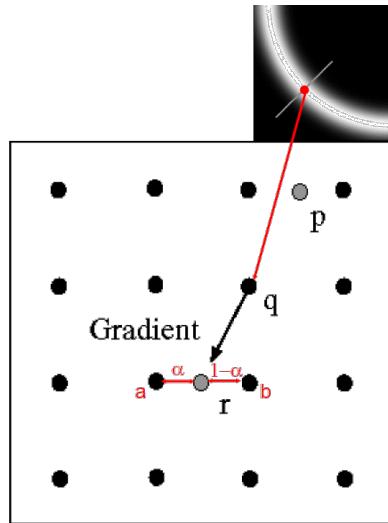


Figura 20: Interpolação do gradiente

A *Non Maximum Suppression* sem interpolação exige que dividamos a grade de pixels 3x3 em 8 secções. Se a direção do gradiente cair entre o ângulo de -22,5 e 22,5, então usaremos os pixels que caírem entre este ângulo (r e q) como o valor a ser comparado com o pixel p. Vejamos a seguinte imagem.

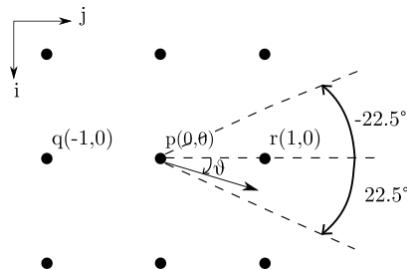


Figura 21: Sem interpolação

Este foi o modo utilizado por mim para este passo como podemos ver no algoritmo *nonmax.m*.

#### 2.2.4 DOUBLE THRESHOLDING

Percebemos que o resultado da *Non Maximum Suppression* não é perfeito, algumas arestas podem não ser bordas e há algum ruído na imagem. O *Double thresholding* irá tratar disso. Define-se dois limites, um limite alto e um baixo. Pixels com um valor alto são mais prováveis de serem arestas. Por exemplo, pode-se escolher o limite alto como 0,7, isso significa que todos os pixels com um valor maior que 0,7 serão uma borda forte. Você também pode escolher um limite baixo de 0,3, isso significa que todos os pixels menores do que 0,3 não são uma borda e seriam definidos como 0. Os valores entre 0,3 e 0,7 seriam bordas fracas, noutras palavras, nós não sabemos se estas são arestas reais ou não arestas de todo. O passo seguinte explicará como podemos determinar qual

borda fraca é uma borda real.

### 2.2.5 EDGE TRACKING BY HYSTERESIS

Agora que determinamos quais são as arestas fortes e as arestas fracas, é preciso determinar quais das arestas fracas são arestas reais. Para fazer isso, realizamos um algoritmo de rastreamento de borda.

As arestas finais são determinadas suprimindo todas as arestas que não estão conectadas a uma aresta forte. Bordas fortes podem ser imediatamente incluídas na imagem final da aresta. As bordas fracas são incluídas se e somente se estiverem conectadas a arestas fortes. É claro que a lógica é que o ruído e outras pequenas variações dificilmente resultarão numa borda forte com o ajuste adequado dos limites (superior e inferior). Assim, as arestas fortes serão arestas reais devido a serem arestas verdadeiras na imagem original. As bordas fracas podem ser arestas reais devido às variações de ruído e cor ou devido às bordas verdadeiras, isto é, bordas fracas conectadas a arestas fortes serão arestas reais.

### 2.2.6 Tutorial Canny Detector

Tal como o tutorial anterior, para iniciar este tutorial selecionamos a pasta onde estão os ficheiros (ou seja, a pasta "Problema\_2"), posteriormente selecionamos o ficheiro *CannyDetector.m* e é esse mesmo ficheiro que vamos correr, ou seja, após selecionarmos, clicamos em **RUN**. De salientar que consoante as escolhas do utilizador, os dados pedidos variam.

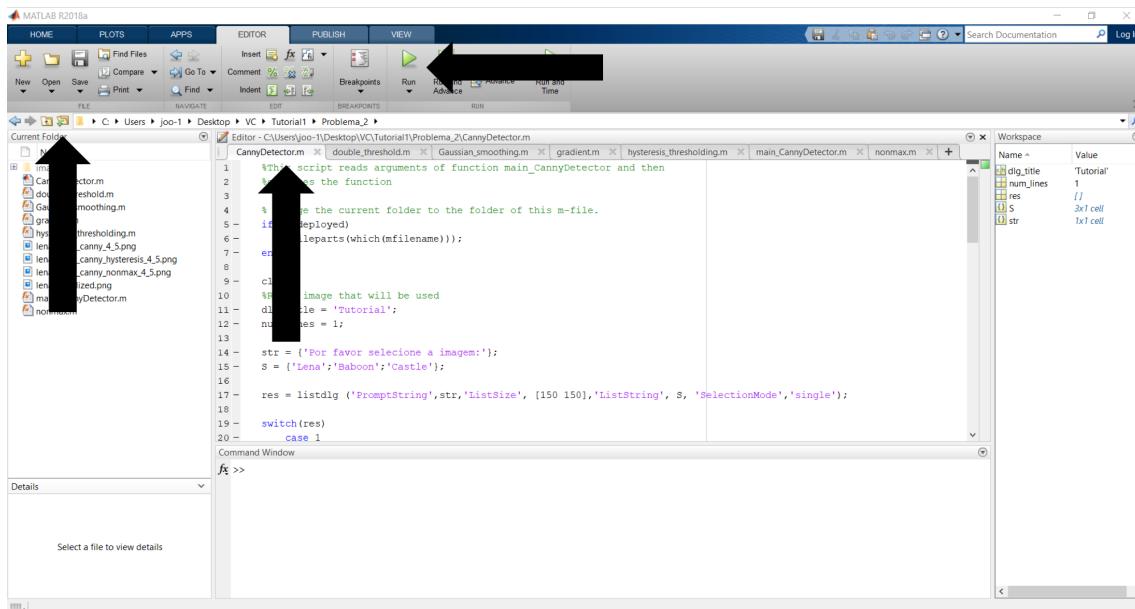


Figura 22: Iniciar

Quando o programa é iniciado será exibida uma lista com as imagens disponíveis e seleciona-se uma delas. Neste caso a imagem selecionada foi a Lena.

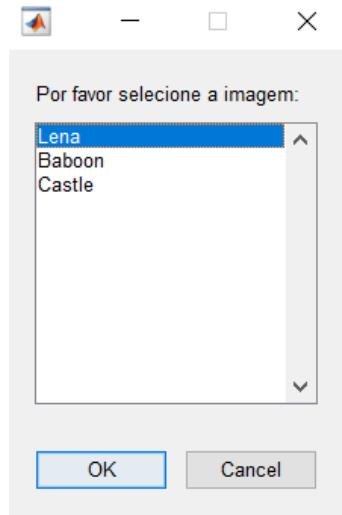


Figura 23: Seleção da imagem

Neste momento, é nos pedido para indicar uma variância para a aplicação do filtro gaussiano. Neste exemplo colocamos uma variância de 0.12.

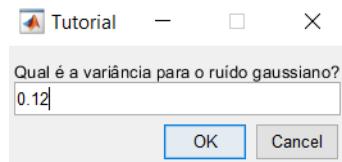


Figura 24: Seleção variância filtro gaussiano

Após isto é pedido para fornecer o tamanho da suavização gaussiana que pretendemos aplicar onde colocamos, neste exemplo, um tamanho de 4.

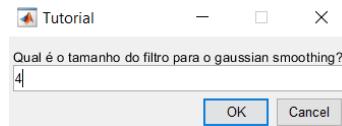


Figura 25: Seleção tamanho Gaussian Smoothing

De seguida, iremos indicar o valor que pretendemos para a variância da suavização gaussiana e, como podemos ver pela figura seguinte, o valor fornecido é 5.

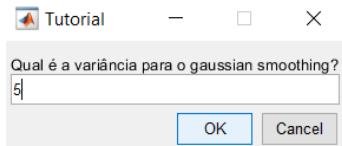


Figura 26: Seleção variância Gaussian Smoothing

Depois de fornecermos as informações anteriores são nos geradas 3 imagens com os gradientes direcionais e com a magnitude do gradiente.

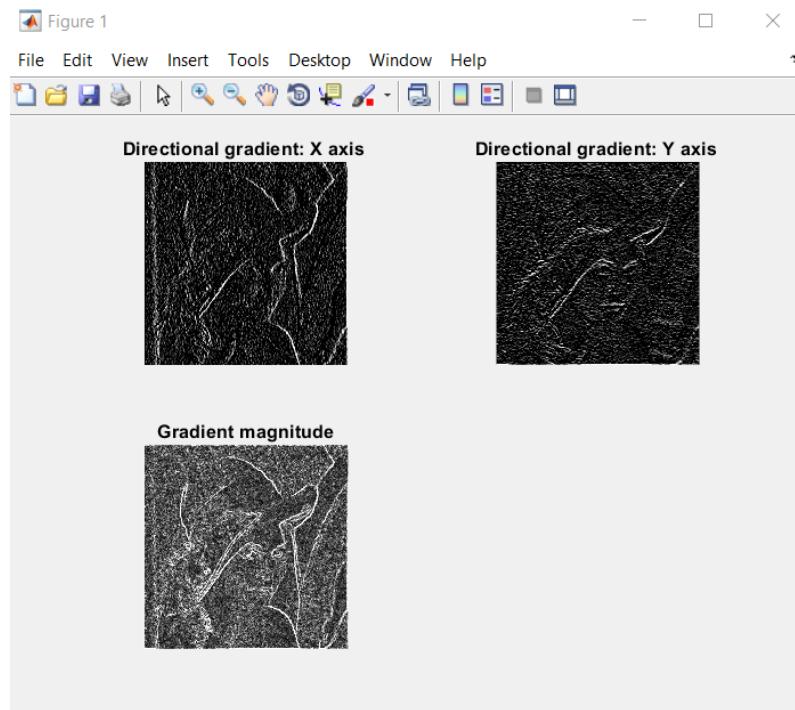


Figura 27: Imagens do Gradiente

Por último, é pedido os valores que pretendemos definir para os limites, superior e inferior, do *thresholding*

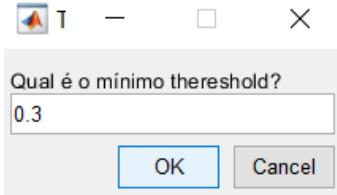


Figura 28: Seleção limite inferior thresholding

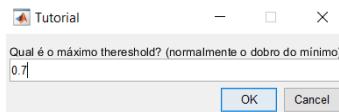


Figura 29: Seleção limite superior thresholding

No fim do programa, são guardadas, como *output* na pasta selecionada inicialmente, 4 imagens que são apresentadas na figura seguinte.

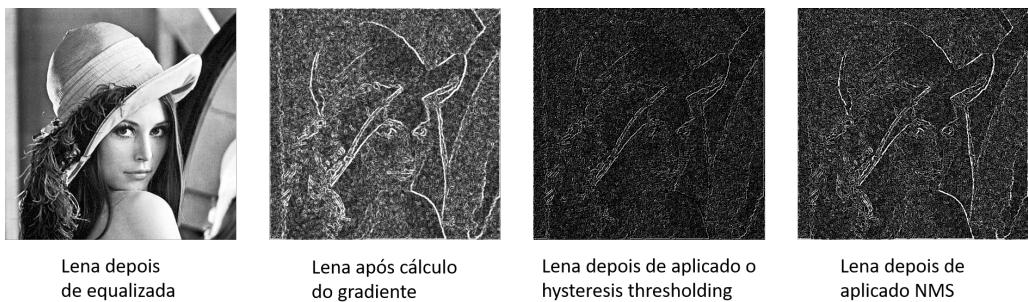


Figura 30: Resultados obtidos

### 3 Conclusão

Com este tutorial foi possível aprimorar os conhecimentos adquiridos relativamente ao processamento de imagens digitais bem como foi possível desenvolver competências na criação de algoritmos na linguagem **Matlab**.

Neste trabalho, existiram algumas dificuldades nomeadamente na criação de uma interface gráfica, na aplicação de teoremas como o *Fast Fourier Transform* ou em construir os algoritmos que aplicuem os filtros de suavização gaussiana ou *butterworth*.

Relativamente à construção do algoritmo para o *Canny Edge Detector*, que é considerado como ótima técnica de deteção de bordas, para além do que foi mencionado anteriormente, serviu para compreender melhor a suavização de imagens ruidosas.

Contudo, acho que o trabalho foi bem realizado e os resultados finais são bastante satisfatórios permitindo responder a todas as perguntas colocadas no enunciado.