Universidade do Minho
Escola de Engenharia

Image Filtering and Edge Detection

First Tutorial of Vision Computer

Tutorial submitted by:

Diogo Nogueira	Gonçalo Costa	Guilherme Martins
A78957	PG42839	A70782

Mestrado (Integrado) em Eng. Informática 2020/2021

Conteúdo

Smooth Images

Algoritmo

1. smoothfilters

Nesta função define-se o tipo de algoritmo de noise e o tipo de filtro de domínio que se pretende aplicar na imagem. Após se perceber os tipos de parâmetros de *input* e *output* da função, chama-se a função principal *Main_smoothfilters* passando como parâmetros:

- A imagem;
- O tipo de noise a aplicar;
- Os parâmetros para se aplicar o ruído;
- O tipo de filtro de domínio ou domínio de frequência a aplicar (S*patial* ou F*requency*);
- O tipo de S*moothing* a aplicar dependendo do tipo de filtro de domínio/domínio de frequência;
- O parâmetro *butterType* usado na função *butter*.

2. main_smoothfilters

Esta função principal vai receber os parâmetros e executar os algoritmos. Começa por transformar a imagem para escala de cinzas e de seguida adiciona o Noise à imagem chamando a função *addNoiseImage* que cria duas condições que verificam o tipo de Noise a aplicar.

Após a aplicação de Noise, adiciona o tipo de *Smoothing* na imagem a partir do filtro recebido como parâmetro. No caso de o filtro ser do tipo *Spatial*, é chamada a função *spatialDomain* que a partir de três condições verifica o tipo de filtro de *Smoothing* a aplicar. No caso de o filtro ser do tipo *Frequency*, é chamada a função *frequencyDomain* que verifica também o tipo de filtro de *Smoothing* a aplicar a partir de 2 condições.

Finalmente, após a aplicação de filtros de *Smoothing* à imagem, é calculado o DFT (*Discrete Fast Fourier Transform*) da imagem original, da imagem com Noise aplicado e da imagem após a aplicação dos filtros.

Interpretação e Análise dos Resultados

Interpretação dos Resultados

Para simplificar a Análise dos Resultados obtidos, a ideia passou por organizar o *Script* "smoothfilters.m" consoante o tipo de Testes pensados para o programa, fornecendo assim uma ajuda imperativa na redação e esquematização desses mesmos *outputs*.

Assim, e com base nos *Inputs* e *Algorithms* pedidos no Enunciado deste Trabalho Prático, criaram-se os seguintes tipo de Testes, que se encontram divididos consoante o tipo de Imagens que se pretende gerar:

1. *Noisy Images*

Tipo de Ruído	Noise *Salt & Pepper*	Noise *Gaussian*
Parâmetro a Variar	Valor da Densidade (d)	Valor da Média Ruído Gaussiano (m)
Valores Usados	d = 0.05	m = 0.05
	d = 0.1	m = 0.1
	d = 0.5	m = 0.5
Fixando o valor de m = 0.1 para o *Noise Gaussian*		
Parâmetro a Variar		Valor da Variância Ruído Gaussiano (var_gauss)
Valores Usados		var_gauss = 0.1
		var_gauss = 0.5

- Para a introdução do Ruído em Modo *Salt & Pepper*, e uma vez que foi usada a função imnoise do MATLAB, fez-se apenas variar o valor da Densidade do Ruído;
- Para o Modo *Gaussian*, e também usando a mesma função, aproveitaram-se os diferentes parâmetros que essa função aceita e além de se fazer variar o valor da Média do Ruído Gaussiano, fixou-se um desses valores e alterou-se a Variância desse mesmo Ruído.

2. *Smoothed Images*

2.1. Domínio Filtragem *Spatial*

Tipo de Ruído	Noise *Salt & Pepper*	Noise *Gaussian*
Fixando o valor de d = 0.1 e m = 0.1 para o *Noise Salt & Pepper* e *Noise Gaussian*, respetivamente		
Tipo de Filtro	*Average*	
Parâmetro a Variar	Valor do *Kernel* (k)	
Valores Usados	k = 5	
	k = 10	
	k = 20	
Tipo de Filtro	*Gaussian*	
Parâmetro a Variar	Valor do *FilterSize* (Tem de ser um ODD)	
Valores Usados	filterSize = 3	
	filterSize = 7	
	filterSize = 21	
Tipo de Filtro	*Median*	
Parâmetro a Variar	Sem Variação de Parâmetros	

- Para a aplicação do efeito *Smooth* nas várias Imagens através dos diferentes algoritmos integrados nos vários Filtros de Domínios, teve de existir uma distinção daqueles que fazem parte do Domínio de Filtragem em questão;
- Teve de se fixar os valores de d e m, de modo a gerar as imagens com os respetivos Tipos de Ruído e, posteriormente, aplicar o Filtro correspondente;
- Os Parâmetros a Variar são equivalentes para os dois Tipos de Ruído.

2.2. Domínio Filtragem *Frequency*

Tipo de Ruído	Noise *Salt & Pepper*	Noise *Gaussian*
Fixando o valor de d = 0.1 e m = 0.1 para o *Noise Salt & Pepper* e *Noise Gaussian*, respetivamente		
Tipo de Filtro	*Gaussian*	
Parâmetro a Variar	Valor do *Filter Size* (Tem de ser um ODD)	
Valores Usados	filterSize = 3	
	filterSize = 7	
	filterSize = 21	
Tipo de Filtro	*ButterWorth*	
Parâmetro a Variar	Valor do *Filter Order*	
Valores Usados	filterOrder = 5	
	filterOrder = 10	
	filterOrder = 20	
Parâmetro a Variar	Valor do *Cutoff*	
Valores Usados	cutOff = 0.05	
	cutOff = 0.3	
	cutOff = 0.5	

- No Domínio de Filtragem *Frequency*, e tendo em conta que os Tipos de Filtro possíveis a usar variam do anterior, também os Parâmetros sofreram alterações;
- Tal como acontece no Domínio anterior, as diferentes variações aplicam-se às Imagens geradas com os dois Tipos de Ruído.

Análise dos Resultados

Para simplificar a Análise dos Resultados e ir de encontro ao que é solicitado no Enunciado deste Trabalho Prático, fez-se a criação dos *Discrete Fast Fourier Transform* referentes a todo o processo de Adição de Ruído e subsequente Aplicação Filtro de *Smooth*.

Estes DFT's serão anexados na secção das *Smoothed Images*.

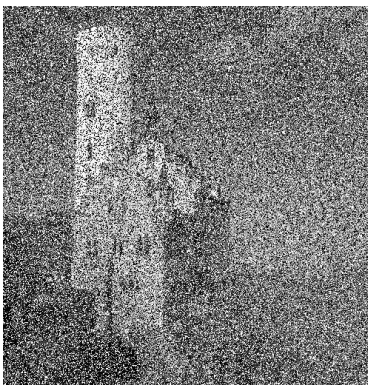
Dessa forma, enrique-se toda a análise e consegue-se validar determinadas observações visíveis a olho nu pelas próprias Imagens em si.

Noisy Images

1. *Salt & Pepper Noise*

Densidade Ruído (d) = 0.05

Densidade Ruído (d) = 0.1



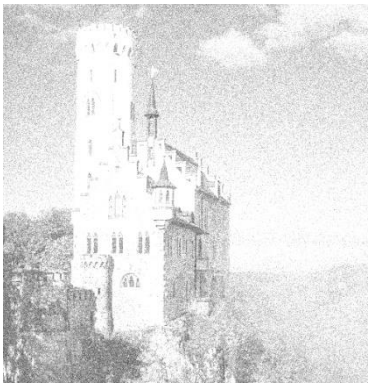
Densidade Ruído (d) = 0.5

- Como seria de esperar, à medida que o Valor da Densidade do Ruído aumenta, a imagem apresenta um teor de ruído maior;
- Constata-se que os vários "pontinhos" que criam toda a visão do Ruído criam uma espécie de *Salt* e *Pepper*, dado que uns apresentam o preto, e outros apresentam o branco.

2. *Gaussian Noise*

Média Ruído Gaussiano (m) = 0.05

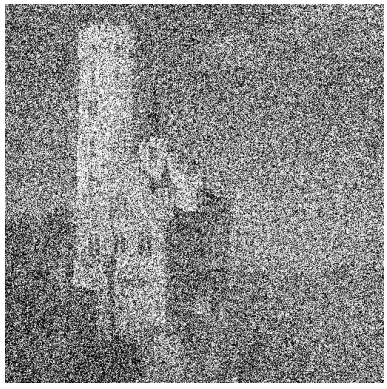
Média Ruído Gaussiano (m) = 0.1



Média Ruído Gaussiano (m) = 0.5

Fixando o valor Da Média do Ruído Gaussiano em 0.1, podem ser explorados/testados diferentes valores para o Parâmetro *Gaussian Noise Variance*.

Média Ruído Gaussiano (m) = 0.05 *Gaussian Noise Variance* (var_gauss) = 0.1



Média Ruído Gaussiano (m) = 0.05 *Gaussian Noise Variance* (var_gauss) = 0.5

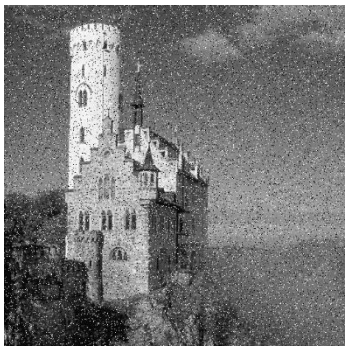
- Observa-se que à medida que o valor da **Média do Ruído Gaussiano aumenta**, **aumenta** também com ele o **brilho da imagem**;
- Mantendo-se o valor dessa Média num valor aceitável, e **aumentando-se** apenas o valor do ***Gaussian Noise Variance***, **aumenta** com ele o teor de **Ruído da Imagem**.

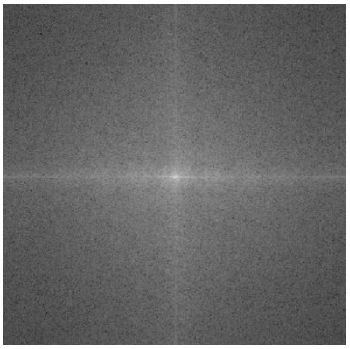
Smoothed Images

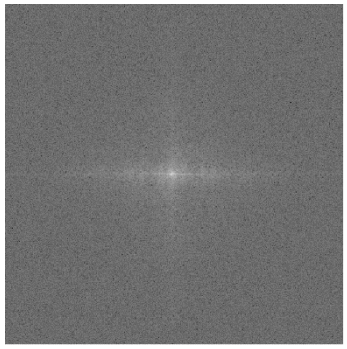
1. Domínio de Filtragem *Spatial*

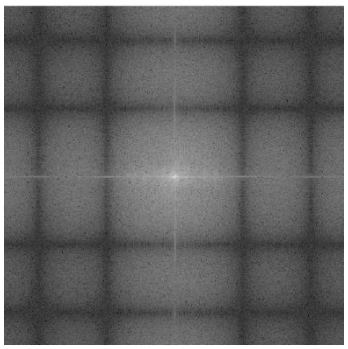
1.1.1. *Salt & Pepper Noise*

1.1.1.1. Filtro *Average* (Variação Valor *Kernel*)





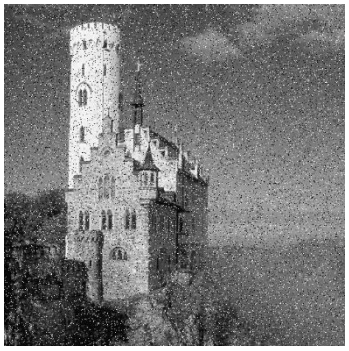


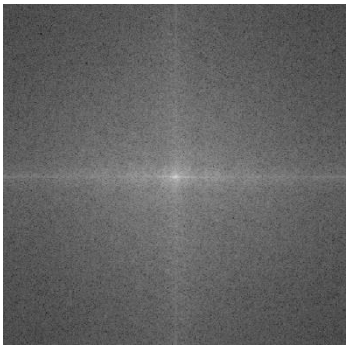
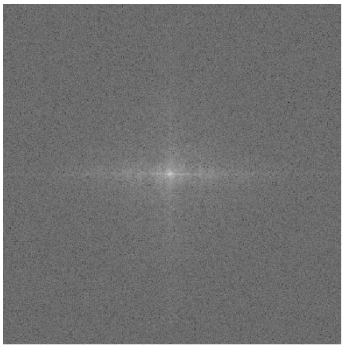
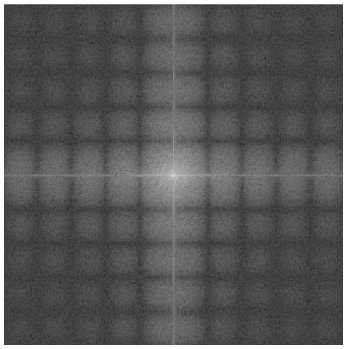


DFT Imagem Original

DFT Imagem Ruído *Salt & Pepper*

DFT Imagem Filtrada *Spatial Average* em que Kernel (k) = 5

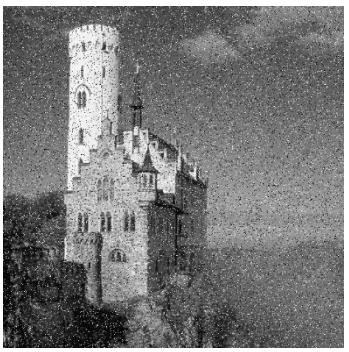



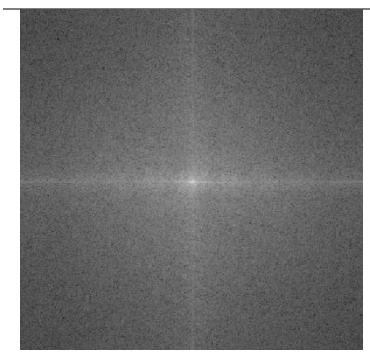
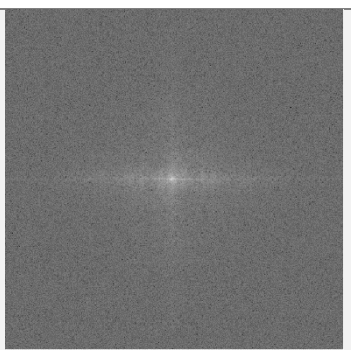
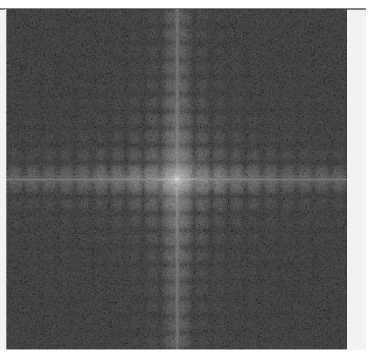




DFT Imagem Original

DFT Imagem Ruído *Salt & Pepper*

DFT Imagem Filtrada *Spatial Average* em que Kernel (k) = 10

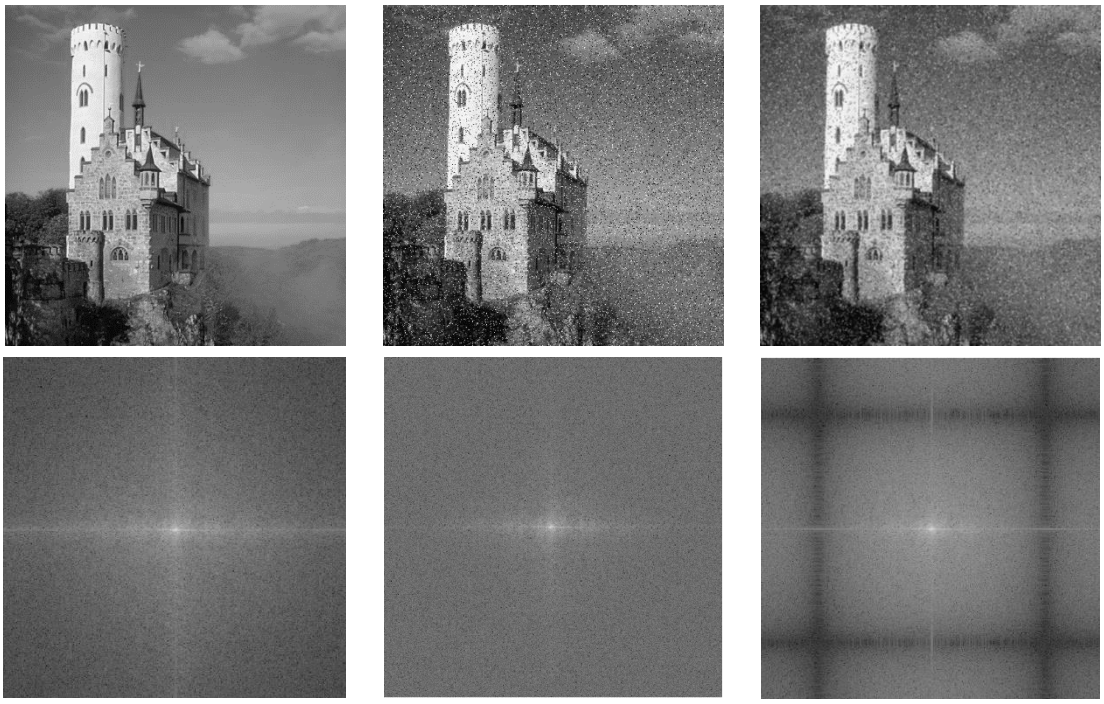
DFT Imagem Original

DFT Imagem Ruído *Salt & Pepper*

DFT Imagem Filtrada *Spatial Average* em que Kernel (k) = 15

- Através da análise da evolução das Imagens e respetivos DFT's nota-se que à medida que o valor do ***Kernel*** aumenta, a **resolução da Imagem diminui**;
- Deduz-se assim que o ideal para reduzir este ruído, concorrendo ao Filtro *Average* seja diminuir ao valor do *Kernel*, tentando encontrar assim um valor ideal.

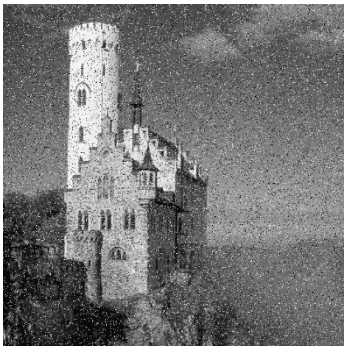
1.1.1.2. Filtro *Gaussian* (Variação Valor *Filter Size*)

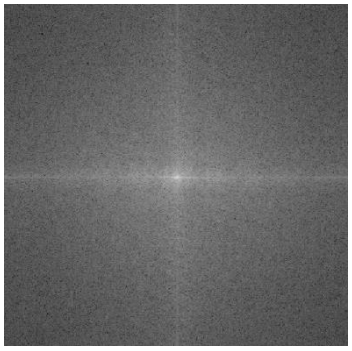
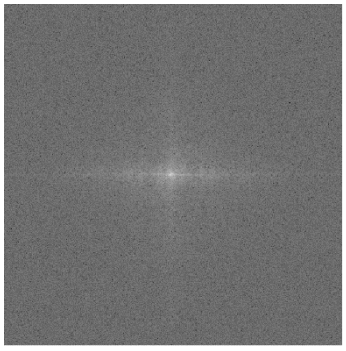
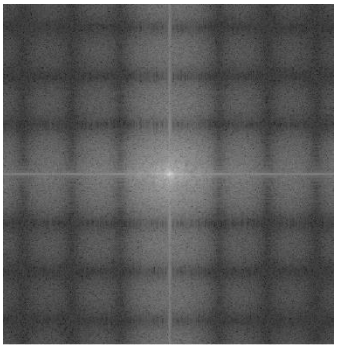


DFT Imagem Original

DFT Imagem Ruído *Salt & Pepper*

DFT Imagem Filtrada *Spatial Gaussian* em que **filterSize** = 3

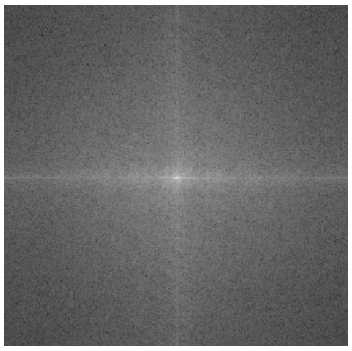
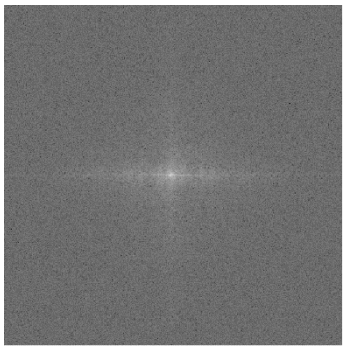
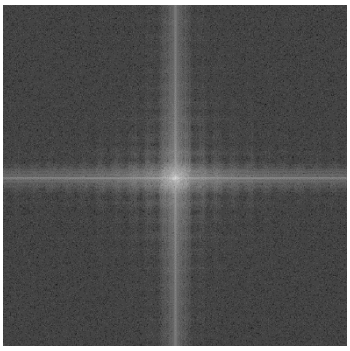




DFT Imagem Original

DFT Imagem Ruído *Salt & Pepper*

DFT Imagem Filtrada *Spatial Gaussian* em que **filterSize** = 7



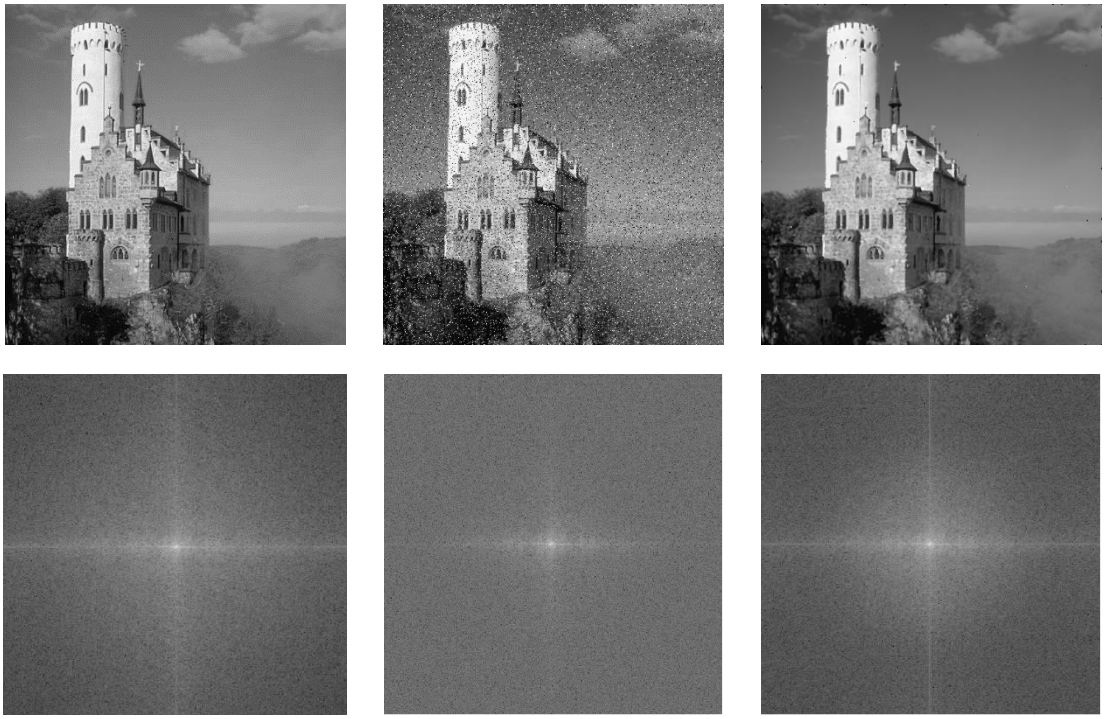
DFT Imagem Original

DFT Imagem Ruído *Salt & Pepper*

DFT Imagem Filtrada *Spatial Gaussian* em que **filterSize** = 21

- Existe uma eficácia em reduzir o Ruído *Salt & Pepper*;
- Apesar desta eficácia, a Imagem dada como *output* torna-se demasiado desfocada e com pouquíssima resolução, o que torna esta solução não tão eficiente como seria objetivado.

1.1.1.3. Filtro *Median*

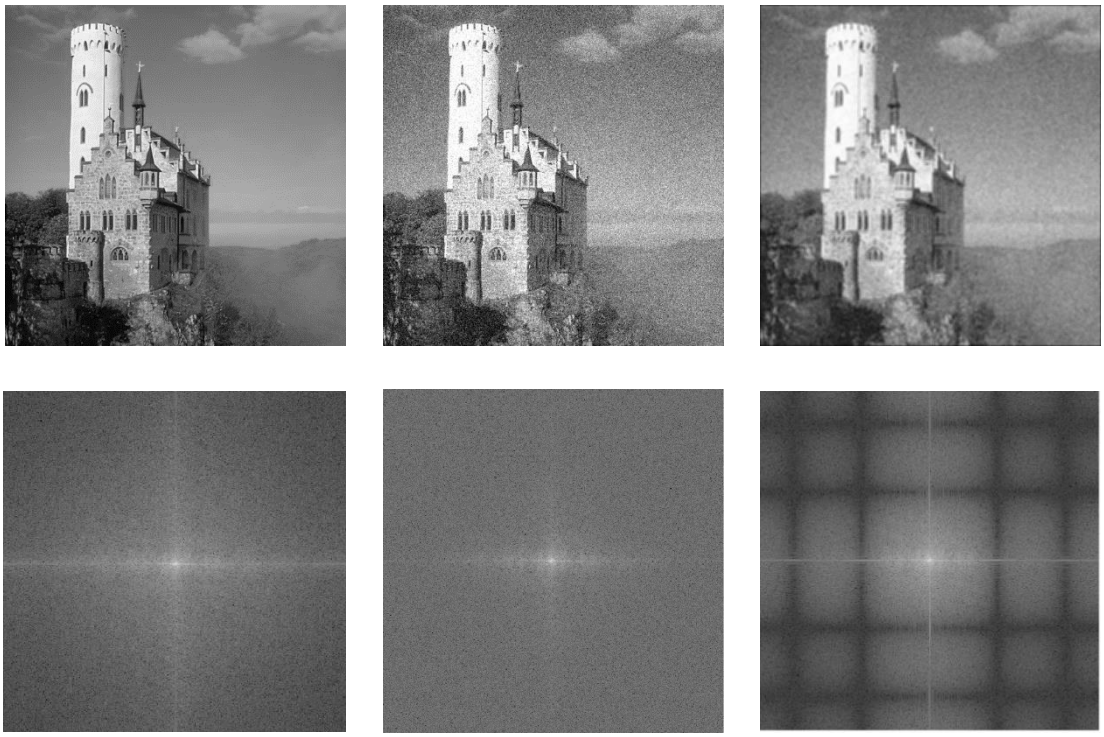


DFT Imagem Original | DFT Imagem Ruído *Salt & Pepper* | DFT Imagem Filtrada Spatial Median

- O Filtro *Median* revela-se assim o **filtro mais eficaz** dos três apresentados;
- Isso é observável pelo **DFT Original** e pelo **DFT Imagem Filtrada *Spatial Median***.
    - Os DFT's são muito similares.

1.1.2. *Gaussian Noise*

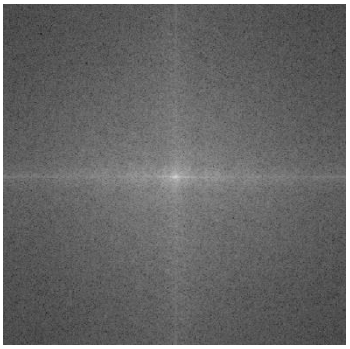
1.1.2.1. Filtro *Average* (Variação Valor *Kernel*)



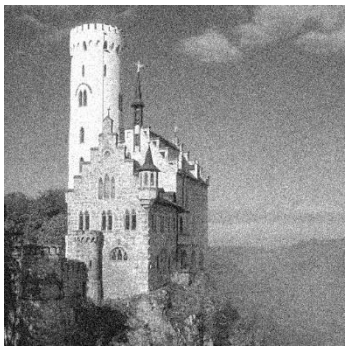
DFT Imagem Original

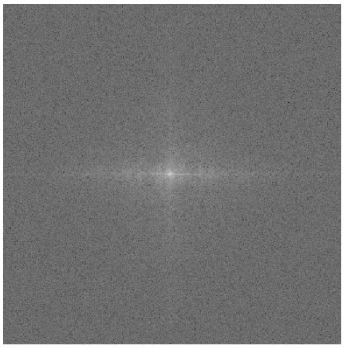
DFT Imagem Ruído *Gaussian*

DFT Imagem Filtrada *Spatial Average* em que Kernel (k) = 5

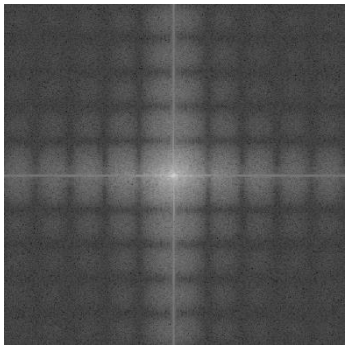


DFT Imagem Original

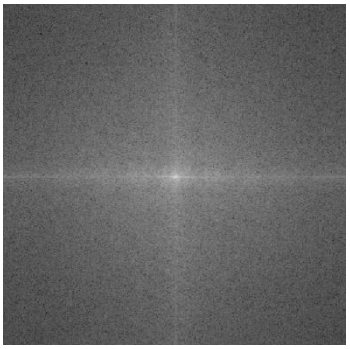




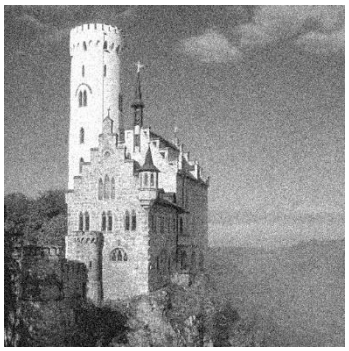
DFT Imagem Ruído
Gaussian

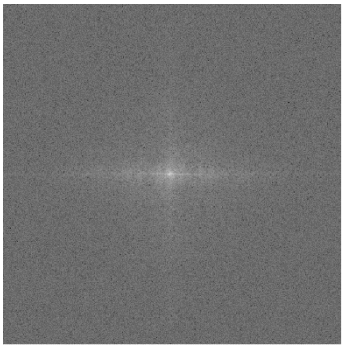


DFT Imagem Filtrada
Spatial Average
em que **Kernel (k)** = 10

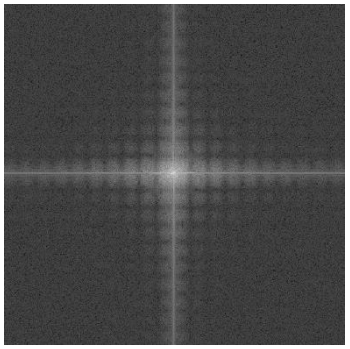


DFT Imagem Original





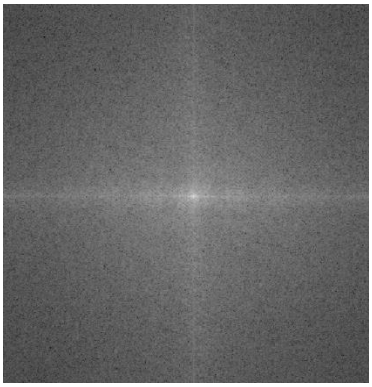
DFT Imagem Ruído
Gaussian

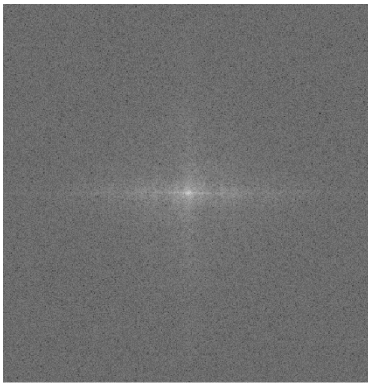


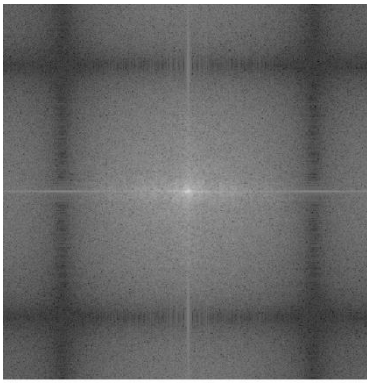
DFT Imagem Filtrada
Spatial Average
em que **Kernel (k)** = 20

- Há uma piora significativa em termos de redução do Ruído aquando do aumento do valor do *Kernel;*
- A Image perde (quase) na totalidade a sua resolução.

1.1.2.2. Filtro *Gaussian* (Variação Valor *Filter Size*)



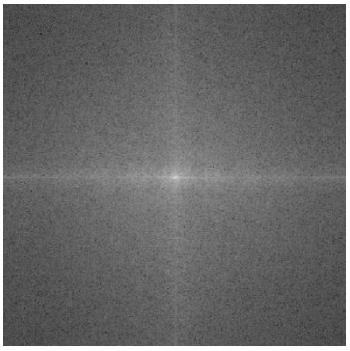




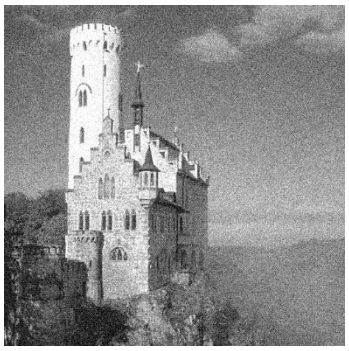
DFT Imagem Original

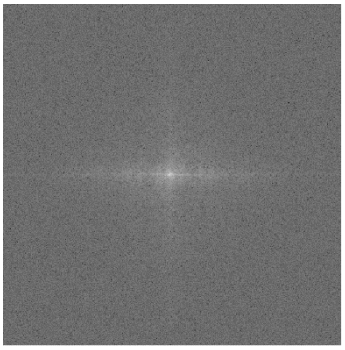
DFT Imagem Ruído *Gaussian*

DFT Imagem Filtrada *Spatial Gaussian* em que filterSize = 3

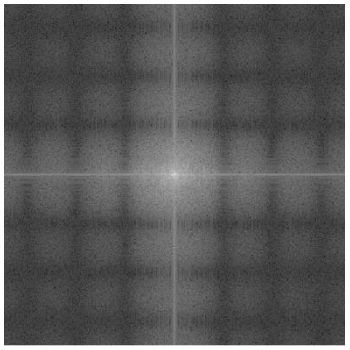


DFT Imagem Original

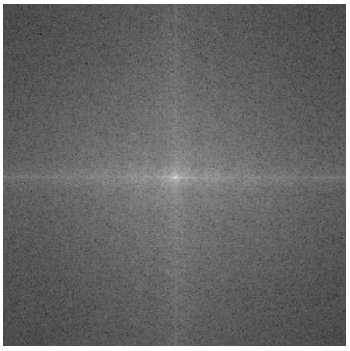




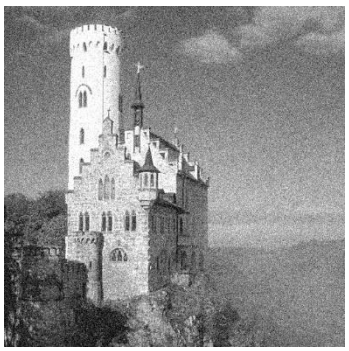
DFT Imagem Ruído
Gaussian

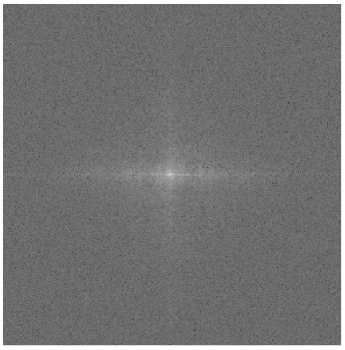


DFT Imagem Filtrada
Spatial Gaussian
em que **filterSize** = 7

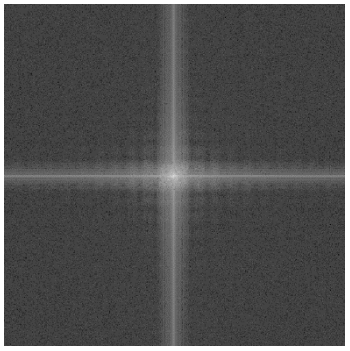


DFT Imagem Original





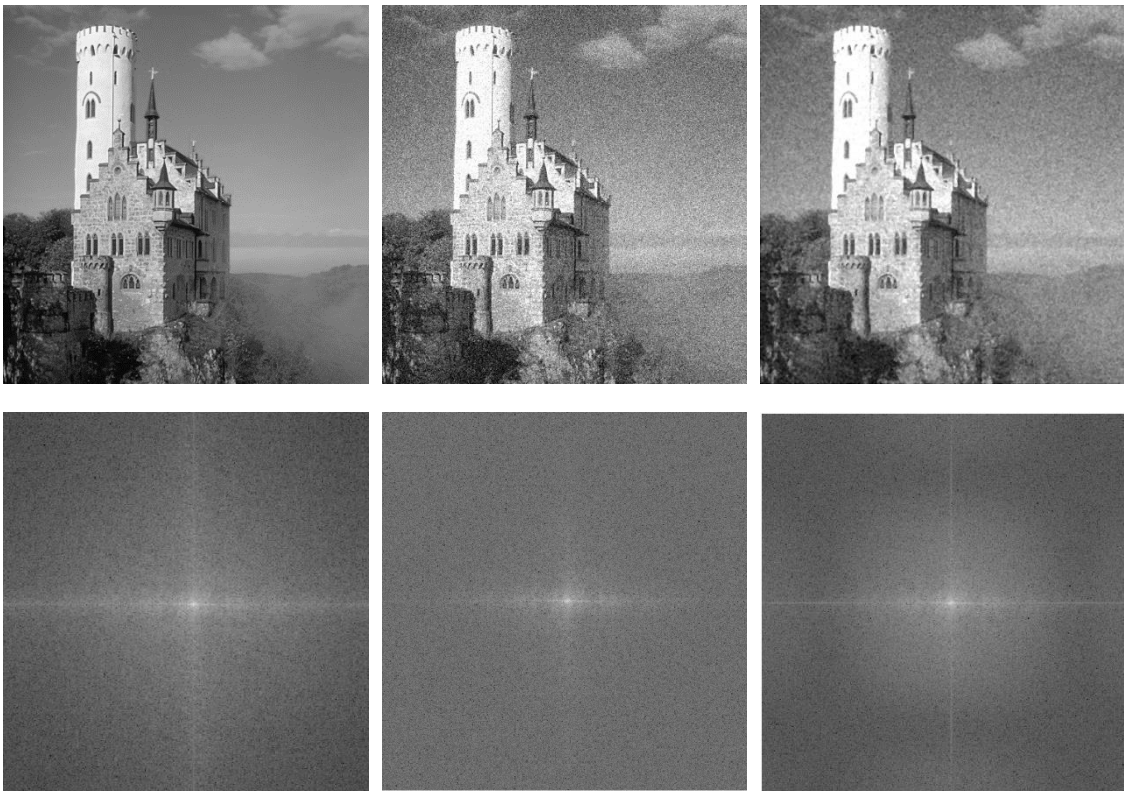
DFT Imagem Ruído
Gaussian



DFT Imagem Filtrada
Spatial Gaussian
em que **filterSize** = 21

- Resultados similares aos anteriores;
- Há uma ligeira melhoria percetível também pelo DFT da Imagem Filtrada.

1.1.2.3. Filtro *Median*



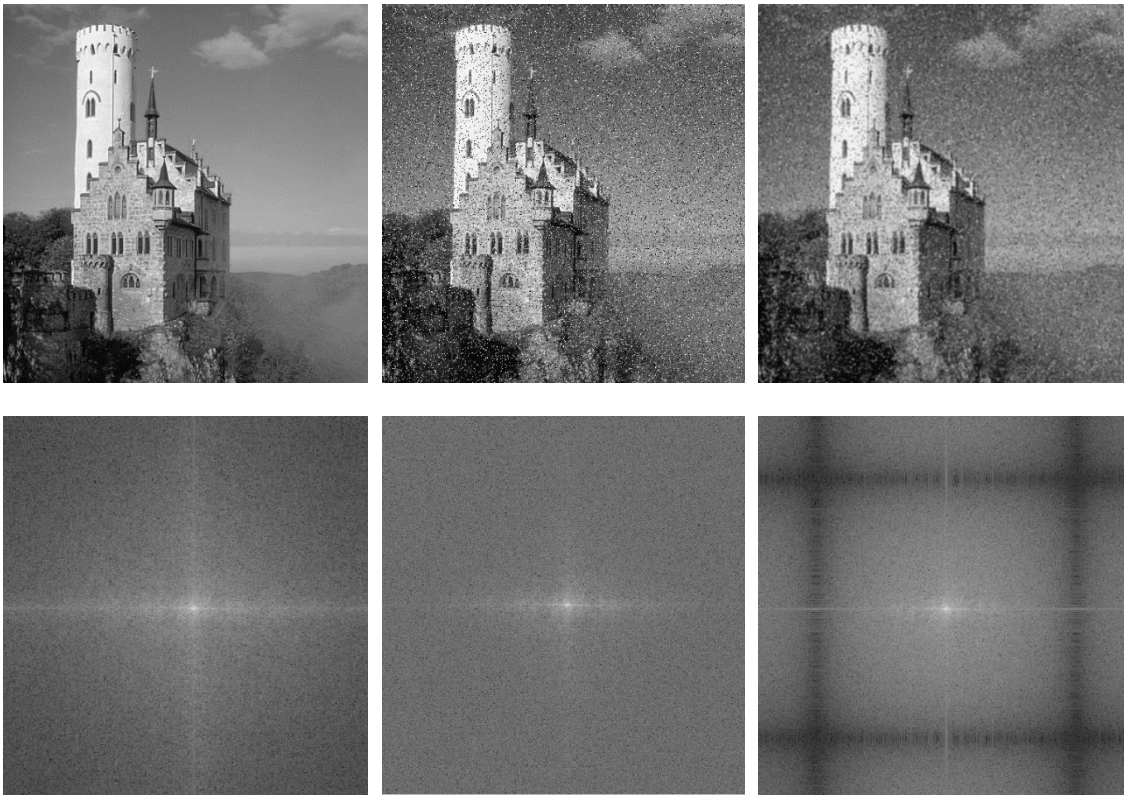
DFT Imagem Original | DFT Imagem Ruído *Gaussian* | DFT Imagem Filtrada Spatial Median

- Igualmente ao que acontece com as Imagens com Ruído *Salt & Pepper*, o Filtro *Median* torna a ser o que melhor resultado apresenta;
- Resultado muito similar à Imagem/DFT Original.

2. Domínio de Filtragem *Frequency*

2.1.1. *Salt & Pepper Noise*

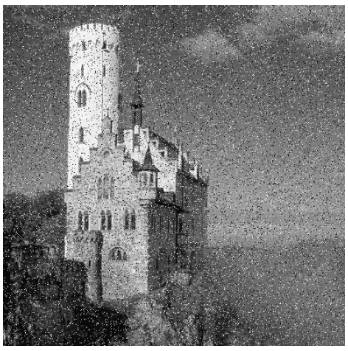
2.1.1.1. Filtro *Gaussian* (Variação Valor *Filter Size*)

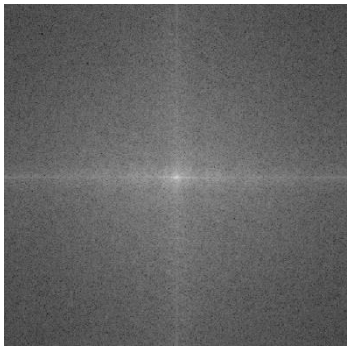
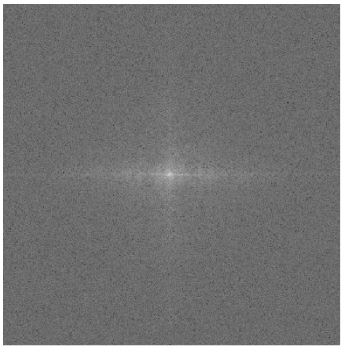
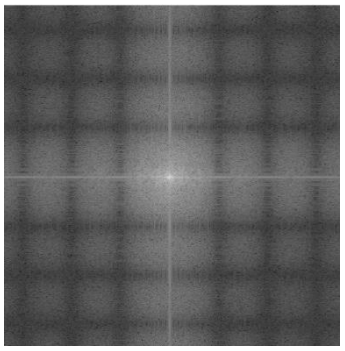


DFT Imagem Original

DFT Imagem Ruído *Salt & Pepper*

DFT Imagem Filtrada *Frequency Gaussian* em que **filterSize** = 3

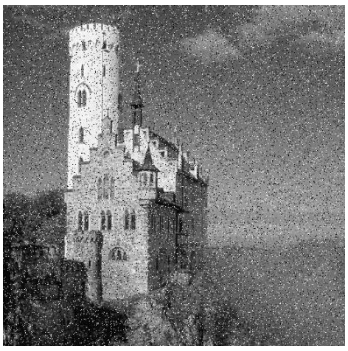



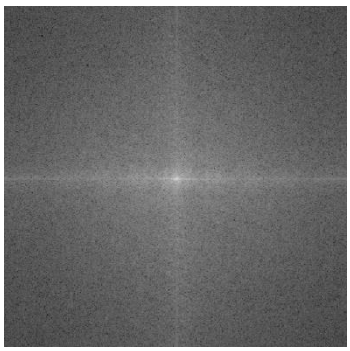
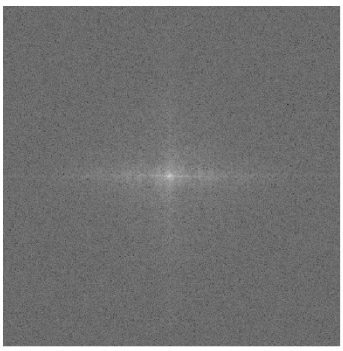
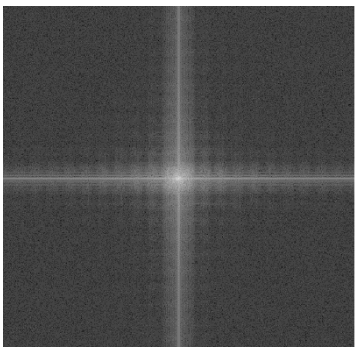




DFT Imagem Original

DFT Imagem Ruído *Salt & Pepper*

DFT Imagem Filtrada *Frequency Gaussian* em que **filterSize** = 7

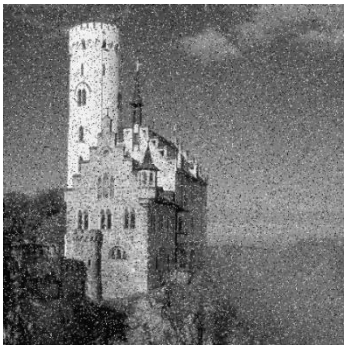
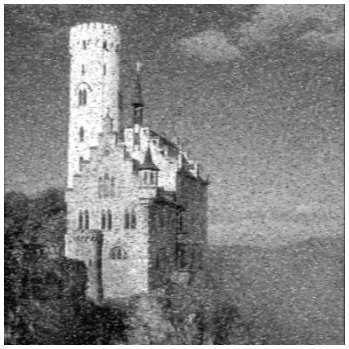
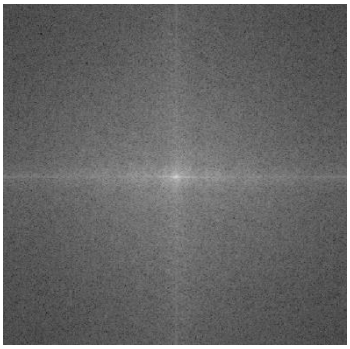
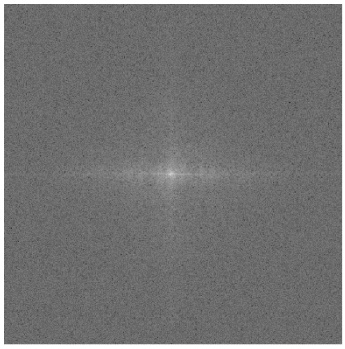
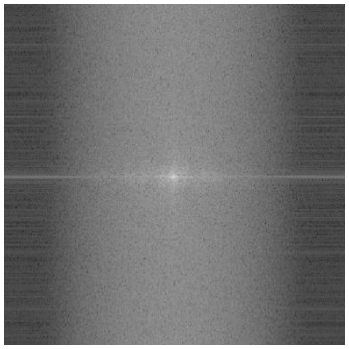




DFT Imagem Original

DFT Imagem Ruído *Salt & Pepper*

DFT Imagem Filtrada *Frequency Gaussian* em que **filterSize** = 21

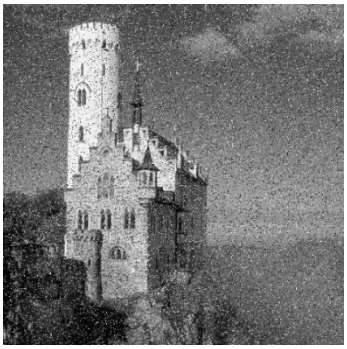
2.1.1.2. Filtro *ButterWorth* (Variação *Filter Order*)

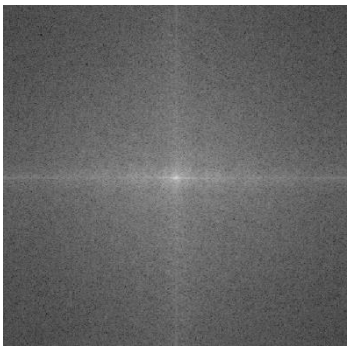
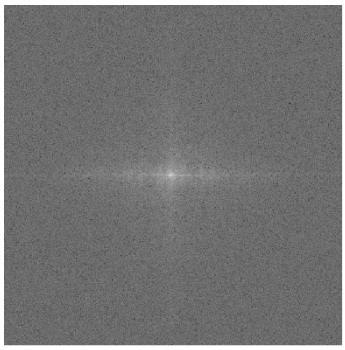
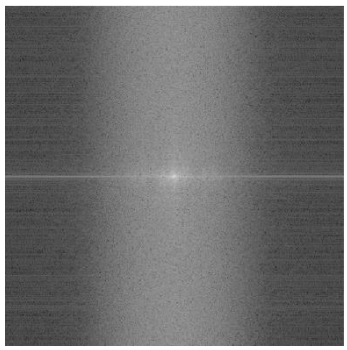







DFT Imagem Original

DFT Imagem Ruído *Salt & Pepper*

DFT Imagem Filtrada *Frequenc ButterWorth* em que filterOrder = 5

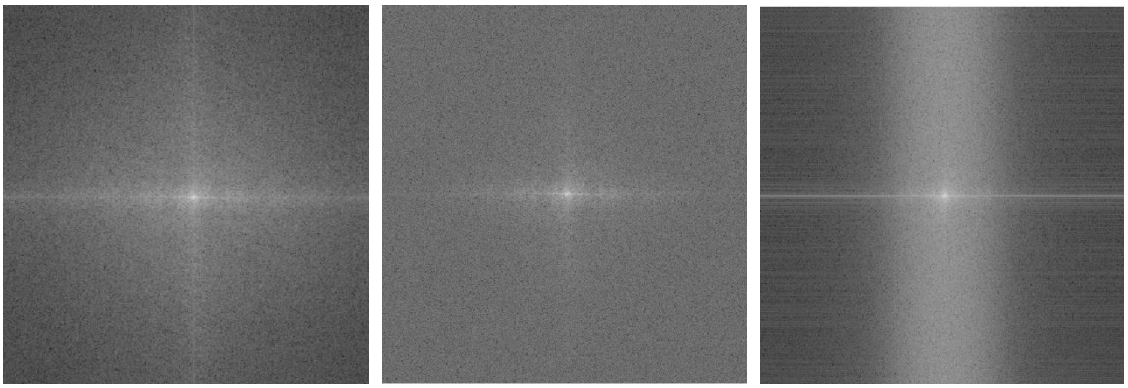



DFT Imagem Original

DFT Imagem Ruído *Salt & Pepper*

DFT Imagem Filtrada *Frequenc ButterWorth* em que filterOrder = 10



DFT Imagem Original

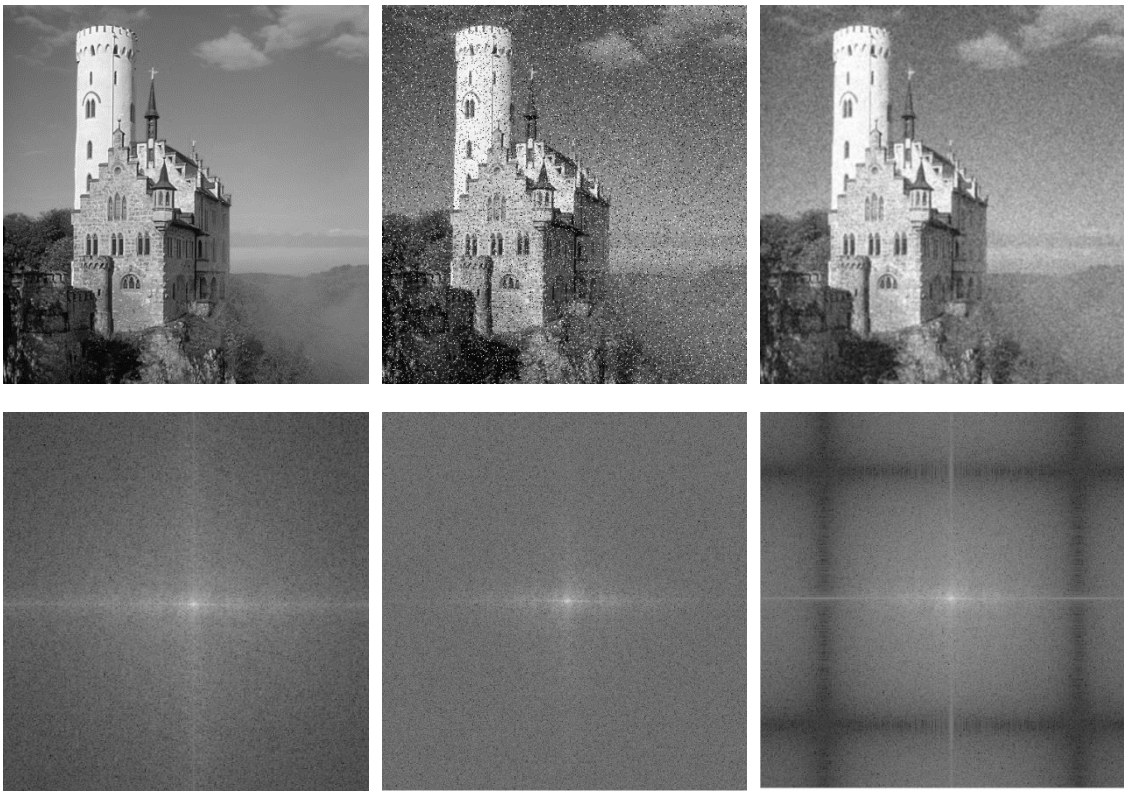
DFT Imagem Ruído *Salt & Pepper*

DFT Imagem Filtrada *Frequenc ButterWorth* em que filterOrder = 20

- Para o Filtro *Gaussian*, observa-se que à medida que o Valor do *Filter Size* aumenta, a Imagem acaba por se tornar mais desfocada;
- Existiu a opção de apenas colocar o butterType no modo *Low*, dado que seria um meio termo entre uma imagem nítida e com um contraste mais ideal;
- Em termos de código foram também geradas as Imagens relativas à Variação do Valor de *Cutoff*;
- Pelo que se analisa do *Filter Order*, o ideal é manter um valor baixo, dado que a Imagem se torna cada vez mais clara à medida que esse mesmo valor aumenta;
- Apesar disto já acontecer no Domínio de Filtragem *Spatial*, os resultados obtidos neste novo Domínio de Filtragem são claramente superiores.

2.1.2. *Gaussian Noise*

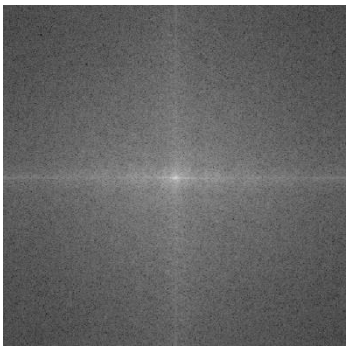
2.1.2.1. Filtro *Gaussian* (Variação Valor *Filter Size*)



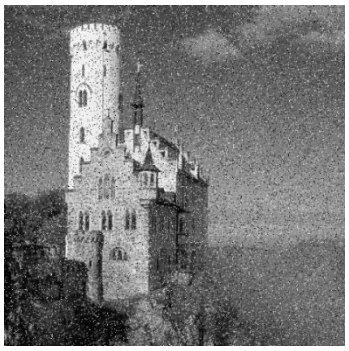
DFT Imagem Original

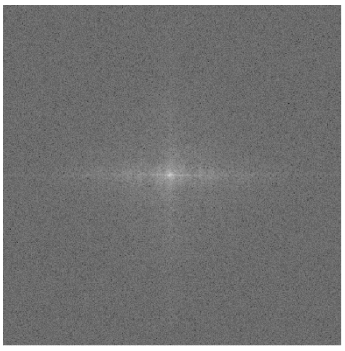
DFT Imagem Ruído *Gaussian*

DFT Imagem Filtrada *Frequency Gaussian* em que filterSize = 3

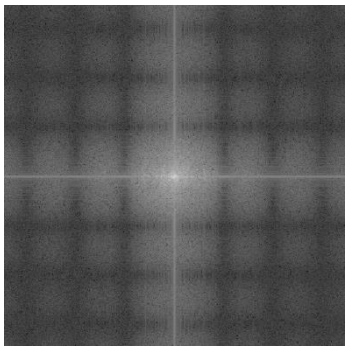


DFT Imagem Original

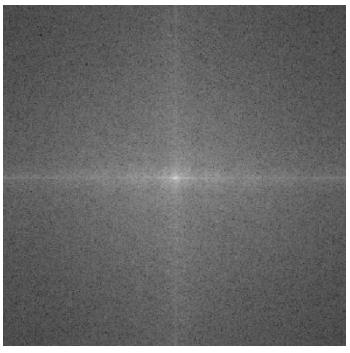




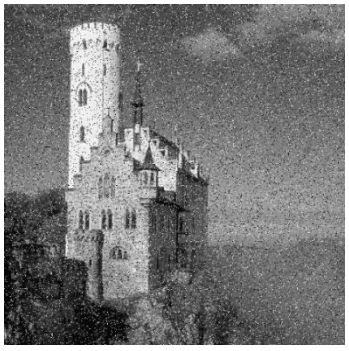
DFT Imagem Ruído *Gaussian*

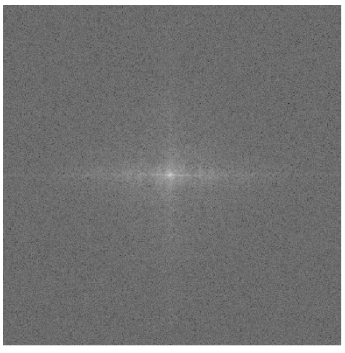


DFT Imagem Filtrada *Frequency Gaussian* em que **filterSize** = 7

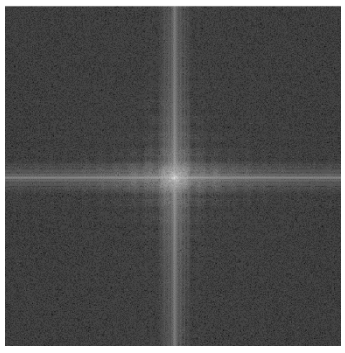


DFT Imagem Original



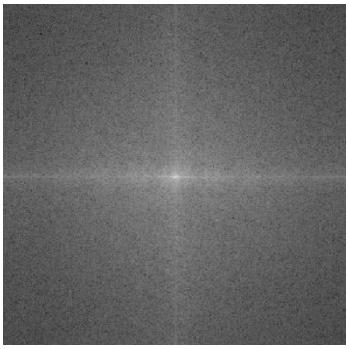


DFT Imagem Ruído *Gaussian*

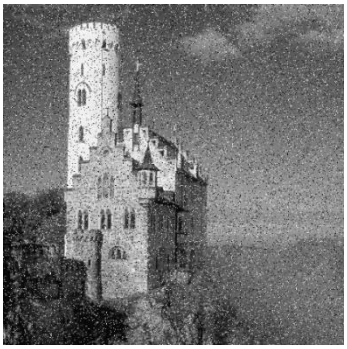


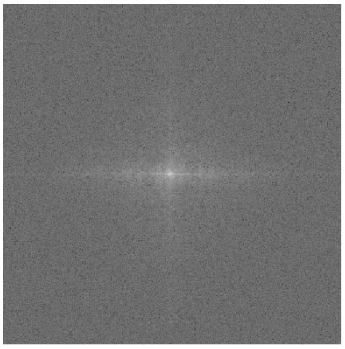
DFT Imagem Filtrada *Frequency Gaussian* em que **filterSize** = 21

2.1.2.2. Filtro *ButterWorth* (Variação *Filter Order*)

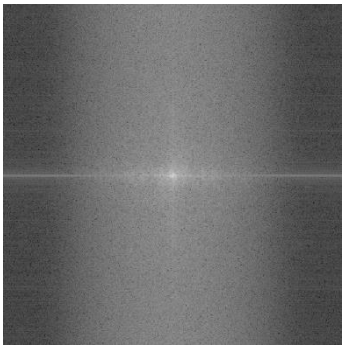


DFT Imagem Original

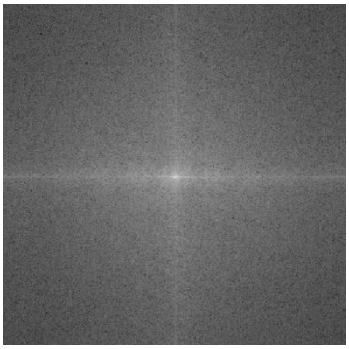




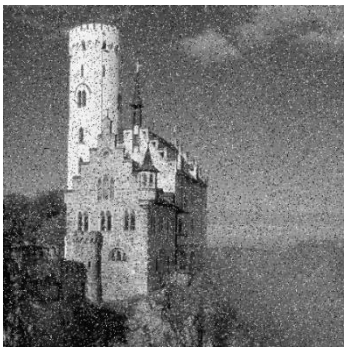
DFT Imagem Ruído *Gaussian*

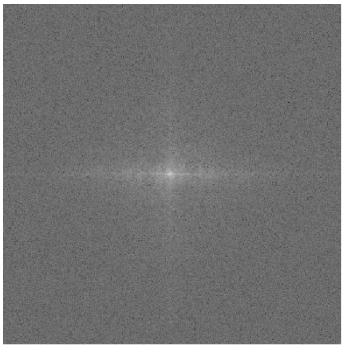


DFT Imagem Filtrada *Frequenc ButterWorth* em que **filterOrder** = 5

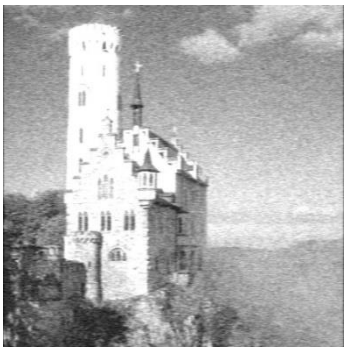


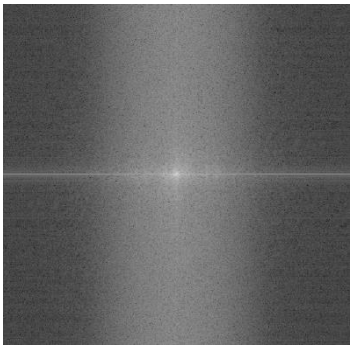
DFT Imagem Original



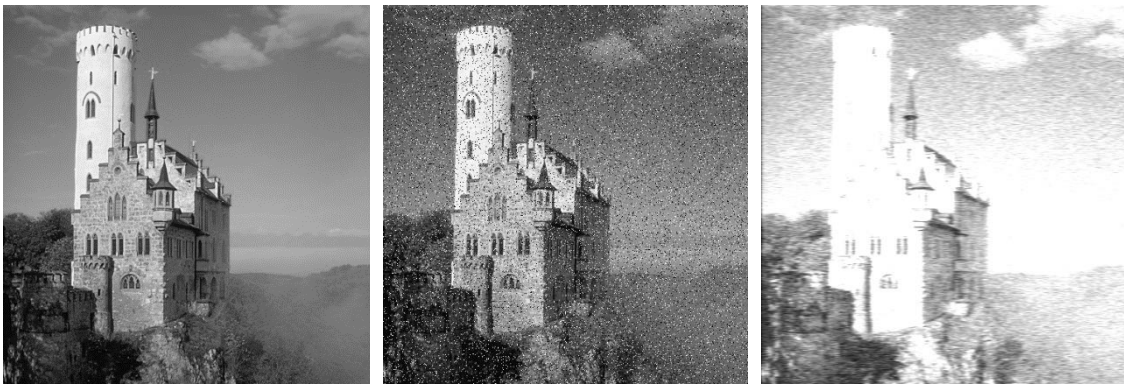


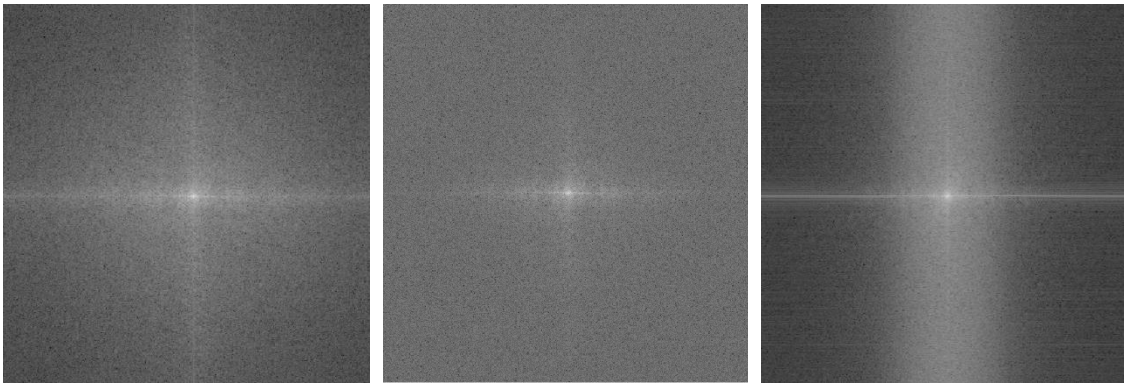
DFT Imagem Ruído *Gaussian*





DFT Imagem Filtrada *Frequenc ButterWorth* em que **filterOrder** = 10





DFT Imagem Original

DFT Imagem Ruído *Gaussian*

DFT Imagem Filtrada *Frequenc ButterWorth* em que filterOrder = 20

- Conclui-se que o Filtro *Gaussian* deste Domínio de Filtragem funciona bem tanto para a Redução do Ruído *Salt & Pepper*, como para o Ruído *Gaussian*.
- Em termos de Filtro *ButterWorth*, os resultados são similares ao que foi visto anteriormente.

Detect Edges with Canny Detector

Algoritmo

1. main _CrannyDetector

Esta função principal vai receber como parâmetros:

- O nome da imagem;
- A imagem com *Gaussian Noise* aplicado;
- Os *filterSize*;
- O sigma (*standardDeviation*).

A função começa por passar a imagem pelo processo de adição de ruído através do *Gaussian Noise* para posteriormente se aplicar a função *Gaussian_smoothing* que através do filtro *Gaussian* do domínio de filtragem *Spatial* faz o *Smoothing* da imagem com ruido. Essa imagem é guardada como a imagem antes do *Nonmax Supression.*

De seguida chama-se a função *gradient* que calcula a Magnitude e Direção do Gradiente a partir da imagem com ruido. Com os valores estabelecidos, é necessário normalizar as direções consoante o ângulo com a função *normalize_directions* para se poder calcular a *Nonmax Supression* e gerar a imagem que se vai guardar como a imagem após o *Nonmax Supression*.

Por fim é necessário gerar a imagem após *Hysteresis Thresholdind*, para isso é chamada a função *double_threshold* que recebe a imagem e devolve as arestas fortes e fracas da imagem. Com as arestas calculadas é possível obter o resultado do *Hysteresis Thresholding* com a função *hysteresis_thresholding* e assim gerar a imagem após este processo ser aplicado.

Interpretação e Análise dos Resultados

Para a Segunda Parte deste Tutorial pensou-se em algo mais simplificado, alterando-se apenas o valor do ***Filter Size***, observando-se assim a diferença que pode efetivamente fazer no *output* criado pelos vários Algoritmos.

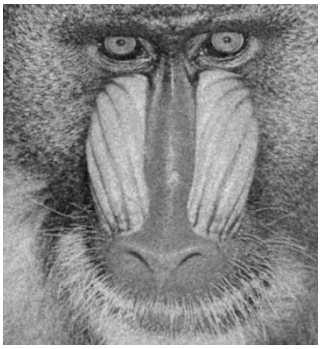
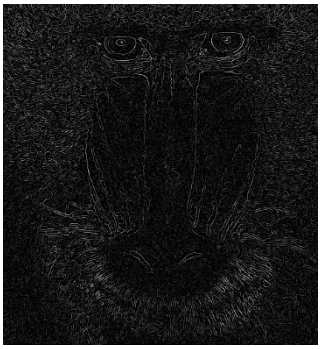
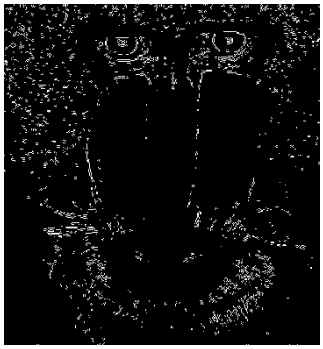
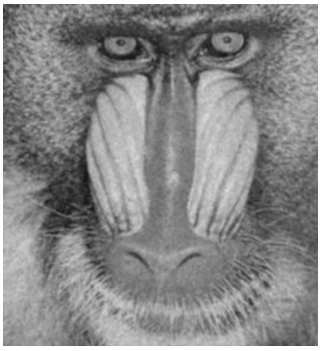
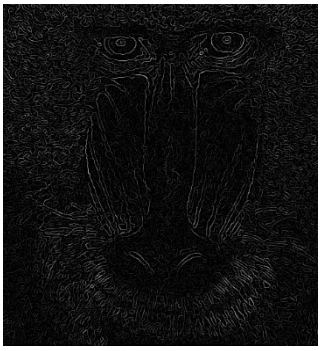
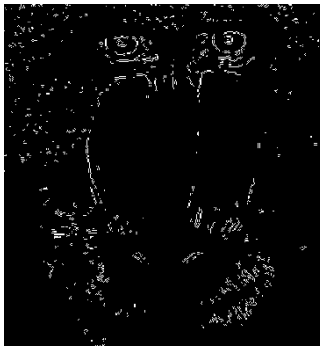
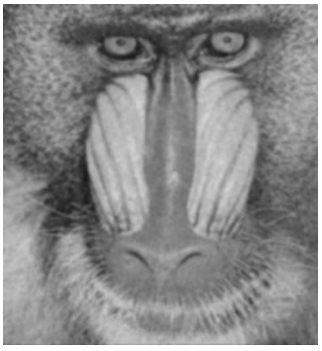
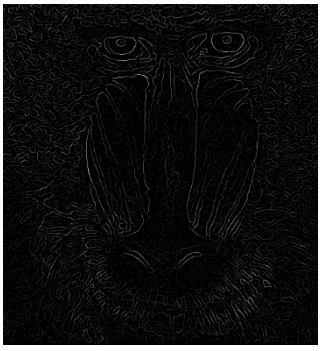
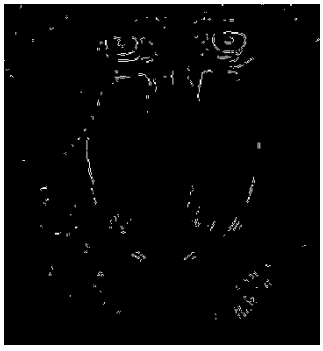
	Imagem com *Gaussian Smoothing*	Imagem *After Nonmax Suppression*	Imagem *After Hysteresis Thresholding*
Em que filterSize = 3			
Em que filterSize = 5			
Em que filterSize = 11			

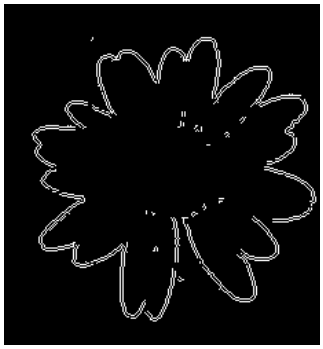
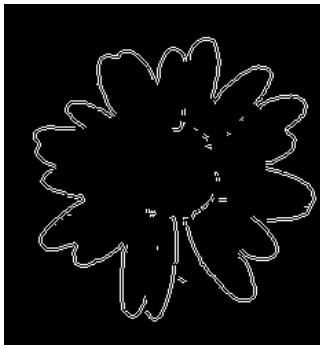
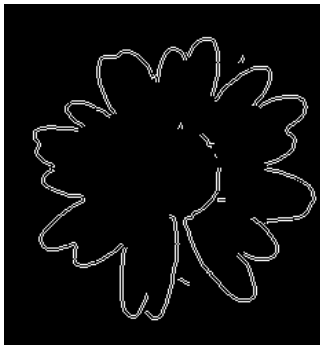
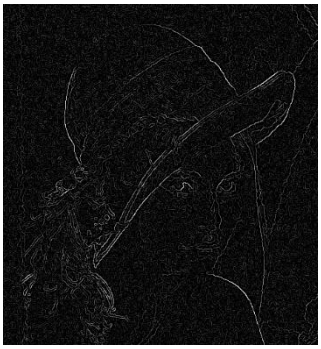
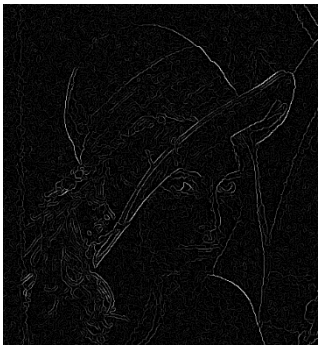
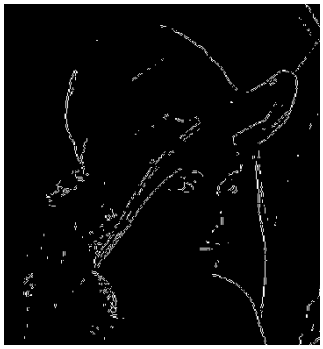
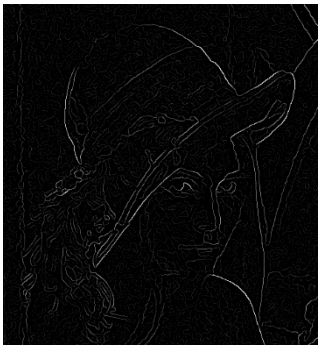
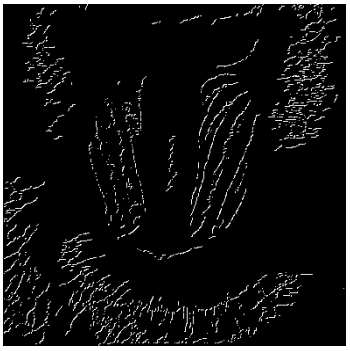
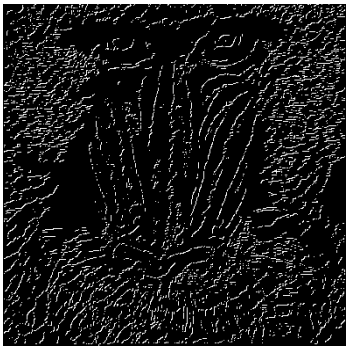
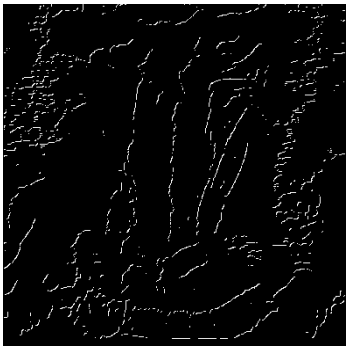
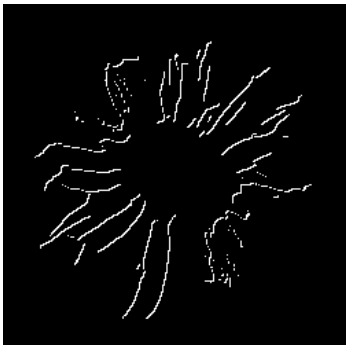
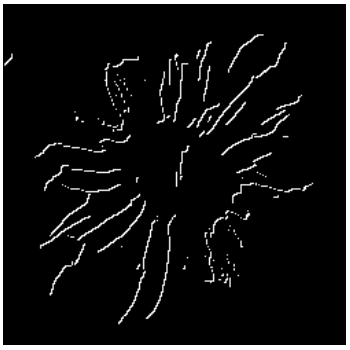
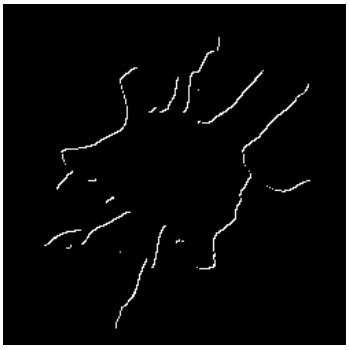
	Imagem com *Gaussian Smoothing*	Imagem *After Nonmax Suppression*	Imagem *After Hysteresis Thresholding*
Em que filterSize = 3			
Em que filterSize = 5			
Em que filterSize = 11			

	Imagem com *Gaussian Smoothing*	Imagem *After Nonmax Suppression*	Imagem *After Hysteresis Thresholding*
Em que filterSize = 3			
Em que filterSize = 5			
Em que filterSize = 11			

- O aumento do *Filter Size* não favorece o reconhecimento das *edges* da Imagem, de modo geral;
- Note-se que tanto na Imagem *After Nonmax Suppression* como *After Hyteresis Thresholding*, os resultados são visivelmente melhores para o *Filter Size* de menor valor;
- Algo que também não passa despercebido são os resultados mais satisfatórios na Imagem *Flower*, que por ter um contorno mais basilar, torna-se mais simples da parte do MATLAB fazer todo esse reconhecimento.

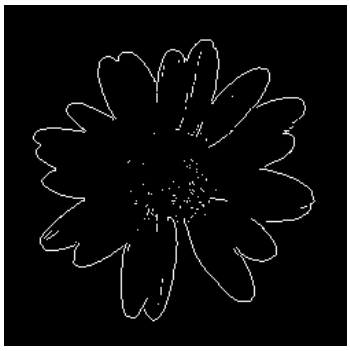
Comparação Resultados com *MATLAB's Edge Function*

Canny Edge Detection Method

Edges da Imagem em que filterSize = 0 e standardDeviation = 2	*Edges* da Imagem em que filterSize = 0.5 e standardDeviation = 2	*Edges* da Imagem em que filterSize = 0.5 e standardDeviation = 6
		
		
		

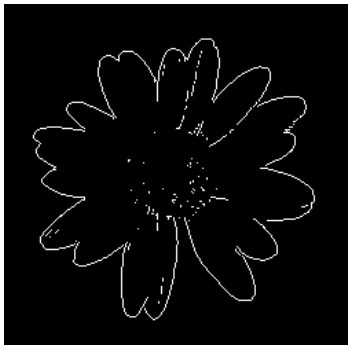
Sobel Edge Detection Method





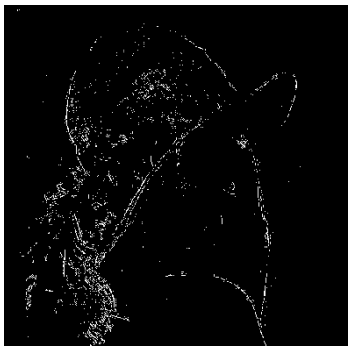
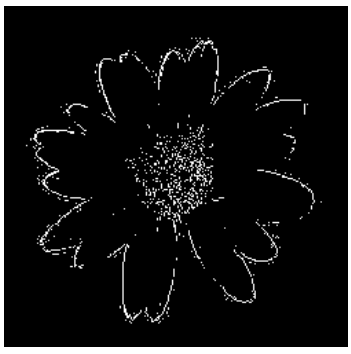
Prewitt Edge Detection Method





- Os Resultados são muito similares em ambos os Métodos;
- Para o Método *Canny*, para um valor intermédio do *Filter Size* e um valor mais baixo do *Standard Deviation*, os resultados são melhores;
- Isso valida a ideia anterior de que os resultados são melhores quando o valor do *Filter Size* é também ele menor.

Laplacian Edge Detection Method

Edges da Imagem em que filterSize = 0 e standardDeviation = 2	*Edges* da Imagem em que filterSize = 0.5 e standardDeviation = 0.5
	
	
	

- Para a Imagem do *Baboon* os valores ideais encontram-se na segunda coluna, mas para a Imagem da Lena e da *Flower*, os valores ideais encontram-se na primeira coluna, o que sugere quais poderiam ser os valores usados nos Algoritmos criados pelo grupo.