COPPE/UFRJ

REDUÇÃO DE RUÍDO EM IMAGENS

André Luiz Nunes Targino da Costa

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Eduardo Antônio Barros da Silva

Rio de Janeiro
Julho de 2009

REDUÇÃO DE RUÍDO EM IMAGENS

André Luiz Nunes Targino da Costa

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Aprovada por:

Prof. Eduardo Antônio Barros da Silva, Ph.D.

Prof. José Gabriel Rodríguez Carneiro Gomes, Ph.D.

Prof. Luiz Carlos Pacheco Rodrigues Velho, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
JULHO DE 2009

Targino da Costa, André Luiz Nunes

Redução de Ruído em Imagens/André Luiz Nunes Targino da Costa. – Rio de Janeiro: UFRJ/COPPE, 2009.

XXVIII, 351 p.: il.; 29, 7cm.

Orientador: Eduardo Antônio Barros da Silva

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2009.

Referências Bibliográficas: p. 84 – 93.

1. Denoising. 2. Morfologia. 3. Wavelets. 4. Estatística. I. da Silva, Eduardo Antônio Barros. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Dedico esta obra à memória de minha mãe, Madalena Nunes Targino da Costa, e de meu pai, Antonio Targino da Costa Filho, meu maior ídolo, que me ensinou tudo aquilo que eu precisava aprender sobre a vida. Ele era acima de tudo um ser humano íntegro e de caráter. Embora tenhamos tido pouco tempo juntos, foi o suficiente para se tornar eterno.

Agradecimentos

Agradeço a todos aqueles que contribuíram para que esta obra fosse concluída. Um agradecimento especial ao meu irmão, Luiz Eduardo, por me ajudar nos mais diversos momentos e demonstrar ser uma pessoa extremamente companheira e prestativa. Um outro agradecimento especial ao meu amigo Fred, por sempre estar disposto a me ouvir e me ajudar nos momentos mais difíceis.

Agradeço também ao meu amigo Tadeu, por se mostrar ser uma pessoa extremamente solícita, compreensiva e preocupada com o andamento desta dissertação; ao meu orientador Eduardo, por me dar uma orientação digna e esclarecer todos os pontos que até então não estavam consolidados em minha mente; e à equipe CoppeTeX por disponibilizar o *template* desta dissertação.

Para terminar, agradeço também aos meus demais amigos do LPS; aos meu amigos da empresa FIT; à Paty, por ter dado um colorido todo especial durante um bom tempo à minha vida; e ao demais amigos que me ajudaram, além, é claro, de um agradecimento todo especial aos meu pais, porque sem eles eu nem existiria!

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

REDUÇÃO DE RUÍDO EM IMAGENS

André Luiz Nunes Targino da Costa

Julho/2009

Orientador: Eduardo Antônio Barros da Silva

Programa: Engenharia Elétrica

Neste trabalho propomos a utilização de dois novos métodos de redução de ruído em imagens, um baseado em wavelets e outro baseado em morfologia matemática. Para tal, expusemos conceitos básicos sobre imagem e ruído, onde apresentamos modelos probabilísticos de alguns dos tipos mais comuns de ruído presentes em imagens, mencionando onde e como eles costumam ocorrer e exemplificando-os através de imagens ilustrativas. Após, implementamos alguns dos métodos baseados em estatística encontrados na literatura, focando principalmente na família dos filtros sigma, além de alguns métodos baseados em limiarização de coeficientes de transformadas wavelet. Por fim, testamos cada um dos métodos implementados e cada um dos métodos propostos em imagens corrompidas com ruído natural e em imagens corrompidas artificialmente com cada um dos tipos de ruído apresentados. Após uma extensa análise subjetiva e objetiva, em termos de PSNR, dos resultados, ficou evidente o potencial do método baseado em wavelets proposto e dos métodos baseados em morfologia matemática propostos, onde estes últimos não só se mostraram eficientes, como se puseram como as melhores opções dentre os métodos implementados no tocante aos aspectos analisados.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

IMAGE DENOISING

André Luiz Nunes Targino da Costa

July/2009

Advisor: Eduardo Antônio Barros da Silva

Department: Electrical Engineering

In this work, we propose the use of two new methods of image denoising, one based on wavelets and another based on mathematical morphology. For this purpose we have first outlined the basic concepts of image and noise, presenting some of the more common image noise models. We also have indicated where and how they might occur and have shown sample images illustrating the various degradations. Thereafter, we implemented some of the methods based on statistics found in literature, focusing on the family of sigma filters. Some methods based on wavelet shrinkage have also been implemented. Finally, we assessed the proposed methods using images corrupted by natural noise and using images artificially corrupted by each type of image noise models that we have presented. After extensive subjective and PSNR based objective analysis of the results, we conclude that the proposed wavelets based method and the proposed mathematical morphology based methods are promising. The latter were not only efficient, but they were also the best options among all implemented methods regarding the analyzed aspects.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O objetivo da presente dissertação é descrever tanto os métodos que fazem parte do estado da arte no tocante à *redução de ruído em imagens*, quanto os métodos que foram desenvolvidos pelo autor. A presente dissertação visa também gerar um documento com a maior riqueza de detalhes que se fizer possível para servir de referência a futuros pesquisadores da área. Para tal, o conteúdo desta obra será dividido em 6 capítulos e 4 apêndices. O presente capítulo descreverá de forma breve e resumida como se dará a divisão do restante do texto.

O capítulo 2, intitulado *Metodologia*, explicará a metodologia de pesquisa adotada na presente dissertação. Uma breve discussão sobre os procedimentos adotados será realizada. Além disso, será realizada uma revisão bibliográfica do assunto em questão. Por fim, será apresentada uma simples descrição do material utilizado durante a pesquisa, bem como algumas nuâncias relativas à implementação das técnicas abordadas que culminaram na criação de um software, denominado de *Image Denoiser*, para uma melhor apreciação deste trabalho.

O capítulo 3, intitulado *Métodos de Redução de Ruído Baseados em Estatística*, explorará em detalhes cada um dos métodos baseados em estatística implementados. Uma análise mais profunda será realizada em torno da família dos *filtros sigma*. Neste capítulo serão dadas ainda algumas definições relativas a imagem e a ruído. Serão apresentados os modelos probabilísticos de alguns dos tipos mais comuns de ruído presentes em imagens, sendo mencionado onde e como eles costumam ocorrer e sendo os mesmos exemplificados através de imagens ilustrativas.

O capítulo 4, intitulado *Métodos de Redução de Ruído Baseados em Wavelets*,

explorará em detalhes cada um dos métodos baseados em *wavelets* implementados. A análise será realizada em torno dos métodos baseados em limiarização de coeficientes de transformadas *wavelet*, começando por um método de funcionamento básico típico, seguido de um método com uma proposta de melhoria baseada em produtos multiescalas. Por fim, será apresentada a nossa proposta de um novo método de âmbito mais geral e ao mesmo tempo mais intuitivo do que os métodos nos quais ele foi baseado.

O capítulo 5, intitulado *Métodos de Redução de Ruído Baseados em Morfologia Matemática*, explorará em detalhes o método baseado em morfologia matemática proposto e cada uma de suas variações que serão aqui implementadas. Tais métodos serão concebidos com o intuito de se explorar as características espaciais de uma imagem baseado na forma com que nós seres humanos as exploramos.

O capítulo 6, intitulado *Conclusão*, agrupará todas as informações relativas às conclusões tiradas nos capítulos 3, 4 e 5 de forma clara, concisa e resumida, além, é claro, de apresentar conclusões mais amplas sobre a pesquisa como um todo.

O apêndice A, intitulado *Provas*, apresentará demonstrações matemáticas relativas a diversas equações utilizadas ao longo do texto, e que caso tivessem sido realizadas ao longo do mesmo, poderiam deixá-lo muito condensado e de difícil leitura.

O apêndice B, intitulado *Resultados*, exibirá resultados extensos de todos os métodos implementados com relação a cada uma das imagens utilizadas na presente obra, de forma a proporcionar uma maior compreensão sobre as diversas possibilidades de configuração de cada método.

O apêndice C, intitulado *Códigos*, mostrará os códigos utilizados no software *Image Denoiser* de todos os métodos implementados, de maneira que os mesmos possam ser reproduzidos com o mínimo de esforço possível.

Por fim, o apêndice D, intitulado *Imagens*, exibirá uma lista de todas as imagens que serão utilizadas nesta obra.

Capítulo 2

Metodologia

Neste capítulo explicaremos a metodologia de pesquisa adotada na presente dissertação. Uma breve discussão sobre os procedimentos adotados será realizada. Além disso, realizaremos uma revisão bibliográfica do assunto em questão. Por fim, apresentaremos uma simples descrição do material utilizado durante a pesquisa, bem como algumas nuâncias relativas à implementação das técnicas abordadas que culminaram na criação de um software, denominado de *Image Denoiser*, para uma melhor apreciação deste trabalho.

Neste ponto, para facilitar o entendimento deste capítulo, sumarizaremos a metodologia através dos seguintes itens (que seguem em ordem cronológica):

- Definição do escopo;
- Revisão da literatura;
- Nuâncias da pesquisa.

A seguir daremos então início à descrição de cada item acima mencionado.

2.1 Definição do escopo

Este trabalho foi concebido com o objetivo de realizar a redução de ruído em imagens. Para tal: realizaremos à redução de ruído em imagens corrompidas com ruído natural (real), ou seja, imagens contaminadas com ruído no processo de aquisição das mesmas, por imperfeições do hardware responsável para tal; e realizaremos também

à redução de ruído em imagens corrompidas com ruído artificial (simulado), que tenham sido contaminadas por meio de software após o processo de aquisição das mesmas.

2.2 Revisão da literatura

Uma vez definido o escopo, conforme dita a metodologia científica tradicional, foi realizada uma vasta revisão da literatura existente no que concerne a filtros de redução de ruído em imagens. Ao todo, 62 artigos [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62] foram lidos nessa primeira verificação [1]. Após uma breve análise dos mesmos, foi possível observar que muitos deles se valiam de técnicas um tanto recorrentes. Dessa forma, decidiu-se por agrupar os artigos de acordo com as ferramentas matemáticas que estes utilizavam majoritariamente. Dentre as classes de filtros encontradas, duas se destacaram bastante das outras com relação ao grau de incidência. São elas:

- Filtros Baseados em Estatística [3, 14, 36, 59, 60];
- Filtros Baseados em *Wavelets* [9, 10, 17, 19, 23, 24, 28, 33, 34].

Tenda em vista a reincidência de artigos referentes a essas duas classes de filtros, uma leitura mais aprofundada dos mesmos foi realizada [2]. Em decorrência desse fato, houve um aumento natural do conteúdo bibliográfico deste trabalho que passou a contar com mais 32 artigos [63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94], resultando em um total de 94 artigos consultados para a realização da presente obra.

Neste ponto do trabalho, houve uma redução natural da gama de artigos que seriam então analisados a fundo. Com relação aos *Filtros Baseados em Estatística*,

[1] O presente trabalho foi originado de um projeto sobre realce de imagens de ressonância magnética, por isso a revisão inicial da literatura é constituída basicamente de artigos relacionados ao tema de tal projeto.

[2] Na verdade houve também mais uma classe de filtros bastante recorrente, a dos filtros baseados em difusão anisotrópica [4,11,20,21,26,30,31,32]. Contudo, a qualidade dos resultados apresentada por tais artigos foi considerada insuficiente para justificar uma maior investida nos mesmos.

optou-se por dar uma maior ênfase aos artigos sobre a família de filtros denominada de *filtros sigma* [64, 66, 68, 69, 70, 71]. Já com relação aos *Filtros Baseados em Wavelets*, foi feita uma revisão mais abrangente da literatura do que a realizada inicialmente. Nessa nova revisão, foi possível observar o potencial de tais filtros conforme pode ser observado em [72, 73, 75, 81, 83, 85, 87, 88, 92]. Assim, decidiu-se por seguir então uma linha de pesquisa bastante recorrente dos métodos baseados em *wavelets*, a linha de *Wavelet Shrinkage*.

Uma vez que as linhas de pesquisa ficaram bem definidas, foi dado prosseguimento à implementação dos filtros de acordo com as condições descritas a seguir.

2.3 Nuâncias da pesquisa

Para a implementação das técnicas supracitadas, foi utilizado o software *MATLAB® & SIMULINK® Student Version* [3]. O ambiente de desenvolvimento e testes possuia a seguinte configuração:

- Disco rígido de 160GB SATA (5400rpm);
- Memória RAM de 2GB (1GB $\times$ 2) DDR2-533 SDRAM;
- Placa de vídeo Intel® Graphics Media Accelerator X3100 para portáteis;
- Processador Intel® Pentium® Dual-Core T2390 de 1,86GHz, 1MB L2 Cache;
- Windows Vista® Home Premium de 32 bits com Service Pack 1;
- Tela XBRITE™ de 15,4 polegadas TFT com resolução de 1280 $\times$ 800 pixels.

Devido à limitação da resolução da tela, associada à limitação do software e do sistema operacional utilizados, foram utilizadas imagens com resolução máxima de 1264 $\times$ 674 pontos. Essa limitação se fez necessária para que todas as imagens pudessem ser exibidas respeitando o critério de um ponto por pixel, não inserindo portanto distorções atreladas a escalamentos e/ou interpolações indevidas. Além disso, em razão do conteúdo exposto na literatura referenciada, neste trabalho foram utilizadas apenas imagens em tons de cinza, com 256 valores possíveis para cada

[3] A versão utilizada foi a 7.7.0.471 (R2008b) de 17 de setembro de 2008.

pixel. Ao todo, foram utilizadas 50 imagens (que podem ser observadas no apêndice D) para se testar os métodos implementados.

Definido o ambiente de trabalho e a matéria-prima a ser utilizada, foi dado prosseguimento à implementação das técnicas, verificação dos resultados e discussão dos mesmos. A partir da observação dos resultados, foi possível perceber algumas limitações nos métodos implementados. Em conseqüência disso, foi proposta uma melhoria em um dos métodos implementados, mais especificamente o método descrito em [34], e, além disso, foi proposto um novo método aparentemente inovador baseado em *Morfologia Matemática*.

Visando promover uma melhor continuidade do trabalho, foi desenvolvido um Software em MATLAB® com uma interface gráfica amigável, com intuito de facilitar a aplicação de filtros em imagens. Tal Software, denominado de *Image Denoiser*, foi desenvolvido de forma modular, permitindo assim a inclusão de outros filtros por parte do usuário, caso desejado. O código do Software, bem como os códigos dos filtros utilizados pelo mesmo encontram-se no apêndice C.

Nos capítulos a seguir iremos explorar em detalhes cada um dos métodos na ordem em que eles foram implementados. O capítulo 3 dissertará sobre os *Filtros Baseados em Estatística*, o capítulo 4 dissertará sobre os *Filtros Baseados em Wavelets* e, por fim, o capítulo 5 dissertará sobre os *Filtros Morfológicos*.

Capítulo 3

Métodos de Redução de Ruído Baseados em Estatística

Neste capítulo exploraremos em detalhes cada um dos métodos baseados em estatística implementados. Uma análise mais profunda será realizada em torno da família dos *filtros sigma*. Cabe salientar que durante muito tempo a classe dos *métodos estatísticos* foi a classe de métodos mais utilizada para redução de ruído em imagens. Contudo, devido a algumas limitações que serão expostas ao longo do capítulo, tal classe foi pouco a pouco cedendo lugar a novas classes de métodos, que passaram a levar em consideração outros aspectos da imagem além da estatística.

Para começarmos, a fim de tornar a leitura do capítulo mais clara, iremos realizar algumas definições a respeito da nomenclatura matemática que será utilizada, seguidas de algumas definições relativas a imagem e a ruído. Após, apresentaremos os modelos probabilísticos de alguns dos tipos mais comuns de ruído presentes em imagens, mencionando onde e como eles costumam ocorrer e exemplificando-os através de imagens ilustrativas. Por fim, discutiremos os pontos fortes e fracos de cada um dos métodos implementados com relação a cada tipo de ruído apresentado.

3.1 Definições

Devido aos diversos tipos de convenções e notações diferentes encontrados na literatura, faremos aqui algumas definições que serão utilizadas não só neste capítulo como em todo o resto do presente texto, como pode ser visto a seguir.

Função densidade de probabilidade

Considerando uma variável aleatória V, com possíveis resultados r, denotaremos a função densidade de probabilidade da mesma por $f_V(r)$. Da mesma forma, denotaremos a função densidade de probabilidade de uma seqüencia aleatória bidimensional S no ponto (d_1, d_2), com possíveis resultados r, por $f_S(r; (d_1, d_2))$. Ainda, denotaremos a função densidade de probabilidade conjunta entre duas seqüências aleatórias bidimensionais S_1 e S_2 nos pontos (d_1, d_2) e (d_3, d_4) respectivamente, com possíveis resultados r_1 e r_2 respectivamente, por $f_{S_1,S_2}(r_1, r_2; (d_1, d_2), (d_3, d_4))$.

Mediana de uma variável aleatória

Considerando uma variável aleatória V, com possíveis resultados r, denotaremos a mediana da mesma por:

$$\tilde{V} \triangleq r_m, \text{ tal que } \int_{-\infty}^{r_m} f_V(r)\,\mathrm{d}r = \int_{r_m}^{\infty} f_V(r)\,\mathrm{d}r \tag{3.1}$$

Média de uma variável aleatória

Considerando uma variável aleatória V, com possíveis resultados r, denotaremos a média da mesma por:

$$\overline{V} \triangleq \int_{-\infty}^{\infty} r f_V(r)\,\mathrm{d}r \tag{3.2}$$

Variância de uma variável aleatória

Considerando uma variável aleatória V, com possíveis resultados r, denotaremos a variância da mesma por:

$$\sigma_V^2 \triangleq \int_{-\infty}^{\infty} (r - \overline{V})^2 f_V(r)\,\mathrm{d}r \tag{3.3}$$

Desvio-padrão de uma variável aleatória

Considerando uma variável aleatória V, com possíveis resultados r, denotaremos o desvio-padrão da mesma por:

$$\sigma_V \triangleq \sqrt{\sigma_V^2} \tag{3.4}$$

3.1.1 Imagem

Nesta seção faremos defnições relativas à matéria-prima de nosso estudo: as imagens. Salientamos que todas as imagens que serão utilizadas neste trabalho são imagens digitalizadas (a lista completa das mesmas pode ser observada no apêndice D). Para melhor as distinguirmos nas diferentes situações em que serão apresentadas, de acordo com o contexto denotaremos uma imagem de 3 diferentes maneiras:

- uma imagem sem ruído será representada através da letra I;
- uma imagem com ruído será denotada pela letra X;
- uma imagem filtrada será identificada pela letra Y.

Com relação aos pixels das imagens, a localização dos mesmos será representada através do par ordenado (d_1, d_2), onde d_1 identificará a coluna e d_2 identificará a linha em que se encontra o pixel [1]. Assim, o valor do pixel de localização (d_1, d_2) de uma imagem sem ruído, por exemplo, será denotado por $I(d_1, d_2)$.

No que concerne à estatística, de acordo com [95], em representações estatísticas de imagens o valor de um pixel pode ser considerado como sendo um mapeamento realizado por uma *variável aleatória*, já que não sabemos *a priori* qual o valor do pixel. Dessa forma, podemos estender o raciocínio e dizer que uma imagem digitalizada é uma *tupla bidimensional* de dimensões $D_1 \times D_2$ cujos elementos são resultados de mapeamentos realizados por $D_1 \times D_2$ variáveis aleatórias. Ou seja, uma imagem digitalizada pode ser considerada como sendo uma *realização* de uma *seqüência aleatória discreta bidimensional*.

Neste ponto encontramos uma problema. Se quiséssemos realizar um cálculo probabilístico preciso de uma imagem ruidosa por exemplo, precisaríamos do conhecimento da função densidade de probabilidade, $f_X(r; (d_1, d_2))$, da seqüência aleatória discreta bidimensional em questão. Ou ainda, se quiséssemos realizar uma medida estatística precisa, necessitaríamos de uma quantidade infinita de realizações para se obter precisamente a função $f_X(r; (d_1, d_2))$. Contudo, como não temos o conhecimento de $f_X(r; (d_1, d_2))$ e nem temos uma quantidade infinita de realizações,

[1] Escolhemos essa ordenação com o intuito de nos adequarmos à convenção costumeiramente utilizada para se representar a resolução de uma imagem em colunas × linhas.

não podemos realizar nem um cálculo probabilístico preciso e nem uma medida estatística precisa [2]. Dessa forma, optamos por efetuar uma estimação da função $f_X(r;(d_1,d_2))$ para podermos prosseguir.

Com base nesse argumento, cabe deixar claro que todas as medidas aqui realizadas serão apenas estimativas. Em virtude dessa limitação, todas as definições a seguir serão denotadas com o uso do símbolo x, ao invés do símbolo X, para se caracterizar que as mesmas se tratam na verdade de estimativas amostrais. Assim, considerando $\mathbb{X}$ como sendo o multiconjunto [3] que contém todas as amostras que serão utilizadas para se realizar uma estimativa e N como sendo o número de elementos desse multiconjunto, definimos [4]:

$$\text{Mediana} : \tilde{x} \triangleq x\left[\frac{N+1}{2}\right] \in (\mathbb{X}, \leq) \tag{3.5}$$

$$\text{Média} : \overline{x} \triangleq \frac{1}{N}\sum_{n=1}^{N} x[n] \tag{3.6}$$

$$\text{Variância} : \sigma_x^2 \triangleq \frac{1}{N-1}\sum_{n=1}^{N}(x[n]-\overline{x})^2 \tag{3.7}$$

$$\text{Desvio padrão} : \sigma_x \triangleq \sqrt{\sigma_x^2} \tag{3.8}$$

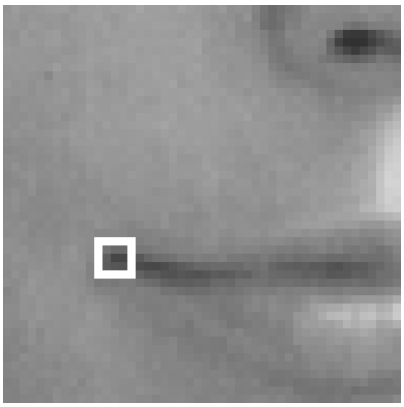
Uma observação pertinente é que como iremos realizar estimativas de medianas somente em casos em que haja um número ímpar de elementos, não é necessário definir a mesma para o caso em que haja um número par de elementos. Outra observação muito pertinente é que para todos os efeitos de estimação, os valores dos pixels de imagens (ou de blocos de imagens) poderão ser considerados como sendo elementos de um multiconjunto. Dessa forma, não cabe manter nenhuma espécie de ordenação entre eles já que em multiconjuntos, por definição, a ordem dos elementos não importa. Para ilustrar esse conceito, exemplos de estimativas para um bloco 3×3 da imagem *Lena* são exibidos na figura 3.1.

[2] Aqui nos referimos a probabilidade como algo de caráter teórico (abstrato) e a estatística como algo de caráter prático (concreto).

[3] Aqui consideramos um multiconjunto como sendo um conjunto que permita a repetição de seus elementos [96].

[4] Nesta dissertação, denotaremos o n-ésimo elemento do multiconjunto $\mathbb{X}$ totalmente ordenado pela relação de ordem $\leq$ por $x[n] \in (\mathbb{X}, \leq)$.

$\Downarrow$



$\Downarrow$

115	105	105
94	70	71
92	87	84

$\Downarrow$

115	105	105	94	70	71	92	87	84

Ou ainda:

70	115	84	105	94	92	105	87	71

$\Downarrow$

$$\tilde{x} = 92; \quad \overline{x} = 91.\overline{4}; \quad \sigma_x^2 \approx 235.2778; \quad \sigma_x \approx 15.3388$$

Figura 3.1: Exemplos de estimativas para um bloco 3×3 da imagem *Lena*.

Agora que terminamos as definições relativas a imagem, daremos então prosseguimento às definições relativas a ruído.

3.1.2 Ruído

Nesta seção faremos definições relativas à parte indesejada das imagens: o ruído. Consideraremos todo ruído como sendo um processo aleatório e denotaremos uma realização sua pela letra R. Com base nisso, podemos definir uma imagem ruídosa X como sendo uma função de uma imagem sem ruído I e de uma realização de ruído R da seguinte forma:

$$X = f(I, R) \tag{3.9}$$

Ruído aditivo

Felizmente, muitas vezes o ruído pode ser classificado como aditivo. Quando isso ocorrer, definiremos X como:

$$X = I + R \tag{3.10}$$

Ruído multiplicativo

Existem também vezes em que o ruído pode ser classificado como multiplicativo. Quando isso ocorrer, definiremos X como:

$$X = I \times R \tag{3.11}$$

Ruído estatísticamente independente da imagem

Diremos que o ruído é estatísticamente independente da imagem quando a função densidade de probabilidade conjunta de I e R for igual ao produto de suas respectivas funções densidade de probabilidade marginais, isto é, quando:

$$\begin{aligned} &f_{I,R}(r_I, r_R; (d_1, d_2), (d_1, d_2)) \\ &= f_I(r_I; (d_1, d_2)) \times f_R(r_R; (d_1, d_2)), \forall (d_1, d_2) \in D_1 \times D_2 \end{aligned} \tag{3.12}$$

Caso contrário diremos que o ruído é estatísticamente dependente da imagem.

Ruído composto de variáveis aleatórias independentes e identicamente distribuídas

Diremos que o ruído é composto de variáveis aleatórias independentes e identicamente distribuídas quando a função densidade de probabilidade conjunta entre dois pontos de R for igual ao produto das funções densidade de probabilidade marginais de tais pontos, e quando a função densidade de probabilidade de R for a mesma para todos os pontos de R, isto é, quando:

$$\begin{aligned} &f_{R,R}(r_1, r_2; (d_1, d_2), (d_3, d_4)) \\ &= f_R(r_1; (d_1, d_2)) \times f_R(r_2; (d_3, d_4)), \forall (d_1, d_2), (d_3, d_4) \in D_1 \times D_2 \end{aligned} \tag{3.13}$$

e

$$f_R(r; (d_1, d_2)) = f_R(r), \forall (d_1, d_2) \in D_1 \times D_2 \tag{3.14}$$

Respectivamente.

Tipos de ruído

De acordo com [97], existem basicamente 6 tipos de ruído que costumam afetar imagens [5]. São eles:

- Ruído térmico (Gaussiano);
- Ruído do tipo "sal e pimenta" (Impulsivo);
- Ruído de quantização (Uniforme);
- Ruído relativo à contagem de fótons (Poisson);
- Ruído relativo à granulação em fotografias (Binomial);
- Ruídos *Heavy-tailed*.

[5] Alguém pode argumentar que além dos 6 tipos de ruído apresentados também existe o ruído do tipo *speckle*. Contudo, de acordo com [97], *speckle* é um tipo de distorção em imagens criado por luz coerente ou por efeitos atmosféricos, não sendo tecnicamente considerado um tipo de ruído como os outros.

Uma forma de caracterização dos ruídos *Heavy-tailed* pode ser encontrada em [97]. De acordo com a mesma, dizer que um ruído tem "heavy tails" significa dizer que para valores grandes de r, $f_{HT}(r)$ se aproxima de 0 mais lentamente do que uma função densidade de probabilidade gaussiana se aproxima.

Com relação aos outros 5 tipos de ruído que foram listados, uma apresentação mais detalhada de suas características será realizada seguir. O mesmo porém não será feito com relação aos ruídos *Heavy-tailed* pois estes fogem ao escopo desta dissertação.

No mais, não podemos deixar de enfatizar que normalmente uma imagem acaba sendo corrompida por mais de um tipo de ruído. Um exemplo disso são as imagens adquiridas por meio de um CCD, que de acordo com [97] podem ter cada um de seus pixels definidos como:

$$X(d_1, d_2) = I(d_1, d_2) + R_F(\propto I(d_1, d_2)) + R_T + R_L, \tag{3.15}$$

onde: $R_F(\propto I(d_1, d_2))$ é um resultado de uma realização do ruído relativo à contagem de fótons (ver seção 3.1.6) com variância proporcional a $I(d_1, d_2)$; R_T é um resultado de uma realização do ruído térmico (ver seção 3.1.3) com variância proporcional à temperatura e ao tempo de exposição; e R_L é um resultado de uma realização de um ruído que é função do processo de leitura, independente da imagem e do tempo de exposição, e cuja função densidade de probabilidade é definida por uma distribuição de Poisson [98].

Por fim, antes de partirmos para as descrições mais detalhadas dos 5 primeiros tipos de ruído listados, cabe uma última e importante observação. Como sabemos, as imagens que serão aqui tratadas são digitalizadas. Contudo, as definições de ruído a seguir serão feitas através de funções densidade de probabilidades contínuas. Dessa forma, quando uma realização de tais ruídos corromper uma imagem, iremos implicitamente considerar que o resultado r de tal realização foi arredondado para o valor inteiro mais próximo dentro da faixa de 0 a 255, evitando assim qualquer confusão.

3.1.3 Ruído térmico (Gaussiano)

O ruído de térmico, T, é comumente modelado como sendo:

- Aditivo;
- Independente do sinal;
- Composto de variáveis aleatórias independentes e identicamente distribuídas.

Se nos basearmos em [98], com $\sigma_T^2 > 0$, podemos dizer que sua função densidade de probabilidade é gaussiana (ou normal) e pode ser definida como:

$$f_T(r) \triangleq \frac{1}{\sqrt{2\pi\sigma_T^2}} e^{-\frac{(r-\overline{T})^2}{2\sigma_T^2}} \tag{3.16}$$

Tal tipo de ruído é comumente encontrado em câmeras coloridas ou em quaisquer outros dispositivos onde as imagens sejam adquiridas por meio de um CCD.

Além disso, sob condições frequentemente razoáveis, a equação 3.16 é utilizada para modelar a função densidade de probabilidade de outros tipos de ruído, como por exemplo o ruído relativo à contagem de fótons e o ruído relativo à granulação de filmes fotográficos.

Tipicamente, uma imagem corrompida com este tipo de ruído terá um aspecto chuviscado como podemos observar na figura 3.2b.

(a) Imagem original

(b) Imagem corrompida

Figura 3.2: Imagem *Lena* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$).

3.1.4 Ruído do tipo "sal e pimenta" (Impulsivo)

O ruído do tipo "sal e pimenta", $S\&P$, é comumente modelado como sendo:

- Aditivo;
- Independente do sinal;
- Composto de variáveis aleatórias independentes e identicamente distribuídas.

Se nos basearmos em [97], com probabilidade de erro ϵ tal que $0 < \epsilon \leq 1$, podemos dizer que sua função densidade de probabilidade é do tipo impulsiva e pode ser dada por (prova no apêndice A.1):

$$f_{S\&P}(r) = (1-\epsilon)\delta(r) + \lim_{u\to\infty}\left[\frac{\epsilon}{2}\delta(r-u) + \frac{\epsilon}{2}\delta(r+u)\right] \tag{3.17}$$

Tal tipo de ruído é comumente encontrado em situações onde haja poeira dentro da câmera, elementos defeituosos no CCD, erros em conversores analógico-digitais ou erros de transmissão de imagens através de links digitais ruidosos.

O ruído do tipo "sal e pimenta" pode também se referir a uma grande variedade de processos que resultam no mesmo tipo básico de degradação em imagens: somente poucos pixels são ruidosos, porém bastante ruídosos.

Tipicamente, uma imagem corrompida com este tipo de ruído terá um aspecto salpicado, com pontos brancos (sal) e pontos pretos (pimenta), como podemos observar na figura 3.3b.

(a) Imagem original

(b) Imagem corrompida

Figura 3.3: Imagem *Lena* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$).

3.1.5 Ruído de quantização (Uniforme)

O ruído de quantização, Q, é comumente modelado como sendo:

- Aditivo;
- Independente do sinal;
- Composto de variáveis aleatórias independentes e identicamente distribuídas.

Se nos basearmos em [97], com passo Δ tal que $\Delta > 0$, podemos dizer que sua função densidade de probabilidade é do tipo uniforme e pode ser definida como:

$$f_Q(r) \triangleq \begin{cases} \frac{1}{\Delta} & \text{, se } -\frac{\Delta}{2} \leq r \leq \frac{\Delta}{2} \\ 0 & \text{, caso contrário} \end{cases} \tag{3.18}$$

Tal tipo de ruído é comumente encontrado em situações onde, como o próprio nome já diz, é realizada uma quantização do sinal original.

Uma observação importante é que quando o número de níveis de quantização é pequeno, o ruido de quantização se torna dependente do sinal, passa a não ser mais composto de variáveis aleatórias independentes e identicamente distribuídas e nem é mais uniformemente distribuido.

Tipicamente, uma imagem corrompida com este tipo de ruído terá um aspecto similar ao de uma rampa transformada numa escada. Pequenas graduações na intensidade são perdidas e passam a existir grandes regiões de intensidade constante separadas por contornos evidentes como podemos observar na figura 3.4b.

(a) Imagem original

(b) Imagem corrompida

Figura 3.4: Imagem *Lena* corrompida por ruído de quantização ($\Delta = 16$).

3.1.6 Ruído relativo à contagem de fótons (Poisson)

O ruído relativo à contagem de fótons, F, é comumente modelado como sendo:

- Aditivo;
- Dependente do sinal.

Se nos basearmos em [97], com $\sigma_F^2 > 0$, podemos dizer que sua função densidade de probabilidade é de Poisson e pode ser dada por (prova no apêndice A.2):

$$f_F(r) = e^{-\sigma_F^2} \sum_{k=-\sigma_F^2}^{\infty} \frac{\sigma_F^{2(k+\sigma_F^2)}}{(k+\sigma_F^2)!}\delta(r-k) \tag{3.19}$$

Tal tipo de ruído é comumente encontrado em dispositivos que se valem da contagem de fótons para se adquirir uma imagem. Fundamentalmente, quanto mais fótons contados em tal região, maior será o valor do pixel. Contudo, devido à variação no número de fótons detectados, o valor de um pixel é influenciado por um ruído cuja variância é proporcional à intensidade de luz captada por tal pixel.

Um observação interessante é que quando σ_F^2 possui um valor alto, o teorema do limite central pode ser evocado e a distribuição de poisson passa a ser bem aproximada pela gaussiana de média nula e de variância igual a σ_F^2.

Tipicamente, uma imagem corrompida com este tipo de ruído terá um aspecto chuviscado, sendo que numa região mais clara o chuviscado será mais intenso como podemos observar na figura 3.5b.

(a) Imagem original (b) Imagem corrompida

Figura 3.5: Imagem *Lena* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$).

3.1.7 Ruído relativo à granulação em fotografias (binomial)

O ruído relativo à granulação em fotografias, G, é comumente modelado como sendo:

- Aditivo;
- Dependente do sinal.

Se nos basearmos em [97], com probabilidade de transformação p tal que $0 < p < 1$ e L grãos por pixel com $L \in \mathbb{N}^*$, podemos dizer que sua função densidade de probabilidade é do tipo binomial e pode ser dada por (prova no apêndice A.3):

$$f_G(r) = \sum_{k=-\lfloor Lp \rfloor}^{L-\lfloor Lp \rfloor} \binom{L}{k + \lfloor Lp \rfloor} p^{k + \lfloor Lp \rfloor} (1-p)^{L - \lfloor Lp \rfloor - k} \delta(r-k) \tag{3.20}$$

Tal tipo de ruído é comumente encontrado em dispositivos que utilizam filme fotográfico para gravar uma imagem. Um filme fotográfico possui milhões de pequenos grãos. Quando a luz atinge o filme, alguns dos grãos se transformam e outros não. Se considerarmos todos os grãos idênticos e com mesmo p, o número de grãos indevidos que irão se transformar será aleatório seguindo uma distribuição binomial.

Contudo, existem casos em que uma distribuição binomial pode ser bem aproximada por uma distribuição de Poisson ou por uma distribuição gaussiana [97].

Tipicamente, uma imagem corrompida com este tipo de ruído terá um aspecto granulado como podemos observar na figura 3.6b.

(a) Imagem original (b) Imagem corrompida

Figura 3.6: Imagem *Lena* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$).

Por fim, na figura 3.7 a seguir podemos comparar as diferentes degradações causadas pelos 5 tipos de ruídos apresentados.

(a) Original

(b) Térmico

(c) Do tipo “sal e pimenta”

(d) De quantização

(e) Relativo à contagem de fótons

(f) Relativo à granulação em fotografias

Figura 3.7: Imagem *Lena* corrompida pelos 5 tipos de ruído apresentados.

Tendo terminado todas as definições realizadas neste capítulo, podemos agora explorar em detalhes cada um dos *métodos estatísticos* implementados. Para tal, nos valeremos de algumas informações que se aplicarão a todos os métodos que serão descritos nas próximas seções.

Como sabemos, pixels de localizações muito próximas tendem a possuir valores muito parecidos. Entretanto, quando uma imagem I é corrompida por uma realização de ruído R, tais pixels da imagem resultante X deixam de possuir valores tão parecidos. Para tentar amenizar essa distorção ocasionada pelo ruído, filtraremos a imagem X de modo a gerar uma imagem Y o mais parecida possível com a imagem I.

Para gerar a imagem Y, todos os métodos implementados neste capítulo compartilharão de um mesmo procedimento bastante conhecido. Eles utilizarão os pixels contidos em uma janela de observação de dimensões $D \times D$, centrada em um determinado pixel da imagem X, para poder calcular o valor do pixel de mesma localização da imagem Y. Após o término do cálculo, o processo será então repetido para todos os outros pixels da imagem X, gerando assim a imagem filtrada Y.

Dessa forma, para efeitos de cálculo do valor do pixel $Y(d_1, d_2)$, a menos que seja dito explicitamente outra coisa, consideraremos $N = D^2$ nas equações 3.5, 3.6 e 3.7, e definiremos $\mathbb{X}$ como sendo o multiconjunto cujos elementos $x[n]$ são os valores dos pixels contidos na janela de dimensões $D \times D$ centrada em $X(d_1, d_2)$. Essas considerações serão importantes para uma correta interpretação das equações que serão utilizadas por cada método.

Com relação à discussão dos resultados, utilizaremos imagens com ruído natural e artificial. Para análise de imagens com ruído natural, serão exibidos exemplos de filtragem da imagem *Moedas* (figura D.1) que possibilitem uma discussão subjetiva dos resultados relativos a cada um dos métodos. Com relação a imagens com ruído artificial, serão utilizadas as tabelas 3.1 e 3.2 para mostrar os melhores resultados, em termos de PSNR (do inglês *Peak Signal-to-Noise Ratio*), dos métodos estatísticos implementados para as imagens relativas à imagem *Lena* (figura D.3) e à imagem *Peppers* (figura D.4) respectivamente. Tais tabelas foram construídas levando-se em conta os resultados de todas as filtragens realizadas, resultados estes que podem ser vistos na sua totalidade no apêndice B.

3.2 Filtro mediana

O filtro mediana se baseia na idéia de que, como o pixel que estamos avaliando é o pixel central de uma janela, tal pixel provavelmente deveria possuir o valor central (mediana) da mesma janela. Esse raciocínio é válido se pensarmos que estamos numa região homogênea da imagem ou em qualquer região em que haja um gradiente de tons de cinza. Contudo, se pensarmos que estamos no ponto mais alto ou no ponto mais baixo de uma aresta, o raciocínio do filtro mediana acaba falhando, já que neste caso o valor do pixel provavelmente deveria ser o valor extremo de uma janela. Nestes casos, o filtro mediana tende a fazer com que o valor do pixel da imagem Y seja muito diferente do valor do respectivo pixel da imagem I. Contudo, nos outros casos ele tende a fazer com que os mesmos sejam muito parecidos.

A figura 3.8 a seguir mostra os resultados da filtragem da imagem *Moedas* pelo filtro mediana, com diferentes tamanhos de janela. Para a implementação do filtro foi utilizada a equação 3.5, tal que $Y(d_1, d_2) = \tilde{x}$. O código do referido filtro, utilizado no software *Image Denoiser*, pode ser visto no apêndice C.1.

Como se pode observar, o filtro mediana tende a embaçar a imagem. Isso pode ser explicado pelo fato de que os pontos mais altos ou mais baixos de uma aresta são substituídos pela mediana da janela o que faz com que as arestas sejam suavizadas. Além disso, percebe-se que quanto maior é a janela, mais embaçada fica a imagem, já que quanto maior é a janela, menos parecidos os pixels nela contidos tendem a ser, e consequentemente menos precisa tende a ser a estimativa do valor do pixel central com relação ao valor original pela mediana.

Com relação a imagens com ruído artificial, pela forma com que funciona o filtro mediana, é de se esperar que o mesmo seja muito bom em termos de PNSR para casos onde existam poucos pixels afetados por ruído (ainda que sejam muito afetados), mas seja apenas regular para casos onde existam muitos pixels afetados por ruído (ainda que sejam pouco afetados). As tabelas 3.1 e 3.2 confirmam esse raciocínio e mostram que o filtro mediana foi o que obteve o maior PSNR para imagens contaminadas por ruído do tipo "sal e pimenta", e em contrapartida foi o que obteve o segundo pior PSNR para imagens contaminadas por ruido de quantização. Com relação aos demais tipos de ruído o filtro mediana teve um desempenho regular no tocante ao PSNR.

(a) Imagem original

(b) Imagem filtrada (janela 3×3)

(c) Imagem filtrada (janela 5×5)

(d) Imagem filtrada (janela 7×7)

Figura 3.8: Resultados da filtragem da imagem *Moedas* pelo filtro mediana.

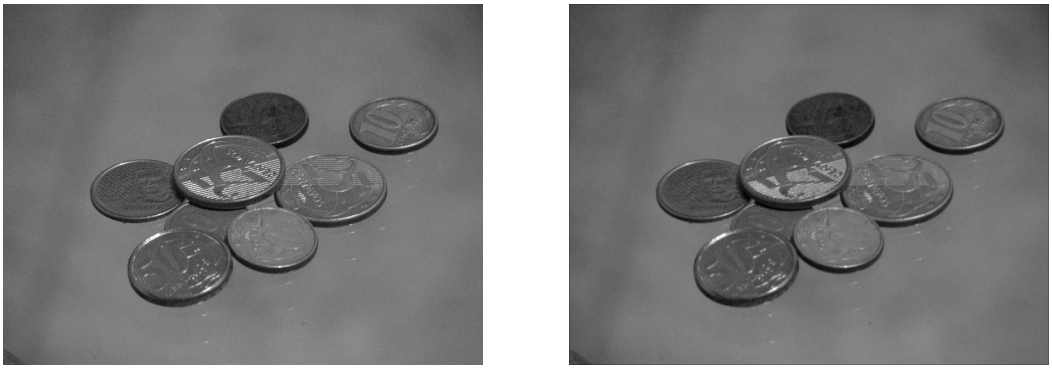
Para uma análise mais completa dos resultados do filtro mediana, recomenda-se a leitura do apêndice B.1.

3.3 Filtro média

O filtro média possui um raciocínio um tanto quanto parecido com o do filtro mediana. A diferença é que ao invés de substituir o pixel central de uma janela pelo valor central (mediana) da mesma, o filtro média opta por substituir o valor do pixel central de uma janela pela média aritmética dos valores dos pixels nela contidos.

A figura 3.9 a seguir mostra os resultados da filtragem da imagem *Moedas* pelo filtro média, com diferentes tamanhos de janela. Para a implementação do filtro foi utilizada a equação 3.6, tal que $Y(d_1, d_2) = \overline{x}$. O código do referido filtro, utilizado no software *Image Denoiser*, pode ser visto no apêndice C.2.

Como se pode observar, o filtro média tende a embaçar a imagem, mais até do que o filtro mediana. Esse embaçamento pode ser explicado pelo fato de que o filtro



(a) Imagem original

(b) Imagem filtrada (janela 3×3)

(c) Imagem filtrada (janela 5×5)

(d) Imagem filtrada (janela 7×7)

Figura 3.9: Resultados da filtragem da imagem *Moedas* pelo filtro média.

média é um filtro passa-baixas. Além disso, por ser um passa-baixas, quanto maior for a janela, mais embaçada a imagem ficará.

Com relação a imagens com ruído artificial, as tabelas 3.1 e 3.2 mostram que o filtro média é o pior filtro em termos de PSNR em 90% dos casos. Tal desempenho já era esperado tendo em vista que o filtro média filtra o conteúdo de altas freqüências de uma imagem. Além disso, a sua baixa eficiência com relação ao ruido do tipo "sal e pimenta" pode ser facilmente explicada, já que a média numa janela que contenha algum pixel severamente corrompido por ruído também será consideravelmente corrompida, ainda que de forma menos drástica. Assim, se somente um dos pixels teve o seu valor alterado em r, a média terá o seu valor alterado em $\frac{r}{D^2}$.

Para uma análise mais completa dos resultados do filtro média, recomenda-se a leitura do apêndice B.2.

3.4 Filtro sigma aditivo

O filtro sigma aditivo apresentado em [99] foi desenvolvido com o objetivo de filtrar imagens que tenham sido corrompidas por ruídos aditivos. Felizmente, conforme visto anteriormente nas seções 3.1.3, 3.1.4, 3.1.5, 3.1.6 e 3.1.7, alguns dos tipos mais comuns de ruído que costumam afetar imagens podem ser modelados como aditivos. Isso faz com que o filtro em questão tenha, em teoria, bastante utilidade no processo de redução de ruído em imagens.

O filtro sigma aditivo pode ser considerado como uma evolução do filtro média. No presente filtro, existe um critério para selecionar quais pixels serão utilizados para se efetuar o cálculo da média. Apenas pixels com valores próximos o suficiente do pixel central de uma janela entram no cálculo da média. Para expressarmos isso matematicamente, podemos primeiro calcular os coeficientes $a[n]$ (que serão utilizados mais a frente) da seguinte maneira:

$$a[n] = \begin{cases} 1 & \text{, se } |x[n] - X(d_1, d_2)| \leq \Lambda \\ 0 & \text{, caso contrário,} \end{cases} \tag{3.21}$$

onde Λ é um limiar comumente definido em função da estatística da imagem [6].

Cada coeficiente $a[n]$ calculado pela equação anterior servirá então de peso para o respectivo $x[n]$ no cálculo da média ponderada m_x da seguinte maneira:

$$m_x = \frac{\sum_{k=1}^{D^2} a[k]x[k]}{\sum_{k=1}^{D^2} a[k]} \tag{3.22}$$

A figura 3.10 a seguir mostra os resultados da filtragem da imagem *Moedas* pelo filtro sigma aditivo, com diferentes tamanhos de janela e de Λ [7]. Para a implementação do filtro foram utilizadas as equações 3.21 e 3.22, tal que $Y(d_1, d_2) = m_x$. O código do referido filtro, utilizado no software *Image Denoiser*, pode ser visto no apêndice C.3.

[6] Nos exemplos a seguir, Λ assumirá sempre um valor proporcional à estimativa σ_x do desvio-padrão da imagem X, dada pela equação 3.8.

[7] Na verdade existe ainda um outro parâmetro K que será explicado mais a frente. Para evitar o seu efeito neste momento no filtro sigma aditivo, fizemos $K = 0$.

(a) Imagem original

(b) Imagem filtrada (janela 3×3, $\Lambda = \sigma_x$)

(c) Imagem filtrada (janela 3×3, $\Lambda = 2\sigma_x$)

(d) Imagem filtrada (janela 5×5, $\Lambda = \sigma_x$)

(e) Imagem filtrada (janela 5×5, $\Lambda = 2\sigma_x$)

Figura 3.10: Resultados da filtragem da imagem *Moedas* pelo filtro sigma aditivo.

Podemos perceber que o filtro sigma aditivo embaça menos a imagem do que os filtros média e mediana. Isso pode ser explicado pelo fato de que somente pixels com valores próximos do valor de pixel central são utilizados para se fazer a média. Além disso, podemos perceber também que quando aumentamos o valor de Λ a imagem também fica mais embaçada. Isso ocorre pois quanto maior o valor de Λ, mais parecido será o comportamento do atual filtro com o filtro média, que como sabemos tende a embaçar a imagem.

O filtro sigma aditivo explicado até aqui parece ser um tanto quanto eficiente na filtragem de imagens ruidosas. Contudo, o seu cálculo se baseia no valor do pixel central. Assim, caso o pixel central de uma janela tenha sido bastante corrompido por ruído, ele será bem diferente dos demais, e existirão muito poucos pixels contidos na janela que possuam valores próximos ao dele. Quando isso ocorrer, o cálculo de m_x será bastante impreciso já que o pixel central é um dos pixels que está atrapalhando bastante o cálculo da média, e, neste caso, o filtro sigma aditivo como exposto até aqui não será tão útil. Para evitar que isso ocorra, o filtro sigma aditivo tem mais um parâmetro. Caso o número de pixels semelhantes ao pixel central seja menor ou igual a K, então o pixel central será considerado como um pixel muito ruídoso e, quando isso ocorrer, o seu valor será substituído pela média dos valores dos 4 pixels que são vizinhos imediatos do mesmo, conforme mostra a equação a seguir:

$$\Sigma_x = \begin{cases} \frac{X(d_1,d_2-1)+X(d_1-1,d_2)+X(d_1+1,d_2)+X(d_1,d_2+1)}{4} & \text{, se } \sum_{k=1}^{D^2} a[k] \leq K \\ m_x & \text{, caso contrário} \end{cases} \tag{3.23}$$

A figura 3.11 a seguir mostra os resultados da filtragem da imagem *Moedas* pelo filtro sigma aditivo, com uma janela de dimensões 3×3, $\Lambda = \sigma_x$ e diferentes valores de K. Para a implementação do filtro foram utilizadas as equações 3.21, 3.22 e 3.23, tal que $Y(d_1, d_2) = \Sigma_x$. O código do referido filtro, utilizado no software *Image Denoiser*, pode ser visto no apêndice C.3.

Podemos perceber que com a inclusão de K o filtro sigma passa a embaçar mais a imagem do que anteriormente, quando consideramos grandes valores de K. Isso pode ser explicado pelo fato de que a média de seus adjacentes é muito próxima do filtro média, e embora tire ruídos impulsivos, tira-os embaçando a imagem principalmente em arestas.

Com relação a imagens com ruído artificial, as tabelas 3.1 e 3.2 mostram que o filtro sigma aditivo é o melhor filtro em termos de PSNR em 40% dos casos, o segundo melhor em 40% dos casos e o terceiro melhor em 20% dos casos. Esses percentuais são bastante expressivos se considerarmos por exemplo que o filtro sigma adaptativo, que será explicado a seguir, se trata teoricamente de uma evolução do filtro sigma aditivo.

(a) Imagem original

(b) Imagem filtrada ($K = 1$)

(c) Imagem filtrada ($K = 3$)

(d) Imagem filtrada ($K = 5$)

(e) Imagem filtrada ($K = 7$)

(f) Imagem filtrada ($K = 9$)

Figura 3.11: Resultados da filtragem da imagem *Moedas* pelo filtro sigma aditivo (janela 3×3, $\Lambda = \sigma_x$).

Para uma análise mais completa dos resultados do filtro sigma aditivo, recomenda-se a leitura do apêndice B.3.

3.5 Filtro sigma adaptativo

O filtro sigma adaptativo de [64] é uma evolução do filtro sigma aditivo. Como podemos observar na seção 3.4, o filtro sigma aditivo usa o mesmo Λ na equação 3.21 para regiões planas e arestas. Contudo, de acordo com [64], é desejável que o parâmetro Λ seja ajustado de acordo com as variâncias locais, isto é, tenha o seu valor diminuído em regiões homogêneas e tenha o seu valor aumentado em arestas. Dessa forma, Λ na equação 3.21 passa a ser substituído por $\alpha\sigma_x$, conforme mostra a equação a seguir:

$$a[n] = \begin{cases} 1 & \text{, se } |x[n] - X(d_1, d_2)| \leq \alpha\sigma_x \\ 0 & \text{, caso contrário,} \end{cases} \tag{3.24}$$

onde α é uma constante de proporcionalidade escolhida arbitrariamente e σ_x é a estimativa do desvio-padrão da janela centrada em $X(d_1, d_2)$ dada pela equação 3.8 [8].

Ainda, de acordo com [64], é necessário realizar uma filtragem condicional para se reduzir a distorção do sinal. Dessa forma, caso o pixel central seja muito parecido com a média dos valores dos pixels contidos na janela em questão, o filtro sigma adaptativo não será aplicado, conforme mostra a equação a seguir:

$$A_x = \begin{cases} \Sigma_x & \text{, se } |\overline{x} - X(d_1, d_2)| > \sigma_x \\ X(d_1, d_2) & \text{, caso contrário,} \end{cases} \tag{3.25}$$

onde $\overline{x}$ é a estimativa da média da janela centrada em $X(d_1, d_2)$ dada pela equação 3.6.

A figura 3.12 a seguir mostra os resultados da filtragem da imagem *Moedas* pelo filtro sigma adaptativo, com diferentes tamanhos de janela, de α e de K. Para a implementação do filtro foram utilizadas as equações 3.24, 3.22, 3.23 e 3.25, tal que $Y(d_1, d_2) = A_x$. O código do referido filtro, utilizado no software *Image Denoiser*, pode ser visto no apêndice C.4.

Podemos perceber que o grau de embaçamento diminuiu se comparado ao filtro sigma aditivo. Isso pode ser explicado pela inclusão da filtragem condicional, que

[8] Em [64] existem ainda 3 parâmetros denotados por w_1, w_2 e w_3, responsáveis pelo ajuste do parâmetro Λ de acordo com as variâncias locais. Aqui fizemos $w_1 = 1$, $w_2 = 0$ e $w_3 = 0$.

(a) Imagem original

(b) Imagem filtrada (Janela 3×3, $\alpha = 1$, $K = 3$)

(c) Imagem filtrada (Janela 3×3, $\alpha = 2$, $K = 3$)

(d) Imagem filtrada (Janela 5×5, $\alpha = 1$, $K = 5$)

(e) Imagem filtrada (Janela 5×5, $\alpha = 2$, $K = 5$)

Figura 3.12: Resultados da filtragem da imagem *Moedas* pelo filtro sigma adaptativo.

fez com que de fato houvesse uma redução da distorção do sinal. Graças a ela, um número menor de pixels tiveram os seus valores substituídos pela média ponderada dos valores dos pixels contidos em suas respectivas janelas. O ponto negativo dessa filtragem condicional é que a redução de ruído ocasionada pelo atual filtro se torna claramente menor do que a do filtro sigma aditivo. Isso leva a crer que, embora o

filtro sigma adaptativo seja teoricamente de uma evolução do filtro sigma aditivo, em se tratando de redução de ruído em imagens, o atual filtro é menos eficiente do que o seu predecessor.

Com relação a imagens com ruído artificial, as tabelas 3.1 e 3.2 mostram que o filtro sigma adaptativo é o melhor filtro em termos de PSNR em se tratando de ruído de quantização. Isso mostra que a sua adaptabilidade quanto aos diferentes tipos de regiões de uma imagem, isto é, regiões homogêneas e arestas, é realmente eficiente. Com relação ao ruído do tipo "sal e pimenta", o filtro sigma adaptativo foi um dos piores em termos de PSNR, já que os valores dos pixels corrompidos por ruído impulsivo alteram sensivelmente as estimativas das variâncias locais, o que faz com que a adaptabilidade do presente filtro acabe sendo extremamente prejudicada. Com relação aos demais tipos de ruído o filtro sigma adaptativo teve um desempenho regular no tocante ao PSNR.

Para uma análise mais completa dos resultados do filtro sigma adaptativo, recomenda-se a leitura do apêndice B.4.

3.6 Filtro sigma multiplicativo

O filtro sigma multiplicativo apresentado em [100] foi desenvolvido com o objetivo de filtrar imagens que tenham sido corrompidas por ruídos multiplicativos. Conforme visto anteriormente nas seções 3.1.3, 3.1.4, 3.1.5, 3.1.6 e 3.1.7, o fato de que alguns dos tipos mais comuns de ruído que costumam afetar imagens tenham sido modelados como aditivos, não impede o uso do filtro sigma multiplicativo. Isso porque poderíamos ao invés de tentar realizar a redução de ruído da imagem X, buscar realizar a redução de ruído da imagem e^X por exemplo. Caso fizéssemos isso, nós teríamos que $e^X = e^I \times e^R$ já que de acordo com a equação 3.10, $X = I + R$. Assim, se definíssemos novas imagens X' e I', e uma nova realização de ruído R' tais que $X' = e^X$, $I' = e^I$ e $R' = e^R$, teríamos $X' = I' \times R'$, que é exatamente a definição de uma imagem X' corrompida por ruído multiplicativo, de acordo com a equação 3.11.

A única modificação feita no filtro sigma multiplicativo com relação ao filtro

sigma aditivo, é que a equação 3.21 passa a ser substituída pela seguinte equação:

$$a[n] = \begin{cases} 1 & \text{, se } \dfrac{|x[n] - X(d_1, d_2)|}{X(d_1, d_2)} \leq \Lambda \\ 0 & \text{, caso contrário,} \end{cases} \tag{3.26}$$

onde Λ mais uma vez é um limiar comumente definido em função da estatística da imagem [9].

A figura 3.13 a seguir mostra os resultados da filtragem da imagem *Moedas* pelo filtro sigma multiplicativo, com diferentes tamanhos de janela, de α e de K. Para a implementação do filtro foram utilizadas as equações 3.26, 3.22 e 3.23, tal que $Y(d_1, d_2) = \Sigma_x$. O código do referido filtro, utilizado no software *Image Denoiser*, pode ser visto no apêndice C.5.

A diferença visual entre os filtros sigma aditivo e multiplicativo é muito pequena. Isso pode ser explicado porque de fato na média as equações 3.21 e 3.26 são equivalentes (prova no apêndice A.4) [10].

Com relação a imagens com ruído artificial, as tabelas 3.1 e 3.2 mostram que o filtro sigma multiplicativo é o melhor filtro em termos de PSNR em se tratando de ruído relativo à contagem de fótons, o que faz sentido, já que a variância de tal tipo de ruído num dado ponto é igual ao valor da imagem I no mesmo ponto. Com relação aos demais tipos de ruído o filtro sigma multiplicativo teve um desempenho regular no tocante ao PSNR.

Para uma análise mais completa dos resultados do filtro sigma multiplicativo, recomenda-se a leitura do apêndice B.5.

3.7 Filtro sigma híbrido

O filtro sigma híbrido de [66] é uma evolução do filtro sigma multiplicativo. Como podemos observar na seção 3.6, o filtro sigma multiplicativo usa o mesmo Λ na equação 3.26 para todos os pixels de uma imagem. Contudo, de acordo com [66], é desejável que o parâmetro Λ seja ajustado com o objetivo de aumentar a capacidade

[9] No exemplo a seguir, Λ assumirá um valor proporcional à razão entre a estimativa σ_x do desvio-padrão da imagem X, dada pela equação 3.8, e a estimativa $\overline{x}$ da média da imagem X, dada pela equação 3.6.

[10] Isso é claro se consideramos $\Lambda = \alpha\sigma_x$ na equação 3.21 e $\Lambda = \alpha\frac{\sigma_x}{\overline{x}}$ na equação 3.26

(a) Imagem original

(b) Imagem filtrada (Janela 3×3, $\Lambda = \frac{\sigma_x}{\overline{x}}$, $K = 3$)

(c) Imagem filtrada (Janela 3×3, $\Lambda = 2\frac{\sigma_x}{\overline{x}}$, $K = 3$)

(d) Imagem filtrada (Janela 5×5, $\Lambda = \frac{\sigma_x}{\overline{x}}$, $K = 5$)

(e) Imagem filtrada (Janela 5×5, $\Lambda = 2\frac{\sigma_x}{\overline{x}}$, $K = 5$)

Figura 3.13: Resultados da filtragem da imagem *Moedas* pelo filtro sigma multiplicativo.

tanto de suprimir pulsos ruidosos quanto preservar arestas, contornos e finas linhas, ajustando-se às estruturas locais básicas da imagem. Com base nisso, o atual filtro procura utilizar informação espacial em conjunto com a informação estatística da imagem para obter um valor de Λ mais adequado para cada região da imagem.

O procedimento adotado pelo filtro sigma híbrido para combinar informação espacial e estatística pode ser descrito da seguinte forma. Chamaremos de M_S a matriz formada pelos valores dos pixels contidos em uma sub-janela de dimensões $D_S \times D_S$ centrada em $X(d_1, d_2)$, tal que $D_S \leq D$. Dessa forma, a informação sobre a localização espacial dos pixels é preservada. Com o intuito de utilizar a informação espacial dos pixels, criaremos um multiconjunto $\mathbb{S}$ composto de 4 elementos: a média dos elementos da linha $\frac{D_s+1}{2}$; a média dos elementos da coluna $\frac{D_s+1}{2}$; a média dos elementos da diagonal principal; e a média dos elementos da diagonal secundária. Assim, de acordo com [66], uma boa estimativa $\widehat{\overline{x}}$ para a média da janela em questão pode ser dada pela mediana de um multiconjunto formado por 3 elementos: $s[1] \in (\mathbb{S}, \leq)$; $s[4] \in (\mathbb{S}, \leq)$; e $X(d_1, d_2)$. Dessa forma, visando uma maior robustez, $X(d_1, d_2)$ e Λ na equação 3.26 passam a ser substituídos por $\widehat{\overline{x}}$ e $\alpha\frac{\sigma_x}{\overline{x}}$, respectivamente, conforme mostra a equação a seguir:

$$a[n] = \begin{cases} 1 & \text{, se } \dfrac{|x[n] - \widehat{\overline{x}}|}{\widehat{\overline{x}}} \leq \alpha\dfrac{\sigma_x}{\overline{x}} \\ 0 & \text{, caso contrário,} \end{cases} \tag{3.27}$$

onde α é uma constante de proporcionalidade escolhida arbitrariamente, σ_x é a estimativa do desvio-padrão da janela centrada em $X(d_1, d_2)$ dada pela equação 3.8 e $\overline{x}$ é a estimativa da média da janela centrada em $X(d_1, d_2)$ dada pela equação 3.6.

A figura 3.14 a seguir mostra os resultados da filtragem da imagem *Moedas* pelo filtro sigma híbrido, com diferentes tamanhos de janela, de sub-janela, de α e de K. Para a implementação do filtro foram utilizadas as equações 3.27, 3.22 e 3.23, tal que $Y(d_1, d_2) = \Sigma_x$. O código do referido filtro, utilizado no software *Image Denoiser*, pode ser visto no apêndice C.6.

Podemos perceber que o filtro sigma híbrido embaça mais a imagem do que o filtro sigma adaptativo, contudo, com relação aos outros filtros até aqui implementados ele embaça menos. Isso é sinal de que embora o atual filtro utilize alguma informação espacial da imagem, o mesmo não consegue o fazer de forma satisfatória.

Com relação a imagens com ruído artificial, as tabelas 3.1 e 3.2 mostram que o filtro sigma multiplicativo não foi o melhor filtro em termos de PSNR em nenhum caso. Contudo, ele foi bastante regular em termos de PSNR em todos os casos. O ponto negativo é que o atual filtro foi pior do que o filtro sigma multiplicativo em

(a) Imagem original

(b) Imagem filtrada (Janela 3×3, subjanela 3×3, $\alpha = 1$, $K = 3$)

(c) Imagem filtrada (Janela 3×3, subjanela 3×3, $\alpha = 2$, $K = 3$)

(d) Imagem filtrada (Janela 5×5, subjanela 5×5, $\alpha = 1$, $K = 5$)

(e) Imagem filtrada (Janela 5×5, subjanela 5×5, $\alpha = 2$, $K = 5$)

Figura 3.14: Resultados da filtragem da imagem *Moedas* pelo filtro sigma híbrido.

60% dos casos e pior do que o filtro sigma aditivo em 80% dos casos, o que representa que a evolução proporcionada pelo filtro híbrido no tocante ao PSNR não foi nada satisfatória.

Para uma análise mais completa dos resultados do filtro híbrido, recomenda-se a leitura do apêndice B.6.

3.8 Conclusões

Ao longo do capítulo, pudemos observar que os métodos estatísticos em geral tendem a embaçar a imagem. Um fator que a princípio poderia mudar esse quadro seria o formato das janelas de observação. Contudo, ainda que fossem utilizadas janelas em formatos de diamante ou de disco por exemplo, os resultados provavelmente não seriam muito diferentes dos apresentados, já que o motivo principal do embaçamento das imagens está relacionado a outro ponto, que será discutido a seguir.

Os métodos estatísticos não levam em conta qualquer informação espacial sobre a imagem, salvo o filtro sigma híbrido que já começa a tentar usar de alguma forma a informação sobre a disposição espacial dos pixels. Essa limitação, de não se valer da informação espacial dos pixels, limita bastante os métodos estatísticos, já que para nós seres humanos interessa muito mais o conteúdo espacial de uma imagem do que o conteúdo estatístico da mesma [101].

Um exemplo claro da limitação dos métodos estatísticos pode ser percebido se utilizarmos como exemplo os blocos da figura 3.15. Podemos notar de antemão que o Bloco B é uma permutação dos elementos do Bloco A, e vice-versa.

50	50	100	150	150
50	50	100	150	150
50	50	100	150	150
50	50	100	150	150
50	50	100	150	150

(a) Bloco A

50	150	50	100	150
100	50	50	150	50
50	50	100	150	150
150	50	150	150	100
50	100	150	50	150

(b) Bloco B

Figura 3.15: Exemplos de 2 blocos hipotéticos diferentes.

Tais blocos hipotéticos distintos são filtrados exatamente do mesmo jeito por qualquer um dos métodos estatísticos aqui implementados. Enquanto pode-se dizer que o conteúdo do Bloco A é claramente desprovido de ruído, no bloco B o conteúdo do mesmo possui claramente pixels contaminados por ruído. Dessa forma, o ideal seria não filtrar o bloco A e filtrar o bloco B de alguma forma. Contudo, os métodos estatísticos aqui implementados não têm como dar tratamentos diferentes aos 2 blo-

cos, o que deixa claro a limitação dos métodos estatísticos com relação à informação espacial.

Esse pequeno exemplo leva a crer que para realizarmos uma filtragem mais eficiente de uma imagem, do ponto de vista perceptual, precisaríamos de métodos que levassem em consideração a informação espacial de uma imagem. Alguns desses métodos serão explorados em detalhes nos capítulos a seguir, onde, no capítulo 4 exploraremos a classe de métodos baseada em *wavelets* e no capítulo 5 iremos propor uma nova classe de métodos baseada em *morfologia matemática*.

	T	$S\&P$	Q	F	G
Mediana	32,4525	**32,8072**	28,9034	31,9231	33,8332
Média	31,1684	26,6758	27,9647	30,9393	31,6481
Sigma aditivo	**33,6237**	32,5366	29,5406	32,9737	**35,5917**
Sigma adaptativo	31,4326	29,7256	**29,5845**	30,4692	34,1125
Sigma multiplicativo	33,2355	31,3384	29,4301	**33,1126**	35,4824
Sigma híbrido	32,8070	32,0993	29,5641	32,0577	34,7661

Tabela 3.1: Melhores resultados de PSNR dos filtros estatísticos (imagem *Lena*).

	T	$S\&P$	Q	F	G
Mediana	32,7621	**33,1648**	29,1780	32,6508	34,4521
Média	31,1745	26,5962	28,1018	31,1763	31,7661
Sigma aditivo	**34,0606**	32,3282	29,6507	33,5984	**35,8916**
Sigma adaptativo	31,5821	29,8922	**29,7741**	31,2572	34,6830
Sigma multiplicativo	32,9704	29,2722	29,4649	**33,7965**	35,4274
Sigma híbrido	32,8316	31,7653	29,7271	32,6740	34,9422

Tabela 3.2: Melhores resultados de PSNR dos filtros estatísticos (imagem *Peppers*).

Capítulo 4

Métodos de Redução de Ruído Baseados em Wavelets

Neste capítulo exploraremos em detalhes cada um dos métodos baseados em *wavelets* implementados. Nossa análise será realizada em torno dos métodos baseados em limiarização de coeficientes de transformadas *wavelet*, começando por um método de funcionamento básico típico, seguido de um método com uma proposta de melhoria baseada em produtos multiescalas. Por fim, apresentaremos a nossa proposta de um novo método de âmbito mais geral e ao mesmo tempo mais intuitivo do que os métodos nos quais ele foi baseado. Cabe salientar ainda que a classe dos *métodos wavelets* talvez seja hoje a classe de métodos mais utilizada para redução de ruído em imagens.

Para começarmos, a fim de tornar a leitura do capítulo mais clara, iremos realizar algumas definições a respeito da nomenclatura matemática que será utilizada. Após, discutiremos os pontos fortes e fracos de cada um dos métodos implementados neste capítulo com relação a cada tipo de ruído apresentado no capítulo 3, e faremos ainda uma análise comparativa desses métodos com os implementados no capítulo 3.

4.1 Definições

Devido aos diversos tipos de convenções e notações diferentes encontrados na literatura, faremos aqui algumas definições que serão utilizadas neste capítulo, como pode ser visto a seguir.

Convolução de duas funções bidimensionais contínuas

Considerando duas funções bidimensionais contínuas f_{c_1} e f_{c_2}, denotaremos a convolução entre ambas no ponto (d_1, d_2) por:

$$f_{c_1}(d_1, d_2) ** f_{c_2}(d_1, d_2) \triangleq \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f_{c_1}(\tau_1, \tau_2) f_{c_2}(d_1 - \tau_1, d_2 - \tau_2) \, \mathrm{d}\tau_1 \, \mathrm{d}\tau_2 \tag{4.1}$$

Convolução de duas funções bidimensionais discretas

Considerando duas funções bidimensionais discretas f_{d_1} e f_{d_2}, denotaremos a convolução entre ambas no ponto (d_1, d_2) por:

$$f_{d_1}[d_1, d_2] ** f_{d_2}[d_1, d_2] \triangleq \lim_{l \to \infty} \sum_{n_1=-l}^{l} \sum_{n_2=-l}^{l} f_{d_1}[n_1, n_2] f_{d_2}[d_1 - n_1, d_2 - n_2] \tag{4.2}$$

Dilatação de uma função bidimensional

Considerando uma função bidimensional $f(d_1, d_2)$, denotaremos a dilatação da mesma por um fator de escala s por:

$$f_s(d_1, d_2) \triangleq \frac{1}{s^2} f\left(\frac{d_1}{s}, \frac{d_2}{s}\right) \tag{4.3}$$

Funções wavelet e de escalamento

Denotaremos por ψ^h a função *wavelet* na direção horizontal. Da mesma forma, denotaremos por ψ^v a função *wavelet* na direção vertical. Por fim, denotaremos por θ a função de escalamento.

Transformadas wavelet de uma função bidimensional

Considerando uma função bidimensional $f(d_1, d_2)$, denotaremos a transformada *wavelet* na direção horizontal da mesma no ponto (d_1, d_2) na escala s por:

$$W_s^h(d_1, d_2) \triangleq f(d_1, d_2) ** \psi_s^h(d_1, d_2) \tag{4.4}$$

Da mesma forma, denotaremos a transformada *wavelet* na direção vertical de tal função no ponto (d_1, d_2) na escala s por:

$$W_s^v(d_1, d_2) \triangleq f(d_1, d_2) * * \psi_s^v(d_1, d_2) \tag{4.5}$$

Por conseqüência, diremos também que W_s^h é a transformada *wavelet* na direção horizontal de f na escala s, assim como W^v é a transformada *wavelet* na direção vertical de f na escala s.

4.2 Filtro baseado em wavelets

De uma forma geral, os filtros baseados em *wavelets* encontrados na literatura se baseiam na idéia de que grande parte da informação que nos interessa em uma imagem está contida em suas arestas, o que, de acordo com [101], faz bastante sentido. Com base nisso, os filtros baseados em *wavelets* exploram a capacidade que as transformadas *wavelet* possuem de decompor imagens no domínio do espaço em diferentes escalas para poder detectar assim suas arestas.

Como sabemos, as arestas contidas numa imagem estão intrinsicamente relacionadas à informação de gradiente ao longo de tal imagem. Assim, caso obtivéssemos uma descrição de tal gradiente, estaríamos obtendo também a informação das localizações das arestas de uma imagem. Um método comumente utilizado para se obter informação do gradiente de uma imagem é o método baseado em transformadas *wavelet*.

Em tal método, as funções *wavelet* ψ^h e ψ^v são derivadas parciais da função de escalamento θ, conforme pode ser visto na equação a seguir:

$$\psi^h(d_1, d_2) \triangleq \frac{\partial \theta(d_1, d_2)}{\partial d_1} \tag{4.6}$$

$$\psi^v(d_1, d_2) \triangleq \frac{\partial \theta(d_1, d_2)}{\partial d_2} \tag{4.7}$$

Onde $\psi^h(d_1, d_2)$ é a função *wavelet* que denota a derivada horizontal parcial da função de escalamento bidimensional $\theta(d_1, d_2)$ e $\psi^v(d_1, d_2)$ é a função *wavelet* que denota a derivada vertical parcial de tal função de escalamento bidimensional.

Ao fazermos isso, de acordo com [84] poderemos dizer que as transformadas *wavelet* na escala s de uma imagem são proporcionais às convoluções entre a imagem e a função de escalamento de mesma escala, conforme mostra a equação a seguir:

$$W_s^h(d_1, d_2) = s\frac{\partial(f(d_1, d_2) * * \theta_s(d_1, d_2))}{\partial d_1} \tag{4.8}$$

$$W_s^v(d_1, d_2) = s\frac{\partial(f(d_1, d_2) * * \theta_s(d_1, d_2))}{\partial d_2}, \tag{4.9}$$

onde W_s^h é a transformada *wavelet* horizontal da imagem X e W_s^v na escala s é a transformada *wavelet* vertical da mesma imagem também na escala s.

Em particular, de acordo com [34], quando θ_s é uma função gaussiana bidimensional, detectar os extremos locais de W_s^h e W_s^v é equivalente ao detector de arestas de Canny [102]. Neste ponto surge uma importante observação. Como estamos trabalhando com imagens digitalizadas, existe a necessidade de se efetuar o cálculo de W_s^h e de W_s^v para casos discretos. Para atingir esse objetivo, podemos nos valer de um método introduzido por [84], que faz com que a função de escalamento θ no atual contexto possa ser aproximada para o caso discreto por:

$$\theta[d_1, d_2] = H[d_1]H[d_2], \tag{4.10}$$

onde:

$$H[d] = 0,125\delta[d+1] + 0,375\delta[d] + 0,375\delta[d-1] + 0,125\delta[d-2] \tag{4.11}$$

Ainda de acordo com [84], podemos também gerar uma aproximação para o caso discreto de ψ^h e de ψ^v conforme mostram as equações a seguir:

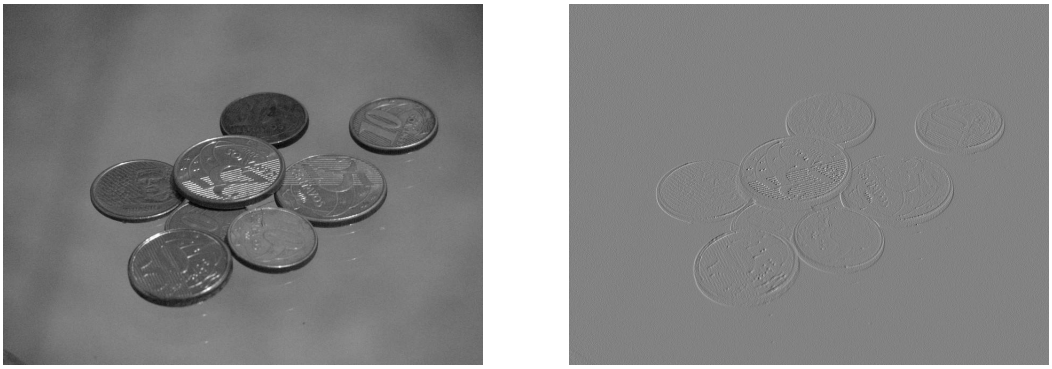
$$\psi^h[d_1, d_2] = G[d_1]\delta[d_2] \tag{4.12}$$

$$\psi^v[d_1, d_2] = \delta[d_1]G[d_2], \tag{4.13}$$

onde:

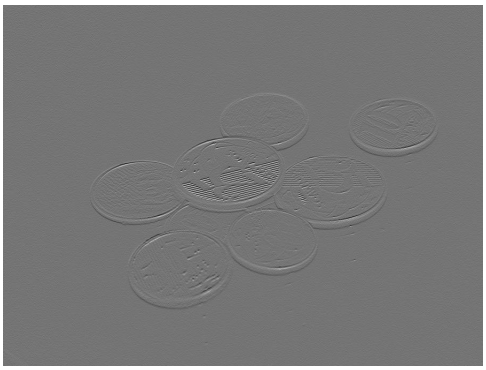
$$G[d] = -2\delta[d] + 2\delta[d-1] \tag{4.14}$$

A figura 4.1 a seguir mostra os resultados da decomposição da imagem *Moedas* em coeficientes de transformadas *wavelet* utilizando-se as equações 4.11, 4.10, 4.14, 4.12, 4.13, 4.4 e 4.5, tal que $f = X$ e $s = 1$.



(a) Imagem original

(b) W_1^h



(c) W_1^v

(d) $X * * \theta_1$

Figura 4.1: Decomposição da imagem *Moedas* em coeficientes de transformadas *wavelet*.

Podemos observar pela figura acima que as transformadas *wavelet* conseguiram alcançar o objetivo de localizar as arestas de uma imagem. Contudo, para termos uma informação mais robusta sobre as arestas, é necessário realizar a decomposição da imagem X também para outras escalas. Assim, novamente de acordo com [84], podemos aproximar aproximar a função de escalamento θ_s, com $s > 1$, no atual contexto para o caso discreto por:

$$\theta_s[d_1, d_2] = H_s[d_1]H_s[d_2], \tag{4.15}$$

onde:

$$H_s[d] = 0,125\delta[d+2^s+2^{s-1}]+0,375\delta[d+2^{s-1}]+0,375\delta[d-2^{s-1}]+0,125\delta[d-2^s+2^{s-1}] \tag{4.16}$$

Ainda de acordo com [84], podemos também gerar uma aproximação para o caso discreto de ψ_s^h e de ψ_s^v, com $s > 1$, conforme mostram as equações a seguir:

$$\psi_s^h[d_1, d_2] = \frac{1}{\lambda[s]} G_s[d_1]\delta[d_2] \tag{4.17}$$

$$\psi_s^v[d_1, d_2] = \frac{1}{\lambda[s]} \delta[d_1]G_s[d_2], \tag{4.18}$$

onde:

$$G_s[d] = -2\delta[d + 2^{s-1}] + 2\delta[d - 2^{s-1}] \tag{4.19}$$

e

$$\lambda[s] = 1,5\delta[s-1] + 1,12\delta[s-2] + 1,03\delta[s-3] + 1,01\delta[s-4] + \sum_{k=5}^{J} \delta[s-k], \tag{4.20}$$

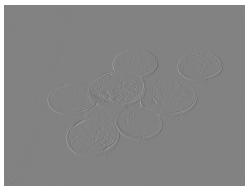
onde J é o número de escalas no qual será decomposta a imagem X.

A figura 4.2 a seguir mostra os resultados da decomposição da imagem *Moedas* em coeficientes de transformadas *wavelet* em várias escalas utilizando-se as equações 4.11, 4.10, 4.14, 4.12, 4.13, 4.4, 4.5, 4.16, 4.15, 4.20, 4.19, 4.17 e 4.18, tal que $f = X$.

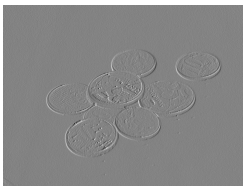
Agora que já conseguimos decompor a imagem X através de coeficientes de transformadas *wavelet* em várias escalas, podemos proceder ao processo de filtragem da mesma. Para tal, conforme mencionado no início do presente capítulo, os métodos de redução de ruído em imagens baseados em *wavelets*, se valem de um processo de limiarização de coeficientes de transformadas *wavelet*. Em tais métodos, um determinado coefiente de alguma transformada *wavelet* só terá seu valor conservado caso tal coeficiente seja maior ou igual a um determinado limiar. O presente filtro utiliza o *limiar universal* introduzido em [90] definido pela seguinte equação:

$$t_s^d = \sigma_s\sqrt{2\ln M}, \tag{4.21}$$

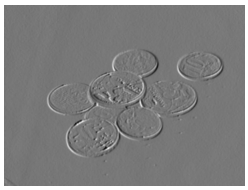
(a) Imagem original



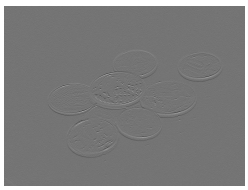
(b) W_1^h



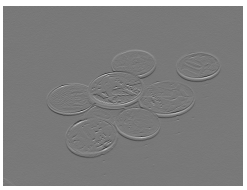
(c) W_2^h



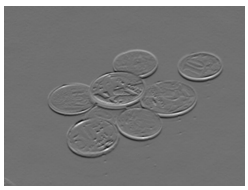
(d) W_3^h



(e) W_1^v



(f) W_2^v



(g) W_3^v

(h) $X * * \theta_1$

(i) $X * * \theta_2$

(j) $X * * \theta_3$

Figura 4.2: Decomposição da imagem *Moedas* em coeficientes de transformadas *wavelet* nas escalas 1, 2 e 3.

onde M é o número total de pixels de tal imagem e σ_s é uma estimativa do desvio-padrão do ruído encontrada através do *estimador MAV* (do inglês *Median Absolute Value*) definido em [34].

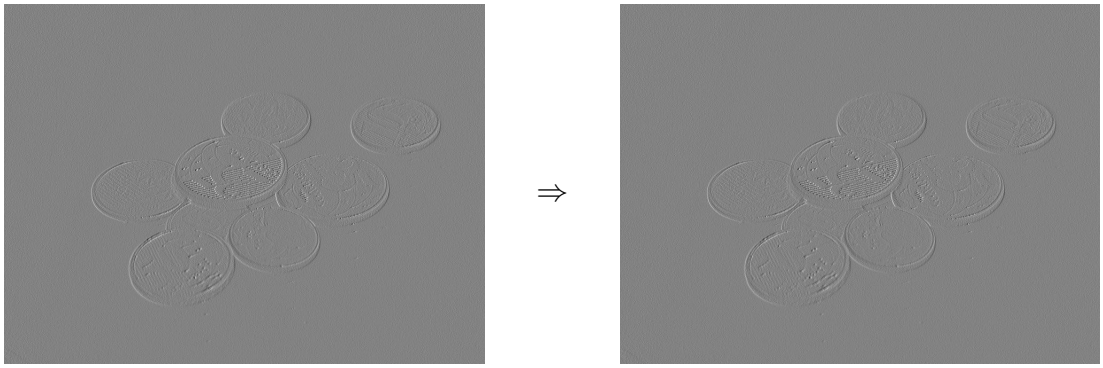
Podemos aplicar tal limiar para realizar a filtragem de cada coeficiente da transformada *wavelet* de direção d e escala s da imagem X, através da seguinte equação:

$$F_s^d(d_1, d_2) = \begin{cases} W_s^d(d_1, d_2) & \text{, se } W_s^d(d_1, d_2) > t_s^d \\ 0 & \text{, caso contrário,} \end{cases} \tag{4.22}$$

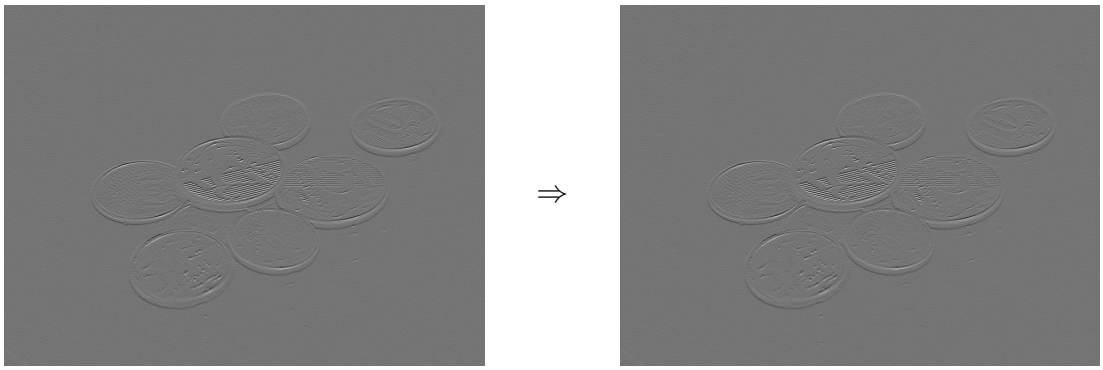
onde F_s^d é a transformada *wavelet* de direção d e escala s filtrada da imagem X.

A figura 4.3 a seguir mostra os resultados das filtragens das transformadas *wavelet* utilizando-se as equações 4.11, 4.10, 4.14, 4.12, 4.13, 4.4, 4.5, 4.16, 4.15, 4.20, 4.19,

4.17, 4.18, 4.21 e 4.22, tal que $f = X$ e $s = 1$.



(a) W_1^h (b) F_1^h

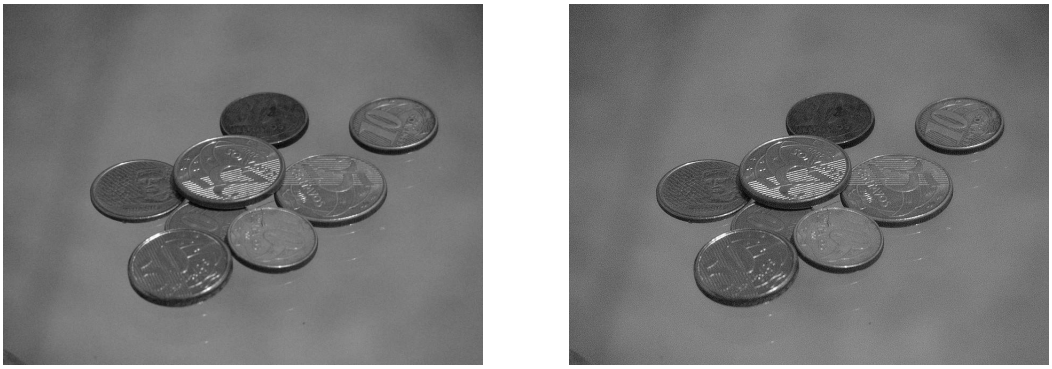


(c) W_1^v (d) F_1^v

Figura 4.3: Transformas *wavelet* da imagem *Moedas*, e suas respectivas versões filtradas.

Conforme podemos perceber, praticamente não há diferença visível entre as versões filtrada e não filtrada de cada transformada. A figura 4.4 a seguir mostra os resultados da filtragem da imagem *Moedas* pelo filtro baseado em *wavelets*. Para a implementação do filtro foram utilizadas as equações 4.11, 4.10, 4.14, 4.12, 4.13, 4.4, 4.5, 4.16, 4.15, 4.20, 4.19, 4.17, 4.18, 4.21 e 4.22, tal que $f = X$ e $J = 5$. O código do referido filtro, utilizado no software *Image Denoiser*, pode ser visto na seção C.7.

Como podemos observar, com base numa inspeção visual, de fato o filtro baseado em *wavelets* praticamente não realizou qualquer filtragem na imagem original. O que mostra que, apesar de termos conseguido encontrar um modo de encontrar arestas para realizar a redução de ruído em uma imagem, não conseguimos utilizá-lo de maneira eficiente. Ainda que o filtro baseado em *wavelets* não deixe as imagens embaçadas como os filtros estatísticos deixavam, o seu resultado pode ser conside-



(a) Imagem original (b) Imagem filtrada

Figura 4.4: Resultado da filtragem da imagem *Moedas* pelo filtro baseado em *wavelets*.

rado pior do que os destes, já que o principal objetivo, que é realizar a redução de ruído em imagens, não foi alcançado satisfatoriamente.

Com relação a imagens com ruído artificial, as tabelas 4.3 e 4.4 refletem tudo o que foi exposto anteriormente, onde o filtro baseado em *wavelets* é o pior em 90% dos casos. Contudo, é importante salientar que o simples fato de termos conseguido identificar a localização das arestas de uma imagem nos dá a indicação de que talvez, utilizando-se outros procedimentos, seja possível realizar uma redução de ruído de imagens de forma satisfatória.

Para uma análise mais completa dos resultados do filtro baseado em *wavelets*, recomenda-se a leitura da seção B.7.

4.3 Filtro baseado em wavelets modificado

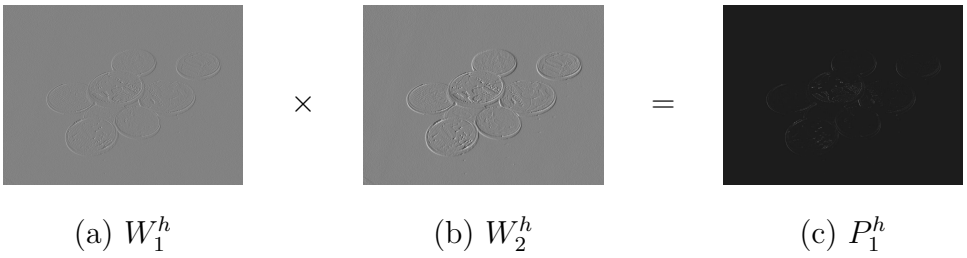
O filtro baseado em *wavelets* modificado de [35] é uma evolução do filtro baseado em *wavelets* apresentado na seção 4.2. O atual filtro se baseia no fato de que o sinal tende a estar muito correlacionado entre as escalas enquanto o ruído tende a estar muito descorrelacionado entre as escalas. Dessa forma, o presente filtro propõe um novo método para se realizar a redução de ruído em imagens, baseado no que chamamos de *produto multiescalas*.

Podemos definir tais produtos através das seguintes equações:

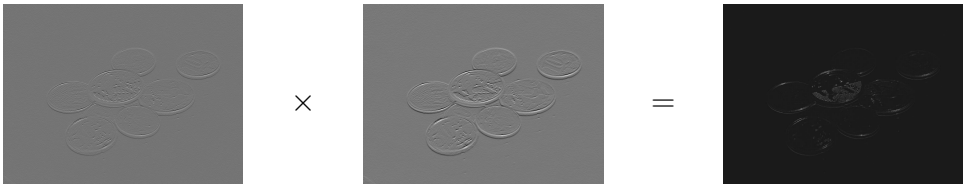
$$P_s^h(d_1, d_2) \triangleq W_s^h(d_1, d_2) W_{s+1}^h(d_1, d_2), \tag{4.23}$$

$$P_s^v(d_1, d_2) \triangleq W_s^v(d_1, d_2) W_{s+1}^v(d_1, d_2) \tag{4.24}$$

onde P_s^h é uma imagem que representada pelos produtos dos coeficientes de mesma localização das transformadas *wavelet* horizontais da imagem X nas escalas s e $s+1$, e P_s^v é uma imagem que representada pelos produtos dos coeficientes de mesma localização das transformadas *wavelet* verticais da imagem X nas escalas s e $s+1$. A figura 4.5 mostra um exemplo de tais produtos.



(a) W_1^h (b) W_2^h (c) P_1^h



(a) W_1^v (b) W_2^v (c) P_1^v

Figura 4.5: Calculo dos produtos dos coeficientes das transformadas *wavelet* nas escalas 1 e 2 da imagem *Moedas*.

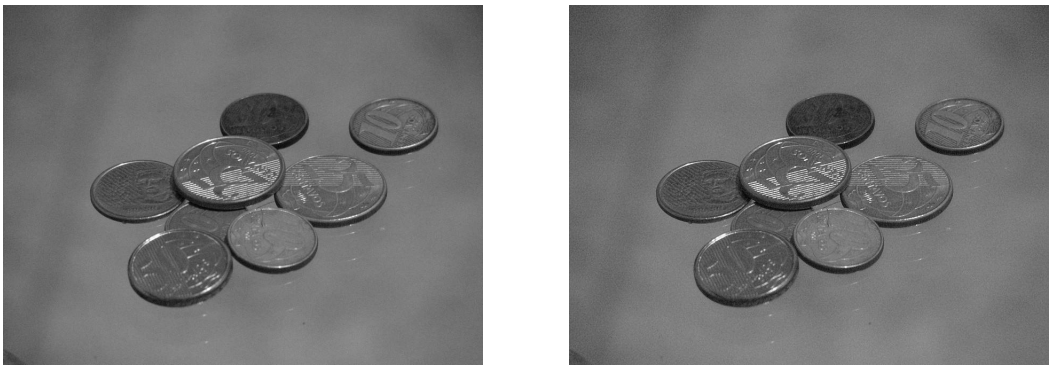
De posse de P_s^h e de P_s^v, o atual filtro realiza o processo de limiarização de coeficientes de transformadas *wavelet* de maneira diferente do filtro da seção anterior. Para tal, o presente filtro utiliza a informação dos produtos entre os coeficientes de escalas adjacentes, ao invés de utilizar os valores dos próprios coeficientes, conforme mostra a equação a seguir:

$$F_s^d(d_1, d_2) = \begin{cases} W_s^d(d_1, d_2) & \text{, se } P_s^d(d_1, d_2) > t_s^d \\ 0 & \text{, caso contrário,} \end{cases} \tag{4.25}$$

Além disso, o filtro baseado em *wavelets* modificado utiliza um novo limiar t_s^d, definido em [35] após um extensivo desenvolvimento matemático onde considera-se

que as imagens são corrompidas apenas por ruídos aditivos, com função densidade de probabilidade gaussiana e compostos de variáveis aleatórias independentes e identicamente distribuídas. Contudo, conforme vimos no capítulo 3, apenas 1 dos 5 tipos mais comuns de ruído que costumam afetar imagens podem ser modelados dessa maneira. Tal constatação deve provavelmente se refletir no resultado da filtragem das imagens.

A figura 4.6 a seguir mostra os resultados da filtragem da imagem *Moedas* pelo filtro baseado em *wavelets* modificado. Para a implementação do filtro foram utilizadas as equações 4.11, 4.10, 4.14, 4.12, 4.13, 4.4, 4.5, 4.16, 4.15, 4.20, 4.19, 4.17, 4.18, 4.23, 4.24 e 4.25, tal que $f = X$ e $J = 5$. O código do referido filtro, utilizado no software *Image Denoiser*, pode ser visto na seção C.7.



(a) Imagem original (b) Imagem filtrada

Figura 4.6: Resultado da filtragem da imagem *Moedas* pelo filtro baseado em *wavelets* modificado.

Como podemos observar, mais uma vez o resultado não foi satisfatório. Embora o resultado seja ligeiramente melhor do que o resultado de seu antecessor, o atual filtro praticamente filtrou muito pouco a imagem original. O que mostra que, apesar de termos conseguido encontrar um modo ainda mais robusto de encontrar arestas para realizar a redução de ruído em uma imagem, não conseguimos utilizá-lo de maneira eficiente. Novamente, ainda que o filtro baseado em *wavelets* modificado não deixe as imagens embaçadas como os filtros estatísticos deixavam, o seu resultado pode ser considerado pior do que os destes, já que o principal objetivo, que é realizar a redução de ruído em imagens, não foi alcançado satisfatoriamente.

Com relação a imagens com ruído artificial, as tabelas 4.1 e 4.2 mostram que

de fato o filtro baseado em *wavelets* modificado é melhor do que o seu antecessor. Nas mesmas, podemos observar que o atual filtro é melhor do que o outro em todos os casos. Em contrapartida, as tabelas 4.3 e 4.4 mostram que o filtro baseado em *wavelets* modificado não possui um bom resultado comparativo com os filtros estatísticos em termos de PSNR. Contudo, mais uma vez é importante salientar que o fato de termos conseguido identificar a localização das arestas de uma imagem de uma forma mais robusta, nos dá a indicação que talvez, utilizando-se outros procedimentos, seja possível realizar uma redução de ruído de imagens de forma satisfatória.

Para uma análise mais completa dos resultados do filtro baseado em *wavelets* modificado, recomenda-se a leitura da seção B.7.

4.4 Filtro baseado em wavelets proposto

O filtro baseado em *wavelets* proposto é um novo método proposto pelo autor que visa melhorar os resultados do filtro baseado em *wavelets* modificado. O presente método se difere do anterior apenas no aspecto relativo ao cálculo do limiar t_s^d. Para tal, o presente filtro utiliza apenas duas considerações de âmbito mais geral e ao mesmo tempo mais intuitivas do que o filtro da seção 4.3. São elas:

- Valores negativos de $P_s^d(d_1, d_2)$ significam que o pixel em questão foi drasticamente corrompido por ruído;
- O ruído que corrompe as imagens é aditivo, nada mais.

Conforme visto anteriormente nas seções 3.1.3, 3.1.4, 3.1.5, 3.1.6 e 3.1.7, alguns dos tipos mais comuns de ruído que costumam afetar imagens podem ser modelados como aditivos. Isso faz com que o filtro em questão tenha, em teoria, bastante utilidade no processo de redução de ruído em imagens. De posse desta informação e das considerações citadas anteriormente, se considerarmos $\mathbb{P}_s^d$ como sendo o multiconjunto que contém os valores de todos os coeficientes de P_s^d menores do que zero, diremos que a estimativa inicial do desvio-padrão σ_s^d do ruído presente em P_s^d é o desvio-padrão dos elementos de tal multiconjunto. Dessa forma, podemos definir o

limiar inicial do presente filtro através da seguinte equação:

$$t_s^d = N\sigma_s^d, \tag{4.26}$$

onde N é o número de desvios-padrões escolhido arbitrariamente que serão utilizados para se calcular o limiar t_s^d.

A partir daí, podemos calcular F_s^d de acordo com a equação 4.25. Após o término de tal cálculo, iremos por hipótese admitir que F_s^d é isento de ruído. Dessa forma, se fizermos $R_s^d = W_s^d - F_s^d$, estaremos encontrando em teoria apenas a parte ruídosa de W_s^d. Dessa forma, poderíamos estimar de forma mais precisa o valor de σ_s^d ao fazermos o mesmo valer o desvio-padrão dos valores dos coeficiente de R_s^d.

De posse do novo valor de σ_s^d, poderíamos calcular de novo o valor de t_s^d, agora porém de uma forma mais precisa. A partir daí, poderíamos repetir o processo descrito no parágrafo anterior até onde acharmos conveniente. Assim, quanto mais vezes executarmos tal processo, mais preciso será o nosso limiar t_s^d.

O presente filtro utiliza como critério de parada a diferença entre os valores dos limiares t_s^d, calculados de forma consecutiva. Assim, se tal diferença for menor do que 1, prosseguimos para a filtragem final da imagem.

A figura 4.7 a seguir mostra os resultados da filtragem da imagem *Moedas* pelo filtro baseado em *wavelets* proposto, com diversos valores de N. Para a implementação do filtro foram utilizadas as equações 4.11, 4.10, 4.14, 4.12, 4.13, 4.4, 4.5, 4.16, 4.15, 4.20, 4.19, 4.17, 4.18, 4.23, 4.24, 4.25 e 4.26, tal que $f = X$ e $J = 5$. O código do referido filtro, utilizado no software *Image Denoiser*, pode ser visto na seção C.7.

Como se pode observar, finalmente atingimos o resultado desejado de realizar a redução de ruído em imagens. Claramente o presente filtro realiza tal tarefa de forma mais satisfatória do que os outros filtros baseados em *wavelets* implementados. Isso se deve provavelmente ao fato de que o cálculo do limiar foi realizado de maneira mais precisa, já que basicamente o processo de filtragem do atual filtro é o mesmo que o de seu antecessor. Além disso, pode-se perceber que o filtro baseado em *wavelets* proposto consegue realizar a redução de ruído em imagens de forma mais clara do que os filtros estatísticos. No mais, podemos perceber que quanto maior o valor de N, com menos detalhes a imagem filtrada fica e mais artefatos são inseridos na

(a) Imagem original

(b) Imagem filtrada ($N = 0,2$)

(c) Imagem filtrada ($N = 0,4$)

(d) Imagem filtrada ($N = 0,6$)

(e) Imagem filtrada ($N = 0,8$)

(f) Imagem filtrada ($N = 1$)

Figura 4.7: Resultados da filtragem da imagem *Moedas* pelo filtro baseado em *wavelets* proposto.

mesma. Se observarmos os resultados relativos à redução de ruído de outras imagens no apêndice B.7, perceberemos que o filtro baseado em *wavelets* proposto acaba inserindo também uma quantidade de artefatos nas imagens filtradas proporcional à intensidade do ruído presente na imagem original.

Com relação a imagens com ruído artificial, as tabelas 4.1 e 4.2 mostram que de fato o filtro baseado em *wavelets* modificado é melhor do que os seus antecessores.

Nas mesmas, podemos observar que o atual filtro é melhor do que os outros em todos os casos. Em contrapartida, as tabelas 4.3 e 4.4 mostram que o filtro baseado em *wavelets* proposto não possui um bom resultado comparativo com os filtros estatísticos em termos de PSNR. Contudo, se avaliarmos a qualidade subjetiva das imagens correspondentes aos melhores resultados de tais filtros (ver apêndice B), veremos que o presente filtro possui uma qualidade subjetiva aparentemente melhor do que os filtros estatísticos. O que leva a crer que o PSNR não seja a medida ideal para se avaliar a qualidade de uma imagem, visto que o mesmo não leva em consideração qualquer informação espacial de uma imagem.

Para uma análise mais completa dos resultados do filtro baseado em *wavelets* proposto, recomenda-se a leitura da seção B.7.

4.5 Conclusões

Ao longo do capítulo, pudemos perceber que os filtros baseados em *wavelets* da literatura que foram implementados tendem a reduzir pouco o ruído nas imagens, mantendo porém uma qualidade subjetiva razoável das imagens. Já com relação ao filtro wavelet proposto, pudemos perceber que o mesmo consegue reduzir consideravelmente o ruído em imagens, eliminando porém uma quantidade considerável de detalhes e inserindo também uma quantidade de artefatos nas imagens filtradas proporcional à intensidade do ruído presente na imagem original.

Podemos concluir também que, como o processo de filtragem de todos os filtros apresentados neste capítulo são basicamente os mesmos, o fator que fez o filtro baseado em *wavelets* proposto apresentar resultados melhores que os seus antecessores foi a forma como se é calculado o limiar utilizado para filtrar os coeficientes de transformadas *wavelet*. Isso mostra que o método iterativo proposto, associado com o método de localização de arestas por produtos conseguem gerar de fato resultados um tanto quanto satisfatórios.

Mais especificamente, pudemos ainda perceber que quanto maior o valor de N, com menos detalhes a imagem filtrada fica e mais artefatos são inseridos na mesma.

No mais, pudemos ainda concluir que o PSNR não demonstrou ser a medida ideal para se avaliar a qualidade de uma imagem, já que o mesmo não leva em consideração

qualquer informação espacial de uma imagem. Isso indica que talvez fosse necessária a proposição de uma nova figura de mérito, que levasse em consideração a informação espacial de uma imagem, para ser utilizada como ferramenta de avaliação objetiva de qualidade de imagens.

	T	$S\&P$	Q	F	G
Wavelets	25,2096	15,1314	28,3566	23,8778	28,5721
Wavelets modificado	27,1641	16,2713	28,5916	26,0006	29,7947
Wavelets proposto	**29,6987**	**21,3223**	**28,7212**	**28,9960**	**31,2592**

Tabela 4.1: Melhores resultados de PSNR dos filtros baseados em *wavelets* (imagem *Lena*).

	T	$S\&P$	Q	F	G
Wavelets	24,6637	14,7654	27,1751	24,0776	27,5676
Wavelets modificado	26,2289	15,7419	27,2937	25,6172	28,3802
Wavelets proposto	**28,0465**	**20,0199**	**27,3696**	**27,6407**	**29,2491**

Tabela 4.2: Melhores resultados de PSNR dos filtros baseados em *wavelets* (imagem *Peppers*).

	T	$S\&P$	Q	F	G
Mediana	32,4525	**32,8072**	28,9034	31,9231	33,8332
Média	31,1684	26,6758	27,9647	30,9393	31,6481
Sigma aditivo	**33,6237**	32,5366	29,5406	32,9737	**35,5917**
Sigma adaptativo	31,4326	29,7256	**29,5845**	30,4692	34,1125
Sigma multiplicativo	33,2355	31,3384	29,4301	**33,1126**	35,4824
Sigma híbrido	32,8070	32,0993	29,5641	32,0577	34,7661
Wavelets	25,2096	15,1314	28,3566	23,8778	28,5721
Wavelets modificado	27,1641	16,2713	28,5916	26,0006	29,7947
Wavelets proposto	29,6987	21,3223	28,7212	28,9960	31,2592

Tabela 4.3: Comparativo entre os melhores resultados de PSNR dos filtros estatísticos e dos filtros baseados em *wavelets* (imagem *Lena*).

	T	$S\&P$	Q	F	G
Mediana	32,7621	**33,1648**	29,1780	32,6508	34,4521
Média	31,1745	26,5962	28,1018	31,1763	31,7661
Sigma aditivo	**34,0606**	32,3282	29,6507	33,5984	**35,8916**
Sigma adaptativo	31,5821	29,8922	**29,7741**	31,2572	34,6830
Sigma multiplicativo	32,9704	29,2722	29,4649	**33,7965**	35,4274
Sigma híbrido	32,8316	31,7653	29,7271	32,6740	34,9422
Wavelets	24,6637	14,7654	27,1751	24,0776	27,5676
Wavelets modificado	26,2289	15,7419	27,2937	25,6172	28,3802
Wavelets proposto	28,0465	20,0199	27,3696	27,6407	29,2491

Tabela 4.4: Comparativo entre os melhores resultados de PSNR dos filtros estatísticos e dos filtros baseados em *wavelets* (imagem *Peppers*).

Capítulo 5

Métodos de Redução de Ruído Baseados em Morfologia Matemática

Neste capítulo exploraremos em detalhes o método baseado em morfologia matemática proposto e cada uma de suas variações aqui implementadas. Tais métodos foram concebidos com o intuito de se explorar as características espaciais de uma imagem baseado na forma como nós seres humanos as exploramos. A nossa intenção com a proposição de tais métodos é apresentar um suposto novo paradigma em redução de ruído em imagens, que honestamente esperamos que passe a ser bastante utilizado daqui pra frente.

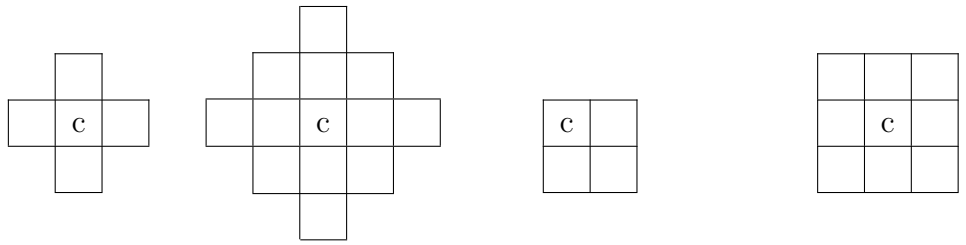
Para começarmos, a fim de tornar a leitura do capítulo mais clara, iremos realizar algumas definições a respeito da nomenclatura matemática que será utilizada. Após, discutiremos os pontos fortes e fracos de cada um dos métodos implementados neste capítulo com relação a cada tipo de ruído apresentado no capítulo 3, e faremos ainda uma análise comparativa desses métodos com os implementados nos capítulos 3 e 4.

5.1 Definições

Devido aos diversos tipos de convenções e notações diferentes encontrados na literatura, faremos aqui algumas definições que serão utilizadas neste capítulo, como pode ser visto a seguir.

Elemento estruturante

De acordo com [103], existem diversos tipos de elementos estruturantes. Contudo, no presente trabalho utilizaremos apenas os elementos estruturantes isotrópicos planos apresentados na figura 5.1.



(a) Disco de raio 1 (b) Disco de raio 2 (c) Quadrado de lado 2 (d) Quadrado de lado 3

Figura 5.1: Elementos estruturantes isotrópicos planos.

Tais elementos estruturantes serão utilizados para se representar o formato das janelas de observação que serão utilizadas neste capítulo. O centro de cada um dos elementos está sendo representado pela letra 'c'.

Erosão

Considerando $\mathbb{X}$ como sendo o multiconjunto cujos elementos $x[n]$ são os valores dos N pixels contidos na janela centrada em $X(d_1, d_2)$ representada pelo elemento estruturante E, denotaremos a erosão do pixel $X(d_1, d_2)$ por tal elemento por:

$$\epsilon_E[X(d_1, d_2)] \triangleq x[1] \in (\mathbb{X}, \leq), \tag{5.1}$$

onde $x[1]$ é o mínimo de $\mathbb{X}$. Diremos também que uma imagem X sofreu uma erosão pelo elemento estruturante E quando todos os pixels de tal imagem tiverem sofrido uma erosão por tal elemento. Denotaremos essa operação por $\epsilon_E(X)$.

Dilatação

Considerando $\mathbb{X}$ como sendo o multiconjunto cujos elementos $x[n]$ são os valores dos N pixels contidos na janela centrada em $X(d_1, d_2)$ representada pelo elemento estruturante E, denotaremos a dilatação do pixel $X(d_1, d_2)$ por tal elemento por:

$$\delta_E[X(d_1, d_2)] \triangleq x[N] \in (\mathbb{X}, \leq), \tag{5.2}$$

onde $x[N]$ é o máximo de $\mathbb{X}$. Diremos também que uma imagem X sofreu uma dilatação pelo elemento estruturante E quando todos os pixels de tal imagem tiverem sofrido uma dilatação por tal elemento. Denotaremos essa operação por $\delta_E(X)$.

Abertura

Diremos que uma imagem X sofreu uma abertura pelo elemento estruturante E quando a mesma tiver sofrido uma erosão pelo elemento estruturante E, seguida de uma dilatação pelo mesmo elemento. Denotaremos essa operação por $\gamma_E(X)$.

Fechamento

Diremos que uma imagem X sofreu um fechamento pelo elemento estruturante E quando a mesma tiver sofrido uma diltação pelo elemento estruturante E, seguida de uma erosão pelo mesmo elemento. Denotaremos essa operação por $\phi_E(X)$.

Conectividade

Definiremos conectividade para vizinhanças de 4 e de 8 pixels. Com relação à vizinhança de 4 pixels, diremos que dois pixels distintos $X(d_1, d_2)$ e $X(d_3, d_4)$ são diretamente 4-conexos quando as seguintes duas condições forem satisfeitas:

- $X(d_1, d_2) = X(d_3, d_4)$;
- $(|d_1 - d_3| = 1 \wedge |d_2 - d_4| = 0) \vee (|d_1 - d_3| = 0 \wedge |d_2 - d_4| = 1)$.

Diremos também que um conjunto de pixels $\mathbb{X}$ é 4-conexo quando cada um de seus elementos for diretamente 4-conexo com pelo menos um dos outros elementos do conjunto.

Ainda, diremos que dois pixels distintos $X(d_1, d_2)$ e $X(d_3, d_4)$ são 4-conexos quando tais pixels pertencerem a um mesmo conjunto 4-conexo.

Com relação à vizinhança de 8 pixels, diremos que dois pixels distintos $X(d_1, d_2)$ e $X(d_3, d_4)$ são diretamente 8-conexos quando as seguintes duas condições forem satisfeitas:

- $X(d_1, d_2) = X(d_3, d_4)$;
- $(|d_1 - d_3| \leq 1) \wedge (|d_2 - d_4| \leq 1)$.

Diremos também que um conjunto de pixels $\mathbb{X}$ é 8-conexo quando cada um de seus elementos for diretamente 8-conexo com pelo menos um dos outros elementos do conjunto.

Por fim, diremos que dois pixels distintos $X(d_1, d_2)$ e $X(d_3, d_4)$ são 8-conexos quando tais pixels pertencerem a um mesmo conjunto 8-conexo.

Tendo terminado todas as definições relativas a este capítulo, podemos agora explorar em detalhes os métodos baseados em morfologia matemática propostos.

5.2 Filtro morfológico

O filtro morfológico que estamos propondo se baseia no fato de que, de acordo com [101], atividades guiadas visualmente praticamente sempre giram em torno de objetos. Dessa forma, surge a necessidade da elaboração de um método que procure ao máximo conservar os objetos e suas formas.

Para idealizar tal método, precisaríamos primeiro fazer com que o mesmo fosse capaz de detectar os objetos e suas formas. Isso poderia ser feito utilizando-se o método de detecção de arestas apresentado no capítulo 4. Contudo, iremos utilizar um outro método baseado em morfologia matemática para realizar tal detecção.

Tal método pode ser descrito da seguinte maneira. Se considerarmos uma imagem X qualquer, podemos dizer que uma aproximação de tal imagem pode ser dada pela operação morfológica de abertura da mesma por um disco de raio 2 por exemplo. De posse dessa aproximação da imagem X, podemos agora encontrar os detalhes que foram perdidos pela operação de abertura diminuindo o valor de cada pixel da imagem original pelo valor do respectivo pixel de sua aproximação, conforme mostra a figura 5.2 a seguir:

(a) Imagem original (b) Abertura da imagem original (c) *Top-hat* da imagem original

Figura 5.2: Calculo do *top-hat* da imagem *Moedas* por um disco de raio 2.

Essa imagem formada pela imagem original menos a abertura da mesma é comumente chamada na literatura de *top-hat*, ou *white top hat*. Esse nome é dado pois, considerando $X(d_1, d_2)$ como a cota do ponto (d_1, d_2) do plano, podemos interpretar uma imagem como sendo um relevo, onde o ponto mais claro da imagem corresponde ao ponto mais alto do relevo, e o ponto escuro da mesma corresponde ao ponto mais baixo do relevo, estaremos encontramos apenas as pequenas saliências voltadas para fora de tal relevo tais que o elemento estruturante em questão, no caso um disco de raio 2, que esteja passando por debaixo do relevo não consiga entrar. Tal operação pode ser descrita através da seguinte equação:

$$\hat{X}_E = X - \gamma_E(X), \tag{5.3}$$

onde $\hat{X}_E$ representa o *top-hat* da imagem X pelo elemento estruturante E.

Embora tenhamos de maneira satisfatória conseguido encontrar alguns dos detalhes que dão forma aos objetos de uma imagem, podemos perceber que os mesmos representam apenas os detalhes relativos aos picos de tal imagem. Dessa forma, torna-se necessário se encontrar também os detalhes relativos aos vales de tal imagem. Para encontrá-los, podemos proceder da seguinte maneira. Se considerarmos a mesma imagem X em questão, podemos dizer que uma outra aproximação de tal imagem pode ser dada pela operação morfológica de fechamento da mesma por um disco de raio 2 por exemplo. De posse dessa outra aproximação da imagem X, podemos agora encontrar os detalhes que foram perdidos pela operação de fechamento diminuindo o valor de cada pixel dessa aproximação pelo valor do respectivo pixel da imagem original, conforme mostra a figura 5.3 a seguir:

(a) Fechamento da imagem original

(b) Imagem original

(c) *Bottom-hat* da imagem original

Figura 5.3: Calculo do *bottom-hat* da imagem *Moedas* por um disco de raio 2.

Essa imagem formada pela abertura da imagem original menos a imagem ori-

ginal é comumente chamada na literatura de *bottom-hat*, ou *black top hat*. Esse nome é dado pois, de maneira análoga à explicação dada para o *top-hat*, estaremos encontramos apenas as pequenas saliências voltadas para dentro do relevo tais que o elemento estruturante em questão, no caso um disco de raio 2, que esteja passando por cima do relevo não consiga entrar. Tal operação pode ser descrita através da seguinte equação:

$$\check{X}_E = \phi_E(X) - X, \tag{5.4}$$

onde $\check{X}_E$ representa o *bottom-hat* da imagem X pelo elemento estruturante E.

Agora que temos duas imagens, $\hat{X}_E$ e $\check{X}_E$, contendo detalhes da imagem X que "cabem" no elemento estruturante, podemos dizer que os detalhes dos objetos de tal imagem X estão basicamente contidos em $\hat{X}_E$ ou em $\check{X}_E$. Entretanto, podemos observar claramente que tais imagens também contém pixels corrompidos por ruído. É neste ponto que entra a parte crucial do raciocínio no qual se baseiam os métodos de redução de ruído em imagens propostos neste capítulo.

Ainda usando a analogia do relevo, por mais corrompido por ruído que estejam as imagens $\hat{X}_E$ e $\check{X}_E$, é de se esperar que tal ruído não consiga se apresentar em forma de montanhas em tal relevo, muito menos em forma de grandes montanhas [1]. Em contrapartida, os objetos que compõem as imagens muito provavelmente terão seus contornos, e conseqüentemente suas formas representadas por grandes montanhas. Dessa forma, podemos nos basear no tamanho dessas montanhas para realizarmos a redução de ruído em $\hat{X}_E$ e em $\check{X}_E$, sem no entanto removermos os contornos dos objetos de uma maneira um tanto quanto satisfatória. Assim, caso uma determinada montanha seja pequena o suficiente, ela será considerada como ruidosa e será removida de $\hat{X}_E$ ou de $\check{X}_E$.

Para determinarmos o tamanho de uma montanha, precisamos primeiro providenciar uma forma que possibilite analisarmos cada montanha individualmente. Para tal, utilizaremos como ferramenta de suporte duas máscaras, uma para $\hat{X}_E$ e

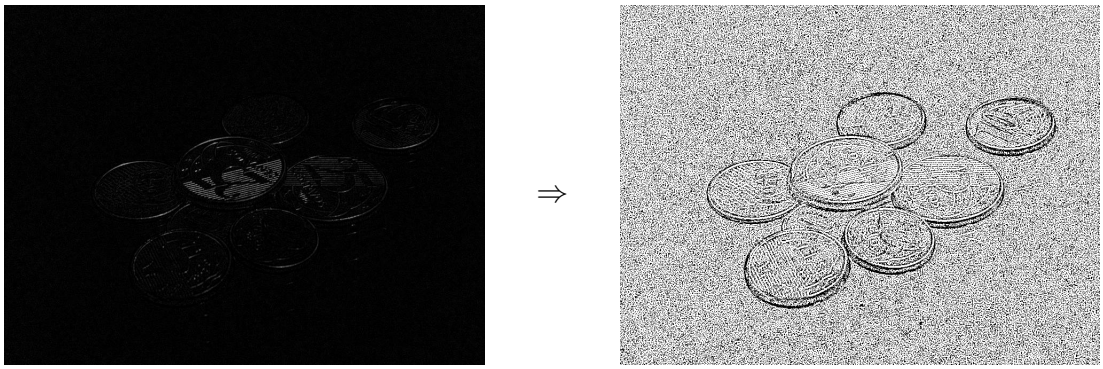
[1] Aqui nos referimos à grandes montanhas àquelas que tenham um grande volume.

outra para $\check{X}_E$, que serão geradas da seguinte forma:

$$\hat{M}(d_1, d_2) = \begin{cases} 1 & \text{, se } \hat{X}_E(d_1, d_2) > 0 \\ 0 & \text{, caso contrário} \end{cases} \tag{5.5}$$

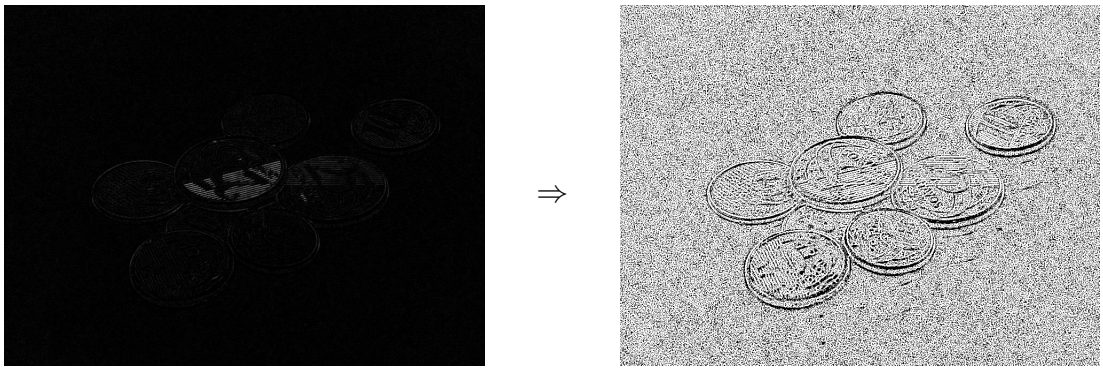
$$\check{M}(d_1, d_2) = \begin{cases} 1 & \text{, se } \check{X}_E(d_1, d_2) > 0 \\ 0 & \text{, caso contrário,} \end{cases} \tag{5.6}$$

onde $\hat{M}$ é a máscara que identifica as localições das montanhas da imagem $\hat{X}_E$ e $\check{M}$ é a máscara que identifica as localições das montanhas da imagem $\check{X}_E$. Na figura 5.4 a seguir podemos visualizar as máscaras das imagens $\hat{X}_E$ e $\check{X}_E$ calculadas de acordo com as equações 5.5 e 5.6.



(a) Top-hat da imagem original

(b) Máscara do top-hat da imagem original



(c) Bottom-hat da imagem original

(d) Máscara do bottom-hat da imagem original

Figura 5.4: *Top-hat* e *bottom-hat* da imagem *Moedas* por um disco de raio 2, e suas respectivas máscaras.

Com o auxílio da máscara $\check{M}$, podemos calcular o volume de cada uma das montanhas localizadas em $\hat{X}_E$ da seguinte maneira. Podemos primeiramente denotar

por $\hat{\mathbb{M}}_{d_1,d_2}$ o conjunto 8-conexo que contenha o elemento $\hat{M}(d_1, d_2)$. Uma importante observação neste ponto é que se $\hat{M}(d_1, d_2)$ e $\hat{M}(d_3, d_4)$ são 8-conexos, então eles fazem parte do mesmo conjunto, donde pode-se concluir que $\hat{\mathbb{M}}_{d_1,d_2} = \hat{\mathbb{M}}_{d_3,d_4}$. De posse de $\hat{\mathbb{M}}_{d_1,d_2}$, podemos agora denotar por $\hat{\mathbb{X}}_{d_1,d_2}$ o multiconjunto cujos elementos $\hat{x}_{d_1,d_2}[n]$ são os valores dos pixels de $\hat{X}_E$ cujas localizações sejam as mesmas dos elementos do conjunto $\hat{\mathbb{M}}_{d_1,d_2}$. Ao fazermos essa associação entre os elementos dos cojuntos $\hat{\mathbb{M}}_{d_1,d_2}$ e $\hat{\mathbb{X}}_{d_1,d_2}$, acabamos de conseguir separar as montanhas da imagem $\hat{X}_E$. Isso poderá ser percebido de maneira mais clara na equação onde realizamos o cálculo do volume das montanhas [2]. De acordo com tal equação, caso o volume de uma determinada montanha seja pequeno o suficiente, ela será considerada como ruidosa e será removida de $\hat{X}_E$:

$$\hat{F}_E(d_1, d_2) = \begin{cases} \hat{X}_E(d_1, d_2) & \text{, se } \sum_{n=1}^{N} \hat{x}_{d_1,d_2}[n] \geq \Upsilon \\ 0 & \text{, caso contrário,} \end{cases} \tag{5.7}$$

onde $\hat{F}_E$ é o *top-hat* filtrado da imagem X pelo elemento estruturante E e Υ é limiar de volume (escolhido arbitrariamente) que determina se uma montanha é considerada ruidosa ou não.

De forma análoga, poderíamos expandir o raciocínio da filtragem do *top-hat* para o *bottom-hat*. Tal expansão nos levaria à seguinte equação:

$$\check{F}_E(d_1, d_2) = \begin{cases} \check{X}_E(d_1, d_2) & \text{, se } \sum_{n=1}^{N} \check{x}_{d_1,d_2}[n] \geq \Upsilon \\ 0 & \text{, caso contrário,} \end{cases} \tag{5.8}$$

onde $\check{F}_E$ é o *bottom-hat* filtrado da imagem X pelo elemento estruturante E.

Neste ponto poderíamos dar prosseguimento ao processo de filtragem da imagem X. Contudo, a mesma seria bastante ineficiente. Podemos dizer isso porque como pode ser observado na figura 5.4, o processo de separar as montanhas foi realizado de forma insatisfatória. Isso porque existem várias pequenas saliências no relevo de X onde o elemento estruturamente não consegue entrar. Quando isso acontece, tanto no *top-hat* quanto no *bottom-hat*, as montanhas acabam ficando em grande parte

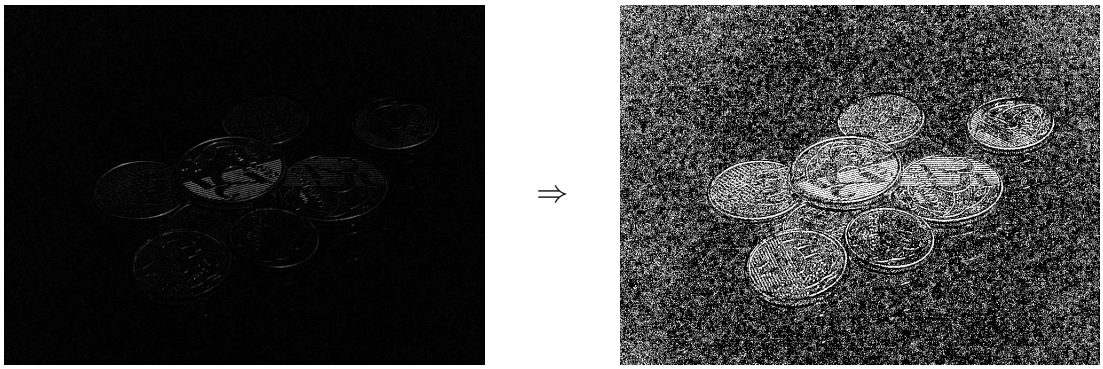
[2] Aqui consideramos o volume de uma montanha como o somatório dos valores dos pixels que a compõem.

conectadas, o que impede um funcionamento eficiente do filtro morfológico. Para evitar que isso ocorra, podemos substituir as equações 5.5 e 5.6 respectivamente pelas seguintes equações:

$$\hat{M}(d_1, d_2) = \begin{cases} 1 & \text{, se } \hat{X}_E(d_1, d_2) > H \\ 0 & \text{, caso contrário} \end{cases} \tag{5.9}$$

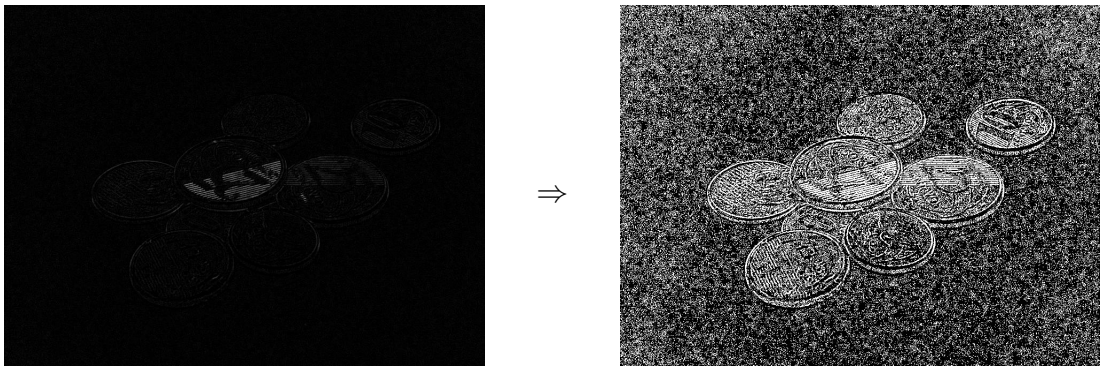
$$\check{M}(d_1, d_2) = \begin{cases} 1 & \text{, se } \check{X}_E(d_1, d_2) > H \\ 0 & \text{, caso contrário,} \end{cases} \tag{5.10}$$

onde H é um limiar de binarização das imagens $\hat{X}_E$ e $\check{X}_E$ escolhido arbitrariamente para promover uma melhor separação das montanhas. Na figura 5.5 a seguir podemos visualizar as máscaras das imagens $\hat{X}_E$ e $\check{X}_E$ calculadas de acordo com as equações 5.9 e 5.10 com $H = 5$.



(a) Top-hat da imagem original

(b) Máscara do top-hat da imagem original

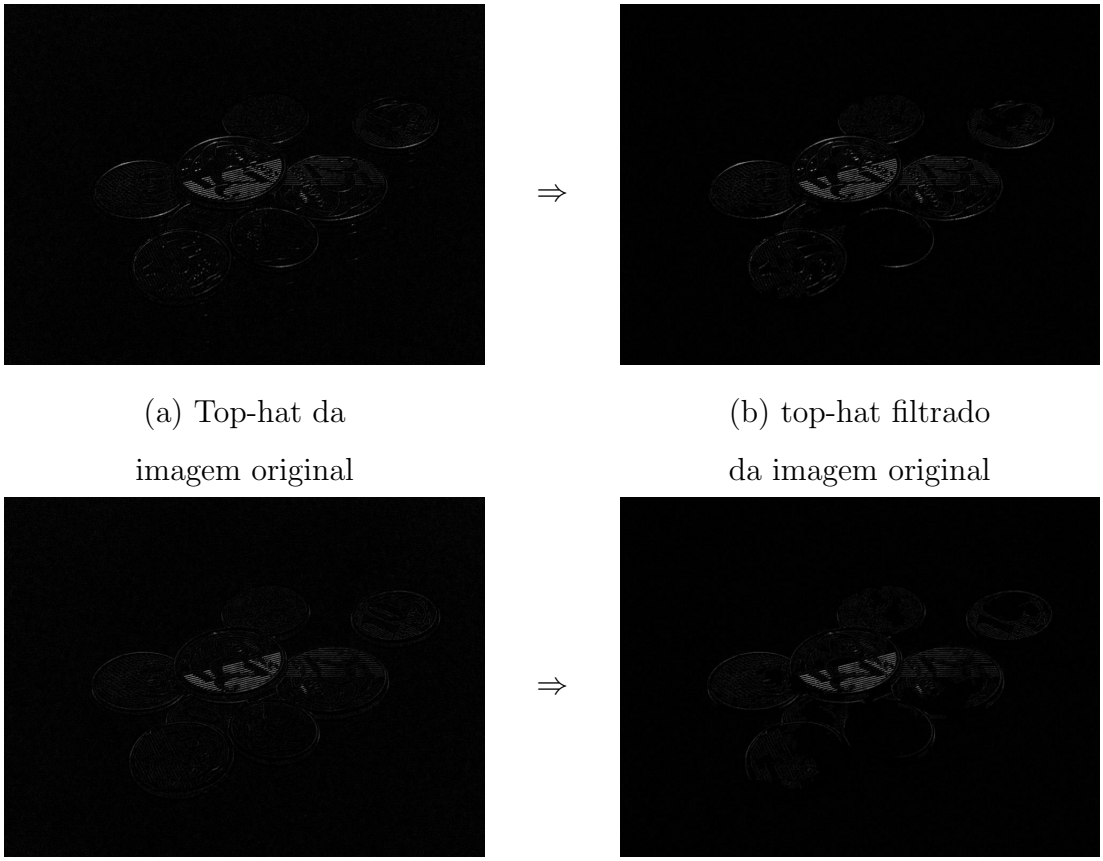


(c) Bottom-hat da imagem original

(d) Máscara do bottom-hat da imagem original

Figura 5.5: *Top-hat* e *bottom-hat* da imagem *Moedas* por um disco de raio 2, e suas respectivas máscaras ($H = 5$).

Com o auxílio dessas novas máscaras podemos agora realizar a filtragem das imagens $\hat{X}_E$ e $\check{X}_E$ de forma mais eficiente. A figura 5.6 a seguir mostra o resultado das filtragens de tais imagens utilizando-se as equações 5.9 e 5.10 com $H = 5$ e as equações 5.7 e 5.8 com $\Upsilon = 11500$.



(a) Top-hat da imagem original

(b) top-hat filtrado da imagem original

(c) Bottom-hat da imagem original

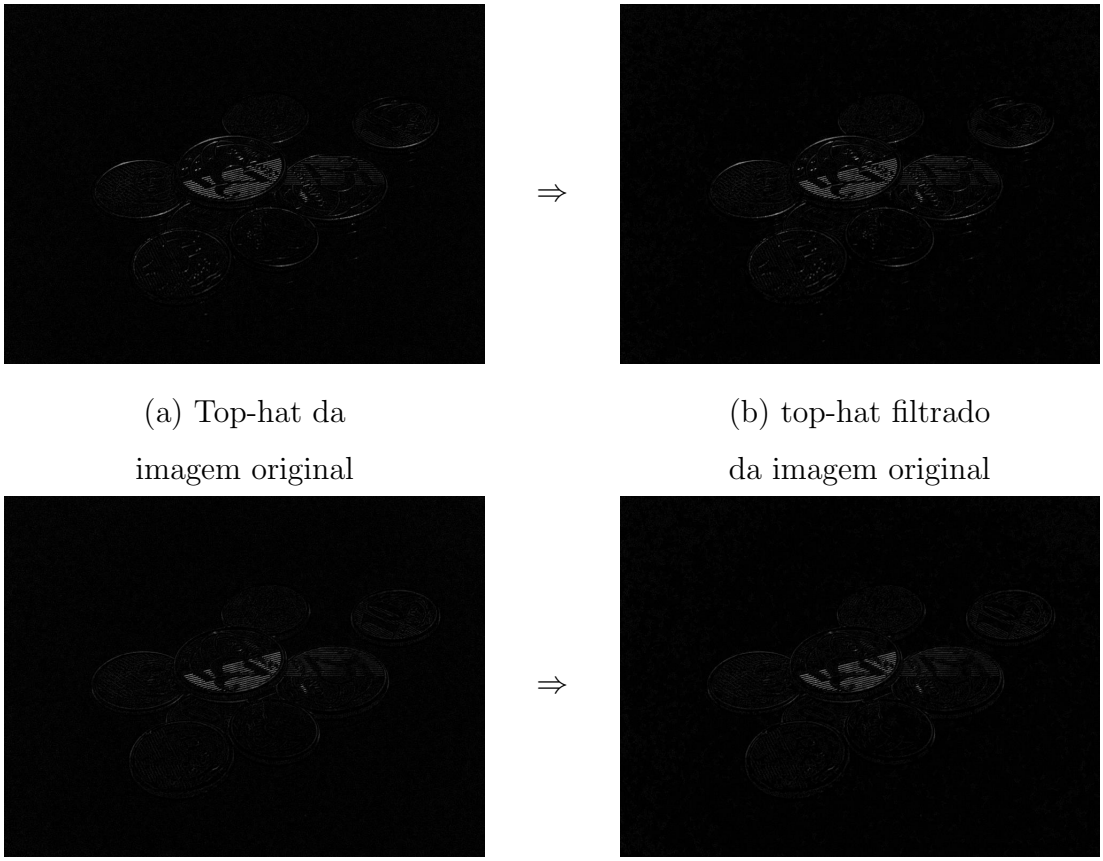
(d) Bottom-hat filtrado da imagem original

Figura 5.6: *Top-hat* e *bottom-hat* da imagem *Moedas* por um disco de raio 2, e suas respectivas versões filtradas ($H = 5$, $\Upsilon = 11500$).

Podemos perceber que grande parte do ruído presente em tais imagens foi suprimido com essa configuração. Em contrapartida, parte da textura de alguns objetos também foi perdida. Isso reflete o principal inconveniente do filtro morfológico, que considera formas muito pequenas como ruidosas e acaba removendo-as das imagens.

Para tentarmos minizar essa limitação do filtro morfológico, podemos diminuir o valor do limiar de volume Υ. Assim, mais montanhas tenderão a serem preservadas no processo de filtragem. Na figura 5.7 a seguir, podemos observar o resultado das filtragens das imagens $\hat{X}_E$ e $\check{X}_E$ utilizando-se as equações 5.9 e 5.10 com $H = 5$ e

as equações 5.7 e 5.8 agora com $\Upsilon = 150$.



(a) Top-hat da imagem original

(b) top-hat filtrado da imagem original

(c) Bottom-hat da imagem original

(d) Bottom-hat filtrado da imagem original

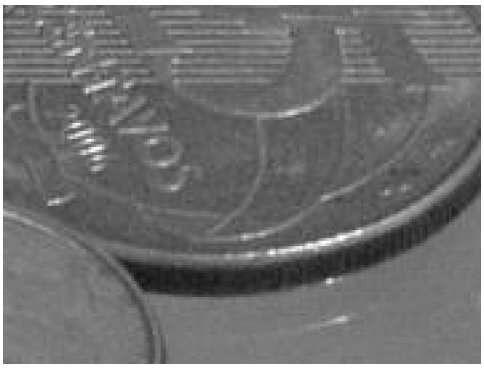
Figura 5.7: *Top-hat* e *bottom-hat* da imagem *Moedas* por um disco de raio 2, e suas respectivas versões filtradas ($H = 5$, $\Upsilon = 150$).

Podemos perceber que desta vez grande parte da textura dos objetos foi preservada. Entretanto, muito menos ruído foi suprimido quando comparamos estes resultados com os da figura 5.6. Isso leva a crer que existe uma relação de compromisso muito grande entre redução de ruído e preservação de formas no filtro morfológico, indicando que uma má escolha dos parâmetros para uma determinada imagem pode fazer com que os resultados sejam desastrosos.

Por fim, de posse das versões filtradas das imagens $\hat{X}_E$ e $\check{X}_E$, podemos finalmente gerar a imagem filtrada Y através da seguinte equação, que possibilita a reconstrução perfeita de X (prova no apêndice A.5):

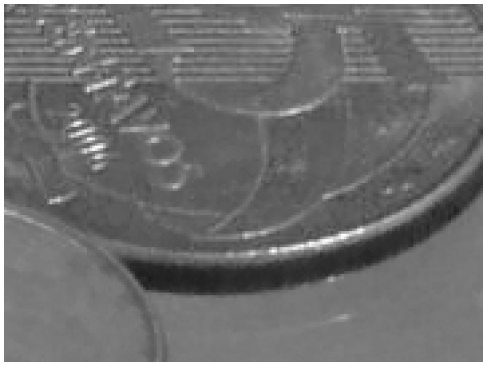
$$Y = \frac{\gamma_E(X) + \phi_E(X) + \hat{F}_E - \check{F}_E}{2} \tag{5.11}$$

A figura 5.8 a seguir mostra os resultados da filtragem da imagem *Moedas* pelo filtro morfológico, por um disco de raio 2 com diferentes valores de H e de Υ. Para a implementação do filtro foram utilizadas as equações 5.3, 5.4, 5.9, 5.10, 5.7, 5.8 e 5.11. O código do referido filtro, utilizado no software *Image Denoiser*, pode ser visto na seção C.8.

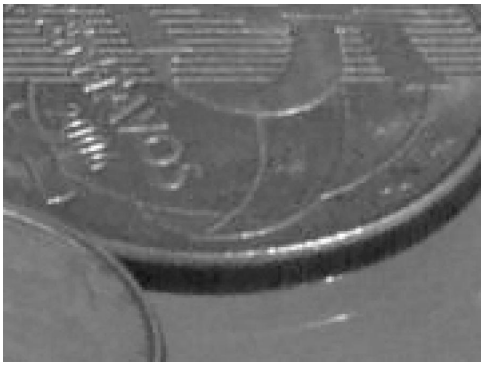


(a) Imagem original

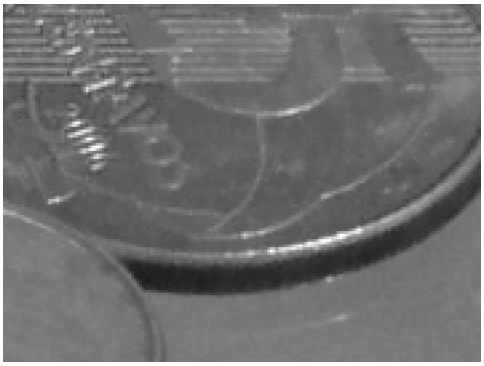
(b) Imagem filtrada ($H = 3$, $\Upsilon = 150$)



(c) Imagem filtrada ($H = 3$, $\Upsilon = 11500$)



(d) Imagem filtrada ($H = 5$, $\Upsilon = 150$)



(e) Imagem filtrada ($H = 5$, $\Upsilon = 11500$)

Figura 5.8: Detalhe do resultado da filtragem da imagem *Moedas* pelo filtro morfológico (disco de raio 2).

Ao compararmos os dois discutidos em detalhes até aqui (figuras 5.8d e 5.8e),

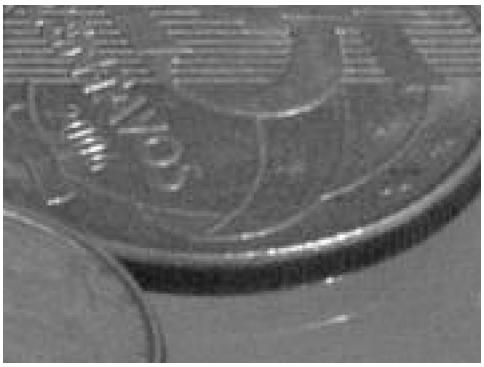
podemos confirmar toda a discussão realizada anteriormente sobre o compromisso entre redução de ruído e preservação de formas no filtro morfológico. De fato, podemos observar que houve uma maior redução de ruído e uma perda maior de detalhes na imagem da figura 5.8e do que na imagem da figura 5.8d. Contudo, a perda de detalhes, que era bastante visível nas imagens relativas ao *top-hat* e ao *bottom-hat*, se mostrou muito menos perceptível na imagem filtrada Y, o que mostra que o inconveniente do filtro morfológico, de considerar formas muito pequenas como ruidosas e removê-las das imagens, não é tão grave assim quanto parece na prática.

Uma outra importante análise pode ser feita ao compararmos as imagens filtradas das figuras 5.8c e 5.8e. Nelas podemos perceber que para um mesmo valor de Υ, quanto maior é o valor de H, maior é a redução de ruído e de pequenas formas de uma imagem. Isso pode ser explicado pelo fato que quanto maior é o valor de H, menores tendem a ser os tamanhos das montanhas, já que as mesmas tendem a serem compostas por menos pixels, o que faz com que mais montanhas tendam a ser eliminadas no processo de filtragem.

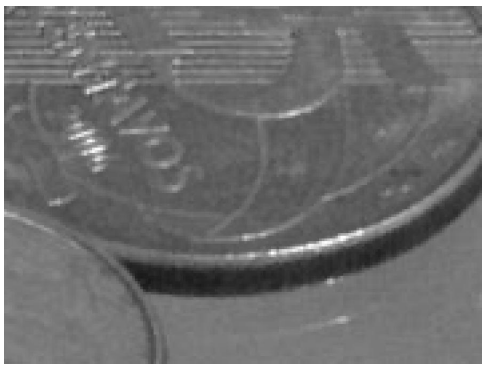
No mais, vale lembrar que poderíamos ter escolhido ainda outro elemento estruturante, além do disco de raio 2, para realizarmos a redução de ruído das imagens. A figura 5.9 mostra os resultados da filtragem da imagem *Moedas* pelo filtro morfológico com $H = 5$ e $\Upsilon = 1000$, por diferentes elementos estruturantes.

Ao analisarmos as imagens da figura 5.9 fica muito difícil perceber diferenças significativas entre as mesmas. Dessa forma, não é possível estabelecer uma relação muito clara entre os diferentes efeitos causados pelos diferentes elementos estruturantes. A princípio só nos é possível afirmar que, para variações de tamanho de um mesmo elemento estruturante, quanto maior for o tamanho de tal elemento, maiores serão as montanhas formadas nos relevos relativos às imagens $\hat{X}_E$ e $\check{X}_E$, e consequentemente, para um mesmo valor de H e de Υ, menor será a redução de ruído e de pequenas formas de uma imagem.

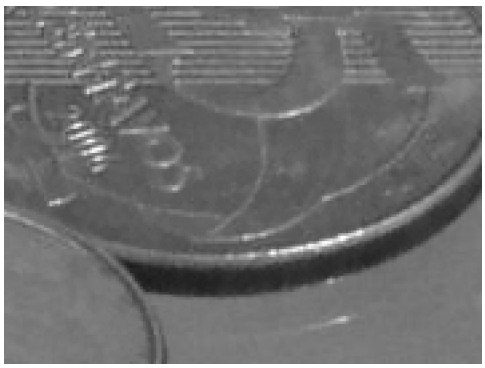
Na comparação com os outros métodos implementados nos capítulos 3 e 4, podemos dizer, de uma maneira geral, que o filtro morfólogico é aparentemente o filtro que consegue realizar uma maior redução de ruído em imagens para uma mesma qualidade perceptual de uma imagem. Pelo que parece também, o filtro que consegue obter resultados mais próximos do filtro morfológico é o filtro baseado em *wavelets*



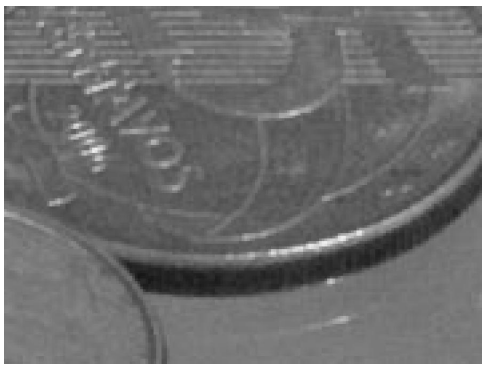
(a) Imagem original



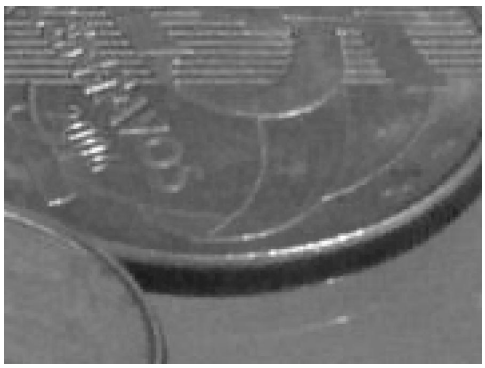
(b) Imagem filtrada (disco de raio 1)



(c) Imagem filtrada (disco de raio 2)



(d) Imagem filtrada (quadrado de lado 2)



(e) Imagem filtrada (quadrado de lado 3)

Figura 5.9: Detalhe do resultado da filtragem da imagem *Moedas* pelo filtro morfológico ($H = 5$, $\Upsilon = 1000$).

proposto da seção 4.4. Entretanto, a perda de detalhes por este ocasionada, assim como os artefatos inseridos pelo mesmo comprometem a qualidade subjetiva da imagem filtrada num grau visivelmente maior do que o comprometimento ocasionado pelo filtro morfológico.

Com relação a imagens com ruído artificial, as tabelas 5.1, 5.2, 5.3 e 5.4 mostram que o filtro morfológico é o melhor filtro para ruído de quantização. Isso talvez

possa ser explicado pela fato de que mudanças abruptas no relevo, ocasionadas pelo processo de quantização, são facilmente detectadas pelo filtro morfológico e possivelmente eliminadas no processo de filtragem. Já com relação ao ruído do tipo "sal e pimenta", o resultado do filtro morfológico é um dos piores de todos, até mesmo com relação à qualidade subjetiva da imagem filtrada. A princípio, isso não deveria acontecer, já que os pixels corrompidos pelo ruído impulsivo em questão tendem a ter boa parte de seus valores discrepantes representados nas imagens $\hat{X}_E$ e $\check{X}_E$, e, quando isso acontece, tais discrepâncias podem ser facilmente eliminadas no processo de filtragem, nem que para isso seja necessário eliminar todas as formas presentes em tais imagens. Dessa forma, o fato do resultado em termos de PSNR para tal tipo de ruído ter sido tão ruim assim, nos remete ao pensamento de que talvez as aproximações da imagem X utilizadas pelo filtro morfológico não sejam as mais úteis, o que nos leva a crer que talvez se utilizássemos outras aproximações da imagem X, poderíamos conseguir resultados melhores. No mais, com relação aos demais tipos de ruído o filtro morfológico teve um bom desempenho no tocante ao PSNR.

Para uma análise mais completa dos resultados do filtro morfológico, recomenda-se a leitura da seção B.10.

5.3 Filtro morfológico II

O filtro morfológico II se trata de uma variação do filtro morfológico explicado na seção 5.2. O atual filtro se difere do outro apenas na maneira como são calculados o *top-hat* e o *bottom-hat* da imagem X. Para tal, o filtro morfológico II utilizada outras aproximações da imagem X, que fazem com que as equações 5.3 e 5.4 sejam substituídas respectivamente pelas seguintes equações:

$$\hat{X}_E = \begin{cases} X - \phi_E[\gamma_E(X)] & \text{, se } X - \phi_E[\gamma_E(X)] > 0 \\ 0 & \text{, caso contrário} \end{cases} \tag{5.12}$$

$$\check{X}_E = \begin{cases} \gamma_E[\phi_E(X)] - X & \text{, se } \gamma_E[\phi_E(X)] - X > 0 \\ 0 & \text{, caso contrário} \end{cases} \tag{5.13}$$

Como $X - \phi_E[\gamma_E(X)]$ e $\gamma_E[\phi_E(X)] - X$ podem apresentar valores negativos, essa

variação do filtro morfológico opta por considerar apenas os valores positivos dos mesmos para o cálculo das imagens $\hat{X}_E$ e $\check{X}_E$. Contudo, seria perfeitamente possível considerar valores negativos para o cálculo das mesmas, como poderemos ver mais adiante na seção 5.4.

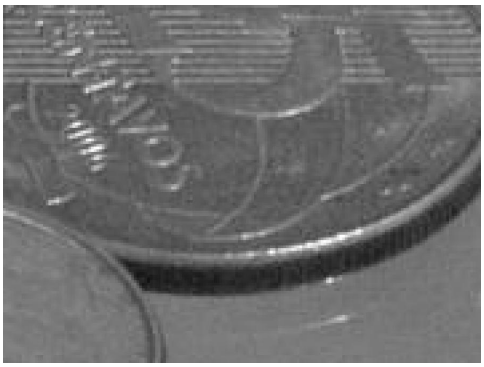
De posse agora das novas versões das imagens $\hat{X}_E$ e $\check{X}_E$, podemos gerar a imagem filtrada Y através da equação a seguir, que possibilita a reconstrução perfeita de X (prova no apêndice A.6):

$$Y = \frac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)] + |\gamma_E[\phi_E(X)] - X| - |X - \phi_E[\gamma_E(X)]|}{2} + \hat{F}_E - \check{F}_E \tag{5.14}$$

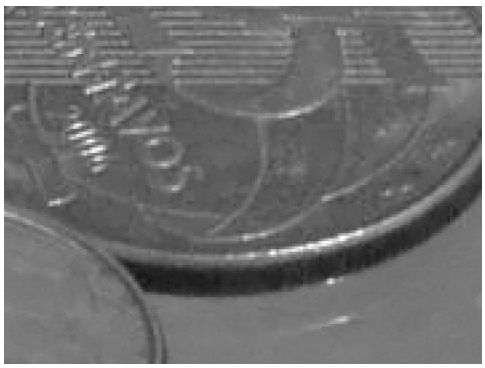
A figura 5.10 a seguir mostra os resultados da filtragem da imagem *Moedas* pelo filtro morfológico II, por um disco de raio 2 com diferentes valores de H e de Υ. Para a implementação do filtro foram utilizadas as equações 5.12, 5.13, 5.9, 5.10, 5.7, 5.8 e 5.14. O código do referido filtro, utilizado no software *Image Denoiser*, pode ser visto na seção C.9.

Se compararmos os resultados do filtro morfológico II com os do filtro implementado na seção 5.2, perceberemos que, para um mesmo valor de H e de Υ, o atual filtro proporciona uma redução de ruído e de pequenas formas em uma imagem bem maior do que a proporcionada pelo seu antecessor. Isso nos dá um indício de que as aproximações da imagem X feitas pelo filtro morfológico II são mais úteis do que as aproximações realizadas pelo filtro morfológico da seção 5.2. Tal consideração acaba nos levando a crer que precisaríamos diminuir o valor de Υ no atual filtro se quiséssemos manter um grau de redução de ruído em imagens similar ao grau apresentado pelo filtro morfológico original. Na comparação com os outros filtros implementados nos capítulos 3 e 4, todas as considerações feitas na seção 5.2 também valem para filtro morfológico II.

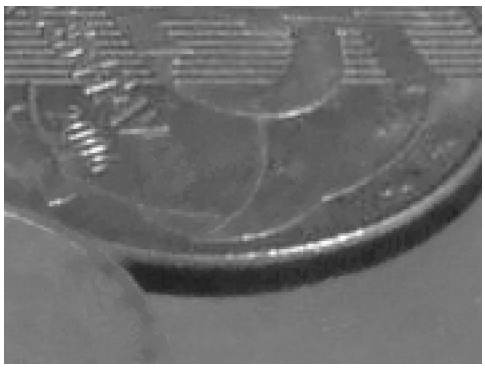
Com relação a imagens com ruído artificial, as tabelas 5.1, 5.2, 5.3 e 5.4 mostram que as novas aproximações da imagem X, dadas por $\phi_E[\gamma_E(X)]$ e por $\gamma_E[\phi_E(X)]$, de fato melhoraram muito o desempenho do atual filtro com relação ao ruído do tipo "sal e pimenta", se comparado com o filtro da seção anterior. Contudo, apesar dos desempenhos com relação a todos os outros tipos de ruído serem bons, os mesmos cairam se comparados aos resultados do filtro morfológico original, o que mostra que



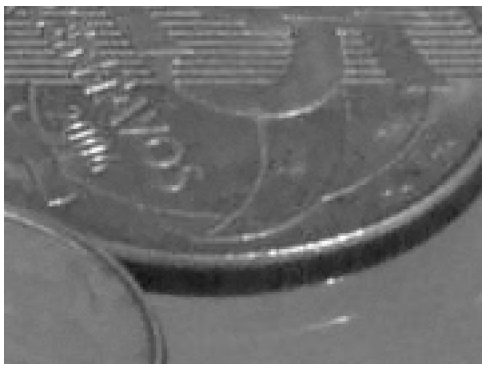
(a) Imagem original



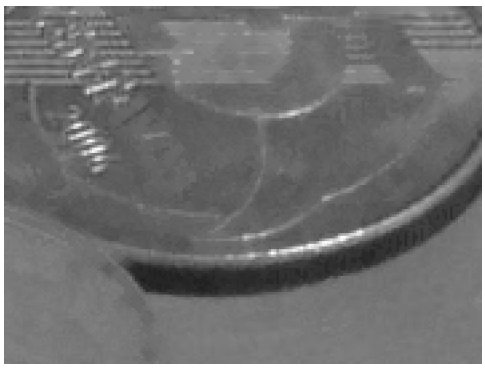
(b) Imagem filtrada ($H = 3$, $\Upsilon = 150$)



(c) Imagem filtrada ($H = 3$, $\Upsilon = 11500$)



(d) Imagem filtrada ($H = 5$, $\Upsilon = 150$)



(e) Imagem filtrada ($H = 5$, $\Upsilon = 11500$)

Figura 5.10: Detalhe do resultado da filtragem da imagem *Moedas* pelo filtro morfológico II (disco de raio 2).

talvez a opção feita de considerar apenas os valores positivos de $X - \phi_E[\gamma_E(X)]$ e de $\gamma_E[\phi_E(X)] - X$ para o cálculo das imagens $\hat{X}_E$ e $\check{X}_E$ não tenha sido uma boa escolha. Podemos dizer isso já que com relação a tais tipos de ruído, é um tanto quanto intuitivo pensar que os pixels contidos nas imagens $\hat{X}_E$ e $\check{X}_E$ tenham que ser tratados com muito mais atenção do que os mesmos teriam que ser tratados se estivéssemos trabalhando com o ruído do tipo "sal e pimenta". De qualquer forma,

é importante enfatizar que os resultados do atual filtro não deixam de ser bons para todos os tipos de ruído.

Para uma análise mais completa dos resultados do filtro morfológico II, recomenda-se a leitura da seção B.11.

5.4 Filtro morfológico III

O filtro morfológico III se trata de uma outra variação do filtro morfológico explicado na seção 5.2. O atual filtro, mais uma vez, se difere do outro na maneira como são calculados o *top-hat* e o *bottom-hat* da imagem X. Para tal, o filtro morfológico III utilizada outras aproximações da imagem X, que fazem com que as equações 5.3 e 5.4 sejam substituídas respectivamente pelas seguintes equações:

$$\hat{X}_E = X - \phi_E[\gamma_E(X)] \tag{5.15}$$

$$\check{X}_E = \gamma_E[\phi_E(X)] - X \tag{5.16}$$

Podemos observar que o atual filtro, diferentemente do filtro exposto na seção 5.3, opta por considerar valores negativos de $X - \phi_E[\gamma_E(X)]$ e de $\gamma_E[\phi_E(X)] - X$ para o cálculo das imagens $\hat{X}_E$ e $\check{X}_E$ respectivamente. Ao fazermos essa consideração, passamos a ter que fazer uma pequena mudança nas equações 5.7 e 5.8 para que possamos calcular os volumes de montanhas que possuam valores de pixels negativos. Dessa forma, as referidas equações passam a ser substituídas respectivamente pelas seguintes equações:

$$\hat{F}_E(d_1, d_2) = \begin{cases} \hat{X}_E(d_1, d_2) & \text{, se } \sum_{n=1}^{N} |\hat{x}_{d_1,d_2}[n]| \geq \Upsilon \\ 0 & \text{, caso contrário} \end{cases} \tag{5.17}$$

$$\check{F}_E(d_1, d_2) = \begin{cases} \check{X}_E(d_1, d_2) & \text{, se } \sum_{n=1}^{N} |\check{x}_{d_1,d_2}[n]| \geq \Upsilon \\ 0 & \text{, caso contrário} \end{cases} \tag{5.18}$$

Agora, podemos gerar a imagem filtrada Y através da equação a seguir, que

possibilita a reconstrução perfeita de X (prova no apêndice A.7):

$$Y = \frac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)] + \hat{F}_E - \check{F}_E}{2} \tag{5.19}$$

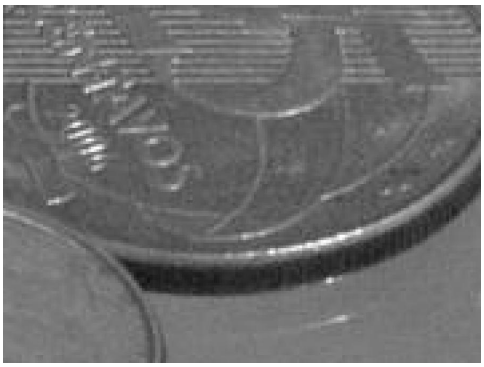
A figura 5.11 a seguir mostra os resultados da filtragem da imagem *Moedas* pelo filtro morfológico III, por um disco de raio 2 com diferentes valores de H e de Υ. Para a implementação do filtro foram utilizadas as equações 5.15, 5.16, 5.9, 5.10, 5.17, 5.18 e 5.19. O código do referido filtro, utilizado no software *Image Denoiser*, pode ser visto na seção C.10.

Se compararmos os resultados do filtro morfológico III com os do filtro implementado na seção 5.3, perceberemos que, para um mesmo valor de H e de Υ, o atual filtro proporciona uma redução de ruído na imagem semelhante e uma redução de pequenas formas na mesma ainda menor do que a proporcionada pelo seu antecessor. Isso nos faz acreditar que encontramos um bom compromisso entre todas as variáveis envolvidas na concepção de um filtro morfológico de uma maneira geral. Na comparação com os outros filtros implementados nos capítulos 3 e 4, todas as considerações feitas na seção 5.2 também valem para filtro morfológico III.

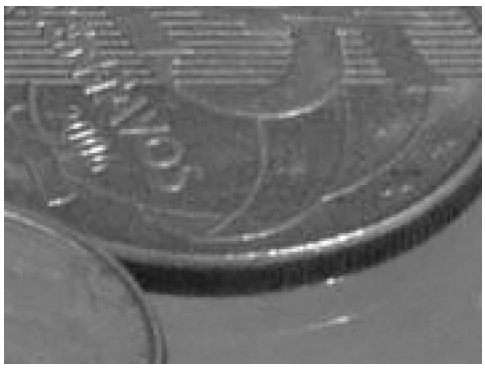
Com relação a imagens com ruído artificial, as tabelas 5.1 e 5.2 mostram que, dentre os filtros morfológicos propostos, o filtro morfológico III é o melhor filtro em termos de PSNR em 40% dos casos, o segundo melhor em 40% dos casos e o terceiro melhor em 20% dos casos. Isso demonstra um desempenho bastante expressivo do atual filtro. Numa comparação com todos os filtros implementados, as tabelas 5.3 e 5.4 mostram também que o filtro morfológico III é o melhor filtro para ruído do tipo "sal e pimenta", além de ser o segundo melhor para ruído de quantização e o terceiro melhor para dois outros casos. Isso definitivamente coloca o atual filtro no posto de melhor filtro dentre todos os outros filtros implementados [3].

Para uma análise mais completa dos resultados do filtro morfológico III, recomenda-se a leitura da seção B.12.

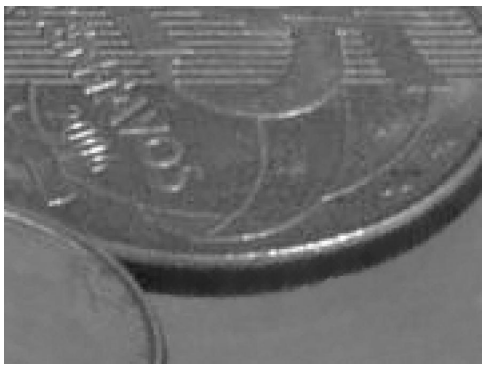
[3]Se levarmos em consideração apenas os resultados objetivos em termos de PSNR, fica claro que o melhor filtro é o filtro sigma aditivo, já que o mesmo é o melhor em 40% dos casos. Contudo, a afirmação de que o filtro morfológico III é o melhor filtro dentre todos os filtros implementados foi feita levando-se em conta também os resultados subjetivos, onde aparentemente o melhor resultado exposto, dentre todos os resultados expostos até então, foi o do filtro morfológico III.



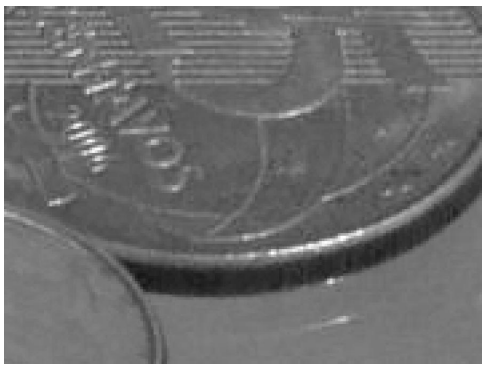
(a) Imagem original



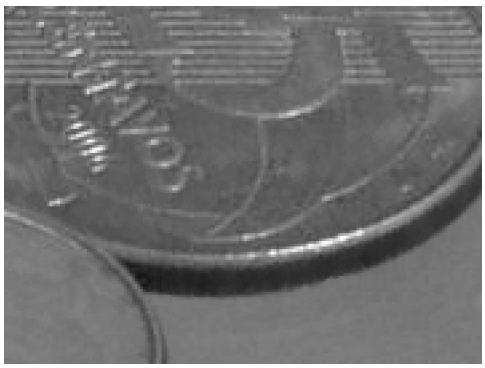
(b) Imagem filtrada ($H = 3$, $\Upsilon = 150$)



(c) Imagem filtrada ($H = 3$, $\Upsilon = 11500$)



(d) Imagem filtrada ($H = 5$, $\Upsilon = 150$)



(e) Imagem filtrada ($H = 5$, $\Upsilon = 11500$)

Figura 5.11: Detalhe do resultado da filtragem da imagem *Moedas* pelo filtro morfológico III (disco de raio 2).

5.5 Filtro morfológico IV

O filtro morfológico IV se trata da última variação que será proposta do filtro morfológico explicado na seção 5.2. O atual filtro se difere do outro apenas na maneira como são calculados o *top-hat* e o *bottom-hat* da imagem X. Para tal, o filtro morfológico IV utilizada outras aproximações da imagem X, que fazem com que as

equações 5.3 e 5.4 sejam substituídas respectivamente pelas seguintes equações:

$$\hat{X}_E = \begin{cases} X - \dfrac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)]}{2} & \text{, se } X - \dfrac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)]}{2} > 0 \\ 0 & \text{, caso contrário} \end{cases} \tag{5.20}$$

$$\check{X}_E = \begin{cases} -\left(X - \dfrac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)]}{2}\right) & \text{, se } X - \dfrac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)]}{2} < 0 \\ 0 & \text{, caso contrário} \end{cases} \tag{5.21}$$

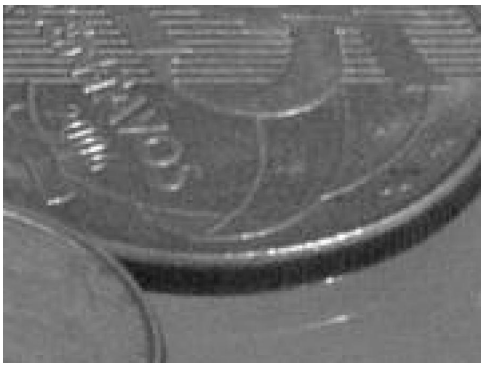
De posse agora das novas versões das imagens $\hat{X}_E$ e $\check{X}_E$, podemos gerar a imagem filtrada Y através da equação a seguir, que possibilita a reconstrução perfeita de X (prova no apêndice A.8):

$$Y = \frac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)]}{2} + \hat{F}_E - \check{F}_E \tag{5.22}$$

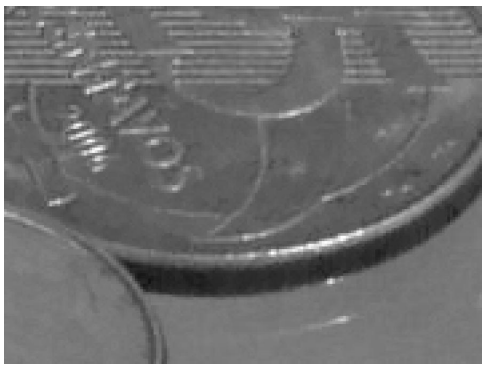
A figura 5.12 a seguir mostra os resultados da filtragem da imagem *Moedas* pelo filtro morfológico IV, por um disco de raio 2 com diferentes valores de H e de Υ. Para a implementação do filtro foram utilizadas as equações 5.20, 5.21, 5.9, 5.10, 5.7, 5.8 e 5.22. O código do referido filtro, utilizado no software *Image Denoiser*, pode ser visto na seção C.11.

Se compararmos os resultados do filtro morfológico IV com os dos outros filtros implementados neste capítulo, perceberemos que, para um mesmo valor de H e de Υ, o atual filtro proporciona uma redução de ruído e de pequenas formas em uma imagem bem maior do que as proporcionadas pelos seus antecessores. Isso nos dá um indício de que as aproximações da imagem X feitas pelo filtro morfológico IV são melhores do que as aproximações realizadas pelos outros filtros morfológicos aqui propostos. Tal consideração acaba nos levando a crer que precisaríamos diminuir o valor de Υ no atual filtro se quiséssemos manter um grau de redução de ruído em imagens similar ao grau apresentado pelos outros filtros morfológicos. Na comparação com os outros filtros implementados nos capítulos 3 e 4, todas as considerações feitas na seção 5.2 também valem para filtro morfológico IV.

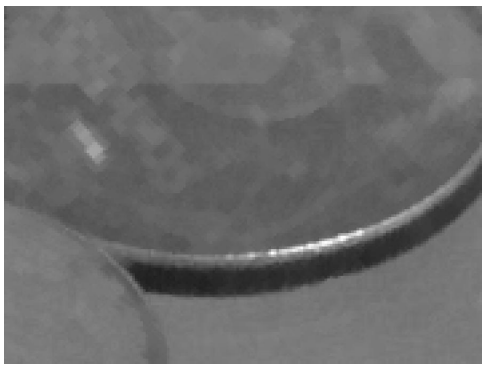
Com relação a imagens com ruído artificial, as tabelas 5.1 e 5.2 mostram que, dentre os filtros morfológicos propostos, o filtro morfológico IV é o melhor filtro



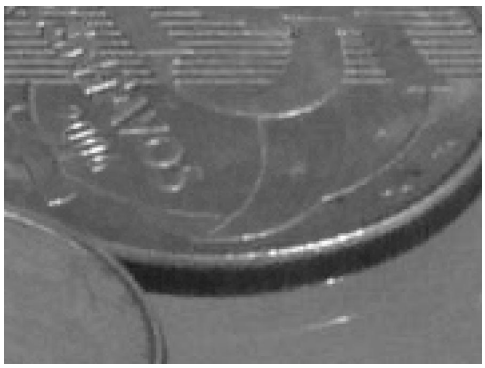
(a) Imagem original



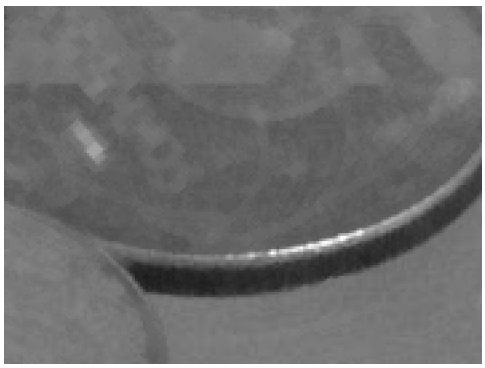
(b) Imagem filtrada ($H = 3$, $\Upsilon = 150$)



(c) Imagem filtrada ($H = 3$, $\Upsilon = 11500$)



(d) Imagem filtrada ($H = 5$, $\Upsilon = 150$)



(e) Imagem filtrada ($H = 5$, $\Upsilon = 11500$)

Figura 5.12: Detalhe do resultado da filtragem da imagem *Moedas* pelo filtro morfológico IV (disco de raio 2).

para ruído relativo a granulação em fotografias, apresentando um desempenho um pouco mais fraco do que o filtro morfológico III em todas as outras situações. Numa comparação com todos os filtros implementados, as tabelas 5.3 e 5.4 mostram que o filtro morfológico IV teve um desempenho apenas regular no tocante ao PSNR com relação a todos os tipos de ruído apresentados.

Para uma análise mais completa dos resultados do filtro morfológico IV,

recomenda-se a leitura da seção B.13.

5.6 Conclusões

Ao longo do capítulo, pudemos observar que os métodos baseados em morfologia matemática conseguem realizar de forma aparentemente satisfatória a redução de ruído em imagens ao preço de remover alguns pequenos detalhes de tais imagens. Contudo, a perda de detalhes, que era bastante visível nas imagens relativas ao *top-hat* e ao *bottom-hat*, se mostrou muito menos perceptível na imagem filtrada Y, o que mostra que o inconveniente do filtro morfológico, de considerar formas muito pequenas como ruidosas e acabar removendo-as das imagens, não é tão grave assim quanto parece.

Além disso, os filtros morfológicos possibilitam também o ajuste de seus 3 parâmetros, isto é, do elemento estruturante E, do limiar de binarização H e do limiar de volume Υ, para a realização da filtragem de uma imagem. Essa possibilidade faz com que, em geral, seja possível se atingir um bom compromisso entre o grau de redução de ruído e a quantidade de detalhes removidos de uma determinada imagem.

Infelizmente, com relação aos elementos estruturantes, não foi possível se estabelecer nenhuma relação clara entre os diferentes efeitos causados pelos diferentes elementos estruturantes. A princípio só se é possível afirmar que, quanto maior for o tamanho de um mesmo elemento estruturante, maiores serão as montanhas formadas nos relevos relativos às imagens $\hat{X}_E$ e $\check{X}_E$, e, consequentemente, para um mesmo valor de H e de Υ, menor será a redução de ruído e de pequenas formas de uma imagem.

Em contraposição aos elementos estruturantes, felizmente foi possível se chegar a algumas conclusões interessantes com relação aos parâmetros H e Υ. De uma maneira geral, podemos sumarizar os efeitos de tais parâmetros dizendo que o limiar de binarização H determina quais serão as montanhas que serão *candidatas* a serem eliminadas, bem como suas formas e seus volumes, e que o limiar de volume Υ determina quais serão as montanhas que serão *de fato* eliminadas. Mais especificamente, quanto maior for o valor do parâmetro H, menor tenderá a ser o número de monta-

nhas canditatas a eliminação e menores serão o volume delas. Com relação ao limiar de volume Υ, quanto maior for este, maior tenderá a ser o número de montanhas que serão eliminadas. Essas constatações nos levam a crer que se aumentarmos o valor de H, precisaremos diminuir o valor de Υ, e vice-versa, se quisermos manter um grau similar de redução de ruído em imagens.

Foi possível perceber ainda que quanto melhores forem as aproximações feitas da imagem X, menores deverão ser os valores de H e de Υ para podermos manter um grau similar de redução de ruído numa determinada imagem. Além disso, embora não tenhamos desenvolvido nenhum método automático para a configuração dos 3 parâmetros dos filtros morfológicos, o curto tempo de execução dos mesmos faz com que seja possível a utilização de uma ferramenta que varie tais parâmetros até que se chegue a um resultado satisfatório.

Por fim, é importante dizer que há muito ainda o que ser explorado com relação aos filtros morfológicos. Aqui, optou-se por usar como critério para a redução de ruído em imagens o volume das montanhas contidas nos relevos relativos às imagens *top-hat* e *bottom-hat*. Contudo, é evidente perceber que diversos outros critérios poderiam ser perfeitamente utilizados para tal.

	T	$S\&P$	Q	F	G
Morfológico	**32,6695**	24,2671	**29,7433**	**32,0226**	34,5837
Morfológico II	31,2561	30,0277	29,4906	30,4906	33,5643
Morfológico III	32,4112	**33,2009**	29,7033	32,0056	34,5753
Morfológico IV	32,3198	33,0986	29,5621	31,9305	**34,8339**

Tabela 5.1: Melhores resultados de PSNR dos filtros morfológicos (imagem *Lena*).

	T	$S\&P$	Q	F	G
Morfológico I	32,3706	23,8653	**29,8528**	32,1996	35,1106
Morfológico II	31,6266	30,3444	29,5675	31,3249	34,0093
Morfológico III	**32,8965**	**33,7602**	29,7816	**32,9418**	34,6502
Morfológico IV	32,7742	33,6442	29,6789	32,8559	**35,3297**

Tabela 5.2: Melhores resultados de PSNR dos filtros morfológicos (imagem *Peppers*).

	T	$S\&P$	Q	F	G
Mediana	32,4525	32,8072	28,9034	31,9231	33,8332
Média	31,1684	26,6758	27,9647	30,9393	31,6481
Sigma aditivo	**33,6237**	32,5366	29,5406	32,9737	**35,5917**
Sigma adaptativo	31,4326	29,7256	29,5845	30,4692	34,1125
Sigma multiplicativo	33,2355	31,3384	29,4301	**33,1126**	35,4824
Sigma híbrido	32,8070	32,0993	29,5641	32,0577	34,7661
Wavelets	25,2096	15,1314	28,3566	23,8778	28,5721
Wavelets modificado	27,1641	16,2713	28,5916	26,0006	29,7947
Wavelets proposto	29,6987	21,3223	28,7212	28,9960	31,2592
Morfológico	32,6695	24,2671	**29,7433**	32,0226	34,5837
Morfológico II	31,2561	30,0277	29,4906	30,4906	33,5643
Morfológico III	32,4112	**33,2009**	29,7033	32,0056	34,5753
Morfológico IV	32,3198	33,0986	29,5621	31,9305	34,8339

Tabela 5.3: Comparativo entre os melhores resultados de PSNR dos filtros estatísticos, dos filtros baseados em *wavelets* e dos filtros morfológicos (imagem *Lena*).

	T	$S\&P$	Q	F	G
Mediana	32,7621	33,1648	29,1780	32,6508	34,4521
Média	31,1745	26,5962	28,1018	31,1763	31,7661
Sigma aditivo	**34,0606**	32,3282	29,6507	33,5984	**35,8916**
Sigma adaptativo	31,5821	29,8922	29,7741	31,2572	34,6830
Sigma multiplicativo	32,9704	29,2722	29,4649	**33,7965**	35,4274
Sigma híbrido	32,8316	31,7653	29,7271	32,6740	34,9422
Wavelets	24,6637	14,7654	27,1751	24,0776	27,5676
Wavelets modificado	26,2289	15,7419	27,2937	25,6172	28,3802
Wavelets proposto	28,0465	20,0199	27,3696	27,6407	29,2491
Morfológico I	32,3706	23,8653	**29,8528**	32,1996	35,1106
Morfológico II	31,6266	30,3444	29,5675	31,3249	34,0093
Morfológico III	32,8965	**33,7602**	29,7816	32,9418	34,6502
Morfológico IV	32,7742	33,6442	29,6789	32,8559	35,3297

Tabela 5.4: Comparativo entre os melhores resultados de PSNR dos filtros estatísticos, dos filtros baseados em *wavelets* e dos filtros morfológicos (imagem *Peppers*).

Capítulo 6

Conclusão

Com base nos resultados, foi possível constatar que dentre todos os filtros implementados, os filtros morfológicos foram os que obtiveram os melhores resultados, considerando-se os resultados objetivos e subjetivos, sendo portanto os mais indicados a serem utilizados para se realizar a redução de ruído em imagens.

Visando organizar melhor os argumentos que suportem tal afirmação, podemos dividir a nossa análise comparativa em duas partes: objetiva e subjetiva.

Com relação à análise comparativa objetiva, que foi realizada em termos de PSNR, foi possível notar que os filtros estatísticos obtiveram um bom desempenho, que os filtros baseados em *wavelets* da literatura obtiveram um desempenho muito ruim, que o filtro wavelet proposto obteve um resultado um pouco ruim e que os filtros morfológicos propostos obtiveram um bom desempenho.

Podemos então sumarizar a análise comparativa objetiva dizendo que os filtros estatísticos e os filtros morfológicos foram os que obtiveram os melhores resultados objetivos.

Com relação à análise comparativa subjetiva, iremos nos valer dos seguintes aspectos para podermos avaliar os filtros implementados:

- Grau de redução de ruído em imagens;
- Grau de embaçamento das imagens filtradas;
- Quantidade de detalhes removidos da imagem original;
- Quantidade de artefatos inseridos nas imagens filtradas.

A respeito dos filtros estatísticos, foi possível concluir que em geral os mesmos tendem a apresentar um bom desempenho com relação à redução de ruído, tendendo porém a embaçar as imagens.

Com relação aos filtros baseados em *wavelets* da literatura que foram implementados, foi possível concluir que os mesmos tendem a reduzir pouco o ruído nas imagens, mantendo porém uma qualidade subjetiva razoável das imagens.

Já com relação ao filtro wavelet proposto, foi possível concluir que o mesmo consegue reduzir consideravelmente o ruído em imagens, eliminando porém uma quantidade considerável de detalhes e inserindo também uma quantidade de artefatos nas imagens filtradas proporcional a intensidade do ruído presente na imagem original.

Finalmente, com relação aos filtros morfológicos propostos, foi possível concluir que os mesmos tendem a reduzir a quantidade de ruído em imagens de forma bastante eficaz, mas possuem um inconveniente de remover alguns detalhes das imagens, mantendo contudo uma qualidade perceptual bastante interessante com relação à imagem filtrada.

Dessa forma, podemos então sumarizar a análise comparativa subjetiva dizendo que os filtros morfológicos foram os que obtiveram os melhores resultados subjetivos.

Assim, se combinarmos as análises objetiva e subjetiva realizadas, poderemos portanto constatar que dentre todos os filtros implementados, os filtros morfológicos foram de fato os que obtiveram os melhores resultados, sendo portanto os mais indicados a serem utilizados para se realizar a redução de ruído em imagens.

No mais, pode-se ainda concluir que o PSNR não demonstrou ser a medida ideal para se avaliar a qualidade de uma imagem, já que o mesmo não leva em consideração qualquer informação espacial de uma imagem. Isso indica que talvez fosse necessária a proposição de uma nova figura de mérito, que levasse em consideração a informação espacial de uma imagem, para ser utilizada como ferramenta de avaliação objetiva de qualidade de imagens.

Sem perda de importância, esta dissertação também pode ser considerada uma referência útil para aqueles que desejam compreender algumas das técnicas envolvidas na redução de ruído em imagens, sendo portanto recomendada para tal. Essa recomendação se deve principalmente a 3 contribuições significativas:

- A descrição de alguns dos tipos mais comuns de ruído que costumam afetar imagens (capítulo 3);
- A proposição de um novo método de redução de ruído em imagens baseado em *Wavelets* (capítulo 4);
- A proposição de um novo método de redução de ruído em imagens baseado em *Morfologia Matemática* (capítulo 5).

Por fim, pode-se dizer que esta dissertação contribuiu fortemente para o desenvolvimento acadêmico e profissional do autor, sendo a mesma de grande valia para seu crescimento.

Referências Bibliográficas

[1] GRAUPE, D., "Applications of signal and image processing to medicine". pp. 2125–2126, June 1988.

[2] YAMAMOTO, H., SUGITA, K., KANZAKI, N., et al., "Magnetic resonance image enhancement using V-filter". pp. 31–35, June 1990.

[3] SANTAGO, P., LINK, K. M., SNYDER, W. E., et al., "Restoration of cardiac magnetic resonance images". pp. 60–67, June 1990.

[4] GERIG, G., KÜBLER, O., KIKINIS, R., et al., "Nonlinear anisotropic filtering of MRI data", *IEEE Transactions on Medical Imaging*, v. 11, n. 2, pp. 221–232, 1992.

[5] SOLTANIAN-ZADEH, H., WINDHAM, J. P., PECK, D. J., et al., "A comparative analysis of several transformations for enhancement and segmentation of magnetic resonance image scene sequences", *IEEE Transactions on Medical Imaging*, v. 11, n. 3, pp. 302–318, 1992.

[6] SOLTANIAN-ZADEH, H., WINDHAM, J. P., YAGLE, A. E., "A multidimensional nonlinear edge-preserving filter for magnetic resonance image restoration", *IEEE Transactions on Image Processing*, v. 4, n. 2, pp. 147–161, 1995.

[7] TANG, K., ASTOLA, J., NEUVO, Y., "Nonlinear multivariate image filtering techniques", *IEEE Transactions on Image Processing*, v. 4, n. 6, pp. 788–798, 1995.

[8] SAPIRO, G., TANNENBAUM, A., YOU, Y.-L., et al., "Experiments on geometric image enhancement". pp. 472–476, November 1994.

[9] UNSER, M., ALDROUBI, A., "A review of wavelets in biomedical applications". pp. 626–638, June 1996.

[10] UNSER, M., "Wavelets, statistics, and biomedical applications". pp. 244–249, June 1996.

[11] TORKAMANI-AZAR, F., TAIT, K. E., "Image recovery using the anisotropic diffusion equation", *IEEE Transactions on Image Processing*, v. 5, n. 11, pp. 1573–1578, 1996.

[12] PALUBINSKAS, G., "Adaptive filtering in magnetic resonance images". pp. 523–527, August 1996.

[13] SOLTANIAN-ZADEH, H., WINDHAM, J. P., "Linear filter design for CNR enhancement of MR images with multiple interfering features". pp. 241–244, September 1996.

[14] TERAMOTO, A., HONBA, I., SUGIE, N., "Improvement of image quality in MR image using adaptive K-nearest neighbor averaging filter". pp. 190–194, September 1997.

[15] CRESPO, J., MAOJO, V., HERRERO, C., et al., "Enhancement of MR images using non-linear techniques". pp. 752–753, November 1996.

[16] WEICKERT, J., TER HAAR ROMENY, B. M., VIERGEVER, M. A., "Efficient and reliable schemes for nonlinear diffusion filtering", *IEEE Transactions on Image Processing*, v. 7, n. 3, pp. 398–410, 1998.

[17] GREGG, R. L., NOWAK, R. D., "Noise removal methods for high resolution MRI". pp. 1117–1121, November 1997.

[18] AHN, C. B., SONG, Y. C., PARK, D. J., "Adaptive template filtering for signal-to-noise ratio enhancement in magnetic resonance imaging", *IEEE Transactions on Medical Imaging*, v. 18, n. 6, pp. 549–556, 1999.

[19] NOWAK, R. D., "Wavelet-based Rician noise removal for magnetic resonance imaging", *IEEE Transactions on Image Processing*, v. 8, n. 10, pp. 1408–1419, 1999.

[20] JIN, J. S., WANG, Y., HILLER, J., "An adaptive nonlinear diffusion algorithm for filtering medical images", *IEEE Transactions on Information Technology in Biomedicine*, v. 4, n. 4, pp. 298–305, 2000.

[21] POSITANO, V., SANTARELLI, M., LANDINI, L., et al., "Nonlinear anisotropic filtering as a tool for SNR enhancement in cardiovascular MRI". pp. 707–710, September 2000.

[22] GRAFIA, M., ECHAVE, I., RUIZ-CABELLO, J., "VQ based Bayesian image filtering". pp. 451–454, September 2000.

[23] VIDAURRAZAGA, M., DIAGO, L., A.CRUZ, "Contrast Enhancement with Wavelet Transform in Radiololgical Images", .

[24] XUE, J.-H., PHILIPS, W., PIZURICA, A., et al., "A novel method for adaptive enhancement and unsupervised segmentation of MRI brain image". pp. 2013–2016, May 2001.

[25] SAHA, P. K., UDUPA, J. K., "Scale-based diffusive image filtering preserving boundary sharpness and fine structures", *IEEE Transactions on Medical Imaging*, v. 20, n. 11, pp. 1140–1155, 2001.

[26] LING, J., BOVIK, A. C., "Smoothing low-SNR molecular images via anisotropic median-diffusion", *IEEE Transactions on Medical Imaging*, v. 21, n. 4, pp. 377–384, 2002.

[27] RAOOF, K., ASFOUR, A., FOURNIER, J. M., "A complete digital magnetic resonance imaging (MRI) system at low magnetic field (0.1 Tesla)". pp. 341–345, May 2002.

[28] RODRIGUEZ, A. O., MANSFIELD, P., AZPIROZ, J., "Magnetic resonance image wavelet enhancer". pp. 2469–2471, October 2001.

[29] LUO, S., HAN, J., "Filtering medical image using adaptive filter". pp. 2727–2729, October 2001.

[30] ABD-ELMONIEM, K. Z., "Feedback coherent anisotropic diffusion for high resolution image enhancement". pp. 693–696, July 2002.

[31] SAMSONOV, A. A., JOHNSON, C. R., "Noise-adaptive anisotropic diffusion filtering of MRI images reconstructed by SENSE (sensitivity encoding) method". pp. 701–704, July 2002.

[32] BAYRAM, E., GE, Y., WYATT, C. L., "Confidence-based anisotropic filtering of magnetic resonance images", *IEEE Engineering in Medicine and Biology Magazine*, v. 21, n. 5, pp. 156–160, 2002.

[33] PIŽURICA, A., PHILIPS, W., LEMAHIEU, I., et al., "A versatile wavelet domain noise filtration technique for medical imaging", *IEEE Transactions on Medical Imaging*, v. 22, n. 3, pp. 323–331, 2003.

[34] BAO, P., ZHANG, L., "Noise reduction for magnetic resonance images via adaptive multiscale products thresholding", *IEEE Transactions on Medical Imaging*, v. 22, n. 9, pp. 1089–1099, 2003.

[35] LYSAKER, M., LUNDERVOLD, A., TAI, X.-C., "Noise removal using fourth-order partial differential equation with applications to medical magnetic resonance images in space and time", *IEEE Transactions on Image Processing*, v. 12, n. 12, pp. 1579–1590, 2003.

[36] GRUBER, P., THEIS, F., STADLTHANNER, K., et al., "Denoising using local ICA and kernel-PCA". pp. 2071–2076, July 2004.

[37] WINKELMANN, S., SCHAEFFTER, T., EGGERS, H., et al., "SNR enhancement in radial SSFP imaging using partial k-space averaging", *IEEE Transactions on Medical Imaging*, v. 24, n. 2, pp. 254–262, 2005.

[38] WONG, W. C., CHUNG, A. C., YU, S. C., "Trilateral filtering for biomedical images". pp. 820–823, April 2004.

[39] RAY, N., ACTON, S. T., "Inclusion filters: a class of self-dual connected operators", *IEEE Transactions on Image Processing*, v. 14, n. 11, pp. 1736–1746, 2005.

[40] LEUNG, E. T., TSOTSOS, J. K., "Adaptive enhancement of cardiac magnetic resonance (CMR) images". pp. 1739–1746, October 2005.

[41] CHENG, H., HUANG, F., "MRI Image Intensity Correction with Extrapolation and Smoothing". pp. 1324–1327, January 2006.

[42] HUANG, F., CHENG, H., VIJAYAKUMAR, S., "Gradient weighted smoothing for MRI intensity correction". pp. 3016–3019, January 2006.

[43] DERONG, Y., YUANYUAN, Z., DONGGUO, L., "Fast Computation of Multiscale Morphological Operations for Local Contrast Enhancement". pp. 3090–3092, January 2006.

[44] LI, W., MEUNIER, J., SOUCY, J.-P., "A 3D Adaptive Wiener Filter for Restoration of SPECT Images Using MRI as Reference Images". pp. 3100–3103, January 2006.

[45] CASTAÑO-MORAGA, C., LENGLET, C., DERICHE, R., et al., "A FAST AND RIGOROUS ANISOTROPIC SMOOTHING METHOD FOR DT-MRI", .

[46] GUO, L., LIU, X., WU, Y., et al., "Adaptive Template Filtering Method for MRI". pp. 1371–1374, August 2006.

[47] GUO, L., WU, Y., LIU, X., et al., "Threshold Optimization of Adaptive Template Filtering for MRI Based on Intelligent Optimization Algorithm". pp. 4763–4766, August 2006.

[48] GHAZEL, M., TRABOULSEE, A., WARD, R. K., "Optimal Filter Design for Multiple Sclerosis Lesions Segmentation from Regions of Interest in Brain MRI". pp. 1–5, August 2006.

[49] LEWIS, E. B., FOX, N. C., "Correction of differential intensity inhomogeneity in longitudinal MR images", .

[50] VONDRA, V., WAJER, F., HALÁMEK, J., et al., "Influence of digital audio filters on image reconstruction in MRI", .

[51] SMOLKA, B., LUKAC, R., CHYDZINSKI, A., et al., "Fast adaptive similarity based impulsive noise reduction filter", .

[52] SANCHEZ-ORTIZ, G. I., RUECKERT, D., BURGER, P., "Knowledge-based tensor anisotropic diffusion of cardiac magnetic resonance images", .

[53] GHITA, O., ROBINSON, K., LYNCH, M., et al., "MRI diffusion-based filtering: a note on performance characterisation", .

[54] STEPISNIK, J., DUH, A., MOHORIC, A., et al., "MRI Edge Enhancement as a Diffusive Discord of Spin Phase Structure", .

[55] "MRI: a new window into filter development, testing & optimization", .

[56] WHITAKER, R. T., "Geometry-Limited Diffusion in the Characterization of Geometric Patches in Images", .

[57] CARMI, E., LIUB, S., ALONA, N., et al., "Resolution enhancement in MRI", .

[58] CIOBANUA, L., WEBBA, A. G., PENNINGTON, C. H., "Signal enhancement by diffusion: experimental observation of the "DESIRE" effect", .

[59] RIO, D. E., RAWLINGS, R. R., KERICH, M. J., et al., "Statistical Methods in the Fourier Domain to Enhance and Classify Images", .

[60] WRANGSJÖ, A., KNUTSSON, H., "Histogram Filters for Noise Reduction", .

[61] HEMMENDORFF, M., "Motion Estimation and Compensation in Medical Imaging", .

[62] RYDELL, J., "Adaptive Spatial Filtering of fMRI Data", .

[63] PERONA, P., MALIK, J., "Scale-space and edge detection using anisotropic diffusion", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 12, n. 7, pp. 629–639, 1990.

[64] JUNG, S.-H., KIM, N.-C., "Adaptive image restoration of sigma filter using local statistics and human visual characteristics", *Electronics Letters*, v. 24, n. 4, pp. 201–202, 1988.

[65] SHENG, Y., XIA, Z.-G., "A comprehensive evaluation of filters for radar speckle suppression". pp. 1559–1561, May 1996.

[66] ALPARONE, L., BARONTI, S., GARZELLI, A., "A hybrid sigma filter for unbiased and edge-preserving speckle reduction". pp. 1409–1411, July 1995.

[67] LAN, T.-H., TEWJK, A. H., KUO, C.-H., "Sigma filtered perceptual image coding at low bit rates". pp. 371–375, October 1999.

[68] LUKUC, R., SMOLKA, B., PLUTUNIOTIS, K. N., et al., "Angular multi-channel sigma filter". pp. 745–8, April 2003.

[69] LUKAC, R., SMOLKA, B., PLATANIOTIS, K., et al., "Generalized adaptive vector sigma filters". pp. 537–40, July 2003.

[70] HUANG, Y., HUI, L., "An adaptive spatial filter for additive Gaussian and impulse noise reduction in video signals". pp. 523–526, December 2003.

[71] GHAZAL, M., AMER, A., GHRAYEB, A., "Homogeneity-based directional sigma filtering of video noise". pp. 97–100, September 2005.

[72] MELNIK, V., EGIAZARIAN, K., SHMULEVICH, I., et al., "A tree of median pyramidal decompositions with an application to signal denoising". pp. 117–120, June 2000.

[73] MELNIK, V. P., SHMULEVICH, I., EGIAZARIAN, K., et al., "Block-median pyramidal transform: analysis and denoising applications", *IEEE Transactions on Signal Processing*, v. 49, n. 2, pp. 364–372, 2001.

[74] EGIAZARIAN, K., KATKOVNIK, V., ASTOLA, J., "Adaptive window size image denoising based on ICI rule". pp. 1869–1872, May 2001.

[75] KATKOVNIK, V., "A multiresolution nonparametric regression for spatially adaptive image de-noising", *IEEE Signal Processing Letters*, v. 11, n. 10, pp. 798–801, 2004.

[76] ERCOLE, C., FOI, A., KATKOVNIK, V., et al., "SPATIO-TEMPORAL POINTWISE ADAPTIVE DENOISING OF VIDEO 3D NON-PARAMETRIC REGRESSION APPROACH", .

[77] A, R. F., B, S. A., B, M., et al., "ADAPTIVE-SIZE BLOCK TRANSFORMS FOR POISSONIAN IMAGE DEBLURRING", .

[78] FOI, A., DABOV, K., KATKOVNIK, V., et al., "Shape-Adaptive DCT for Denoising and Image Reconstruction", .

[79] KATKOVNIK, V., "Multiresolution local polynomial regression: A new approach to pointwise spatial adaptation", .

[80] KATKOVNIK, V., PALIY, D., EGIAZARIAN, K., et al., "FREQUENCY DOMAIN BLIND DECONVOLUTION IN MULTIFRAME IMAGING USING ANISOTROPIC SPATIALLY-ADAPTIVE DENOISING", .

[81] MINASYAN, S., ASTOLA, J., EGIAZARIAN, K., et al., "Parametric Haar-Like Transforms in Image Denoising". pp. 2629–2632, October 2006.

[82] KATKOVNIK, V., EGIAZARIAN, K., SHMULEVICH, I., "ADAPTIVE VARYING WINDOW SIZE SELECTION BASED ON INTERSECTION OF CONFIDENCE INTERVALS RULE", .

[83] JIANG, L., YANG, W., "Adaptive Magnetic Resonance Image Denoising Using Mixture Model and Wavelet Shrinkage". In: *Proc. VIIth Digital Image Computing: Techniques and Applications*, pp. 831–838, 2003.

[84] MALLAT, S., ZHONG, S., "Characterization of signals from multiscale edges", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 14, n. 7, pp. 710–732, 1992.

[85] XU, Y., WEAVER, J. B., HEALY, D. M., et al., "Wavelet transform domain filters: a spatially selective noise filtration technique", *IEEE Transactions on Image Processing*, v. 3, n. 6, pp. 747–758, 1994.

[86] SADLER, B. M., SWAMI, A., "Analysis of multiscale products for step detection and estimation", *IEEE Transactions on Information Theory*, v. 45, n. 3, pp. 1043–1051, 1999.

[87] PAN, Q., ZHANG, L., DAI, G., et al., "Two denoising methods by wavelet transform", *IEEE Transactions on Signal Processing*, v. 47, n. 12, pp. 3401–3406, 1999.

[88] CHANG, S. G., YU, B., VETTERLI, M., "Adaptive wavelet thresholding for image denoising and compression", *IEEE Transactions on Image Processing*, v. 9, n. 9, pp. 1532–1546, 2000.

[89] DONOHO, D. L., JOHNSTONE, I. M., "Adapting to unknown smoothness via wavelet shrinkage", *Journal of the American Statistical Association*, v. 90, n. 1, pp. 1200–1224, 1995.

[90] DONOHO, D. L., JOHNSTONE, I. M., "Ideal spatial adaptation by wavelet shrinkage", *Biometrika*, v. 81, n. 1, pp. 425–455, 1994.

[91] COIFMAN, R. R., DONOHO, D. L., "Translation-invariant de-noising". In: *Wavelets and Statistics, Springer Lecture Notes in Statistics 103*, pp. 125–150, 1995.

[92] SHI, F., SELESNICK, I. W., "An elliptically contoured exponential mixture model for wavelet based image denoising", *Applied and Computational Harmonic Analysis*, v. 23, n. 1, pp. 131–151, 2007.

[93] OLHEDE, S. C., "Hyperanalytic Denoising", *IEEE Transactions on Image Processing*, v. 16, n. 6, pp. 1522–1537, 2007.

[94] KIM, B.-G., PARK, D.-J., "Novel Noncontrast-Based Edge Descriptor for Image Segmentation", *IEEE Transactions on Circuits and Systems for Video Technology*, v. 16, n. 9, pp. 1086–1095, 2006.

[95] JAIN, A. K., *Fundamentals of Digital Image Processing*. 1st ed. Prentice-Hall Inc.: Upper Saddle River, New Jersey 07458, U.S.A., 1989.

[96] BLIZARD, W. D., "Multiset theory", *Notre Dame J. Formal Logic*, v. 30, n. 1, pp. 36–66, 1988.

[97] BOVIK, A. C., *Handbook of Image & Video Processing*. 2nd ed. Academic Press: 30 Corporate Drive, Suite 400, Burlington, MA 01803, USA, 2005.

[98] PEYTON Z. PEEBLES, J., *Probability, random variables, and random signal principles*. 4th ed. McGraw-Hill: 1221 Avenue of the Americas, New York, NY 10200, 2001.

[99] LEE, J. S., "Digital image smoothing and the sigma filter", *Comput. Graphics Image Processing*, v. 24, pp. 255–269, 1983.

[100] LEE, J. S., "A Simple Speckle Smoothing Algorithm for Synthetic Aperture Radar Images", *IEEE Trans. Systems Man Cybern.*, v. 13, n. 1, pp. 85–89, 1983.

[101] BLAKE, R., SEKULER, R., *Perception*. 5th ed. McGraw-Hill: 1221 Avenue of the Americas, New York, NY 10200, 2006.

[102] CANNY, J., "A Computational Approach to Edge Detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. PAMI-8, n. 6, pp. 679–698, 1986.

[103] SOILLE, P., *Morphological image analysis: principles and applications*. 1st ed. Springer: Berlin; Heidelberg; New York; Barcelona; Hongkong; London; Milan; Paris; Singapore; Tokyo, 1999.

Apêndice A

Provas

Neste apêndice encontram-se as provas matemáticas relativas a diversas equações que foram utilizadas nesta dissertação. Mais precisamente, as seguintes demostrações serão aqui expostas:

- Função densidade de probabilidade do ruído do tipo "sal e pimenta" (equação 3.17);
- Função densidade de probabilidade do ruído relativo à contagem de fótons (equação 3.19);
- Função densidade de probabilidade do ruído relativo à granulação em fotografias (equação 3.20);
- Equivalência entre sigmas aditivo e multiplicativo (equações 3.21 e 3.26);
- Reconstrução perfeita do filtro morfológico (equação 5.11);
- Reconstrução perfeita do filtro morfológico II (equação 5.14);
- Reconstrução perfeita do filtro morfológico III (equação 5.19);
- Reconstrução perfeita do filtro morfológico IV (equação 5.22).

A.1 Ruído do tipo "sal e pimenta"

Demonstração. De acordo com [97]

$$Prob(X = I) = 1 - p$$
$$Prob(X = 0) = \frac{p}{2}$$
$$Prob(X = 255) = \frac{p}{2}$$

Mas como $X = I + S\&P$, e como a imagem satura em 0 e em 255 podemos dizer que:

$$f_{S\&P}(r) \triangleq (1-\epsilon)\delta(r) + \lim_{u\to\infty}\left[\frac{\epsilon}{2}\delta(r-u) + \frac{\epsilon}{2}\delta(r+u)\right] \qquad \text{C.Q.D.}$$

A.2 Ruido relativo a contagem de fótons

Demonstração. De acordo com [97]:

$$f_X(r) \triangleq e^{-\lambda}\sum_{k=0}^{\infty}\frac{\lambda^k}{k!}\delta(r-k)$$

Mas $X = I + F$. Assim, considerando $I = \overline{X}$, então $F = X - \overline{X}$. Além disso, $\overline{X} = \lambda = \sigma_F^2$. Daí:

$$\begin{aligned} f_F(r) &= f_X(r+\lambda) \\ &= e^{-\lambda}\sum_{k=0}^{\infty}\frac{\lambda^k}{k!}\delta(r+\lambda-k) \\ &= e^{-\lambda}\sum_{k=-\lambda}^{\infty}\frac{\lambda^{(k+\lambda)}}{(k+\lambda)!}\delta(r-k) \\ &= e^{-\sigma_F^2}\sum_{k=-\sigma_F^2}^{\infty}\frac{\sigma_F^{2(k+\sigma_F^2)}}{(k+\sigma_F^2)!}\delta(r-k) \qquad \text{C.Q.D.} \end{aligned}$$

A.3 Ruido relativo à granulação em fotografias

Demonstração. De acordo com [97]:

$$f_X(r) \triangleq \sum_{k=0}^{L}\binom{L}{k}p^k(1-p)^{L-k}\delta(r-k)$$

Mas $X = I + G$. Assim, considerando $I = \overline{X}$, então $G = X - \overline{X}$. Além disso, $\overline{X} = Lp$. Daí:

$$
\begin{aligned}
f_G(r) &= f_X(r + Lp) \\
&= \sum_{k=0}^{L} \binom{L}{k} p^k (1-p)^{L-k} \delta(r + Lp - k) \\
&= \sum_{k=-Lp}^{L-Lp} \binom{L}{k+Lp} p^{k+Lp} (1-p)^{L-Lp-k} \delta(r-k) \\
\therefore f_G(r) &= \sum_{k=-\lfloor Lp \rfloor}^{L-\lfloor Lp \rfloor} \binom{L}{k+\lfloor Lp \rfloor} p^{k+\lfloor Lp \rfloor} (1-p)^{L-\lfloor Lp \rfloor - k} \delta(r-k) \qquad \text{C.Q.D.}
\end{aligned}
$$

A.4 Equivalência entre sigma aditivo e multiplicativo

Demonstração. Sigma aditivo:

$$
a[n] = \begin{cases} 1 & \text{, se } |x[n] - X(d_1, d_2)| \leq \Lambda_A \\ 0 & \text{, caso contrário} \end{cases}
$$

Onde $\Lambda_A = \propto \sigma_x$.

Sigma multiplicativo:

$$
a[n] = \begin{cases} 1 & \text{, se } \dfrac{|x[n] - X(d_1, d_2)|}{X(d_1, d_2)} \leq \Lambda_M \\ 0 & \text{, caso contrário} \end{cases}
$$

Onde $\Lambda_M = \propto \frac{\sigma_x}{\bar{x}}$.

A equação anterior pode ser rescrita como:

$$
a[n] = \begin{cases} 1 & \text{, se } |x[n] - X(d_1, d_2)| \leq \Lambda_M X(d_1, d_2) \\ 0 & \text{, caso contrário} \end{cases}
$$

Mas:

$$
\begin{aligned}
\overline{\Lambda_M X(d_1,d_2)} &= \int_{-\infty}^{\infty} \Lambda_M X(d_1,d_2) f_X(r)\,\mathrm{d}r \\
&= \Lambda_M \int_{-\infty}^{\infty} X(d_1,d_2) f_X(r)\,\mathrm{d}r \\
&=\propto \frac{\sigma_x}{\overline{x}} \int_{-\infty}^{\infty} X(d_1,d_2) f_X(r)\,\mathrm{d}r \\
&=\propto \frac{\sigma_x}{\overline{x}} \overline{x} \\
&=\propto \sigma_x \\
&= \Lambda_A \\
\therefore \overline{\Lambda_M X(d_1,d_2)} &= \overline{\Lambda_A}
\end{aligned}
$$

C.Q.D.

A.5 Reconstrução perfeita do filtro morfológico

Demonstração. As equações 5.3 e 5.4 nos dizem que:

$$\hat{X}_E = X - \gamma_E(X) \tag{A.1}$$

E que:

$$\check{X}_E = \phi_E(X) - X \tag{A.2}$$

Daí, se fizermos A.1 − A.2, teremos:

$$
\begin{aligned}
\hat{X}_E - \check{X}_E &= 2X - \gamma_E(X) - \phi_E(X) \\
\therefore X &= \frac{\gamma_E(X) + \phi_E(X) + \hat{X}_E - \check{X}_E}{2}
\end{aligned}
$$

Assim, se a após a filtragem de $\hat{X}_E$ e de $\check{X}_E$, $\hat{F}_E$ for igual a $\hat{X}_E$ e $\check{F}_E$ for igual a $\check{X}_E$, então para haver uma reconstrução perfeita de X, Y deverá ser igual a X, o que siginifica dizer que:

$$Y = \frac{\gamma_E(X) + \phi_E(X) + \hat{X}_E - \check{X}_E}{2}$$

C.Q.D.

A.6 Reconstrução perfeita do filtro morfológico II

Demonstração. As equações 5.12 e 5.13 nos dizem que:

$$\hat{X}_E = \begin{cases} X - \phi_E[\gamma_E(X)] & \text{, se } X - \phi_E[\gamma_E(X)] > 0 \\ 0 & \text{, caso contrário} \end{cases}$$

E que:

$$\check{X}_E = \begin{cases} \gamma_E[\phi_E(X)] - X & \text{, se } \gamma_E[\phi_E(X)] - X > 0 \\ 0 & \text{, caso contrário} \end{cases}$$

Podemos então reescrevê-las da seguinte maneira respectivamente:

$$\hat{X}_E = \frac{|X - \phi_E[\gamma_E(X)]| + X - \phi_E[\gamma_E(X)]}{2} \tag{A.3}$$

$$\hat{X}_E = \frac{|\gamma_E[\phi_E(X)] - X| + \gamma_E[\phi_E(X)] - X}{2} \tag{A.4}$$

Daí, se fizermos A.3 − A.4, teremos:

$$\begin{aligned} \hat{X}_E - \check{X}_E &= \frac{2X - \phi_E[\gamma_E(X)] - \gamma_E[\phi_E(X)] + |X - \phi_E[\gamma_E(X)]| - |\gamma_E[\phi_E(X)] - X|}{2} \\ \therefore X &= \frac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)] - |X - \phi_E[\gamma_E(X)]| + |\gamma_E[\phi_E(X)] - X|}{2} \\ &\quad + \hat{X}_E - \check{X}_E \end{aligned}$$

Assim, se a após a filtragem de $\hat{X}_E$ e de $\check{X}_E$, $\hat{F}_E$ for igual a $\hat{X}_E$ e $\check{F}_E$ for igual a $\check{X}_E$, então para haver uma reconstrução perfeita de X, Y deverá ser igual a X, o que siginifica dizer que:

$$Y = \hat{F}_E - \check{F}_E + \frac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)] - |X - \phi_E[\gamma_E(X)]| + |\gamma_E[\phi_E(X)] - X|}{2}$$

C.Q.D.

A.7 Reconstrução perfeita do filtro morfológico III

Demonstração. As equações 5.15 e 5.16 nos dizem que:

$$\hat{X}_E = X - \phi_E[\gamma_E(X)] \tag{A.5}$$

$$\check{X}_E = \gamma_E[\phi_E(X)] - X \tag{A.6}$$

Daí, se fizermos A.5 − A.6, teremos:

$$\hat{X}_E - \check{X}_E = 2X - \phi_E[\gamma_E(X)] - \gamma_E[\phi_E(X)]$$
$$\therefore X = \frac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)] + \hat{X}_E - \check{X}_E}{2}$$

Assim, se a após a filtragem de $\hat{X}_E$ e de $\check{X}_E$, $\hat{F}_E$ for igual a $\hat{X}_E$ e $\check{F}_E$ for igual a $\check{X}_E$, então para haver uma reconstrução perfeita de X, Y deverá ser igual a X, o que siginifica dizer que:

$$X = \frac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)] + \hat{F}_E - \check{F}_E}{2} \qquad \text{C.Q.D.}$$

A.8 Reconstrução perfeita do filtro morfológico IV

Demonstração. As equações 5.20 e 5.21 nos dizem que:

$$\hat{X}_E = \begin{cases} X - \dfrac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)]}{2} & \text{, se } X - \dfrac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)]}{2} > 0 \\ 0 & \text{, caso contrário} \end{cases}$$

E que:

$$\check{X}_E = \begin{cases} -\left(X - \dfrac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)]}{2}\right) & \text{, se } X - \dfrac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)]}{2} < 0 \\ 0 & \text{, caso contrário} \end{cases}$$

Podemos então reescrevê-las da seguinte maneira respectivamente:

$$\hat{X}_E = \frac{\left|X - \dfrac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)]}{2}\right| + X - \dfrac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)]}{2}}{2} \tag{A.7}$$

$$\check{X}_E = \frac{\left|X - \dfrac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)]}{2}\right| - \left(X - \dfrac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)]}{2}\right)}{2} \tag{A.8}$$

Daí, se fizermos A.7 − A.8, teremos:

$$\hat{X}_E - \check{X}_E = \frac{2\left(X - \frac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)]}{2}\right)}{2} =$$
$$= X - \frac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)]}{2}$$
$$\therefore X = \frac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)]}{2} + \hat{X}_E - \check{X}_E$$

Assim, se a após a filtragem de $\hat{X}_E$ e de $\check{X}_E$, $\hat{F}_E$ for igual a $\hat{X}_E$ e $\check{F}_E$ for igual a $\check{X}_E$, então para haver uma reconstrução perfeita de X, Y deverá ser igual a X, o que siginifica dizer que:

$$Y = \frac{\phi_E[\gamma_E(X)] + \gamma_E[\phi_E(X)]}{2} + \hat{F}_E - \check{F}_E \qquad \text{C.Q.D.}$$

Apêndice B

Resultados

Neste apêndice, serão exibidos resultados extensos de todos os métodos implementados com relação a cada uma das imagens utilizadas na presente obra, de forma a proporcionar uma maior compreensão sobre as diversas possibilidades de configuração de cada método.

No caso das imagens corrompidas por ruído simulado, as figuras contendo os resultados das mesmas são geradas utilizando-se as configurações representadas em negrito nas referentes tabelas.

B.1 Filtro mediana

B.1.1 Imagem Moedas

(a) Imagem original

(b) Imagem filtrada (janela 3×3)

(c) Imagem filtrada (janela 5×5)

(d) Imagem filtrada (janela 7×7)

Figura B.1: Resultados da filtragem da imagem *Moedas* pelo filtro mediana.

B.1.2 Imagem Máscaras

(a) Imagem original

(b) Imagem filtrada (janela 3×3)

(c) Imagem filtrada (janela 5×5)

(d) Imagem filtrada (janela 7×7)

Figura B.2: Resultados da filtragem da imagem *Máscaras* pelo filtro mediana.

B.1.3 Imagem Lena

	T	$S\&P$	Q	F	G
Janela 3×3	**32,4525**	**32,8072**	**28,9034**	**31,9231**	**33,8332**
Janela 5×5	30,5604	30,5402	27,5451	30,3621	30,9899
Janela 7×7	28,7353	28,6939	26,4319	28,6288	28,9641

Tabela B.1: Resultados de PSNR do filtro mediana (imagem *Lena*).

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.3: Imagem *Lena* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro mediana.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.4: Imagem *Lena* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro mediana.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.5: Imagem *Lena* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro mediana.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.6: Imagem *Lena* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro mediana.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

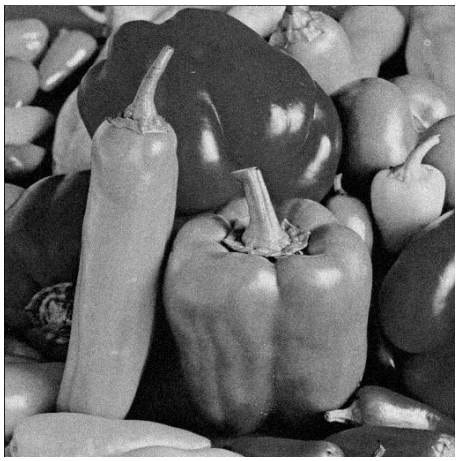
Figura B.7: Imagem *Lena* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro mediana.

B.1.4 Imagem Peppers

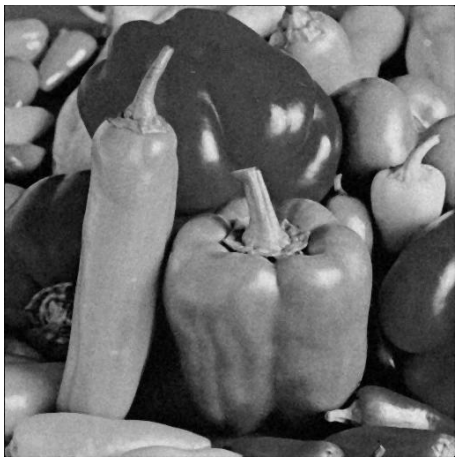
	T	$S\&P$	Q	F	G
Janela 3×3	**32,7621**	**33,1648**	**29,1780**	**32,6508**	**34,4521**
Janela 5×5	31,6331	31,6815	28,2171	31,6157	32,2872
Janela 7×7	29,9883	29,9291	27,2763	29,9871	30,3617

Tabela B.2: Resultados de PSNR do filtro mediana (imagem *Peppers*).

(a) Imagem original



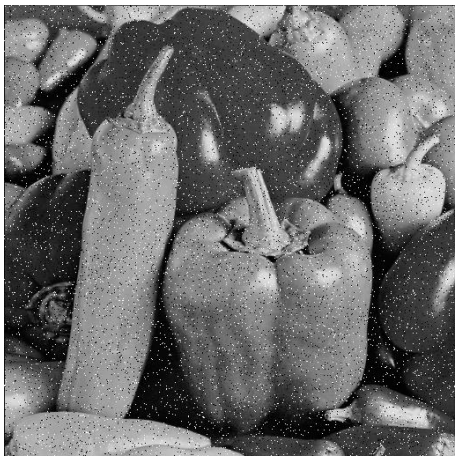
(b) Imagem corrompida



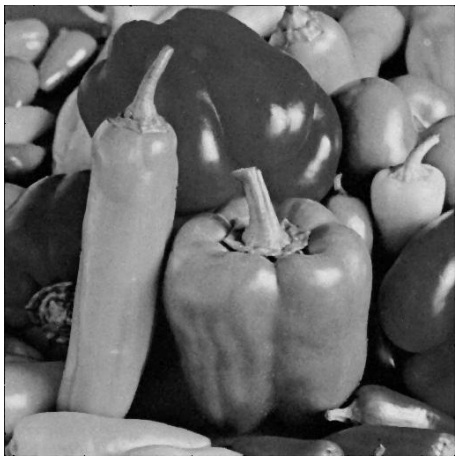
(c) Imagem filtrada

Figura B.8: Imagem *Peppers* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro mediana.

(a) Imagem original



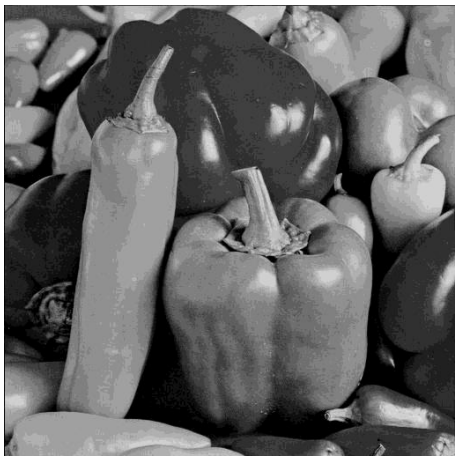
(b) Imagem corrompida



(c) Imagem filtrada

Figura B.9: Imagem *Peppers* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro mediana.

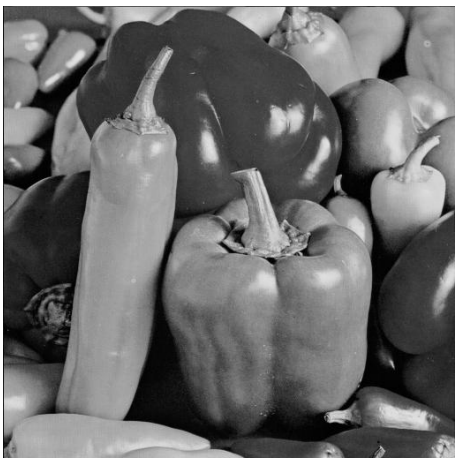
(a) Imagem original



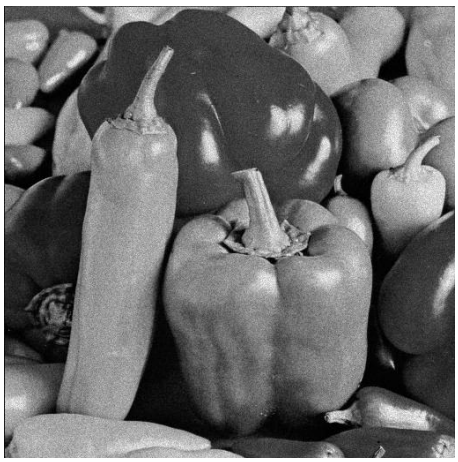
(b) Imagem corrompida

(c) Imagem filtrada

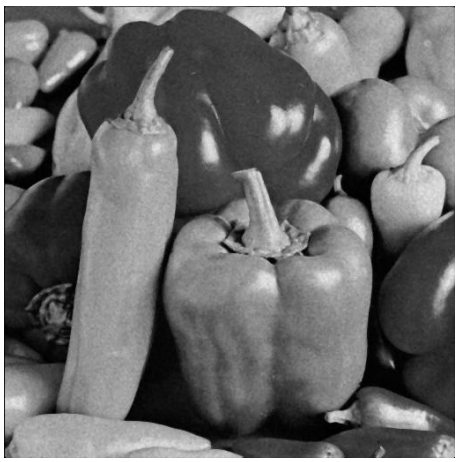
Figura B.10: Imagem *Peppers* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro mediana.



(a) Imagem original

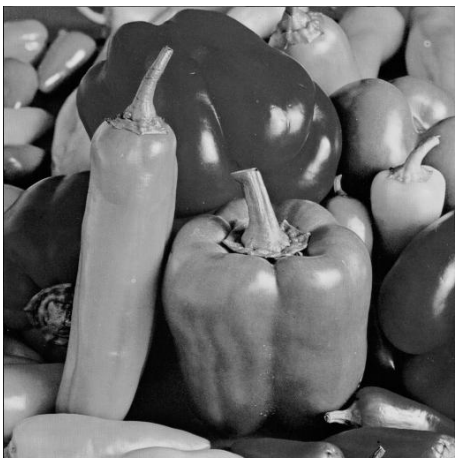


(b) Imagem corrompida

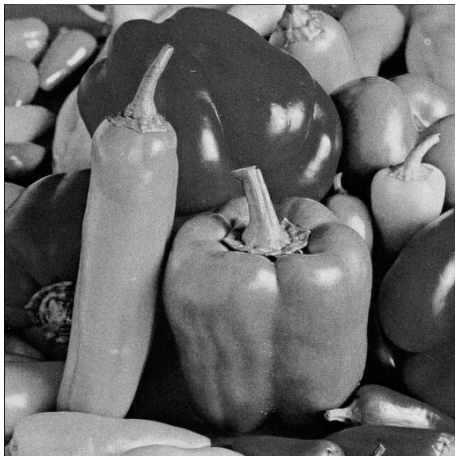


(c) Imagem filtrada

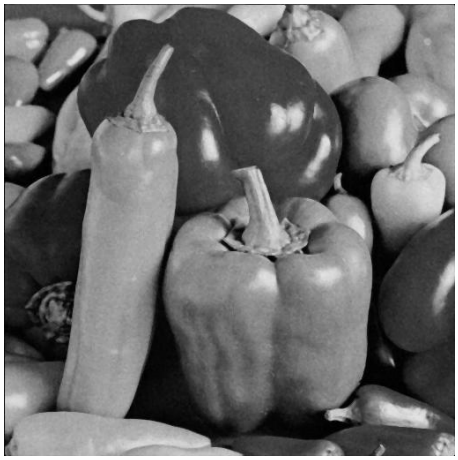
Figura B.11: Imagem *Peppers* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro mediana.



(a) Imagem original



(b) Imagem corrompida



(c) Imagem filtrada

Figura B.12: Imagem *Peppers* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro mediana.

B.2 Filtro média

B.2.1 Imagem Moedas

(a) Imagem original

(b) Imagem filtrada (janela 3×3)

(c) Imagem filtrada (janela 5×5)

(d) Imagem filtrada (janela 7×7)

Figura B.13: Resultados da filtragem da imagem *Moedas* pelo filtro média.

B.2.2 Imagem Máscaras

(a) Imagem original

(b) Imagem filtrada (janela 3×3)

(c) Imagem filtrada (janela 5×5)

(d) Imagem filtrada (janela 7×7)

Figura B.14: Resultados da filtragem da imagem *Máscaras* pelo filtro média.

B.2.3 Imagem Lena

	T	$S\&P$	Q	F	G
Janela 3×3	**31,1684**	26,4986	**27,9647**	**30,9393**	**31,6481**
Janela 5×5	28,2326	**26,6758**	26,1143	28,1847	28,3081
Janela 7×7	26,3373	25,5928	24,7930	26,3224	26,3618

Tabela B.3: Resultados de PSNR do filtro média (imagem *Lena*).

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.15: Imagem *Lena* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro média.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.16: Imagem *Lena* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro média.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.17: Imagem *Lena* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro média.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.18: Imagem *Lena* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro média.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

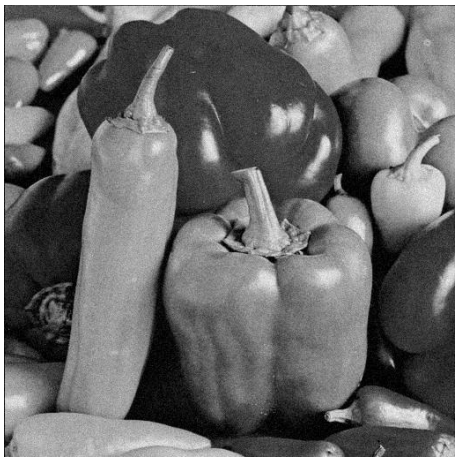
Figura B.19: Imagem *Lena* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro média.

B.2.4 Imagem Peppers

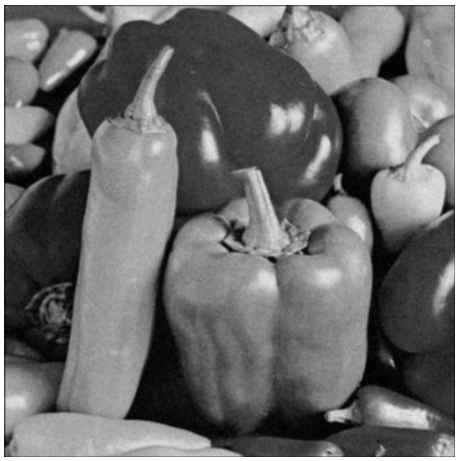
	T	$S\&P$	Q	F	G
Janela 3×3	**31,1745**	26,1795	**28,1018**	**31,1763**	**31,7661**
Janela 5×5	28,4274	**26,5962**	26,3292	28,4763	28,5795
Janela 7×7	26,5072	25,5693	24,9815	26,5524	26,5827

Tabela B.4: Resultados de PSNR do filtro média (imagem *Peppers*).

(a) Imagem original



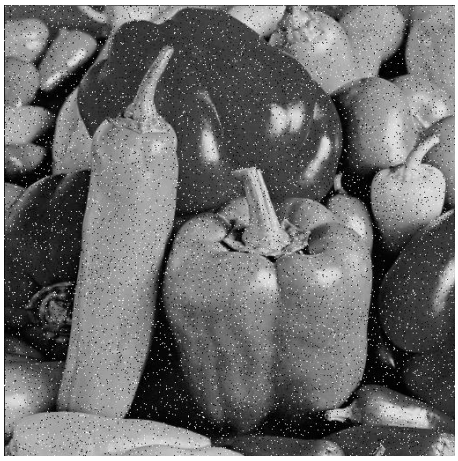
(b) Imagem corrompida



(c) Imagem filtrada

Figura B.20: Imagem *Peppers* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro média.

(a) Imagem original

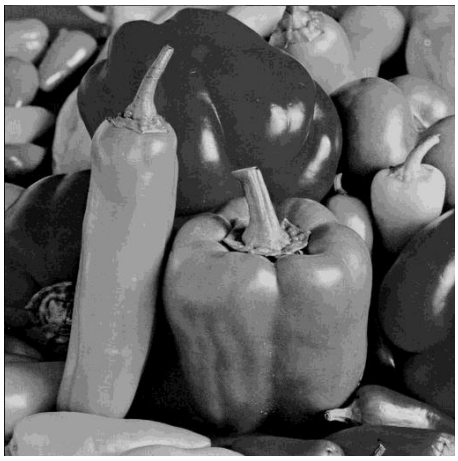


(b) Imagem corrompida

(c) Imagem filtrada

Figura B.21: Imagem *Peppers* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro média.

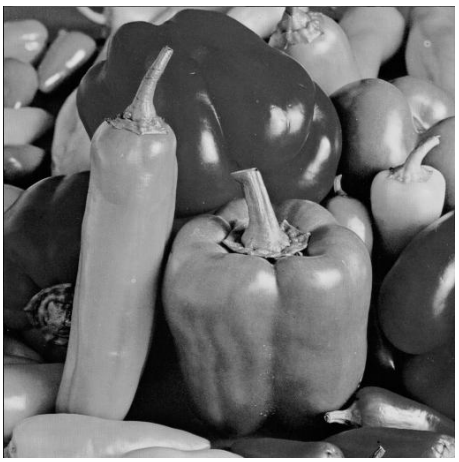
(a) Imagem original



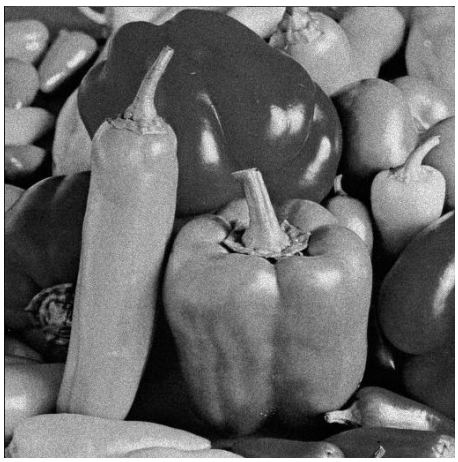
(b) Imagem corrompida

(c) Imagem filtrada

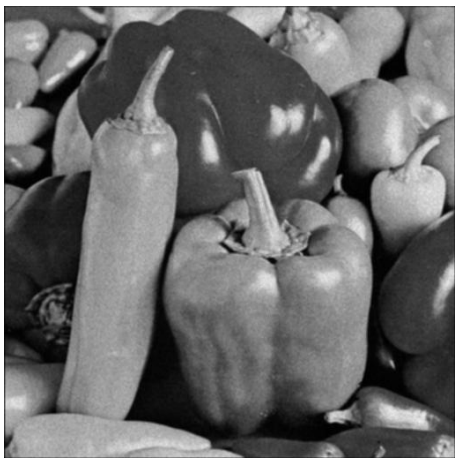
Figura B.22: Imagem *Peppers* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro média.



(a) Imagem original

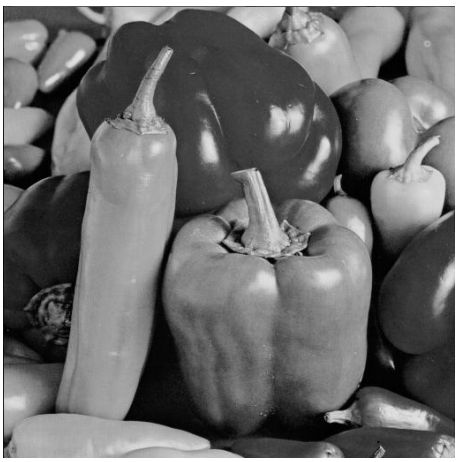


(b) Imagem corrompida

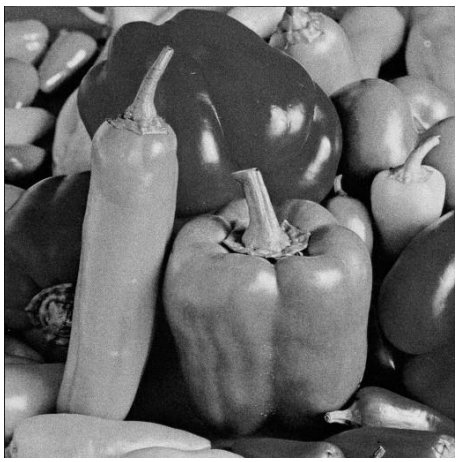


(c) Imagem filtrada

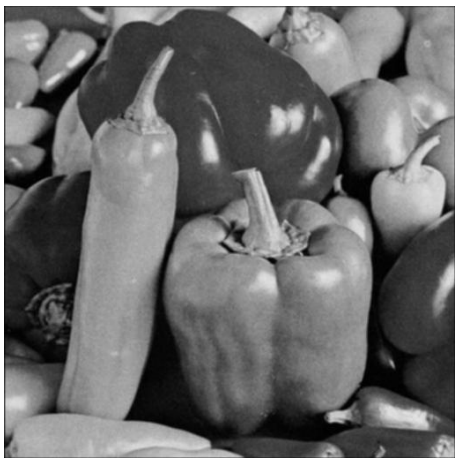
Figura B.23: Imagem *Peppers* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro média.



(a) Imagem original



(b) Imagem corrompida



(c) Imagem filtrada

Figura B.24: Imagem *Peppers* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro média.

B.3 Filtro sigma aditivo

B.3.1 Imagem Moedas

(a) Imagem original

(b) Imagem filtrada (janela 3×3, $\Lambda = 0,5\sigma_x$, $K = 3$)

(c) Imagem filtrada (janela 3×3, $\Lambda = 1,5\sigma_x$, $K = 3$)

(d) Imagem filtrada (janela 5×5, $\Lambda = 0,5\sigma_x$, $K = 5$)

(e) Imagem filtrada (janela 5×5, $\Lambda = 1,5\sigma_x$, $K = 5$)

Figura B.25: Resultados da filtragem da imagem *Moedas* pelo filtro sigma aditivo.

B.3.2 Imagem Máscaras

(a) Imagem original

(b) Imagem filtrada (janela 3×3, $\Lambda = 0,5\sigma_x$, $K = 3$)

(c) Imagem filtrada (janela 3×3, $\Lambda = 1,5\sigma_x$, $K = 3$)

(d) Imagem filtrada (janela 5×5, $\Lambda = 0,5\sigma_x$, $K = 5$)

(e) Imagem filtrada (janela 5×5, $\Lambda = 1,5\sigma_x$, $K = 5$)

Figura B.26: Resultados da filtragem da imagem *Máscaras* pelo filtro sigma aditivo.

B.3.3 Imagem Lena

Janela	Λ	K	T	$S\&P$	Q	F	G
3×3	$0,5\sigma_x$	1	32,6804	25,3407	**29,5406**	30,7615	35,5562
		3	32,6729	30,5695	29,4298	31,2454	35,2288
		5	32,4272	28,2965	29,1911	31,4030	34,4171
	σ_x	1	**33,6237**	25,5186	29,1538	32,9668	34,6245
		3	33,6036	**32,5366**	29,1469	**32,9737**	34,6144
		5	33,4912	31,9436	29,1051	32,8738	34,5042
	$1,5\sigma_x$	1	33,1365	25,4016	28,9176	32,7439	33,9199
		3	33,1365	31,9749	28,9175	32,7441	33,9200
		5	33,1152	32,1252	28,9117	32,7232	33,9053
	$2\sigma_x$	1	32,7366	25,2776	28,7455	32,3799	33,4272
		3	32,7366	30,9945	28,7455	32,3799	33,4272
		5	32,7304	31,4354	28,7431	32,3730	33,4197
5×5	$0,5\sigma_x$	1	33,3036	21,8934	29,4425	31,3286	**35,5917**
		4	33,2498	31,1138	29,3960	31,4899	35,4428
		10	32,9154	30,1401	29,1223	31,6684	34,6340
	σ_x	1	32,9435	21,8805	28,6069	32,6436	33,2154
		4	32,9343	31,0115	28,6046	32,6393	33,2127
		10	32,8452	31,5530	28,5619	32,5588	33,1194
	$1,5\sigma_x$	1	31,5111	21,8702	27,9564	31,3969	31,6089
		4	31,5111	29,8419	27,9564	31,3969	31,6082
		10	31,4984	30,6889	27,9509	31,3825	31,5974
	$2\sigma_x$	1	30,5047	22,1195	27,4791	30,4276	30,5955
		4	30,5047	28,8254	27,4791	30,4276	30,5955
		10	30,4999	29,7874	27,4774	30,4235	30,5917

Tabela B.5: Resultados de PSNR do filtro sigma aditivo (imagem *Lena*).

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.27: Imagem *Lena* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro sigma aditivo.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.28: Imagem *Lena* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro sigma aditivo.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.29: Imagem *Lena* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro sigma aditivo.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.30: Imagem *Lena* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro sigma aditivo.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

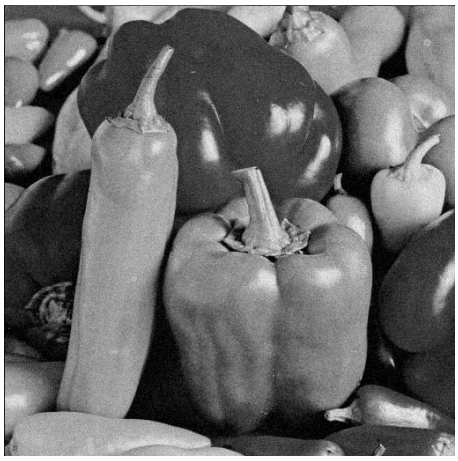
Figura B.31: Imagem *Lena* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro sigma aditivo.

B.3.4 Imagem Peppers

Janela	Λ	K	T	$S\&P$	Q	F	G
3×3	$0,5\sigma_x$	1	33,5161	25,2573	**29,6507**	32,3579	**35,8916**
		3	33,3702	31,5501	29,5494	32,4881	35,5269
		5	33,0296	29,6022	29,3633	32,3553	34,8775
	σ_x	1	33,6545	25,3006	29,2873	**33,5984**	34,8463
		3	33,6083	**32,3282**	29,2788	33,5532	34,7945
		5	33,5123	32,0028	29,2284	33,4202	34,6393
	$1,5\sigma_x$	1	33,0105	25,1817	28,9872	33,0390	33,9573
		3	33,0100	31,3653	28,9872	33,0369	33,9572
		5	32,9766	31,6706	28,9771	33,0083	33,9283
	$2\sigma_x$	1	32,5328	25,1353	28,7687	32,5481	33,3324
		3	32,5328	30,2522	28,7687	32,5481	33,3324
		5	32,5225	30,7606	28,7649	32,5372	33,3226
5×5	$0,5\sigma_x$	1	**34,0606**	21,6285	29,5004	32,9427	35,5979
		4	33,9610	31,3230	29,4617	32,9459	35,4537
		10	33,5955	31,2085	29,2794	32,8174	34,8527
	σ_x	1	33,0159	21,6470	28,7642	33,1158	33,4590
		4	33,0076	30,7498	28,7618	33,1065	33,4522
		10	32,9138	31,6340	28,7289	33,0209	33,3437
	$1,5\sigma_x$	1	31,6945	21,7613	28,1602	31,8455	32,0008
		4	31,6945	29,6139	28,1602	31,8455	32,0008
		10	31,6749	30,6927	28,1530	31,8311	31,9831
	$2\sigma_x$	1	30,6588	22,0640	27,6199	30,7704	30,8810
		4	30,6588	28,7580	27,6199	30,7704	30,8810
		10	30,6535	29,7031	27,6169	30,7633	30,8766

Tabela B.6: Resultados de PSNR do filtro sigma aditivo (imagem *Peppers*).

(a) Imagem original

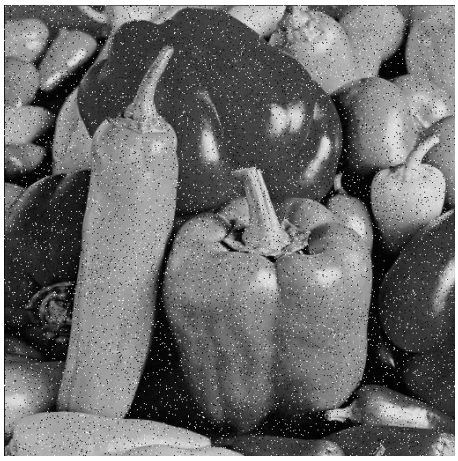


(b) Imagem corrompida

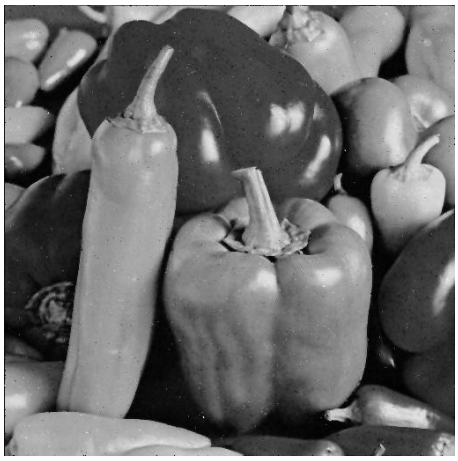
(c) Imagem filtrada

Figura B.32: Imagem *Peppers* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro sigma aditivo.

(a) Imagem original



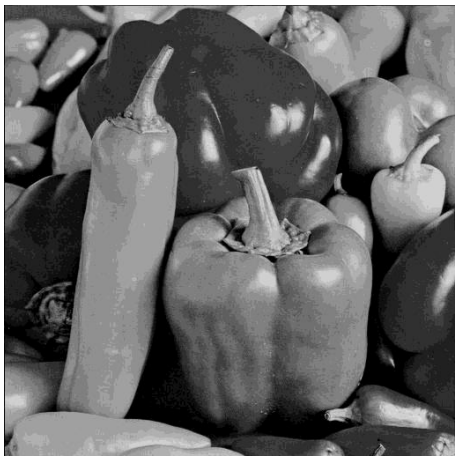
(b) Imagem corrompida



(c) Imagem filtrada

Figura B.33: Imagem *Peppers* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro sigma aditivo.

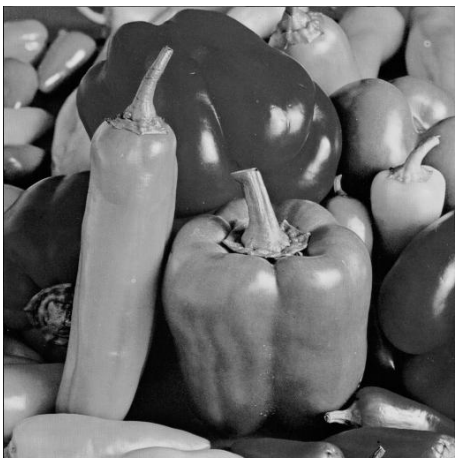
(a) Imagem original



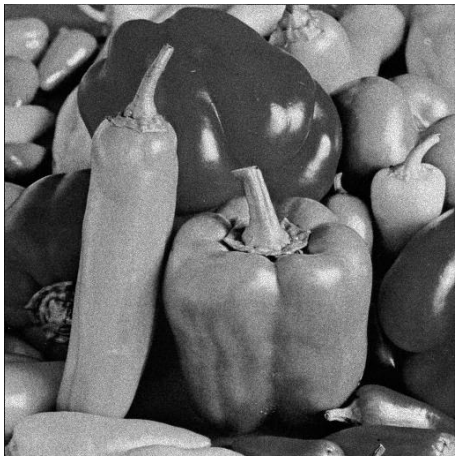
(b) Imagem corrompida

(c) Imagem filtrada

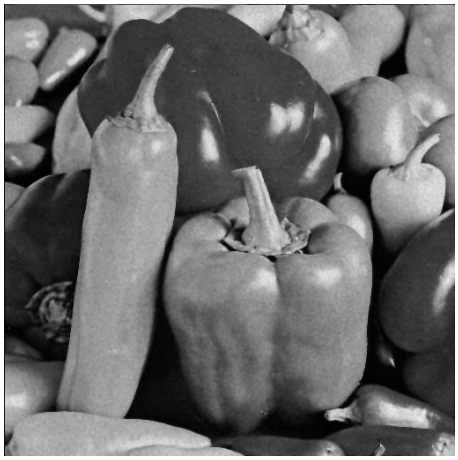
Figura B.34: Imagem *Peppers* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro sigma aditivo.



(a) Imagem original

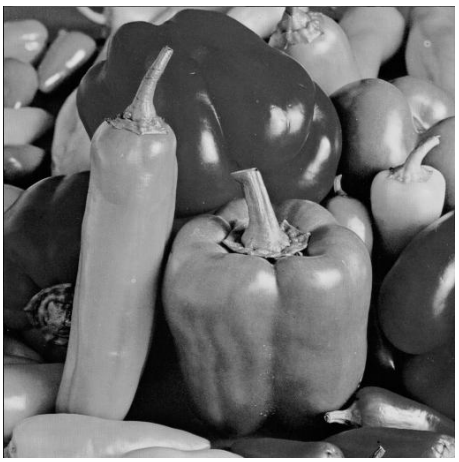


(b) Imagem corrompida

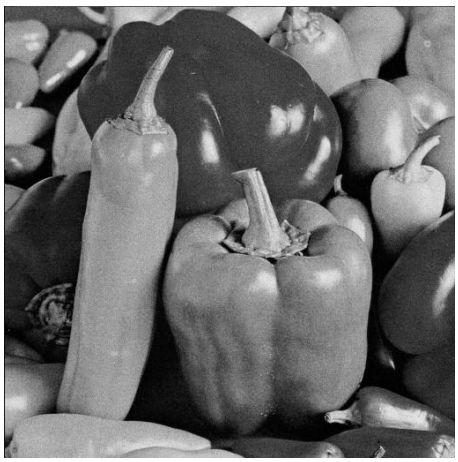


(c) Imagem filtrada

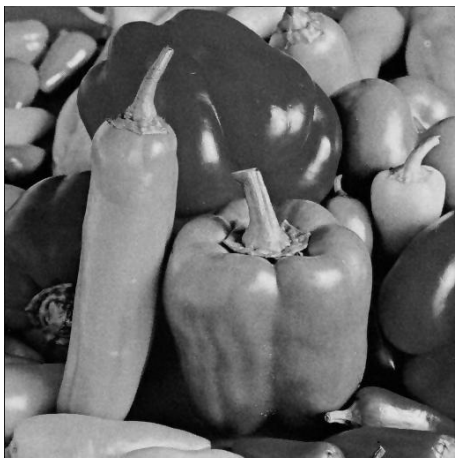
Figura B.35: Imagem *Peppers* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro sigma aditivo.



(a) Imagem original



(b) Imagem corrompida



(c) Imagem filtrada

Figura B.36: Imagem *Peppers* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro sigma aditivo.

B.4 Filtro sigma adaptativo

B.4.1 Imagem Moedas

(a) Imagem original

(b) Imagem filtrada (Janela 3×3, $\alpha = 0,5$, $K = 3$)

(c) Imagem filtrada (Janela 3×3, $\alpha = 1,5$, $K = 3$)

(d) Imagem filtrada (Janela 5×5, $\alpha = 0,5$, $K = 5$)

(e) Imagem filtrada (Janela 5×5, $\alpha = 1,5$, $K = 5$)

Figura B.37: Resultados da filtragem da imagem *Moedas* pelo filtro sigma adaptativo.

B.4.2 Imagem Máscaras

(a) Imagem original

(b) Imagem filtrada (Janela 3×3, $\alpha = 0,5$, $K = 3$)

(c) Imagem filtrada (Janela 3×3, $\alpha = 1,5$, $K = 3$)

(d) Imagem filtrada (Janela 5×5, $\alpha = 0,5$, $K = 5$)

(e) Imagem filtrada (Janela 5×5, $\alpha = 1,5$, $K = 5$)

Figura B.38: Resultados da filtragem da imagem *Máscaras* pelo filtro sigma adaptativo.

B.4.3 Imagem Lena

Janela	α	K	T	$S\&P$	Q	F	G
3×3	0,5	1	29,8787	24,5429	29,5335	28,7238	33,1587
		3	31,2695	29,2800	29,3563	30,3178	33,8410
		5	31,3227	29,3450	29,3306	30,3791	33,8390
	1	1	29,9349	24,5367	**29,5845**	28,7347	33,2940
		3	30,9862	29,3613	29,4766	29,9660	33,8654
		5	31,3349	29,4258	29,3435	30,3951	33,8683
	1,5	1	30,6234	24,7086	29,5652	29,4517	33,7894
		3	31,0173	29,4137	29,5321	29,9525	33,9686
		5	31,3487	29,6346	29,4206	30,3824	34,0050
	2	1	31,2664	25,6656	29,4891	30,2097	34,0857
		3	31,3368	29,1249	29,4835	30,3210	**34,1125**
		5	**31,4326**	**29,7256**	29,4491	**30,4692**	34,1106
5×5	0,5	1	29,4107	21,4451	29,5812	28,0984	32,9736
		4	30,5959	28,3386	29,4542	29,5378	33,6847
		10	31,3388	28,5011	29,2466	30,4660	33,8196
	1	1	29,9831	21,4315	29,5519	28,6901	33,4105
		4	30,4366	28,3038	29,5226	29,2612	33,7008
		10	31,2146	28,6714	29,3297	30,2784	33,9072
	1,5	1	30,7289	21,5377	29,3758	29,5817	33,7131
		4	30,8740	28,1906	29,3683	29,7735	33,7950
		10	31,1527	28,7659	29,3077	30,1806	33,8549
	2	1	31,0634	21,8133	29,0869	30,1145	33,4446
		4	31,0907	27,9742	29,0855	30,1550	33,4576
		10	31,1598	28,6962	29,0711	30,2694	33,4708

Tabela B.7: Resultados de PSNR do filtro sigma adaptativo (imagem *Lena*).

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.39: Imagem *Lena* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro sigma adaptativo.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.40: Imagem *Lena* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro sigma adaptativo.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.41: Imagem *Lena* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro sigma adaptativo.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.42: Imagem *Lena* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro sigma adaptativo.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

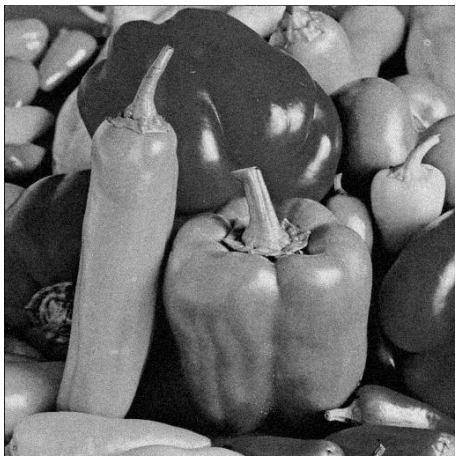
Figura B.43: Imagem *Lena* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro sigma adaptativo.

B.4.4 Imagem Peppers

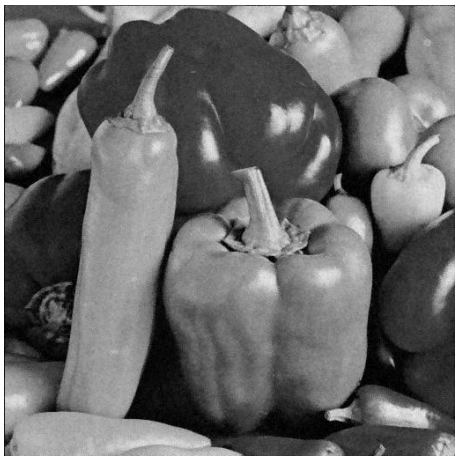
Janela	α	K	T	$S\&P$	Q	F	G
3×3	0,5	1	29,9920	24,4071	29,7172	29,5259	33,6618
		3	31,4485	29,4141	29,6019	31,0990	34,3962
		5	31,5023	29,5335	29,5817	31,1615	34,4064
	1	1	30,0483	24,3979	29,7693	29,5420	33,8292
		3	31,1275	29,3938	29,6869	30,7409	34,3934
		5	31,5143	29,6349	29,5968	31,1779	34,4435
	1,5	1	30,7414	24,5457	**29,7741**	30,2711	34,3446
		3	31,1331	29,4264	29,7355	30,7359	34,5136
		5	31,5051	29,8412	29,6519	31,1646	34,5452
	2	1	31,4061	25,4479	29,7241	31,0330	34,6691
		3	31,4687	29,2098	29,7115	31,1264	**34,6836**
		5	31,5781	**29,8922**	29,6798	31,2561	34,6725
5×5	0,5	1	29,5224	21,1860	29,7531	28,9145	33,4995
		4	30,7596	28,3494	29,6851	30,3253	34,2485
		10	**31,5821**	28,6921	29,5734	**31,2572**	34,4846
	1	1	30,1054	21,1940	29,7698	29,5236	33,9737
		4	30,5617	28,2658	29,7382	30,0518	34,2417
		10	31,4325	28,7665	29,6362	31,0807	34,5502
	1,5	1	30,9297	21,3026	29,6951	30,4512	34,4202
		4	31,0712	28,1623	29,6795	30,6239	34,4890
		10	31,3805	28,8098	29,6312	30,9977	34,5470
	2	1	31,4250	21,5286	29,5383	31,0535	34,3778
		4	31,4522	28,1168	29,5285	31,0881	34,3819
		10	31,5207	28,8244	29,5119	31,1832	34,3844

Tabela B.8: Resultados de PSNR do filtro sigma adaptativo (imagem *Peppers*).

(a) Imagem original



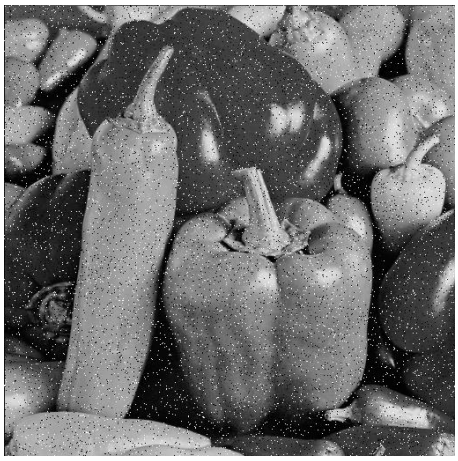
(b) Imagem corrompida



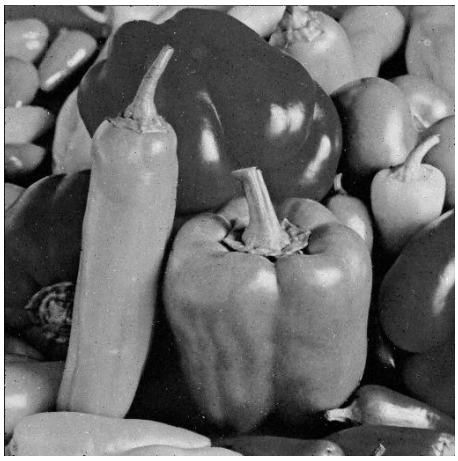
(c) Imagem filtrada

Figura B.44: Imagem *Peppers* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro sigma adaptativo.

(a) Imagem original



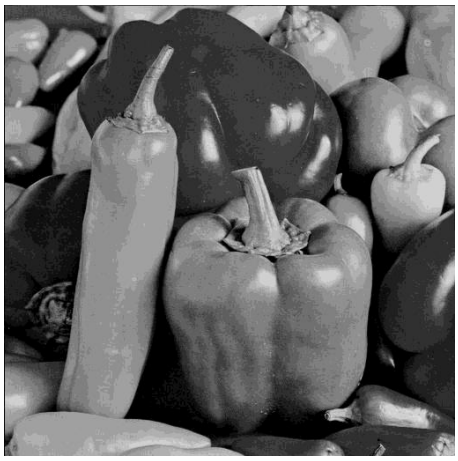
(b) Imagem corrompida



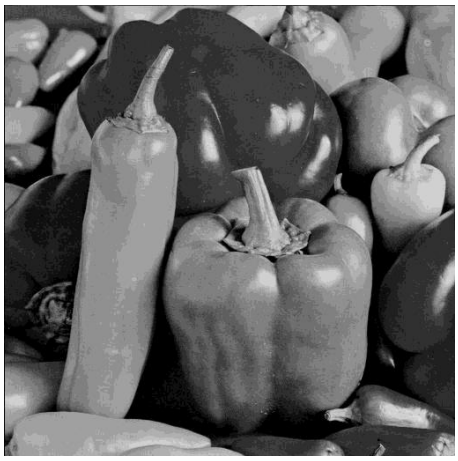
(c) Imagem filtrada

Figura B.45: Imagem *Peppers* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro sigma adaptativo.

(a) Imagem original

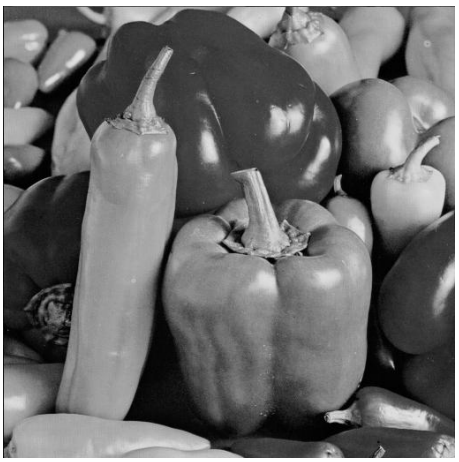


(b) Imagem corrompida

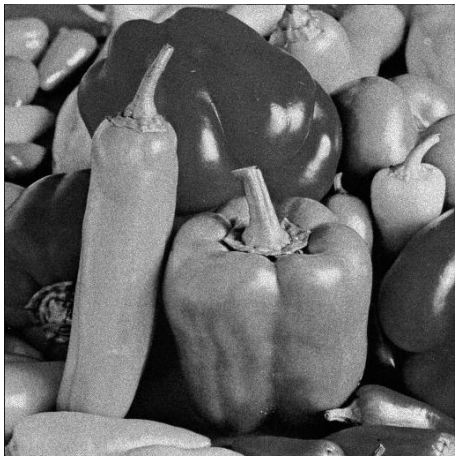


(c) Imagem filtrada

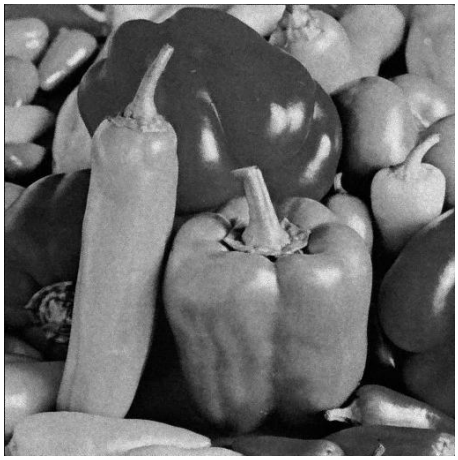
Figura B.46: Imagem *Peppers* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro sigma adaptativo.



(a) Imagem original

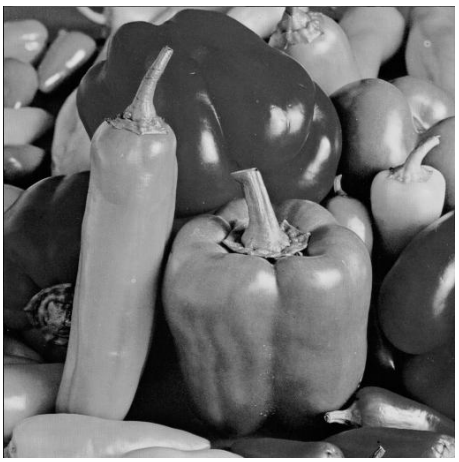


(b) Imagem corrompida

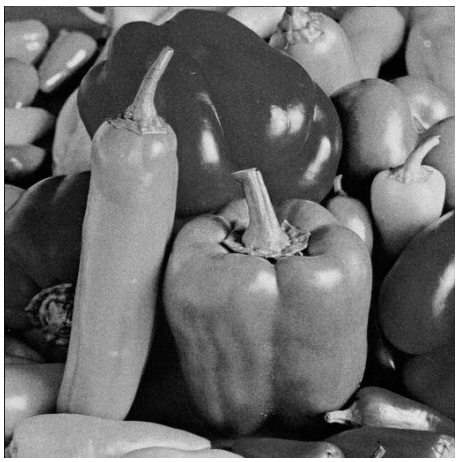


(c) Imagem filtrada

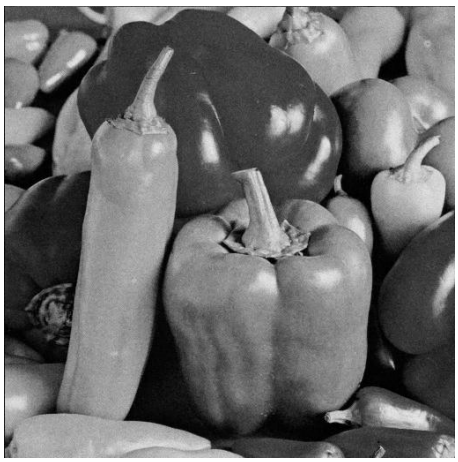
Figura B.47: Imagem *Peppers* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro sigma adaptativo.



(a) Imagem original



(b) Imagem corrompida



(c) Imagem filtrada

Figura B.48: Imagem *Peppers* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro sigma adaptativo.

B.5 Filtro sigma multiplicativo

B.5.1 Imagem Moedas

(a) Imagem original

(b) Imagem filtrada (Janela 3×3, $\Lambda = 0,5\frac{\sigma_x}{\overline{x}}$, $K = 3$)

(c) Imagem filtrada (Janela 3×3, $\Lambda = 1,5\frac{\sigma_x}{\overline{x}}$, $K = 3$)

(d) Imagem filtrada (Janela 5×5, $\Lambda = 0,5\frac{\sigma_x}{\overline{x}}$, $K = 5$)

(e) Imagem filtrada (Janela 5×5, $\Lambda = 1,5\frac{\sigma_x}{\overline{x}}$, $K = 5$)

Figura B.49: Resultados da filtragem da imagem *Moedas* pelo filtro sigma multiplicativo.

B.5.2 Imagem Máscaras

(a) Imagem original

(b) Imagem filtrada (Janela 3×3, $\Lambda = 0,5\frac{\sigma_x}{\overline{x}}$, $K = 3$)

(c) Imagem filtrada (Janela 3×3, $\Lambda = 1,5\frac{\sigma_x}{\overline{x}}$, $K = 3$)

(d) Imagem filtrada (Janela 5×5, $\Lambda = 0,5\frac{\sigma_x}{\overline{x}}$, $K = 5$)

(e) Imagem filtrada (Janela 5×5, $\Lambda = 1,5\frac{\sigma_x}{\overline{x}}$, $K = 5$)

Figura B.50: Resultados da filtragem da imagem *Máscaras* pelo filtro sigma multiplicativo.

B.5.3 Imagem Lena

Janela	Λ	K	T	$S\&P$	Q	F	G
3×3	$0,5\frac{\sigma_x}{\overline{x}}$	1	32,2594	27,0017	**29,4301**	31,5471	35,3453
		3	32,5484	29,4141	29,3604	31,7418	34,8584
		5	32,5390	27,4299	29,1849	31,7481	34,2039
	$\frac{\sigma_x}{\overline{x}}$	1	33,1464	27,3791	29,0142	**33,1126**	34,8521
		3	**33,2355**	**31,3384**	29,0144	33,0824	34,7365
		5	33,1649	29,9916	29,0057	32,9122	34,4333
	$1,5\frac{\sigma_x}{\overline{x}}$	1	33,1140	28,1714	28,7954	32,9139	34,2213
		3	33,1539	31,2130	28,7936	32,8904	34,1882
		5	33,1067	30,7402	28,7941	32,7947	34,0581
	$2\frac{\sigma_x}{\overline{x}}$	1	32,9928	28,8623	28,7442	32,7398	33,9187
		3	33,0108	30,7575	28,7420	32,7213	33,9077
		5	32,9744	30,6091	28,7332	32,6703	33,8387
5×5	$0,5\frac{\sigma_x}{\overline{x}}$	1	32,6240	24,2338	29,2894	32,2586	**35,4824**
		4	32,7674	30,6921	29,2639	32,3111	35,2160
		10	32,7726	28,7739	29,0515	32,2246	34,3035
	$\frac{\sigma_x}{\overline{x}}$	1	32,7704	24,7035	28,4807	33,0005	33,7073
		4	32,8241	30,8547	28,4788	32,9842	33,6760
		10	32,7654	30,7359	28,4626	32,8289	33,4432
	$1,5\frac{\sigma_x}{\overline{x}}$	1	31,7442	25,9752	27,8292	31,7486	32,0452
		4	31,7620	30,0395	27,8283	31,7392	32,0390
		10	31,7386	30,5093	27,8230	31,6842	31,9821
	$2\frac{\sigma_x}{\overline{x}}$	1	30,8649	27,8811	27,4091	30,8329	31,0489
		4	30,8735	29,5690	27,4085	30,8296	31,0472
		10	30,8554	29,9470	27,4019	30,8005	31,0243

Tabela B.9: Resultados de PSNR do filtro sigma multiplicativo (imagem *Lena*).

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.51: Imagem *Lena* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro sigma multiplicativo.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.52: Imagem *Lena* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro sigma multiplicativo.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.53: Imagem *Lena* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro sigma multiplicativo.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.54: Imagem *Lena* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro sigma multiplicativo.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

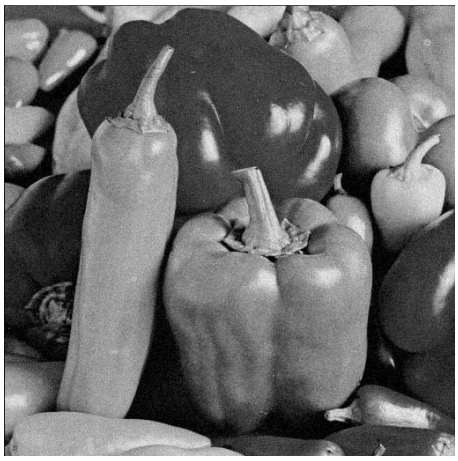
Figura B.55: Imagem *Lena* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro sigma multiplicativo.

B.5.4 Imagem Peppers

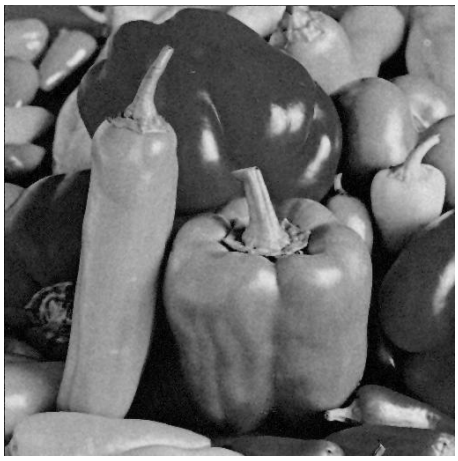
Janela	Λ	K	T	$S\&P$	Q	F	G
3×3	$0,5\frac{\sigma_x}{\overline{x}}$	1	32,3782	25,6850	**29,4649**	33,1883	**35,4274**
		3	32,6257	28,2186	29,3942	32,9580	34,7547
		5	32,7279	26,9234	29,2832	32,7894	34,3636
	$\frac{\sigma_x}{\overline{x}}$	1	**32,9704**	26,4086	29,1504	33,5574	34,8124
		3	32,8799	29,1270	29,1324	33,2478	34,3617
		5	32,8924	28,3224	29,1115	33,1012	34,1447
	$1,5\frac{\sigma_x}{\overline{x}}$	1	32,8083	27,5918	28,8813	33,0770	34,1174
		3	32,6849	28,9243	28,8689	32,8563	33,8285
		5	32,6171	28,4049	28,8698	32,7471	33,6685
	$2\frac{\sigma_x}{\overline{x}}$	1	31,5533	26,9698	28,0555	31,6819	32,3632
		3	31,5643	27,6120	28,0527	31,6477	32,3339
		5	31,5080	27,3087	28,0558	31,5511	32,2152
5×5	$0,5\frac{\sigma_x}{\overline{x}}$	1	32,5701	22,8981	29,2005	**33,7965**	35,2108
		4	32,6251	28,9867	29,1612	33,4852	34,6736
		10	32,8063	27,8762	29,0517	33,2640	34,1777
	$\frac{\sigma_x}{\overline{x}}$	1	32,4484	23,6255	28,5332	33,1002	33,4631
		4	32,3675	**29,2722**	28,5189	32,8975	33,2669
		10	32,3006	28,8496	28,5043	32,6791	32,9776
	$1,5\frac{\sigma_x}{\overline{x}}$	1	31,5546	26,2267	27,9319	31,8467	32,0903
		4	31,5023	29,0104	27,9212	31,7419	32,0133
		10	31,4135	28,8836	27,9308	31,6139	31,8354
	$2\frac{\sigma_x}{\overline{x}}$	1	29,3300	25,8013	26,4772	29,4497	29,5765
		4	29,3376	27,5583	26,4753	29,4396	29,5664
		10	29,3319	27,4519	26,4744	29,4054	29,5344

Tabela B.10: Resultados de PSNR do filtro sigma multiplicativo (imagem *Peppers*).

(a) Imagem original



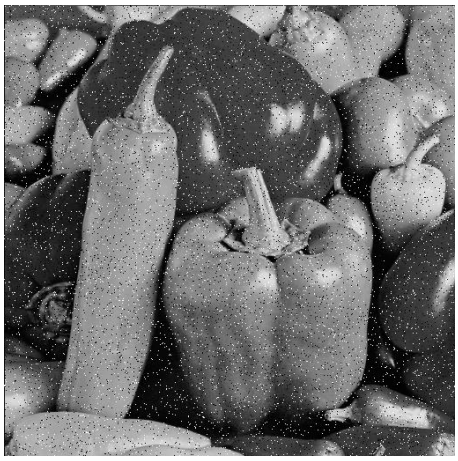
(b) Imagem corrompida



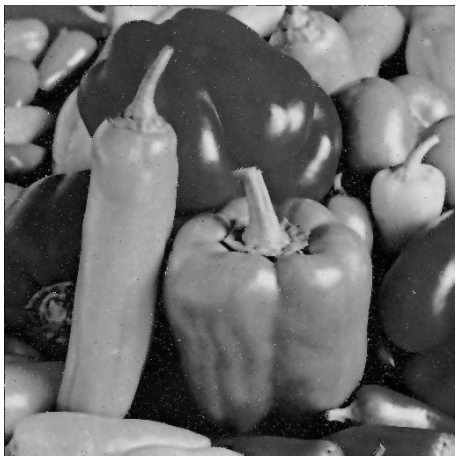
(c) Imagem filtrada

Figura B.56: Imagem *Peppers* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro sigma multiplicativo.

(a) Imagem original



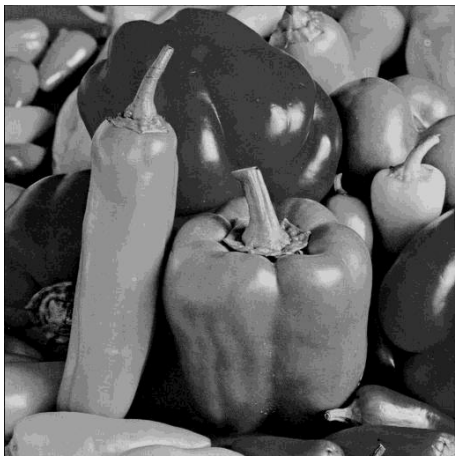
(b) Imagem corrompida



(c) Imagem filtrada

Figura B.57: Imagem *Peppers* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro sigma multiplicativo.

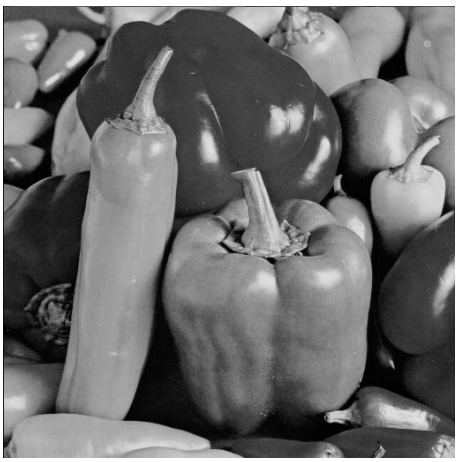
(a) Imagem original



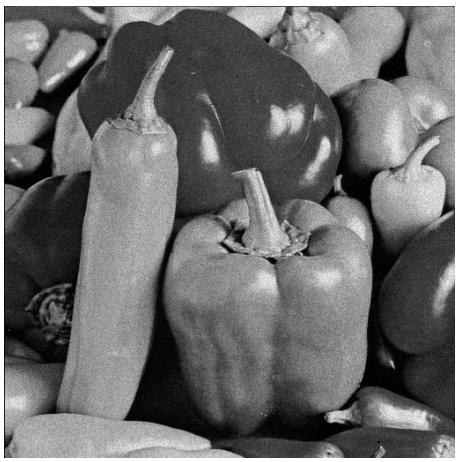
(b) Imagem corrompida

(c) Imagem filtrada

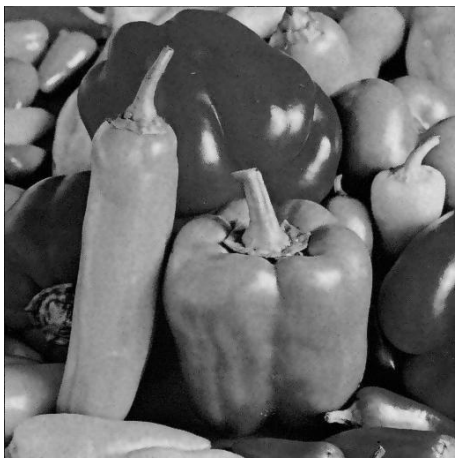
Figura B.58: Imagem *Peppers* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro sigma multiplicativo.



(a) Imagem original

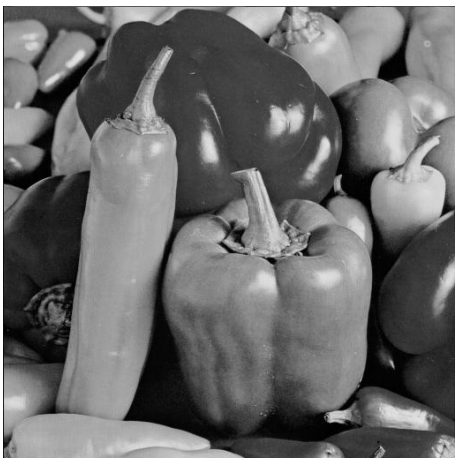


(b) Imagem corrompida

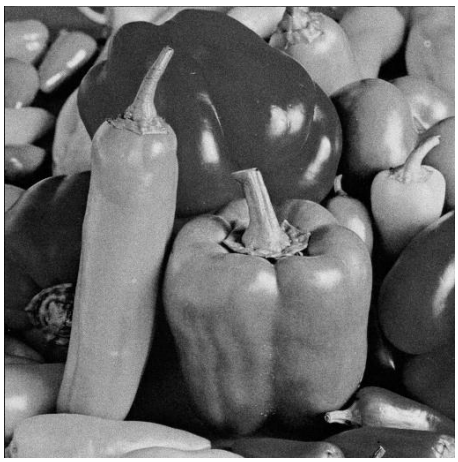


(c) Imagem filtrada

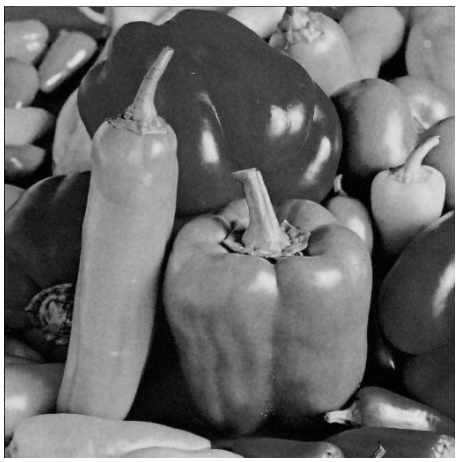
Figura B.59: Imagem *Peppers* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro sigma multiplicativo.



(a) Imagem original



(b) Imagem corrompida



(c) Imagem filtrada

Figura B.60: Imagem *Peppers* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro sigma multiplicativo.

B.6 Filtro sigma híbrido

B.6.1 Imagem Moedas

(a) Imagem original

(b) Imagem filtrada (Janela 3×3, subjanela 3×3, $\alpha = 0,5$, $K = 3$)

(c) Imagem filtrada (Janela 3×3, subjanela 3×3, $\alpha = 1,5$, $K = 3$)

(d) Imagem filtrada (Janela 5×5, subjanela 5×5, $\alpha = 0,5$, $K = 5$)

(e) Imagem filtrada (Janela 5×5, subjanela 5×5, $\alpha = 1,5$, $K = 5$)

Figura B.61: Resultados da filtragem da imagem *Moedas* pelo filtro sigma híbrido.

B.6.2 Imagem Máscaras

(a) Imagem original

(b) Imagem filtrada (Janela 3×3, subjanela 3×3, $\alpha = 0,5$, $K = 3$)

(c) Imagem filtrada (Janela 3×3, subjanela 3×3, $\alpha = 1,5$, $K = 3$)

(d) Imagem filtrada (Janela 5×5, subjanela 5×5, $\alpha = 0,5$, $K = 5$)

(e) Imagem filtrada (Janela 5×5, subjanela 5×5, $\alpha = 1,5$, $K = 5$)

Figura B.62: Resultados da filtragem da imagem *Máscaras* pelo filtro sigma híbrido.

B.6.3 Imagem Lena

Janela	Sub-janela	α	K	T	$S\&P$	Q	F	G
3×3	3×3	0,5	1	31,2021	29,5947	**29,5641**	30,1282	34,0012
			3	32,1806	30,4852	29,3765	31,2508	34,2420
			5	32,4061	28,3197	29,1149	31,5487	33,8935
		1	1	31,4335	29,5535	29,3480	30,4430	34,0209
			3	31,6857	30,9492	29,3529	30,6969	34,1324
			5	32,3616	31,1405	29,3178	31,5243	34,2711
		1,5	1	32,6504	31,2563	29,1636	31,8762	**34,7661**
			3	32,6784	31,6814	29,1719	31,8887	34,7660
			5	**32,8070**	31,8549	29,1713	32,0281	34,7441
		2	1	31,6026	27,8464	28,1414	31,2324	32,4848
			3	31,6060	29,9689	28,1428	31,2304	32,4806
			5	31,6036	30,2321	28,1425	31,2220	32,4601
5×5	5×5	0,5	1	32,2417	30,5950	29,4130	31,3786	34,6078
			4	32,3030	31,1601	29,3943	31,4588	34,5582
			10	32,5838	29,9198	29,1752	31,7895	34,1423
		1	1	32,5834	31,5787	28,9820	31,9276	34,3048
			4	32,5830	31,7719	28,9826	31,9261	34,3014
			10	32,5966	**32,0993**	28,9952	31,9605	34,2276
		1,5	1	32,4618	30,5197	28,3887	32,0577	33,4360
			4	32,4620	30,5238	28,3890	**32,0577**	33,4360
			10	32,4550	30,6084	28,3912	32,0446	33,4162
		2	1	29,9708	29,0997	26,9433	29,8183	30,2432
			4	29,9709	29,1090	26,9434	29,8183	30,2432
			10	29,9687	29,1291	26,9438	29,8151	30,2407

Tabela B.11: Resultados de PSNR do filtro sigma híbrido (imagem *Lena*).

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.63: Imagem *Lena* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro sigma híbrido.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.64: Imagem *Lena* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro sigma híbrido.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.65: Imagem *Lena* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro sigma híbrido.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.66: Imagem *Lena* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro sigma híbrido.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

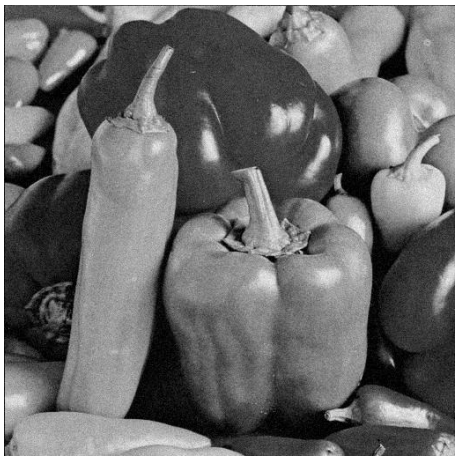
Figura B.67: Imagem *Lena* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro sigma híbrido.

B.6.4 Imagem Peppers

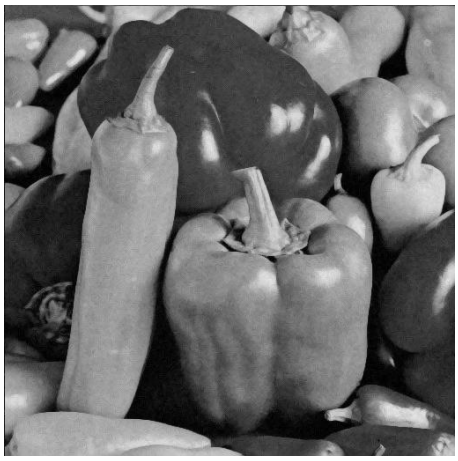
Janela	Sub-janela	α	K	T	$S\&P$	Q	F	G
3×3	3×3	0,5	1	31,1775	28,7537	**29,7271**	30,8008	34,2519
			3	32,0699	28,6929	29,5403	31,7256	34,2195
			5	32,3401	26,9500	29,3172	31,9500	33,9863
		1	1	31,5155	28,6284	29,5949	31,1748	34,4522
			3	31,6161	29,3588	29,5925	31,2468	34,2071
			5	32,2426	29,0804	29,5753	31,9894	34,2798
		1,5	1	32,5511	31,2260	29,2451	32,3804	34,8921
			3	32,5723	31,2308	29,2533	32,3750	34,8570
			5	32,6628	30,3496	29,2561	32,4377	34,7360
		2	1	31,8017	28,4208	28,2658	31,7911	32,9174
			3	31,8095	29,9450	28,2671	31,7792	32,9006
			5	31,7903	29,5692	28,2639	31,7405	32,8360
5×5	5×5	0,5	1	32,3229	30,3513	29,5885	32,0864	**34,9422**
			4	32,2165	30,0786	29,5445	31,9680	34,5924
			10	32,5307	28,3582	29,3934	32,2482	34,2212
		1	1	**32,8316**	**31,7653**	29,3091	**32,6740**	34,8303
			4	32,7148	31,6249	29,2849	32,5564	34,6954
			10	32,5627	30,7434	29,2947	32,4138	34,3115
		1,5	1	32,7028	31,1814	28,5594	32,6352	33,7973
			4	32,6972	31,2026	28,5592	32,6336	33,7913
			10	32,6685	31,2490	28,5578	32,5916	33,7332
		2	1	30,4743	29,6147	27,1843	30,4659	30,8600
			4	30,4744	29,6235	27,1844	30,4658	30,8598
			10	30,4651	29,5951	27,1833	30,4517	30,8414

Tabela B.12: Resultados de PSNR do filtro sigma híbrido (imagem *Peppers*).

(a) Imagem original



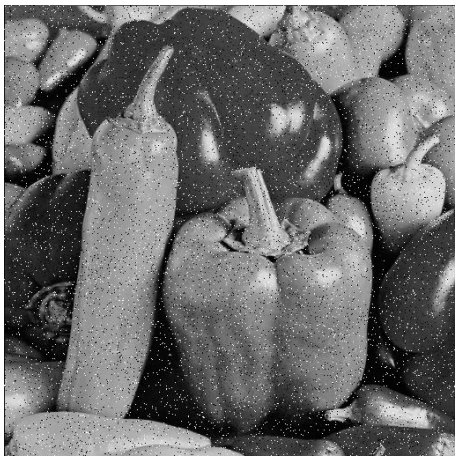
(b) Imagem corrompida



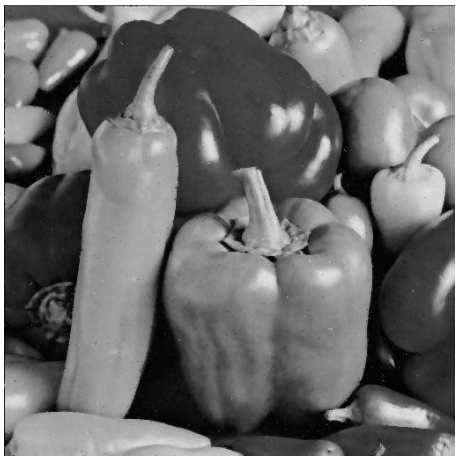
(c) Imagem filtrada

Figura B.68: Imagem *Peppers* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro sigma híbrido.

(a) Imagem original



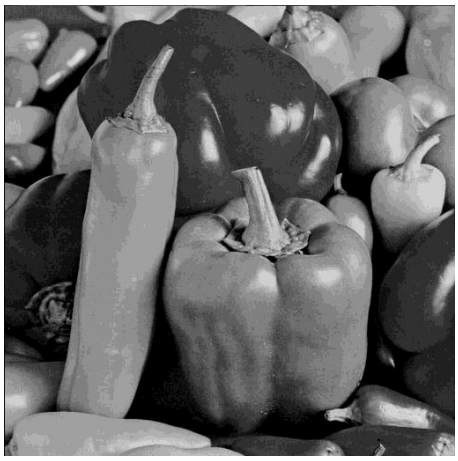
(b) Imagem corrompida



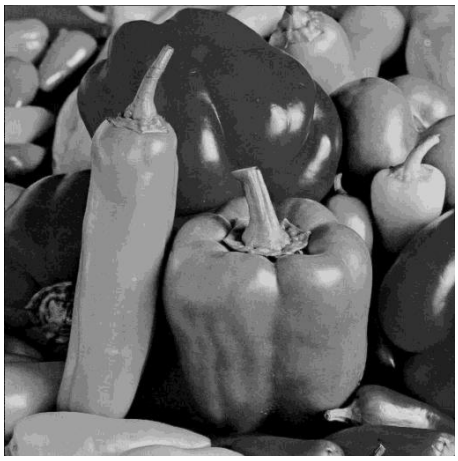
(c) Imagem filtrada

Figura B.69: Imagem *Peppers* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro sigma híbrido.

(a) Imagem original

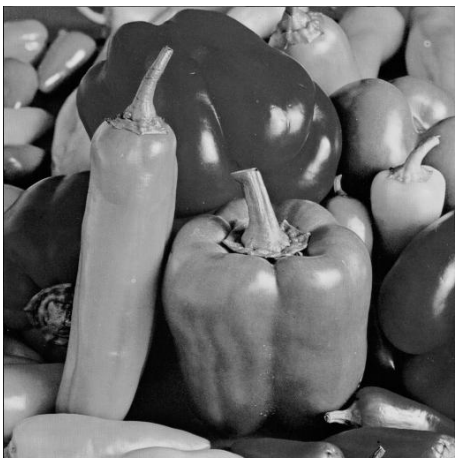


(b) Imagem corrompida

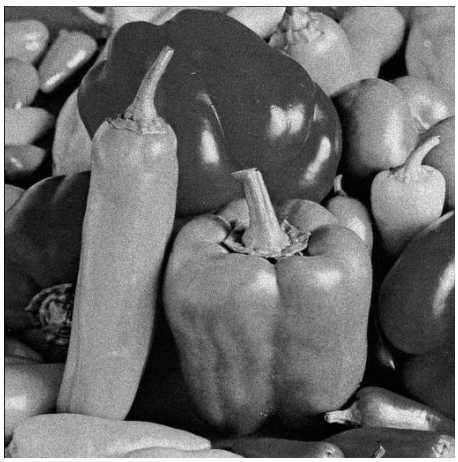


(c) Imagem filtrada

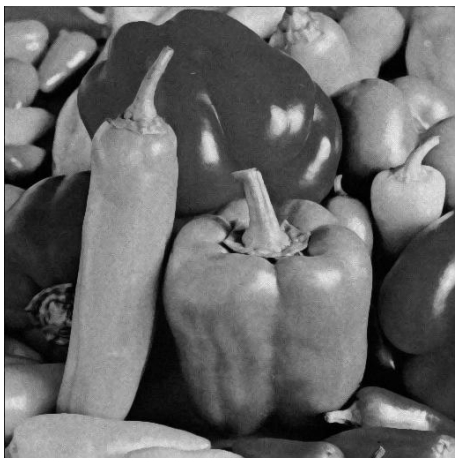
Figura B.70: Imagem *Peppers* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro sigma híbrido.



(a) Imagem original

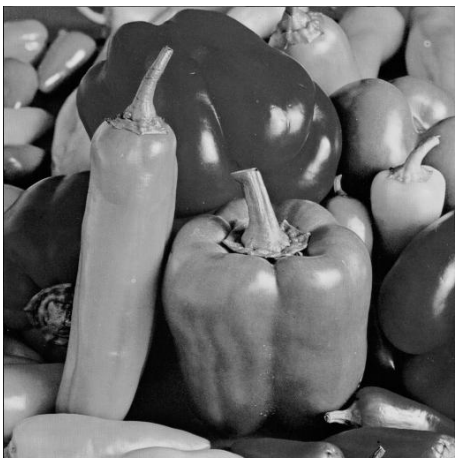


(b) Imagem corrompida

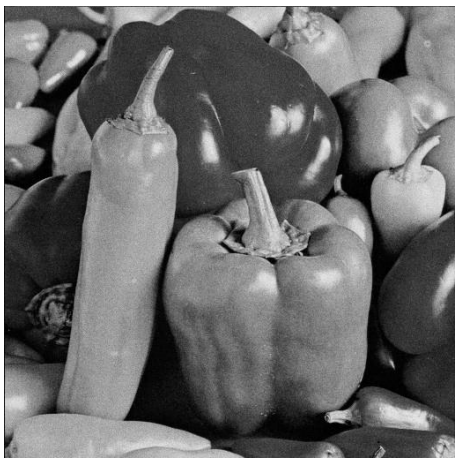


(c) Imagem filtrada

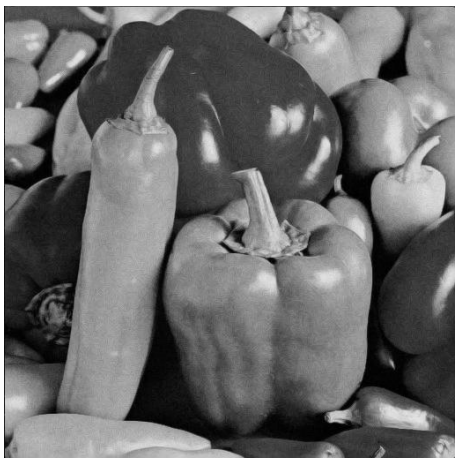
Figura B.71: Imagem *Peppers* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro sigma híbrido.



(a) Imagem original



(b) Imagem corrompida

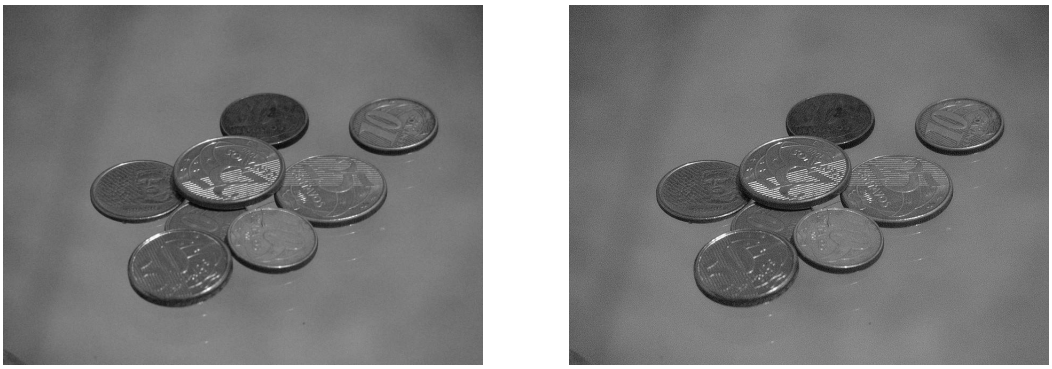


(c) Imagem filtrada

Figura B.72: Imagem *Peppers* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro sigma híbrido.

B.7 Filtro baseado em wavelets

B.7.1 Imagem Moedas



(a) Imagem original (b) Imagem filtrada

Figura B.73: Resultado da filtragem da imagem *Moedas* pelo filtro baseado em *wavelets*.

B.7.2 Imagem Máscaras

(a) Imagem original

(b) Imagem filtrada

Figura B.74: Resultado da filtragem da imagem *Máscaras* pelo filtro baseado em *wavelets*.

B.7.3 Imagem Lena

T	$S\&P$	Q	F	G
25,2096	**15,1314**	**28,3566**	**23,8778**	**28,5721**

Tabela B.13: Resultados de PSNR do filtro baseado em *wavelets* (imagem *Lena*).

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.75: Imagem *Lena* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro baseado em *wavelets*.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.76: Imagem *Lena* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro baseado em *wavelets*.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.77: Imagem *Lena* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro baseado em *wavelets*.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.78: Imagem *Lena* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro baseado em *wavelets*.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

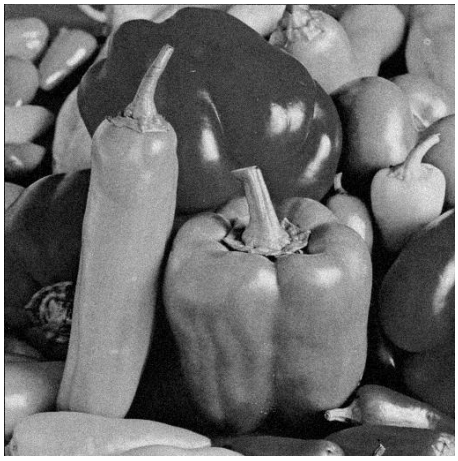
Figura B.79: Imagem *Lena* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro baseado em *wavelets*.

B.7.4 Imagem Peppers

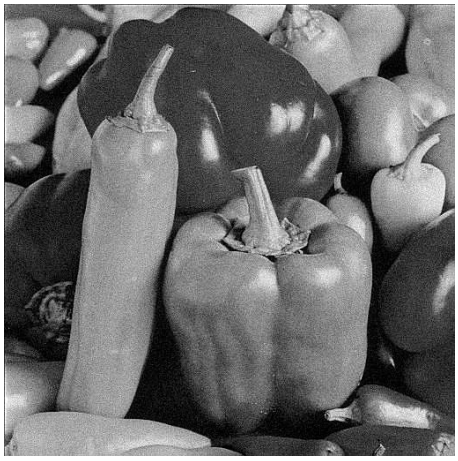
T	$S\&P$	Q	F	G
24,6637	**14,7654**	**27,1751**	**24,0776**	**27,5676**

Tabela B.14: Resultados de PSNR do filtro baseado em *wavelets* (imagem *Peppers*).

(a) Imagem original



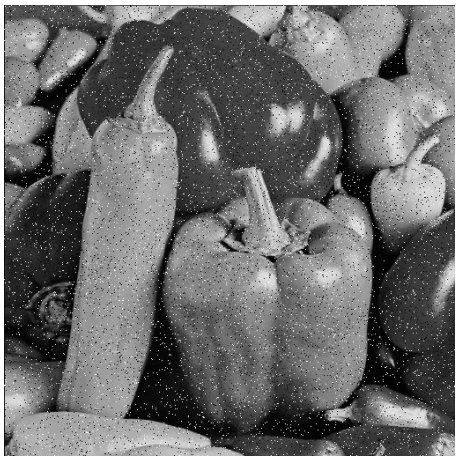
(b) Imagem corrompida



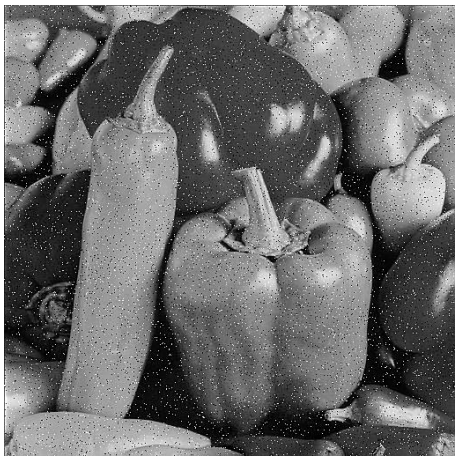
(c) Imagem filtrada

Figura B.80: Imagem *Peppers* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro baseado em *wavelets*.

(a) Imagem original



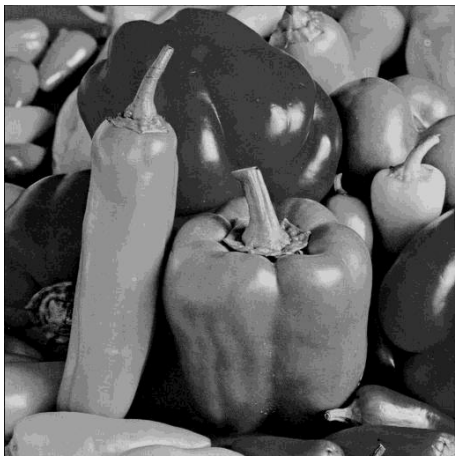
(b) Imagem corrompida



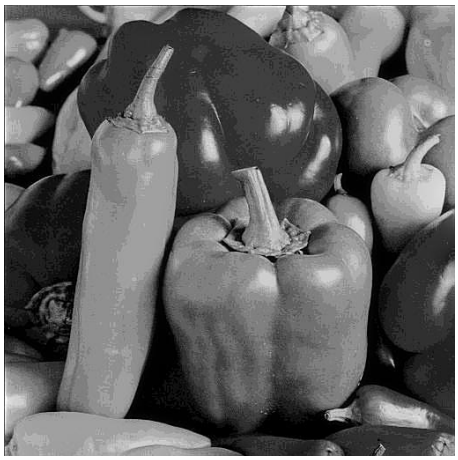
(c) Imagem filtrada

Figura B.81: Imagem *Peppers* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro baseado em *wavelets*.

(a) Imagem original

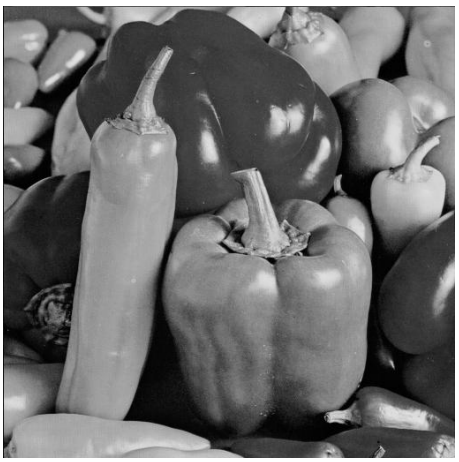


(b) Imagem corrompida

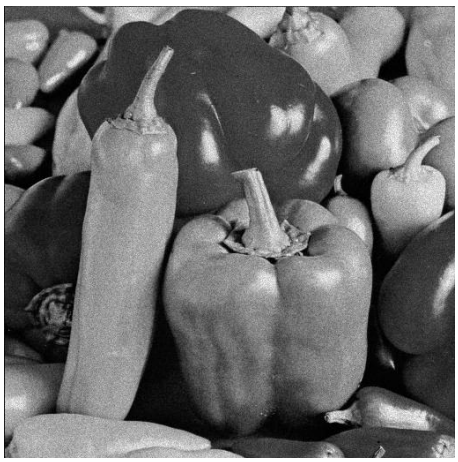


(c) Imagem filtrada

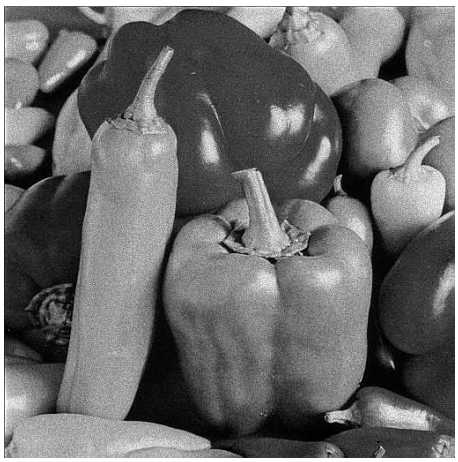
Figura B.82: Imagem *Peppers* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro baseado em *wavelets*.



(a) Imagem original

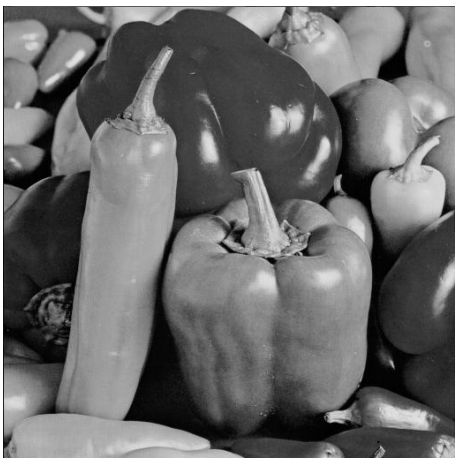


(b) Imagem corrompida

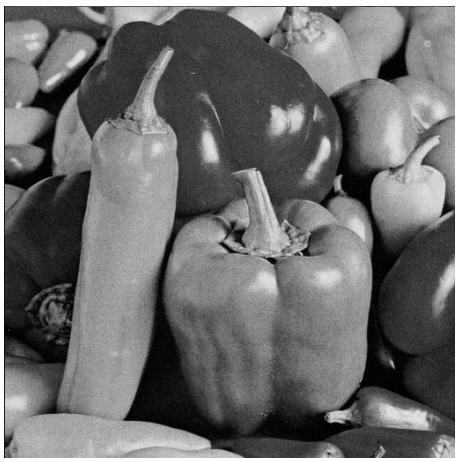


(c) Imagem filtrada

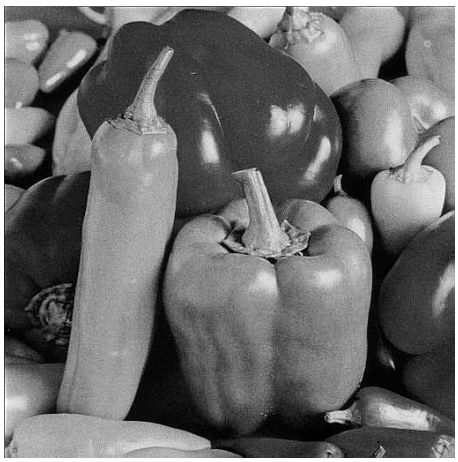
Figura B.83: Imagem *Peppers* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro baseado em *wavelets*.



(a) Imagem original



(b) Imagem corrompida

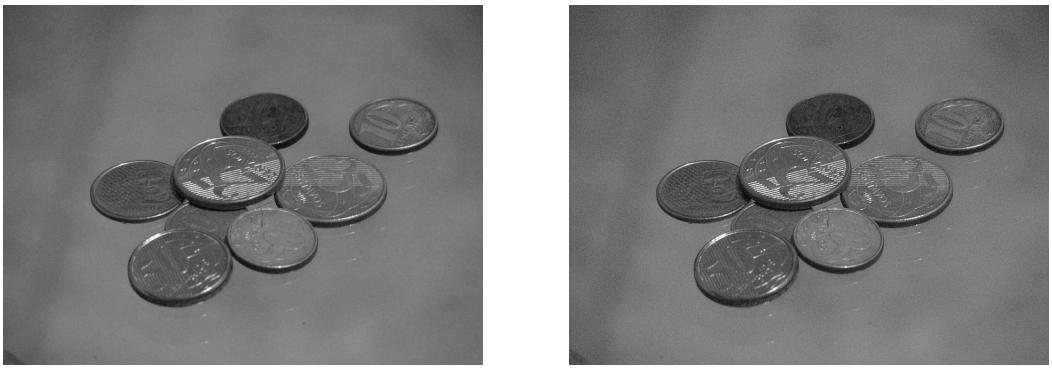


(c) Imagem filtrada

Figura B.84: Imagem *Peppers* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro baseado em *wavelets*.

B.8 Filtro baseado em wavelets modificado

B.8.1 Imagem Moedas



(a) Imagem original (b) Imagem filtrada

Figura B.85: Resultado da filtragem da imagem *Moedas* pelo filtro baseado em *wavelets* modificado.

B.8.2 Imagem Máscaras

(a) Imagem original

(b) Imagem filtrada

Figura B.86: Resultado da filtragem da imagem *Máscaras* pelo filtro baseado em *wavelets* modificado.

B.8.3 Imagem Lena

T	$S\&P$	Q	F	G
27,1641	**16,2713**	**28,5916**	**26,0006**	**29,7947**

Tabela B.15: Resultados de PSNR do filtro baseado em *wavelets* modificado (imagem *Lena*).

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.87: Imagem *Lena* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro baseado em *wavelets* modificado.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.88: Imagem *Lena* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro baseado em *wavelets* modificado.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.89: Imagem *Lena* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro baseado em *wavelets* modificado.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.90: Imagem *Lena* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro baseado em *wavelets* modificado.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

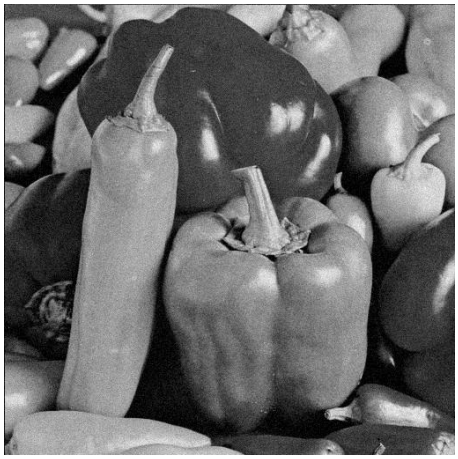
Figura B.91: Imagem *Lena* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro baseado em *wavelets* modificado.

B.8.4 Imagem Peppers

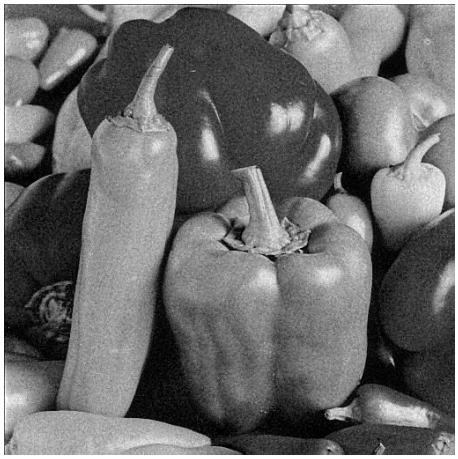
T	$S\&P$	Q	F	G
26,2289	**15,7419**	**27,2937**	**25,6172**	**28,3802**

Tabela B.16: Resultados de PSNR do filtro baseado em *wavelets* modificado (imagem *Peppers*).

(a) Imagem original



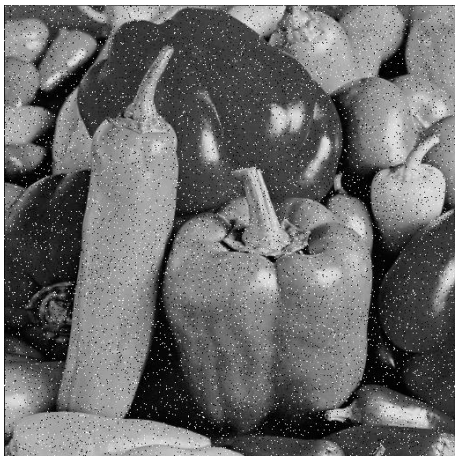
(b) Imagem corrompida



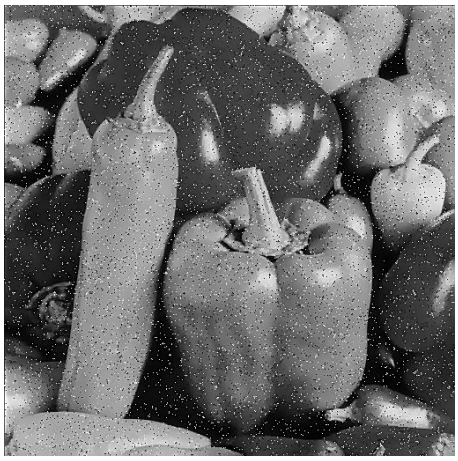
(c) Imagem filtrada

Figura B.92: Imagem *Peppers* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro baseado em *wavelets* modificado.

(a) Imagem original



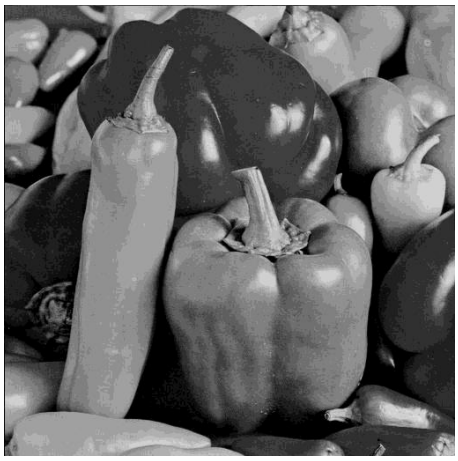
(b) Imagem corrompida



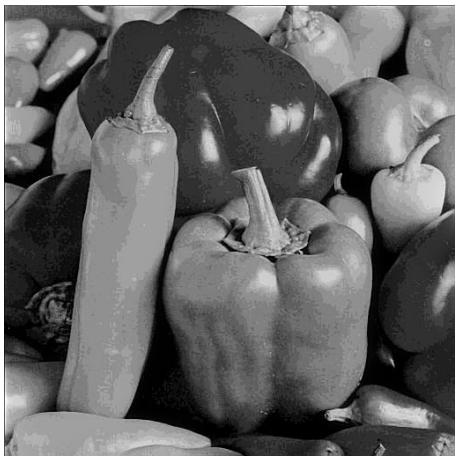
(c) Imagem filtrada

Figura B.93: Imagem *Peppers* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro baseado em *wavelets* modificado.

(a) Imagem original

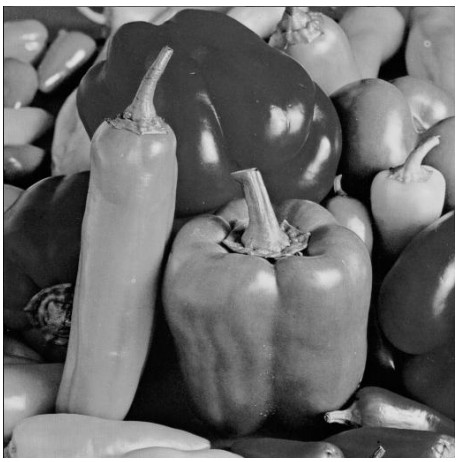


(b) Imagem corrompida

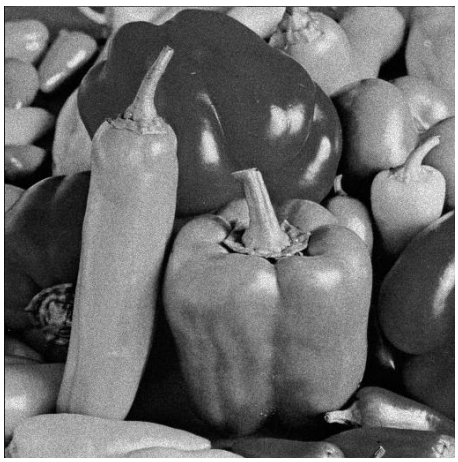


(c) Imagem filtrada

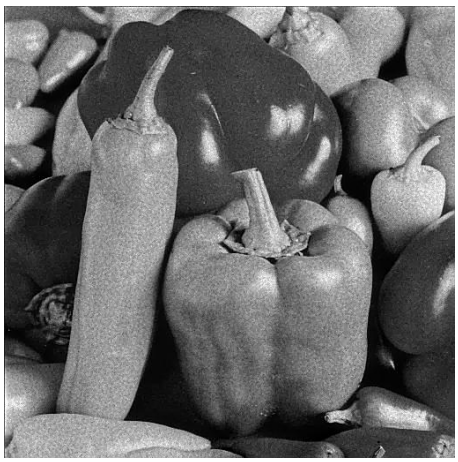
Figura B.94: Imagem *Peppers* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro baseado em *wavelets* modificado.



(a) Imagem original

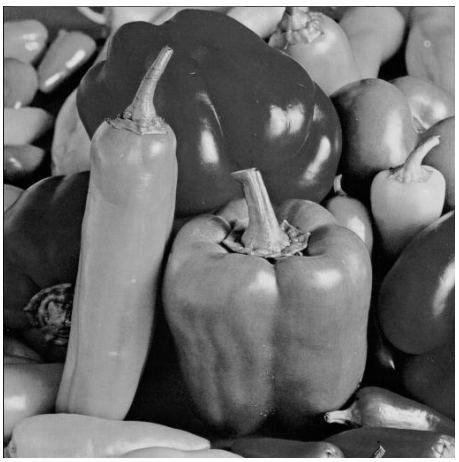


(b) Imagem corrompida

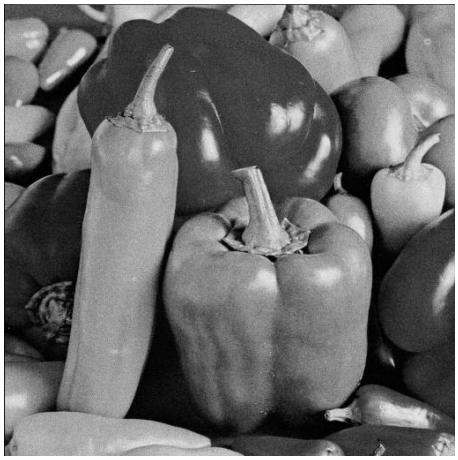


(c) Imagem filtrada

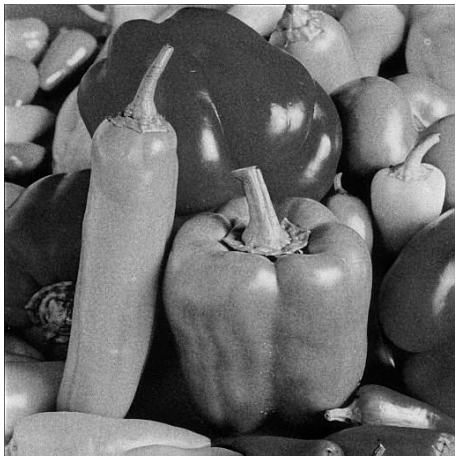
Figura B.95: Imagem *Peppers* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro baseado em *wavelets* modificado.



(a) Imagem original



(b) Imagem corrompida



(c) Imagem filtrada

Figura B.96: Imagem *Peppers* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro baseado em *wavelets* modificado.

B.9 Filtro baseado em wavelets proposto

B.9.1 Imagem Moedas

(a) Imagem original

(b) Imagem filtrada ($N = 0,1$)

(c) Imagem filtrada ($N = 0,3$)

(d) Imagem filtrada ($N = 0,5$)

(e) Imagem filtrada ($N = 0,7$)

(f) Imagem filtrada ($N = 0,9$)

Figura B.97: Resultados da filtragem da imagem *Moedas* pelo filtro baseado em *wavelets* proposto.

B.9.2 Imagem Máscaras

(a) Imagem original

(b) Imagem filtrada ($N = 0,1$)

(c) Imagem filtrada ($N = 0,3$)

(d) Imagem filtrada ($N = 0,5$)

(e) Imagem filtrada ($N = 0,7$)

(f) Imagem filtrada ($N = 0,9$)

Figura B.98: Resultados da filtragem da imagem *Máscaras* pelo filtro baseado em *wavelets* proposto.

B.9.3 Imagem Lena

	T	$S\&P$	Q	F	G
$N = 0,05$	27,1883	16,0508	28,6871	25,8366	30,2243
$N = 0,10$	27,7970	16,2464	28,7196	26,4396	30,6737
$N = 0,15$	28,3054	16,4207	**28,7212**	26,9707	30,9678
$N = 0,20$	28,7237	16,5862	28,6974	27,4332	31,1479
$N = 0,25$	29,0500	16,7673	28,6553	27,8282	31,2368
$N = 0,30$	29,2993	16,9521	28,5995	28,1430	**31,2592**
$N = 0,35$	29,4767	17,1399	28,5327	28,4002	31,2378
$N = 0,40$	29,5978	17,3359	28,4574	28,6027	31,1751
$N = 0,45$	29,6649	17,5363	28,3762	28,7577	31,0768
$N = 0,50$	29,6941	17,7652	28,2926	28,8630	30,9714
$N = 0,55$	**29,6987**	17,9985	28,2093	28,9299	30,8622
$N = 0,60$	29,6774	18,2421	28,1292	28,9724	30,7375
$N = 0,65$	29,6331	18,4792	28,0481	**28,9960**	30,6084
$N = 0,70$	29,5770	18,7189	27,9665	28,9868	30,4869
$N = 0,75$	29,5057	18,9553	27,8843	28,9641	30,3596
$N = 0,80$	29,4328	19,1854	27,8037	28,9292	30,2401
$N = 0,85$	29,3521	19,4184	27,7241	28,8860	30,1094
$N = 0,90$	29,2666	19,6384	27,6445	28,8270	29,9782
$N = 0,95$	29,1654	19,8527	27,5643	28,7593	29,8471
$N = 1,00$	29,0630	20,0805	27,4812	28,6806	29,7118
$N = 1,05$	28,9605	20,3224	27,3988	28,6068	29,5795
$N = 1,10$	28,8556	20,5629	27,3175	28,5208	29,4545
$N = 1,15$	28,7489	20,8175	27,2358	28,4371	29,3282
$N = 1,20$	28,6411	21,0685	27,1570	28,3474	29,2035
$N = 1,25$	28,5374	**21,3223**	27,0805	28,2528	29,0772

Tabela B.17: Resultados de PSNR do filtro baseado em *wavelets* proposto (imagem *Lena*).

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.99: Imagem *Lena* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro baseado em *wavelets* proposto.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.100: Imagem *Lena* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro baseado em *wavelets* proposto.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.101: Imagem *Lena* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro baseado em *wavelets* proposto.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.102: Imagem *Lena* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro baseado em *wavelets* proposto.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

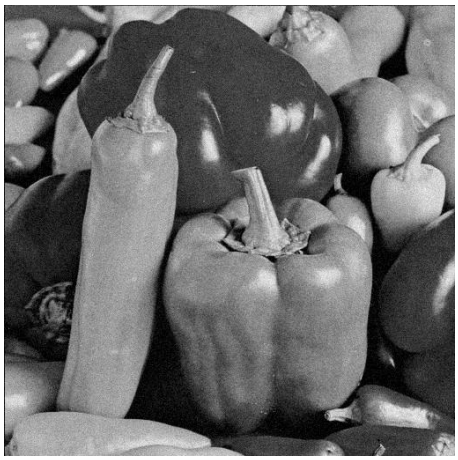
Figura B.103: Imagem *Lena* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro baseado em *wavelets* proposto.

B.9.4 Imagem Peppers

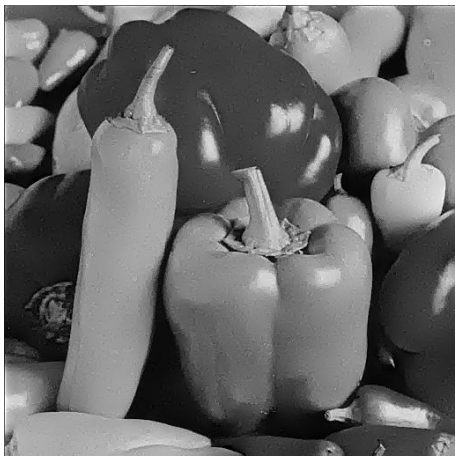
	T	$S\&P$	Q	F	G
$N = 0,05$	26,4236	15,6149	**27,3696**	25,8396	28,7837
$N = 0,10$	26,9622	15,7974	27,3677	26,3107	29,0575
$N = 0,15$	27,3703	15,9568	27,3392	26,7064	29,2058
$N = 0,20$	27,6550	16,1044	27,2825	27,0026	**29,2491**
$N = 0,25$	27,8584	16,2538	27,2115	27,2369	29,2386
$N = 0,30$	27,9877	16,4032	27,1271	27,4100	29,1771
$N = 0,35$	28,0388	16,5482	27,0280	27,5157	29,0736
$N = 0,40$	**28,0465**	16,7010	26,9303	27,5802	28,9649
$N = 0,45$	28,0378	16,8598	26,8318	27,6267	28,8470
$N = 0,50$	27,9971	17,0321	26,7345	**27,6407**	28,7256
$N = 0,55$	27,9320	17,2167	26,6370	27,6276	28,5870
$N = 0,60$	27,8546	17,3952	26,5354	27,5879	28,4469
$N = 0,65$	27,7673	17,5762	26,4373	27,5380	28,3154
$N = 0,70$	27,6774	17,7581	26,3490	27,4869	28,1910
$N = 0,75$	27,5891	17,9468	26,2539	27,4240	28,0639
$N = 0,80$	27,4990	18,1290	26,1642	27,3584	27,9461
$N = 0,85$	27,4090	18,3464	26,0802	27,2886	27,8234
$N = 0,90$	27,3193	18,5563	25,9997	27,2171	27,7096
$N = 0,95$	27,2357	18,7825	25,9228	27,1451	27,6024
$N = 1,00$	27,1431	19,0096	25,8494	27,0709	27,4983
$N = 1,05$	27,0544	19,2142	25,7742	26,9944	27,3982
$N = 1,10$	26,9734	19,4324	25,7014	26,9184	27,3001
$N = 1,15$	26,8866	19,6314	25,6315	26,8437	27,2081
$N = 1,20$	26,7983	19,8287	25,5631	26,7691	27,1126
$N = 1,25$	26,7191	**20,0199**	25,4968	26,6940	27,0232

Tabela B.18: Resultados de PSNR do filtro baseado em *wavelets* proposto (imagem *Peppers*).

(a) Imagem original



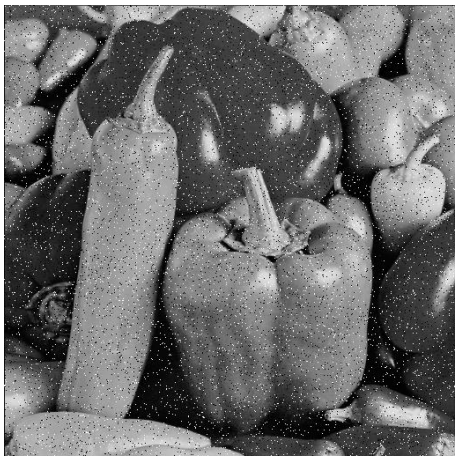
(b) Imagem corrompida



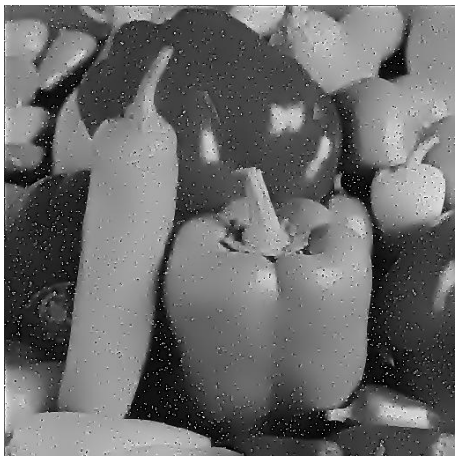
(c) Imagem filtrada

Figura B.104: Imagem *Peppers* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro baseado em *wavelets* proposto.

(a) Imagem original



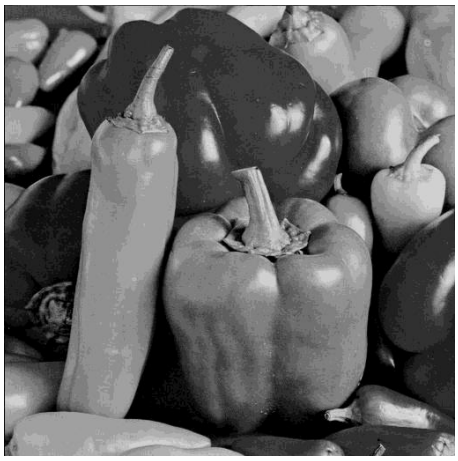
(b) Imagem corrompida



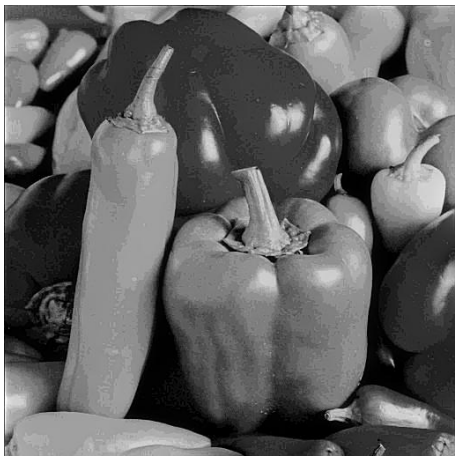
(c) Imagem filtrada

Figura B.105: Imagem *Peppers* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro baseado em *wavelets* proposto.

(a) Imagem original

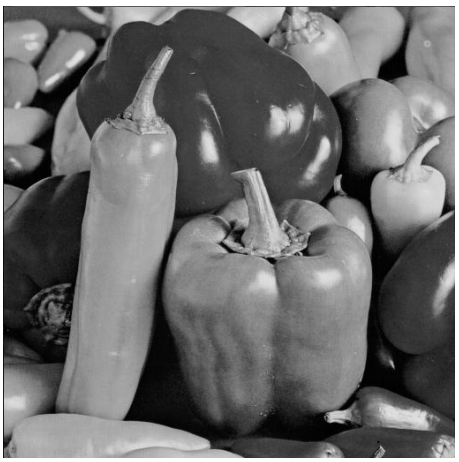


(b) Imagem corrompida

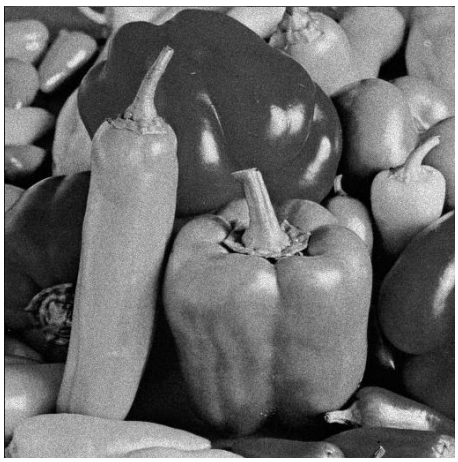


(c) Imagem filtrada

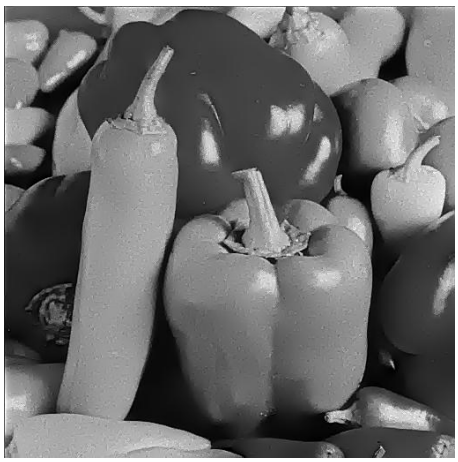
Figura B.106: Imagem *Peppers* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro baseado em *wavelets* proposto.



(a) Imagem original

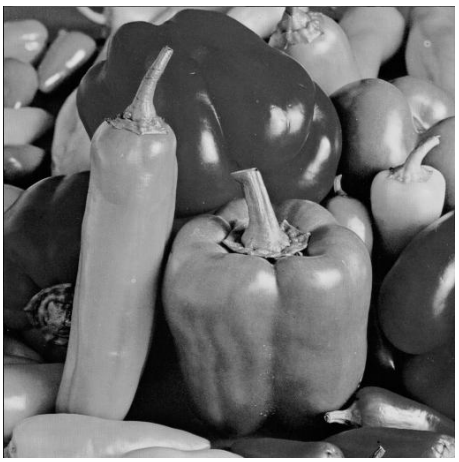


(b) Imagem corrompida

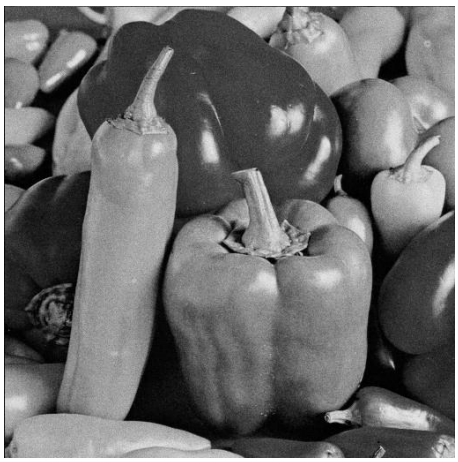


(c) Imagem filtrada

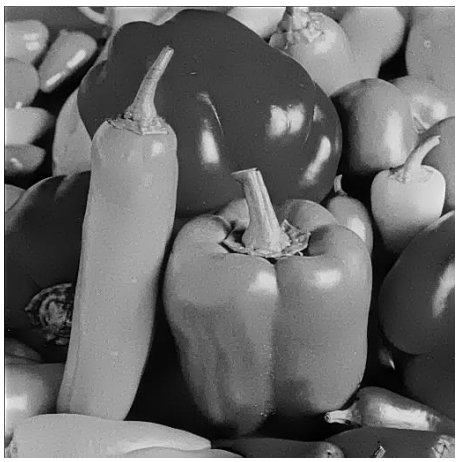
Figura B.107: Imagem *Peppers* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro baseado em *wavelets* proposto.



(a) Imagem original



(b) Imagem corrompida



(c) Imagem filtrada

Figura B.108: Imagem *Peppers* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro baseado em *wavelets* proposto.

B.10 Filtro morfológico

B.10.1 Imagem Moedas

(a) Imagem original

(b) Imagem filtrada ($H = 3$, $\Upsilon = 150$)

(c) Imagem filtrada ($H = 3$, $\Upsilon = 11500$)

(d) Imagem filtrada ($H = 5$, $\Upsilon = 150$)

(e) Imagem filtrada ($H = 5$, $\Upsilon = 11500$)

Figura B.109: Resultado da filtragem da imagem *Moedas* pelo filtro morfológico (disco de raio 2).

B.10.2 Imagem Máscaras

(a) Imagem original

(b) Imagem filtrada ($H = 10$, $\Upsilon = 2000$)

(c) Imagem filtrada ($H = 10$, $\Upsilon = 4000$)

(d) Imagem filtrada ($H = 12$, $\Upsilon = 2000$)

(e) Imagem filtrada ($H = 12$, $\Upsilon = 4000$)

Figura B.110: Resultado da filtragem da imagem *Máscaras* pelo filtro morfológico (disco de raio 2).

B.10.3 Imagem Lena

E	Υ	H	T	$S\&P$	Q	F	G
Disco de raio 1	149	3	28,8830	18,3589	29,6848	27,4052	33,1261
		5	29,2287	18,3933	29,6545	27,5716	33,8322
	2981	3	29,1436	18,4750	29,5829	27,5212	34,3984
		5	30,8215	18,7421	29,3987	27,9625	**34,5837**
	178482301	3	**32,6695**	**24,2671**	29,2915	31,9019	34,5365
		5	32,5612	24,2552	29,2341	31,8149	34,3263
Disco de raio 2	149	3	28,7036	18,3360	29,6198	27,3002	32,5185
		5	28,7395	18,3415	29,6142	27,3218	32,7096
	2981	3	28,7127	18,3491	**29,7433**	27,2975	32,5769
		5	28,7723	18,3689	29,6994	27,3331	33,0805
	178482301	3	31,4902	23,7005	28,3458	31,1707	32,2450
		5	31,3889	23,6797	28,2883	31,0844	32,1367
Quadrado de lado 2	149	3	29,0585	18,3550	29,6644	27,4784	33,5198
		5	29,5769	18,3938	29,5980	27,7244	33,9001
	2981	3	29,9928	18,4697	29,4812	27,7403	34,4867
		5	31,8242	18,8329	29,4314	28,6342	34,1794
	178482301	3	32,1851	24,2041	29,4812	31,4193	34,4698
		5	32,0139	24,1905	29,4314	31,2985	34,1794
Quadrado de lado 3	149	3	28,7008	18,3342	29,6465	27,2988	32,5771
		5	28,7678	18,3393	29,6502	27,3347	32,9611
	2981	3	28,7050	18,3440	29,6859	27,3032	32,7552
		5	28,8206	18,3627	29,5931	27,3818	33,9630
	178482301	3	32,4663	23,8633	28,9175	**32,0226**	33,7435
		5	32,3614	23,8440	28,8559	31,9314	33,5989

Tabela B.19: Resultados de PSNR do filtro morfológico (imagem *Lena*).

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.111: Imagem *Lena* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro morfológico.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.112: Imagem *Lena* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro morfológico.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.113: Imagem *Lena* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro morfológico.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.114: Imagem *Lena* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro morfológico.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

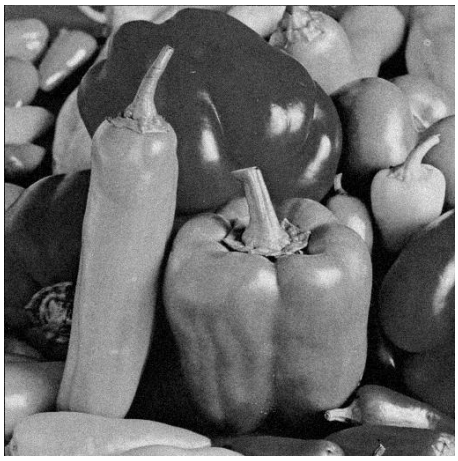
Figura B.115: Imagem *Lena* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro morfológico.

B.10.4 Imagem Peppers

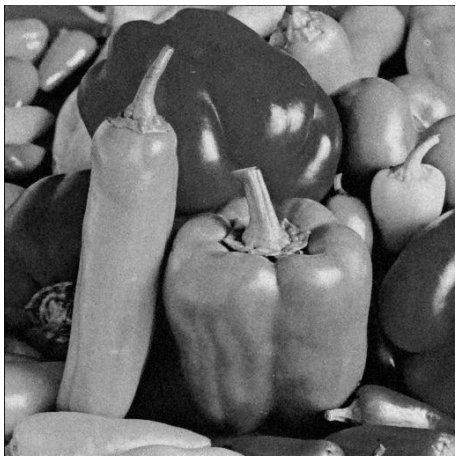
E	Υ	H	T	$S\&P$	Q	F	G
Disco de raio 1	149	3	29,0442	18,0295	**29,8528**	28,2986	33,6745
		5	29,3956	18,0538	29,8236	28,5104	34,3847
	2981	3	29,3459	18,1618	29,6647	28,4815	34,9794
		5	31,0227	18,4791	29,6073	29,0003	**35,1106**
	178482301	3	32,0685	23,8038	29,3076	31,9088	33,8351
		5	31,9740	23,7941	29,2599	31,8253	33,6781
Disco de raio 2	149	3	28,8474	18,0129	29,7981	28,1590	33,0467
		5	28,8849	18,0166	29,7833	28,1886	33,2304
	2981	3	28,8588	18,0319	29,7267	28,1592	33,0479
		5	28,9194	18,0660	29,5884	28,2131	33,5061
	178482301	3	31,6423	23,2543	28,8163	31,6247	32,6922
		5	31,5477	23,2366	28,7664	31,5492	32,6085
Quadrado de lado 2	149	3	29,2240	18,0281	29,8185	28,3970	34,0567
		5	29,7400	18,0532	29,7667	28,6479	34,4309
	2981	3	30,3058	18,1480	29,7386	28,6939	35,0696
		5	32,0013	18,5222	29,6939	29,4758	34,7780
	178482301	3	**32,3706**	**23,8653**	29,7386	**32,1996**	35,0696
		5	32,1937	23,8529	29,6939	32,0620	34,7780
Quadrado de lado 3	149	3	28,8506	18,0112	29,8191	28,1684	33,1170
		5	28,9185	18,0155	29,8107	28,2243	33,4726
	2981	3	28,8596	18,0225	29,7550	28,1772	33,1987
		5	28,9967	18,0570	29,6173	28,3066	34,5033
	178482301	3	32,1647	23,4041	29,1374	32,0898	33,4892
		5	32,0635	23,3874	29,0837	32,0023	33,3695

Tabela B.20: Resultados de PSNR do filtro morfológico (imagem *Peppers*).

(a) Imagem original



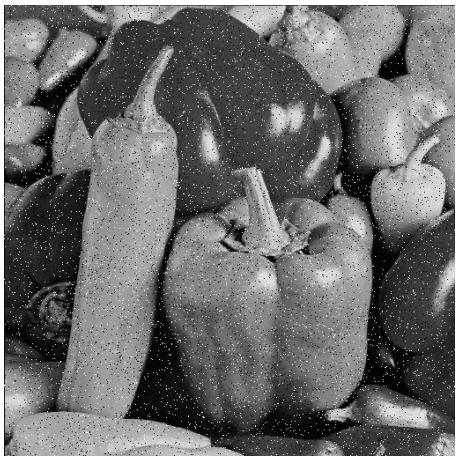
(b) Imagem corrompida



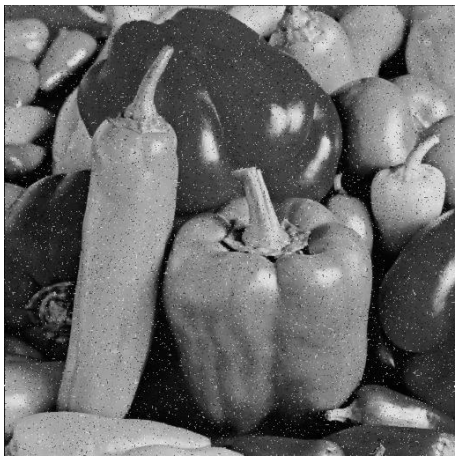
(c) Imagem filtrada

Figura B.116: Imagem *Peppers* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro morfológico.

(a) Imagem original



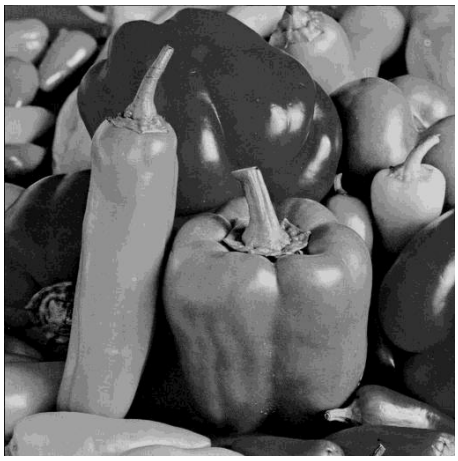
(b) Imagem corrompida



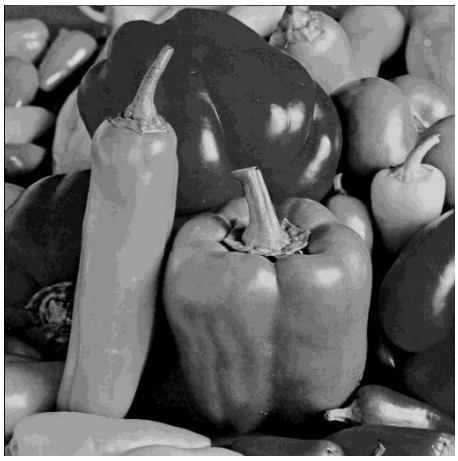
(c) Imagem filtrada

Figura B.117: Imagem *Peppers* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro morfológico.

(a) Imagem original

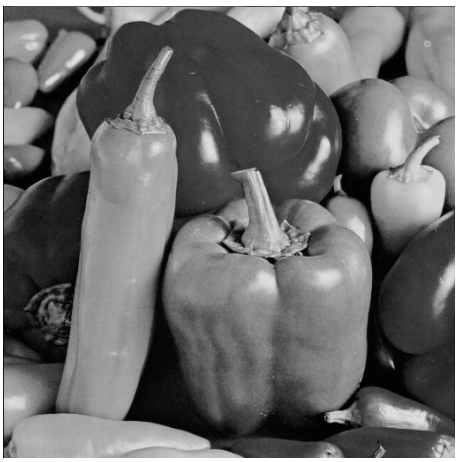


(b) Imagem corrompida

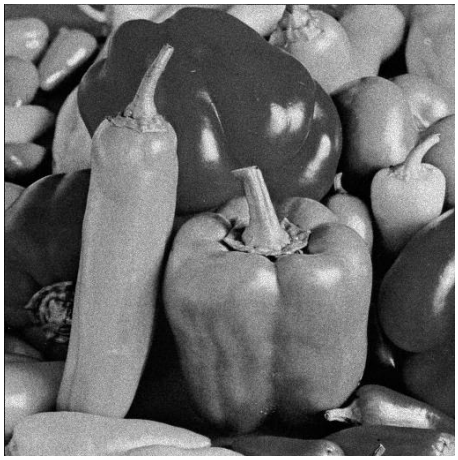


(c) Imagem filtrada

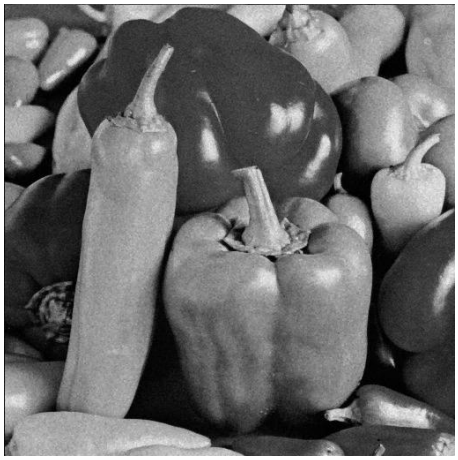
Figura B.118: Imagem *Peppers* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro morfológico.



(a) Imagem original

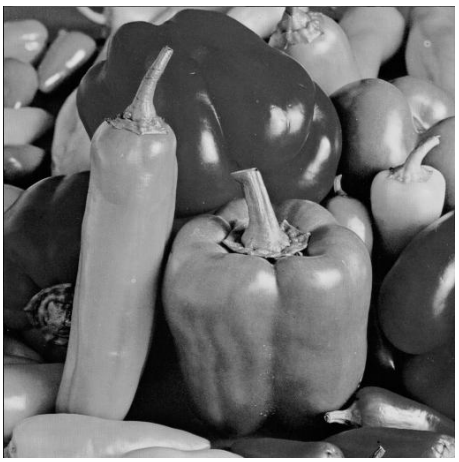


(b) Imagem corrompida

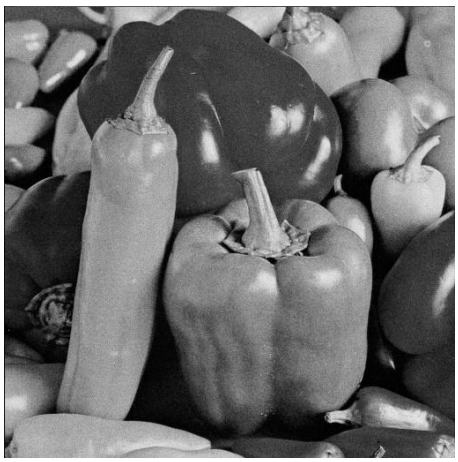


(c) Imagem filtrada

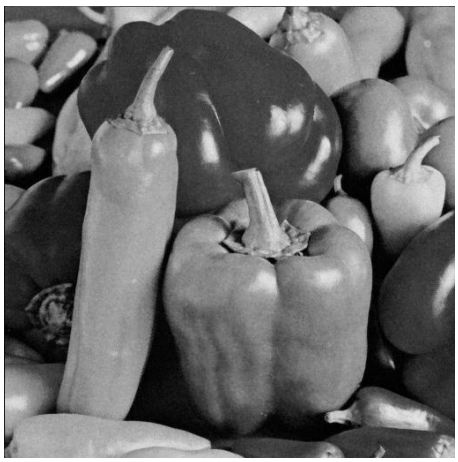
Figura B.119: Imagem *Peppers* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro morfológico.



(a) Imagem original



(b) Imagem corrompida



(c) Imagem filtrada

Figura B.120: Imagem *Peppers* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro morfológico.

B.11 Filtro morfológico II

B.11.1 Imagem Moedas

(a) Imagem original

(b) Imagem filtrada ($H = 3$, $\Upsilon = 150$)

(c) Imagem filtrada ($H = 3$, $\Upsilon = 11500$)

(d) Imagem filtrada ($H = 5$, $\Upsilon = 150$)

(e) Imagem filtrada ($H = 5$, $\Upsilon = 11500$)

Figura B.121: Resultado da filtragem da imagem *Moedas* pelo filtro morfológico II (disco de raio 2).

B.11.2 Imagem Máscaras

(a) Imagem original

(b) Imagem filtrada ($H = 10$, $\Upsilon = 2000$)

(c) Imagem filtrada ($H = 10$, $\Upsilon = 4000$)

(d) Imagem filtrada ($H = 12$, $\Upsilon = 2000$)

(e) Imagem filtrada ($H = 12$, $\Upsilon = 4000$)

Figura B.122: Resultado da filtragem da imagem *Máscaras* pelo filtro morfológico II (disco de raio 2).

B.11.3 Imagem Lena

E	Υ	H	T	$S\&P$	Q	F	G
Disco de raio 1	149	3	28,9089	18,3738	29,4535	27,3986	33,0237
		5	29,2447	18,4186	29,3467	27,5539	33,4104
	2981	3	29,0165	18,5532	28,5804	27,2915	32,5807
		5	29,8710	18,9945	28,2125	27,4941	31,7444
	178482301	3	29,8449	28,9693	27,9244	29,2535	31,1391
		5	29,7695	28,9002	27,9071	29,1749	31,0635
Disco de raio 2	149	3	28,7086	18,3423	29,4829	27,3029	32,4850
		5	28,7387	18,3503	29,3956	27,3150	32,5426
	2981	3	28,7124	18,3744	29,1124	27,2631	32,2886
		5	28,7189	18,4065	28,7047	27,2089	31,7630
	178482301	3	25,6398	24,3062	25,3359	25,1536	26,5482
		5	25,4943	24,1790	25,3174	25,0221	26,4716
Quadrado de lado 2	149	3	29,2327	18,3684	29,4323	27,5409	33,5282
		5	29,7913	18,4229	29,3083	27,8276	**33,5643**
	2981	3	30,3478	18,5550	28,8238	27,6539	33,0424
		5	31,0450	19,1786	28,7969	28,5713	32,7611
	178482301	3	**31,2561**	**30,0277**	28,8238	**30,4906**	33,0122
		5	31,0600	29,9534	28,7969	30,3466	32,7611
Quadrado de lado 3	149	3	28,7127	18,3395	**29,4906**	27,3009	32,5570
		5	28,7838	18,3467	29,4012	27,3223	32,7632
	2981	3	28,6912	18,3632	28,9527	27,2797	32,2593
		5	28,7272	18,3953	28,5190	27,2154	31,7566
	178482301	3	27,5506	25,6805	26,7888	26,9298	28,8074
		5	27,4125	25,5482	26,7739	26,7948	28,7537

Tabela B.21: Resultados de PSNR do filtro morfológico II (imagem *Lena*).

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.123: Imagem *Lena* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro morfológico II.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.124: Imagem *Lena* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro morfológico II.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.125: Imagem *Lena* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro morfológico II.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.126: Imagem *Lena* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro morfológico II.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

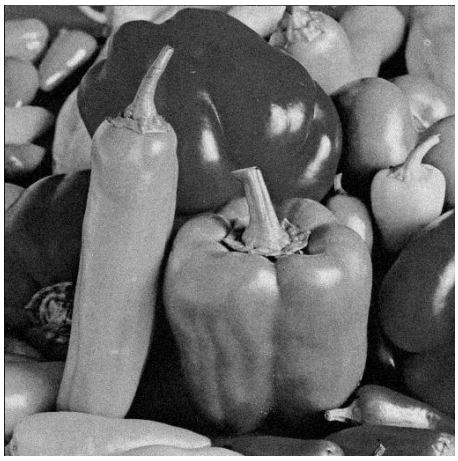
Figura B.127: Imagem *Lena* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro morfológico II.

B.11.4 Imagem Peppers

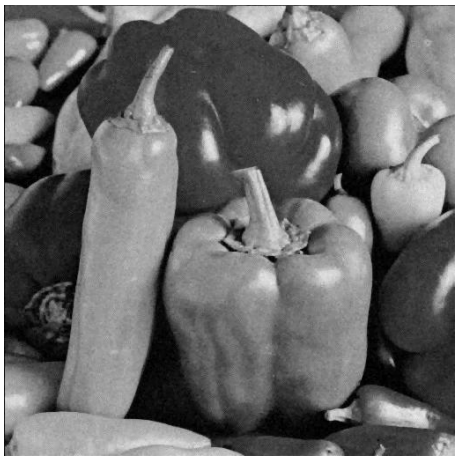
E	Υ	H	T	$S\&P$	Q	F	G
Disco de raio 1	149	3	29,0525	18,0389	29,4934	28,2586	33,3680
		5	29,3867	18,0713	29,4146	28,4419	33,6934
	2981	3	29,1227	18,2443	28,6306	28,1924	32,4959
		5	30,0052	18,6760	28,5832	28,4440	32,1795
	178482301	3	28,6937	27,8355	27,4054	28,6248	29,6795
		5	28,6383	27,7883	27,4082	28,5728	29,6426
Disco de raio 2	149	3	28,8496	18,0180	**29,5675**	28,1395	32,9220
		5	28,8736	18,0241	29,4089	28,1351	32,8816
	2981	3	28,8491	18,0570	28,6617	28,0840	32,5271
		5	28,8161	18,1100	28,1628	27,9685	31,9308
	178482301	3	25,9181	24,0378	25,9398	25,7124	27,0593
		5	25,7738	23,9225	25,9439	25,6007	27,0066
Quadrado de lado 2	149	3	29,4049	18,0383	29,5493	28,4505	33,9614
		5	29,9635	18,0739	29,4708	28,7336	**34,0093**
	2981	3	30,6342	18,2185	29,2094	28,6460	33,7591
		5	31,3877	18,8270	29,1986	29,4666	33,5144
	178482301	3	**31,6266**	**30,3444**	29,2094	**31,3249**	33,7591
		5	31,4169	30,2668	29,1986	31,1643	33,5144
Quadrado de lado 3	149	3	28,8573	18,0152	29,5633	28,1467	32,9786
		5	28,9150	18,0213	29,4487	28,1725	33,0621
	2981	3	28,8339	18,0364	28,6782	28,0903	32,3830
		5	28,8474	18,0864	28,3471	28,0577	31,8300
	178482301	3	27,1561	25,4325	26,7929	26,8568	28,3885
		5	27,0331	25,3115	26,8002	26,7567	28,3598

Tabela B.22: Resultados de PSNR do filtro morfológico II (imagem *Peppers*).

(a) Imagem original



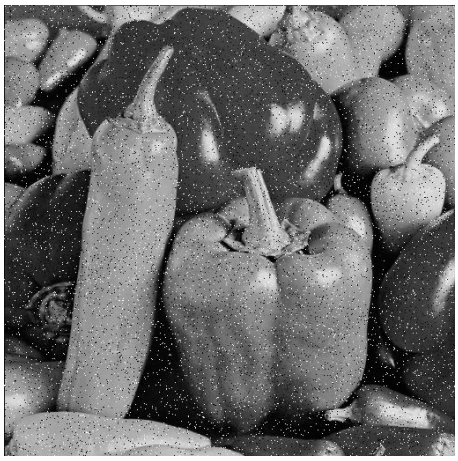
(b) Imagem corrompida



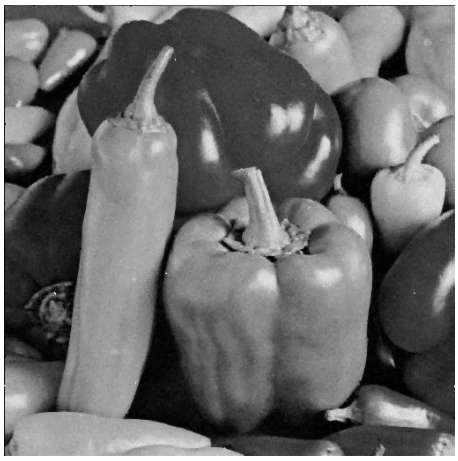
(c) Imagem filtrada

Figura B.128: Imagem *Peppers* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro morfológico II.

(a) Imagem original



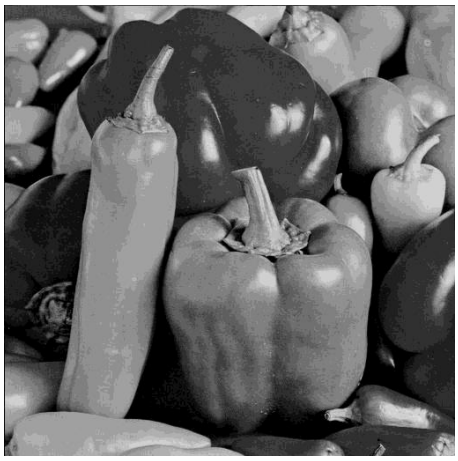
(b) Imagem corrompida



(c) Imagem filtrada

Figura B.129: Imagem *Peppers* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro morfológico II.

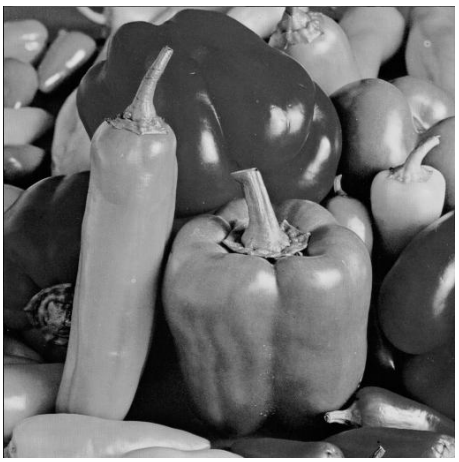
(a) Imagem original



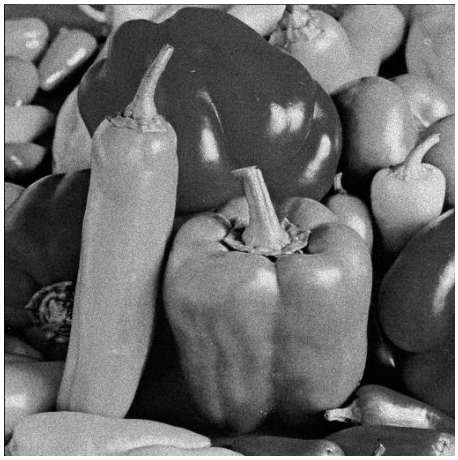
(b) Imagem corrompida

(c) Imagem filtrada

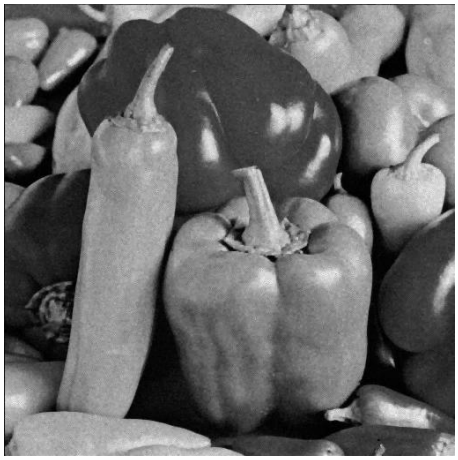
Figura B.130: Imagem *Peppers* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro morfológico II.



(a) Imagem original

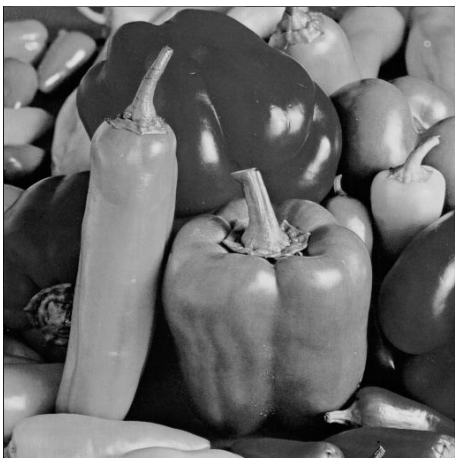


(b) Imagem corrompida

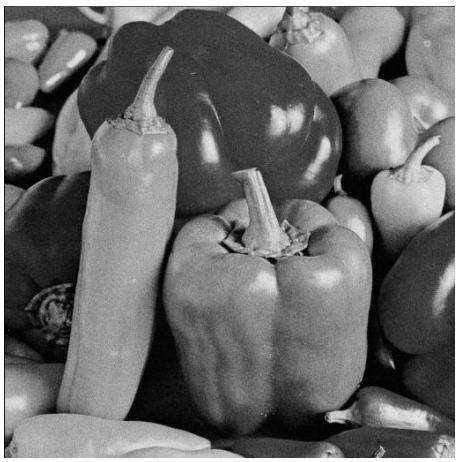


(c) Imagem filtrada

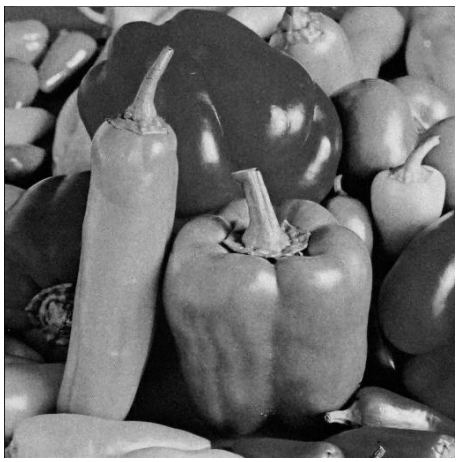
Figura B.131: Imagem *Peppers* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro morfológico II.



(a) Imagem original



(b) Imagem corrompida



(c) Imagem filtrada

Figura B.132: Imagem *Peppers* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro morfológico II.

B.12 Filtro morfológico III

B.12.1 Imagem Moedas

(a) Imagem original

(b) Imagem filtrada ($H = 3$, $\Upsilon = 150$)

(c) Imagem filtrada ($H = 3$, $\Upsilon = 11500$)

(d) Imagem filtrada ($H = 5$, $\Upsilon = 150$)

(e) Imagem filtrada ($H = 5$, $\Upsilon = 11500$)

Figura B.133: Resultado da filtragem da imagem *Moedas* pelo filtro morfológico III (disco de raio 2).

B.12.2 Imagem Máscaras

(a) Imagem original

(b) Imagem filtrada ($H = 10$, $\Upsilon = 2000$)

(c) Imagem filtrada ($H = 10$, $\Upsilon = 4000$)

(d) Imagem filtrada ($H = 12$, $\Upsilon = 2000$)

(e) Imagem filtrada ($H = 12$, $\Upsilon = 4000$)

Figura B.134: Resultado da filtragem da imagem *Máscaras* pelo filtro morfológico III (disco de raio 2).

B.12.3 Imagem Lena

E	Υ	H	T	$S\&P$	Q	F	G
Disco de raio 1	149	3	28,7118	18,3403	29,6606	27,3057	32,6625
		5	28,8377	18,3581	29,6140	27,3755	33,2707
	2981	3	28,7300	18,3628	29,6040	27,3153	32,9431
		5	28,9933	18,4484	29,3837	27,4631	**34,5753**
	178482301	3	31,9780	31,9868	28,4620	31,6132	32,7815
		5	31,8404	31,9107	28,4281	31,4937	32,6183
Disco de raio 2	149	3	28,6878	18,3325	29,6151	27,2896	32,4773
		5	28,7101	18,3354	29,6145	27,3054	32,6622
	2981	3	28,6883	18,3365	**29,7033**	27,2904	32,4970
		5	28,7265	18,3477	29,6302	27,3142	32,9332
	178482301	3	29,5327	28,9877	27,0090	29,4111	29,7827
		5	29,4908	28,9496	26,9918	29,3777	29,7662
Quadrado de lado 2	149	3	28,7340	18,3368	29,6213	27,3189	32,8584
		5	28,9860	18,3557	29,5123	27,4241	33,4959
	2981	3	28,7572	18,3524	29,4334	27,3553	33,5907
		5	29,5834	18,4684	29,1705	27,5756	34,1111
	178482301	3	**32,4112**	**33,2009**	28,9986	**32,0056**	33,7450
		5	32,1273	33,0702	28,9617	31,7823	33,4102
Quadrado de lado 3	149	3	28,6831	18,3317	29,6313	27,2857	32,4896
		5	28,7209	18,3346	29,6259	27,3100	32,8390
	2981	3	28,6839	18,3341	29,6908	27,2858	32,5484
		5	28,7325	18,3444	29,5313	27,3317	33,5239
	178482301	3	30,8933	30,3285	27,8072	30,7155	31,3867
		5	30,8271	30,2819	27,7788	30,6602	31,3340

Tabela B.23: Resultados de PSNR do filtro morfológico III (imagem *Lena*).

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.135: Imagem *Lena* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro morfológico III.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.136: Imagem *Lena* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro morfológico III.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.137: Imagem *Lena* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro morfológico III.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.138: Imagem *Lena* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro morfológico III.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

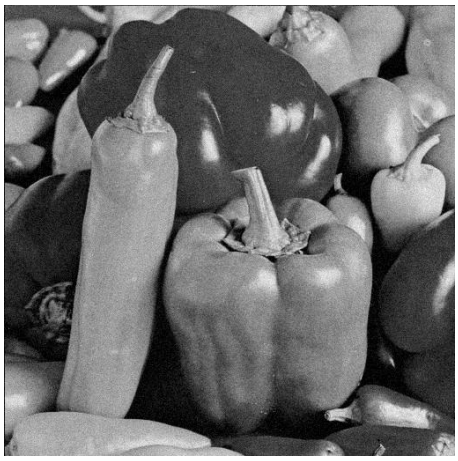
Figura B.139: Imagem *Lena* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro morfológico III.

B.12.4 Imagem Peppers

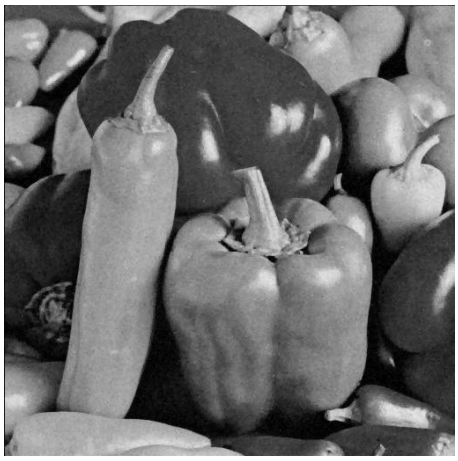
E	Υ	H	T	$S\&P$	Q	F	G
Disco de raio 1	149	3	28,8569	18,0142	**29,7816**	28,1726	33,1794
		5	28,9970	18,0252	29,7493	28,2685	33,7751
	2981	3	28,8821	18,0397	29,5915	28,1911	33,2267
		5	29,1624	18,1842	29,3583	28,3827	34,6085
	178482301	3	30,2370	30,4763	27,8811	30,3431	30,7624
		5	30,1392	30,4220	27,8619	30,2578	30,6710
Disco de raio 2	149	3	28,8311	18,0100	29,7574	28,1491	33,0098
		5	28,8589	18,0122	29,7494	28,1730	33,1861
	2981	3	28,8327	18,0164	29,7260	28,1429	32,9967
		5	28,8816	18,0395	29,4849	28,1788	33,2983
	178482301	3	29,0901	28,4279	27,1257	29,1624	29,4047
		5	29,0479	28,3916	27,1143	29,1347	29,3963
Quadrado de lado 2	149	3	28,8972	18,0147	29,7681	28,2057	33,3954
		5	29,1569	18,0288	29,6887	28,3395	34,0330
	2981	3	28,9377	18,0401	29,5218	28,2511	34,0735
		5	29,7856	18,2023	29,4056	28,5580	34,5034
	178482301	3	**32,8965**	**33,7602**	29,4172	**32,9418**	**34,6502**
		5	32,5817	33,6073	29,3886	32,6837	34,2989
Quadrado de lado 3	149	3	28,8328	18,0094	29,7748	28,1521	33,0401
		5	28,8751	18,0124	29,7754	28,1910	33,3637
	2981	3	28,8361	18,0149	29,6663	28,1545	33,0020
		5	28,9017	18,0422	29,4535	28,2188	33,8117
	178482301	3	29,6366	29,2526	27,5246	29,6950	30,0655
		5	29,5786	29,2136	27,5074	29,6503	30,0301

Tabela B.24: Resultados de PSNR do filtro morfológico III (imagem *Peppers*).

(a) Imagem original



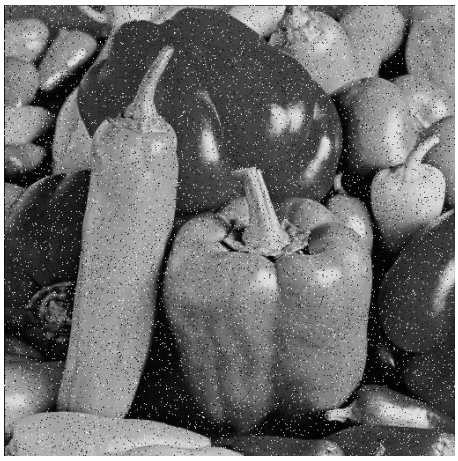
(b) Imagem corrompida



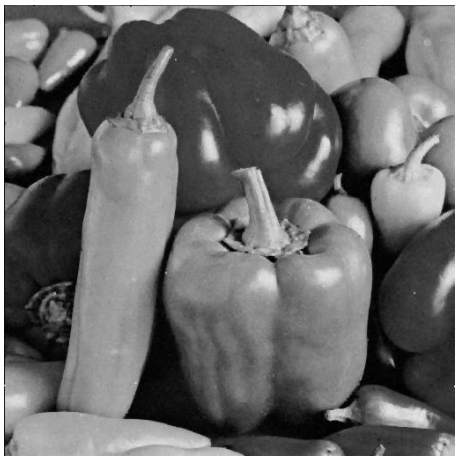
(c) Imagem filtrada

Figura B.140: Imagem *Peppers* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro morfológico III.

(a) Imagem original



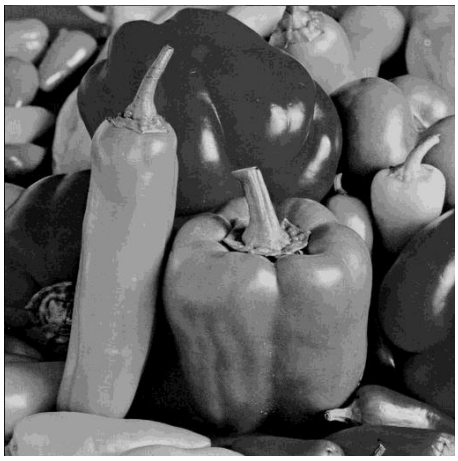
(b) Imagem corrompida



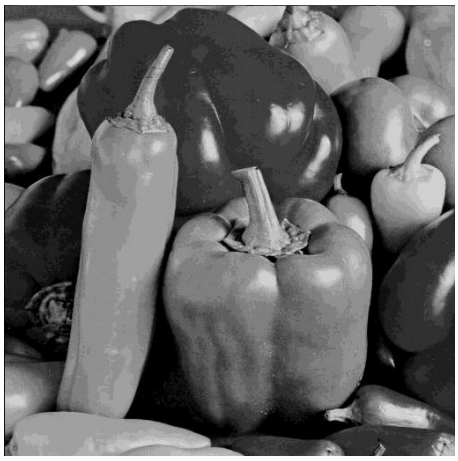
(c) Imagem filtrada

Figura B.141: Imagem *Peppers* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro morfológico III.

(a) Imagem original

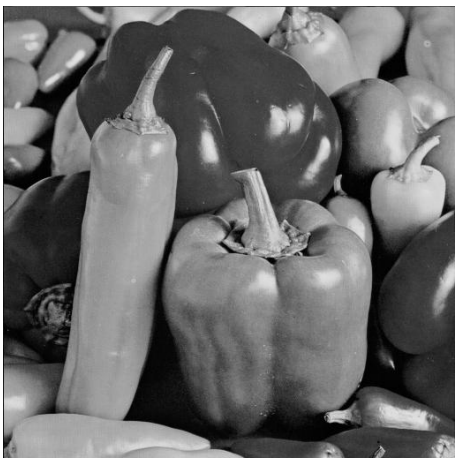


(b) Imagem corrompida

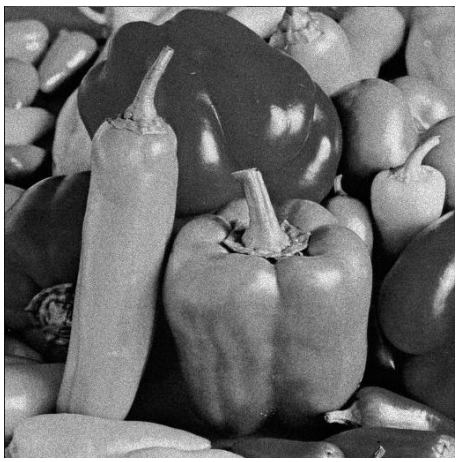


(c) Imagem filtrada

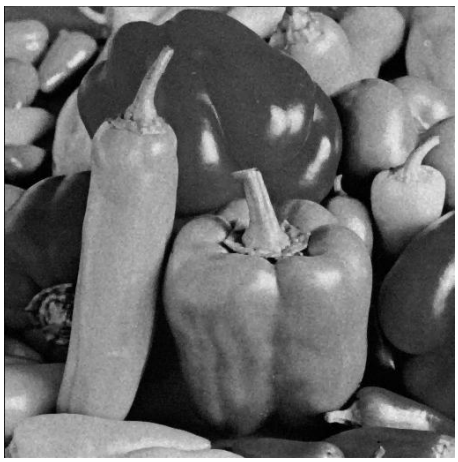
Figura B.142: Imagem *Peppers* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro morfológico III.



(a) Imagem original

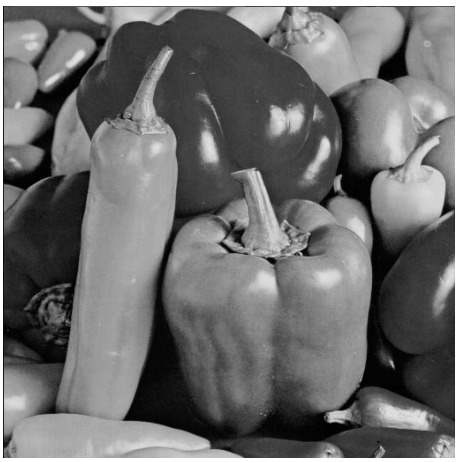


(b) Imagem corrompida

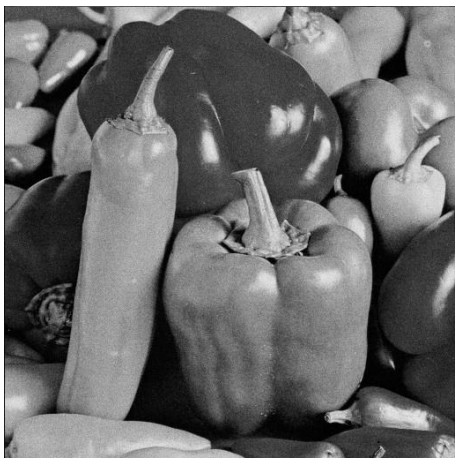


(c) Imagem filtrada

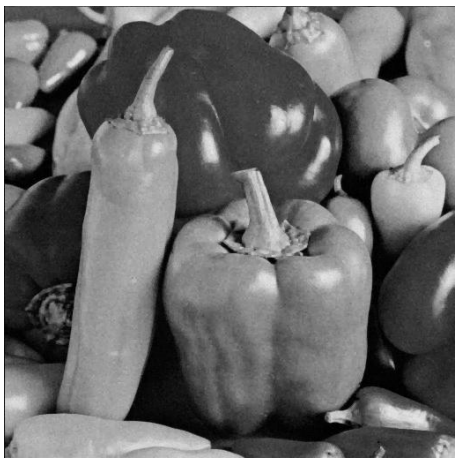
Figura B.143: Imagem *Peppers* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro morfológico III.



(a) Imagem original



(b) Imagem corrompida



(c) Imagem filtrada

Figura B.144: Imagem *Peppers* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro morfológico III.

B.13 Filtro morfológico IV

B.13.1 Imagem Moedas

(a) Imagem original

(b) Imagem filtrada ($H = 3$, $\Upsilon = 150$)

(c) Imagem filtrada ($H = 3$, $\Upsilon = 11500$)

(d) Imagem filtrada ($H = 5$, $\Upsilon = 150$)

(e) Imagem filtrada ($H = 5$, $\Upsilon = 11500$)

Figura B.145: Resultado da filtragem da imagem *Moedas* pelo filtro morfológico IV (disco de raio 2).

B.13.2 Imagem Máscaras

(a) Imagem original

(b) Imagem filtrada ($H = 10$, $\Upsilon = 2000$)

(c) Imagem filtrada ($H = 10$, $\Upsilon = 4000$)

(d) Imagem filtrada ($H = 12$, $\Upsilon = 2000$)

(e) Imagem filtrada ($H = 12$, $\Upsilon = 4000$)

Figura B.146: Resultado da filtragem da imagem *Máscaras* pelo filtro morfológico IV (disco de raio 2).

B.13.3 Imagem Lena

E	Υ	H	T	$S\&P$	Q	F	G
Disco de raio 1	149	3	30,6911	18,6035	29,5416	28,4287	**34,8339**
		5	31,8584	18,9354	29,4352	29,5026	34,4244
	2981	3	32,1050	22,8461	28,5504	30,7191	32,8076
		5	31,5749	30,6858	28,4521	31,2961	32,3313
	178482301	3	31,8844	31,9206	28,4587	31,5458	32,6405
		5	31,5164	31,6273	28,4125	31,2526	32,3014
Disco de raio 2	149	3	29,8663	18,5911	29,5363	28,0219	34,2923
		5	30,8515	18,8011	29,4963	28,7106	34,5701
	2981	3	31,6257	20,8338	28,8601	29,4521	32,7519
		5	31,0894	25,5683	28,3392	30,5763	31,7237
	178482301	3	29,5081	28,9863	26,9962	29,3942	29,7375
		5	29,3830	28,9178	26,9680	29,2971	29,6046
Quadrado de lado 2	149	3	31,0017	18,5802	29,4655	28,6609	34,4791
		5	31,6977	18,9096	29,3525	29,6367	33,9438
	2981	3	**32,3198**	23,1028	28,9989	31,3733	33,6220
		5	31,8337	32,4365	28,9595	31,5509	33,1840
	178482301	3	32,2975	**33,0986**	28,9989	**31,9305**	33,6220
		5	31,8337	32,6926	28,9595	31,5509	33,1840
Quadrado de lado 3	149	3	30,0619	18,5364	**29,5621**	28,0645	34,6216
		5	31,1899	18,7450	29,4890	28,8228	34,5988
	2981	3	32,1599	20,1640	28,6775	29,4963	32,6766
		5	31,2853	26,1212	28,3398	30,8791	31,7236
	178482301	3	30,8469	30,3221	27,7983	30,6854	31,3000
		5	30,6212	30,2093	27,7619	30,5145	31,0675

Tabela B.25: Resultados de PSNR do filtro morfológico IV (imagem *Lena*).

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.147: Imagem *Lena* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro morfológico IV.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.148: Imagem *Lena* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro morfológico IV.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.149: Imagem *Lena* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro morfológico IV.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

Figura B.150: Imagem *Lena* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro morfológico IV.

(a) Imagem original

(b) Imagem corrompida

(c) Imagem filtrada

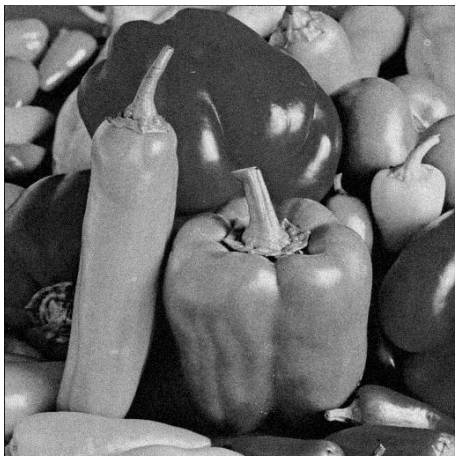
Figura B.151: Imagem *Lena* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro morfológico IV.

B.13.4 Imagem Peppers

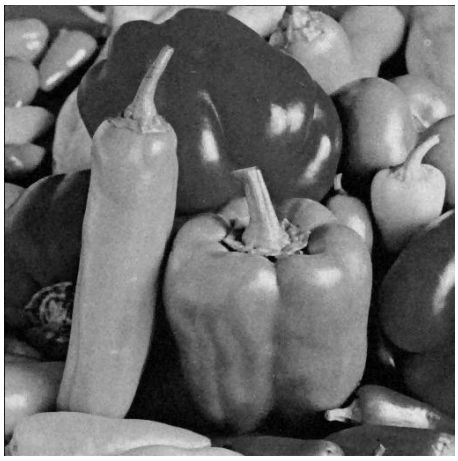
E	Υ	H	T	$S\&P$	Q	F	G
Disco de raio 1	149	3	30,8992	18,2199	29,6604	29,4194	**35,3297**
		5	32,1084	18,4763	29,5713	30,4195	34,9167
	2981	3	32,6157	23,3928	29,0933	31,8919	33,8456
		5	32,0369	30,8056	29,0464	32,2513	33,3993
	178482301	3	30,1744	30,4308	27,8775	30,2933	30,6858
		5	29,9194	30,2136	27,8489	30,0887	30,4866
Disco de raio 2	149	3	30,0648	18,2016	29,6776	28,9703	34,7780
		5	31,1050	18,3663	29,6211	29,6083	35,0650
	2981	3	31,9848	20,9851	28,8050	30,2353	32,9224
		5	31,7490	26,8483	28,5326	31,5548	32,4239
	178482301	3	29,0729	28,4317	27,1129	29,1475	29,3721
		5	28,9523	28,3688	27,0950	29,0542	29,2636
Quadrado de lado 2	149	3	31,2184	18,2045	29,6253	29,5914	35,0262
		5	31,8985	18,4542	29,5697	30,5600	34,5458
	2981	3	**32,7742**	23,0625	29,4196	32,3225	34,5195
		5	32,2607	32,9689	29,3847	32,4427	34,0466
	178482301	3	32,7742	**33,6442**	29,4196	**32,8559**	34,5195
		5	32,2607	33,1822	29,3847	32,4427	34,0466
Quadrado de lado 3	149	3	30,2624	18,1643	**29,6789**	29,0311	35,0122
		5	31,4041	18,3375	29,6073	29,7306	35,0548
	2981	3	32,4934	20,4987	28,8668	30,5048	33,4516
		5	31,9810	27,1472	28,7658	32,1110	32,8050
	178482301	3	29,6001	29,2489	27,5160	29,6661	30,0089
		5	29,4194	29,1580	27,4882	29,5327	29,8505

Tabela B.26: Resultados de PSNR do filtro morfológico IV (imagem *Peppers*).

(a) Imagem original



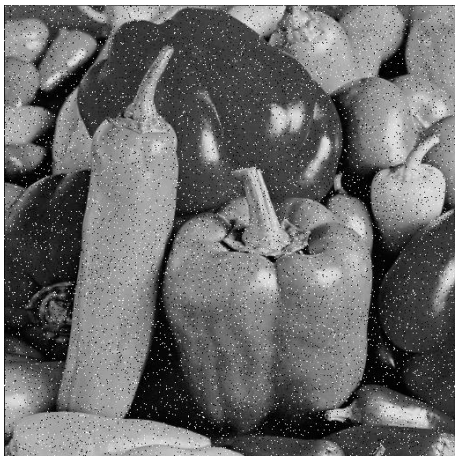
(b) Imagem corrompida



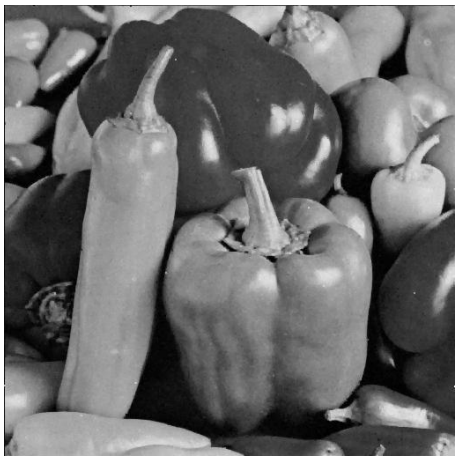
(c) Imagem filtrada

Figura B.152: Imagem *Peppers* corrompida por ruído térmico ($\overline{T} = 0$ e $\sigma_T^2 = 10$) filtrada pelo filtro morfológico IV.

(a) Imagem original



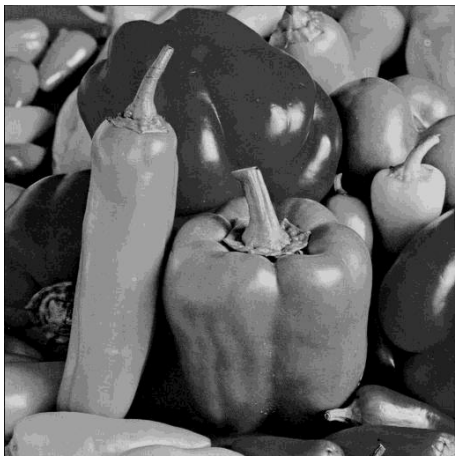
(b) Imagem corrompida



(c) Imagem filtrada

Figura B.153: Imagem *Peppers* corrompida por ruído do tipo "sal e pimenta" ($\epsilon = 0.05$) filtrada pelo filtro morfológico IV.

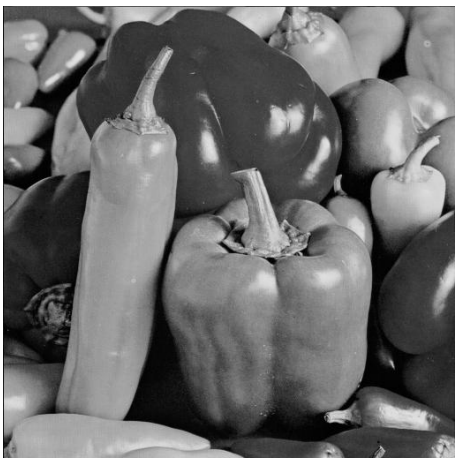
(a) Imagem original



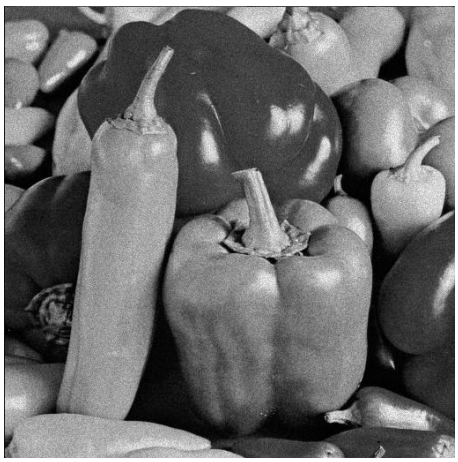
(b) Imagem corrompida

(c) Imagem filtrada

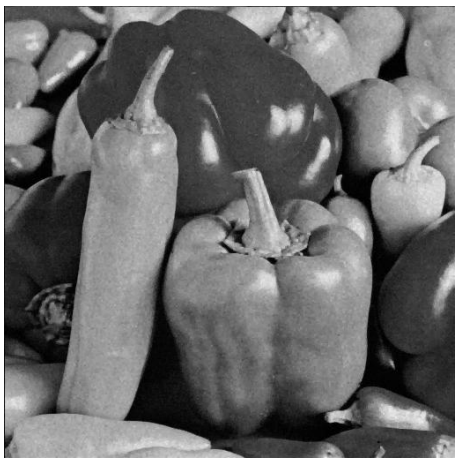
Figura B.154: Imagem *Peppers* corrompida por ruído de quantização ($\Delta = 16$) filtrada pelo filtro morfológico IV.



(a) Imagem original

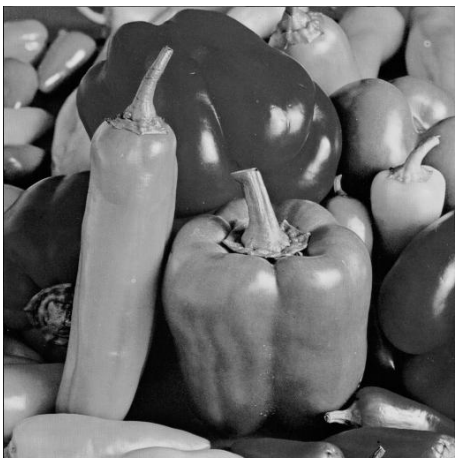


(b) Imagem corrompida

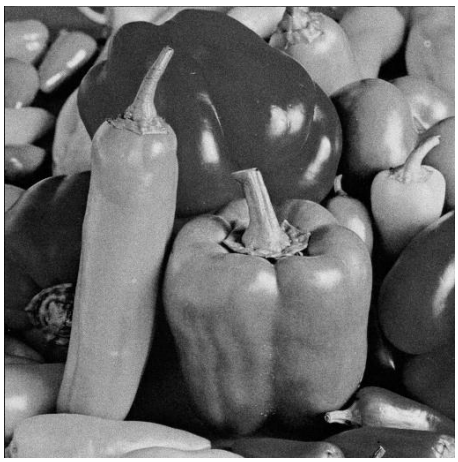


(c) Imagem filtrada

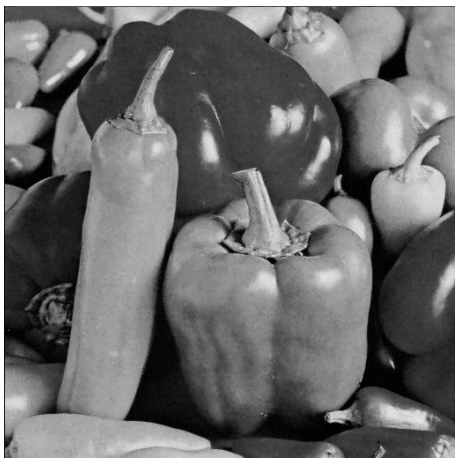
Figura B.155: Imagem *Peppers* corrompida por ruído relativo à contagem de fótons (para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$) filtrada pelo filtro morfológico IV.



(a) Imagem original



(b) Imagem corrompida



(c) Imagem filtrada

Figura B.156: Imagem *Peppers* corrompida por ruído relativo à granulação em fotografias (para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$) filtrada pelo filtro morfológico IV.

Apêndice C

Códigos

C.1 Filtro mediana

```
function filteredImage = IDMedianFilter(originalImage,
    windowSize)

%%%%%%%%%%%%%%%%%%%%
% S E T T I N G S %
%%%%%%%%%%%%%%%%%%%%

global OK;
global IMAGE_NUMBER;
global METHOD_NAME;
fileMethodName = METHOD_NAME;
fileMethodName(findstr(METHOD_NAME, ' ') + 1) = upper(
    METHOD_NAME(findstr(METHOD_NAME, ' ') + 1));
fileMethodName(1) = upper(fileMethodName(1));

%%%%%%%%%%%%%%%%
% F I L T E R %
%%%%%%%%%%%%%%%%

filter = @(x) median(x(:));
```

```
filteredImage = myNlFilter(originalImage, windowSize, filter
    );

%%%%%%%%%%%%%%%%%
% D I S P L A Y %
%%%%%%%%%%%%%%%%%

if OK
    OK = 0;
    return;
end

figure;
imshow(filteredImage, []);
set(gcf, 'numbertitle', 'off', 'name', ['Filtragem da imagem
    ' num2str(IMAGE_NUMBER) ' por ' METHOD_NAME ' com janela
    de ' num2str(windowSize(1)) ' x ' num2str(windowSize(2))
    ]);
truesize;

%%%%%%%%%%
% S A V E %
%%%%%%%%%%

imwrite((filteredImage - min(min(filteredImage))) / max(max(
    filteredImage - min(min(filteredImage)))), ['Resultados/'
    num2str(IMAGE_NUMBER) '/filtrada' strrep(fileMethodName,
    ' ', '') 'Janela' num2str(windowSize(1)) 'x' num2str(
    windowSize(2)) '.jpg']);

%%%%%%%%%%%%%
% C L E A R %
```

```
%%%%%%%%%%%%%

clear OK;
clear IMAGE_NUMBER;
clear METHOD_NAME;
```

C.2 Filtro média

```
function filteredImage = IDMeanFilter(originalImage,
    windowSize)

%%%%%%%%%%%%%%%%%%%%%
% S E T T I N G S %
%%%%%%%%%%%%%%%%%%%%%

global OK;
global IMAGE_NUMBER;
global METHOD_NAME;
fileMethodName = METHOD_NAME;
fileMethodName(findstr(METHOD_NAME, ' ') + 1) = upper(
    METHOD_NAME(findstr(METHOD_NAME, ' ') + 1));
fileMethodName(1) = upper(fileMethodName(1));

%%%%%%%%%%%%%%%%%
% F I L T E R %
%%%%%%%%%%%%%%%%%

filter = @(x) round(mean(x(:)));
filteredImage = myNlFilter(originalImage, windowSize, filter
    );

%%%%%%%%%%%%%%%%%%%
% D I S P L A Y %
```

```
%%%%%%%%%%%%%%%%%%

if OK
    OK = 0;
    return;
end

figure;
imshow(filteredImage, []);
set(gcf, 'numbertitle', 'off', 'name', ['Filtragem da imagem
    ' num2str(IMAGE_NUMBER) ' por ' METHOD_NAME ' com janela
    de ' num2str(windowSize(1)) ' x ' num2str(windowSize(2))
   ]);
truesize;

%%%%%%%%%%
% S A V E %
%%%%%%%%%%

imwrite((filteredImage - min(min(filteredImage))) / max(max(
   filteredImage - min(min(filteredImage)))), ['Resultados/'
    num2str(IMAGE_NUMBER) '/filtrada' strrep(fileMethodName,
    ' ', '') 'Janela' num2str(windowSize(1)) 'x' num2str(
   windowSize(2)) '.jpg']);

%%%%%%%%%%%%%
% C L E A R %
%%%%%%%%%%%%%

clear OK;
clear IMAGE_NUMBER;
clear METHOD_NAME;
```

C.3 Filtro sigma aditivo

```
function filteredImage = IDAditiveSigmaFilter(originalImage,
    windowSize, numberOfStandardDeviations,
    minimumNumberOfLikelihoodPixels)

%%%%%%%%%%%%%%%%%%%%
% S E T T I N G S %
%%%%%%%%%%%%%%%%%%%%

global OK;
global IMAGE_NUMBER;
global METHOD_NAME;
fileMethodName = METHOD_NAME;
fileMethodName(findstr(METHOD_NAME, ' ') + 1) = upper(
    METHOD_NAME(findstr(METHOD_NAME, ' ') + 1));
fileMethodName(1) = upper(fileMethodName(1));

%%%%%%%%%%%%%%%%%
% F I L T E R %
%%%%%%%%%%%%%%%%%

threshold = numberOfStandardDeviations*std(originalImage(:))
    ;
filteredImage = myNlFilter(originalImage, windowSize,
    @aditiveSigmaFilter, threshold,
    minimumNumberOfLikelihoodPixels);

%%%%%%%%%%%%%%%%%%%
% D I S P L A Y %
%%%%%%%%%%%%%%%%%%%

if OK
```

```
    OK = 0;
    return;
end

figure;
imshow(filteredImage, []);
set(gcf, 'numbertitle', 'off', 'name', ['Filtragem da imagem
    ' num2str(IMAGE_NUMBER) ' por ' METHOD_NAME ' com janela
    de ' num2str(windowSize(1)) ' x ' num2str(windowSize(2))
    ' D = ' sprintf('%f', numberOfStandardDeviations) ' e K
  = ' num2str(minimumNumberOfLikelihoodPixels)]);
truesize;

%%%%%%%%%%
% S A V E %
%%%%%%%%%%

imwrite((filteredImage - min(min(filteredImage))) / max(max(
   filteredImage - min(min(filteredImage)))), ['Resultados/'
    num2str(IMAGE_NUMBER) '/filtrada' strrep(fileMethodName,
    ' ', '') 'Janela' num2str(windowSize(1)) 'x' num2str(
   windowSize(2)) 'D=' sprintf('%f',
   numberOfStandardDeviations) 'K=' num2str(
   minimumNumberOfLikelihoodPixels) '.jpg']);

%%%%%%%%%%%
% C L E A R %
%%%%%%%%%%%

clear OK;
clear IMAGE_NUMBER;
clear METHOD_NAME;
```

```
%%%%%%%%%%%%%%%%%%%%%%
% F U N C T I O N S %
%%%%%%%%%%%%%%%%%%%%%%

function filteredPixel = aditiveSigmaFilter(window,
    threshold, minimumNumberOfLikelihoodPixels)
delta = zeros(size(window));
for x = 1:size(window, 1)
    for y = 1:size(window, 2)
        if abs(window(x, y) - window((size(window, 1) + 1) /
            2, (size(window, 2) + 1) / 2)) <= threshold
            delta(x, y) = 1;
        end
    end
end
if sum(sum(delta)) > minimumNumberOfLikelihoodPixels
    filteredPixel = round(sum(sum(window .* delta)) / sum(
        sum(delta)));
else
    filteredPixel = round((window((size(window, 1) + 1) / 2
        - 1, (size(window, 2) + 1) / 2) + window((size(window
        , 1) + 1) / 2 + 1, (size(window, 2) + 1) / 2) +
        window((size(window, 1) + 1) / 2, (size(window, 2) +
        1) / 2 - 1) + window((size(window, 1) + 1) / 2,(size(
        window, 2) + 1) / 2 + 1)) / 4);
end
```

C.4 Filtro sigma adaptativo

```
function filteredImage = IDAdaptiveSigmaFilter(originalImage
    , windowSize, numberOfStandardDeviations,
    minimumNumberOfLikelihoodPixels)
```

```
%%%%%%%%%%%%%%%%%%%%
% S E T T I N G S %
%%%%%%%%%%%%%%%%%%%%

global OK;
global IMAGE_NUMBER;
global METHOD_NAME;
fileMethodName = METHOD_NAME;
fileMethodName(findstr(METHOD_NAME, ' ') + 1) = upper(
    METHOD_NAME(findstr(METHOD_NAME, ' ') + 1));
fileMethodName(1) = upper(fileMethodName(1));

%%%%%%%%%%%%%%%%%%
% F I L T E R %
%%%%%%%%%%%%%%%%%%

filteredImage = myNlFilter(originalImage, windowSize,
    @adaptiveSigmaFilter, numberOfStandardDeviations,
    minimumNumberOfLikelihoodPixels);

%%%%%%%%%%%%%%%%%%%%
% D I S P L A Y %
%%%%%%%%%%%%%%%%%%%%

if OK
    OK = 0;
    return;
end

figure;
imshow(filteredImage, []);
```

```
set(gcf, 'numbertitle', 'off', 'name', ['Filtragem da imagem
    ' num2str(IMAGE_NUMBER) ' por ' METHOD_NAME ' com janela
    de ' num2str(windowSize(1)) ' x ' num2str(windowSize(2))
    ' D = ' sprintf('%f', numberOfStandardDeviations) ' e K
   = ' num2str(minimumNumberOfLikelihoodPixels)]);
truesize;

%%%%%%%%%%
% S A V E %
%%%%%%%%%%

imwrite((filteredImage - min(min(filteredImage))) / max(max(
    filteredImage - min(min(filteredImage)))), ['Resultados/'
     num2str(IMAGE_NUMBER) '/filtrada' strrep(fileMethodName,
     ' ', '') 'Janela' num2str(windowSize(1)) 'x' num2str(
    windowSize(2)) 'D=' sprintf('%f',
    numberOfStandardDeviations) 'K=' num2str(
    minimumNumberOfLikelihoodPixels) '.jpg']);

%%%%%%%%%%%%
% C L E A R %
%%%%%%%%%%%%

clear OK;
clear IMAGE_NUMBER;
clear METHOD_NAME;

%%%%%%%%%%%%%%%%%%%%%%
% F U N C T I O N S %
%%%%%%%%%%%%%%%%%%%%%%

function filteredPixel = adaptiveSigmaFilter(window,
```

```
    numberOfStandardDeviations,
    minimumNumberOfLikelihoodPixels)
threshold = numberOfStandardDeviations * sqrt(var(window(:))
    );
if abs(window((size(window, 1) + 1) / 2, (size(window, 2) +
    1) / 2) - mean(mean(window))) > sqrt(var(window(:)))
     filteredPixel = sigmaFilter(window, threshold,
         minimumNumberOfLikelihoodPixels);
else
     filteredPixel = window((size(window, 1) + 1) / 2, (size(
         window, 2) + 1) / 2);
end

function filteredPixel = sigmaFilter(window, threshold,
    minimumNumberOfLikelihoodPixels)
delta = zeros(size(window));
for x = 1:size(window, 1)
     for y = 1:size(window, 2)
          if abs(window(x, y) - window((size(window, 1) + 1) /
              2, (size(window, 2) + 1) / 2)) <= threshold
               delta(x, y) = 1;
          end
     end
end
if sum(sum(delta)) > minimumNumberOfLikelihoodPixels
     filteredPixel = round(sum(sum(window .* delta)) / sum(
         sum(delta)));
else
     filteredPixel = round((window((size(window, 1) + 1) / 2
         - 1, (size(window, 2) + 1) / 2) + window((size(window
         , 1) + 1) / 2 + 1, (size(window, 2) + 1) / 2) +
         window((size(window, 1) + 1) / 2, (size(window, 2) +
```

```
        1) / 2 - 1) + window((size(window, 1) + 1) / 2,(size(
        window, 2) + 1) / 2 + 1)) / 4);
end
```

C.5 Filtro sigma multiplicativo

```
function filteredImage = IDMultiplicativeSigmaFilter(
    originalImage, windowSize, numberOfStandardDeviations,
    minimumNumberOfLikelihoodPixels)

%%%%%%%%%%%%%%%%%%%%
% S E T T I N G S %
%%%%%%%%%%%%%%%%%%%%

global OK;
global IMAGE_NUMBER;
global METHOD_NAME;
fileMethodName = METHOD_NAME;
fileMethodName(findstr(METHOD_NAME, ' ') + 1) = upper(
    METHOD_NAME(findstr(METHOD_NAME, ' ') + 1));
fileMethodName(1) = upper(fileMethodName(1));

%%%%%%%%%%%%%%%%
% F I L T E R %
%%%%%%%%%%%%%%%%

threshold = numberOfStandardDeviations*(std(originalImage(:)
    )/mean(originalImage(:)));
filteredImage = myNlFilter(originalImage, windowSize,
    @multiplicativeSigmaFilter, threshold,
    minimumNumberOfLikelihoodPixels);

%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%
% D I S P L A Y %
%%%%%%%%%%%%%%%%%%

if OK
    OK = 0;
    return;
end

figure;
imshow(filteredImage, []);
set(gcf, 'numbertitle', 'off', 'name', ['Filtragem da imagem
    ' num2str(IMAGE_NUMBER) ' por ' METHOD_NAME ' com janela
    de ' num2str(windowSize(1)) ' x ' num2str(windowSize(2))
    ' D = ' sprintf('%f', numberOfStandardDeviations) ' e K
   = ' num2str(minimumNumberOfLikelihoodPixels)]);
truesize;

%%%%%%%%%%%
% S A V E %
%%%%%%%%%%%

imwrite((filteredImage - min(min(filteredImage))) / max(max(
    filteredImage - min(min(filteredImage)))), ['Resultados/'
    num2str(IMAGE_NUMBER) '/filtrada' strrep(fileMethodName,
    ' ', '') 'Janela' num2str(windowSize(1)) 'x' num2str(
    windowSize(2)) 'D=' sprintf('%f',
    numberOfStandardDeviations) 'K=' num2str(
    minimumNumberOfLikelihoodPixels) '.jpg']);

%%%%%%%%%%%%%
% C L E A R %
%%%%%%%%%%%%%
```

```
clear OK;
clear IMAGE_NUMBER;
clear METHOD_NAME;

%%%%%%%%%%%%%%%%%%%%%%%%
% F U N C T I O N S %
%%%%%%%%%%%%%%%%%%%%%%%%

function filteredPixel = multiplicativeSigmaFilter(window,
    threshold, minimumNumberOfLikelihoodPixels)
delta = zeros(size(window));
for x = 1:size(window, 1)
    for y = 1:size(window, 2)
        if (window(x, y) / window((size(window, 1) + 1) / 2,
            (size(window, 2) + 1) / 2) <= 1 + threshold) &&
           (window(x, y) / window((size(window, 1) + 1) / 2,
            (size(window, 2) + 1) / 2) >= 1 - threshold)
            delta(x, y) = 1;
        end
    end
end
if sum(sum(delta)) > minimumNumberOfLikelihoodPixels
    filteredPixel = round(sum(sum(window .* delta)) / sum(
        sum(delta)));
else
    filteredPixel = round((window((size(window, 1) + 1) / 2
        - 1, (size(window, 2) + 1) / 2) + window((size(window
        , 1) + 1) / 2 + 1, (size(window, 2) + 1) / 2) +
        window((size(window, 1) + 1) / 2, (size(window, 2) +
        1) / 2 - 1) + window((size(window, 1) + 1) / 2,(size(
        window, 2) + 1) / 2 + 1)) / 4);
```

```
end
```

C.6 Filtro sigma híbrido

```
function filteredImage = IDHybridSigmaFilter(originalImage,
    windowSize, subWindowSize, numberOfStandardDeviations,
    minimumNumberOfLikelihoodPixels)

%%%%%%%%%%%%%%%%%%%%
% S E T T I N G S %
%%%%%%%%%%%%%%%%%%%%

global OK;
global IMAGE_NUMBER;
global METHOD_NAME;
fileMethodName = METHOD_NAME;
fileMethodName(findstr(METHOD_NAME, ' ') + 1) = upper(
   METHOD_NAME(findstr(METHOD_NAME, ' ') + 1));
fileMethodName(1) = upper(fileMethodName(1));

%%%%%%%%%%%%%%%%%
% F I L T E R %
%%%%%%%%%%%%%%%%%

filteredImage = myNlFilter(originalImage, windowSize,
    @hybridSigmaFilter, subWindowSize,
    numberOfStandardDeviations,
    minimumNumberOfLikelihoodPixels);

%%%%%%%%%%%%%%%%%%%
% D I S P L A Y %
%%%%%%%%%%%%%%%%%%%
```

```
if OK
    OK = 0;
    return;
end

figure;
imshow(filteredImage, []);
set(gcf, 'numbertitle', 'off', 'name', ['Filtragem da imagem
    ' num2str(IMAGE_NUMBER) ' por ' METHOD_NAME ' com janela
    de ' num2str(windowSize(1)) ' x ' num2str(windowSize(2))
    ', subjanela de ' num2str(subWindowSize(1)) ' x '
   num2str(subWindowSize(2)) ', D = ' sprintf('%f',
   numberOfStandardDeviations) ' e K = ' num2str(
   minimumNumberOfLikelihoodPixels)]);
truesize;

%%%%%%%%%%
% S A V E %
%%%%%%%%%%

imwrite((filteredImage - min(min(filteredImage))) / max(max(
   filteredImage - min(min(filteredImage)))), ['Resultados/'
    num2str(IMAGE_NUMBER) '/filtrada' strrep(fileMethodName,
    ' ', '') 'Janela' num2str(windowSize(1)) 'x' num2str(
   windowSize(2)) 'Subjanela' num2str(subWindowSize(1)) 'x'
   num2str(subWindowSize(2)) 'D=' sprintf('%f',
   numberOfStandardDeviations) 'K=' num2str(
   minimumNumberOfLikelihoodPixels) '.jpg']);

%%%%%%%%%%%%
% C L E A R %
%%%%%%%%%%%%
```

```
clear OK;
clear IMAGE_NUMBER;
clear METHOD_NAME;

%%%%%%%%%%%%%%%%%%%%%%%%
% F U N C T I O N S %
%%%%%%%%%%%%%%%%%%%%%%%%

function filteredPixel = hybridSigmaFilter(window,
    subWindowSize, numberOfStandardDeviations,
    minimumNumberOfLikelihoodPixels)
Z1 = round(mean(diag(window((size(window, 1) + 1) / 2 - (
    subWindowSize(1) - 1) / 2:(size(window, 1) + 1) / 2 + (
    subWindowSize(1) - 1) / 2, (size(window, 2) + 1) / 2 - (
    subWindowSize(2) - 1) / 2:(size(window, 1) + 1) / 2 + (
    subWindowSize(2) - 1) / 2))));
Z2 = round(mean(window((size(window, 1) + 1) / 2 - (
    subWindowSize(1) - 1) / 2:(size(window, 1) + 1) / 2 + (
    subWindowSize(1) - 1) / 2,(size(window, 2) + 1) / 2)));
Z3 = round(mean(diag(fliplr(window((size(window, 1) + 1) / 2
     - (subWindowSize(1) - 1) / 2:(size(window, 1) + 1) / 2 +
     (subWindowSize(1) - 1) / 2, (size(window, 2) + 1) / 2 -
    (subWindowSize(2) - 1) / 2:(size(window, 1) + 1) / 2 + (
    subWindowSize(2) - 1) / 2)))));
Z4 = round(mean(window((size(window, 1) + 1) / 2, (size(
    window, 2) + 1) / 2 - (subWindowSize(1) - 1) / 2:(size(
    window, 2) + 1) / 2 + (subWindowSize(1) - 1) / 2)));
Zmax = max([Z1 Z2 Z3 Z4]);
Zmin = min([Z1 Z2 Z3 Z4]);
overral = median([Zmax Zmin window((size(window, 1) + 1) /
    2, (size(window, 2) + 1) / 2)]);
```

```
threshold = numberOfStandardDeviations * sqrt(var(window(:))
    / mean(mean(window .^ 2))) * overral;
delta = zeros(size(window));
for x = 1:size(window, 1)
    for y = 1:size(window, 2)
        if abs(window(x, y) - overral) <= threshold
            delta(x, y) = 1;
        end
    end
end
if sum(sum(delta)) > minimumNumberOfLikelihoodPixels
    filteredPixel = round(sum(sum(window .* delta))/sum(sum(
        delta)));
else
    filteredPixel = round((window((size(window, 1) + 1) / 2
        - 1, (size(window, 2) + 1) / 2) + window((size(window
        , 1) + 1) / 2 + 1, (size(window, 2) + 1) / 2) +
        window((size(window, 1) + 1) / 2, (size(window, 2) +
        1) / 2 - 1) + window((size(window, 1) + 1) / 2, (size
        (window, 2) + 1) / 2 + 1)) / 4);
end
```

C.7 Filtro baseado em wavelets

```
function reconstructedImage =
    AndreTarginoTrabalhoFinalWavelets(image, estimationMethod
    , thresholdMethod, N)
% Denoise MRI via wavelet shrinkage in the following way:
%       AndreTarginoTrabalhoFinalWavelets(image,
    estimationMethod, thresholdMethod)
% where,
%       MRI: can be any number from '0' to '3';
%       estimationMethod: can be 'Bao&Zhang', 'Donoho', '
```

```
    Targino' or 'TarginoRecursive';
%        thresholdMethod: can be 'Bao&Zhang', 'Targino' or '
    TarginoRecursive'.

%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% UFRJ - Universidade Federal do Rio de Janeiro
                                                  %
% COPPE - Instituto Alberto Luiz Coimbra de Pós-Graduação e
    Pesquisa de Engenharia %
% PEE - Programa de Engenharia Elétrica
                                                          %
% Disciplina: Tópicos Especiais em Wavelets
                                                      %
% Professor: Luiz Wagner Pereira Biscainho
                                                       %
% Aluno: André Luiz Nunes Targino da Costa
                                                       %
%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Closing everything.
% close all

% Setting default options if not assigned during function
    call.
% numberOfArguments = nargin;
% if numberOfArguments == 0
%     MRI = '0';
%     estimationMethod = 'TarginoRecursive';
```

```
%       thresholdMethod = 'TarginoRecursive';
% end
% clear numberOfArguments;

% Starting counting execution time.
tic;

% Reading image.
%image = double(dicomread([MRI  '.dcm']))/2^12;

%%%%%%%%%%%%%%%%%%%%%%%%%
% P A R A M E T E R S %
%%%%%%%%%%%%%%%%%%%%%%%%%

% Normalization coeficients for wavelets.
lambda = [1.5  1.12  1.03  1.01  ones(1,ceil(log2(size(image,2))
    ) - 3)];

% Number of scales.
J = 5;

% Number of standard deviations on Targino's threshold
    method.
% N = 0.5;

%%%%%%%%%%%%%%%%%%%
% F I L T E R S %
%%%%%%%%%%%%%%%%%%%

% Zero scale filters.
%       -1      0       1       2
Ho = [0.125  0.375  0.375  0.125];
```

```
%       0  1
Go = [-2  2];
%          -3          -2          -1          0           1           2
Ko = [0.0078125  0.054685  0.171875  -0.171875  -0.054685
    -0.0078125];
%          -3          -2          -1          0           1           2
            3
Lo = [0.0078125  0.046875  0.1171875  0.65625  0.1171875
    0.046875  0.0078125];
%                    -2     -1      0       1
Hoconjugado = [0.125  0.375  0.375  0.125];
%    0
D = 1;

% Upsampled filters at all scales.
for scale = 1:J
    H(scale,1:(size(Ho,2) - 1)*2^scale + 1) = [kron(Ho(1:
        size(Ho,2) - 1),[1 zeros(1,2^scale - 1)]) Ho(size(Ho
        ,2))];
    G(scale,1:(size(Go,2) - 1)*2^scale + 1) = [kron(Go(1:
        size(Go,2) - 1),[1 zeros(1,2^scale - 1)]) Go(size(Go
        ,2))];
    K(scale,1:(size(Ko,2) - 1)*2^scale + 1) = [kron(Ko(1:
        size(Ko,2) - 1),[1 zeros(1,2^scale - 1)]) Ko(size(Ko
        ,2))];
    L(scale,1:(size(Lo,2) - 1)*2^scale + 1) = [kron(Lo(1:
        size(Lo,2) - 1),[1 zeros(1,2^scale - 1)]) Lo(size(Lo
        ,2))];
end
Hconjugado = H;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% T R A N S F O R M A T I O N %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Creating a mirrored image like in a cosine transform.
imageMirrored = mirror(image);

% Calculating the DWT at first scale.
aux = conv2(Go,D,imageMirrored);
W1(:,:,1) = aux(size(image,1) + 1:2*size(image,1),size(image
    ,2) + 1:2*size(image,2));
clear aux;
aux = conv2(D,Go,imageMirrored);
W2(:,:,1) = aux(size(image,1) + 1:2*size(image,1),size(image
    ,2) + 1:2*size(image,2));
clear aux;
aux = conv2(Ho,Ho,imageMirrored);
S(:,:,1) = aux(size(image,1) + 2:2*size(image,1) + 1,size(
   image,2) + 2:2*size(image,2) + 1);
clear aux;

% Calculating the DWT at all other scales.
SMirrored = mirror(S(:,:,1));
for scale = 1:J - 1
    aux = 1/lambda(scale)*conv2(G(scale,1:(size(Go,2) - 1)
        *2^scale + 1),D,SMirrored);
    W1(:,:,scale + 1) = aux(size(image,1) + 2^(scale - 1) +
        1:2*size(image,1) + 2^(scale - 1),size(image,2) +
        1:2*size(image,2));
    clear aux;
    aux = 1/lambda(scale)*conv2(D,G(scale,1:(size(Go,2) - 1)
        *2^scale + 1),SMirrored);
    W2(:,:,scale + 1) = aux(size(image,1) + 1:2*size(image
```

```
        ,1),size(image,2) + 2^(scale - 1) + 1:2*size(image,2)
        + 2^(scale - 1));
    clear aux;
    aux = conv2(H(scale,1:(size(Ho,2) - 1)*2^scale + 1),H(
        scale,1:(size(Ho,2) - 1)*2^scale + 1),SMirrored);
    S(:,:,scale + 1) = aux(size(image,1) + 1 + 2^scale + 2^(
        scale - 1):2*size(image,1) + 2^scale + 2^(scale - 1),
        size(image,2) + 1 + 2^scale + 2^(scale - 1):2*size(
        image,2) + 2^scale + 2^(scale - 1));
    clear aux;
    SMirrored = mirror(S(:,:,scale + 1));
    imwrite((W1(:,:,scale) - min(min(W1(:,:,scale))))/max(
        max(W1(:,:,scale) - min(min(W1(:,:,scale))))), ['
        wavelet1Escala' sprintf('%d', scale) '.bmp']);
    imwrite((W2(:,:,scale) - min(min(W2(:,:,scale))))/max(
        max(W2(:,:,scale) - min(min(W2(:,:,scale))))), ['
        wavelet2Escala' sprintf('%d', scale) '.bmp']);
    imwrite((S(:,:,scale) - min(min(S(:,:,scale))))/max(max(
        S(:,:,scale) - min(min(S(:,:,scale))))), ['
        imagemEscala' sprintf('%d', scale) '.bmp']);
end
imwrite((W1(:,:,J) - min(min(W1(:,:,J))))/max(max(W1(:,:,J)
    - min(min(W1(:,:,J))))), ['wavelet1Escala' sprintf('%d',
    J) '.bmp']);
imwrite((W2(:,:,J) - min(min(W2(:,:,J))))/max(max(W2(:,:,J)
    - min(min(W2(:,:,J))))), ['wavelet2Escala' sprintf('%d',
    J) '.bmp']);
imwrite((S(:,:,J) - min(min(S(:,:,J))))/max(max(S(:,:,J) -
    min(min(S(:,:,J))))), ['imagemEscala' sprintf('%d', J) '.
    bmp']);
clear SMirrored;
```

```
%%%%%%%%%%%%%%%%%%%%%%
% S H R I N K A G E %
%%%%%%%%%%%%%%%%%%%%%%

% Calculating the products of the wavelet transforms at
    adjacents scales.
for scale = 1:J - 1
    P1(:,:,scale) = W1(:,:,scale).*W1(:,:,scale + 1);
    P2(:,:,scale) = W2(:,:,scale).*W2(:,:,scale + 1);
    imwrite((P1(:,:,scale) - min(min(P1(:,:,scale))))/max(
        max(P1(:,:,scale) - min(min(P1(:,:,scale))))), ['
        Produto1Escalas' sprintf('%d%d', scale, scale + 1) '.
        bmp']);
    imwrite((P2(:,:,scale) - min(min(P2(:,:,scale))))/max(
        max(P2(:,:,scale) - min(min(P2(:,:,scale))))), ['
        Produto2Escalas' sprintf('%d%d', scale, scale + 1) '.
        bmp']);
end

% Finding the wavelet coeficients at the diagonal direction
    for noise estimations.
% [h0 h1 g0 g1] = wfilters('db4');
load Daubechies4.mat
aux = conv2(h1,h1,imageMirrored);
W = aux(size(image,1) + 1:2*size(image,1) + 0,size(image,2)
    + 1:2*size(image,2) + 0);
clear aux;
clear imageMirrored;

% Parameters for noise estimations.
normalizedNoiseStandardDeviation = [2.8284 0.7395 0.3173
    0.1531];
```

```
noiseCorrelation = [0.3586 0.5504 0.5957 0.6063];

% Noise estimation.
switch estimationMethod
    case 'Bao&Zhang'
        % Paul Bao & Lei Zhang noise estimation.
        sigma2F = mean(mean(W.^2));
        sigmaF = sqrt(sigma2F);
        aux = find(abs(W(:)) > sigmaF);
        Wa = W(aux);
        clear aux;
        aux = find(abs(W(:)) <= sigmaF);
        Wb = W(aux);
        clear aux;
        sigmaAw = sqrt(mean(Wa'.^2));
        sigmaBw = sqrt(mean(Wb'.^2));
        sigmaG = sqrt(sigmaAw^2 - 2.9^2*sigmaBw^2);
        r = sigmaG/sigmaBw;
        noiseStandardDeviation = sigmaF/sqrt(1 + r^2);
        for scale = 1:J - 1
            noise1StandardDeviation(scale) =
                normalizedNoiseStandardDeviation(scale)*
                noiseStandardDeviation;
            noise2StandardDeviation(scale) =
                normalizedNoiseStandardDeviation(scale)*
                noiseStandardDeviation;
        end
    case 'Donoho'
        % David L. Donoho e Iain M. Johnstone noise
            estimation.
        noiseStandardDeviation = median(abs(W(:)))/0.6745;
        for scale = 1:J - 1
```

```
            noise1StandardDeviation(scale) =
                normalizedNoiseStandardDeviation(scale)*
                noiseStandardDeviation;
            noise2StandardDeviation(scale) =
                normalizedNoiseStandardDeviation(scale)*
                noiseStandardDeviation;
        end
    case {'Targino','TarginoRecursive'}
        % My noise estimation.
        for scale = 1:J - 1
            aux = P1(:,:,scale);
            nonSignalPoints = find(aux(:) < 0);
            aux(nonSignalPoints) = 0;
            signal = aux;
            N1(:,:,scale) = W1(:,:,scale) - sqrt(signal);
            clear aux;
            clear nonSignalPoints;
            clear signal;
            aux = P2(:,:,scale);
            nonSignalPoints = find(aux(:) < 0);
            aux(nonSignalPoints) = 0;
            signal = aux;
            N2(:,:,scale) = W2(:,:,scale) - sqrt(signal);
            clear aux;
            clear nonSignalPoints;
            clear signal;
            noise1StandardDeviation(scale) = sqrt(variance(
                N1(:,:,scale)));
            noise2StandardDeviation(scale) = sqrt(variance(
                N2(:,:,scale)));
        end
end
```

```
% Setting threshold.
switch thresholdMethod
    case 'Donoho'
        threshold1(scale) = noise1StandardDeviation(scale)*
          sqrt(2*log(size(image,1)*size(image,2)));
        threshold2(scale) = noise2StandardDeviation(scale)*
          sqrt(2*log(size(image,1)*size(image,2)));
    case 'Bao&Zhang'
        % Paul Bao & Lei Zhang threshold estimation.
        for scale = 1:J - 1
            if scale == J - 1
                k(scale) = sqrt(1 + 2*noiseCorrelation(scale
                  )^2)*noise1StandardDeviation(scale)^2;
                productNoiseMean1(scale) = noiseCorrelation(
                  scale)*noise1StandardDeviation(scale)^2;
                productNoiseMean2(scale) = noiseCorrelation(
                  scale)*noise2StandardDeviation(scale)^2;
            else
                k(scale) = sqrt(1 + 2*noiseCorrelation(scale
                  )^2)*noise1StandardDeviation(scale)*
                  noise1StandardDeviation(scale + 1);
                productNoiseMean1(scale) = noiseCorrelation(
                  scale)*noise1StandardDeviation(scale)*
                  noise1StandardDeviation(scale + 1);
                productNoiseMean2(scale) = noiseCorrelation(
                  scale)*noise2StandardDeviation(scale)*
                  noise2StandardDeviation(scale + 1);
            end
            productNoiselessImageMean1(scale) = mean(mean(P1
              (:,:,scale))) - productNoiseMean1(scale);
            productNoiselessImageMean2(scale) = mean(mean(P2
```

```
                (:,:,scale))) - productNoiseMean2(scale);
            threshold1(scale) = 5 * k(scale)*(1 +
                productNoiseMean1(scale)/
                productNoiselessImageMean1(scale));
            threshold2(scale) = 5 * k(scale)*(1 +
                productNoiseMean2(scale)/
                productNoiselessImageMean2(scale));
        end
    case {'Targino','TarginoRecursive'}
        % My threshold estimation.
        for scale = 1:J - 1
            threshold1(scale) = N*noise1StandardDeviation(
                scale)^2;
            threshold2(scale) = N*noise2StandardDeviation(
                scale)^2;
        end
end

% My recursive noise end threshold estimation.
if strcmp(estimationMethod, 'TarginoRecursive') & strcmp(
    thresholdMethod, 'TarginoRecursive')
    for scale = 1:J - 1
        aux = P1(:,:,scale);
        nonSignalPoints = find(aux(:) < threshold1(scale));
        aux(nonSignalPoints) = 0;
        signal = aux;
        N1(:,:,scale) = W1(:,:,scale) - sqrt(signal);
        clear aux;
        clear nonSignalPoints;
        clear signal;
        aux = P2(:,:,scale);
        nonSignalPoints = find(aux(:) < threshold2(scale));
```

```
        aux(nonSignalPoints) = 0;
        signal = aux;
        N2(:,:,scale) = W2(:,:,scale) - sqrt(signal);
        clear aux;
        clear nonSignalPoints;
        clear signal;
        noise1StandardDeviation(scale) = sqrt(variance(N1
            (:,:,scale)));
        noise2StandardDeviation(scale) = sqrt(variance(N2
            (:,:,scale)));
        oldThreshold1(scale) = threshold1(scale);
        oldThreshold2(scale) = threshold2(scale);
        threshold1(scale) = N*noise1StandardDeviation(scale)
            ^2;
        threshold2(scale) = N*noise2StandardDeviation(scale)
            ^2;
    end
    while (abs(threshold1(1) - oldThreshold1(1)) > 255^-2) &
        (abs(threshold1(2) - oldThreshold1(2)) > 255^-2) & (
       abs(threshold1(3) - oldThreshold1(3)) > 255^-2) & (
       abs(threshold1(4) - oldThreshold1(4)) > 255^-2) & (
       abs(threshold2(1) - oldThreshold2(1)) > 255^-2) & (
       abs(threshold2(2) - oldThreshold2(2)) > 255^-2) & (
       abs(threshold2(3) - oldThreshold2(3)) > 255^-2) & (
       abs(threshold2(4) - oldThreshold2(4)) > 255^-2)
        for scale = 1:J - 1
            aux = P1(:,:,scale);
            nonSignalPoints = find(aux(:) < threshold1(scale
                ));
            aux(nonSignalPoints) = 0;
            signal = aux;
            N1(:,:,scale) = W1(:,:,scale) - sqrt(signal);
```

```
            clear aux;
            clear nonSignalPoints;
            clear signal;
            aux = P2(:,:,scale);
            nonSignalPoints = find(aux(:) < threshold2(scale
                ));
            aux(nonSignalPoints) = 0;
            signal = aux;
            N2(:,:,scale) = W2(:,:,scale) - sqrt(signal);
            clear aux;
            clear nonSignalPoints;
            clear signal;
            noise1StandardDeviation(scale) = sqrt(variance(
                N1(:,:,scale)));
            noise2StandardDeviation(scale) = sqrt(variance(
                N2(:,:,scale)));
            oldThreshold1(scale) = threshold1(scale);
            oldThreshold2(scale) = threshold2(scale);
            threshold1(scale) = N*noise1StandardDeviation(
                scale)^2;
            threshold2(scale) = N*noise2StandardDeviation(
                scale)^2;
        end
    end
end

% Doing the shrinkage.
if strcmp(thresholdMethod, 'Donoho')
    for scale = 1:J - 1
        aux1 = W1(:,:,scale);
        shrink1 = find(abs(aux1(:)) < threshold1(scale));
        aux2 = W1(:,:,scale);
```

```
        aux2(shrink1) = 0;
        W1(:,:,scale) = aux2;
        clear aux1;
        clear aux2;
        clear shrink1;

        aux1 = W2(:,:,scale);
        shrink2 = find(abs(aux1(:)) < threshold2(scale));
        aux2 = W2(:,:,scale);
        aux2(shrink2) = 0;
        W2(:,:,scale) = aux2;
        clear aux1;
        clear aux2;
        clear shrink2;
    end
else
    for scale = 1:J - 1
        aux1 = P1(:,:,scale);
        shrink1 = find(aux1(:) < threshold1(scale));
        aux2 = W1(:,:,scale);
        aux2(shrink1) = 0;
        W1(:,:,scale) = aux2;
        clear aux1;
        clear aux2;
        clear shrink1;

        aux1 = P2(:,:,scale);
        shrink2 = find(aux1(:) < threshold2(scale));
        aux2 = W2(:,:,scale);
        aux2(shrink2) = 0;
        W2(:,:,scale) = aux2;
        clear aux1;
```

```
        clear aux2;
        clear shrink2;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% R E C O N S T R U C T I O N %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Calculating the IDWT at all scales but the first one.
scale = J;
while (scale > 1)
    W1Mirrored = mirror(W1(:,:,scale));
    W2Mirrored = mirror(W2(:,:,scale));
    SMirrored = mirror(S(:,:,scale));
    aux = lambda(scale)*conv2(K(scale - 1,1:(size(Ko,2) - 1)
        *2^(scale - 1) + 1),L(scale - 1,1:(size(Lo,2) - 1)
        *2^(scale - 1) + 1),W1Mirrored);
    aux2 = lambda(scale)*conv2(L(scale - 1,1:(size(Lo,2) -
        1)*2^(scale - 1) + 1),K(scale - 1,1:(size(Ko,2) - 1)
        *2^(scale - 1) + 1),W2Mirrored);
    aux3 = conv2(Hconjugado(scale - 1,1:(size(Hoconjugado,2)
         - 1)*2^(scale - 1) + 1),Hconjugado(scale - 1,1:(size
        (Hoconjugado,2) - 1)*2^(scale - 1) + 1),SMirrored);
    temp = aux(size(image,1) + 2*2^(scale - 1) + 2^(scale -
        2) + 1:2*size(image,1) + 2*2^(scale - 1) + 2^(scale -
         2),size(image,2) + 3*2^(scale - 1) + 1:2*size(image
        ,2) + 3*2^(scale - 1));
    temp2 = aux2(size(image,1) + 3*2^(scale - 1) + 1:2*size(
        image,1) + 3*2^(scale - 1),size(image,2) + 2*2^(scale
         - 1) + 2^(scale - 2) + 1:2*size(image,2) + 2*2^(
        scale - 1) + 2^(scale - 2));
```

```
    temp3 = aux3(size(image,1) + 1 + 2^(scale - 1) + 2^(
        scale - 2):2*size(image,1) + 2^(scale - 1) + 2^(scale
        - 2),size(image,2) + 1 + 2^(scale - 1) + 2^(scale -
        2):2*size(image,2) + 2^(scale - 1) + 2^(scale - 2));
    S(:,:,scale - 1) = temp + temp2 + temp3;
    clear aux;
    clear aux2;
    clear aux3;
    clear temp;
    clear temp2;
    clear temp3;
    clear W1Mirrored;
    clear W2Mirrored;
    clear SMirrored;
    scale = scale - 1;
    imwrite((W1(:,:,scale) - min(min(W1(:,:,scale))))/max(
        max(W1(:,:,scale) - min(min(W1(:,:,scale))))), ['
        wavelet1ShrinkadaEscala' sprintf('%d', scale) '.bmp'
        ]);
    imwrite((W2(:,:,scale) - min(min(W2(:,:,scale))))/max(
        max(W2(:,:,scale) - min(min(W2(:,:,scale))))), ['
        wavelet2ShrinkadaEscala' sprintf('%d', scale) '.bmp'
        ]);
    imwrite((S(:,:,scale) - min(min(S(:,:,scale))))/max(max(
        S(:,:,scale) - min(min(S(:,:,scale))))), ['
        imagemShrinkadaEscala' sprintf('%d', scale) '.bmp']);
end

% Calculating the IDWT at the first scale.
W1Mirrored = mirror(W1(:,:,1));
W2Mirrored = mirror(W2(:,:,1));
SMirrored = mirror(S(:,:,1));
```

```
aux = lambda(1)*conv2(Ko,Lo,W1Mirrored);
aux2 = lambda(1)*conv2(Lo,Ko,W2Mirrored);
aux3 = conv2(Hoconjugado,Hoconjugado,SMirrored);
temp = aux(size(image,1) + 4:2*size(image,1) + 3,size(image
    ,2) + 4:2*size(image,2) + 3);
temp2 = aux2(size(image,1) + 4:2*size(image,1) + 3,size(
    image,2) + 4:2*size(image,2) + 3);
temp3 = aux3(size(image,1) + 3:2*size(image,1) + 2,size(
    image,2) + 3:2*size(image,2) + 2);
reconstructedImage = temp + temp2 + temp3;
clear aux;
clear aux2;
clear aux3;
clear temp;
clear temp2;
clear temp3;
clear W1Mirrored;
clear W2Mirrored;
clear SMirrored;
imwrite((W1(:,:,1) - min(min(W1(:,:,1))))/max(max(W1(:,:,1)
    - min(min(W1(:,:,1))))), ['wavelet1ShrinkadaEscala'
    sprintf('%d', 1) '.bmp']);
imwrite((W2(:,:,1) - min(min(W2(:,:,1))))/max(max(W2(:,:,1)
    - min(min(W2(:,:,1))))), ['wavelet2ShrinkadaEscala'
    sprintf('%d', 1) '.bmp']);
imwrite((S(:,:,1) - min(min(S(:,:,1))))/max(max(S(:,:,1) -
    min(min(S(:,:,1))))), ['imagemShrinkadaEscala' sprintf('%
    d', 1) '.bmp']);
imwrite((image - min(min(image)))/max(max(image - min(min(
    image)))), 'imagemOriginal.bmp');
imwrite((reconstructedImage - min(min(reconstructedImage)))/
    max(max(reconstructedImage - min(min(reconstructedImage))
```

```
    )), 'imagemFiltrada.bmp');

%%%%%%%%%%%%%%%%%%
% D I S P L A Y %
%%%%%%%%%%%%%%%%%%

% Showing original and filtered images.
% figure;
% imshow(image, []);
% truesize;
% title('Imagem original');
% figure;
% imshow(reconstructedImage, []);
% truesize;
% title('Imagem filtrada');

% Finishing count and exhibiting execution time.
tempoTotal = toc

% Clearing everything.
% clear all;

%%%%%%%%%%%%%%%%%%%%%%
% F U N C T I O N S %
%%%%%%%%%%%%%%%%%%%%%%

function mirrorImage = mirror(image)
% Create a mirrored image like in a cosine transform in the
    following way:
%       mirrorImage = mirror(image)
% where,
%       image: can be any image.
```

```
imageD = fliplr(image);
imageB = flipud(image);
imageDB = flipud(imageD);

mirrorImage = [imageDB imageB imageDB; imageD image imageD;
    imageDB imageB imageDB];

function sigma2 = variance(image)
% Calculate variance of image pixels in the following way:
%       sigma2 = variance(image)
% where,
%       image: can be any image.

sigma2 = var(image(:)');
```

C.8 Filtro morfológico

```
function filteredImage = IDMorphologicalFilter(originalImage
    , diskRadius, hatThreshold, bumpVolumeThreshold)

%%%%%%%%%%%%%%%%%%%%
% S E T T I N G S %
%%%%%%%%%%%%%%%%%%%%

global IMAGE_NUMBER;
global METHOD_NAME;
global ALL_IMAGES;
global ALL_FILES;
global OK;
connectedNeighborhood = 8;
showedAll = 0;
showedAndSavedFilteredImage = 0;
```

```
filteredImage = originalImage;
fileMethodName = METHOD_NAME;
fileMethodName(findstr(METHOD_NAME, ' ') + 1) = upper(
    METHOD_NAME(findstr(METHOD_NAME, ' ') + 1));
fileMethodName(1) = upper(fileMethodName(1));

%%%%%%%%%%%%%%%
% F I L T E R %
%%%%%%%%%%%%%%%

filteringProgressBar = waitbar(0, ['Filtrando por '
    METHOD_NAME '...'], 'CreateCancelBtn', 'delete(gcf);
    global OK; OK = 1; clear OK;', 'KeyPressFcn', @cancel);
handle = get(filteringProgressBar, 'children');
set(handle(2), 'string', 'Cancelar');
structuringElement = strel('disk', diskRadius);
openedImage = imopen(originalImage, structuringElement);
closedImage = imclose(originalImage, structuringElement);
topHatImage = imtophat(originalImage, structuringElement);
bottomHatImage = imbothat(originalImage, structuringElement)
    ;
topMask = zeros(size(topHatImage));
topMaskSignalPoints = find(topHatImage > hatThreshold);
topMask(topMaskSignalPoints) = 1;
topBumpMap = bwlabel(topMask, connectedNeighborhood);
topBumpVolume = zeros(max(max(topBumpMap)),1);
bottomMask = zeros(size(bottomHatImage));
bottomMaskSignalPoints = find(bottomHatImage > hatThreshold)
    ;
bottomMask(bottomMaskSignalPoints) = 1;
bottomBumpMap = bwlabel(bottomMask, connectedNeighborhood);
bottomBumpVolume = zeros(max(max(bottomBumpMap)),1);
```

```
for x = 1:size(originalImage, 1)
    for y = 1:size(originalImage, 2)
        if topBumpMap(x, y) > 0
            topBumpVolume(topBumpMap(x, y)) = topBumpVolume(
                topBumpMap(x, y)) + topHatImage(x, y);
        end
        if bottomBumpMap(x, y) > 0
            bottomBumpVolume(bottomBumpMap(x, y)) =
                bottomBumpVolume(bottomBumpMap(x, y)) +
                bottomHatImage(x, y);
        end
    end
    if OK
        OK = 0;
        return;
    else
        if not(OK)
            set(filteringProgressBar, 'name', [num2str(round
                (100 * (x / (2 * size(originalImage, 1)))))
                '% concluído']);
        end
        waitbar(x / (2 * size(originalImage, 1)));
    end
end
filteredTopHatImage = topHatImage;
filteredBottomHatImage = bottomHatImage;
for x = 1:size(originalImage, 1)
    for y = 1:size(originalImage, 2)
        if topBumpMap(x, y) > 0
            if topBumpVolume(topBumpMap(x, y)) <
                bumpVolumeThreshold
                 filteredTopHatImage(x, y) = 0;
```

```
            end
        end
        if bottomBumpMap(x, y) > 0
            if bottomBumpVolume(bottomBumpMap(x, y)) <
               bumpVolumeThreshold
                filteredBottomHatImage(x, y) = 0;
            end
        end
    end
    if OK
        OK = 0;
        return;
    else
        if not(OK)
            set(filteringProgressBar, 'name', [num2str(round
               (100 * (0.5 + x / (2 * size(originalImage, 1)
               )))) '% concluído']);
        end
        waitbar(0.5 + x / (2 * size(originalImage, 1)));
    end
end
filteredImage = round((openedImage + filteredTopHatImage +
   closedImage - filteredBottomHatImage)/2);
figure(1);
uicontrol('FontName','default', 'BackgroundColor', 'default'
   , 'style', 'text', 'units', 'normalized', 'Position',
   [0.35 0.025 0.30 0.020], 'string', ['Tempo de execução: '
    num2str(floor(toc)) 's']);
close(filteringProgressBar);
OK = 0;

%%%%%%%%%%%%%%%%%%
```

```
%DISPLAY%
%%%%%%%%%%%%%%%%%

while 1

    if and(ALL_IMAGES == 1, not(showedAll))
        figure;
        imshow(openedImage, []);
        set(gcf, 'numberTitle', 'off', 'name', ['Abertura da
            imagem ' num2str(IMAGE_NUMBER) ' por '
           METHOD_NAME ' com R = ' num2str(diskRadius)]);
        truesize;
        figure;
        imshow(closedImage, []);
        set(gcf, 'numberTitle', 'off', 'name', ['Fechamento
           da imagem ' num2str(IMAGE_NUMBER) ' por '
           METHOD_NAME ' com R = ' num2str(diskRadius)]);
        truesize;
        figure;
        imshow(topHatImage, []);
        set(gcf, 'numberTitle', 'off', 'name', ['Top hat da
           imagem ' num2str(IMAGE_NUMBER) ' por '
           METHOD_NAME ' com R = ' num2str(diskRadius)]);
        truesize;
        figure;
        imshow(bottomHatImage, []);
        set(gcf, 'numberTitle', 'off', 'name', ['Bottom hat
           da imagem ' num2str(IMAGE_NUMBER) ' por '
           METHOD_NAME ' com R = ' num2str(diskRadius)]);
        truesize;
        figure;
        imshow(filteredTopHatImage, []);
```

```
    set(gcf, 'numberTitle', 'off', 'name', ['Top hat
        filtrado da imagem ' num2str(IMAGE_NUMBER) ' por
        ' METHOD_NAME ' com R = ' num2str(diskRadius) ',
       H = ' num2str(hatThreshold) ' e V = ' num2str(
        bumpVolumeThreshold)]);
    truesize;
    figure;
    imshow(filteredBottomHatImage, []);
    set(gcf, 'numberTitle', 'off', 'name', ['Bottom hat
        filtrado da imagem ' num2str(IMAGE_NUMBER) ' por
        ' METHOD_NAME ' com R = ' num2str(diskRadius) ',
       H = ' num2str(hatThreshold) ' e V = ' num2str(
        bumpVolumeThreshold)]);
    truesize;
    figure;
    imshow(topMask);
    set(gcf, 'numberTitle', 'off', 'name', ['Máscara do
        top hat da imagem ' num2str(IMAGE_NUMBER) ' por '
         METHOD_NAME ' com R = ' num2str(diskRadius) ' e
       H = ' num2str(hatThreshold)]);
    truesize;
    figure;
    imshow(bottomMask);
    set(gcf, 'numberTitle', 'off', 'name', ['Máscara do
        bottom hat da imagem ' num2str(IMAGE_NUMBER) '
        por ' METHOD_NAME ' com R = ' num2str(diskRadius)
         ' e H = ' num2str(hatThreshold)]);
    truesize;
    showedAll = 1;
end
if not(showedAndSavedFilteredImage)
    figure;
```

```
        imshow(filteredImage, []);
        set(gcf, 'numberTitle', 'off', 'name', ['Filtragem
            da imagem ' num2str(IMAGE_NUMBER) ' por '
            METHOD_NAME ' com R = ' num2str(diskRadius) ', H
            = ' num2str(hatThreshold) ' e V = ' num2str(
            bumpVolumeThreshold)]);
        truesize;
    end

%%%%%%%%%%
% S A V E %
%%%%%%%%%%

    if ALL_FILES == 1
        imwrite((openedImage - min(min(openedImage))) / max(
            max(openedImage - min(min(openedImage)))), ['
            Resultados/' num2str(IMAGE_NUMBER) '/abertura'
            strrep(fileMethodName, ' ', '') 'R=' num2str(
            diskRadius) '.jpg']);
        imwrite((closedImage - min(min(closedImage))) / max(
            max(closedImage - min(min(closedImage)))), ['
            Resultados/' num2str(IMAGE_NUMBER) '/fechamento'
            strrep(fileMethodName, ' ', '') 'R=' num2str(
            diskRadius) '.jpg']);
        imwrite((topHatImage - min(min(topHatImage))) / max(
            max(topHatImage - min(min(topHatImage)))), ['
            Resultados/' num2str(IMAGE_NUMBER) '/topHat'
            strrep(fileMethodName, ' ', '') 'R=' num2str(
            diskRadius) '.jpg']);
        imwrite((bottomHatImage - min(min(bottomHatImage)))
            / max(max(bottomHatImage - min(min(bottomHatImage
            )))), ['Resultados/' num2str(IMAGE_NUMBER) '/
```

```
        bottomHat' strrep(fileMethodName, ' ', '') 'R='
        num2str(diskRadius) '.jpg']);
    imwrite((filteredTopHatImage - min(min(
        filteredTopHatImage))) / max(max(
        filteredTopHatImage - min(min(filteredTopHatImage
        )))), ['Resultados/' num2str(IMAGE_NUMBER) '/
        topHatFiltrado' strrep(fileMethodName, ' ', '') '
        R=' num2str(diskRadius) 'H=' num2str(hatThreshold
        ) 'V=' num2str(bumpVolumeThreshold) '.jpg']);
    imwrite((filteredBottomHatImage - min(min(
        filteredBottomHatImage))) / max(max(
        filteredBottomHatImage - min(min(
        filteredBottomHatImage)))), ['Resultados/'
        num2str(IMAGE_NUMBER) '/bottomHatFiltrado' strrep
        (fileMethodName, ' ', '') 'R=' num2str(diskRadius
        ) 'H=' num2str(hatThreshold) 'V=' num2str(
        bumpVolumeThreshold) '.jpg']);
    imwrite(topMask, ['Resultados/' num2str(IMAGE_NUMBER
        ) '/mascaraTopHat' strrep(fileMethodName, ' ', ''
        ) 'R=' num2str(diskRadius) 'H=' num2str(
        hatThreshold) '.jpg']);
    imwrite(bottomMask, ['Resultados/' num2str(
        IMAGE_NUMBER) '/mascaraBottomHat' strrep(
        fileMethodName, ' ', '') 'R=' num2str(diskRadius)
         'H=' num2str(hatThreshold) '.jpg']);
end
if not(showedAndSavedFilteredImage)
    imwrite((filteredImage - min(min(filteredImage))) /
        max(max(filteredImage - min(min(filteredImage))))
        , ['Resultados/' num2str(IMAGE_NUMBER) '/filtrada
        ' strrep(fileMethodName, ' ', '') 'R=' num2str(
        diskRadius) 'H=' num2str(hatThreshold) 'V='
```

```
            num2str(bumpVolumeThreshold) '.jpg']);
        showedAndSavedFilteredImage = 1;
    end

    if and(ALL_IMAGES == 1, ALL_FILES == 1)
        return;
    end

    uiwait(1);

    if OK
        OK = 0;
        return;
    end

end

%%%%%%%%%%%%%
% C L E A R %
%%%%%%%%%%%%%

clear IMAGE_NUMBER;
clear METHOD_NAME;
clear ALL_IMAGES;
clear ALL_FILES;
clear OK;

%%%%%%%%%%%%%%%%%%%%%%%
% F U N C T I O N S %
%%%%%%%%%%%%%%%%%%%%%%%

function cancel(src, evnt)
```

```
global OK;
if evnt.Key == 'escape'
    delete(gcf);
    OK = 1;
end
clear OK;
```

C.9 Filtro morfológico II

```
function filteredImage = IDMorphologicalFilterII(
    originalImage, diskRadius, hatThreshold,
    bumpVolumeThreshold)

%%%%%%%%%%%%%%%%%%%%
% S E T T I N G S %
%%%%%%%%%%%%%%%%%%%%

global IMAGE_NUMBER;
global METHOD_NAME;
global ALL_IMAGES;
global ALL_FILES;
global OK;
connectedNeighborhood = 8;
showedAll = 0;
showedAndSavedFilteredImage = 0;
filteredImage = originalImage;
fileMethodName = METHOD_NAME;
fileMethodName(findstr(METHOD_NAME, ' ') + 1) = upper(
    METHOD_NAME(findstr(METHOD_NAME, ' ') + 1));
fileMethodName(1) = upper(fileMethodName(1));

%%%%%%%%%%%%%%%%
% F I L T E R %
```

```
%%%%%%%%%%%%%%

filteringProgressBar = waitbar(0, ['Filtrando por '
    METHOD_NAME '...'], 'CreateCancelBtn', 'delete(gcf);
    global OK; OK = 1; clear OK;', 'KeyPressFcn', @cancel);
handle = get(filteringProgressBar, 'children');
set(handle(2), 'string', 'Cancelar');
structuringElement = strel('disk', diskRadius);
closedOpenedImage = imclose(imopen(originalImage,
    structuringElement), structuringElement);
topHatImage = originalImage - closedOpenedImage;
topHatImage = round((topHatImage + abs(topHatImage))/2);
openedClosedImage = imopen(imclose(originalImage,
    structuringElement), structuringElement);
bottomHatImage = openedClosedImage - originalImage;
bottomHatImage = round((bottomHatImage + abs(bottomHatImage)
    )/2);
topMask = zeros(size(topHatImage));
topMaskSignalPoints = find(topHatImage > hatThreshold);
topMask(topMaskSignalPoints) = 1;
topBumpMap = bwlabel(topMask, connectedNeighborhood);
topBumpVolume = zeros(max(max(topBumpMap)),1);
bottomMask = zeros(size(bottomHatImage));
bottomMaskSignalPoints = find(bottomHatImage > hatThreshold)
    ;
bottomMask(bottomMaskSignalPoints) = 1;
bottomBumpMap = bwlabel(bottomMask, connectedNeighborhood);
bottomBumpVolume = zeros(max(max(bottomBumpMap)),1);
for x = 1:size(originalImage, 1)
    for y = 1:size(originalImage, 2)
        if topBumpMap(x, y) > 0
            topBumpVolume(topBumpMap(x, y)) = topBumpVolume(
```

```
                topBumpMap(x, y)) + topHatImage(x, y);
        end
        if bottomBumpMap(x, y) > 0
            bottomBumpVolume(bottomBumpMap(x, y)) =
                bottomBumpVolume(bottomBumpMap(x, y)) +
                bottomHatImage(x, y);
        end
    end
    if OK
        OK = 0;
        return;
    else
        if not(OK)
            set(filteringProgressBar, 'name', [num2str(round
                (100 * (x / (2 * size(originalImage, 1)))))
                '% concluído']);
        end
        waitbar(x / (2 * size(originalImage, 1)));
    end
end
filteredTopHatImage = topHatImage;
filteredBottomHatImage = bottomHatImage;
for x = 1:size(originalImage, 1)
    for y = 1:size(originalImage, 2)
        if topBumpMap(x, y) > 0
            if topBumpVolume(topBumpMap(x, y)) <
                bumpVolumeThreshold
                 filteredTopHatImage(x, y) = 0;
            end
        end
        if bottomBumpMap(x, y) > 0
            if bottomBumpVolume(bottomBumpMap(x, y)) <
```

```
                    bumpVolumeThreshold
                        filteredBottomHatImage(x, y) = 0;
                end
            end
        end
        if OK
            OK = 0;
            return;
        else
            if not(OK)
                set(filteringProgressBar, 'name', [num2str(round
                    (100 * (0.5 + x / (2 * size(originalImage, 1)
                    )))) '% concluído']);
            end
            waitbar(0.5 + x / (2 * size(originalImage, 1)));
        end
    end
    filteredImage = round((closedOpenedImage - abs(originalImage
        - closedOpenedImage) + openedClosedImage + abs(
        openedClosedImage - originalImage))/2 +
        filteredTopHatImage - filteredBottomHatImage);
    figure(1);
    uicontrol('FontName','default', 'BackgroundColor', 'default'
        , 'style', 'text', 'units', 'normalized', 'Position',
        [0.35 0.025 0.30 0.020], 'string', ['Tempo de execução: '
        num2str(floor(toc)) 's']);
    close(filteringProgressBar);
    OK = 0;

    %%%%%%%%%%%%%%%%%%%
    % D I S P L A Y %
    %%%%%%%%%%%%%%%%%%%
```

```
while 1

    if and(ALL_IMAGES == 1, not(showedAll))
        figure;
        imshow(closedOpenedImage, []);
        set(gcf, 'numberTitle', 'off', 'name', ['Fechamento
            da abertura da imagem ' num2str(IMAGE_NUMBER) '
            por ' METHOD_NAME ' com R = ' num2str(diskRadius)
            ]);
        truesize;
        figure;
        imshow(openedClosedImage, []);
        set(gcf, 'numberTitle', 'off', 'name', ['Abertura do
             fechamento da imagem ' num2str(IMAGE_NUMBER) '
            por ' METHOD_NAME ' com R = ' num2str(diskRadius)
            ]);
        truesize;
        figure;
        imshow(topHatImage, []);
        set(gcf, 'numberTitle', 'off', 'name', ['Top hat da
            imagem ' num2str(IMAGE_NUMBER) ' por '
            METHOD_NAME ' com R = ' num2str(diskRadius)]);
        truesize;
        figure;
        imshow(bottomHatImage, []);
        set(gcf, 'numberTitle', 'off', 'name', ['Bottom hat
            da imagem ' num2str(IMAGE_NUMBER) ' por '
            METHOD_NAME ' com R = ' num2str(diskRadius)]);
        truesize;
        figure;
        imshow(filteredTopHatImage, []);
```

```
    set(gcf, 'numberTitle', 'off', 'name', ['Top hat
        filtrado da imagem ' num2str(IMAGE_NUMBER) ' por
        ' METHOD_NAME ' com R = ' num2str(diskRadius) ',
       H = ' num2str(hatThreshold) ' e V = ' num2str(
        bumpVolumeThreshold)]);
    truesize;
    figure;
    imshow(filteredBottomHatImage, []);
    set(gcf, 'numberTitle', 'off', 'name', ['Bottom hat
        filtrado da imagem ' num2str(IMAGE_NUMBER) ' por
        ' METHOD_NAME ' com R = ' num2str(diskRadius) ',
       H = ' num2str(hatThreshold) ' e V = ' num2str(
        bumpVolumeThreshold)]);
    truesize;
    figure;
    imshow(topMask);
    set(gcf, 'numberTitle', 'off', 'name', ['Máscara do
        top hat da imagem ' num2str(IMAGE_NUMBER) ' por '
         METHOD_NAME ' com R = ' num2str(diskRadius) ' e
       H = ' num2str(hatThreshold)]);
    truesize;
    figure;
    imshow(bottomMask);
    set(gcf, 'numberTitle', 'off', 'name', ['Máscara do
        bottom hat da imagem ' num2str(IMAGE_NUMBER) '
        por ' METHOD_NAME ' com R = ' num2str(diskRadius)
         ' e H = ' num2str(hatThreshold)]);
    truesize;
    showedAll = 1;
end
if not(showedAndSavedFilteredImage)
    figure;
```

```
        imshow(filteredImage, []);
        set(gcf, 'numberTitle', 'off', 'name', ['Filtragem
            da imagem ' num2str(IMAGE_NUMBER) ' por '
            METHOD_NAME ' com R = ' num2str(diskRadius) ', H
            = ' num2str(hatThreshold) ' e V = ' num2str(
            bumpVolumeThreshold)]);
        truesize;
    end

%%%%%%%%%%
% S A V E %
%%%%%%%%%%

    if ALL_FILES == 1
        imwrite((closedOpenedImage - min(min(
            closedOpenedImage))) / max(max(closedOpenedImage
            - min(min(closedOpenedImage)))), ['Resultados/'
            num2str(IMAGE_NUMBER) '/fechamentoDaAbertura'
            strrep(fileMethodName, ' ', '') 'R=' num2str(
            diskRadius) '.jpg']);
        imwrite((openedClosedImage - min(min(
            openedClosedImage))) / max(max(openedClosedImage
            - min(min(openedClosedImage)))), ['Resultados/'
            num2str(IMAGE_NUMBER) '/aberturaDoFechamento'
            strrep(fileMethodName, ' ', '') 'R=' num2str(
            diskRadius) '.jpg']);
        imwrite((topHatImage - min(min(topHatImage))) / max(
            max(topHatImage - min(min(topHatImage)))), ['
            Resultados/' num2str(IMAGE_NUMBER) '/topHat'
            strrep(fileMethodName, ' ', '') 'R=' num2str(
            diskRadius) '.jpg']);
        imwrite((bottomHatImage - min(min(bottomHatImage)))
```

```
        / max(max(bottomHatImage - min(min(bottomHatImage
        ))))), ['Resultados/' num2str(IMAGE_NUMBER) '/
        bottomHat' strrep(fileMethodName, ' ', '') 'R='
        num2str(diskRadius) '.jpg']);
    imwrite((filteredTopHatImage - min(min(
        filteredTopHatImage))) / max(max(
        filteredTopHatImage - min(min(filteredTopHatImage
        ))))), ['Resultados/' num2str(IMAGE_NUMBER) '/
        topHatFiltrado' strrep(fileMethodName, ' ', '') '
        R=' num2str(diskRadius) 'H=' num2str(hatThreshold
        ) 'V=' num2str(bumpVolumeThreshold) '.jpg']);
    imwrite((filteredBottomHatImage - min(min(
        filteredBottomHatImage))) / max(max(
        filteredBottomHatImage - min(min(
        filteredBottomHatImage)))), ['Resultados/'
        num2str(IMAGE_NUMBER) '/bottomHatFiltrado' strrep
        (fileMethodName, ' ', '') 'R=' num2str(diskRadius
        ) 'H=' num2str(hatThreshold) 'V=' num2str(
        bumpVolumeThreshold) '.jpg']);
    imwrite(topMask, ['Resultados/' num2str(IMAGE_NUMBER
        ) '/mascaraTopHat' strrep(fileMethodName, ' ', ''
        ) 'R=' num2str(diskRadius) 'H=' num2str(
        hatThreshold) '.jpg']);
    imwrite(bottomMask, ['Resultados/' num2str(
        IMAGE_NUMBER) '/mascaraBottomHat' strrep(
        fileMethodName, ' ', '') 'R=' num2str(diskRadius)
         'H=' num2str(hatThreshold) '.jpg']);
end
if not(showedAndSavedFilteredImage)
    imwrite((filteredImage - min(min(filteredImage))) /
        max(max(filteredImage - min(min(filteredImage))))
        , ['Resultados/' num2str(IMAGE_NUMBER) '/filtrada
```

```
            ' strrep(fileMethodName, ' ', '') 'R=' num2str(
            diskRadius) 'H=' num2str(hatThreshold) 'V='
            num2str(bumpVolumeThreshold) '.jpg']);
        showedAndSavedFilteredImage = 1;
    end

    if and(ALL_IMAGES == 1, ALL_FILES == 1)
        return;
    end

    uiwait(1);

    if OK
        OK = 0;
        return;
    end

end

%%%%%%%%%%%%%
% C L E A R %
%%%%%%%%%%%%%

clear IMAGE_NUMBER;
clear METHOD_NAME;
clear ALL_IMAGES;
clear ALL_FILES;
clear OK;

%%%%%%%%%%%%%%%%%%%%%%%%%
% F U N C T I O N S %
%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function cancel(src, evnt)
global OK;
if evnt.Key == 'escape'
    delete(gcf);
    OK = 1;
end
clear OK;
```

C.10 Filtro morfológico III

```
function filteredImage = IDMorphologicalFilterIII(
    originalImage, diskRadius, hatThreshold,
    bumpVolumeThreshold)

%%%%%%%%%%%%%%%%%%%%
% S E T T I N G S %
%%%%%%%%%%%%%%%%%%%%

global IMAGE_NUMBER;
global METHOD_NAME;
global ALL_IMAGES;
global ALL_FILES;
global OK;
connectedNeighborhood = 8;
showedAll = 0;
showedAndSavedFilteredImage = 0;
filteredImage = originalImage;
fileMethodName = METHOD_NAME;
fileMethodName(findstr(METHOD_NAME, ' ') + 1) = upper(
    METHOD_NAME(findstr(METHOD_NAME, ' ') + 1));
fileMethodName(1) = upper(fileMethodName(1));
```

```
%%%%%%%%%%%%%%%
% F I L T E R %
%%%%%%%%%%%%%%%

filteringProgressBar = waitbar(0, ['Filtrando por '
    METHOD_NAME '...'], 'CreateCancelBtn', 'delete(gcf);
    global OK; OK = 1; clear OK;', 'KeyPressFcn', @cancel);
handle = get(filteringProgressBar, 'children');
set(handle(2), 'string', 'Cancelar');
structuringElement = strel('disk', diskRadius);
closedOpenedImage = imclose(imopen(originalImage,
    structuringElement), structuringElement);
openedClosedImage = imopen(imclose(originalImage,
    structuringElement), structuringElement);
topHatImage = abs(originalImage - closedOpenedImage);
signTopHatImage = sign(originalImage - closedOpenedImage);
bottomHatImage = abs(openedClosedImage - originalImage);
signBottomHatImage = sign(openedClosedImage - originalImage)
    ;
topMask = zeros(size(topHatImage));
topMaskSignalPoints = find(topHatImage > hatThreshold);
topMask(topMaskSignalPoints) = 1;
topBumpMap = bwlabel(topMask, connectedNeighborhood);
topBumpVolume = zeros(max(max(topBumpMap)),1);
bottomMask = zeros(size(bottomHatImage));
bottomMaskSignalPoints = find(bottomHatImage > hatThreshold)
    ;
bottomMask(bottomMaskSignalPoints) = 1;
bottomBumpMap = bwlabel(bottomMask, connectedNeighborhood);
bottomBumpVolume = zeros(max(max(bottomBumpMap)),1);
for x = 1:size(originalImage, 1)
    for y = 1:size(originalImage, 2)
```

```
            if topBumpMap(x, y) > 0
                topBumpVolume(topBumpMap(x, y)) = topBumpVolume(
                    topBumpMap(x, y)) + topHatImage(x, y);
            end
            if bottomBumpMap(x, y) > 0
                bottomBumpVolume(bottomBumpMap(x, y)) =
                    bottomBumpVolume(bottomBumpMap(x, y)) +
                    bottomHatImage(x, y);
            end
        end
        if OK
            OK = 0;
            return;
        else
            if not(OK)
                set(filteringProgressBar, 'name', [num2str(round
                    (100 * (x / (2 * size(originalImage, 1)))))
                    '% concluído']);
            end
            waitbar(x / (2 * size(originalImage, 1)));
        end
    end
    filteredTopHatImage = topHatImage;
    filteredBottomHatImage = bottomHatImage;
    for x = 1:size(originalImage, 1)
        for y = 1:size(originalImage, 2)
            if topBumpMap(x, y) > 0
                if topBumpVolume(topBumpMap(x, y)) <
                    bumpVolumeThreshold
                     filteredTopHatImage(x, y) = 0;
                end
            end
```

```
            if bottomBumpMap(x, y) > 0
                if bottomBumpVolume(bottomBumpMap(x, y)) <
                    bumpVolumeThreshold
                     filteredBottomHatImage(x, y) = 0;
                end
            end
        end
        if OK
            OK = 0;
            return;
        else
            if not(OK)
                set(filteringProgressBar, 'name', [num2str(round
                    (100 * (0.5 + x / (2 * size(originalImage, 1)
                    )))) '% concluído']);
            end
            waitbar(0.5 + x / (2 * size(originalImage, 1)));
        end
    end
    filteredImage = round((filteredTopHatImage.*signTopHatImage
        + closedOpenedImage + openedClosedImage -
        filteredBottomHatImage.*signBottomHatImage)/2);
    figure(1);
    uicontrol('FontName','default', 'BackgroundColor', 'default'
        , 'style', 'text', 'units', 'normalized', 'Position',
        [0.35 0.025 0.30 0.020], 'string', ['Tempo de execução: '
         num2str(floor(toc)) 's']);
    close(filteringProgressBar);
    OK = 0;

    %%%%%%%%%%%%%%%%%%
    % D I S P L A Y %
```

```
%%%%%%%%%%%%%%%%%%

while 1

    if and(ALL_IMAGES == 1, not(showedAll))
        figure;
        imshow(closedOpenedImage, []);
        set(gcf, 'numberTitle', 'off', 'name', ['Fechamento
            da abertura da imagem ' num2str(IMAGE_NUMBER) '
            por ' METHOD_NAME ' com R = ' num2str(diskRadius)
            ]);
        truesize;
        figure;
        imshow(openedClosedImage, []);
        set(gcf, 'numberTitle', 'off', 'name', ['Abertura do
             fechamento da imagem ' num2str(IMAGE_NUMBER) '
            por ' METHOD_NAME ' com R = ' num2str(diskRadius)
            ]);
        truesize;
        figure;
        imshow(topHatImage, []);
        set(gcf, 'numberTitle', 'off', 'name', ['Top hat da
            imagem ' num2str(IMAGE_NUMBER) ' por '
            METHOD_NAME ' com R = ' num2str(diskRadius)]);
        truesize;
        figure;
        imshow(bottomHatImage, []);
        set(gcf, 'numberTitle', 'off', 'name', ['Bottom hat
            da imagem ' num2str(IMAGE_NUMBER) ' por '
            METHOD_NAME ' com R = ' num2str(diskRadius)]);
        truesize;
        figure;
```

```
        imshow(filteredTopHatImage, []);
        set(gcf, 'numberTitle', 'off', 'name', ['Top hat
            filtrado da imagem ' num2str(IMAGE_NUMBER) ' por
            ' METHOD_NAME ' com R = ' num2str(diskRadius) ',
           H = ' num2str(hatThreshold) ' e V = ' num2str(
            bumpVolumeThreshold)]);
        truesize;
        figure;
        imshow(filteredBottomHatImage, []);
        set(gcf, 'numberTitle', 'off', 'name', ['Bottom hat
            filtrado da imagem ' num2str(IMAGE_NUMBER) ' por
            ' METHOD_NAME ' com R = ' num2str(diskRadius) ',
           H = ' num2str(hatThreshold) ' e V = ' num2str(
            bumpVolumeThreshold)]);
        truesize;
        figure;
        imshow(topMask);
        set(gcf, 'numberTitle', 'off', 'name', ['Máscara do
            top hat da imagem ' num2str(IMAGE_NUMBER) ' por '
             METHOD_NAME ' com R = ' num2str(diskRadius) ' e
           H = ' num2str(hatThreshold)]);
        truesize;
        figure;
        imshow(bottomMask);
        set(gcf, 'numberTitle', 'off', 'name', ['Máscara do
            bottom hat da imagem ' num2str(IMAGE_NUMBER) '
            por ' METHOD_NAME ' com R = ' num2str(diskRadius)
             ' e H = ' num2str(hatThreshold)]);
        truesize;
        showedAll = 1;
    end
    if not(showedAndSavedFilteredImage)
```

```
        figure;
        imshow(filteredImage, []);
        set(gcf, 'numberTitle', 'off', 'name', ['Filtragem
            da imagem ' num2str(IMAGE_NUMBER) ' por '
            METHOD_NAME ' com R = ' num2str(diskRadius) ', H
            = ' num2str(hatThreshold) ' e V = ' num2str(
            bumpVolumeThreshold)]);
        truesize;
    end

%%%%%%%%%%
% S A V E %
%%%%%%%%%%

    if ALL_FILES == 1
        imwrite((closedOpenedImage - min(min(
            closedOpenedImage))) / max(max(closedOpenedImage
            - min(min(closedOpenedImage)))), ['Resultados/'
            num2str(IMAGE_NUMBER) '/fechamentoDaAbertura'
            strrep(fileMethodName, ' ', '') 'R=' num2str(
            diskRadius) '.jpg']);
        imwrite((openedClosedImage - min(min(
            openedClosedImage))) / max(max(openedClosedImage
            - min(min(openedClosedImage)))), ['Resultados/'
            num2str(IMAGE_NUMBER) '/aberturaDoFechamento'
            strrep(fileMethodName, ' ', '') 'R=' num2str(
            diskRadius) '.jpg']);
        imwrite((topHatImage - min(min(topHatImage))) / max(
            max(topHatImage - min(min(topHatImage)))), ['
            Resultados/' num2str(IMAGE_NUMBER) '/topHat'
            strrep(fileMethodName, ' ', '') 'R=' num2str(
            diskRadius) '.jpg']);
```

```
        imwrite((bottomHatImage - min(min(bottomHatImage)))
            / max(max(bottomHatImage - min(min(bottomHatImage
            )))), ['Resultados/' num2str(IMAGE_NUMBER) '/
            bottomHat' strrep(fileMethodName, ' ', '') 'R='
            num2str(diskRadius) '.jpg']);
        imwrite((filteredTopHatImage - min(min(
            filteredTopHatImage))) / max(max(
            filteredTopHatImage - min(min(filteredTopHatImage
            )))), ['Resultados/' num2str(IMAGE_NUMBER) '/
            topHatFiltrado' strrep(fileMethodName, ' ', '') '
            R=' num2str(diskRadius) 'H=' num2str(hatThreshold
            ) 'V=' num2str(bumpVolumeThreshold) '.jpg']);
        imwrite((filteredBottomHatImage - min(min(
            filteredBottomHatImage))) / max(max(
            filteredBottomHatImage - min(min(
            filteredBottomHatImage)))), ['Resultados/'
            num2str(IMAGE_NUMBER) '/bottomHatFiltrado' strrep
            (fileMethodName, ' ', '') 'R=' num2str(diskRadius
            ) 'H=' num2str(hatThreshold) 'V=' num2str(
            bumpVolumeThreshold) '.jpg']);
        imwrite(topMask, ['Resultados/' num2str(IMAGE_NUMBER
            ) '/mascaraTopHat' strrep(fileMethodName, ' ', ''
            ) 'R=' num2str(diskRadius) 'H=' num2str(
            hatThreshold) '.jpg']);
        imwrite(bottomMask, ['Resultados/' num2str(
            IMAGE_NUMBER) '/mascaraBottomHat' strrep(
            fileMethodName, ' ', '') 'R=' num2str(diskRadius)
             'H=' num2str(hatThreshold) '.jpg']);
    end
    if not(showedAndSavedFilteredImage)
        imwrite((filteredImage - min(min(filteredImage))) /
            max(max(filteredImage - min(min(filteredImage))))
```

```
            , ['Resultados/' num2str(IMAGE_NUMBER) '/filtrada
            ' strrep(fileMethodName, ' ', '') 'R=' num2str(
            diskRadius) 'H=' num2str(hatThreshold) 'V='
            num2str(bumpVolumeThreshold) '.jpg']);
        showedAndSavedFilteredImage = 1;
    end

    if and(ALL_IMAGES == 1, ALL_FILES == 1)
        return;
    end

    uiwait(1);

    if OK
        OK = 0;
        return;
    end

end

%%%%%%%%%%%%%
% C L E A R %
%%%%%%%%%%%%%

clear IMAGE_NUMBER;
clear METHOD_NAME;
clear ALL_IMAGES;
clear ALL_FILES;
clear OK;

%%%%%%%%%%%%%%%%%%%%%%%
% F U N C T I O N S %
```

```
%%%%%%%%%%%%%%%%%%%%%%

function cancel(src, evnt)
global OK;
if evnt.Key == 'escape'
    delete(gcf);
    OK = 1;
end
clear OK;
```

C.11 Filtro morfológico IV

```
function filteredImage = IDMorphologicalFilterIV(
    originalImage, diskRadius, hatThreshold,
    bumpVolumeThreshold)

%%%%%%%%%%%%%%%%%%%%%%
% S E T T I N G S %
%%%%%%%%%%%%%%%%%%%%%%

global IMAGE_NUMBER;
global METHOD_NAME;
global ALL_IMAGES;
global ALL_FILES;
global OK;
connectedNeighborhood = 8;
showedAll = 0;
showedAndSavedFilteredImage = 0;
filteredImage = originalImage;
fileMethodName = METHOD_NAME;
fileMethodName(findstr(METHOD_NAME, ' ') + 1) = upper(
    METHOD_NAME(findstr(METHOD_NAME, ' ') + 1));
fileMethodName(1) = upper(fileMethodName(1));
```

```
%%%%%%%%%%%%%%%
% F I L T E R %
%%%%%%%%%%%%%%%

filteringProgressBar = waitbar(0, ['Filtrando por '
    METHOD_NAME '...'], 'CreateCancelBtn', 'delete(gcf);
    global OK; OK = 1; clear OK;', 'KeyPressFcn', @cancel);
handle = get(filteringProgressBar, 'children');
set(handle(2), 'string', 'Cancelar');
structuringElement = strel('disk', diskRadius);
closedOpenedImage = imclose(imopen(originalImage,
    structuringElement), structuringElement);
openedClosedImage = imopen(imclose(originalImage,
    structuringElement), structuringElement);
topBottomHatImage = originalImage - round((closedOpenedImage
    + openedClosedImage) / 2);
topHatImage = round((topBottomHatImage + abs(
    topBottomHatImage))/2);
bottomHatImage = round((abs(topBottomHatImage) -
    topBottomHatImage)/2);
topMask = zeros(size(topHatImage));
topMaskSignalPoints = find(topHatImage > hatThreshold);
topMask(topMaskSignalPoints) = 1;
topBumpMap = bwlabel(topMask, connectedNeighborhood);
topBumpVolume = zeros(max(max(topBumpMap)),1);
bottomMask = zeros(size(bottomHatImage));
bottomMaskSignalPoints = find(bottomHatImage > hatThreshold)
    ;
bottomMask(bottomMaskSignalPoints) = 1;
bottomBumpMap = bwlabel(bottomMask, connectedNeighborhood);
bottomBumpVolume = zeros(max(max(bottomBumpMap)),1);
```

```
for x = 1:size(originalImage, 1)
    for y = 1:size(originalImage, 2)
        if topBumpMap(x, y) > 0
            topBumpVolume(topBumpMap(x, y)) = topBumpVolume(
                topBumpMap(x, y)) + topHatImage(x, y);
        end
        if bottomBumpMap(x, y) > 0
            bottomBumpVolume(bottomBumpMap(x, y)) =
                bottomBumpVolume(bottomBumpMap(x, y)) +
                bottomHatImage(x, y);
        end
    end
    if OK
        OK = 0;
        return;
    else
        if not(OK)
            set(filteringProgressBar, 'name', [num2str(round
                (100 * (x / (2 * size(originalImage, 1)))))
                '% concluído']);
        end
        waitbar(x / (2 * size(originalImage, 1)));
    end
end
filteredTopHatImage = topHatImage;
filteredBottomHatImage = bottomHatImage;
for x = 1:size(originalImage, 1)
    for y = 1:size(originalImage, 2)
        if topBumpMap(x, y) > 0
            if topBumpVolume(topBumpMap(x, y)) <
                bumpVolumeThreshold
                 filteredTopHatImage(x, y) = 0;
```

```
            end
        end
        if bottomBumpMap(x, y) > 0
            if bottomBumpVolume(bottomBumpMap(x, y)) <
               bumpVolumeThreshold
                filteredBottomHatImage(x, y) = 0;
            end
        end
    end
    if OK
        OK = 0;
        return;
    else
        if not(OK)
            set(filteringProgressBar, 'name', [num2str(round
               (100 * (0.5 + x / (2 * size(originalImage, 1)
               )))) '% concluído']);
        end
        waitbar(0.5 + x / (2 * size(originalImage, 1)));
    end
end
filteredImage = round(closedOpenedImage/2 +
   filteredTopHatImage + openedClosedImage/2 -
   filteredBottomHatImage);
figure(1);
uicontrol('FontName','default', 'BackgroundColor', 'default'
   , 'style', 'text', 'units', 'normalized', 'Position',
   [0.35 0.025 0.30 0.020], 'string', ['Tempo de execução: '
    num2str(floor(toc)) 's']);
close(filteringProgressBar);
OK = 0;
```

```
%%%%%%%%%%%%%%%%%%
% D I S P L A Y %
%%%%%%%%%%%%%%%%%%

while 1

    if and(ALL_IMAGES == 1, not(showedAll))
        figure;
        imshow(closedOpenedImage, []);
        set(gcf, 'numberTitle', 'off', 'name', ['Fechamento
            da abertura da imagem ' num2str(IMAGE_NUMBER) '
            por ' METHOD_NAME ' com R = ' num2str(diskRadius)
            ]);
        truesize;
        figure;
        imshow(openedClosedImage, []);
        set(gcf, 'numberTitle', 'off', 'name', ['Abertura do
             fechamento da imagem ' num2str(IMAGE_NUMBER) '
            por ' METHOD_NAME ' com R = ' num2str(diskRadius)
            ]);
        truesize;
        figure;
        imshow(topHatImage, []);
        set(gcf, 'numberTitle', 'off', 'name', ['Top hat da
            imagem ' num2str(IMAGE_NUMBER) ' por '
            METHOD_NAME ' com R = ' num2str(diskRadius)]);
        truesize;
        figure;
        imshow(bottomHatImage, []);
        set(gcf, 'numberTitle', 'off', 'name', ['Bottom hat
            da imagem ' num2str(IMAGE_NUMBER) ' por '
            METHOD_NAME ' com R = ' num2str(diskRadius)]);
```

```
truesize;
figure;
imshow(filteredTopHatImage, []);
set(gcf, 'numberTitle', 'off', 'name', ['Top hat
    filtrado da imagem ' num2str(IMAGE_NUMBER) ' por
    ' METHOD_NAME ' com R = ' num2str(diskRadius) ',
    H = ' num2str(hatThreshold) ' e V = ' num2str(
    bumpVolumeThreshold)]);
truesize;
figure;
imshow(filteredBottomHatImage, []);
set(gcf, 'numberTitle', 'off', 'name', ['Bottom hat
    filtrado da imagem ' num2str(IMAGE_NUMBER) ' por
    ' METHOD_NAME ' com R = ' num2str(diskRadius) ',
    H = ' num2str(hatThreshold) ' e V = ' num2str(
    bumpVolumeThreshold)]);
truesize;
figure;
imshow(topMask);
set(gcf, 'numberTitle', 'off', 'name', ['Máscara do
    top hat da imagem ' num2str(IMAGE_NUMBER) ' por '
     METHOD_NAME ' com R = ' num2str(diskRadius) ' e
    H = ' num2str(hatThreshold)]);
truesize;
figure;
imshow(bottomMask);
set(gcf, 'numberTitle', 'off', 'name', ['Máscara do
    bottom hat da imagem ' num2str(IMAGE_NUMBER) '
    por ' METHOD_NAME ' com R = ' num2str(diskRadius)
     ' e H = ' num2str(hatThreshold)]);
truesize;
showedAll = 1;
```

```
end
if not(showedAndSavedFilteredImage)
    figure;
    imshow(filteredImage, []);
    set(gcf, 'numberTitle', 'off', 'name', ['Filtragem
        da imagem ' num2str(IMAGE_NUMBER) ' por '
        METHOD_NAME ' com R = ' num2str(diskRadius) ', H
        = ' num2str(hatThreshold) ' e V = ' num2str(
        bumpVolumeThreshold)]);
    truesize;
end

%%%%%%%%%%
% S A V E %
%%%%%%%%%%

    if ALL_FILES == 1
        imwrite((closedOpenedImage - min(min(
            closedOpenedImage))) / max(max(closedOpenedImage
            - min(min(closedOpenedImage)))), ['Resultados/'
            num2str(IMAGE_NUMBER) '/fechamentoDaAbertura'
            strrep(fileMethodName, ' ', '') 'R=' num2str(
            diskRadius) '.jpg']);
        imwrite((openedClosedImage - min(min(
            openedClosedImage))) / max(max(openedClosedImage
            - min(min(openedClosedImage)))), ['Resultados/'
            num2str(IMAGE_NUMBER) '/aberturaDoFechamento'
            strrep(fileMethodName, ' ', '') 'R=' num2str(
            diskRadius) '.jpg']);
        imwrite((topHatImage - min(min(topHatImage))) / max(
            max(topHatImage - min(min(topHatImage)))), ['
            Resultados/' num2str(IMAGE_NUMBER) '/topHat'
```

```
        strrep(fileMethodName, ' ', '') 'R=' num2str(
        diskRadius) '.jpg']);
    imwrite((bottomHatImage - min(min(bottomHatImage)))
        / max(max(bottomHatImage - min(min(bottomHatImage
        )))), ['Resultados/' num2str(IMAGE_NUMBER) '/
        bottomHat' strrep(fileMethodName, ' ', '') 'R='
        num2str(diskRadius) '.jpg']);
    imwrite((filteredTopHatImage - min(min(
        filteredTopHatImage))) / max(max(
        filteredTopHatImage - min(min(filteredTopHatImage
        )))), ['Resultados/' num2str(IMAGE_NUMBER) '/
        topHatFiltrado' strrep(fileMethodName, ' ', '') '
        R=' num2str(diskRadius) 'H=' num2str(hatThreshold
        ) 'V=' num2str(bumpVolumeThreshold) '.jpg']);
    imwrite((filteredBottomHatImage - min(min(
        filteredBottomHatImage))) / max(max(
        filteredBottomHatImage - min(min(
        filteredBottomHatImage)))), ['Resultados/'
        num2str(IMAGE_NUMBER) '/bottomHatFiltrado' strrep
        (fileMethodName, ' ', '') 'R=' num2str(diskRadius
        ) 'H=' num2str(hatThreshold) 'V=' num2str(
        bumpVolumeThreshold) '.jpg']);
    imwrite(topMask, ['Resultados/' num2str(IMAGE_NUMBER
        ) '/mascaraTopHat' strrep(fileMethodName, ' ', ''
        ) 'R=' num2str(diskRadius) 'H=' num2str(
        hatThreshold) '.jpg']);
    imwrite(bottomMask, ['Resultados/' num2str(
        IMAGE_NUMBER) '/mascaraBottomHat' strrep(
        fileMethodName, ' ', '') 'R=' num2str(diskRadius)
         'H=' num2str(hatThreshold) '.jpg']);
end
if not(showedAndSavedFilteredImage)
```

```
            imwrite((filteredImage - min(min(filteredImage))) /
                max(max(filteredImage - min(min(filteredImage))))
                , ['Resultados/' num2str(IMAGE_NUMBER) '/filtrada
                ' strrep(fileMethodName, ' ', '') 'R=' num2str(
                diskRadius) 'H=' num2str(hatThreshold) 'V='
                num2str(bumpVolumeThreshold) '.jpg']);
            showedAndSavedFilteredImage = 1;
        end

        if and(ALL_IMAGES == 1, ALL_FILES == 1)
            return;
        end

        uiwait(1);

        if OK
            OK = 0;
            return;
        end

end

%%%%%%%%%%%%%
% C L E A R %
%%%%%%%%%%%%%

clear IMAGE_NUMBER;
clear METHOD_NAME;
clear ALL_IMAGES;
clear ALL_FILES;
clear OK;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%
% F U N C T I O N S %
%%%%%%%%%%%%%%%%%%%%%%%%

function cancel(src, evnt)
global OK;
if evnt.Key == 'escape'
    delete(gcf);
    OK = 1;
end
clear OK;
```

Apêndice D

Imagens

Neste apêndice encontram-se as imagens que foram utilizadas nesta dissertação. Mais precisamente, foram utilizadas duas imagens com resolução de 899×674 pixels para análises de resultados subjetivos e duas imagens com resolução de 512×512 pixels para análises de resultados objetivos. As imagens que foram utilizadas para análises de resultados subjetivos podem ser observadas na figura D.1. As imagens que foram utilizadas para análises de resultados ojetivos podem ser observadas na figura D.2. As versões da imagem *Lena* corrompidas por cada um dos 5 tipos de ruído apresentados nas seções 3.1.3, 3.1.4, 3.1.5, 3.1.6 e 3.1.7 podem ser observadas na figura D.3. As versões da imagem *Peppers* corrompidas por cada um dos mesmos 5 tipos de ruído podem ser observadas na figura D.4.

Para se gerar as versões das imagens *Lena* e *Peppers* corrompidas por ruído térmico utilizou-se a equação 3.16 sendo que $\overline{T} = 0$ e $\sigma_T^2 = 10$. Para se gerar as versões das imagens *Lena* e *Peppers* corrompidas por ruído do tipo "sal e pimenta" utilizou-se a equação 3.17 sendo que $\epsilon = 0.05$. Para se gerar as versões das imagens *Lena* e *Peppers* corrompidas por ruído de quantização realizou-se uma quantização das mesmas com passo $\Delta = 16$. Para se gerar as versões das imagens *Lena* e *Peppers* corrompidas por ruído relativo à contagem de fótons utilizou-se a equação 3.19 sendo que para cada ponto (d_1, d_2) foi usado um σ_F^2 diferente, tal que $\sigma_F^2 = I(d_1, d_2)$. Para se gerar as versões das imagens *Lena* e *Peppers* corrompidas por ruído relativo à granulação em fotografias utilizou-se a equação 3.20 sendo que para cada ponto (d_1, d_2) foi usado um p diferente, tal que $p = \frac{I(d_1,d_2)}{L}$ e $L = 255$.

(a) Imagem *Moedas*.

(b) Imagem *Máscaras*.

Figura D.1: Imagens utilizadas para análises de resultados subjetivos.

(a) Imagem *Lena*.

(b) Imagem *Peppers*.

Figura D.2: Imagens utilizadas para análises de resultados objetivos.

(a) Imagem original.

(b) Imagem com ruído térmico.

(c) Imagem com ruído do tipo "sal e pimenta".

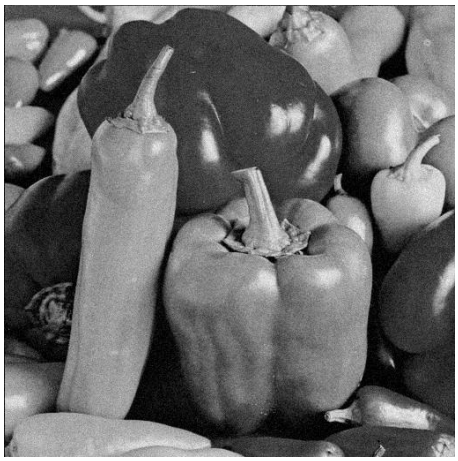
(d) Imagem com ruído de quantização.

(e) Imagem com ruído relativo à contagem de fótons.

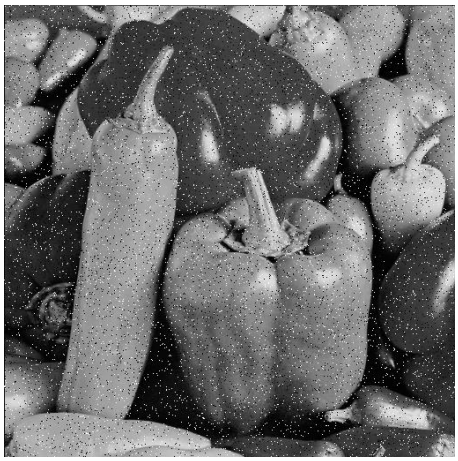
(f) Imagem com ruído relativo à granulação em fotografias.

Figura D.3: Versões da imagem *Lena* corrompidas por cada um dos 5 tipos de ruído apresentados.

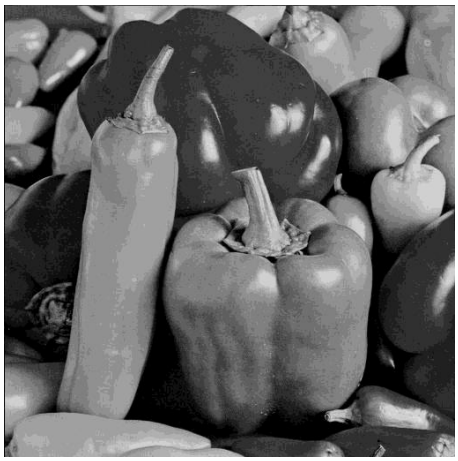
(a) Imagem original.



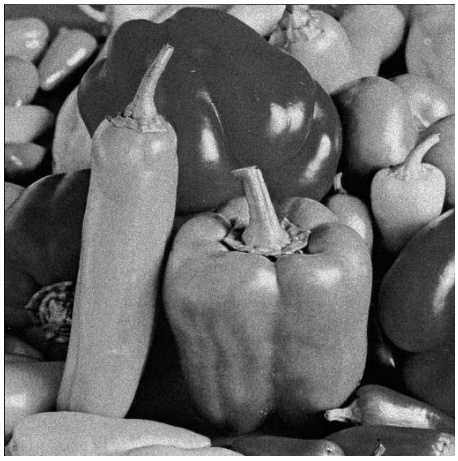
(b) Imagem com ruído térmico.



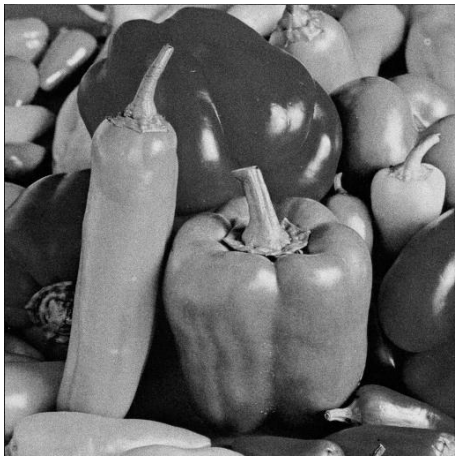
(c) Imagem com ruído do tipo "sal e pimenta".



(d) Imagem com ruído de quantização.



(e) Imagem com ruído relativo à contagem de fótons.



(f) Imagem com ruído relativo à granulação em fotografias.

Figura D.4: Versões da imagem *Peppers* corrompidas por cada um dos 5 tipos de ruído apresentados.