

**Universidade do Minho**

Escola de Engenharia

Mestrado Integrado em Engenharia Informática

**Unidade Curricular de**

**Visão por Computador**

Ano Letivo de 2019/2020

**Tutorial-1 Image Filtering and Edge Detection**

**A78824 Mariana Lino Costa**

**A76387 André Ramalho**

Novembro,2019

**VC**

**Índice**

[Smooth Images 3](#_Toc25876829)

[Implementação do código 3](#_Toc25876830)

[Análise dos resultados 5](#_Toc25876831)

[Noise Salt&Pepper 5](#_Toc25876832)

[Noise Gaussian 5](#_Toc25876833)

[Spatial Domain 6](#_Toc25876834)

[DFT - Discrete Fast Fourier Transform 13](#_Toc25876835)

[Frequency Domain 14](#_Toc25876836)

[Canny Detector 24](#_Toc25876837)

[Implementação do código 24](#_Toc25876838)

[Output de imagens 28](#_Toc25876839)

# Smooth Images

## Implementação do código

Foi proposto a elaboração de um programa que aplica ruído a uma imagem, e depois tenta aplicar um filtro de forma a o corrigir. Para além disso, o programa também calcula o DFT de cada imagem. De forma a facilitar e simplificar todo o código do programa, procurou-se usar, sempre que possível, funções fornecidas pelo Matlab.

* **main\_smoothfilters.m**

Começou-se por perceber quais seriam os parâmetros de input e output da função. O programa **smoothfilters** vai receber uma imagem e devolver duas imagens transformadas, uma *noisy image* e uma *smoothed image*. De forma a o utilizador do programa poder ter opção de escolha do tipo de noise que quer aplicar na imagem ou mesmo alterar os diferentes campos para uma filtragem, foi necessário a implementação de uma função principal com vários inputs:

1. Uma imagem em escala preta e branca – F.
2. O tipo de noise que se quer aplicar: Salt & Pepper(SP) ou Gaussian(G) - type\_noise.
3. O noise\_param, parâmetro para se aplicar o ruído.
4. O filtro de domínio - filtering\_domain - para escolher o tipo de filtro de domínio que se quer utilizar, domínio espacial ou domínio de frequência, para isso basta escrever “spatial” ou “frequency”.
5. O tipo de alisamento - type\_smoothing - dependendo do filtro de domínio escolhido. Caso se escolha *Spatial*, o tipo de smoothing pode ser ***Gaussian***, ***Average*** ou ***Median***, e se for *Frequency* pode ser ***Gaussian*** ou ***Butterworth***.
6. O filter\_param que é um array de parâmetros para o filtro.
7. O butterType que é um parâmetro usado na função predefinida do Matlab *butter*, “l*ow*”, “*high*”, “*stop*” ou “*bandpass*”.

De forma a se organizar melhor o código, decidiu-se o dividir em 3 funções.

* **addNoise.m**

A função **addNoise** adiciona o ruído *Salt&Pepper* ou *Gaussian* à imagem e tem três inputs:

1. A Imagem a preto e branco.
2. O tipo de noise: *Salt & Pepper*(SP) ou *Gaussian*(G) - type\_noise.
3. E o parâmetro de ruído- noise\_param:
   1. No *Salt&Pepper*, o parâmetro representa a densidade, usada na função predefinida do Matlab: **imnoise**.
   2. No *Gaussian* o parâmetro é o sigma, usado na função predefinida do Matlab: **imnoise.**

* **smoothSpatial.m**

A função **smoothSpatial** aplica o filtro de domínio espacial e tem três inputs.

1. A Imagem já com o Noise aplicado.
2. O tipo de *smoothing*: **Gaussian, Average** ou **Median** - type\_smoothing.
3. E um *array* de parâmetros para cada filtro - filter\_param:
   1. No Gaussian, os parâmetros são o kernel/filterSize e o desvio/sigma, usados na função predefinida do Matlab:  **imgaussfilt.**
   2. No Average, o *array* apresenta apenas um parâmetro, o kernel.

Nota: O filtro **Median** foi feito através de uma função predefinida no matlab: **medfilt2**, onde só é passado um parâmetro: a imagem já com o ruído aplicado.

* **smoothFrequency.m**

A função **smoothFrequency** aplica o filtro de domínio de Frequência e tem quatro inputs.

1. A Imagem já com o Noise aplicado.
2. O tipo de smoothing: **Gaussian** ou **Butterworth** - type\_smoothing.
3. O array de parâmetros para cada filtro -filter\_param:
   1. No Gaussian, os parâmetros são o kernel/filterSize e o desvio/sigma, usados na função predefinida do Matlab: **imgaussfilt.**
   2. No Butterworth, os parâmetros são o *order* e *cutoff*, usados na função predefinida do Matlab: **butter.**
4. O butterType, que também é passado como parâmetro na função **butter** para se aplicar um filtro do tipo butterworth numa imagem. O butterType pode ser “*low*”, “*high*”, “*stop*” ou “*bandpass*”.

## 

## Análise dos resultados

### Noise Salt&Pepper

Uma imagem com terra, exterior

Descrição gerada automaticamenteUma imagem com primata, animal, mamífero, fotografia

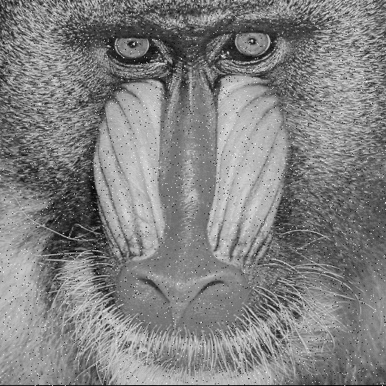
Descrição gerada automaticamenteImagem com aplicação de ruído de tipo ***Salt&Pepper*** com várias densidades.

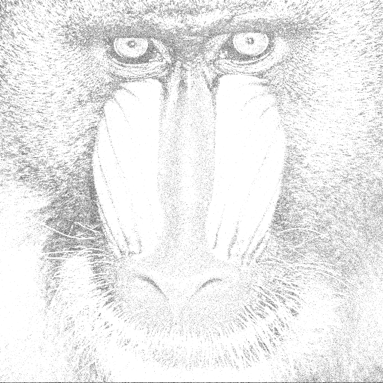
Figura 2 - Salt&Pepper, densidade=0.1

Figura 1 - Salt&Pepper, densidade=0.02

Figura 3 - Salt&Pepper, densidade = 0.5

### Noise Gaussian

Uma imagem com animal, primata, mamífero, olhar

Descrição gerada automaticamenteUma imagem com primata, animal, mamífero, macaco

Descrição gerada automaticamenteImagem com aplicação de ruído de tipo ***Gaussian*** com variação do valor do parâmetro de ruído.

Figura 4 – Gaussian com m=0.02

Figura 6 - Gaussian com m=0.5

Figura 5 - Gaussian com m=0.1

### 

### 

De forma a se analisar a transformação das imagens e perceber o objetivo da aplicação de cada filtro, decidiu-se averiguar também o DFT (*Discrete Fast Fourier Transform*) respetivo de cada imagem (da original, da ruidosa e por fim, da imagem filtrada).

### Spatial Domain

**Aplicação de filtro de *Spatial domain* numa imagem com ruído Salt&Pepper de 0.05 de densidade.**

* **Gaussian** 
  + Variação do *filterSize*

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| imagem original | Imagem com ruído Salt&Pepper  densidade = 0.05 | Filtro Gaussian com filterSize=3 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído Salt&Pepper  densidade = 0.05 | Filtro Gaussian com filterSize=5 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| imagem original | Imagem com ruído Salt&Pepper  densidade = 0.05 | Filtro Gaussian com filterSize=35 |

O filtro conseguiu disfarçar o ruído Salt&Pepper, no entanto a imagem foi ficando desfocada e com menos resolução, o que não é o esperado.

* **Average**
  + Variação do kernel

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído Salt&Pepper  densidade = 0.05 | Filtro Average, k=5 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído Salt&Pepper  densidade = 0.05 | Filtro Average, k=10 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | densidade = 0.05 | Filtro Average com k=20 |

Com o aumento do kernel, a imagem vai ficando com menos resolução, o ideal para tentar corrigir o ruído é usar um kernel relativamente baixo.

* **Median**

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | densidade= 0.05 | Filtro Median |

O filtro *median* é sem dúvida o filtro mais indicado para o tipo de ruído Salt&Pepper, podemos confirmar através do DFT de cada imagem. O dft da **median** é idêntico ao da original.

**Aplicação de filtro de *Spatial* *domain* numa imagem com ruído *Gaussian* com sigma=0.05.**

* ***Gaussian*** 
  + Variação do *filterSize*

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído  Gaussian  m = 0.05 | Filtro Gaussian com filterSize=3 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído  Gaussian  m = 0.05 | Filtro Gaussian com filterSize=5 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído  Gaussian  m = 0.05 | Filtro Gaussian com filterSize=35 |

O filtro *Gaussian* não conseguiu corrigir o ruído, obtém-se melhores resultados quando o filterSize é mais pequeno.

* ***Average***
  + Variação do *kernel*

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | imagem com ruído  *Gaussian*  m = 0.05 | Filtro *Average* com k=5 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | imagem com ruído  *Gaussian*  m = 0.05 | Filtro *Average* com k=10 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | imagem com ruído  Gaussian  m = 0.05 | Filtro Average com k=20 |

O Filtro Average não resolveu o ruido na imagem, e foi piorando como aumento do kernel pois a imagem começou a ficar muito desfocada.

* **Median**

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído  *Gaussian*  m = 0.05 | Filtro *Median* |

O filtro *median*, mais uma vez foi o mais indicado para se corrigir este tipo de noise, o *Discrete Fourier Transform* da imagem filtrada aproxima-se do da imagem original.

### DFT - Discrete Fast Fourier Transform

Para o cálculo do dft da imagem original, ruidosa e filtrada, começou-se por obter o Fourier *Transform* da imagem através da função predefinida do Matlab: **fft2**, de seguida tentou-se obter o centro do espectro com a função **fftshift** e finalmente aplicou-se a transformação do log. Com as imagens dos DFTs foi possível se fazer uma melhor análise das diferenças entre a imagem original, a com o ruido aplicado e a filtrada.

### 

### Frequency Domain

**Aplicação de filtro de *Frequency domain* numa imagem com ruído Salt&Pepper com densidade de 0.05.**

* ***Gaussian***
  + Variação do *filterSize*

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | imagem com ruído Salt&Pepper  densidade = 0.05 | Imagem com filtro Gaussian com filterSize=3 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído Salt&Pepper  densidade = 0.05 | Imagem com filtro Gaussian com *filterSize*=5 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído Salt&Pepper  densidade = 0.05 | Imagem com filtro Gaussian com *filterSize*=35 |

À medida que se aumenta o tamanho do *kernel*, a imagem vai ficando levemente mais desfocada, mas o filtro *Gaussian* no *frequency domain* é muito mais indicado para o ruído Salt&Pepper do que o filtro *Gaussian* no *spatial domain*.

* ***Butterworth***
  + Variação do *butterType*

O *butterType* pode ser *low, high, bandpass* ou *stop*. O *Low-Pass* desfoca a imagem, enquanto que o *High-Pass* torna a imagem mais nítida e com mais detalhe, mas em contrapartida reduz imenso o contraste da imagem. Por esse motivo optou-se por usar sempre o *butterType low,* e apenas analisar os resultados variando os parâmetros: *cutoff* e o *order.*

* + Variação do ***cutoff*** para o *Low*

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído Salt&Pepper  densidade = 0.05 | Imagem com butterType=low e cutoff=0.2 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído *Salt&Pepper*  densidade = 0.05 | Imagem com *butterType*=*low* e *cutoff*=0.5 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído *Salt&Pepper*  densidade = 0.05 | *Imagem* com *butterType*=*low* e *cutoff*=0.7 |

* + Variação do Order para o Low

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído *Salt&Pepper*  densidade = 0.05 | Imagem com *butterType=low* e *order*=2 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído Salt&Pepper  densidade = 0.05 | Imagem com butterType=low e order=10 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído Salt&Pepper  densidade = 0.05 | Imagem com butterType=low e order=20 |

Analisando as variações do *cutoff* e *order*, e também os dfts das imagens, verificou-se que a melhor forma de se usar o filtro butterWorth para uma imagem com ruído de Salt&Pepper, é usar um cutoff mediano (0.5) e um *order* baixo(2). O filtro *butterWorth* é o filtro mais adequado para este tipo de ruído.

**Aplicação de filtro de *Frequency domain* numa imagem com ruído Gaussian com sigma de 0.05.**

* **Gaussian**
  + Variação do filterSize

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído  Gaussian  m = 0.05 | Imagem com filtro Gaussian com filterSize=3 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído  Gaussian  m = 0.05 | Imagem com filtro Gaussian com filterSize=5 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído  Gaussian  m = 0.05 | Imagem com filtro Gaussian com filterSize=35 |

O filtro *Gaussian* funciona tão bem para o noise *Salt&Pepper* como para o *Gaussian*.

* **Butterworth**
  + Variação do ***cutoff*** para o Low

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído  *Gaussian*  m = 0.05 | Imagem com butterType=low e cutoff=0.2 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído  Gaussian  m = 0.05 | Imagem com butterType=low e cutoff=0.5 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído  Gaussian  m = 0.05 | Imagem com butterType=low e cutoff=0.7 |

* + Variação do Order para Low

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído  Gaussian  m = 0.05 | Imagem com butterType=low e order=2 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído  Gaussian  m = 0.05 | Imagem com butterType=low e order=10 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| Imagem original | Imagem com ruído  Gaussian  m = 0.05 | Imagem com butterType=low e order=20 |

O melhor filtro de domínio *Frequency* para o tipo de noise *Gaussian*, é o **butterworth** com um *cutoff* mediano (0.5) e um *order* baixo (2).

# 

# Canny Detector

## Implementação do código

* **CannyDetector.m**

A função **CannyDetector.m** chama a função principal, **main\_CannyDetector.m,** que vai ter como *input* uma imagem com ruído do tipo *Gaussian* aplicado e os parâmetros para se filtrar a imagem, o filterSize e o sigma. Esta função principal vai chamar todas as funções auxiliares de forma a seguir um algoritmo para detetar arestas usando um Canny Detector. Tem como output três imagens, a antes do “*nonmax suppression*” - BEFORE, a após o “*nonmax suppression*” - NM e a após o “*hysteresis thresholding*” - H.

O algoritmo começa por colocar um filtro *Gaussian* à imagem recebida como parâmetro, de seguida acha a magnitude e direção do gradiente, faz a supressão *nonmax*, acha as arestas fortes e fracas e finalmente faz o *hysteresis thresholding*, de forma a obtermos o terceiro resultado, as arestas que realmente importam na imagem.

* **AddNoiseG.m**

Esta função apenas coloca a imagem original a preto e branco e aplica um ruído do tipo *Gaussian* à imagem com a função **imnoise**.

* **Gaussian\_smoothing.m**

Esta função recebe como parâmetros a imagem com o ruído aplicado e os parâmetros para aplicar na função **imgaussfilt,** de forma a transformar a imagem com um filtro *Gaussian* do *Spacial Domain*.

Figura 7 - Imagem com o filtro Gaussian aplicado.

* **gradient.m**

A função **gradient** recebe a imagem e retorna a magnitude e direção do gradiente da imagem.

Usou-se a função predefinida do Matlab**imgradientxy** para se obter o gradiente na vertical e horizontal.

De seguida, com a função **imgradient,** passou-se como parâmetro, o gradiente vertical e horizontal obtidos anteriormente e calculou-se a magnitude e direção do gradiente.



Figura 8 - Imagem do gradiente.

* **nonmax.m e normalize\_directions.m**

A função **nonmax** tem o objetivo de melhorar os contornos da imagem, tornando-os mais finos e nítidos.

Começou-se por obter para cada pixel a orientação do gradiente:

* Se o ângulo estiver entre -180º e -157.5º ou entre -22.5º e 22.5º, comparou-se a força do pixel atual com os pixéis a “Este” e a “Oeste”
* Se o ângulo estiver entre 22.5º e 67.5º ou entre -112.5º e -157.5º, comparou-se a força do pixel atual com os pixéis a “Nordeste” e a “Sudoeste”
* Se o ângulo estiver entre 67.5º e 112.5º ou entre -67.5º e -112.5º, comparou-se a força do pixel atual com os pixéis a “Norte” e a “Sul”
* Se o ângulo estiver entre 112.5º e 157.5º ou entre -22.5º e -67.5º, comparou-se a força do pixel atual com os pixéis a “Noroeste” e a “Sudeste”

Caso a força dos contornos do pixel atual seja menor que a dos vizinhos, suprimime-se o valor, passando-o a zero.

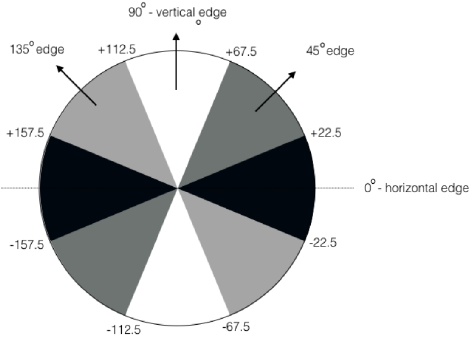
****

Figura 9 - Imagem após o nonmax suppression.

* **double\_threshold.m**

A função **double\_threshold** tem como objetivo dividir a imagememarestas fortes e arestas fracas. Ela recebe um limite alto (*high*) e um limite baixo (low), as arestas acima do limite alto são consideradas arestas fortes, as arestas que estão entre o limite baixo e o alto são as arestas fracas. Para isso usou-se a função predefinida **edge** que devolve todas as arestas acima do limite passado.

Conclui-se que para se achar as arestas fortes bastou fazer a **edge** para o limite superior. De forma a se descobrir também as arestas fracas, teve que se calcular a **edge** com o limite inferior e subtrair as arestas fortes.

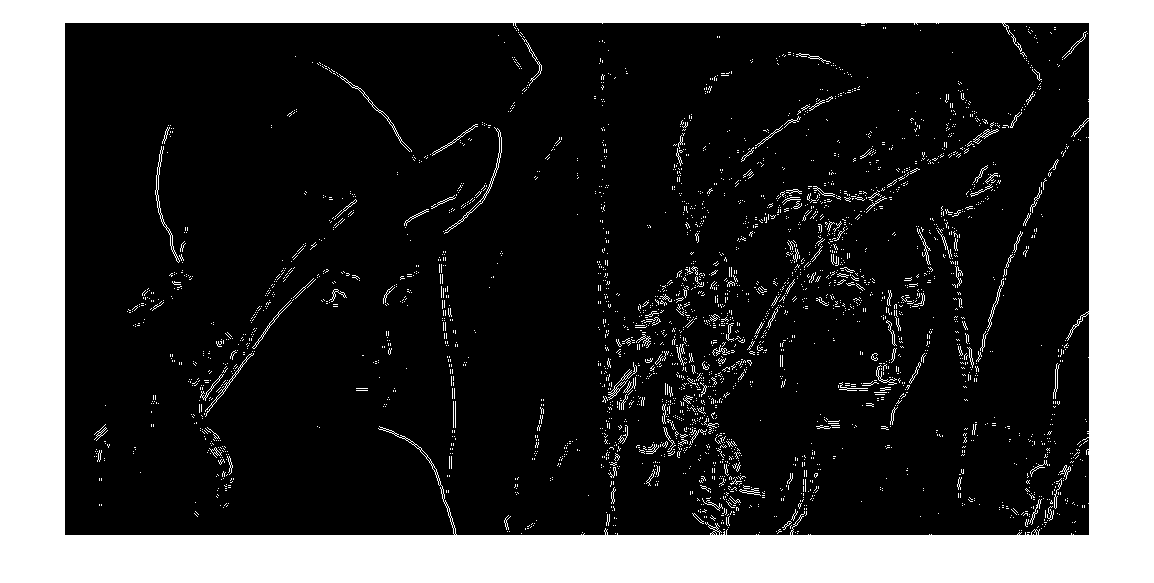


Figura 10 - Imagem após o double threshold, onde estão representadas as arestas fortes à esquerda e as fracas à direita.

* **hysteresis\_thresholding.m**

A função **hysteresis\_thresholding** tem como objetivo retirar ruido e outras variações numa imagem. Como as arestas fortes são arestas da imagem original e as arestas fracas tanto podem ser como podem ser apenas ruído, vai se suprimir todas as arestas fracas que não estão ligadas diretamente a arestas fortes.

Para isso, vai se verificar cada posição da matriz e, se o valor da posição no *strong* for 0 (cor preta) e no *weak* for 1(cor branca), vamos verificar todos os valores nas posições vizinhas da posição da matriz *strong*. Caso haja um valor na vizinhança igual a 1(branco), então o valor da posição no *strong* passa a ser 1(branco) também. Desta forma vai resultar uma imagem parecida com a obtida anteriormente das arestas *strong*, mas agora com os pontos *weak* mais próximos das arestas.

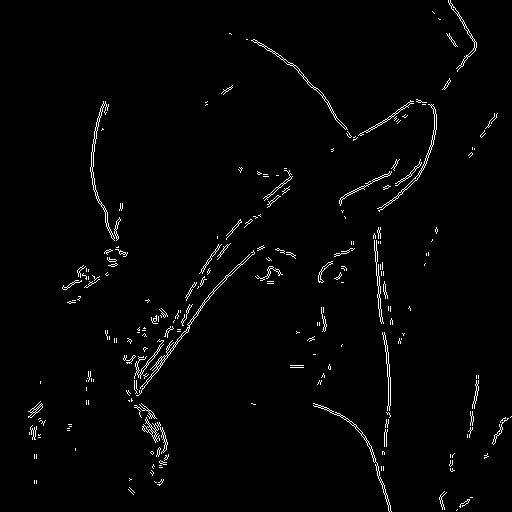


Figura 11 - Imagem após o hysteresis thresholding.

## 

## 

## Output de imagens

|  |  |  |
| --- | --- | --- |
| Gaussian Smoothing | Nonmax Suppression | Hysteresis thresholding |
|  |  |  |

### 

|  |  |  |
| --- | --- | --- |
| Gaussian Smoothing | Nonmax Suppression | Hysteresis thresholding |
|  |  |  |

### 

|  |  |  |
| --- | --- | --- |
| Gaussian Smoothing | Nonmax Suppression | Hysteresis thresholding |
|  |  |  |