Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e Computação
Departamento de Comunicações

Análise da Utilização de Filtros Lineares e Não-Lineares na Recuperação de Imagens Degradadas

Alessandra Boaventura Rabelo

Orientador: Prof. Dr. Luiz César Martini

Campinas – SP – Brasil
Fevereiro, 2002

Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e Computação
Departamento de Comunicações

Análise da Utilização de Filtros Lineares e Não-Lineares na Recuperação de Imagens Degradadas

Alessandra Boaventura Rabelo

Orientador: Prof. Dr. Luiz César Martini

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação (FEEC) da Universidade Estadual de Campinas (UNICAMP), como parte dos requisitos exigidos para obtenção do título de Mestre em Engenharia Elétrica.

Área de Concentração:
Telecomunicações e Telemática

Campinas – SP – Brasil
Fevereiro, 2002

Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e Computação
Departamento de Comunicações

Análise da Utilização de Filtros Lineares e Não-Lineares na Recuperação de Imagens Degradadas

Autora: Alessandra Boaventura Rabelo

Dissertação de Mestrado defendida e aprovada em vinte e seis de fevereiro de dois mil e dois, pela banca examinadora constituída pelos professores:

Prof. Dr. Luiz César Martini
DECOM/FEEC/UNICAMP

Profa. Dra. Nina Sumiko Tomita Hirata
IME/USP

Prof. Dr. Yuzo Iano
DECOM/FEEC/UNICAMP

Prof. Dr. João Baptista T. Yabu-uti
DECOM/FEEC/UNICAMP

Campinas – SP – Brasil
Fevereiro, 2002

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

R112a

Rabelo, Alessandra Boaventura
Análise da utilização de filtros lineares e não-lineares na recuperação de imagens degradadas / --Campinas, SP: [s.n.], 2002.

Orientador: Luiz César Martini.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Processamento de sinais – Técnicas digitais. 2. Processamento de imagens – Técnicas digitais. 3. Filtros elétricos digitais. I. Martini, Luiz César. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

A minha mãe, meu pai, irmãos, e demais familiares e amigos de quem inúmeras vezes recebi o incentivo necessário para o sucesso de vida acadêmica. Meus amigos, muito obrigado pela ajuda preciosa.

AGRADECIMENTOS

Agradeço ao meu orientador Dr. Luiz César Martini, pela orientação, por sua competência, dedicação e amizade que foram importantes para o êxito do meu trabalho. Agradeço especialmente o grande apoio que me deu durante esses anos.

Agradeço aos demais participantes da banca examinadora Profa. Dra. Nina Sumiko Tomita Hirata, Prof. Dr. Yuzo Iano e Prof. Dr. João Baptista T. Yabu-uti.

Agradeço a minha família pela força, apoio, carinho e incentivo.

A todos os colegas e amigos, particularmente o Wilson, Romis, Lucia e Sonia, e muitos outros pela ajuda, amizade e convívio diário.

Agradeço a todos que, de uma forma direta ou indireta, contribuíram e conviveram comigo nestes anos de vida acadêmica.

Meu agradecimento aos meus professores do Centro Universitário do Triângulo – UNIT - Uberlândia, pelo apoio e incentivo que me deram para que eu pudesse iniciar este mestrado e a todos que de uma forma ou outra acreditaram em mim.

A todos os amigos e funcionários da FEEC, em especial a Lúcia, Tatiane, Nelson, Noemia e todos da CPG-FEEC, pela preciosa ajuda.

Agradeço ao Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq - pelo suporte financeiro concedido a este trabalho desde o início.

RESUMO

Nesta tese de conclusão de mestrado apresentaremos algoritmos, simulações e resultados da utilização de filtros na restauração de imagens degradadas por algum tipo de ruído. Para isto utilizamos filtros lineares e não-lineares. Os filtros implementados foram: Filtro condicional de média aritmética (CMA), Filtro mediano ponderado central (CWM), Filtro mediano (MA), Filtro média aritmética ponderada (MAP), Filtro condicional mediano (RCM), Filtro condicional seletor de posição (RCRS), Filtro de ordem estatística (OS), Filtro de ordem estatística ponderada simples (SWOS) e Filtro de ordem estatística ponderada (WOS). A função WIENER2 do Matlab também foi utilizada. Apresentaremos algoritmos, simulações computacionais e resultados que ilustram o comportamento e desempenho dos filtros acima citados. Mostramos através de exemplos e resultados que o desempenho dos filtros não-lineares muitas vezes é bem superior ao dos lineares. Finalmente apresentamos conclusões e propostas para trabalhos futuros.

ABSTRACT

In this mastership conclusion thesis we present some algorithms, simulations and results from filters utilization for the restoration of images that were degraded by some kind of noise. To do that, we used linear and non-linear filters. The implemented filters were: Conditional Median Arithmetic Filter (CMA), Center Weighted Median Filter (CWM), Median Filtering (MA), Arithmetic Weighed Median Filter (MAP), Rank Conditioned Median Filter (RCM), Rank Conditioned Rank Selection Filter (RCRS), Order Statistic Filter (OS), Simply Weighted Order Statistic Filter (SWOS) and Weighted Order Statistic Filter (WOS). The Matlab WIENER2 function was also used. We will present algorithms, computer simulations and results that show the behavior and the performance of the mentioned filters. We show through some examples and results that non-linear filters performance is in many cases much higher than linear ones. At the end we present some conclusions and proposals for future works.

INDICE

CAPÍTULO 1 – INTRODUÇÃO 1
1.1 Motivação 1
1.2 Origem do Interesse em Processamento de Imagens Digitais 1
1.3 Objetivo Global 6
1.4 Descrição Sucinta dos Capítulos 7

CAPÍTULO 2 – CONCEITOS BÁSICOS 9
2.1 Imagens Digitais 9
2.2 Modelo de Imagem 12
2.3 Amostragem e Quantização 14
2.4 Relacionamentos Básicos entre Pixels 19
2.4.1 Vizinhos de um Pixel 19
2.5 Processamento de Imagens 20
2.5.1 Restauração 22
2.5.2 Segmentação 22
2.5.3.1 Operações Morfológicas 22
2.5.3 Extração de Atributos 25
2.5.4 Classificação 26
2.8 Processamento Baseado em Áreas ou Regiões da Imagem 26

CAPÍTULO 3 – REALCE DE IMAGENS 29
3.1 Filtragem Espacial 29

3.1.1 Notação Básica 30
3.1.2 Filtragem Espacial Linear 31
3.1.3 Filtragem Espacial Não-Linear 35
3.2 Dados e Funções Auxiliares 36
3.3 Filtro de Wiener 39
3.4 Filtro de Média Aritmética – MA 40
3.5 Filtro de Média Aritmética Ponderada – MAP 41
3.6 Filtro Condicional de Média Aritmética – CMA 44
3.7 Filtro Mediano 46
3.8 Filtro Mediano Ponderado Central – CWM 47
3.9 Filtro de Ordem Estatística – OS 49
3.10 Filtro de Ordem Estatística Ponderada Simples – SWOS 52
3.11 Filtro de Ordem Estatística Ponderada – WOS 53
3.12 Filtro Condicional Mediano – RCM 55
3.13 Filtro de Condicional Seletor de Posição – RCRS 57

CAPÍTULO 4 – APRESENTAÇÃO E DISCUSSÃO DOS RESULTADOS 69
4.1 Gráficos de desempenho dos Filtros implementados 71
4.1.1 Filtros e imagens com ruído sal e pimenta e gaussiano 71
4.1.2 Filtro RCRS, com diferentes janelas 78
4.1.3 Filtro Wiener, com diferentes janelas 81
4.1.4 Filtros, considerando diferentes janelas e ruído sal e pimenta 82
4.1.5 Filtros, considerando diferentes janelas e gaussiano 89
4.1.6 Imagens boat e lena 94
4.1.7 Imagens montage e mountain 100
4.1.8 Imagens olhodigital e flor 102
4.1.9 Imagens cameraman e kids 104
4.1.10 Gráficos de erro ISNR para imagens diferentes 106
4.2 Desempenho visual dos Filtros 112
4.2.1 Imagem boat, erro MAE 112
4.2.2 Imagem montage, erro ISNR 116
4.3 Algumas Conclusões 130

CAPÍTULO 5 – CONSIDERAÇÕES FINAIS E SUGESTÕES DE TRABALHOS 133

INTERNET LINKS 137

BIBLIOGRAFIA 139

ANEXO 1 – EXEMPLO NUMÉRICO DO PROBLEMA DE BORDAS 143

ANEXO 2 - FUNÇÃO IMNOISE DO MATLAB 149

ANEXO 3 - FUNÇÃO WIENER2 DO MATLAB 161

ANEXO 4 – IMAGENS ORIGINAIS UTILIZADAS 165

ANEXO 5 – SOPT PARA O FILTRO RCRS 173

LISTA DE FIGURAS

Figura 2.1 – Conversão dos eixos para representação de imagens digitais 10

Figura 2.2 – Elementos fundamentais em processamento de imagens digitais 11

Figura 2.3 – Efeito da redução da resolução espacial de uma imagem 16

Figura 2. 4 – Efeito diminuição níveis de cinza de uma imagem 18

Figura 2.5 – Vizinhos do pixel 20

Figura 2.6 – Seqüência tradicional para processamento de imagens 21

Figura 2.7 – Erosão 23

Figura 2.8 – Dilatação 24

Figura 2.9 – Abertura 24

Figura 2.10 – Fechamento 25

Figura 2.11 – Exemplo numérico de vizinhança 27

Figura 3.1 – (a) Subárea da imagem; (b) máscara 3x3 32

Figura 3.2 – Imagem recuperada por um filtro Passa-Baixas 34

Figura 3.3 – Imagem recuperada por um filtro Passa-Altas 35

Figura 3.4 – Imagem desejada 38

Figura 3.5 – Imagem corrompida 38

Figura 3.6 – Saída do filtro Wiener 39

Figura 3.7 – Exemplo numérico filtro MA 40

Figura 3.8 – Saída do filtro MA 41

Figura 3.9 – Exemplo numérico filtro MAP 42

Figura 3.10 – Saída do filtro MAP 43

Figura 3.11 – Exemplo numérico filtro CMA 45

Figura 3.12 – Saída do filtro CMA 45

Figura 3.13 – Exemplo numérico filtro Mediano 46

Figura 3.14 – Saída do filtro Mediano 47

Figura 3.15 – Exemplo numérico filtro CWM 48

Figura 3.16 – Saída do filtro CWM 49

Figura 3.17 – Exemplo numérico filtro OS 50

Figura 3.18 – Saída do filtro OS 51

Figura 3.19 – Exemplo numérico filtro SWOS 52

Figura 3.20 – Saída do filtro SWOS 53

Figura 3.21 – Exemplo numérico filtro WOS 54

Figura 3.22 – Saída do filtro WOS 55

Figura 3.23 – Exemplo numérico filtro RCM 56

Figura 3.24 – Saída do filtro RCM 57

Figura 3.25 – Função Sopt (ruído sal e pimenta) 63

Figura 3.26 – Função Sopt (ruído gaussiano) 63

Figura 3.27 – Imagens recuperada e original 66

Figura 3.28 – Exemplo numérico filtro RCRS 67

Figura 3.29 – Saída do filtro RCRS 67

Figura 4.1 – Vetores de erro, ruído sal e pimenta, máscara 3x3 72

Figura 4.2 – Vetores de erro, ruído gaussiano, máscara 3x3 73

Figura 4.3 – Vetores de erro, ruído gaussiano, máscara 3x3 74

Figura 4.4 – Vetores de erro, ruído sal e pimenta, máscara 5x5 75

Figura 4.5 – Vetores de erro, ruído gaussiano, máscara 5x5 76

Figura 4.6 – Vetores de erro, ruído sal e pimenta, máscaras 7x7 e 9x9 77

Figura 4.7 – Vetores de erro, ruído gaussiano, máscaras 7x7 e 9x9 78

Figura 4.8 – Vetores de erro, ruído sal e pimenta, janelas 3x3, 5x5, 7x7, 9x9 79

Figura 4.9 – Vetores de tempo, ruído sal e pimenta, janelas 3x3, 5x5, 7x7, 9x9 80

Figura 4.10 – Vetores de erro e tempo, ruído gaussiano 80

Figura 4.11 – Vetores de erro e tempo, Wiener, sal e pimenta 81

Figura 4.12 – Vetores de erro e tempo, sal e pimenta e RCRS 82

Figura 4.13 – Vetores de erro e tempo, sal e pimenta e OS 83

Figura 4.14 – Vetores de erro e tempo, sal e pimenta e CMA 83

Figura 4.15 – Vetores de erro e tempo, sal e pimenta e Mediano 84

Figura 4.16 – Vetores de erro e tempo, sal e pimenta e MAP 85

Figura 4.17 – Vetores de erro e tempo, sal e pimenta e CWM 85

Figura 4.18 – Vetores de erro e tempo, sal e pimenta e RCM 86

Figura 4.19 – Vetores de erro e tempo, sal e pimenta e SWOS 87

Figura 4.20 – Vetores de erro e tempo, sal e pimenta e WOS 87

Figura 4.21 – Vetores de erro e tempo, sal e pimenta e Wiener 88

Figura 4.22 – Vetores de erro e tempo, gaussiano 92

Figura 4.23 – Vetores de erro, lena x boat, máscara 5x5 95

Figura 4.24 – Vetores de tempo, lena x boat, máscara 5x5 96

Figura 4.25 – Vetores de erro, lena x boat, RCRS e Médio, 7x7 97

Figura 4.26 – Vetores de tempo, lena x boat, RCRS e Médio, 7x7 98

Figura 4.27 – Vetores de erro, lena x boat, MAP, Wiener e OS 99

Figura 4.28 – Vetores de tempo, lena x boat, MAP, Wiener e OS 100

Figura 4.29 – Vetores de erro e tempo, montage e mountain, sal e pimenta 101

Figura 4.30 – Vetores de erro e tempo, montage e mountain, gaussiano 102

Figura 4.31 – Vetores de erro e tempo, olhodigital e flor, gaussiano 102

Figura 4.32 – Vetores de erro e tempo, olhodigital e flor, sal e pimenta 103

Figura 4.33 – Vetores de erro e tempo, cameraman e kids, sal e pimenta 104

Figura 4.34 – Vetores de erro e tempo, cameraman e kids, gaussiano 105

Figura 4.35 – Ruído sal e pimenta, imagens montage e couple 106

Figura 4.36 – Ruído gaussiano, imagens montage e couple 107

Figura 4.37 – Ruído sal e pimenta, janelas 3x3 e 7x7, couple 108

Figura 4.38 – Ruído gaussiano, janelas 3x3 e 7x7, couple 109

Figura 4.39 – Ruído sal e pimenta e gaussiano, janela 3x3, couple 110

Figura 4.40 – Ruído sal e pimenta e gaussiano, janela 7x7, couple 111

Figura 4.41 – Imagem Corrompida boat, 512x512 112

Figura 4.42 – Imagens na saída dos filtros 114

Figura 4.43 – Vetores erro e tempo gasto 115

Figura 4.44 – Imagem montage, sal e pimenta 5x5 118

Figura 4.45 – Imagem montage, gaussiano 5x5 120

Figura 4.46 – Imagem montage, sal e pimenta 3x3 122

Figura 4.47 – Imagem montage, gaussiano 3x3 125

Figura 4.48 – Ruído sal e pimenta e gaussiano, janela 5x5 126

Figura 4.49 – Ruído sal e pimenta e gaussiano, janela 3x3 127

Figura 4.50 – Ruído sal e pimenta, janelas 3x3 e 5x5 128

Figura 4.51 – Ruído gaussiano, janelas 3x3 e 5x5 129

Figura A1.1 – Imagem utilizada 144

Figura A1.2 – Pixels da imagem observada 144

Figura A1.3 – Pixels de acordo com vizinhança 145

Figura A1.4 – Exemplo, zeros fora da imagem 146

Figura A1.5 – Exemplo, repetir linhas e colunas 147

Figura A1.6 – Exemplo, repetir linhas e colunas circular 147

Figura A4.1 – Imagem original lena.tiff 165

Figura A4.2 – Imagem original montage.gif 166

Figura A4.3 – Imagem original mountain.gif 166

Figura A4.4 – Imagem original olhodigital.tiff 167

Figura A4.5 – Imagem original flor.jpg 167

Figura A4.6 – Imagem original spine.tiff 168

Figura A4.7 – Imagem original tire.tiff 168

Figura A4.8 – Imagem original boat.gif 169

Figura A4.9 – Imagem original cameraman.tiff 169

Figura A4.10 – Imagem original kids.tiff 170

Figura A4.11 – Imagem original tress.tiff 170

Figura A4.12 – Imagem original couple.tiff 171

Figura A4.13 – Imagem original moon.tiff 171

Figura A4.14 – Imagem original Elaine.tiff 172

Figura A5.1 – Imagem original lena.tiff 173

Figura A5.2 – Sopt, ruído SalPimenta, 3x3 174

Figura A5.3 – Sopt, ruído SalPimenta, 5x5 174

Figura A5.4 – Sopt, ruído SalPimenta, 7x7 175

Figura A5.5 – Sopt, ruído SalPimenta, 9x9 175

Figura A5.6 – Sopt, ruído gaussiano, 3x3 176

Figura A5.7 – Sopt, ruído gaussiano, 5x5 176

Figura A5.8 – Sopt, ruído gaussiano, 7x7 177

Figura A5.9 – Sopt, ruído gaussiano, 9x9 177

LISTA DE TABELAS

Tabela 3.1 – Filtros Passa-Baixas espaciais de tamanhos variados 33

Tabela 3.2 – Filtros Passa-Altas espaciais 34

Tabela 3.3 – Processamento para filtro MAP 42

Tabela 3.4 – Imagem 512x512 58

Tabela 3.5 – Fatia das imagens de entrada e desejada 59

Tabela 3.6 – Vetores observação, observação ordenado e saída desejada 60

Tabela 3.7 – Vetor diferença e posição da amostra observada 61

Tabela 3.8 – Vetor erro cumulativo e valor mínimo 61

Tabela 3.9 – Vetor ordenado, posição amostra e erro mínimo 64

Tabela 3.11 – Valor numérico imagens recuperada e original 65

Tabela 4.1 – Intervalos de erro, ruído gaussiano 93

Tabela 4.2 – Intervalos de tempo, ruído gaussiano 94

CAPÍTULO 1

INTRODUÇÃO

Neste capítulo é apresentada a motivação que nos levou ao desenvolvimento desta dissertação de mestrado, o objetivo da mesma e uma breve descrição dos próximos capítulos.

1.1 Motivação

Encontrei em minha pesquisa uma frase que explica tudo em poucas palavras: "Uma imagem vale mais que dez mil palavras" [Autor Anônimo]. Após pesquisar em papers, livros e internet, a pequena quantidade de material no Brasil, a respeito de filtragem não linear fez com que crescesse o interesse no assunto.

1.2 Origem do Interesse em Processamento de Imagens Digitais

O acelerado processo de globalização e informatização observado nos últimos tempos tem nos dado um motivo a mais no que se refere a áreas como processamento de

imagens digitais principalmente quando se trata da melhoria da informação visual para interpretação humana e o processamento de dados de cenas para percepção automática através de máquinas. As imagens, a cada dia que passa, vem representando um papel cada vez maior na vida das pessoas, em todo o mundo. Temos aplicações relacionado à representação gráfica, que nos auxiliam no entendimento e registro de fatos para recordação, aplicações práticas, que incluem automação de tarefas repetitivas e/ou perigosas, e até mesmo a manifestação artística de sentimentos que raramente conseguimos expressar com palavras. Com o advento da tecnologia da informação, surgiram ferramentas poderosas para o tratamento das imagens. Ferramentas estas que vem cumprindo tão importante papel no avanço científico-tecnológico, além de fornecer a capacidade de criação de novas possibilidades visuais.

No Brasil, cientistas e artistas começaram a se interessar pela área de processamento de imagens por computador, na década de 60 e o interesse pelas muitas áreas relacionadas cresceu intensamente desde então. Atualmente temos cursos de graduação, mestrado, doutorado e pós-doutorado para formação de profissionais nesta área, temos também um grande número de encontros nacionais científicos que incluem tópicos relacionados a processamento de imagens digitais (alguns com "status" de internacionais) [Gonzalez, 2000] [1].

Uma das primeiras aplicações de técnicas de processamento de imagens foi desenvolvida para o melhoramento de imagens digitalizadas para jornais, enviadas por meio de cabo submarino de Londres para *New York*. No início dos anos 20 introduziu-se o sistema *Bartlane* de transmissão de imagens via cabo, reduzindo o tempo de transporte de imagens de uma semana para menos de três horas. Era usado um aparelho, especializado para impressão, que codificava a imagem para transmissão a cabo, e as mesmas eram reconstruídas no terminal receptor. Atualmente esta "demora" seria inconcebível, porém na época funcionava muito bem [Gonzalez, 1992, p. 1].

Alguns problemas iniciais enfrentados na tentativa de melhorar a qualidade visual dessas primeiras figuras digitais estavam relacionados com a seleção dos

[1] Prefácio à edição brasileira

processos de impressão e à distribuição dos níveis de cinza. Troca-se o método de impressão por uma técnica baseada na reprodução fotográfica feita a partir de fitas perfuradas feitas no terminal receptor telegráfico.

O primeiro sistema *Bartlane* era capaz de codificar cinco níveis de cinza. Em 1929 esta capacidade foi aumentada para quinze. Durante este período, a introdução de um sistema para revelação de uma chapa de filme através de feixes luminosos modulados por fita de figura codificada melhorou de forma considerável o processo de reprodução.

Melhoramentos nos métodos de processamento para transmissão de figuras digitais continuaram a ser feitos ao longo dos anos. Porém, o surgimento dos computadores digitais de grande porte bem como o programa espacial é que vão nos chamar a atenção para o potencial do processamento de imagens. O emprego de técnicas de computação para o melhoramento de imagens produzidas por uma sonda espacial iniciou-se no *Jet Propulsion Laboratory (Pasadena, Califórnia)*, em 1964, quando imagens da Lua, transmitidas pelo Ranger 7, foram processadas por um computador para corrigir vários tipos de distorção de imagem inerentes à câmera de televisão a bordo. Essas técnicas serviram como base para métodos melhorados de realce e restauração de imagens das missões *Surveyor* para a Lua, a série *Mariner* de missões para Marte, os vôos tripulados da Apolo para a Lua dentre outros [Gonzalez, 1992, p. 2].

Desde 1964 até os dias de hoje a área de processamento de imagens digitais vem crescendo vigorosamente. Procedimentos para realce e restauração de imagens são usados para processar imagens de objetos irrecuperáveis ou resultados experimentais muito caros para repetição. Nos últimos anos o interesse em morfologia matemática, redes neurais, processamento de imagens coloridas, compressão de imagens, reconhecimento de imagens e sistemas de análise baseadas em conhecimento tem crescido espantosamente. Qualquer área onde se possa capturar uma imagem e dela obter resultados é uma área onde pode ser usado um sistema de processamento de imagens. Devido a esta vasta amplitude de áreas de aplicação, sistemas de processamento de imagens são, à priori, interdisciplinares. Abaixo temos alguns exemplos ilustrativos do que acabamos de dizer.

- Sensoriamento Remoto

  Fotografias, enviadas por naves espaciais e satélites em torno da terra, necessitam ser avaliadas e catalogadas com fidelidade e eficiência. Fotos de países podem ser usadas para a obtenção rápida de mapas cartográficos e estudos geográficos, tais como análise do solo e da safra agrícola.

- Microscopia.

  Análise de imagens provindas de microscópios óticos ou eletrônicos, em áreas que variam desde a medicina até a metalurgia, identificando e classificando partículas.

- Física

  Em física e áreas afins, técnicas computacionais rotineiramente realçam imagens de experimentos em áreas como plasmas de alta energia e microscopia eletrônica.

- Medicina.

  Imagens radiográficas de baixa qualidade podem ser realçadas para a obtenção de um diagnóstico mais preciso. Com a introdução da tomografia computadorizada e da ressonância magnética, imagens volumétricas são utilizadas para diversas aplicações, tais como planejamento cirúrgico e diagnóstico.

- Manutenção de obras de arte.

  Evitar a degradação com o tempo de obras de arte não é um processo simples, mas possuindo uma imagem da obra original que pode ser processada para ajudar no processo de restauração para se obter um retrato mais próximo (parecido) com o original. Além disso, existem estudos para a verificação da autenticidade de pinturas através da análise das pinceladas de cada pintor. Em arqueologia, métodos de processamento de imagens têm

restaurado com sucesso figuras fotografadas borradas, que eram os únicos registros disponíveis de artefatos raros perdidos ou danificados.

- Identificação de impressões digitais.

  Através de adequada classificação e armazenamento pode-se identificar uma impressão digital usando um banco de dados associado a um banco de imagens.

- Armazenamento de documentos.

  É comum que grandes quantidades de informação fiquem "encalhadas" devido à dificuldade de acesso a quilos de papel que ficam engavetados e deteriorando. Estes textos podem ser transferidos para um sistema de hipertexto que permita uma consulta rápida e dinâmica. Para tanto, é necessário capturar as imagens dos textos e depois realizar um reconhecimento de caracteres. Segue então um processo de integração ao hipertexto. Muitas vezes estes textos são documentos oficiais que precisam manter suas características, portanto as próprias imagens são armazenadas.

- Medidas de velocidade de escoamento de fluidos.

  Através de partículas suspensas no líquido, pode-se iluminar um corte no escoamento através de um plano de luz, e utilizando uma exposição lenta obtém-se uma fotografia com traços correspondentes às trajetórias das diversas partículas. A partir dos comprimentos dos traços pode-se obter uma medida aproximada da velocidade do fluido.

- Controle de qualidade.

  A aplicação de processamento de imagens na área de controle de qualidade é muito vasta. Praticamente qualquer processo industrial que necessite de alguma monitoração ótica ou visual pode ser automatizado com vantagens. O que pode inviabilizar o processo é o grau de dificuldade de interpretar cada imagem, assim como em qualquer outra aplicação de análise de imagens.

Uma aplicação comum é a verificação de placas de circuito impresso, que possuem uma geometria simples e regras fixas.

- Outras áreas não menos importantes: Astronomia, Fotografia, Vídeo, Efeitos Especiais, etc.

Estes são exemplos da utilização de processamento de imagens em diversas áreas. Já mencionamos que a segunda maior aplicação de técnicas de processamento de imagens se refere a problemas relacionados à percepção por máquina. Neste caso o interesse se concentra na extração de informações da imagem de forma adequada para o processamento computacional. Esta informação, freqüentemente apresenta pouca semelhança com as características utilizadas pelo homem na interpretação do conteúdo de uma imagem. Exemplos do tipo de informação usado em percepção por máquinas são os momentos estatísticos, os coeficientes da transformada de Fourier e medidas de distância multidimensionais.

Problemas típicos de percepção por máquina, que rotineiramente usam técnicas de processamento de imagens, são o reconhecimento automático de caracteres, visão computacional industrial para a montagem e inspeção de produtos, reconhecimento militar, processamento automático de impressões digitais, análise de resultados de raios X e amostras de sangue em tela, processamento de imagens aéreas, de satélites para previsão de tempo e monitoração de plantio [Gonzalez, 1992, p. 5-6].

1.3 Objetivo Global

O objetivo desta tese de mestrado é uma análise da utilização de filtragem (linear e não-linear) na recuperação de imagens degradadas por algum tipo de ruído.

A crescente necessidade de transmissão de imagens e o fato de que, por diversos fatores, a mesma não chega ao seu destino exatamente igual a que foi transmitida, serviu de estímulo à realização deste trabalho, visando contribuir com estudo de métodos usados na restauração destas imagens.

1.4 Descrição Sucinta dos Capítulos

No capítulo 1, teremos uma breve introdução, bem como motivação para o desenvolvimento do trabalho.

No capítulo 2, exporemos vários conceitos relacionados a imagens digitais e algumas notações utilizadas ao longo do desenvolvimento desta dissertação de mestrado.

No capítulo 3 descreveremos o processo de filtragem na recuperação de imagens degradadas e apresentamos os filtros utilizados neste trabalho, ilustrando suas aplicações em uma imagem por ruído do tipo sal e pimenta.

No capítulo 4, apresentaremos os resultados obtidos e uma comparação entre os filtros.

No capítulo 5, teremos as considerações finais e sugestões para trabalhos futuros.

CAPÍTULO 2

CONCEITOS BÁSICOS

Neste capítulo são apresentados alguns conceitos básicos relacionados a Processamento de Imagens Digitais, uma visão geral da área, algumas notações e conceitos matemáticos utilizados ao longo do desenvolvimento desta dissertação de mestrado.

Antes disto, falaremos sobre as especificações do computador utilizado, trata-se de um *Pentium* III, 800 MHz, memória 128 MB, barramento PC-100 e *Microsoft Windows 2000 Professional.*

2.1 Imagens Digitais

Quando nos referimos a uma imagem monocromática ou simplesmente imagem, queremos com isto considerar uma função bidimensional de intensidade da luz f(x, y), onde x e y são as coordenadas espaciais e o valor de f em qualquer ponto (x, y) é proporcional ao brilho (níveis de cinza) da imagem naquele ponto. Muitas vezes o uso de um terceiro eixo, que representa o brilho, torna-se útil.

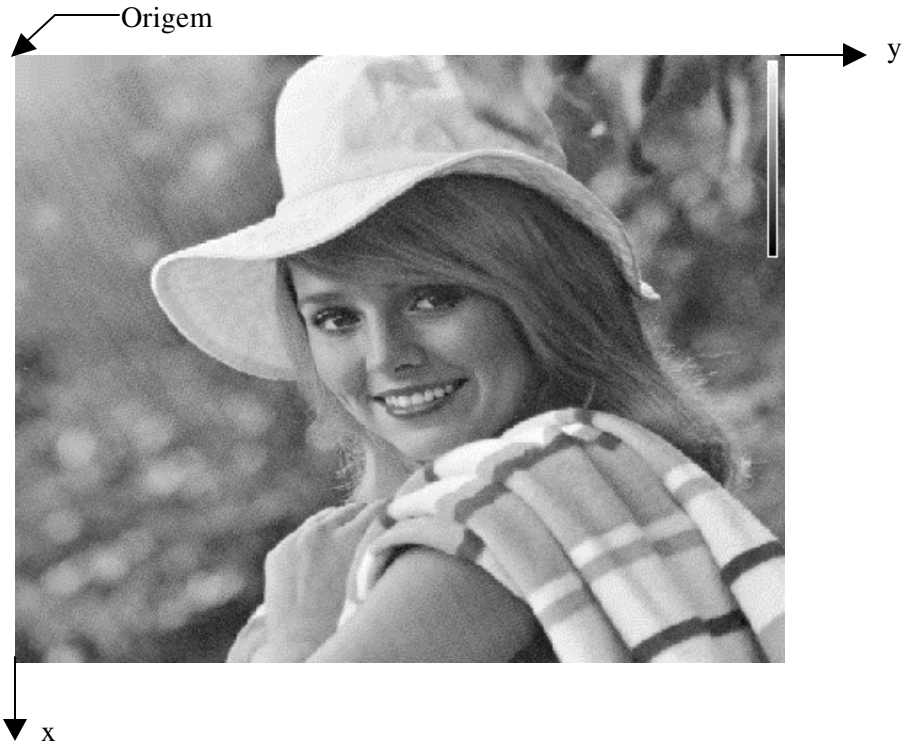


Figura 2.1 Convenção dos eixos para representação de imagens digitais.

A Figura 2.1 mostra a convenção dos eixos (x linhas e y colunas), que geralmente é usada quando se trata de processamento de imagens por computador. No canto direito da figura, podemos visualizar uma barra, que nos mostra uma escala dos níveis de cinza da figura. Esta figura apresenta 256 níveis de cinza. Outra convenção utilizada é atribuir proporcionalmente valores mais altos para áreas de maior brilho e valores menores para áreas de menor brilho, o que faz a altura dos componentes da figura proporcional ao brilho correspondente na imagem.

Imagem digital é uma imagem f(x, y) discretizada tanto em coordenadas espaciais quanto em brilho. A mesma pode ser considerada como sendo uma matriz cujos índices de linhas e de colunas identificam um ponto na imagem e o correspondente valor do elemento da matriz identifica o nível de cinza naquele ponto. Os elementos dessa matriz são os elementos da imagem ou elementos da figura, "pixels" ou "pels", estes dois últimos, uma abreviatura de "*picture elements*" (elementos da figura).

As dimensões da figura (tamanho) podem variar de acordo com a aplicação. Porém matrizes quadradas com níveis de cinza que sejam potências inteiras de dois tem suas vantagens. Podemos citar como exemplo de um tamanho típico, comparável em qualidade de imagem com aquela de uma TV preto e branco, uma matriz 512x512, com 128 níveis de cinza [Gonzalez, 1992, p. 7].

O processamento de imagens digitais abrange uma ampla escala de fundamentos, hardware e software, que são abordados nas seções seguintes.

Para manipulação de uma imagem necessitamos basicamente de um computador e dispositivos especiais para entrada e saída, ou seja, um digitalizador de imagens e um mecanismo de visualização (*display*). Alguns elementos são fundamentais e gerais quando se trata de processamento de imagens digitais, são eles: aquisição (captura), armazenamento, processamento e exibição (visualização) de imagens. A Fig. 2.2 pode nos dar uma visão melhor a respeito desses elementos.

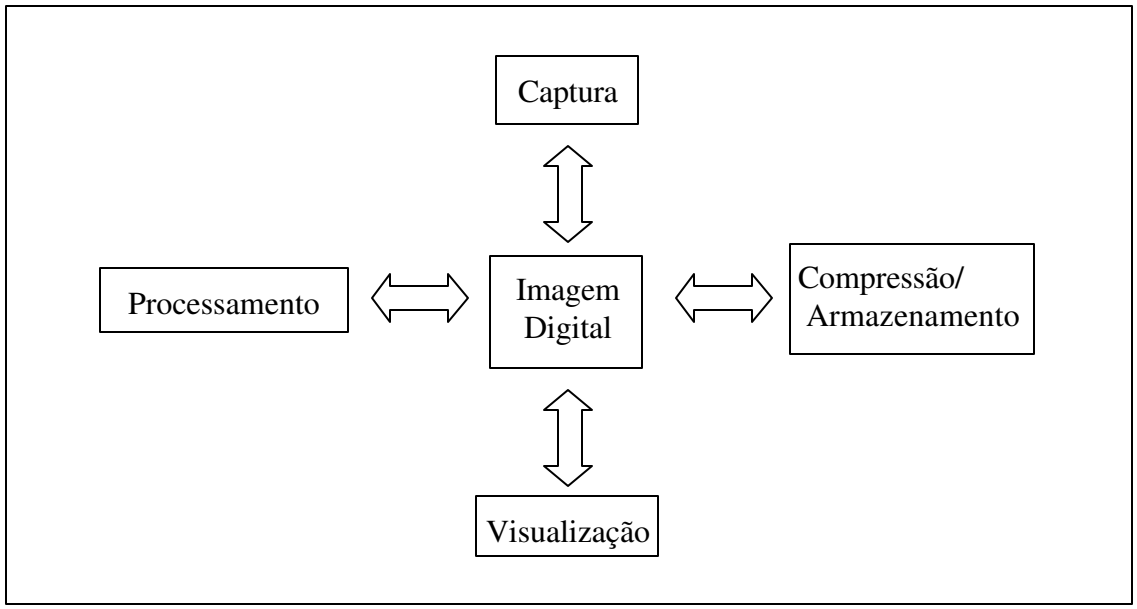


Figura 2.2 Elementos fundamentais em processamento de imagens digitais.

A **Aquisição** da imagem é o mesmo que captura de uma imagem. Para isso necessitamos de dispositivos para passar uma imagem do mundo real para o mundo digital. Esses dispositivos podem ser *Scanners*, *Frames*/Video e Câmeras Digitais. Não

podemos inserir as imagens no computador exatamente da forma que a visualizamos, pois computadores trabalham com números e não imagens. Logo devemos converter as mesmas para uma forma numérica antes de processá-las, é o que chamamos digitalização. **Armazenamento**, embora existam vários formatos de arquivos de imagem (TIFF, PCX, GIF, JPEG) todos armazenam a mesma imagem digital, ou seja, um conjunto de pixels. A manipulação de imagens por computador, onde a entrada e a saída do processo são imagens é chamado **processamento de imagens digitais**. Uma vez que a imagem foi digitalizada, o desejo de reconstruí-la em alguma superfície de visualização (**exibição**) é imediato. Os meios mais comuns são os monitores e as impressoras. Na seção 2.7, detalharemos melhor o processamento de imagens digitais.

A seguir citaremos alguns valores limite, fundamentos e notações necessárias, no estudo de imagens.

2.2 Modelo de Imagem

O termo imagem, como já foi dito, refere-se a uma função de intensidade luminosa bidimensional, denotada por $f(x,y)$ A amplitude (valor) de f nas coordenadas espaciais (x,y) nos dá a intensidade (brilho) da imagem naquele ponto. Devemos lembrar que sendo a luz uma forma de energia, $f(x,y)$ deve ser positiva e finita, ou seja,

$$0 < f(x,y) < ? \tag{2.2-1}$$

As imagens que nós percebemos em nosso dia a dia, nada mais são do que a luz refletida dos objetos. Basicamente $f(x,y)$ pode ser caracterizada por dois componentes que são a quantidade de luz incidindo na cena observada (iluminação) e a quantidade de luz refletida pelos objetos da cena (reflectância). Para representarmos esses dois componentes usaremos $i(x,y)$ para iluminação e $r(x,y)$ para reflectância. O produto desses dois componentes é $f(x,y)$, ou seja, a imagem.

$$\begin{aligned} f(x,y) &= i(x,y)\ r(x,y), \\ 0 &< i(x,y) < ?, \\ 0 &< r(x,y) < 1 \end{aligned} \tag{2.2-2}$$

Ao observarmos a equação 2.2-2 podemos concluir que a reflectância está entre 0 (absorção total) e 1 (reflectância total). A natureza da iluminação vai ser determinada pela fonte de luz e a da reflectância é determinada pelas características dos objetos na cena. Os limites apresentados para $i(x,y)$ e $r(x,y)$ são teóricos. Vamos introduzir alguns valores médios práticos para um melhor entendimento.

Como exemplo temos alguns limites médios que ilustram alguns intervalos típicos de iluminação. Num dia claro, o sol pode produzir mais de 9.000 pé-candelas (*foot-candles*) de iluminação na superfície da terra, já num dia nublado este valor diminui para menos de 1.000 pé-candelas. Em uma noite clara, a lua cheia gera cerca de 0,01 pé-candela de iluminação. Um escritório comercial tem tipicamente cerca de 100 pé-candelas de iluminação. Similarmente, os valores médios de $r(x,y)$ são 0,01 para veludo negro, 0,65 para aço inoxidável, 0,80 para uma parede branca, 0,80 para metal prateado e 0,93 pé-candela para a neve [Gonzalez, 1992, p. 31].

Ao longo desta dissertação denominamos a intensidade de uma imagem monocromática f nas coordenadas (x,y) de nível de cinza ($f(x,y)$) da imagem naquele ponto. Pela equação 2.2-2 pode-se concluir que $f(x,y)$ fica restrito ao intervalo a seguir:

$$L_{min} \; ? \; f(x,y) \; ? \; L_{max} \qquad (2.2\text{-}3)$$

Teoricamente a única restrição imposta a L_{min} é que seja um valor positivo e sobre L_{max} é que seja finito. Porém na prática o que ocorre é que $L_{min} = i_{min}\, r_{min}$ (iluminação mínima x reflectância mínima) e $L_{max} = i_{max}\, r_{max}$ (iluminação máxima x reflectância máxima), o que é bem coerente. Ao considerarmos os valores acima para iluminação e reflectância temos L_{min} ? 0,005 e L_{max} ? 100 para aplicações de processamento de imagens em ambientes fechados.

A chamada escala de cinza, se refere ao intervalo $[L_{min}, L_{max}]$. Este intervalo geralmente é deslocado para $[0, L]$, onde $f(x,y) = 0$ vai significar o negro, $f(x,y) = L$ significará o branco, todos os valores intermediários são considerados níveis de cinza e variam continuamente entre zero e um. Na Figura 2.1 é mostrada uma imagem com

níveis de cinza contidos no intervalo [0, 255], ou seja, temos uma imagem com 256 níveis de cinza.

Na captura da imagem temos algumas passagens muito importantes as quais tem papel decisivo no que se refere à qualidade da imagem. Amostragem e quantização são processos subjetivos e totalmente dependentes da aplicação, mas que merecem ser detalhados.

2.3 Amostragem e Quantização

Para trazer uma imagem do mundo real para o computador utiliza-se a chamada discretização (digitalização), isto é, toma-se valores pontuais ao longo da imagem e armazena-se este valor no correspondente $f(x,y)$. Uma imagem $f(x,y)$ é adequada para processamento computacional se suas coordenadas espaciais (x,y) forem digitalizadas espacialmente (amostragem da imagem) e em amplitude (quantização em níveis de cinza). Resumindo, a digitalização de uma imagem primeiro passa por uma amostragem e depois por uma quantização[Gonzalez, 1992, p. 31-37].

Seja $f(x,y)$ uma imagem, que é aproximada por amostras igualmente espaçadas arranjadas na forma de uma matriz NxM como mostra a equação abaixo:

$$f(x,y)=\begin{pmatrix} f(0,0) & f(0,1) & \dots & f(0,M-1) \\ f(1,0) & f(1,1) & \dots & f(1,M-1) \\ \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \dots & \vdots \\ f(N-1,0) & f(N-1,1) & \dots & f(N-1,M-1) \end{pmatrix} \tag{2.3-1}$$

Observando a matriz acima temos $f(x,y)$ que é denominada imagem digital e cada elemento desta matriz denomina-se um elemento de imagem, pixel. À distância entre as amostras na grade nos fornece a resolução espacial.

Para esse processo de digitalização devemos decidir a respeito dos valores para N e M (dimensões da matriz) e o número de níveis de cinza discretos para cada pixel. É

comum em processamento digital de sinais assumir que estes valores sejam potências inteiras de dois, ou melhor,

$$N = 2^n , \; M = 2^k \text{ e}$$
$$G = 2^m , \qquad (2.3\text{-}2)$$

em que G é o número de níveis de cinza. Seja uma imagem 128x256, com 64 níveis de cinza, teríamos então N = 128, M = 256 e G = 64, conseqüentemente n = 7, k = 8 e m = 6. Considere que b seja o número de bits necessários para armazenar uma imagem digitalizada, podemos determinar este número através da fórmula:

$$b = N \times M \times m$$
$$\text{Se } M = N \text{ temos: } N^2 \times m \qquad (2.3\text{-}3)$$

Ou seja, uma imagem 256x256 pixels (N = M = 256), com 32 níveis de cinza (m = 5) requer 327.680 bits para seu armazenamento. Quando representamos a aproximação de uma imagem contínua, quanto maior for o nível de detalhamento melhor. Porém, observando-se as equações acima percebemos que o aumento da resolução (grau de detalhamento), tem como conseqüência indesejada imediata o fato de requerer maior espaço para armazenamento bem como maior capacidade de processamento. Definir uma imagem "boa" é um processo altamente subjetivo, bem como dependente dos requisitos da aplicação.

A Figura 2.3 mostra o resultado da redução da resolução espacial de uma imagem de 512 (imagem original) para 128, 64 e 32, respectivamente. Devemos ressaltar que em todos os casos as imagens foram representadas com 256 níveis de cinza e a área de exibição 512x512. Para que isso fosse possível os pixels das imagens de resolução menor foram replicados, de maneira a preencher a área de exibição. Esta replicação de pixels produz um efeito xadrez na imagem (como se a imagem fosse composta de pequenos quadrados) e quanto mais baixa a resolução da imagem, mais visível é esse efeito. Para ilustrar o fato foi feito a leitura da imagem original, logo após os pixels foram armazenados de 4 em 4 em imagem128, 8 em 8 em imagem64 e 16 em 16 em imagem32, respectivamente. Para que as imagens fiquem com a mesma dimensão

cada pixel da imagem128 é duplicada 4 vezes, da imagem 64 é duplicado 8 vezes e da imagem32 é duplicado 16 vezes.

Figura 2.3 Efeito da redução da resolução espacial de uma imagem.

A Figura 2.4 ilustra o efeito que a diminuição dos níveis de cinza utilizados para a representação de uma imagem produz no visual da mesma.


Imagem 512 níveis de cinza


Imagem 256 níveis de cinza

(b)


Imagem 128 níveis de cinza

(c)


Imagem 64 níveis de cinza

(d)


Imagem 32 níveis de cinza

(e)


Imagem 16 níveis de cinza

(f)


Imagem 8 níveis de cinza

(g)

(h)

(j)

Figura 2.4 Efeito da diminuição de níveis de cinza de uma imagem.

Para que isto fosse possível inicialmente ajustamos o valor mínimo da imagem em zero (imagem menos o menor pixel) e o máximo em 256, multiplicamos a imagem pelo número de níveis de cinza desejado (512, 256, 128, 64, 32, 16, 8, 4, 2, respectivamente), dividimos pelo pixel de maior valor pertencente à imagem e arredondamos os valores resultantes.

Observando a Figura 2.4 podemos notar claramente que ao diminuirmos os níveis de cinza para representar a imagem temos uma perda de qualidade da imagem. A Figura 2.4(a) mostra a imagem original 512x512. Ao compararmos as Figuras 2.4(a)-(d) a diferença é imperceptível. Ao analisarmos detalhadamente a imagem com 32 níveis de cinza (Figura 2.4(d)), podemos perceber que a mesma desenvolveu um conjunto, quase imperceptível, de sulcos muito finos nas áreas cujos níveis de cinza são suaves. É o que chamamos de falso contorno. Já quando analisamos as Figuras 2.4(f)-(j), podemos notar claramente esse efeito. Podemos perceber claramente que nestes casos necessitávamos de um maior número de níveis de cinza para representar os detalhes. A imagem 2.4(j), por exemplo, foi representada por dois níveis de cinza, ou seja, a imagem contém pixels de valor 0 e 1.

Em muitos casos, a aparência de uma imagem pode ser melhorada com o uso de um esquema adaptativo, onde o processo de quantização depende das características locais da imagem. Uma forma utilizada em geral é representar regiões com muitos

detalhes através de uma quantização mais fina, sendo que o contrário acontece em regiões com poucos detalhes. Esse esquema tem a desvantagem da necessidade de identificação de fronteiras e em se tratando de imagem com regiões uniformes pequenas isso não é justificável. No caso do número de níveis de cinza ter que ser mantido pequeno, o uso de níveis regularmente espaçados no processo de quantização é geralmente desejável.

2.4 Relacionamentos Básicos entre Pixels

Nesta seção consideraremos os relacionamentos básicos entre os pixels de uma imagem. A referência à imagem é feita por $f(x,y)$ e o subconjunto de pixels da imagem será chamada vizinhança de (x,y) [Gonzalez, 1992, p. 40-43].

2.4.1 Vizinhos de um Pixel

Um determinado pixel qualquer, de coordenadas (x,y), possui dois vizinhos horizontais e dois verticais, cujas coordenadas são dadas por:

$$\begin{array}{ccc} & (x-1,y) & \\ (x,y-1) & (x,y) & (x,y+1) \\ & (x+1,y) & \end{array}$$

A esse conjunto de pixels chamamos *vizinhança-4* de (x,y) e representamos por $N_4(x,y)$. Podemos perceber claramente que cada pixel está a uma unidade de distância do pixel central (x,y). É claro também que alguns vizinhos de (x,y) vão ficar fora da imagem ao considerarmos os pixels das bordas.

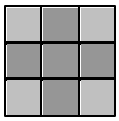
Da mesma forma, (x,y) possui quatro vizinhos diagonais, com coordenadas,

$$\begin{array}{ccc} (x-1,y-1) & & (x-1,y+1) \\ & (x,y) & \\ (x+1,y-1) & & (x+1,y+1) \end{array}$$

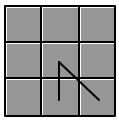
que são denotadas por $N_D(x,y)$. A junção dos vizinhos horizontais e verticais com os vizinhos diagonais dá origem a chamada *vizinhança-8* de (x,y), com coordenadas,

(x-1,y-1)	(x-1,y)	(x-1,y+1)
(x,y-1)	(x,y)	(x,y+1)
(x+1,y-1)	(x+1,y)	(x+1,y+1)

que são representadas por $N_8(x,y)$. Ao analisarmos as vizinhanças $N_4(x,y)$, $N_D(x,y)$ e $N_8(x,y)$ fica claro que alguns pontos estarão fora da imagem quando (x,y) se encontrar nas bordas da imagem.



Vizinhança 4 ($N_4(x,y)$)



Vizinhança 8 ($N_8(x,y)$)

Figura 2.5 Vizinhos do pixel.

Conectividade entre pixels é um conceito importante usado para detectar contornos (bordas) de objetos e componentes de uma região em uma imagem. Para estabelecer se dois pixels estão conectados, tem que ser determinado se eles, de alguma forma, são adjacentes (por exemplo, se eles estão numa vizinhança-4) e se seus valores de níveis de cinza satisfazem ao critério de similaridade (isto é, se eles são semelhantes). Como exemplo, em uma imagem binária na qual os valores dos pixels são 0 e 1, dois pixels podem ser adjacentes através de uma vizinhança-4, mas não são considerados conectados se não tiverem o mesmo valor [Gonzalez, 1992, p. 43].

2.5 Processamento de Imagens

Técnicas voltadas à análise e manipulação de imagens adquiridas por diversos tipos de sensores recebem o nome de Processamento de imagens digitais, ou seja, manipulação de uma imagem por computador onde a entrada e a saída do processo são imagens. Utilizado para melhoramento do aspecto visual da imagem para análise humana e para fornecer outros subsídios para a sua interpretação, inclusive gerando produtos que possam ser posteriormente submetidos a outros processamentos [Scuri,

1999, p.78-80]. É normalmente utilizada como uma etapa de pré-processamento para sistemas de reconhecimento de padrões.

A Figura 2.6 ilustra, através de um diagrama, uma seqüência tradicional para processamento da imagem, da captura à classificação. Podemos assim ter uma idéia melhor das classes de processamento por resultado.

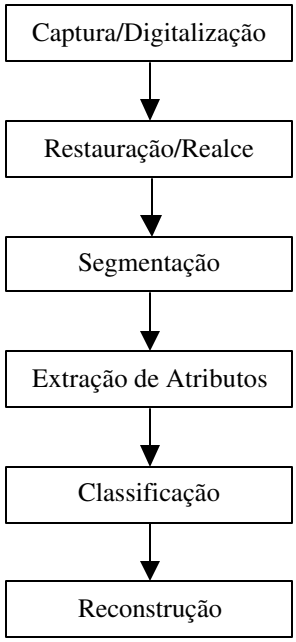


Figura 2.6 Seqüência tradicional para processamento de imagens.

Há um fator muito importante em processamento de imagens, estamos falando de qualidade. Existem duas subdivisões em qualidade de imagem: fidelidade e inteligibilidade. Quando se considera fidelidade o interesse/preocupação é em aproximar a imagem processada da imagem original ou de um padrão estipulado que melhor a represente. Ao considerarmos inteligibilidade, o interesse/preocupação é centrado na informação que se consegue extrair da imagem, seja pelo olho humano, seja por algum processamento.

Como exemplo podemos citar programas de editoração eletrônica que requerem um cuidado grande no que se refere à representação da cor da imagem (fidelidade), o

que não ocorre com programas que tratam de processamento de imagens científicas que em geral não estão preocupados com fidelidade de cor, esta fidelidade é mais importante em se tratando da informação contida na imagem (inteligibilidade).

A seguir descrevemos cada uma das etapas.

2.5.1 Restauração

A restauração busca melhorar a qualidade da imagem para visualização humana, ou seja, compensar distorções específicas, normalmente geradas no momento de aquisição. Quando se pode identificar experimentalmente a função que representa a deformação ou construir um modelo matemático adequado, é possível buscar a função inversa e aplicá-la sobre a imagem deformada. Por exemplo: correção de foco, imagens borradas por movimento. Em todos os casos, a formulação matemática envolvida é extremamente complexa e o custo computacional muito alto e o resultado pode não ser satisfatório.

2.5.2 Segmentação

Quando estamos, por exemplo, no contexto de processamento de imagens científicas, o mais comum é querermos obter dados relacionados com os objetos presentes na imagem. Então são necessárias as operações de Segmentação que procurarão isolar regiões de pixels e operações de Extração de Atributos que vão atuar sobre essas regiões e calcular uma série de parâmetros que as descreverão.

A operação de segmentação mais comum é a limiarização por um tom de corte. Tudo que está acima deste tom vira branco, tudo que está abaixo vira preto, obtendo-se uma imagem binária. A partir desse momento as operações Morfológicas [Gonzalez, 2000, p. 369-399]. são excepcionalmente úteis.

2.5.2.1 Operações morfológicas

A morfologia matemática permite a extração de informações relativas à geometria e à topologia de uma imagem, sendo amplamente utilizada em problemas de reconstrução e restauração de imagens corrompidas por ruídos. As transformações de

imagens através da morfologia matemática são construídas por um conjunto de transformações derivadas de operações elementares.

As operações morfológicas elementares são dilatação e erosão, e, por conseqüência, das operações de abertura e fecho, que permite a eliminação de ruído e a remoção de irrelevâncias de modo semelhante ao caso de imagens binárias.

Erosão para cada pixel branco, se o número de vizinhos brancos for menor que um valor limiar, N o pixel é invertido. Isto faz com que objetos pequenos ou finos sejam eliminados e objetos maiores vão ter sua área diminuída.

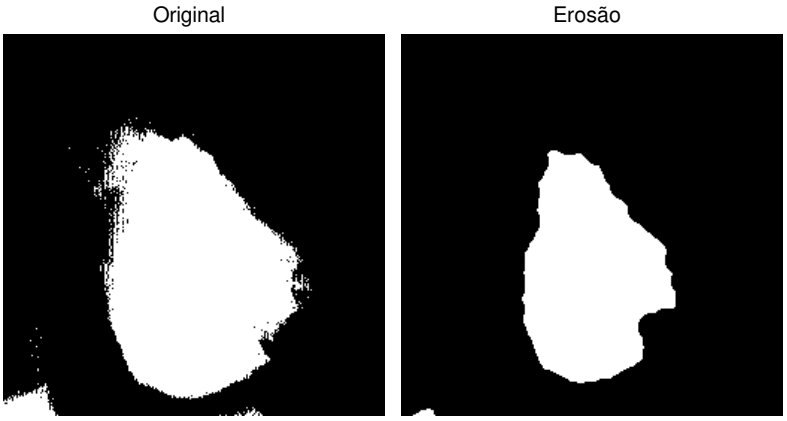


Figura 2.7 Erosão

Dilatação para cada pixel preto, se o número de vizinhos brancos for maior do que um valor limiar, N, o pixel é invertido. Isto faz com que buracos finos ou pequenos sejam eliminados, unindo objetos, e aumentando a sua área.

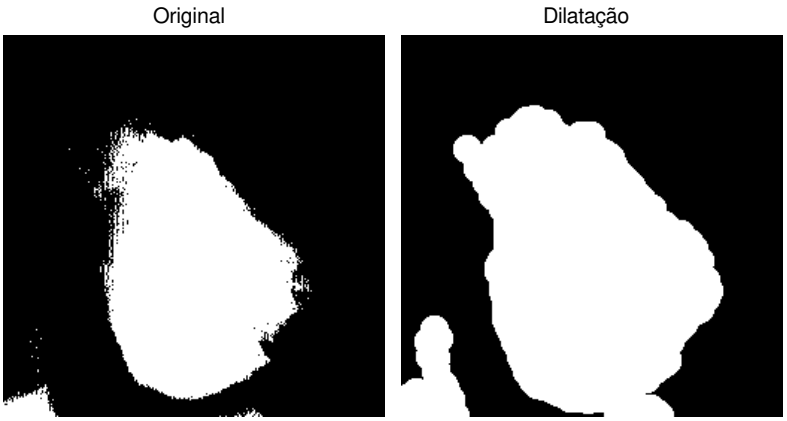


Figura 2.8 Dilatação

Estas operações por si só causam distorções nas áreas dos objetos. No entanto, sua combinação gera resultados interessantes. A remoção de ruído em imagens binárias pode ser efetuada pela aplicação de sucessivas operações morfológicas.

Abertura trata-se de N ciclos de erosão seguidos de N ciclos de dilatação. Faz com que objetos pequenos desapareçam inclui pequenas conexões entre os objetos. Objetos maiores não são afetados.

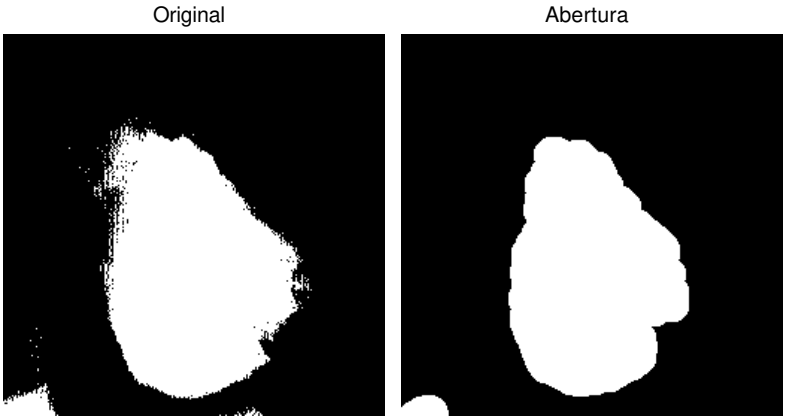


Figura 2.9 Abertura

Fechamento trata-se de N ciclos de dilatação seguidos de N ciclos de erosão. Faz com que buracos pequenos ou separações entre objetos sejam eliminados. Objetos maiores não são afetados.

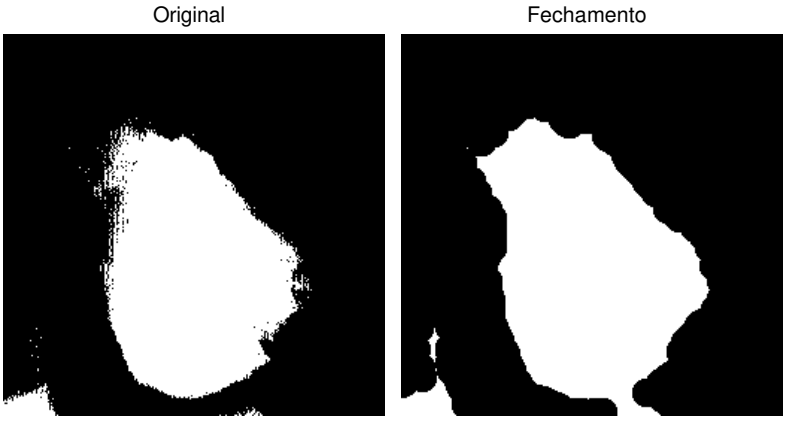


Figura 2.10 Fechamento

Caso o número de ciclos seja grande quando comparado ao diâmetro dos objetos, ocorrem distorções de forma.

2.5.3 Extração de Atributos

Utilizando imagens binárias é muito simples obter dados relevantes, ou atributos, das regiões segmentadas, tais como:

- Número total de objetos.
- Propriedades geométricas, do tipo: Área, Perímetro, Centro de gravidade, Largura máxima e mínima.
- Atributos relacionados à forma, tais como: Circularidade, Concavidade.
- Propriedades de luminância, tais como: Nível de cinza médio de cada região (1º momento da distribuição), Desvio padrão do nível de cinza (2º momento da distribuição), outros momentos estatísticos da distribuição de cada região (*skewness* e *kurtosiss - assimetria e achatamento*) [Scuri, 1999, p. 80].
- Propriedades associadas à textura

2.5.4 Classificação

Uma vez com esses parâmetros coletados, queremos distinguir objetos na imagem agrupando esses parâmetros de acordo com sua semelhança para cada região de pixels encontrada. Feita essa classificação desses parâmetros os objetos estão reconhecidos e podemos agora tomar decisões e relatar fatos relacionados com os objetos do mundo real, ponderando sempre através de uma medida de erro da classificação.

Esse processo é muito complexo e existem diversos níveis de automação. Os mais simples implicam em processos de agrupamento estatístico, para os quais a decisão humana é fundamental. Os mais sofisticados permitem ao computador reconhecer diferentes objetos através de técnicas de inteligência artificial com pouca ou nenhuma intervenção humana. Os processos que possuem intervenção são chamados de supervisionados.

Aqui, a palavra classificação não denota nenhum juízo de valor, mas apenas o grupamento em classes dos diversos objetos obtidos na segmentação, cujos atributos já foram medidos.

Em geral, vários atributos são necessários para uma correta classificação e quanto maior o número de atributos, mais complexo se torna o problema. Dessa forma, é muito importante realizar uma seleção adequada dos atributos disponíveis, visando otimizar o processo.

2.6 Processamento Baseado em Áreas ou Regiões da Imagem

São técnicas aplicadas a uma determinada região da imagem, cujos pontos estão localizados em torno do ponto que se deseja estudar. Essas técnicas têm muitas aplicações na obtenção das características primitivas da imagem, como por exemplo, a extração de contornos para um melhor realce, na suavização, introdução de borramento e atenuar ruído aleatório. Os pixels vizinhos são utilizados no processamento para a

extração de informações. O pixel do centro da vizinhança é normalmente substituído pelo novo valor, resultado da aplicação de algum algoritmo de processamento.

O custo computacional (cálculo) cresce de acordo com o tamanho da vizinhança considerada, bem como com o tipo de representação numérica utilizada. Porém, para a maioria das aplicações e considerando-se os computadores existentes atualmente, pode-se obter resultados muito bons em termos de cálculo, ao se processar imagens de tamanho médio (256x256 ou 512x512).

A Figura 2.7 mostra um exemplo de vizinhança, o ponto central (mais escuro) é a amostra central e os mais claros ilustram vizinhanças 3x3, 5x5, 7x7 e 9x9, respectivamente de dentro para fora. Essas são as vizinhanças utilizadas na maioria das aplicações.

45	51	52	51	51	44	41	41	31
52	45	56	65	56	48	49	42	43
63	54	53	55	51	53	45	43	39
63	52	55	47	51	48	46	44	38
64	58	47	43	54	49	50	41	33
71	56	43	47	61	60	45	40	37
74	48	50	60	79	79	48	38	40
63	52	50	57	94	93	76	47	41
65	65	61	66	55	57	47	40	34

Figura 2.11 Exemplo numérico de vizinhança.

Com relação à vizinhança utilizada devemos nos preocupar com dois fatores Primeiramente, convém que o tamanho da janela sempre seja um número ímpar, caso contrário não se terá um único pixel central. Devemos nos preocupar também com o fato de que a vizinhança pode assumir qualquer tamanho e o crescimento dessa vizinhança pode tornar nosso algoritmo (aplicação) inviável computacionalmente.

No Capítulo 3, vamos falar também de processamento. Vamos abordar processamentos de filtragem, para recuperação de imagens degradadas por algum tipo de ruído. Técnicas de filtragem são transformações da imagem "pixel" a "pixel", que não dependem apenas do nível de cinza de um determinado "pixel", mas também do valor dos níveis de cinza dos "pixels" vizinhos, na imagem original. Estas técnicas são utilizadas visando o melhoramento da qualidade da imagem. Filtragem é um conjunto de técnicas destinadas a corrigir e realçar uma imagem. Tem como objetivo a remoção de características indesejáveis e a melhoria/realce, acentuação de características anteriores ao ruído. Daremos ênfase a filtros não lineares, pois esses apresentaram um melhor desempenho.

CAPÍTULO 3

REALCE DE IMAGENS

Técnicas de realce de imagens têm como principal objetivo o processamento de uma imagem, de modo que o resultado obtido seja mais apropriado para uma aplicação específica do que a imagem original. Neste capítulo discutiremos a filtragem espacial, aplicada na recuperação de imagens degradadas por algum tipo de ruído. O que se percebe quando se trata desse assunto é que as técnicas são específicas, ou seja, são bastante dependentes da aplicação. O melhor método para realce de imagens de raios X pode não ser a melhor abordagem para realce de fotos de Marte transmitidas por sonda espacial.

3.1 Filtragem Espacial

O uso de máscaras (janelas de observação) espaciais para processamento de imagens é chamado filtragem espacial e as máscaras são chamadas filtros espaciais. Nesta tese de mestrado consideraremos filtros lineares e não-lineares, para realce de

imagens. Como os filtros não-lineares tiveram uma melhor resposta, daremos maior ênfase aos mesmos.

Em filtragem espacial linear aplica-se um processo linear sobre a imagem de entrada. Já nos filtros não lineares o que se tem é basicamente uma análise estatística dos valores de níveis de cinza na vizinhança em que o filtro está posicionado.

Os procedimentos de filtragem nos dois casos são os seguintes:

- varredura na imagem de entrada pixel por pixel;
- processamento de cada pixel da entrada, considerando uma vizinhança e utilizando um algoritmo apropriado;
- O novo valor do pixel obtido no item anterior é substituído na imagem de saída na mesma posição ocupada na imagem de entrada.

3.1.1 Notação Básica

Sabemos que uma imagem é uma seqüência discreta de pixels (matriz de pixels). A imagem desejada será representada por {d(n)} e a corrompida por {x(n)}, ou seja, {d(n)} é a saída desejada para o filtro e {x(n)} é a imagem a ser recuperada (entrada do filtro) e *n* é um vetor que contém *v* elementos de forma que $n=[n_1, n_2, ..., n_v]$. Logo se conclui que {d(n)} e {x(n)} são *v* dimensionais [Hardie, 1994].

Para cada elemento n de x teremos uma vizinhança e a esse conjunto de elementos chamaremos janela de observação (matriz observação, máscara, template, ou simplesmente janela). Vamos representar as amostras de uma janela de observação por: $x(n)=[x_1(n), x_2(n), ..., x_N(n)]$, sendo que N é o número de elementos da janela.

Em alguns casos vamos necessitar ordenar este vetor x(n). Este vetor ordenado será representado da seguinte forma: $x^r(n)=[x_{(1)}(n), x_{(2)}(n), ..., x_{(N)}(n)]$, com $x_{(1)}(n) \leq x_{(2)}(n) \leq ... \leq x_{(N)}(n)$.

O relacionamento entre a posição da amostra e sua localização dentro da janela de observação é definida por $r_i(n)$ que é o mesmo que dizer $x_i(n) = x_{(ri(n))}(n)$. Temos então $r(n)= [r_1(n), r_2(n), ..., r_N(n)]$.

Pode-se representar esses dados de outra forma, porém as notações acima são as mais comuns. Utilizaremos mais algumas notações, porém na maioria dos casos, as mesmas serão descritas para filtros específicos. Nas próximas seções falaremos a respeito de filtros lineares e não lineares.

3.1.2 Filtragem Espacial Linear

Os filtros espaciais lineares são utilizados para atenuação, ou mesmo remoção do ruído, que consiste em fazer uma convolução da imagem com uma matriz. O filtro é dito linear devido ao fato de que mantém o produto e a soma da média por um escalar.

Esses filtros baseiam-se na combinação de pixels pertencentes a uma vizinhança do pixel em observação e uma matriz bidimensional (janela de observação ou máscara) que descreve o processo linear a ser aplicado. A isso denominamos convolução. A convolução de uma imagem f(x, y) com o filtro (núcleo ou máscara de convolução) de resposta impulsiva h(x, y) produzirá uma imagem de saída q(x, y), conforme mostrado na equação (3.1), sendo que m, n definem a vizinhança considerada de acordo com o tamanho da máscara de convolução h(x, y) [Oppenheim, 1975, p. 21].

$$q(x,y) = \sum_{m=K1}^{K2} \sum_{n=L1}^{L2} h(m,n) f(x-m, y-n) \qquad (3.1\text{-}1)$$

Geralmente a máscara de convolução é de tamanho 3x3, o que não nos impede de termos máscaras de tamanhos diferentes. A máscara de convolução possui um número ímpar de elementos, ou seja, tem número ímpar de linhas e colunas. Os tamanhos mais utilizados, geralmente são: 3x3, 5x5, 7x7 e 9x9. Já o seu conteúdo depende do tipo de processamento que desejamos realizar [Pratt, 1978].

Independente do filtro linear utilizado, o processo de filtragem consiste em somar os produtos entre os coeficientes da janela de observação e os valores de intensidade dos pixels na janela em uma posição específica da imagem. A Figura 3.1

ilustra uma janela de observação 3x3, cujos coeficientes são w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9 e os níveis de cinza dos pixels da região da imagem, onde está posicionada a janela de observação, são z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9 (entrada), respectivamente.

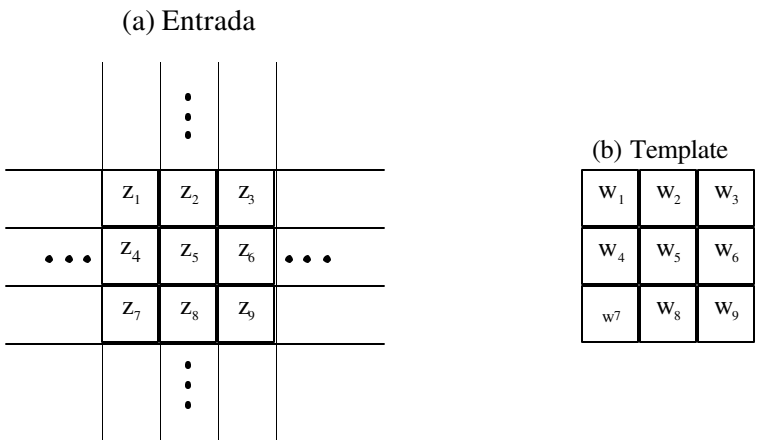


Figura 3.1 (a) Subárea da imagem; (b) máscara 3x3.

A saída é obtida através de R, que obedece o seguinte:

$$R = w_1 z_1 + w_2 z_2 + w_3 z_3 + w_4 z_4 + w_5 z_5 + w_6 z_6 + w_7 z_7 + w_8 z_8 + w_9 z_9 . \quad (3.1\text{-}2),$$

$$R = (w_1 z_1 + w_2 z_2 + \ldots + w_9 z_9) = \sum_{i=1}^{9} w_i z_i \quad (3.1\text{-}3)$$

Substitui-se o centro da janela por R e desloca-se a mesma para a posição seguinte e o processo se repete.

Repete-se esse processo até que todos os pixels sejam varridos. Ao utilizarmos esse processo, quando somamos nove níveis de cinza, o resultado (R) pode ficar fora do intervalo de níveis de cinza válidos. Por esse motivo necessitamos normalizar, ou seja, dividir R por 9.

Ao analisarmos o que acabamos de fazer, percebemos claramente que o resultado desse processamento é a média ponderada de todos os pixels contidos na área da janela

de observação. O uso de janelas cujos coeficientes são iguais a um é chamado média ponderada na vizinhança [Gonzalez, 1992, p. 191].

A Equação 3.1-3 é amplamente usada em processamento de imagens. A seleção adequada dos coeficientes bem como a aplicação da máscara em cada pixel de uma imagem torna possível uma variedade de operações de processamento de imagens úteis, tais como redução de ruído, afinamento de regiões e detecção de bordas. Entretanto, aplicar uma máscara para cada posição dos pixels em uma imagem é uma tarefa computacionalmente bastante dispendiosa. Por exemplo, a aplicação de uma máscara 3x3 a uma imagem de 512x512 requer nove multiplicações e oito adições para cada pixel, resultando num total de 2.359.296 multiplicações e 2.097.152 adições.

Alguns filtros espaciais lineares são mostrados a seguir:

Os Filtros espaciais Passa-Baixas suavizam a imagem atenuando ou eliminando os componentes de altas-freqüências, isto é, o filtro deixa passar as baixas-freqüências. As baixas-freqüências permanecem inalteradas. O mesmo tende a minimizar ruídos e apresenta o efeito de borramento da imagem, por atenuar os componentes que caracterizam bordas e outros detalhes.

A Tabela 3.1 mostra alguns filtros espaciais Passa-Baixas de tamanhos variados (3x3, 5x5 e 7x7 respectivamente). Se considerarmos o primeiro filtro teremos, nesse caso, $w_1 = w_2 = w_3 = w_4 = w_5 = w_6 = w_7 = w_8 = w_9 = 1$ e $R = z_1 + z_2 + z_3 + z_4 + z_5 + z_6 + z_7 + z_8 + z_9$.

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$\frac{1}{25}$

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

$\frac{1}{49}$

1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

Tabela 3.1 Filtros Passa-Baixas espaciais de tamanhos variados.

A Figura 3.2 ilustra o resultado obtido por um filtro Passa-Baixas, janela de observação 3x3 e entrada corrompida por ruído do tipo sal e pimenta (*Salt & Pepper*) com densidade 0,02.

Figura 3.2 Imagem recuperada por um filtro Passa-Baixas.

Ao observarmos a Figura 3.2 fica bem claro o efeito de borramento citado acima; tempo de processamento foi de 7 segundos.

Os filtros espaciais Passa-Altas realçam detalhes, atenuam ou eliminam as componentes de baixa-freqüência. Esses componentes são responsáveis por características que variam lentamente, tais como contraste total e intensidade média, o que produz um realce das bordas (*sharpening)* da imagem. Ou seja, as transições entre regiões diferentes tornam-se mais nítidas. Esses filtros podem ser usados para realçar certas características presentes na imagem, tais como bordas, linhas curvas ou manchas, mas enfatizam o ruído existente na imagem [Lopes, 1996].

$$\frac{1}{9}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad \text{ou} \quad \frac{1}{9}\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Tabela 3.2 Filtros Passa-Altas espaciais.

A Figura 3.3 ilustra o resultado obtido pelo segundo filtro Passa-Altas, mostrado na Tabela 3.2, com janela de observação 3x3 e entrada corrompida por ruído do tipo sal e pimenta com densidade 0,02.

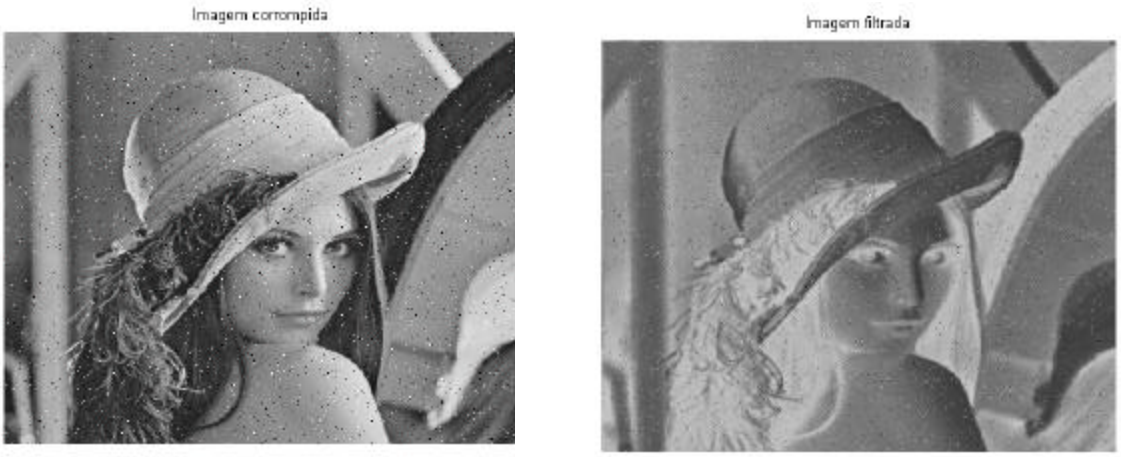


Figura 3.3 Imagem recuperada por um filtro Passa-Altas.

Observando a Figura 3.3 podemos ver que a mesma ficou com as bordas realçadas. O tempo de processamento foi de 2 segundos.

3.1.3 Filtragem Espacial Não-Linear

Filtros não lineares estão aumentando, cada vez mais, a sua importância no processamento de imagens. São bem melhores no que se refere a extrair ruído sem distorcer as características da imagem. Assim como os filtros espaciais lineares, os não-lineares também operam em uma vizinhança. Baseiam-se nos valores dos pixels na vizinhança considerada, fazem uma análise estatística dos valores de níveis de cinza e geram a saída.

Para efeito de comparação e estudo foram implementados vários filtros. Nas sessões abaixo temos filtros lineares e não-lineares.

3.2 Dados e Funções Auxiliares

A partir de agora utilizaremos alguns dados básicos para teste dos filtros implementados. Vamos adotar como padrão uma imagem, uma janela de observação 3x3 e uma imagem a ser recuperada. Isso será feito com objetivo de facilitar os testes e a compreensão dos algoritmos mostrados nas seções seguintes.

Nem todos os filtros vão recuperá-la de forma satisfatória, como vamos constatar mais adiante. Isso se dá pois esse tipo de processamento é altamente dependente da aplicação.

O uso de máscara envolve um problema em sua implementação que deve ser resolvido para que se obtenha bons resultados. Quando uma componente está na borda da imagem o mesmo não possui 8 vizinhos, o que ocorre devido ao domínio da imagem ser limitado. Temos várias soluções típicas (ver Anexo 1, p. 143-148):

- loops especializados para cada tipo de vizinhança
- aumentar a imagem, processar e retornar ao tamanho original. Neste caso podemos agir de três formas:
  - ✍ zeros fora da imagem
  - ✍ repetir linhas e colunas
  - ✍ repetir linhas e colunas como se a imagem fosse circular

Para verificar o desempenho dos algoritmos utilizados para restauração de imagens podemos utilizar os critérios: erro médio absoluto (MAE – *Mean Absolute Error*), erro quadrático médio (MSE – *Mean Square Error*) e melhoramento da relação sinal ruído (ISNR - *Improvement in Signal to Noise Ratio*). Os mesmos são utilizados para efeito de comparação entre os filtros, ou seja, para medir o quanto a imagem recuperada ficou parecida com a imagem desejada.

Para o cálculo do erro médio absoluto (MAE), temos a fórmula 3.2-1, que expressa a diferença entre a imagem recuperada e a imagem desejada. Como o nome já diz temos aqui exatamente o erro médio por pixel entre a imagem original ($d(i,j)$) e a recuperada ($x(i,j)$).

$$MAE = \frac{1}{MN} \sum_{i,j}^{M,N} |x(i,j) - d(i,j)| \tag{3.2-1}$$

Para o cálculo do erro quadrático médio (MSE) é definido pela fórmula 3.2-2, que expressa o somatório da diferença entre a imagem recuperada($x(i,j)$) e a imagem original ($d(i,j)$), ao quadrado, dividido pelo número de elementos da matriz.

$$MSE = \sqrt{\frac{\sum_{i,j}^{M,N} \left[x(i,j) - \hat{d}(i,j) \right]^2}{MN}} \tag{3.2-2}$$

Já no caso do melhoramento da relação sinal ruído (ISNR - *Improvement in Signal to Noise Ratio*) utilizaremos a fórmula 3.2-3. Nesse caso faz-se necessário o conhecimento da imagem corrompida ($x(i,j)$), da imagem restaurada ($\hat{d}(i,j)$) e da imagem original ($d(i,j)$).

$$ISNR = 10\log_{10} \left(\frac{\sum_{i,j}^{M,N} \left[d(i,j) - x(i,j) \right]^2}{\sum_{i,j}^{M,N} \left[d(i,j) - \hat{d}(i,j) \right]^2} \right) \tag{3.2-3}$$

Essas medidas nem sempre refletem as propriedades perceptuais do sistema de visão humana, porém as mesmas nos fornecem medidas objetivas, possibilitando-nos uma comparação entre as diferentes técnicas de restauração [Banham, 1997].

Utilizaremos as fórmulas MAE (3.2-1) ou ISNR (3.2-3), para medir a diferença (erro) entre a imagem original e a imagem recuperada.

Abaixo temos a imagem original e a imagem corrompida.

Figura 3.4 Imagem desejada.

A Figura 3.4 mostra nossa imagem desejada, que é uma imagem de tamanho 512x512, com 256 níveis de cinza. A mesma imagem, corrompida por um ruído do tipo sal e pimenta de densidade 0,02, é visualizada abaixo.

Figura 3.5 Imagem corrompida.

Essa (Figura 3.5) será a imagem de entrada para os filtros mostrados neste capítulo, ou seja, a imagem a ser filtrada. O erro MAE apresentado pela imagem

corrompida é 2,6. Todos os filtros discutidos nessa seção serão aplicados sobre esta imagem corrompida.

3.3 Filtro de Wiener

Inicialmente utilizaremos o filtro de *Wiener,* na tentativa de restaurar a imagem degradada. Para isto temos a função Wiener2 (ver Anexo 3, p. 161-164) do Matlab 6.0.0.88, Release 12, 22 de setembro de 2000. Esse processo está melhor detalhado no livro Gonzalez [Gonzalez, 1992, p. 279].

A Figura 3.6 mostra a saída do filtro de Wiener.

Figura 3.6 Saída do filtro Wiener.

Através da Figura 3.6, percebemos claramente que o filtro de Wiener não apresenta uma boa resposta. Visivelmente a imagem não foi bem recuperada, apesar do erro (MAE), parecer baixo ele aumentou se comparado com o erro inicial que era 2,6. A imagem ficou mais próxima da imagem corrompida do que da desejada. O tempo gasto para o processamento foi de 1,45 segundos e da imagem na saída do filtro foi 5,38.

3.4 Filtro de Média Aritmética – MA

O Filtro de Média Aritmética opera com base na vizinhança dos pixels da imagem de entrada. A idéia é modificar o valor do pixel central em função do seu próprio nível de cinza e o de seus vizinhos. O processamento é feito da seguinte forma: seleciona-se na imagem de entrada do filtro uma janela com as dimensões da janela de observação, soma os valores selecionados e divide pelo número de elementos da janela. Arredonda-se o valor obtido e substitui-se no centro da sub-matriz da imagem de entrada. Desloca-se a janela de observação e o processo se repete, até que toda a imagem de entrada seja varrida. [Gonzalez, 1992, p. 33-36].

A Figura 3.7 ilustra a forma de filtragem utilizada pelo filtro média aritmética (MA).

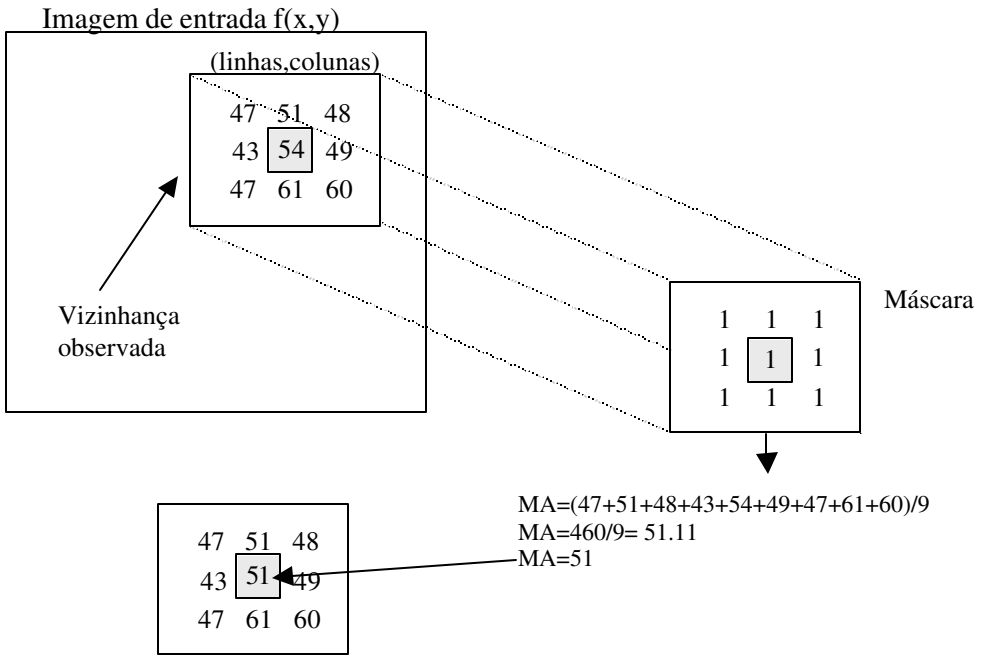


Figura 3.7 Exemplo numérico do filtro MA.

A Figura 3.8 mostra a saída do filtro média aritmética.

Figura 3.8 Saída do filtro MA.

Observando a Figura 3.8 notamos que a imagem na saída ficou bem melhor do que a imagem de entrada, porém não temos uma recuperação muito satisfatória, pelo menos no que se refere ao visual. Isso ocorre, pois o filtro nada mais é do que uma média dos pixels de uma vizinhança. Isso fez com que o filtro apenas suavizasse o ruído. O tempo gasto para o processamento foi 1.8 de segundos e o erro MAE apresentado pela imagem na saída do filtro foi de 16,3.

3.5 Filtro de Média Aritmética Ponderada – MAP

O Filtro de Média Aritmética Ponderada é de uma variação do filtro de Média Aritmética, cuja janela de observação apresenta números diferentes de um. Opera também com base na vizinhança dos pixels da imagem de entrada e o processamento é feito da seguinte forma: posiciona-se a janela de observação sobre a imagem, multiplica os valores dos pixels selecionados pelos pesos correspondentes, soma-se os valores resultantes e divide pela soma dos elementos da janela. Arredonda-se o valor obtido e substitui-se no centro da sub-matriz da imagem de entrada. Desloca-se a janela de observação e o processo se repete, até que toda a imagem de entrada seja varrida.

A Figura 3.9 ilustra a forma de filtragem utilizada pelo filtro média aritmética ponderada (MAP).

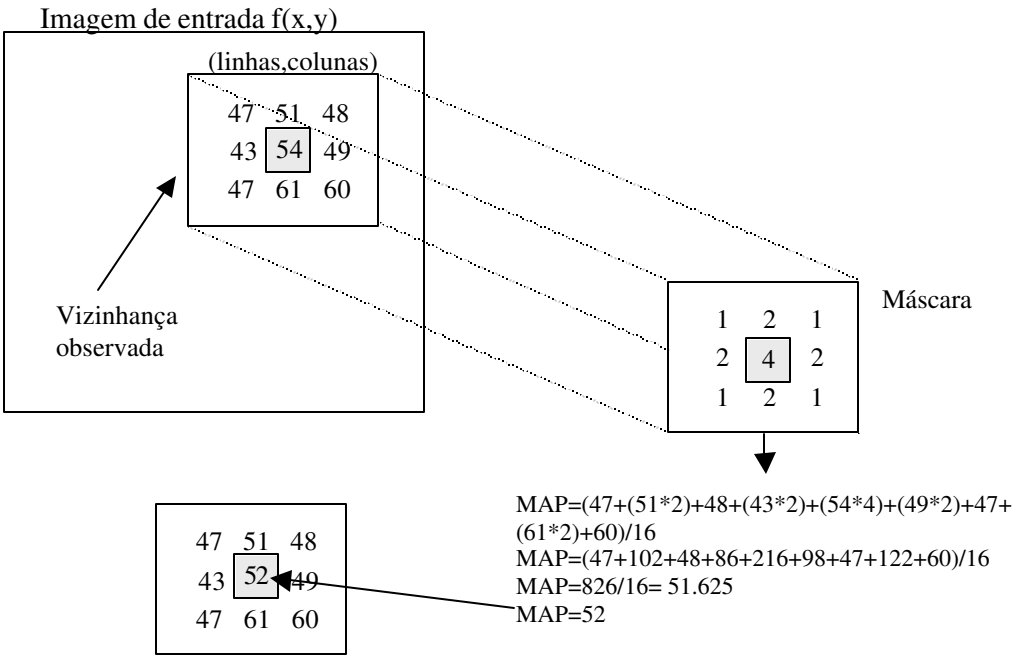


Figura 3.9 Exemplo numérico do filtro MAP.

Para obtenção de um melhor desempenho utilizaremos a técnica descrita em [Gonzalez, 1992, p. 48-51]. A técnica funciona da seguinte forma: ao invés de deslocarmos a janela de observação pela imagem, deslocaremos a imagem e multiplicaremos pelo valor de um dos pixels da janela. A Tabela 3.3, nos dará uma visão melhor desse método.

faux=f*w_5	**fd** = Desloca **f** para a direita
faux=faux+(fd*w_4)	**fd** = Desloca **fd** para baixo
faux=faux+(fd*w_1)	**fd** = Desloca **fd** para a esquerda
faux=faux+(fd*w_2)	**fd** = Desloca **fd** para a esquerda
faux=faux+(fd*w_3)	**fd** = Desloca **fd** para cima
Faux=faux+(fd*w_6)	**fd** = Desloca **fd** para cima
faux=faux+(fd*w_9)	**fd** = Desloca **fd** para a direita
faux=faux+(fd*w_8)	**fd** = Desloca **fd** para a direita
faux=faux+(fd*w_7)	**fd** = Desloca **fd** para a esquerda
	fd = Desloca **fd** para baixo

Tabela 3.3 Processamento para filtro MAP.

Antes teríamos quatro "*loops for*" em cadeia o que, no nosso caso, resultaria em 2.359.296 passagens. Esses números são bastante significativos, principalmente quando se trabalha com o Matlab, onde a execução de "*loops*" é demorada.

Função contendo todos os "*loops for*"

```
for lf=1:512              % lf: número de linhas da imagem de 1 até 512
    for cf=1:512          % cf: número de colunas da imagem de 1 até 512
        for i=1:3         % i: número de linhas da janela de observação de 1 até 3
            for j=1:3     % j: número de colunas da janela de observação de 1 até 3
                faux(lf, cf)=faux(lf, cf) + (f(lf +i-2, cf+j-2) * w(i, j));
            end
        end
    end
end
```

Da outra forma teríamos dois '*loops for*" em cadeia o que, no nesse caso resultaria em 9 passagens.

Abaixo mostramos a função genérica utilizada.

```
for i=-1*deltam:deltam,          % deltam linhas da janela dividido por dois (arredonda)
    for j=-1*deltan:deltan,      % deltan colunas da janela dividido por dois (arredonda)
        temp=temp*0;             % matriz temporária
                                 % tempv e temph são as dimensões da matriz temporária
                                 que foi aumentada para tratar o 'problema das bordas'.
        temp(deltam+1+i:tempv-deltam+i, deltan+1+j:temph- deltan+j) =
                                 f*w(i+deltam+j+deltan+1);
                                 % g matriz cumulativa
        g=g+temp;
    end;
end;
```

A Figura 3.10 mostra a saída do filtro média aritmética ponderada.

Figura 3.10 Saída do filtro MAP.

Observando a Figura 3.10 notamos que a imagem na saída ficou bem melhor do que a imagem de entrada, porém não temos uma recuperação muito satisfatória. Isso ocorre, pois o filtro nada mais é do que uma média dos pixels de uma vizinhança. Isso fez com que o filtro apenas suavizasse o ruído. O tempo gasto para o processamento foi 2 de segundos e o erro MAE apresentado pela imagem na saída do filtro foi de 13,2.

3.6 Filtro Condicional de Média Aritmética – CMA

A saída do Filtro condicional de média aritmética (CMA) realiza a filtragem Passa-Baixas ou suavização de uma imagem contaminada com ruído aleatório. Faz-se uma varredura na imagem inteira, sempre observando o pixel, que se encontra no centro da vizinhança (no nosso caso centro da janela 3x3 observada). Somam-se seus oito vizinhos e se o módulo da diferença entre o valor do pixel e se essa soma for maior que um valor limite U (este valor limite depende da aplicação), substitui-se o pixel central pela média dos oito vizinhos (este valor deve ser arredondado). Caso contrário, o valor do pixel observado continua o mesmo. O crescimento da janela de observação pode inserir um borramento não desejável à imagem. Uma sugestão para o valor do limiar (U) é utilizar a informação do ruído que contaminou a imagem, tal como a variância do ruído [Pérez, 2000].

Este processamento pode ser representado pela fórmula a seguir:

$$CMA = \begin{cases} \frac{1}{8}\sum_{i=1}^{8} P_i, & se \left| P - \frac{1}{8}\sum_{i=1}^{8} P_i \right| > U \\ P, & se \left| P - \frac{1}{8}\sum_{i=1}^{8} P_i \right| > U \end{cases}, \qquad (3.6\text{-}1)$$

onde Pi (i = 1, 2, ..., 8) são os oito vizinhos do pixel central e CMA é a saída do filtro.

A Figura 3.11 mostra um exemplo numérico para melhor compreensão desse filtro. Neste caso o valor de U foi 0.25 [Pérez, 2000].

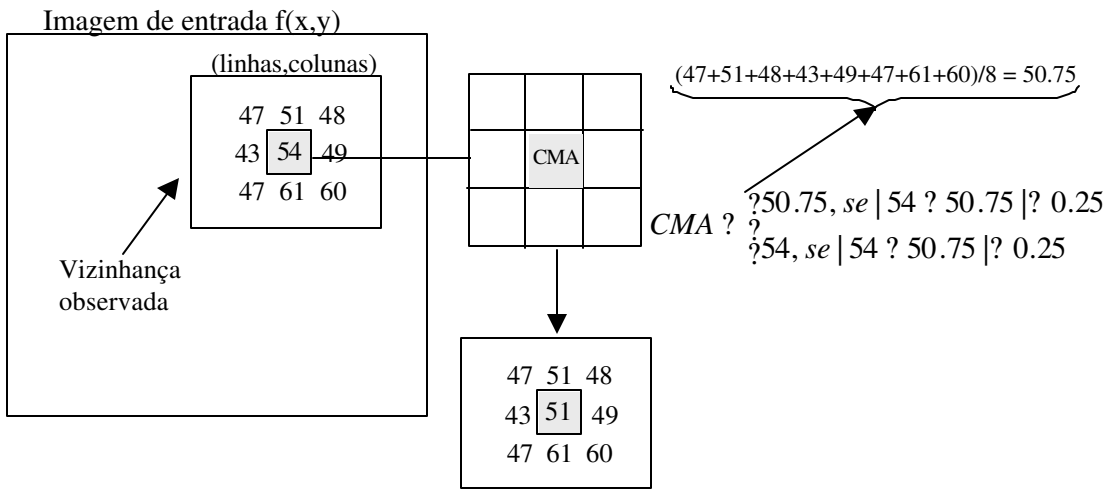


Figura 3.11 Exemplo numérico do filtro CMA.

Repete-se o processo até que toda a imagem seja varrida. A Figura 3.12 mostra a saída do filtro condicional de média aritmética.

Figura 3.12 Saída do filtro CMA.

Ao observarmos a Figura 3.12 notamos que a imagem na saída ficou bem melhor do que a imagem de entrada, porém não temos uma recuperação muito satisfatória. Neste caso o tempo gasto para o processamento foi de 18 segundos e o erro MAE apresentado pela imagem na saída do filtro foi 5,3.

3.7 Filtro Mediano

O filtro mediano é uma técnica de processamento de sinal não-linear bastante útil na supressão de ruídos em imagens e também na preservação das bordas. É útil na supressão de ruído do tipo "sal e pimenta" (*Salt & Pepper*). O processamento correspondente ao filtro mediano consiste de leitura de uma determinada região da imagem (janela de observação), transformação dessa janela em um vetor, ordenação e substituição do pixel central pelo valor mediano [Neuvo, 1991], [Gonzalez, 1992, p. 191], [Hardie, 1994].

Normalmente, a filtragem mediana, para o caso bidimensional, consiste de uma janela de observação (mxn) que varre a imagem com um número ímpar de pixels. A Figura 3.13 mostra um exemplo numérico desse processo de filtragem.

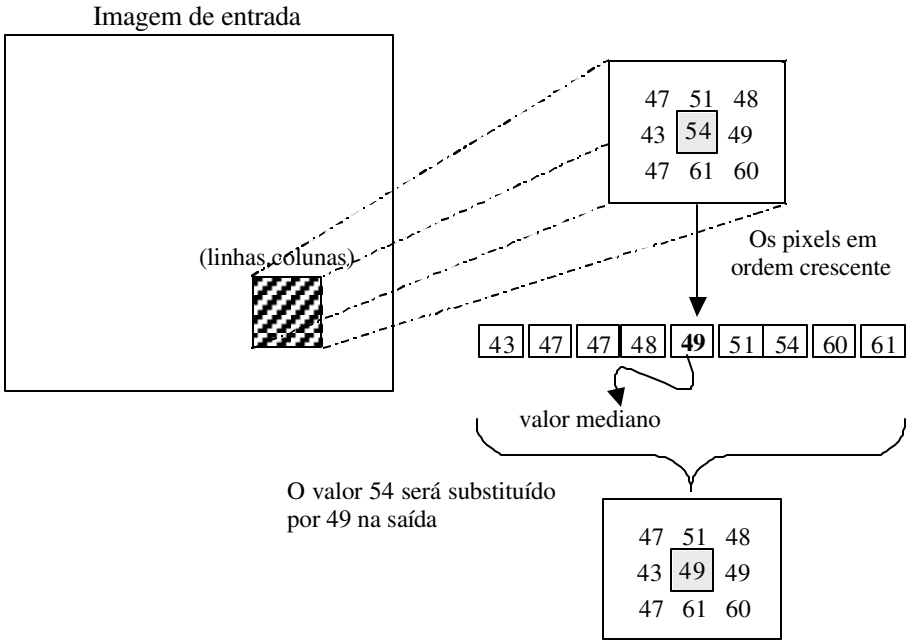


Figura 3.13 Exemplo numérico do filtro Mediano.

O processo se repete, até que toda a imagem seja varrida.

A Figura 3.14 mostra a saída do filtro mediano com janela de observação 3x3.

Figura 3.14 Saída do filtro Mediano.

Nesse caso podemos observar claramente que o filtro recuperou a imagem de maneira satisfatória, o aspecto visual ficou melhor. O tempo gasto para o processamento foi de 18 segundos e o erro MAE apresentado pela imagem na saída do filtro foi 2,7.

3.8 Filtro Mediano Ponderado Central – CWM

O Filtro mediano ponderado central (*Center Weighted Median Filter*) permite um maior controle na preservação dos detalhes e na suavização do ruído. Tem como saída o valor mediano de um conjunto estendido de amostras. Ou seja, a saída do filtro é definida pelo valor mediano do conjunto de amostras. Seleciona-se a janela de observação, transforma-se esta janela em um vetor. Aplica-se a operação ? (significa repetição) à amostra central que vai ser repetida (expandida) de acordo com o valor contido no vetor de pesos correspondente ao centro da janela. A restrição é que $w_?$ (amostra central do vetor de pesos) tem que ser um número inteiro, positivo, ímpar e diferente de zero. No caso de $w_?=1$ (amostra central = 1), temos um filtro mediano [Ko, 1991], [Blume, 1999], [Muneyasu, 1999].

A Figura 3.15, nos ajuda a compreender melhor esse processo.

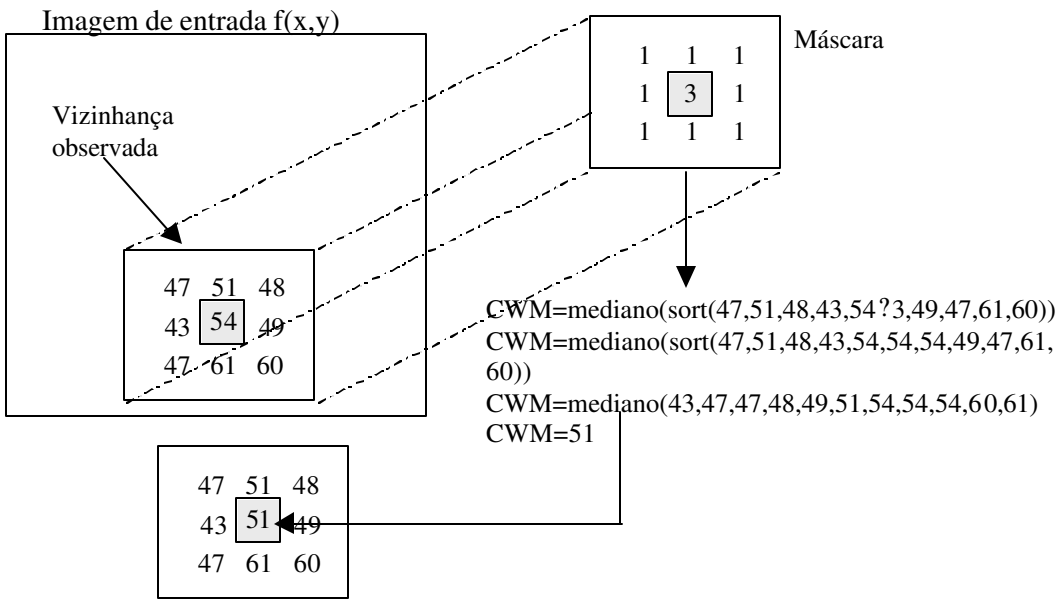


Figura 3.15 Exemplo numérico do filtro CWM.

Tudo que falamos até o momento a respeito de filtro mediano ponderado central pode ser descrito conforme mostra a fórmula de 3.8-1 [Roopert, 1996].

$$Fcwm(x)=median\{x_1, ..., x_??w_?, ..., x_N\} \quad (3.8\text{-}1)$$

Como podemos perceber somente a amostra do centro vai ser repetida, logo a fórmula 3.8-2 abaixo vai produzir o mesmo resultado final [Hardie, 1994].

$$Fcwm(x)=median\{x_{(k)}, x_?, x_{(N-k+1)}\} \quad (3.8\text{-}2)$$

sendo, $k = (N+2- w_?)/2$, para $1? w_?? N$ e $K=1$ para $w_?>N$.

A Figura 3.16 mostra a saída do filtro mediano ponderado central.

Figura 3.16 Saída do filtro CWM.

Ao observarmos a Figura 3.16, percebemos claramente que a imagem na saída do filtro CWM está com um aspecto melhor do que a imagem na entrada (imagem corrompida). O tempo gasto para o processamento foi de 46,4 segundos e o erro MAE apresentado pela imagem na saída do filtro foi 3,2.

3.9 Filtro de Ordem Estatística – OS

A saída do Filtro de ordem estatística (*Order Statistic Filter*, também chamados Filtros seletores de posição, *Rank Selection Filter*), é um dos pixels, do conjunto de amostras observadas. Ou seja, após a leitura da janela de observação, gera-se um vetor de amostras, ordena este vetor e escolhe a amostra de acordo com a ordem desejada [Hardie, 1994], [Barner, 1994], [Barner, 1997], [Arce, 1996], [Paez, 1994], [Pitas, 1991]. Pelo fato de que o ruído tende a ficar nos extremos do vetor, a ordem estatística escolhida geralmente é o pixel central +1 ou –1. No nosso caso como a janela de observação tem tamanho 9 (3x3), teríamos filtro de ordem 4 ou 6.

A Figura 3.17 contém um exemplo numérico que nos ajuda a compreender melhor esse processo.

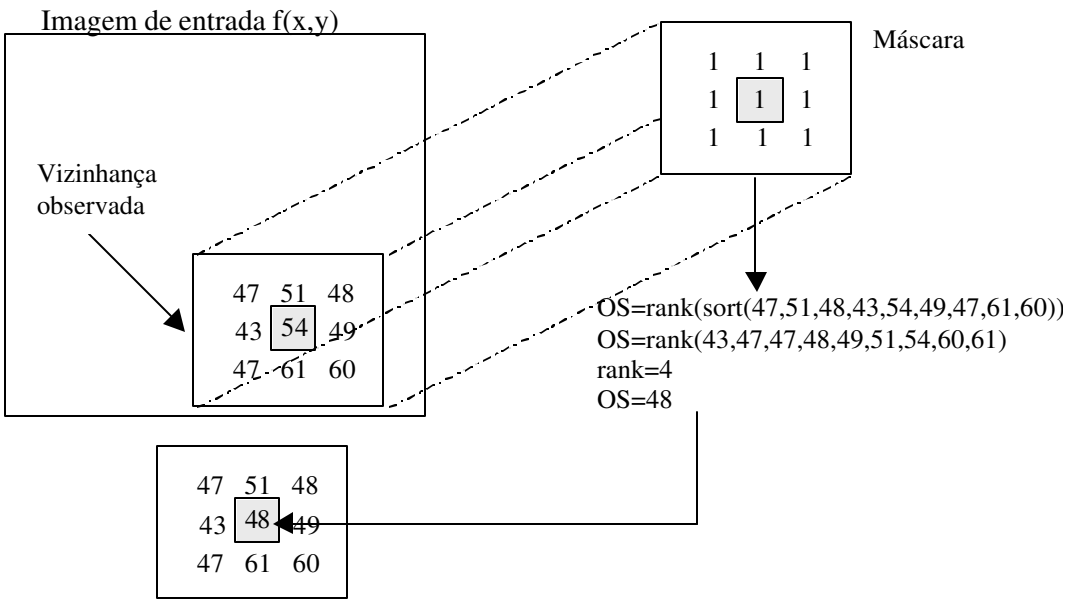


Figura 3.17 Exemplo numérico de filtro OS.

O processo acima é repetido, até que todas as amostras tenham sido percorridas. A tendência dos filtros OS mais sofisticados é preservar mais os detalhes contidos na imagem do que o filtro mediano.

Podemos representar o processamento acima através da fórmula 3.9-1.

$$Fos(x)=(rank(v)(sort\{x_1,x_2, ..., x_N\})), \qquad (3.9\text{-}1)$$

sendo que v significa a ordem do filtro. Ao ordenarmos as amostras selecionadas o ruído tende a migrar para os extremos do vetor, logo no caso de janela de observação 3x3, as ordens 4 e 6 (rank = 4 e rank = 6) tendem a ser mais eficientes.

As principais vantagens dos métodos de filtragem OS são: tendem a preservar melhor as bordas e minimizam o efeito *outliers*. *Outliers* são elementos incomuns dentre um conjunto; é por exemplo se tivéssemos um conjunto "A" qualquer, contendo duas letras e cem números, as letras contidas em "A" são *outliers*. Se considerarmos uma imagem 512x512 com 256 níveis de cinza, contaminada por ruído do tipo impulsivo, um pixel que assumisse o valor 255, em determinadas condições, pode ser considerado *outlier* dentro de uma vizinhança.

Filtros como Filtros Condicionais Seletores de Posição (RCRS), Filtro Mediano, Filtro Mediano Ponderado Central (CWM), Filtro de Ordem Estatística Ponderada (WOS), Filtro de Ordem Estatística Ponderada Simples (SWOS) e outros baseados na seleção da ordem estatística podem ser formulados através dos filtros OS [Himayat, 1994].

O filtro mais simples que se pode descrever nesta ampla classe de filtros OS é o filtro onde a saída coloca uma ordem estatística constante, ou seja, coloca na saída a amostra que está em uma posição pré-determinada. Essa classe de filtros inclui o filtro mediano.

A Figura 3.18 mostra a saída do filtro de ordem estatística, janela de observação 3x3, ordem 4.

Figura 3.18 Saída do filtro OS.

Através da Figura 3.18, percebemos claramente que o filtro OS foi bastante eficiente, a imagem ficou bem parecida com a original. O tempo gasto para o processamento foi de 2,3 segundos e o erro MAE apresentado pela imagem na saída do filtro foi 2,4.

3.10 Filtro de Ordem Estatística Ponderada Simples – SWOS

O Filtro de ordem estatística ponderada simples (*Simply Weighted Order Statistic Filter*), é uma subclasse do filtro WOS [Hardie, 1994]. Nesse filtro a janela de observação contém valores diferentes de um somente na amostra central, o que vai significar a repetição do pixel central. Ou seja, após a leitura da janela de observação, aplica-se à operação ? (significa repetição), a amostra vai ser expandida (repetida) de acordo com o seu valor correspondente do vetor de pesos. Logo após ordena-se o vetor resultante e seleciona a amostra da ordem desejada.

A Figura 3.19 contém um exemplo numérico para melhor compreensão desse processo de filtragem.

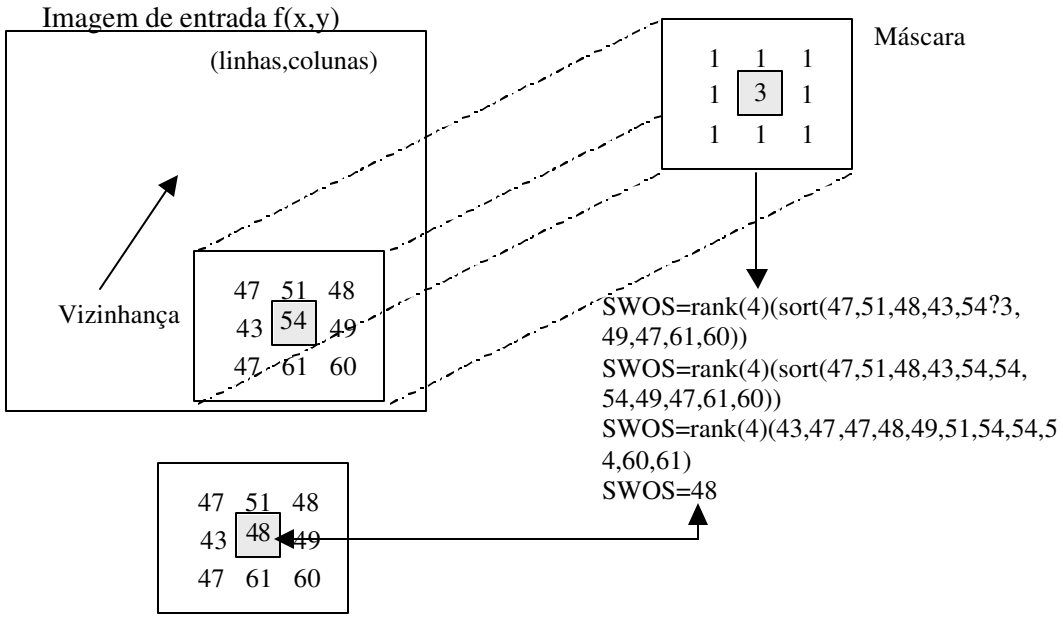


Figura 3.19 Exemplo numérico do filtro SWOS.

O processo acima é repetido, até que todas as amostras tenham sido percorridas.

Podemos representar o processamento acima através da fórmula 3.10-1.

$$Fswos(x)=rank(v)\{x_1,x_2, \ldots, x_?{?}w_?, \ldots x_N\}, \qquad (3.10\text{-}1)$$

Onde x são os elementos da janela de observação e w são os elementos do vetor de pesos.

Como só temos repetição da amostra central, a fórmula a seguir também produzirá a mesma saída.

$$Fswos(x)=median\{x_{(k)}, x_{?},x_{(l)}\} \quad (3.10\text{-}2)$$

Sendo que, 1 ? k ? *l* ? N.

A Figura 3.20 mostra a saída do filtro de ordem estatística ponderada simples.

Figura 3.20 Saída do filtro SWOS.

Analisando a Figura 3.20, observamos que houve uma recuperação parcial da imagem observada e podemos perceber claramente uns pontos pretos que não foram restaurados. O tempo gasto para o processamento foi de 48 segundos e o erro MAE apresentado pela imagem na saída do filtro foi 6,8.

3.11 Filtro de Ordem Estatística Ponderada – WOS

O Filtro de ordem estatística ponderada (*Weighted Order Statistic Filter*) [Neuvo, 1994], [Neuvo, 1991], [Ropert, 1996], [Chakrabarti, 1997], [Yu, 1994], [Flaing, 1998], é um filtro derivado do filtro OS. Esse filtro foi definido devido a limitação dos

filtros SWOS, que só tem ponderação da amostra central. No caso de filtros WOS todas as amostras podem ser ponderadas, ou seja, o vetor de pesos pode ter valores diferentes de um em todas as posições, o que significa a repetição do pixel correspondente. Ou seja, após a leitura da janela de observação, com as mesmas dimensões da janela de pesos (w), aplica-se a operação ? (significa repetição). Os pixels da janela de observação serão repetidos de acordo com o seu peso correspondente. As amostras vão ser expandidas (repetidas), ordenadas e no vetor resultante seleciona-se a amostra da ordem desejada.

A Figura 3.21, nos fornece um exemplo numérico desse processamento.

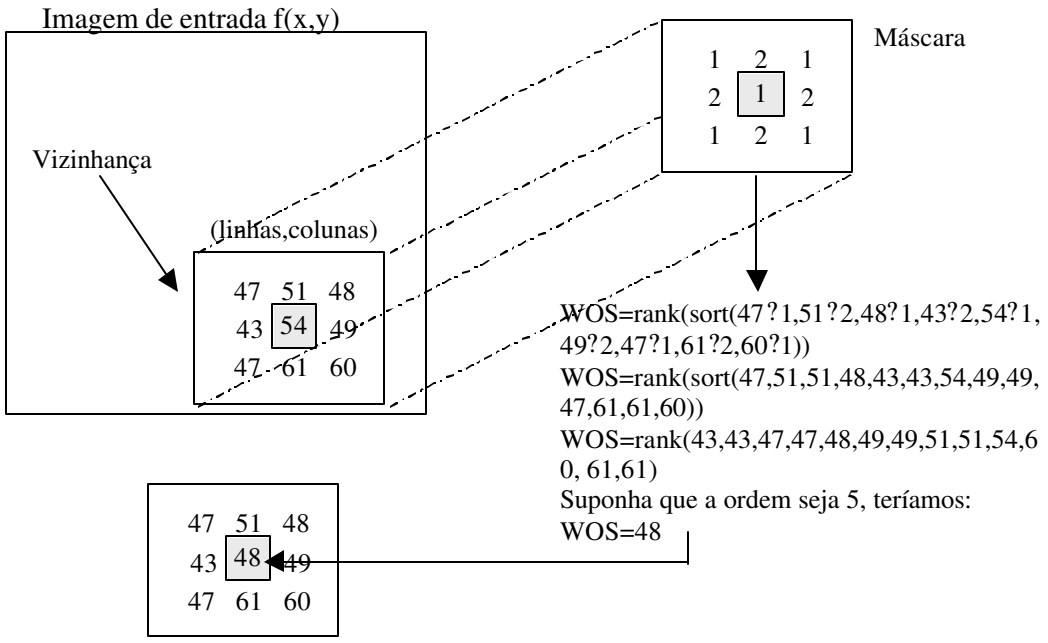


Figura 3.21 Exemplo numérico do filtro WOS.

O processo acima é repetido, até que todas as amostras tenham sido percorridas.

Podemos representar o processamento através da fórmula 3.11-1.

$$Fwos(x)=rank(v)\{x?w\}, \qquad (3.11\text{-}1)$$

onde x são os elementos da janela de observação, v é a ordem do filtro e w são os elementos do vetor de pesos (tem as mesmas dimensões da janela de observação).

A Figura 3.22 mostra a saída do filtro de ordem estatística ponderada.

Figura 3.22 Saída do filtro WOS.

Podemos perceber pela Figura 3.22 que nesse caso o filtro WOS foi bem eficiente, a imagem na saída do filtro ficou bem próxima da original. O tempo gasto para o processamento foi de 310 segundos e o erro MAE apresentado pela imagem na saída do filtro foi 3,9. Aqui temos um 'problema', o filtro demorou consideravelmente para recuperar a imagem, isto nem sempre é aceitável (satisfatório).

3.12 Filtro Condicional Mediano – RCM

A saída do Filtro condicional mediano (*Rank Conditioned Median Filter*) [Hardie, 1994], é calculada da seguinte forma: inicialmente seleciona-se a janela de observação contendo as amostras a serem observadas, transforma-se em um vetor e ordena, armazena a posição da amostra central em r_0 (nova posição da amostra central). Caso r_0 esteja entre k e (N – k + 1), a saída do filtro será o próprio x_0 (amostra central) caso contrário, a saída será o pixel mediano do vetor ordenado. O k é qualquer número de um até N, sendo que, N é o número de elementos da janela de observação. As amostras da janela de observação são representadas por x^r. Com isto para a implementação desse filtro temos uma fórmula básica:

$$RCM \begin{cases} x_?, & \text{se } k \,?\, r_? \,?\, N-k+1 \\ \text{mediano}(x^r), & \text{se } r_? < k \text{ ou } r_? > N-k+1 \end{cases} \tag{3.12-1}$$

Para o exemplo abaixo tomaremos k = 4 e N = 9, janela de observação 3x3. Nesse caso RCM é igual ao pixel mediano do vetor ordenado, se a posição original da amostra central é diferente de 4, 5, ou 6, caso contrário RCM é a amostra central $x_?$.

$$RCM \begin{cases} x_?, & \text{se } 4 \,?\, r_? \,?\, 6 \\ \text{mediano}(x^r), & \text{se } r_? < 4 \text{ ou } r_? > 6 \end{cases} \tag{3.12-2}$$

A Figura 3.23 mostra-nos um exemplo numérico que detalha melhor esse processamento.

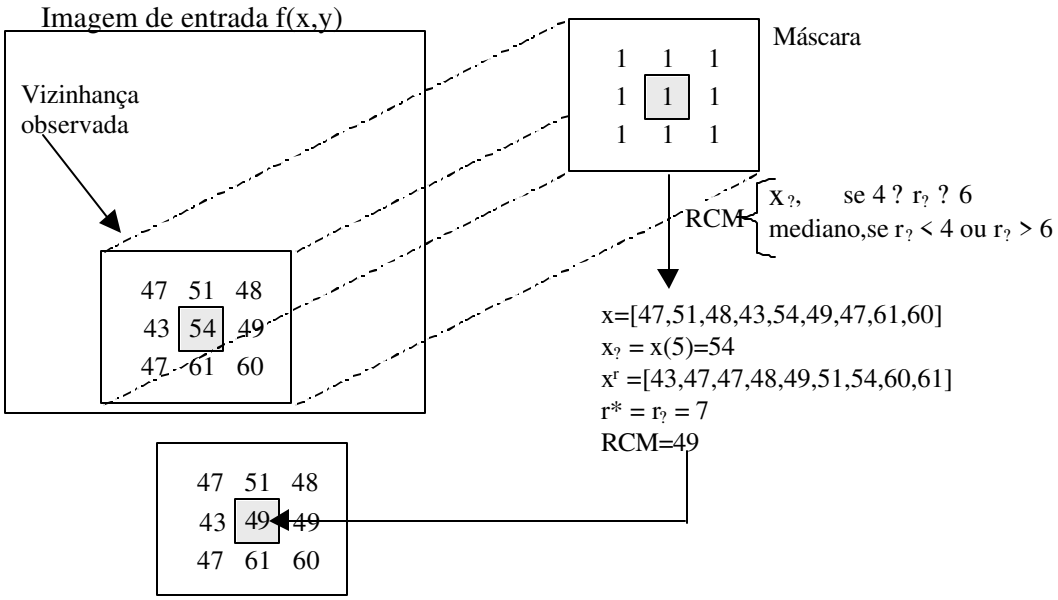


Figura 3.23 Exemplo numérico do filtro RCM.

Como nos filtros anteriores, o processo é repetido, até que todos os pixels tenham sido varridos.

A Figura 3.24 mostra a saída do filtro condicional mediano.

Figura 3.24 Saída do filtro RCM.

A Figura 3.24 mostra a saída do filtro RCM. Mais uma vez a mesma não é muito satisfatória visualmente. O tempo gasto para o processamento foi de 37 segundos e o erro MAE apresentado pela imagem na saída do filtro foi 5,3.

3.13 Filtro Condicional Seletor de Posição – RCRS

Os Filtros condicionais seletores de posição (*Rank Conditioned Rank Selection Filter*) usam a ordem das amostras escolhidas como base para a seleção da posição ou ordem estatística da saída. O número de amostras da janela de observação de entrada determina a ordem do filtro, que pode variar de 0 até o número total de amostras (N) da janela de observação. Filtros de baixa ordem oferecem um bom desempenho e são relativamente simples de otimizar e implementar. O aumento da ordem do filtro tem como conseqüência o aumento de sua complexidade. Trata-se de uma categoria de filtros mais geral do que os OS. A principal diferença está na informação utilizada na decisão da ordem estatística (posição da amostra) que será selecionada na saída [Hardie, 1994].

"A extrema necessidade de recuperação da informação contida em imagens degradadas e o desempenho dos filtros RCRS, em minimizar o ruído sem alterar a estrutura da imagem, é que vem motivando o estudo dessa técnica de filtragem para restauração de imagens" [Pérez, 2000].

De posse de um conjunto de amostras (janela de observação), pretende-se formar uma estimativa da amostra desejada $\hat{d}_?(n)$, cuja localização é ? dentro da janela. A amostra desejada $\hat{d}_?(n)$, é tal que 1???N. Por definição, a saída $\hat{d}_?(n)$ do filtro RCRS é necessariamente uma amostra do vetor de observação x(n).

Detalharemos agora a filtragem RCRS, a partir de um filtro OS. Iniciaremos pelo processo de geração da função ótima do filtro (Sopt).

Seleciona-se a janela de observação {x(n)}, transforma-se em vetor, ordena-se (x^r) e gera-se o vetor contendo a posição das amostras (vetor característico r). O vetor característico dos filtros RCRS contém a posição da amostra central no vetor ordenado. Gera-se então o vetor P(m) que contém a diferença entre o vetor ordenado e a amostra desejada. Acumula-se P(m) no vetor erro cumulativo (R). Isso é feito considerando-se a posição da amostra central no vetor ordenado. Com isso se nossa janela de observação tem N elementos, teremos então N erros cumulativos. Após varrer a imagem inteira acumulando erro, armazena-se o erro mínimo em Sopt.

A seguir mostraremos um exemplo numérico de como obter a função ótima descrita acima. Vamos fazer isso a partir de um exemplo numérico, para melhor compreensão.

Imagem de entrada f(x,y) Imagem desejada $d(n)$

		⋮		
	163	164	0	157
	163	163	156	161
⋯	164	162	0	165 ⋯
	171	160	162	170
		⋮		

		⋮		
	163	164	154	157
⋯	163	163	156	161 ⋯
	164	162	158	165
	171	160	162	170
		⋮		

Tabela 3.4 Imagem 512x512.

A Tabela 3.4 mostra a imagem de entrada f(x,y) (imagem com ruído) e a imagem desejada (imagem original, $d(n)$) ambas de tamanho 512x512. A partir deste momento vamos considerar somente a matriz 4x4, para facilitar a compreensão do processo de filtragem RCRS.

Na Tabela 3.5, temos a imagem de entrada (4x4) que já foi aumentada (duplicou-se as primeiras e últimas linhas e colunas), para evitar o efeito de bordas (ver Anexo 1, p. 143-148).

Imagem de entrada f(x,y)

170	171	160	162	170	171
157	163	164	0	157	163
161	163	163	156	161	163
165	164	162	0	165	164
170	171	160	162	170	171
157	163	164	0	157	163

Imagem desejada $d(n)$

163	164	154	157
163	163	156	161
164	162	158	165
171	160	162	170

Tabela 3.5 Fatia das imagens de entrada e desejada.

Obteremos os vetores de observação a partir da matriz de entrada f(x, y). Utilizou-se neste caso uma janela de observação 3x3. Posicionaremos a janela de observação inicialmente sobre a primeira amostra e depois varreremos toda a imagem. Cada vetor de observação conterá nove elementos. O centro do vetor de observação corresponde ao pixel a ser substituído e o valor equivalente da imagem desejada corresponde ao valor desejado.

Vetores de observação	Vetores ordenados	Saída desejada
170 171 160 157 163 164 161 163 163	157 160 161 163 163 163 164 170 171	163
171 160 162 163 164 0 163 163 156	0 156 160 162 163 163 163 164 171	164
160 162 170 164 0 157 163 156 161	0 156 157 160 161 162 163 164 170	154
162 170 171 0 157 163 156 161 163	0 156 157 161 162 163 163 170 171	157
157 163 164 161 163 163 165 164 162	157 161 162 163 163 163 164 164 165	163
163 164 0 163 163 156 164 162 0	0 0 156 162 163 163 163 164 164	163
164 0 157 163 156 161 162 0 165	0 0 156 157 161 162 163 164 165	156
0 157 163 156 161 163 0 165 164	0 0 156 157 161 163 163 164 165	161
161 163 163 165 164 162 170 171 160	160 161 162 163 163 164 165 170 171	164
163 163 156 164 162 0 171 160 162	0 156 160 162 162 163 163 164 171	162
163 156 161 162 0 165 160 162 170	0 156 160 161 162 162 163 165 170	158
156 161 163 0 165 164 162 170 171	0 156 161 162 163 164 165 170 171	165
165 164 162 170 171 160 157 163 164	157 160 162 163 164 164 165 170 171	171
164 162 0 171 160 162 163 164 0	0 0 160 162 162 163 164 164 171	160
162 0 165 160 162 170 164 0 157	0 0 157 160 162 162 164 165 170	162
0 165 164 162 170 171 0 157 163	0 0 157 162 163 164 165 170 171	170

Tabela 3.6 Vetores observação, observação ordenado e saída desejada.

Após obtenção do Vetor de observação ordenado e da saída desejada, vamos determinar o vetor diferença, que é a diferença entre vetor observação e a saída desejada. Este vetor é chamado P(m) e é obtido através da Fórmula 3.13-1.

$$P(m) = \begin{bmatrix} \left| d(n_m) - x_{(1)}(n_m) \right|^n \\ \left| d(n_m) - x_{(2)}(n_m) \right|^n \\ \vdots \\ \left| d(n_m) - x_{(N)}(n_m) \right|^n \end{bmatrix} \qquad (3.13\text{-}1)$$

Voltando ao exemplo vamos ter os vetores a seguir. Na Tabela 3.7 temos um vetor chamado posição da amostra, que é exatamente a posição da amostra central no vetor ordenado. O mesmo vai indicar quais vetores serão somados para gerar o vetor erro cumulativo.

Vetor diferença - P(m)									Posição Amostra
6	3	2	0	0	0	1	7	8	4
164	8	4	2	1	1	1	0	7	8
154	2	3	6	7	8	9	10	16	1
157	1	0	4	5	6	6	13	14	3
6	2	1	0	0	0	1	1	2	5
163	163	7	1	0	0	0	1	1	7
156	156	0	1	5	6	7	8	9	3
161	161	5	4	0	2	2	3	4	5
4	3	2	1	1	0	1	6	7	6
162	6	2	0	0	1	1	2	9	4
158	2	2	3	4	4	5	7	12	1
165	9	4	3	2	1	0	5	6	7
14	11	9	8	7	7	6	1	0	9
160	160	0	2	2	3	4	4	11	3
162	162	5	2	0	0	2	3	8	6
170	170	13	8	7	6	5	0	1	8

Tabela 3.7 Vetor diferença e posição da amostra observada.

Agora vamos calcular o vetor erro cumulativo que é a soma dos valores do vetor diferença, observando-se a posição das amostras, ou seja, somaremos os vetores cuja posição da amostra central for igual. Como temos, nove posições possíveis teremos nove totais.

Vetor erro cumulativo									Erro mínimo
r = 1 ✍	312	4	5	9	11	12	14	17	28	mim(r1) = 4 ✍ k = 2
r = 2 ✍	0	0	0	0	0	0	0	0	0	mim(r2) = 0 ✍ k = 1
r = 3 ✍	473	317	0	7	12	15	17	25	34	mim(r3) = 0 ✍ k = 3
r = 4 ✍	168	9	4	0	0	1	2	9	17	mim(r4) = 0 ✍ k = 4
r = 5 ✍	167	163	6	4	0	2	3	4	6	mim(r5) = 0 ✍ k = 5
r = 6 ✍	166	165	7	3	1	0	3	9	15	mim(r6) = 0 ✍ k = 6
r = 7 ✍	328	172	11	4	2	1	0	6	7	mim(r7) = 0 ✍ k = 7
r = 8 ✍	334	178	17	10	8	7	6	0	8	mim(r8) = 0 ✍ k = 8
r = 9 ✍	14	11	9	8	7	7	6	1	0	mim(r9) = 0 ✍ k = 9

Tabela 3.8 Vetor erro cumulativo e valor mínimo.

Esse vetor erro cumulativo pode ser descrito pela fórmula 3.13-2.

$$R_{?_m}(m) \leftarrow R_{?_m}(m) + P(M) \tag{3.13-2}$$

Logo após vamos encontrar o valor mínimo para cada r. Esse valor representa o menor erro, a posição deste valor é que dará origem ao k. Quando observamos para r = 9 temos o valor mínimo 33 em duas posições. Nesse caso k pode ser 6 ou 8, o programa implementado vai selecionar o primeiro. Logo, o valor de k seria 6.

O vetor formado pelos valores de k, é chamado de função de classificação ótima "Sopt(r)".

Abaixo temos o algoritmo da seqüência de treinamento, geração do Sopt passo, a passo [Pérez, 2000].

1 - Faça m=1, $S^0_{opt}(r_j) \leftarrow \frac{N-1}{2}$ e $R_{i,j}(0)=0$, para j=1,2,...,$?_M$ e k=1, 2,..., N

2 - Determine o índice do vetor característico $?_m$.

3 - Atualize $R_{?_m}(m)$, pela equação $R_{?_m}(m) \leftarrow R_{?_m}(m-1) + P(m)$.

4 - Faça $S^0_{opt}(r_j) \leftarrow k : R_{?_m,k}(m) \leftarrow \min(R_{?_m,1}(m), R_{?_m,2}(m),..., R_{?_m,N}(m))$.

5 - Se $m = k$ ou o filtro está suficientemente treinado, terminar treinamento, caso contrário, incrementar m e retornar ao passo 2 do algoritmo.

O espaço característico $?_M$ pode ser descrito pela fórmula 3.13-3.

$$?_M \leftarrow \{r_1, r_2, ..., r_{|?_M|}\} \tag{3.13-3}$$

As principais vantagens desse processo são que com essa fase de treinamento teremos um filtro ótimo global para os dados de treinamento e também uma liberdade de escolha do erro.

A Figura 3.25 mostra a função Sopt, gerada através de inserção de densidades de ruído diferentes. Nessa o ruído foi do tipo sal e pimenta, a janela de observação nesse caso é 9x9.

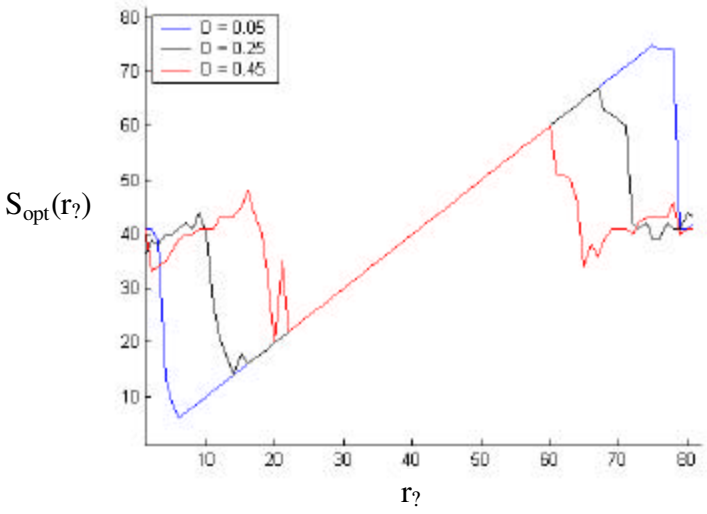


Figura 3.25 Função Sopt (ruído sal e pimenta).

Ao inserirmos outro tipo de ruído (gaussiano) na imagem temos uma função Sopt, um pouco diferente, é o que mostra a Figura 3.26.

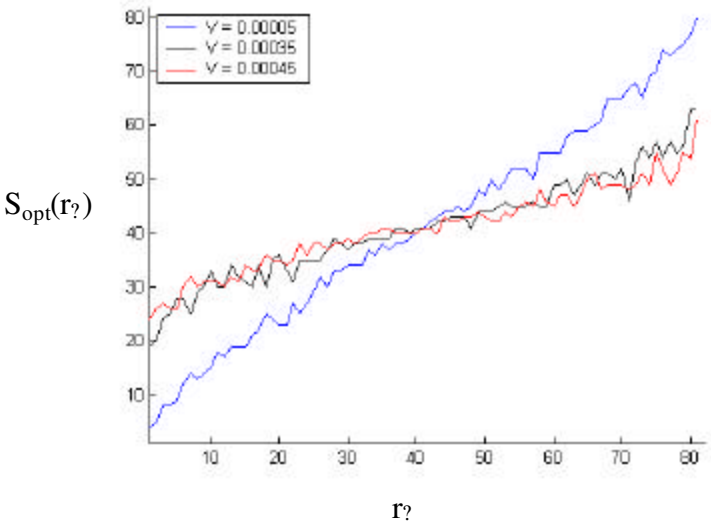


Figura 3.26 Função Sopt (ruído gaussiano).

Gerou-se esta função ótima Sopt como mostra o Anexo 5 (ver Anexo 5, p. 173-177), para que fosse utilizada na restauração das imagens.

Finalmente a saída do filtro RCRS é dada pela seguinte equação:

$$F_{RCRS}\ (x)\ ?\ x_{(Sopt\ (r*))} \qquad (3.13\text{-}4)$$

De posse da função ótima podemos agora recuperar a imagem, podemos agora passar para a fase de filtragem da imagem, que será feita da seguinte forma. Inicialmente temos a leitura do vetor de observação, ordena-se o mesmo gerando os vetores ordenados e armazena-se a posição da amostra central (r). Em nosso exemplo teríamos:

Vetores ordenados	**Posição Amostra (r)**	**Erro mínimo (k)**
157 160 161 163 163 163 164 170 171	4	2
0 156 160 162 163 163 163 164 171	8	1
0 156 157 160 161 162 163 164 170	1	3
0 156 157 161 162 163 163 170 171	3	4
157 161 162 163 163 163 164 164 165	5	5
0 0 156 162 163 163 163 164 164	7	6
0 0 156 157 161 162 163 164 165	3	7
0 0 156 157 161 163 163 164 165	5	8
160 161 162 163 163 164 165 170 171	6	9
0 156 160 162 162 163 163 164 171	4	
0 156 160 161 162 162 163 165 170	1	
0 156 161 162 163 164 165 170 171	7	
157 160 162 163 164 164 165 170 171	9	
0 0 160 162 162 163 164 164 171	3	
0 0 157 160 162 162 164 165 170	6	
0 0 157 162 163 164 165 170 171	8	

Tabela 3.9 Vetor ordenado, posição amostra e erro mínimo.

Como mostra a tabela marcada acima, agora podemos selecionar a saída do filtro, considerando o vetor ordenado, posição amostra central no vetor ordenado e o erro mínimo.

Este processo é feito da seguinte forma: consideremos a primeira linha, nesta a amostra central está na posição 4 (r = 4), o erro mínimo encontra-se na posição 4 (k = 4), logo a saída do filtro será 163. Esse procedimento é feito para o restante das amostras.

Portanto a partir da função de classificação ótima obtida na fase de treinamento do filtro temos:

Imagem saída RCRS			
163	164	**156**	157
163	163	156	161
164	162	**156**	165
171	160	162	170

Imagem original			
163	164	154	157
163	163	156	161
164	162	158	165
171	160	162	170

Tabela 3.10 Valor numérico imagens recuperada e original.

Transformamos esses valores em imagens para observarmos melhor a diferença entre a imagem desejada e a saída do filtro RCRS. Pode-se notar na Figura 3.27 que a diferença é quase imperceptível.

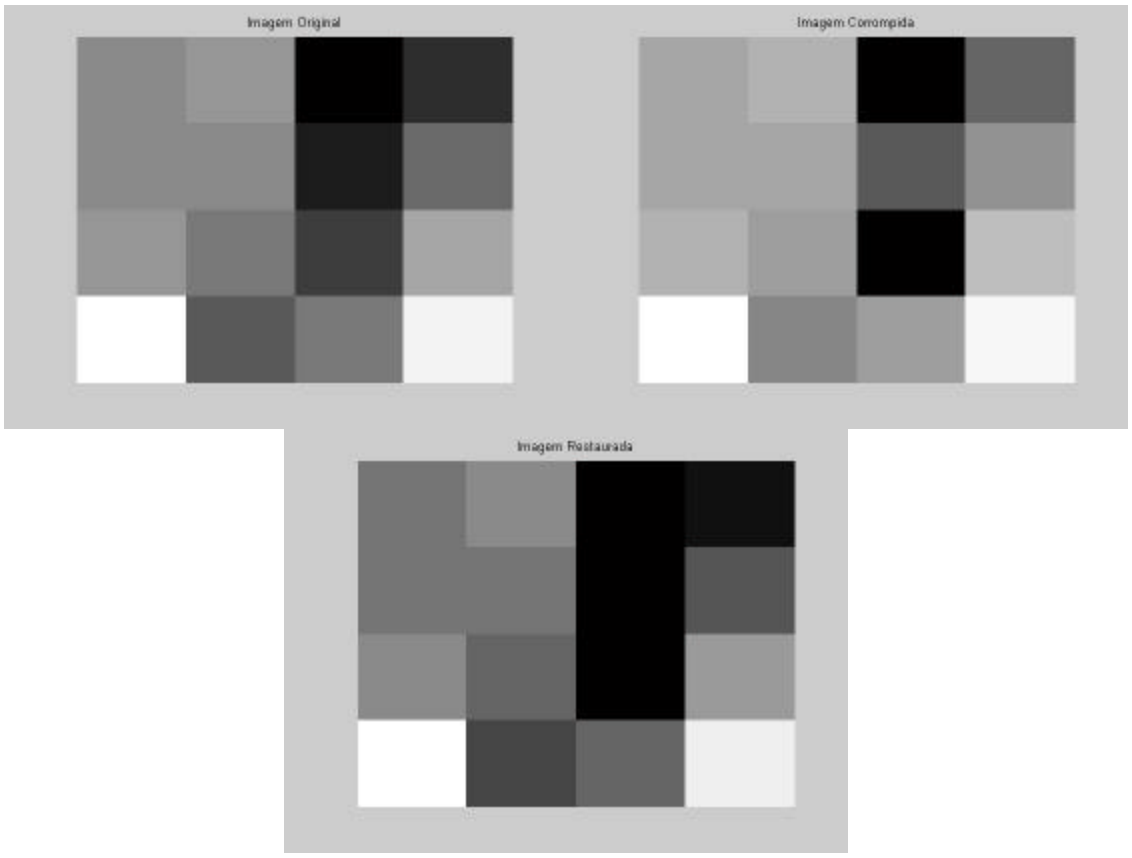


Figura 3.27 Imagens original, corrompida e recuperada.

Podemos perceber claramente, através desse exemplo, que o filtro recuperou a imagem, com quase perfeição; somente dois valores não foram totalmente recuperados. Geralmente a recuperação da imagem por esse processo é bem satisfatória. Nesse nosso exemplo consideramos uma fatia da imagem bem pequena, para que pudéssemos ter uma idéia do processo como um todo. Agora podemos perceber claramente que temos aqui processamento pesado. Em nosso exemplo consideramos uma imagem 4x4, imagine quanto processamento seria necessário se a mesma fosse 512x512.

A Figura 3.28, nos fornece um exemplo numérico desse processamento.

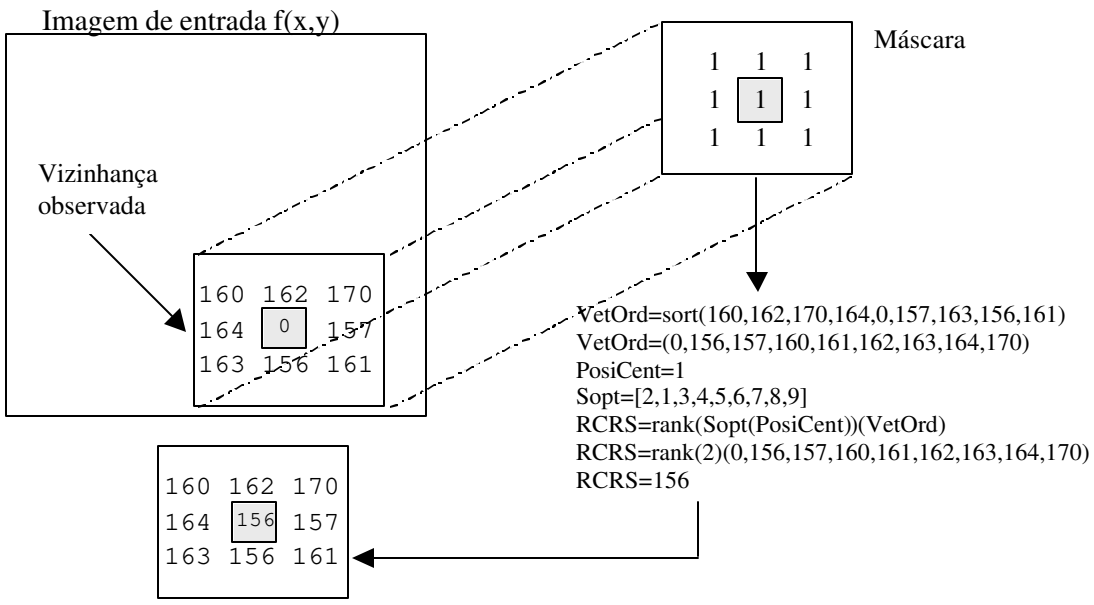


Figura 3.28 Exemplo numérico do filtro RCRS.

A Figura 3.29 mostra a saída do filtro condicional seletor de posição.

Figura 3.29 Saída do filtro RCRS.

A Figura 3.29 mostra que apesar da imagem apresentar alguns pontinhos não pertencentes a imagem desejada, a mesma foi quase totalmente recuperada. O tempo gasto para o processamento foi de 27,6 segundos e o erro MAE apresentado pela imagem na saída do filtro foi 0,6. Isso é mais perceptível ao observarmos o erro (MAE), o mais baixo até o momento. Esse filtro tem resposta ainda melhor quando trabalhamos

com janelas de observação maiores. Porém não podemos esquecer que esse fato implica em aumento do custo computacional.

A função ótima (Sopt) utilizada foi gerada na restauração da imagem elaine.tiff ruído do tipo sal e pimenta de densidade 0.02.

No capítulo 4, faremos algumas simulações e mostraremos alguns resultados obtidos através das mesmas. Utilizaremos para isso os filtros mostrados neste capítulo. Inicialmente analisaremos a eficiência, em seguida o desempenho e a complexidade computacional.

CAPÍTULO 4

APRESENTAÇÃO E DISCUSSÃO DE RESULTADOS OBTIDOS

Nesse capítulo, serão descritas algumas simulações realizadas, bem como os resultados obtidos, tendo como base os filtros apresentados no capítulo 3.

Inicialmente temos testes realizados que comprovam a eficiência dos filtros na recuperação de imagens degradadas. Logo após analisaremos o desempenho dos mesmos. Finalmente faremos uma análise da complexidade computacional de cada filtro observado.

Para os testes utilizaremos várias imagens, encontradas em *Waterloo BragZone* (http://links.uwaterloo.ca/bragzone.base.html) e *The USC-SIPI Image Database* (http://sipi.usc.edu/services/database/Database.html). As mesmas tem tamanhos variados e estão representadas em níveis de cinza (ver Anexo 4, p.165-172).

Após implementação e teste com vários filtros, constatamos o que já foi dito anteriormente, ou seja, nenhum dos filtros implementados retorna uma resposta

satisfatória para todos os tipos de aplicações. O sucesso do processo de filtragem está diretamente ligado com a aplicação.

Antes de entrar efetivamente nos testes propostos, vamos citar a função utilizada para corromper a imagem, para que a simulação seja possível, visto que só temos a imagem original.

Para isso temos a função IMNOISE (ver Anexo 2, p.149-160) que adiciona ruído à imagem, ou seja, degrada a imagem com algum tipo de ruído. Essa função possui alguns parâmetros básicos como a imagem a ser corrompida e o tipo (Type) de ruído. Esse tipo pode ser: *Salt & Pepper* (sal e pimenta) e *Gaussian* (gaussiano).

Para cada tipo a função requer outros parâmetros. No caso de ruído do tipo sal e pimenta temos um parâmetro adicional que é a densidade do ruído (D), com isso a chamada da função ficaria da seguinte forma: imgcorromp = imnoise(img, '*salt & pepper*', D). Já para ruído gaussiano teremos dois parâmetros adicionais que são a média (M) e a variância (V), de forma que a chamada para a função ficaria assim: imgcorromp = imnoise(img, '*gaussian*', M, V).

Nesse capítulo, para verificar o desempenho dos algoritmos utilizados serão calculados o erro médio absoluto (MAE – *Mean Absolute Error*) e o melhoramento da relação sinal ruído (ISNR - *Improvement in Signal to Noise Ratio*), descritos no Capítulo 3.

Parâmetros para os filtros:

- Para o filtro MAP utilizou-se a máscara mostrada na Figura 3.11, acrescentou-se 1 para janelas de observação maiores que 3x3;
- No caso do filtro CMA o limiar será sempre a variância ou densidade do ruído que corrompeu a imagem;
- Para o filtro CWM utilizou-se a máscara mostrada na Figura 3.17, acrescentou-se 1 para janelas de observação maiores que 3x3;

- Para os filtros OS, SWOS e WOS utilizou-se a ordem referente ao pixel central (do vetor de pesos) menos um, ou seja, 3x3 ⇨ ordem 4, 5x5 ⇨ ordem 12, 7x7 ⇨ ordem 24 e 9x9 ⇨ ordem 40;

- Para o filtro SWOS utilizou-se a máscara mostrada na Figura 3.19, acrescentou-se 1 para janelas de observação maiores que 3x3;

- Para o filtro WOS utilizou-se a máscara mostrada na Figura 3.21, acrescentou-se 1 para janelas de observação maiores que 3x3;

- Para o filtro RCM o valor de k utilizado foi a posição do pixel central (no vetor de pesos) menos um, ou seja, 3x3 ⇨ k = 4, 5x5 ⇨ k = 12, 7x7 ⇨ k = 24 e 9x9 ⇨ k = 40);

- Para o filtro RCRS utilizou-se as funções Sopt que estão no Anexo 5 (ver Anexo 5, p.173-177).

A partir desse momento analisaremos os resultados das simulações, obtidos através da aplicação dos filtros propostos em imagens degradadas por ruído.

4.1 Gráficos de Desempenho dos Filtros implementados

Utilizaremos gráficos para que possamos comparar o desempenho dos filtros implementados.

4.1.1 Filtros e imagens com ruídos sal e pimenta e gaussiano

Consideraremos ruído do tipo sal e pimenta, com densidades que variam entre [0.05, 0.5]. Utilizou-se uma janela de observação de tamanho 3x3. A imagem utilizada foi lena (ver Anexo 1, p.143, Figura A4.1).

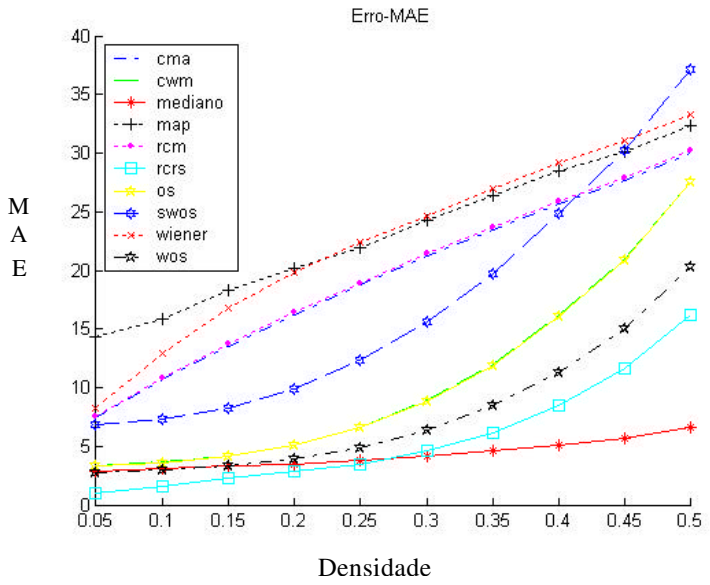


Figura 4.1 Vetores de erro, ruído sal e pimenta, máscara 3x3.

Para ruído sal e pimenta [0.05, 0.5], a imagem na entrada do filtro apresenta erro MAE de 6, 12, 19, 25, 32, 38, 44, 51, 57, 64 respectivamente.

Podemos notar através do gráfico (Figura 4.1) que quando aumentamos a densidade do ruído, o erro MAE aumenta, o que já era de se esperar. Aqui temos o erro MAE compreendido entre [1, 40], na saída dos filtros, sendo que inicialmente estava no intervalo [6, 64]. Notamos também que o aumento do erro apresentado pelo filtro mediano ficou bem baixo. Inicialmente os filtros SWOS, CMA, RCM, WOS e MAP apresentaram um erro MAE maior do que o apresentado pela imagem corrompida.

Os filtros, Mediano, RCRS, WOS, OS, apresentam os melhores resultados, respectivamente, para esse caso.

A seguir utilizaremos as mesmas condições acima para recuperar uma imagem corrompida por ruído do tipo gaussiano com média 0 e variâncias entre [0.05, 0.5]. A imagem utilizada foi lena.

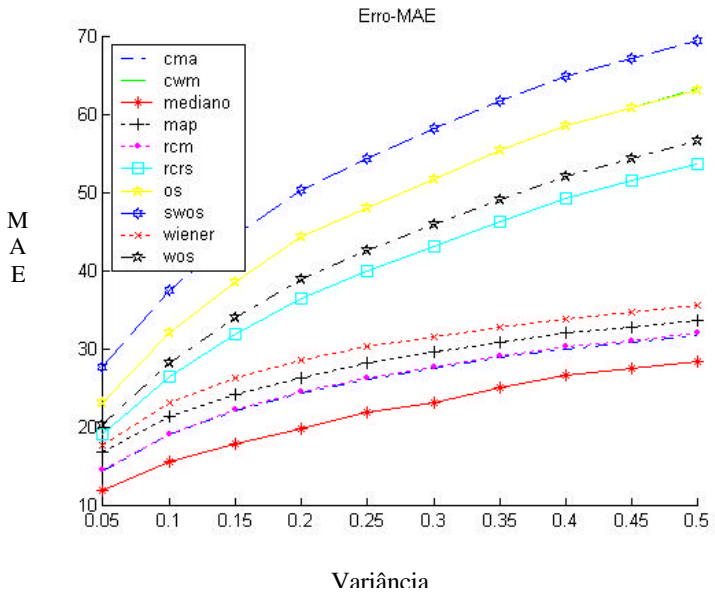


Figura 4.2 Vetores de erro, ruído gaussiano, máscara 3x3.

Para ruído gaussiano com média 0 e variâncias entre [0.05, 0.5], a imagem na entrada do filtro apresenta erro MAE de 33, 44, 52, 58, 62, 66, 70, 72, 75, respectivamente.

Também nesse caso podemos notar claramente que apesar da curva de erro ser um pouco diferente que quando aumentamos a relação sinal ruído, o erro também aumenta. Nesse caso o intervalo de erro MAE foi de [11, 69], na saída dos filtros, sendo que inicialmente o intervalo de erro era de [33, 75]. Ao observarmos os intervalos de erro apresentados temos que todos os filtros apresentaram uma imagem na saída melhor do que a imagem inicial (entrada do filtro).

Os filtros, Mediano, RCRS, RCM, CMA, MAP e Wiener, apresentam os melhores resultados, respectivamente, para esse caso.

Considerado ruído do tipo gaussiano com média 0 e variâncias entre [0.001, 0.01]. Vamos observar como se comporta a nossa curva de erro. A imagem utilizada foi lena.

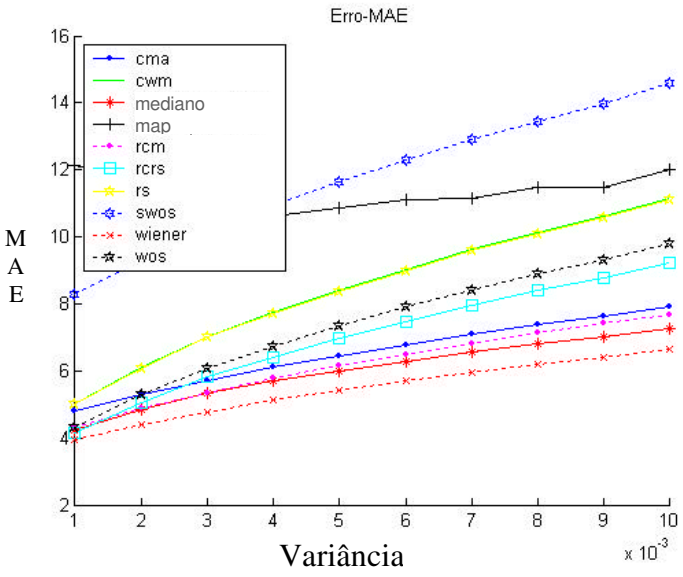


Figura 4.3 Vetores de erro, ruído gaussiano, máscara 3x3.

Na Figura 4.3, como já era de se esperar, o erro apresentado foi menor, ou seja, o intervalo de erro MAE foi [4, 14.5], na saída dos filtros, sendo que inicialmente estava entre [13, 37].

Os filtros, Wiener, Mediano, RCM, CMA e RCRS, apresentam os melhores resultados, respectivamente, para esse caso.

Nesse momento vamos aumentar o tamanho da janela de observação para ver o que ocorrerá com as curvas de erro.

Consideraremos ruído do tipo sal e pimenta, com densidades que vão variar entre [0.05, 0.5], agora a janela de observação assumirá tamanho 5x5. A imagem utilizada foi lena.

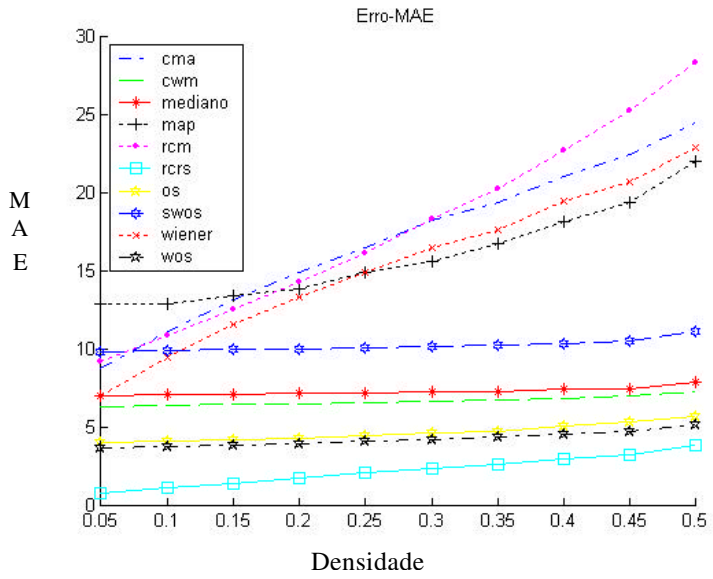


Figura 4.4 Vetores de erro, ruído sal e pimenta, máscara 5x5.

Para ruído sal e pimenta [0.05, 0.5], a imagem na entrada do filtro apresenta erro MAE de 6, 12, 19, 25, 32, 38, 44, 51, 57, 64 respectivamente.

Os filtros CWM, Mediano, Wiener, CMA, RCM, SWOS e MAP inicialmente apresentaram um erro MAE maior do que o inicial, no caso maior que aproximadamente 6. Nesse caso o intervalo de erro MAE foi de 0.7 (RCRS) a 35 (RCM), sendo que inicialmente o intervalo de erro era de [6, 64].

Os filtros, RCRS, WOS, OS, CWM, Mediano e SWOS, apresentam os melhores resultados, respectivamente, para esse caso.

O fato de aumentarmos o tamanho da janela de observação tem como conseqüências imediatas uma imagem melhor na saída e um custo computacional maior (o processamento vai ser mais demorado). Vamos perceber isso claramente nos testes abaixo. Outro fato importante é que esse aumento fez com que principalmente os filtros RCRS, OS e CWM tivessem significativa diminuição do erro.

Para o próximo teste, temos ruído do tipo gaussiano com média 0 e variâncias entre [0.001, 0.01], com janela de observação de tamanho 5x5. A imagem utilizada foi lena.

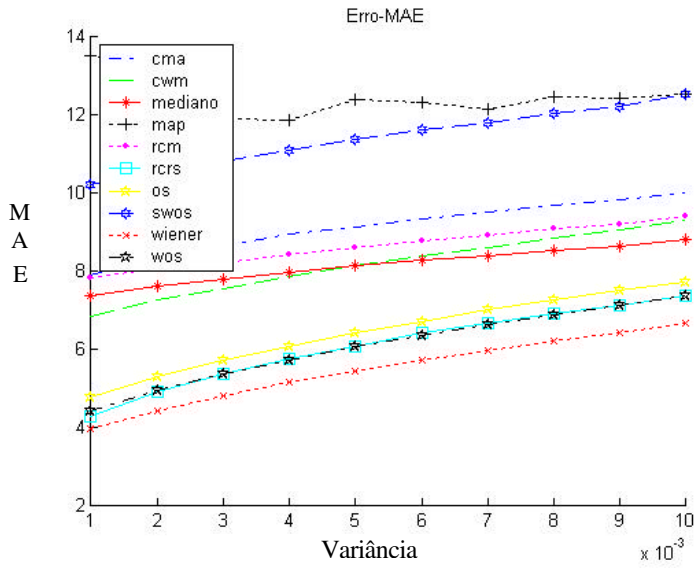


Figura 4.5 Vetores de erro, ruído gaussiano, máscara 5x5.

Para ruído gaussiano com média 0 e variâncias entre [0.001, 0.01], a imagem na entrada do filtro apresenta erro MAE de 6, 9, 11, 12, 14, 16, 17, 18, 19, respectivamente.

Também nesse caso ao compararmos a Figura 4.3 (3x3) e a Figura 4.5 (5x5), inicialmente não temos diferença e quando aumentamos o ruído a diferença fica um pouco mais perceptível. O intervalo de erro MAE passou de [4, 15.5] para [4, 13.6].

Os filtros, Wiener, RCRS, WOS e OS, apresentam os melhores resultados, respectivamente, para esse caso.

Consideraremos a seguir o ruído do tipo sal e pimenta, com densidades variando entre [0.05, 0.5]. A janela de observação assumirá os tamanhos 7x7 e 9x9 respectivamente. A imagem na entrada do filtro é montage.

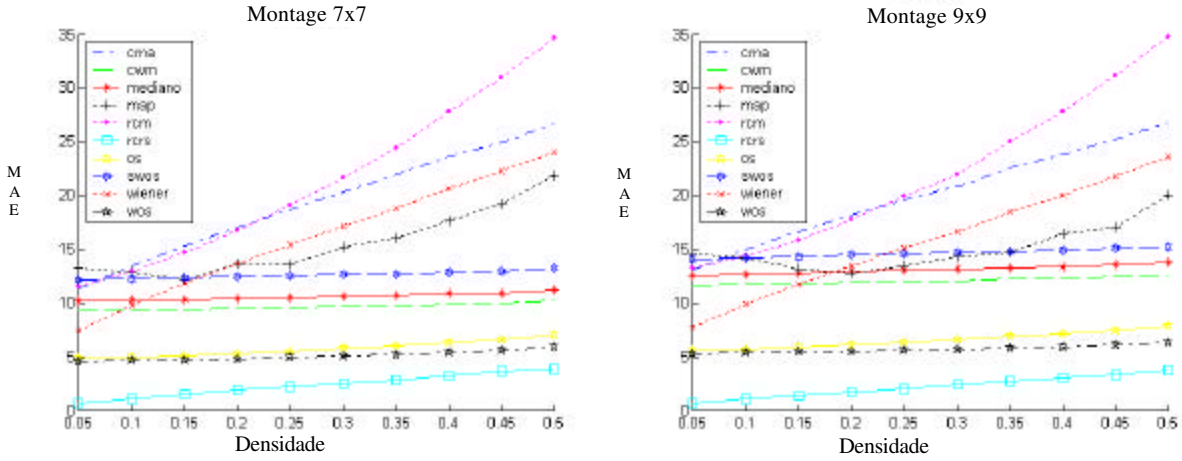


Figura 4.6 Vetores de erro, ruído sal e pimenta, janelas 7x7 e 9x9.

Para ruído sal e pimenta [0.05, 0.5], a imagem na entrada do filtro apresenta erro MAE de 6, 19, 25, 32, 38, 45, 51, 57, 64, respectivamente.

Observando a Figura 4.6, podemos notar uma melhora bem pequena do erro apresentado pelos algoritmos quando aumentamos o tamanho da janela de observação de 7x7 para 9x9.

Os filtros RCRS, WOS e OS apresentam os melhores resultados, para janelas 7x7 e 9x9.

Para o próximo gráfico temos ruído do tipo gaussiano com média 0 e variâncias entre [0.05, 0.5], com janela de observação de tamanho 7x7 e 9x9, respectivamente. A imagem na entrada do filtro é montage.

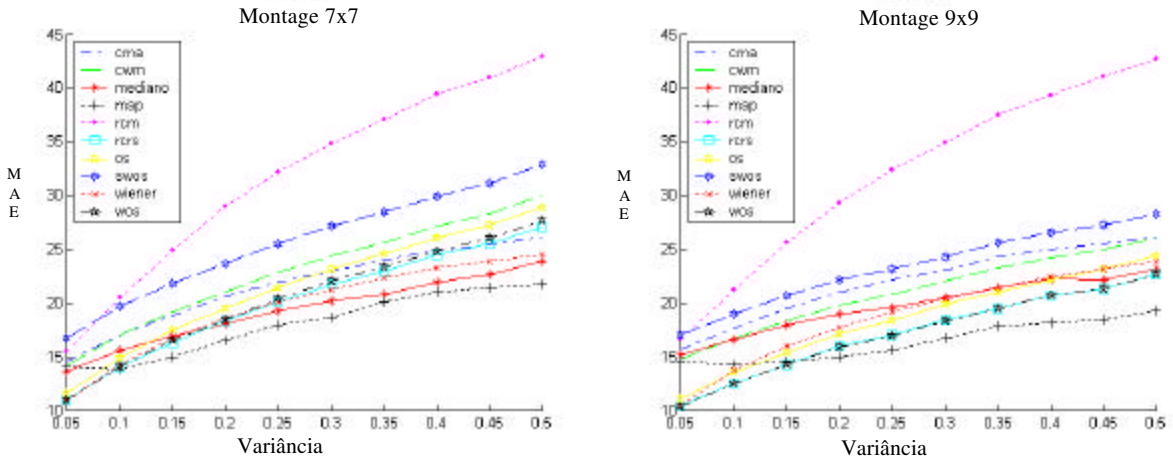


Figura 4.7 Vetores de tempo, ruído gaussiano, janelas 7x7 e 9x9.

Para ruído gaussiano com média 0 e variâncias entre [0.05, 0.5], a imagem na entrada do filtro apresenta erro MAE de 38, 51, 59, 65, 70, 73, 78, 81, 83, respectivamente.

Também nesse caso temos uma pequena melhora, que deve ser considerada juntamente com o aumento do custo computacional que ocorre devido ao aumento do tamanho da janela de observação.

Os filtros apresentaram resultados similares para janela 7x7 e 9x9.

4.1.2 Filtro RCRS, com diferentes janelas.

Agora utilizaremos somente o filtro RCRS na recuperação da imagem (lena) e utilizaremos para isso janelas de observação de tamanhos 3x3, 5x5, 7x7 e 9x9. O ruído inserido foi do tipo sal e pimenta, com densidade variando de [0.05, 0.5].

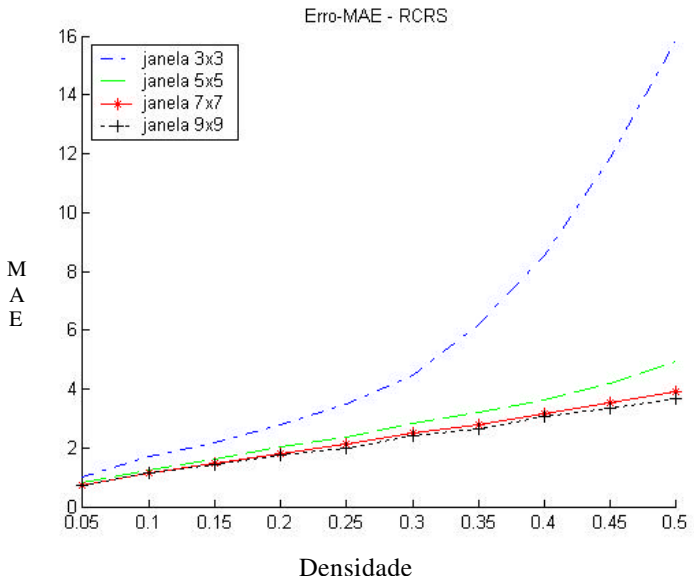


Figura 4.8 Vetores de erro, ruído sal e pimenta, janelas 3x3, 5x5, 7x7 e 9x9.

Para ruído sal e pimenta [0.05, 0.5], a imagem na entrada do filtro apresenta erro MAE de 6, 12, 19, 25, 32, 38, 44, 51, 57, 64 respectivamente.

Agora podemos analisar o que ocorre com o erro se aumentarmos a janela de observação. Para a Figura 4.8 temos os intervalos de erro, [1.0, 15.8], [0.8, 4.9], [0.711, 3.9], [0.715, 3.7] respectivamente, onde o erro diminui juntamente com o aumento do tamanho da janela de observação. Um dado importante e bem interessante é que essa diminuição do erro se dá principalmente quando passamos da janela 3x3 para 5x5, a partir daí a diminuição é bem menor. O filtro RCRS tem um melhor desempenho para janela de observação maior do que 3x3.

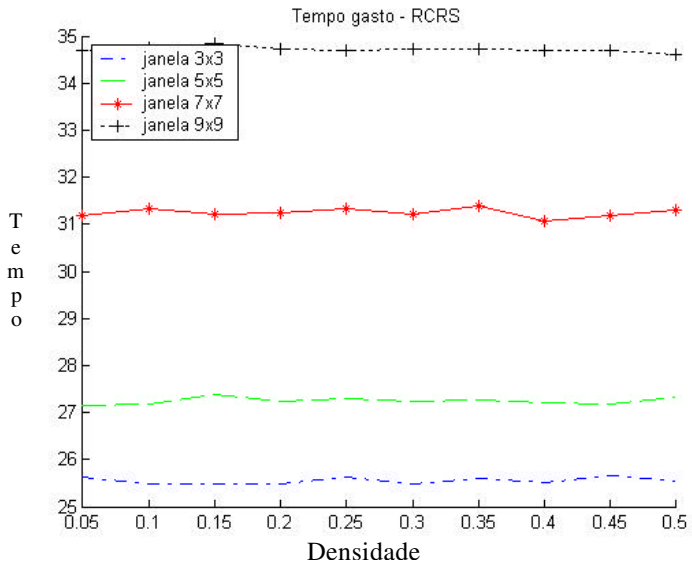


Figura 4.9 Vetores de tempo, ruído sal e pimenta, janelas 3x3, 5x5, 7x7 e 9x9.

A Figura 4.9 mostra-nos os vetores de tempo que estão nos intervalos de [25.5, 25.7], [27.1, 27.3], [31.06, 31.4] e [34.6, 34.8] segundos, respectivamente, onde podemos perceber que o custo computacional aumenta juntamente com o aumento do tamanho da janela de observação.

Agora utilizaremos filtro RCRS, imagem (lena), ruído gaussiano.

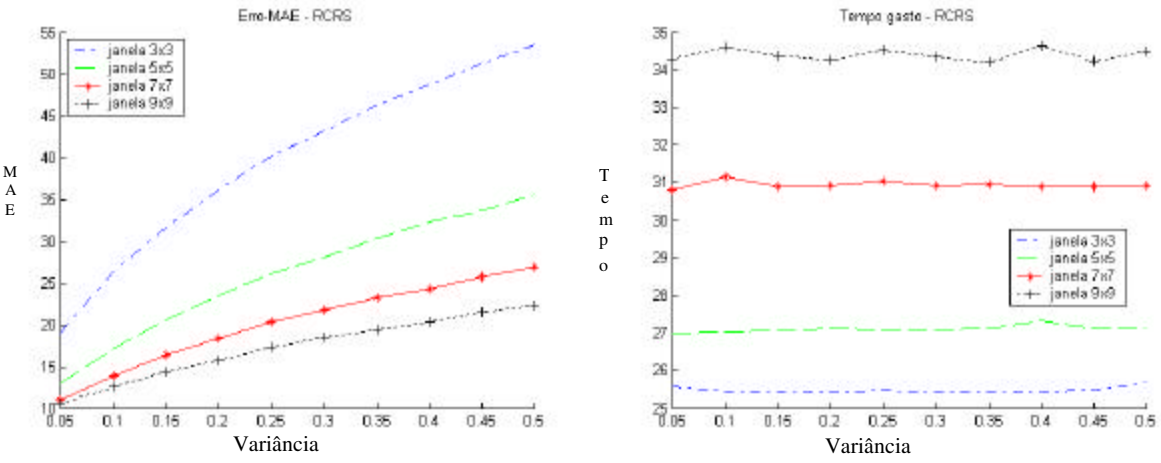


Figura 4.10 Vetores de erro e tempo, ruído gaussiano.

Para ruído gaussiano com média 0 e variâncias entre [0.05, 0.5], a imagem na entrada do filtro apresenta erro MAE de 33, 44, 52, 58, 62, 66, 70, 72, 75, respectivamente.

Os intervalos de erro são, [19.1, 53.4], [12.9, 35.6], [10.9, 26.9] e [10.5, 22.3] respectivamente, onde o erro diminui quando aumentamos o tamanho da janela de observação. A Figura 4.9 mostra-nos os vetores de tempo [25.4, 25.7], [27.0, 27.4], [30.8, 31.1] e [34.2, 34.6] segundos, respectivamente, onde podemos perceber que o custo computacional aumenta juntamente com o tamanho da janela de observação. Quando aumentamos o ruído gaussiano, o erro MAE aumenta e o tempo fica quase inalterado.

4.1.3 Filtro Wiener, com diferentes janelas.

Agora utilizaremos filtro Wiener, imagem (boat), ruído sal e pimenta.

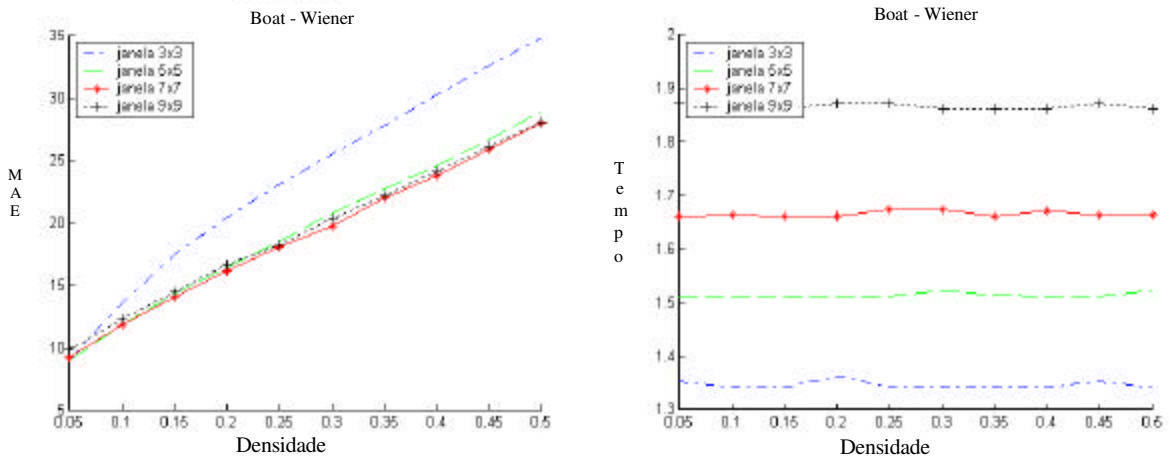


Figura 4.11 Vetores erro e tempo, Wiener, sal e pimenta.

Para ruído sal e pimenta [0.05, 0.5], a imagem na entrada do filtro apresenta erro MAE de 6, 12, 19, 25, 32, 38, 44, 51, 57, 64 respectivamente.

Para janela de observação 3x3 temos o intervalo de erro [8.9, 34.8], 5x5 o intervalo de erro é [8.8, 28.9], 7x7 o intervalo de erro é [9.2, 27.9], 9x9 o intervalo de erro é [9.8, 28.1]. Nesse caso também, à medida que o tamanho da janela de observação cresce, temos uma diminuição do intervalo de erro. O inverso ocorre com o tempo de

processamento que aumenta juntamente com o aumento do tamanho da janela de observação.

4.1.4 Filtros, considerando diferentes janelas e ruído sal e pimenta.

Tendo em vista os resultados obtidos anteriormente pelos filtros RCRS (seção 4.1.2) e Wiener (seção 4.1.2) vamos agora fazer o mesmo teste com uma mesma imagem, com todos os filtros e observar o que vai ocorrer.

Filtraremos a imagem spine, ruído sal e pimenta e janelas 3x3, 5x5, 7x7 e 9x9. As imagens na entrada do filtro apresentam erros MAE de: 6, 12, 19, 25, 32, 38, 44, 51, 57, 64 respectivamente. Como foi inserido nas imagens as mesmas densidades de ruído, o erro MAE inicial é o mesmo.

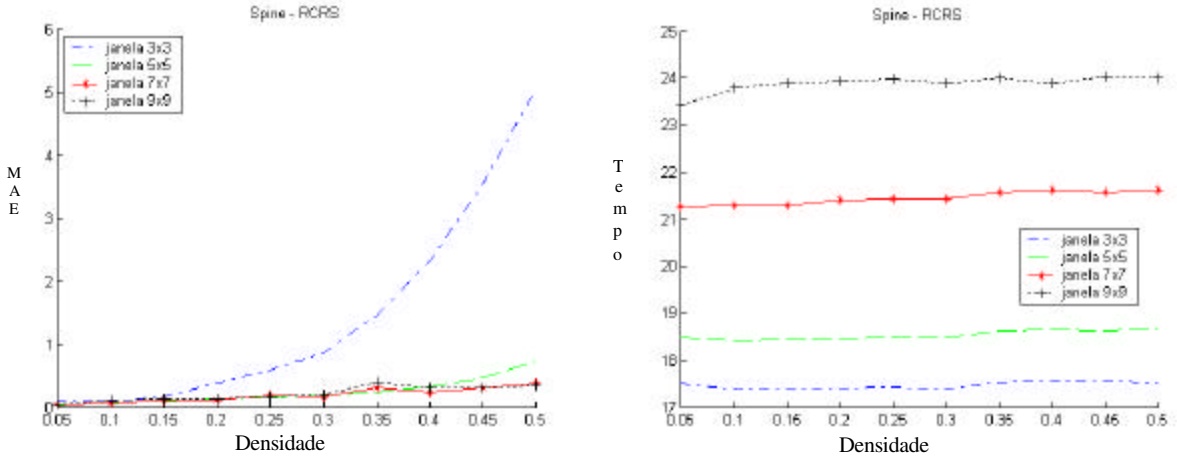


Figura 4.12 Vetores erro e tempo, sal e pimenta e RCRS.

O filtro utilizado para recuperar a imagem foi RCRS, os intervalos de erro são [0.12, 5], [0.18, 0.8], [0.2, 0.42], [0.22 , 0.4], respectivamente. Notamos a partir desses valores que o intervalo de erro MAE ficou menor quando aumentamos o tamanho da janela de observação. Os intervalos de tempo são [17.4, 17.7], [18.3, 18.6], [20.6, 21.3], [22.7, 23.5] segundos, temos nesse caso um aumento pequeno no custo computacional.

Com relação ao intervalo de erro inicial temos uma considerável melhora nesse caso.

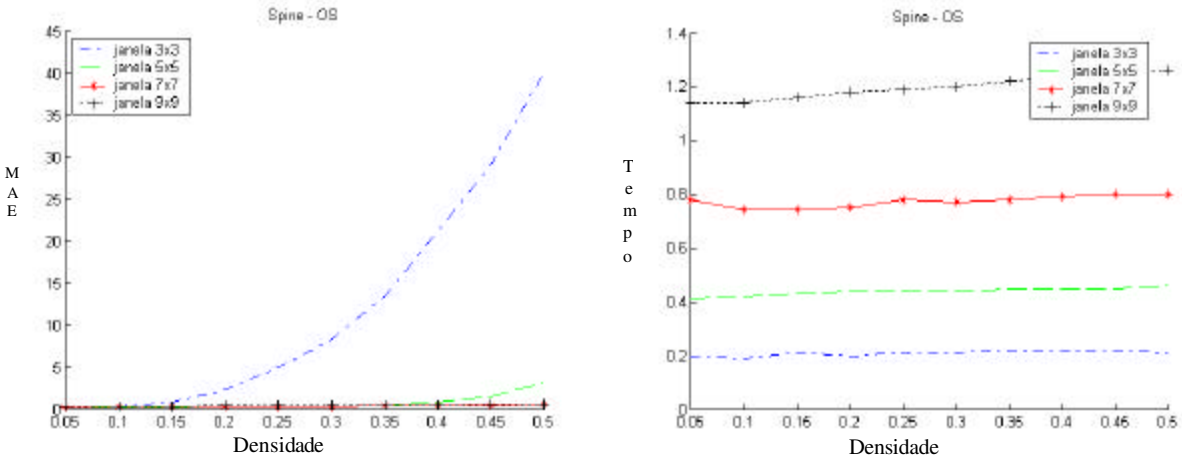


Figura 4.13 Vetores erro e tempo, sal e pimenta e OS.

O filtro utilizado para recuperar a imagem foi OS, os intervalos de erro são [0.7, 40], [0.6, 3.1], [0.7, 2.2], [0.9, 2.4], respectivamente. Notamos a partir desses intervalos que da janela tamanho 3x3 para 5x5 houve uma considerável diminuição do erro e as janelas 7x7 e 9x9 tem valores bem próximos. O filtro OS tem uma melhor resposta quando utilizamos janelas de observação maiores que 3x3. Os intervalos de tempo são [0.21, 1.12], [0.40, 0.44], [0.68, 0.76], [1.04, 1.19] segundos. Quando aumentamos a janela de observação o tempo de processamento aumenta.

Com relação ao intervalo de erro inicial temos uma considerável melhora nesse caso.

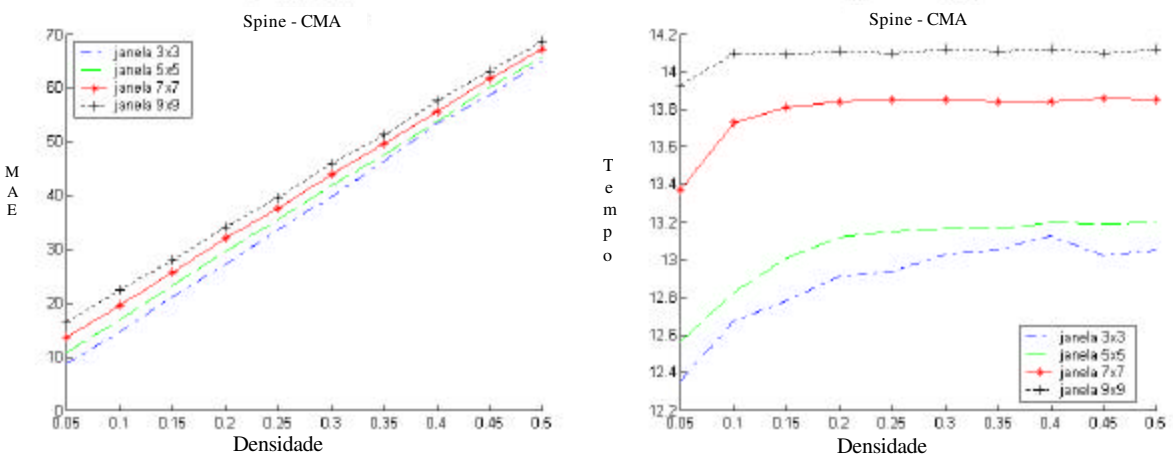


Figura 4.14 Vetores erro e tempo, sal e pimenta e CMA.

O filtro utilizado para recuperar a imagem foi CMA, os intervalos de erro são [8.7, 64.9], [10.7, 65.8], [13.6, 67.3], [16.4, 68.7], respectivamente. Notamos a partir desses intervalos que o erro aumenta quando aumentamos o tamanho da janela de observação. O motivo para isto é o filtro trabalhar com a média na vizinhança. Os intervalos de tempo são [12.4, 13.1], [12.6, 13.2], [13.4, 13.9], [13.9, 14.1] segundos, temos também nesse caso um aumento do custo computacional quando aumentamos o tamanho da janela de observação.

O intervalo de erro MAE (imagem de entrada) é menor do que o apresentado na saída do filtro nesse caso.

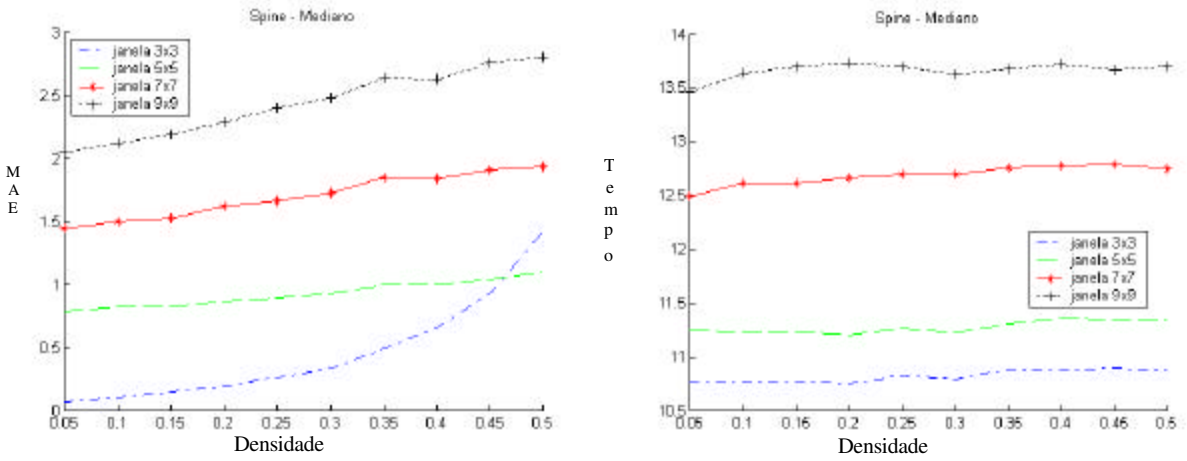


Figura 4.15 Vetores erro e tempo, sal e pimenta e Mediano.

O filtro utilizado para recuperar a imagem foi Mediano, os intervalos de erro são [0.2, 1.4], [0.82, 1.2], [1.46, 1.96], [2.1, 2.8], respectivamente. Notamos a partir desses intervalos que o erro aumenta quando aumentamos o tamanho da janela de observação. Os intervalos de tempo são [10.77, 10.83], [11.1, 11.2], [12.2, 12.5], [13.1, 13.3] segundos, temos também nesse caso um aumento do custo computacional quando aumentamos o tamanho da janela de observação.

O intervalo de erro MAE (imagem de entrada) é menor do que o apresentado na saída do filtro para esse caso.

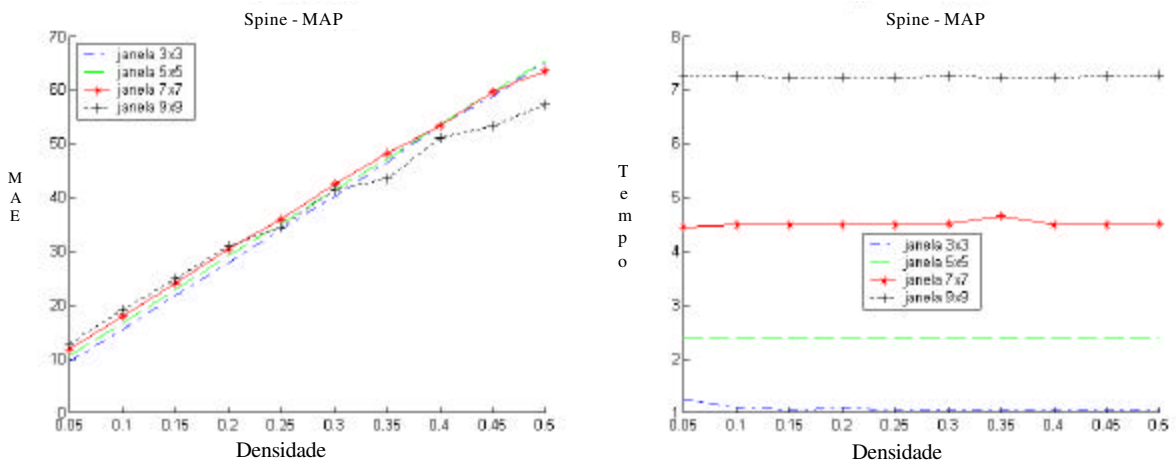


Figura 4.16 Vetores erro e tempo, sal e pimenta e MAP.

O filtro utilizado para recuperar a imagem foi MAP, os intervalos de erro são [9.4, 65.1], [10.48, 65.5], [11.7, 63.6], [12.9, 57.1], respectivamente. Notamos que não houve muita diferença quando aumentamos o tamanho da janela de observação. Os intervalos de tempo são [1.0, 1.2], [2.37, 2.38], [4.5, 4.7], [7.2, 7.3] segundos, temos também nesse caso um aumento do custo computacional quando aumentamos o tamanho da janela de observação.

O intervalo de erro MAE (imagem de entrada) é menor do que o apresentado na saída do filtro para esse caso.

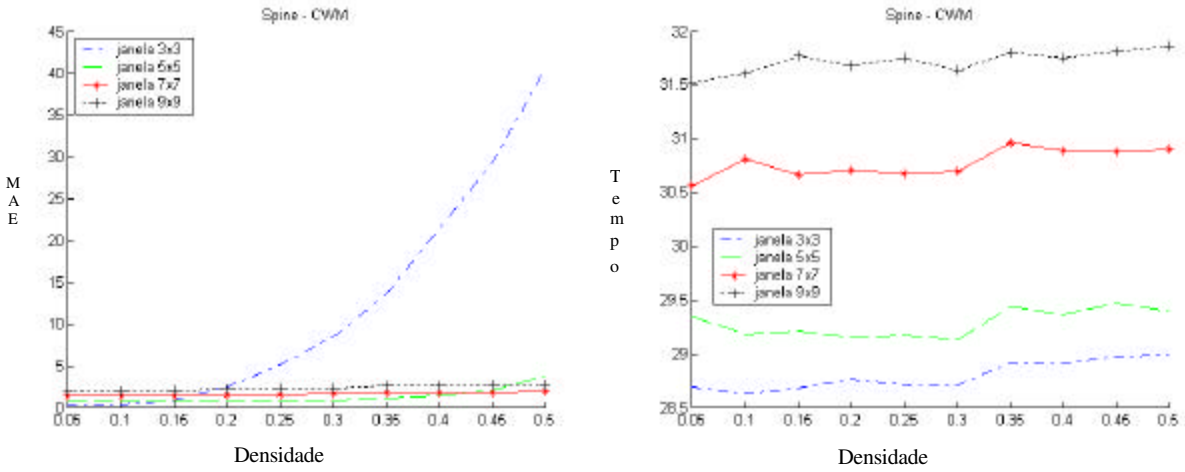


Figura 4.17 Vetores erro e tempo, sal e pimenta e CWM.

O filtro utilizado para recuperar a imagem foi CWM, os intervalos de erro são [0.2, 40], [0.7, 3.7], [1.3, 1.9], [1.8, 2.6], respectivamente. Notamos a partir desses intervalos que o erro caem de janela 3x3 para 5x5 e janela 5x5, para janela 7x7, depois começa a subir de janela 7x7 para janela 9x9. Este filtro tem uma melhor resposta para janelas de observação maiores que 3x3. Os intervalos de tempo são [28.5, 28.7], [28.96, 29.15], [30.3, 30.6], [31.2, 31.4] segundos, também nesse caso o aumento do tamanho da janela de observação leva a um aumento do custo computacional.

Com relação ao intervalo de erro inicial temos uma considerável melhora nesse caso.

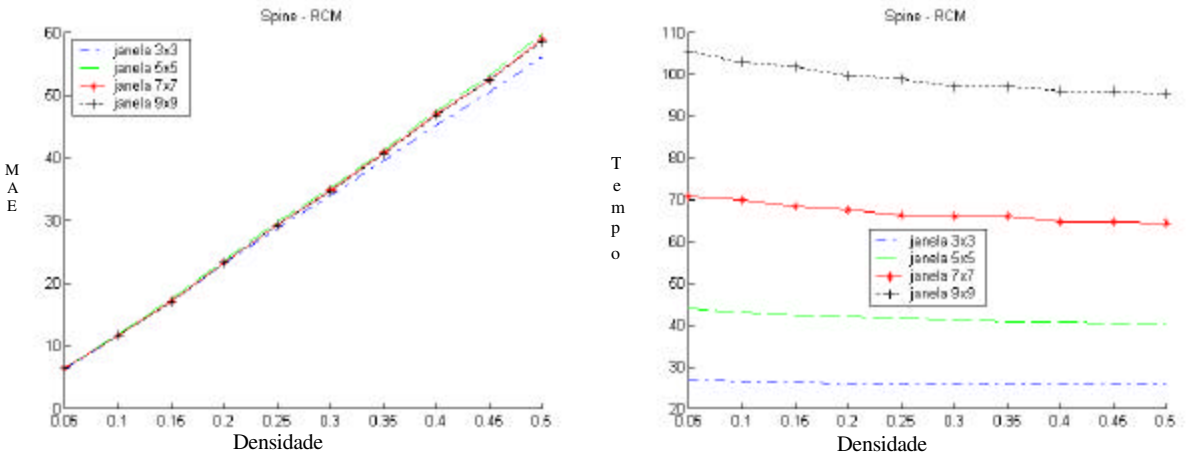


Figura 4.18 Vetores erro e tempo, sal e pimenta e RCM.

O filtro utilizado para recuperar a imagem foi RCM, os intervalos de erro são [8.7, 64.8], [8.5, 63.6], [9.9, 64.9], [11.2, 65.6], respectivamente. Notamos que o erro apresentado pela imagem na saídas dos filtros não se alterou muito com o aumento do tamanho da janela de observação. Os intervalos de tempo são [25.9, 27.9], [42.8, 47.8], [70.5, 79.5], [106.5, 121.7] segundos, também nesse caso o aumento da janela significa o aumento do custo computacional.

O intervalo de erro MAE (imagem de entrada) é menor do que o apresentado na saída do filtro para esse caso.

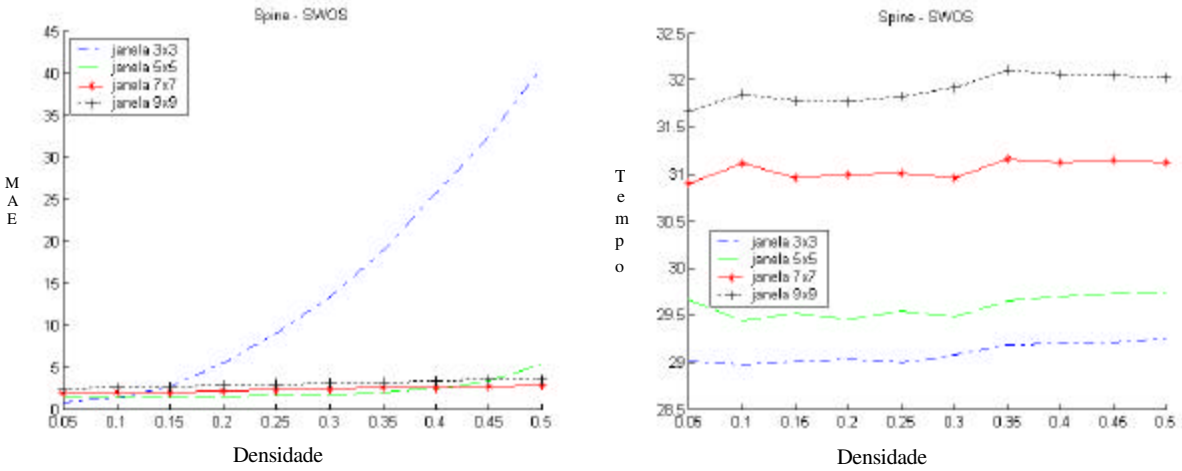


Figura 4.19 Vetores erro e tempo, sal e pimenta e SWOS.

O filtro utilizado para recuperar a imagem foi SWOS, os intervalos de erro são [0.8, 40], [1.3, 5.3], [1.9, 2.8], [2.5, 3.5], respectivamente. Notamos que para janela 3x3 o filtro apresenta o maior intervalo de erro. Quando aumentamos o tamanho da janela de observação o intervalo de erro diminui. Os intervalos de tempo são [29.0, 29.2], [29.5, 29.6], [31.0, 31.2], [31.6, 31.9] segundos, também nesse caso o aumento da janela significa o aumento do custo computacional.

Com relação ao intervalo de erro inicial temos uma considerável melhora nesse caso.

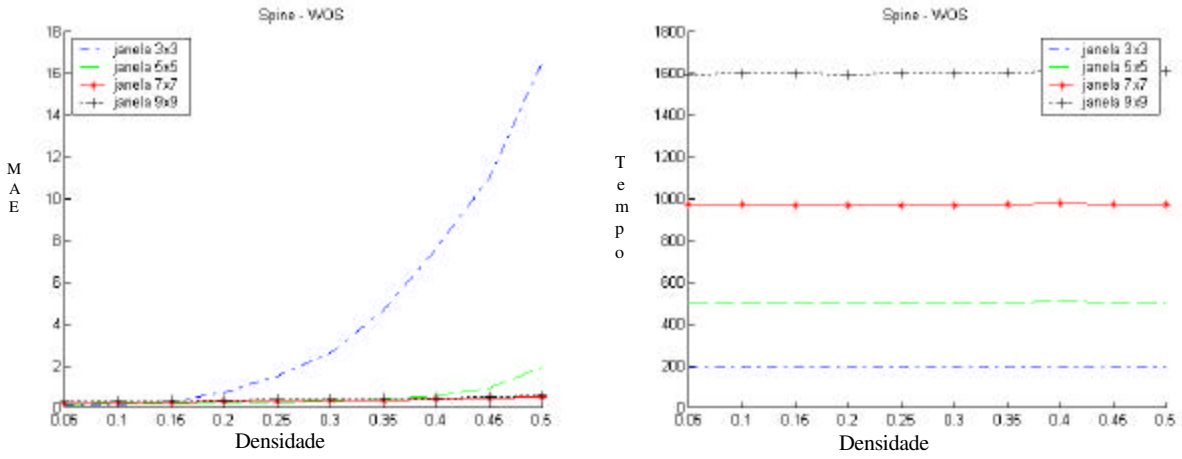


Figura 4.20 Vetores erro e tempo, sal e pimenta e WOS.

O filtro utilizado para recuperar a imagem foi WOS, os intervalos de erro são [0.2, 17.1], [0.4, 2.9], [0.6, 1.9], [0.8, 2.1], respectivamente. Notamos a partir desses intervalos que o erro inicial vai aumentar e o final vai diminuir, quando aumentamos o tamanho da janela, exceto para janela 9x9 onde o erro final aumenta. Os intervalos de tempo são [191405, 191886], [496424, 498056], [969183, 972138], [1600352, 1604558] segundos, aqui podemos perceber o elevado custo computacional que aumenta juntamente com o aumento do tamanho da janela. O filtro WOS tem a janela de observação com valores que podem ser diferentes de um e que significam repetição, isto faz com que o vetor de trabalhado aumente seu tamanho, requer mais memória e fica mais lento.

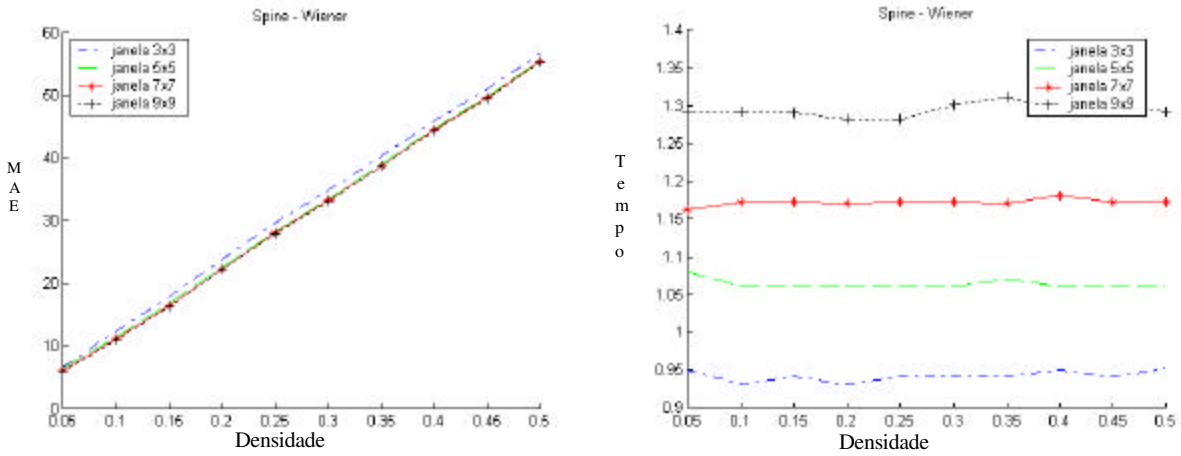


Figura 4.21 Vetores erro e tempo, sal e pimenta e Wiener.

O filtro utilizado para recuperar a imagem foi Wiener, os intervalos de erro são [7.56, 55.8], [7.0, 55.2], [7.3, 55.4], [7.5, 55.3], respectivamente. Quando aumentamos o tamanho da janela de observação quase não se nota diferença no intervalo de erro, que para janela de observação 3x3 é um pouco maior. Os intervalos de tempo são [0.89, 1.4], [0.99, 1.02], [1.11, 1.13], [1.23, 1.24], temos também nesse caso um aumento do custo computacional (tempo de processamento), quando aumentamos o tamanho da janela de observação.

O intervalo de erro MAE (imagem de entrada) é menor do que o apresentado na saída do filtro para esse caso.

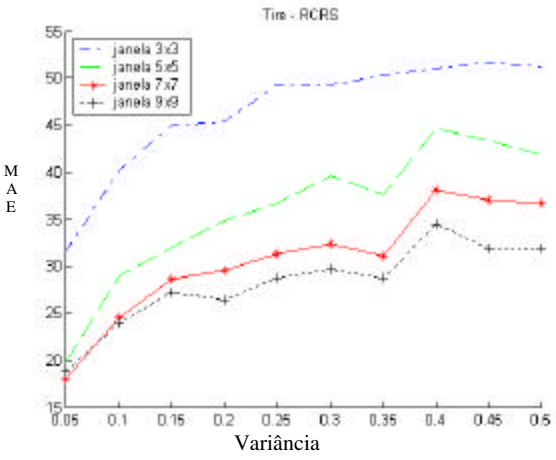
Os filtros RCRS, OS, CWM, SWOS e WOS, quando aumentamos o tamanho da janela de observação de 3x3 para 5x5 o erro diminui. Ao aumentarmos a janela de observação 5x5 para 7x7 e de 7x7 para 9x9, temos um erro bem próximo. Já o tempo de processamento aumenta juntamente com o aumento do tamanho da janela de observação. Os filtros derivados do filtro de ordem estatística não tiveram uma boa resposta quando trabalhamos com janela de observação 3x3.

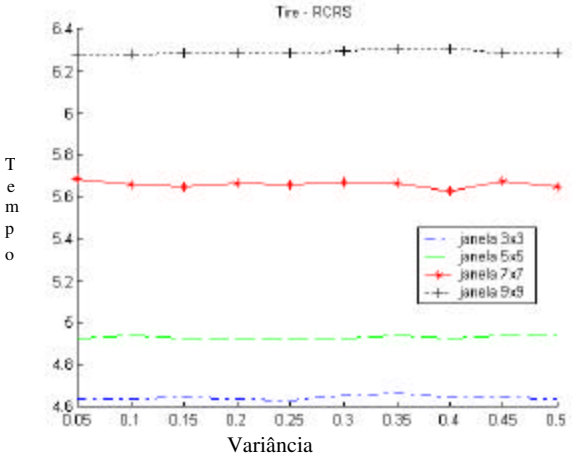
Os filtros CMA e Mediano, quando aumentamos o tamanho da janela de observação tanto o erro quanto o tempo de processamento aumentam.

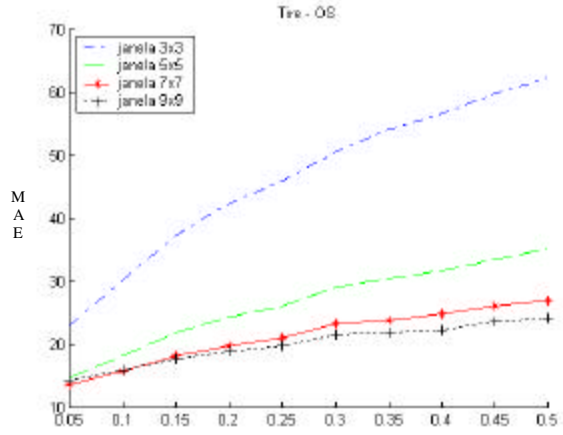
Os filtros MAP, RCM e Wiener, quando aumentamos o tamanho da janela de observação os erros na saída se confundem, já o tempo de processamento aumenta juntamente com o aumento do tamanho da janela de observação.

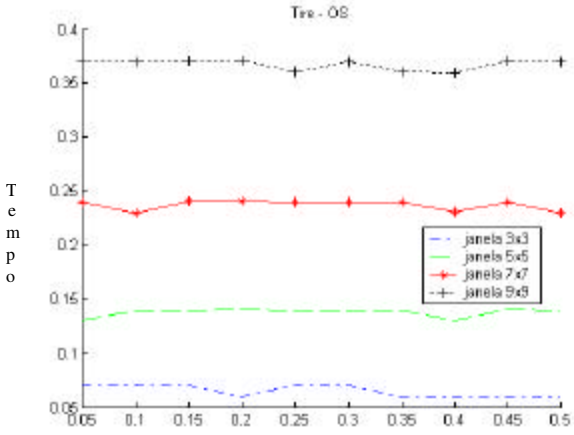
4.1.5 Filtros considerando, diferentes janelas e ruído gaussiano.

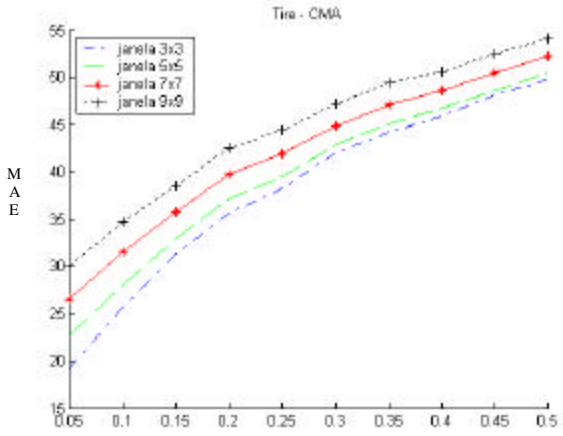
A seguir filtraremos a imagem tire, ruído gaussiano e janelas 3x3, 5x5, 7x7 e 9x9.

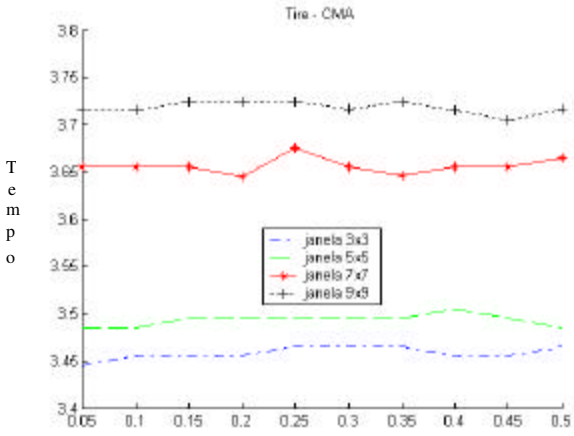


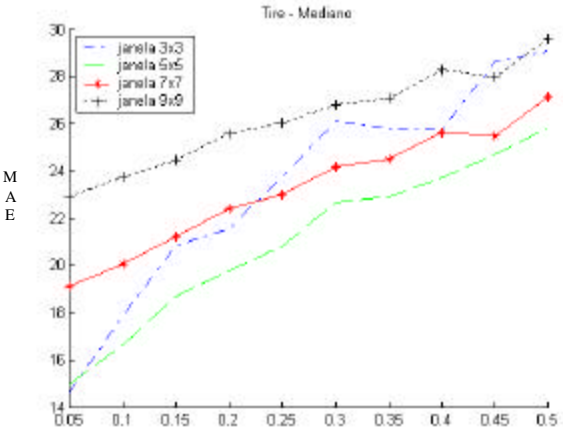


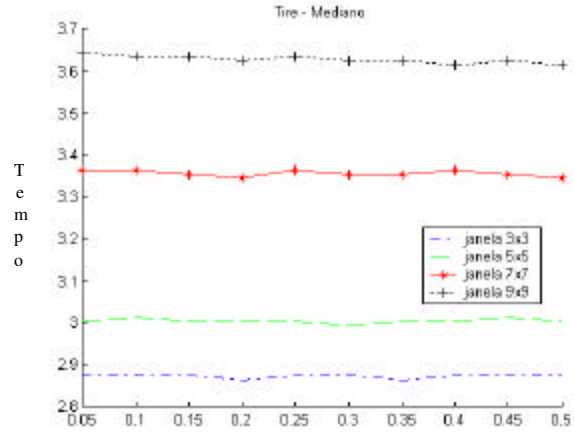

Tire - OS
janela 3x3
janela 5x5
janela 7x7
janela 9x9
MAE
Variância

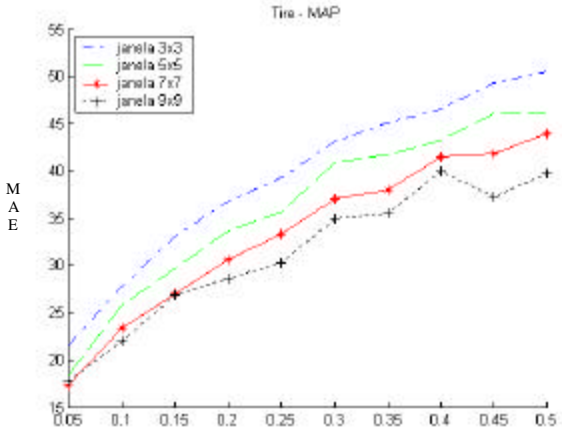

Tire - OS
janela 3x3
janela 5x5
janela 7x7
janela 9x9
Tempo
Variância

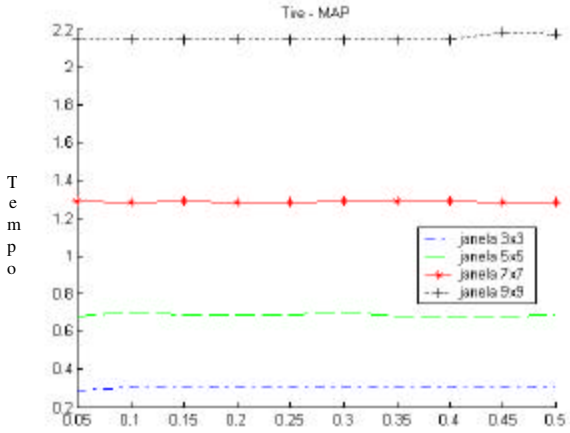

Tire - CMA
janela 3x3
janela 5x5
janela 7x7
janela 9x9
MAE
Variância

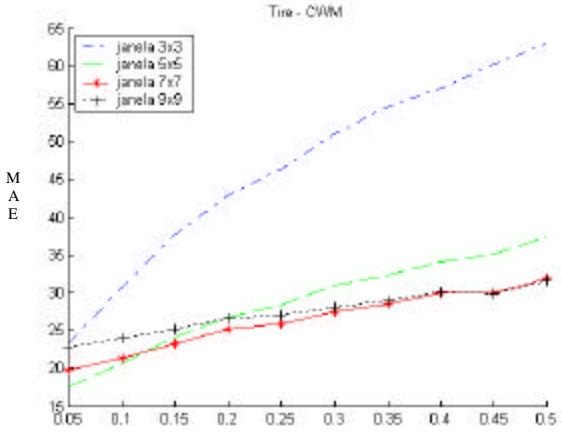

Tire - CMA
janela 3x3
janela 5x5
janela 7x7
janela 9x9
Tempo
Variância

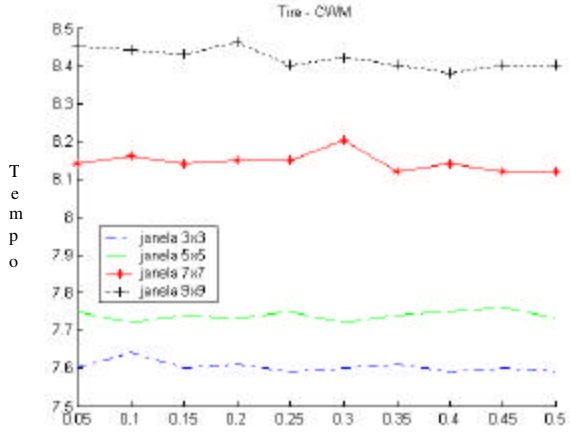

Tire - Mediano
janela 3x3
janela 5x5
janela 7x7
janela 9x9
MAE
Variância

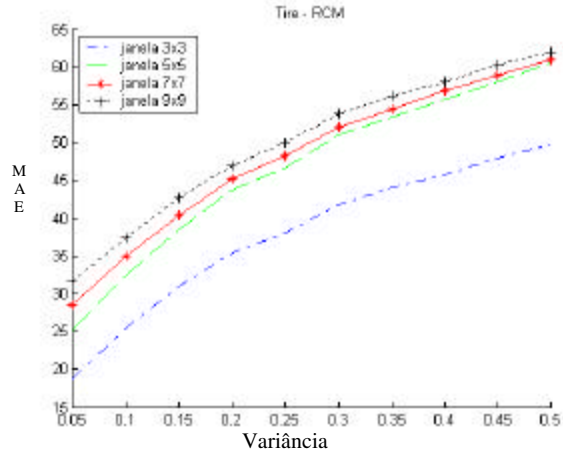

Tire - Mediano
janela 3x3
janela 5x5
janela 7x7
janela 9x9
Tempo
Variância

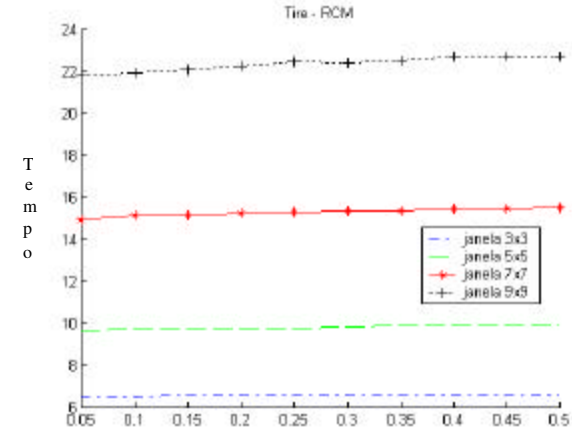

Tire - MAP
janela 3x3
janela 5x5
janela 7x7
janela 9x9
MAE
Variância


Tire - MAP
janela 3x3
janela 5x5
janela 7x7
janela 9x9
Tempo
Variância


Tire - CWM
janela 3x3
janela 5x5
janela 7x7
janela 9x9
MAE
Variância


Tire - CWM
janela 3x3
janela 5x5
janela 7x7
janela 9x9
Tempo
Variância


Tire - RCM
janela 3x3
janela 5x5
janela 7x7
janela 9x9
MAE
Variância


Tire - RCM
janela 3x3
janela 5x5
janela 7x7
janela 9x9
Tempo
Variância

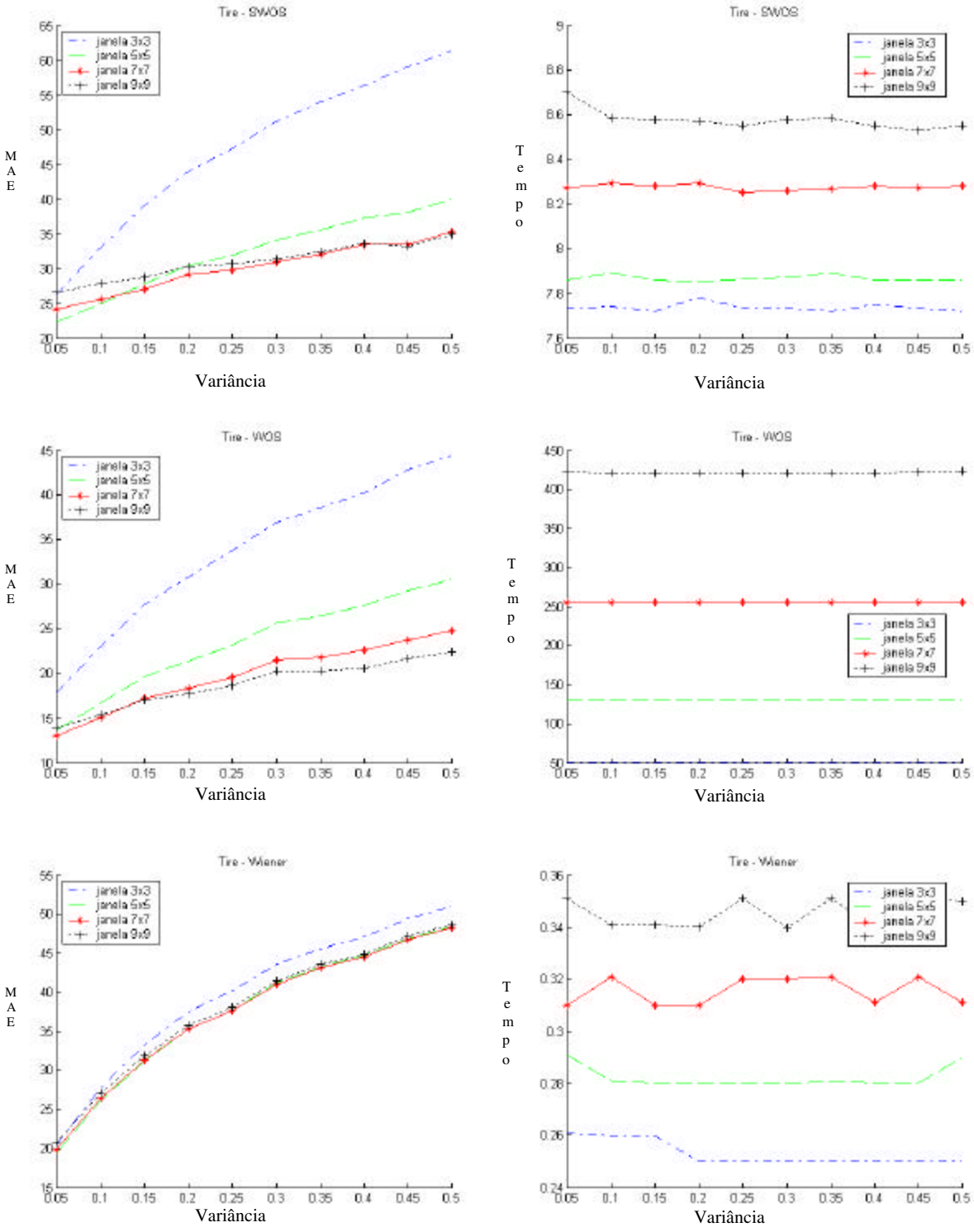


Figura 4.22 Vetores erro e tempo, gaussiano.

Para ruído gaussiano com média 0 e variâncias entre [0.05, 0.5], a imagem na entrada do filtro apresenta erro MAE de 33, 44, 52, 58, 62, 66, 70, 72, 75, respectivamente.

Acima temos os vetores de erro e tempo referentes, a cada filtro quando inserimos um ruído do tipo gaussiano. As Tabelas 4.1 e 4.2 contém os respectivos intervalos de erro e tempo.

Filtro/Erro	3x3	5x5	7x7	9x9
RCRS	[32.5, 51.8]	[20.6, 46.5]	[17.7, 36.7]	[19.6, 31.7]
OS	[22.8, 62.6]	[14.5, 36.7]	[13.2, 27.4]	[13.9, 24.3]
CMA	[18.9, 49.9]	[22.5, 50.4]	[26.3, 51.6]	[29.87, 54.1]
Mediano	[14.3, 28.9]	[14.8, 25.2]	[18.9, 26.7]	[22.8, 28.5]
MAP	[21.2, 50.7]	[18.2, 46.2]	[16.8, 44.5]	[17.1, 39.6]
CWM	[22.6, 63.9]	[17.2, 36.4]	[19.4, 30.8]	[22.6, 30.4]
RCM	[18.8, 49.7]	[25.2, 60.5]	[28.6, 60.9]	[31.6, 62.1]
SWOS	[25.8, 61.9]	[21.9, 39.6]	[23.8, 34.6]	[26.5, 33.8]
WOS	[17.8, 44.5]	[13.6, 30.5]	[12.9, 24.7]	[13.9, 22.6]
Wiener	[19.9, 51.5]	[18.9, 48.9]	[19.5, 48.3]	[20.3, 48.9]

Tabela 4.1 Intervalos de erro, ruído gaussiano.

Pelos intervalos de erro e os gráficos (Figura 4.22), dos filtros RCRS, OS, MAP, CWM, WOS e SWOS, pode-se perceber que os mesmos ficaram menores, quando aumentamos o tamanho da janela de observação. Já os filtros CMA, Mediano e RCM, o intervalo de erro MAE aumentou, quando aumentamos o tamanho da janela de observação. O filtro de Wiener apresenta intervalos de erros que se confundem, exceto para janela de observação 3x3 que é maior.

Filtro/Tempo	3x3	5x5	7x7	9x9
RCRS	[4.8, 6.7]	[5.1, 6.4]	[5.8, 5.9]	[6.4, 6.5]
OS	[0.08, 0.1]	[0.18, 0.2]	[0.32, 0.33]	[0.5, 0.52]
CMA	[3.5, 3.53]	[3.56, 3.59]	[3.73, 3.79]	[3.77, 3.95]
Mediano	[2.9, 3.2]	[3.1, 3.4]	[3.4, 3.5]	[3.6, 3.7]
MAP	[0.3, 0.6]	[0.6, 0.7]	[1.0, 1.1]	[1.7, 1.8]
CWM	[7.9, 8.6]	[8.0, 8.3]	[8.18, 8.22]	[8.4, 8.5]
RCM	[6.68, 6.97]	[10.5, 11.2]	[15.9, 16.3]	[23.5, 23.7]
SWOS	[7.9, 9.3]	[8.1, 8.6]	[8.3, 8.5]	[8.5,8.7]
WOS	[49.9, 50.9]	[130.7, 135.4]	[254.8, 264.2]	[420.9, 421.8]
Wiener	[0.2, 1.01]	[0.27, 0.37]	[0.29, 0.31]	[0.32, 0.34]

Tabela 4.2 Intervalos de tempo, ruído gaussiano.

Quando observamos a tabela e os gráficos de tempo (Figura 4.22) dos filtros temos que todos aumentam o tempo de processamento quando aumentamos o tamanho da janela de observação. Filtros, considerando imagens diferentes.

Passaremos agora a outro teste, onde utilizaremos o mesmo filtro na restauração de duas imagens diferentes, corrompidas pelo mesmo tipo e densidade de ruído.

4.1.6 Imagens boat e lena.

As imagens são boat (205x232) e lena (512x512), corrompidas por ruído do tipo sal e pimenta, densidade variando de [0.05, 0.5], com janela de observação de tamanho 5x5, os filtros são filtros Mediano e RCRS.

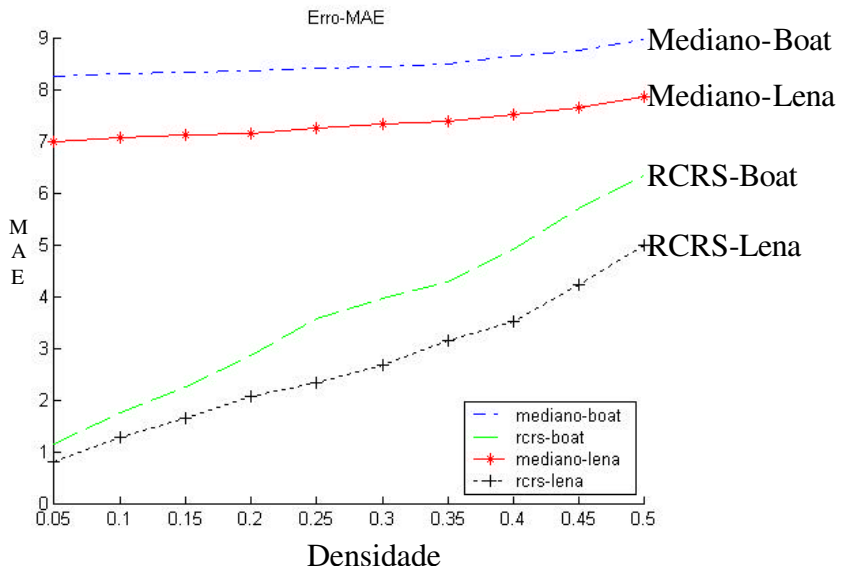


Figura 4.23 Vetores de erro, lena x boat, máscara 5x5.

Para ruído sal e pimenta densidade entre [0.05, 0.5], a imagem na entrada do filtro apresenta erro MAE de 6, 12, 19, 25, 32, 38, 44, 51, 57, 64 respectivamente, para as duas imagens. Isto ocorre pois inserimos a mesma densidade de ruído nas duas imagens.

Para a imagem lena temos os intervalos de erro [8.2, 8.9] (mediano) e [1.1, 6.4] (RCRS). Para a imagem boat temos os intervalos de erro [6.9, 7.9] (mediano) e [0.8, 4.9] (RCRS). As curvas de erro apresentadas pelos dois filtros são bem diferentes, porém podemos observar que o comportamento dos dois algoritmos com relação às duas imagens é bem similar, a maior diferença é que a imagem lena vai apresentar um erro maior e o gráfico vai estar deslocado para cima. A imagem boat tem mais detalhes (ver Anexo 4, p.165-172) o que dificulta a recuperação.

Vamos observar agora o que vai ocorrer com os vetores de tempo correspondentes.

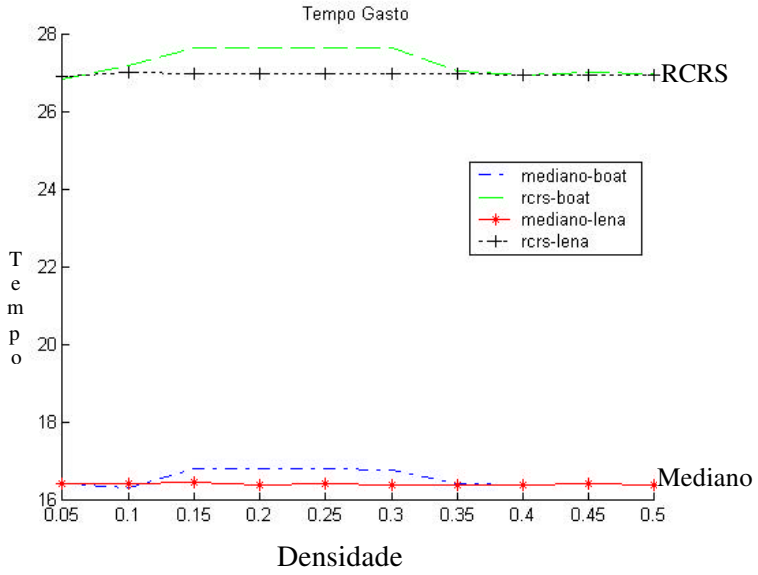


Figura 4.24 Vetores de tempo, lena x boat, máscara 5x5.

As curvas de tempo apresentadas pelos dois filtros com relação as duas imagens é bem similar, o filtro RCRS apresenta um tempo maior para as duas imagens. O tamanho de ambas as imagens é 512x512, o que explica o gráfico de tempo ser bem similar.

A seguir temos as imagens boat e lena, corrompidas por ruído do tipo gaussiano com média 0 e variâncias entre [0.05, 0.5], janela de observação de tamanho 7x7 e os filtros de Mediano e RCRS.

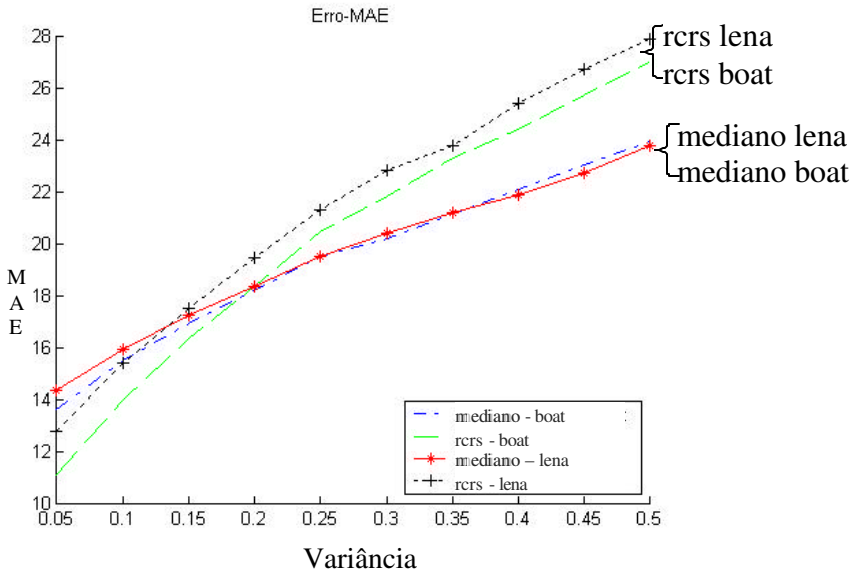


Figura 4.25 Vetores erro, lena x boat, RCRS e Mediano, 7x7.

Para ruído gaussiano com média 0 e variâncias entre [0.05, 0.5], a imagem na entrada do filtro apresenta erro MAE de 33, 44, 52, 58, 62, 66, 70, 72, 75, respectivamente, para as duas imagens. Isto ocorre, pois, a variância do ruído é a mesma para as duas imagens.

Para a imagem lena temos os intervalos de erro [11.3, 11.9] (mediano) e [1.1, 5.4] (RCRS). Para a imagem boat temos os intervalos de erro [10.1, 11.1] (mediano) e [0.7, 3.9] (RCRS). Nesse, o comportamento dos dois algoritmos com relação às duas imagens também é similar ao apresentado na Figura 4.23, com a imagem lena gerando um erro maior. As curvas de erro são bem parecidas, principalmente no que se refere ao filtro mediano. Observamos também que para pouco ruído o filtro (inicio do gráfico) RCRS nos retornou um resultado melhor, porém quando aumentamos o ruído o filtro mediano apresenta um melhor desempenho (os gráficos se cruzam mais ou menos em variância igual a 0.2).

Observando os resultados apresentados na Figura 4.23 e Figura 4.25, temos que para ruído do tipo sal e pimenta o filtro RCRS apresenta um erro menor para as duas imagens. Para ruído gaussiano, inicialmente ocorre o mesmo (variância até mais ou menos 0.2), depois o filtro mediano apresenta um erro menor, para as duas imagens.

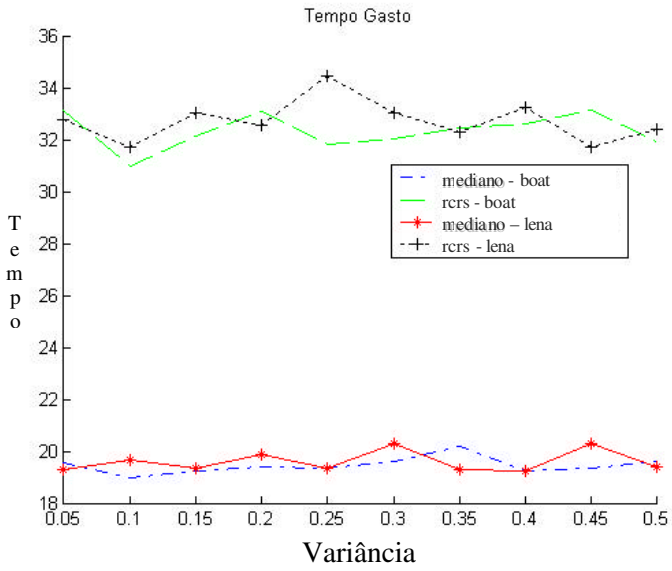


Figura 4.26 Vetores tempo, lena x boat, RCRS e Mediano, 7x7.

O filtro RCRS requer um maior tempo de processamento e o tempo é similar para as duas imagens. O tamanho de ambas as imagens é 512x512, o que explica o gráfico de tempo ser bem similar.

No caso a seguir temos as imagens boat e lena, corrompidas por ruído do tipo sal e pimenta, densidade variando de [0.05, 0.5], com janela de observação de tamanho 7x7 e os filtros de Média aritmética ponderada, Wiener e Ordem estatística.

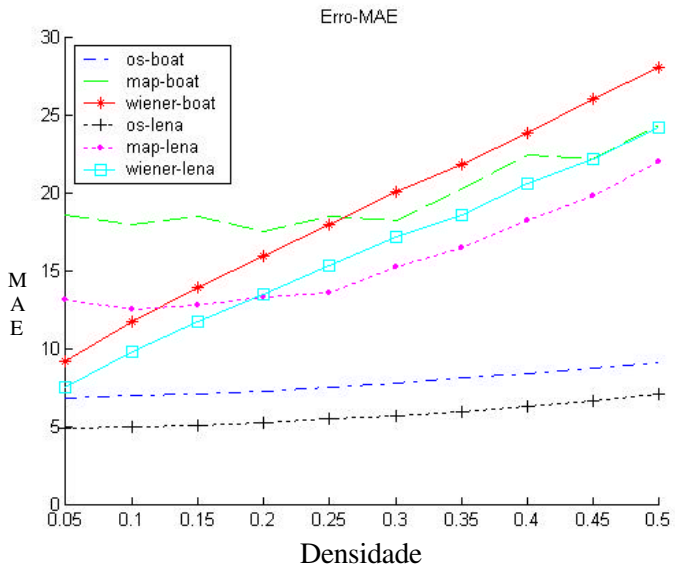


Figura 4.27 Vetores erro, lena x boat, MAP, Wiener e OS.

Para a imagem lena temos os intervalos de erro [4.8, 7.1] (OS), [13.1, 21.9] (MAP) e [7.5, 24.1] (Wiener). Para a imagem boat temos os intervalos de erro [6.8, 9.1] (OS), [18.5, 24.3] (MAP) e [9.2, 27.9] (Wiener). Também nesse caso o comportamento dos três algoritmos com relação às duas imagens é bem similar ao comportamento observado na Figura 4.23. As curvas de erro são bem parecidas. O filtro OS tem uma nos retornou um resultado melhor, para as duas imagens. Para pouco ruído o filtro MAP tem um resultado melhor do que o de Wiener e quando aumentamos o ruído as posições se invertem.

Vamos observar agora o que vai ocorrer com os vetores de tempos correspondentes.

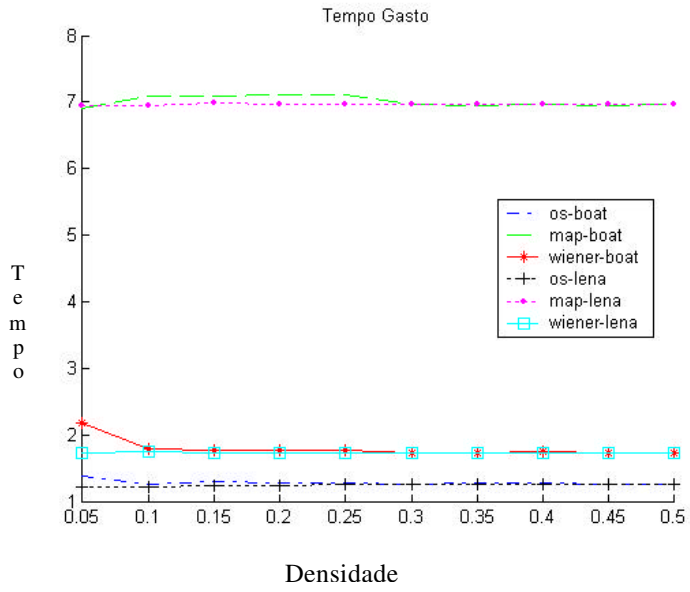


Figura 4.28 Vetores erro, lena x boat, MAP, Wiener e OS.

Também aqui podemos observar que o comportamento dos filtros com relação às duas imagens é bem similar. As curvas de tempo são bem diferentes e pode-se observar que o comportamento dos três algoritmos com relação às duas imagens é bem similar, sendo que a maior diferença é que o filtro MAP apresenta um tempo maior se comparado com os outros dois. O tamanho de ambas as imagens é 512x512, o que explica o gráfico de tempo se confundir.

Faremos testes similares aos acima, só que dessa vez utilizaremos várias imagens (ver Anexo 4, p.165-172).

4.1.7 Imagens montage e mountain.

Para o teste abaixo temos ruído sal e pimenta, imagens montage e mountain, janela de observação 7x7.

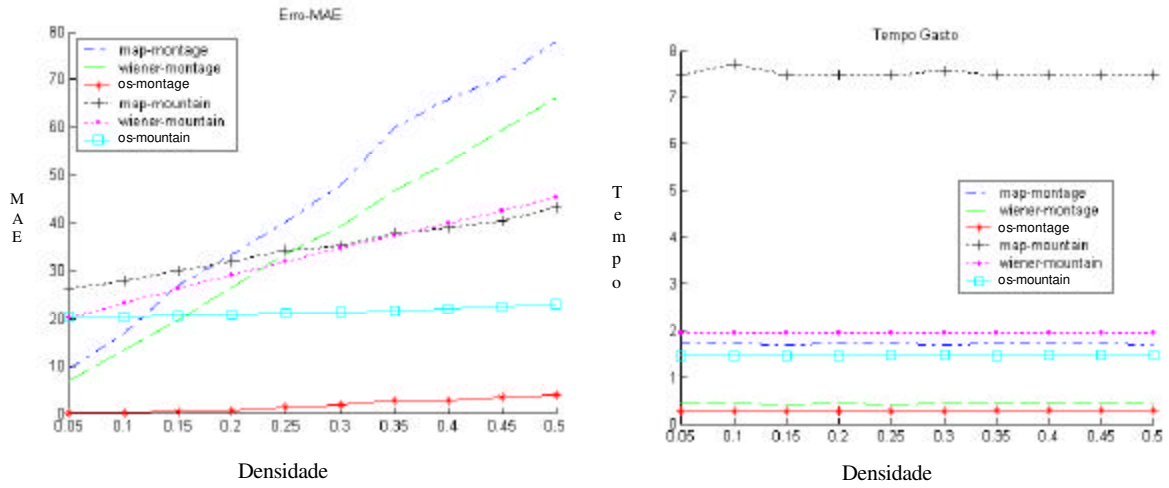


Figura 4.29 Vetores erro e tempo montage e mountain, sal e pimenta.

Nesse caso as imagens apresentam tamanhos diferentes, 256x256 (montage) e 480x640 (montain), os vetores de tempo e erro ficam deslocados para cima para a imagem maior.

Para a imagem montage temos os intervalos de erro [0.1, 2.2] (OS), [10.1, 78.9] (MAP) e [7.5, 65.4] (Wiener). Para a imagem mountain temos os intervalos de erro [20.1, 22.1] (OS), [25.1, 42.5] (MAP) e [19.9, 45.5] (Wiener). Também nesse caso, o comportamento dos três algoritmos com relação às duas imagens é bem similar ao comportamento observado na Figura 4.23. As curvas de erro são bem parecidas. O filtro OS tem uma nos retornou um resultado melhor, para as duas imagens.

Para o teste abaixo temos ruído gaussiano, imagens montage e mountain, janela de observação 7x7.

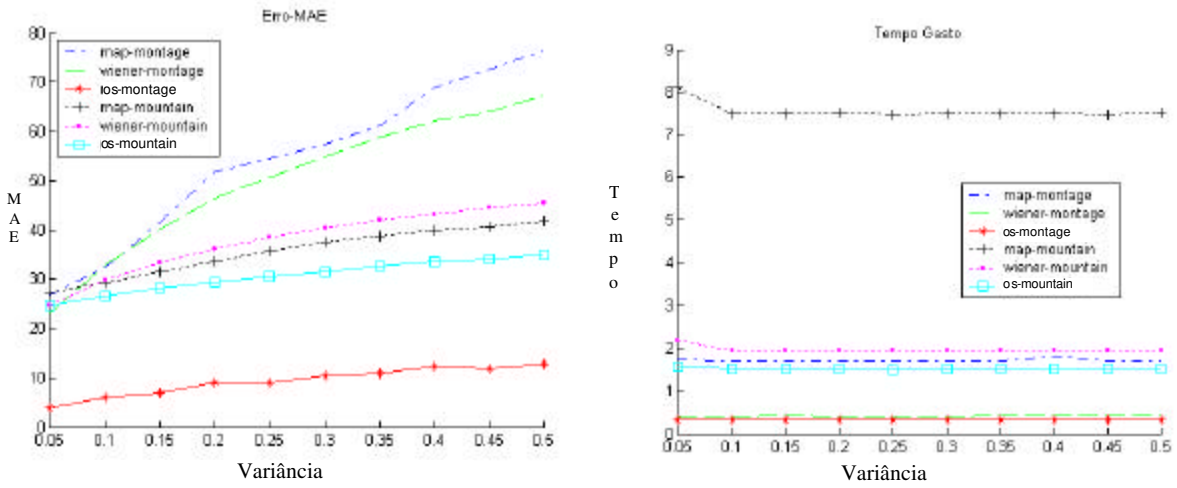


Figura 4.30 Vetores erro e tempo montage e mountain, gaussiano.

Para a imagem montage temos os intervalos de erro [2.2, 12.2] (OS), [24.6, 76.8] (MAP) e [22.5, 68.9] (Wiener). Para a imagem mountain temos os intervalos de erro [23.1, 35.2] (OS), [24.5, 41.2] (MAP) e [23.2, 46.3] (Wiener). Também nesse caso, o comportamento dos três algoritmos com relação às duas imagens é bem similar ao comportamento observado na Figura 4.23. As curvas de erro são bem parecidas. O filtro OS tem uma nos retornou um resultado melhor, para as duas imagens.

4.1.8 Imagens olhodigital e flor.

Para o teste abaixo temos ruído gaussiano, imagens olhodigital e flor, janela de observação 5x5.

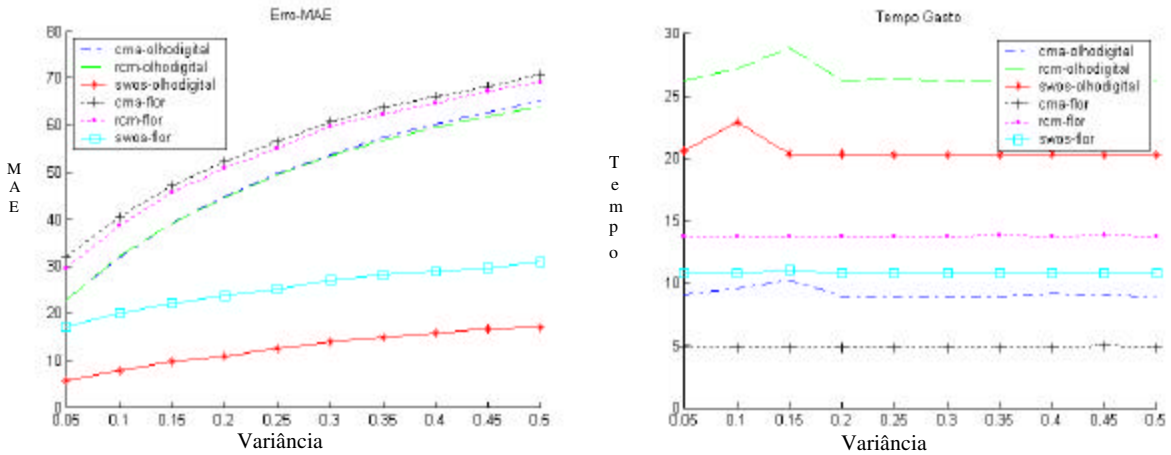


Figura 4.31 Vetores erro e tempo olhodigital e flor, gaussiano.

Para ruído gaussiano com média 0 e variâncias entre [0.05, 0.5], a imagem na entrada do filtro apresenta erro MAE de 33, 44, 52, 58, 62, 66, 70, 72, 75, respectivamente, para as duas imagens.

Os intervalos de erros da figura acima são [22.73, 65.1], [22.74, 64.0], [5.7, 16.9], para olhodigital e [31.9, 70.8], [29.6, 69.1], [17.1, 31.0] para flor, os filtros utilizados são CMA, RCM e SWOS respectivamente. Da primeira imagem (olhodigital), para a segunda (flor) temos um aumento do intervalo de erro. A imagem flor tem os detalhes mais nítidos (diferença de cores bem definidas) tornando a recuperação mais fácil. O filtro SWOS apresenta um melhor desempenho, seguido pelo filtro RCM e o CMA.

Os intervalos de tempo são [8.9, 10.2], [26.1, 28.8], [20.2, 22.9], para olhodigital (273x442) e [4.78, 4.98], [13.7, 13.8] e [10.7, 10.97] para flor (256x250), com os mesmos filtros, o tempo aumenta para a imagem maior, o que já era de se esperar.

Para o próximo teste ruído sal e pimenta, imagens olhodigital e flor, janela de observação 5x5.

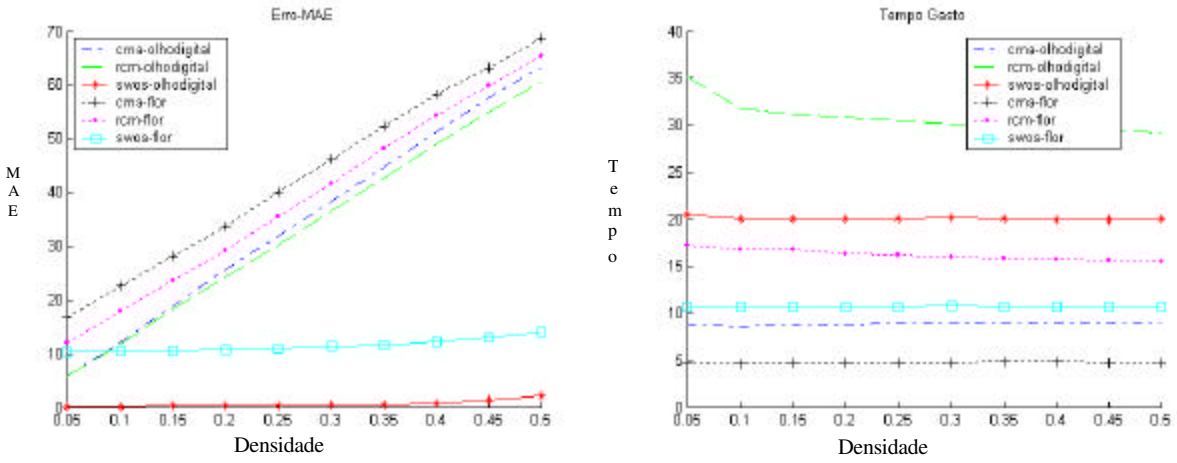


Figura 4.32 Vetores erro e tempo olhodigital e flor, sal e pimenta.

Para ruído sal e pimenta [0.05, 0.5], a imagem na entrada do filtro apresenta erro MAE de 6, 12, 19, 25, 32, 38, 44, 51, 57, 64 respectivamente, para as duas imagens.

Os intervalos de erros da figura acima são [5.8, 63.4], [5.7, 60.6], [0.06, 2.1], para olhodigital e [16.7, 68.7], [12.1, 65.3], [10.3, 14.0], para flor, os filtros utilizados são CMA, RCM e SWOS respectivamente. Da primeira imagem (olhodigital), para a segunda (flor), temos um aumento do intervalo de erro. O filtro SWOS apresenta um melhor desempenho, seguido pelo filtro RCM e o CMA.

Os intervalos de tempo são [8.6, 8.9], [29.2, 35.2], [19.9, 20.6], para olhodigital (273x442) e [4.6, 4.8], [15.5, 17.2], [10.6, 10.8], para flor (256x250), com os mesmos filtros, o tempo aumenta para a imagem maior, o que já era de se esperar.

Tanto para ruído gaussiano como sal e pimenta o melhor desempenho foi o do filtro SWOS, depois RCM e por fim o CMA.

4.1.9 Imagens cameraman e kids.

Para o próximo teste ruído sal e pimenta, imagens cameraman e kids, janela de observação 9x9.

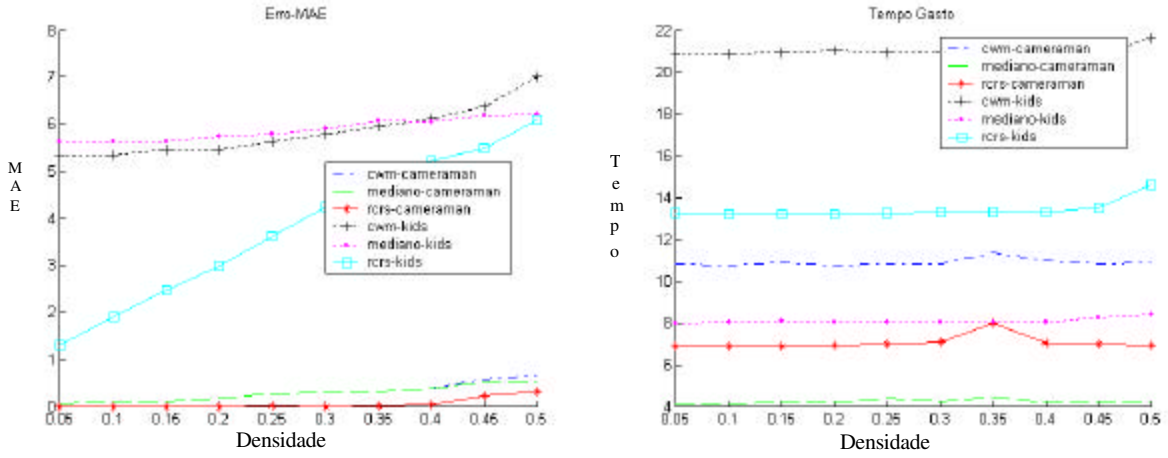


Figura 4.33 Vetores erro e tempo cameraman e kids, sal e pimenta.

Para ruído sal e pimenta [0.05, 0.5], a imagem na entrada do filtro apresenta erro MAE de 6, 12, 19, 25, 32, 38, 44, 51, 57, 64 respectivamente, para as duas imagens.

Os intervalos de erros da figura acima são [0.06, 0.68], [0.06, 0.5], [0, 0.3], para cameraman e [5.3, 7.0], [5.6, 6.2], [1.6, 5.6], para kids, os filtros utilizados são CWM, Mediano e RCRS respectivamente. Da primeira imagem (cameraman), para a segunda

(kids), temos um aumento do intervalo de erro, a imagem cameraman tem menos detalhes que a imagem kids. O filtro RCRS apresenta um melhor desempenho, seguido pelo filtro mediano e o CWM, cujos erros se confundem.

Os intervalos de tempo são [10.8, 11.3], [4.1, 4.4], [6.9, 7.9], para cameraman (512x512) e [20.9, 21.6], [7.9, 8.4], [13.2, 14.6], para kids (400x318), com os mesmos filtros, o tempo aumenta para a imagem maior, o que já era de se esperar.

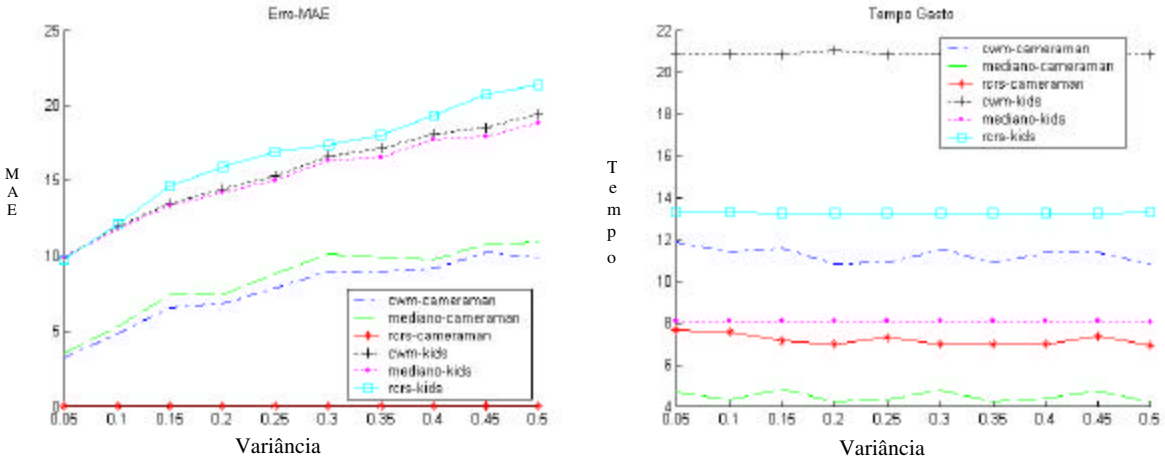


Figura 4.34 Vetores erro e tempo cameraman e kids, gaussiano.

Para ruído gaussiano com média 0 e variâncias entre [0.05, 0.5], a imagem na entrada do filtro apresenta erro MAE de 33, 44, 52, 58, 62, 66, 70, 72, 75, respectivamente, para as duas imagens.

Os intervalos de erros da figura acima são [3.3, 10.2], [3.5, 10.9], [0.02, 0.07], para cameraman e [9.9, 19.4], [9.92, 18.8], [9.8, 21.3], para kids, os filtros utilizados são CWM, Mediano e RCRS respectivamente. Da primeira imagem (cameraman), para a segunda (kids), temos um aumento do intervalo de erro. O filtro RCRS apresenta um melhor desempenho, seguido pelo filtro CWM e o mediano.

Os intervalos de tempo são [10.8, 11.9], [4.2, 4.8], [6.9, 7.6], para cameraman (512x512) e [20.8, 21.0], [8.05, 8.09], [13.2, 13.3], para kids (400x318), com os mesmos filtros, o tempo aumenta para a imagem maior, o que já era de se esperar.

Tanto para ruído sal e pimenta quanto para ruído gaussiano o filtro RCRS apresentou uma performance melhor, seguido pelos filtros CWM e Mediano. Para ruído sal e pimenta o erro dos filtros CWM e mediano se confundem.

4.1.10 Gráficos de erro ISNR para imagens diferentes.

Agora vamos comparar os gráficos de erro gerados por imagens diferentes (montage – 256x256 e couple – 494x512).

Aqui temos ruído sal e pimenta, janela 3x3 e imagens montage e couple.

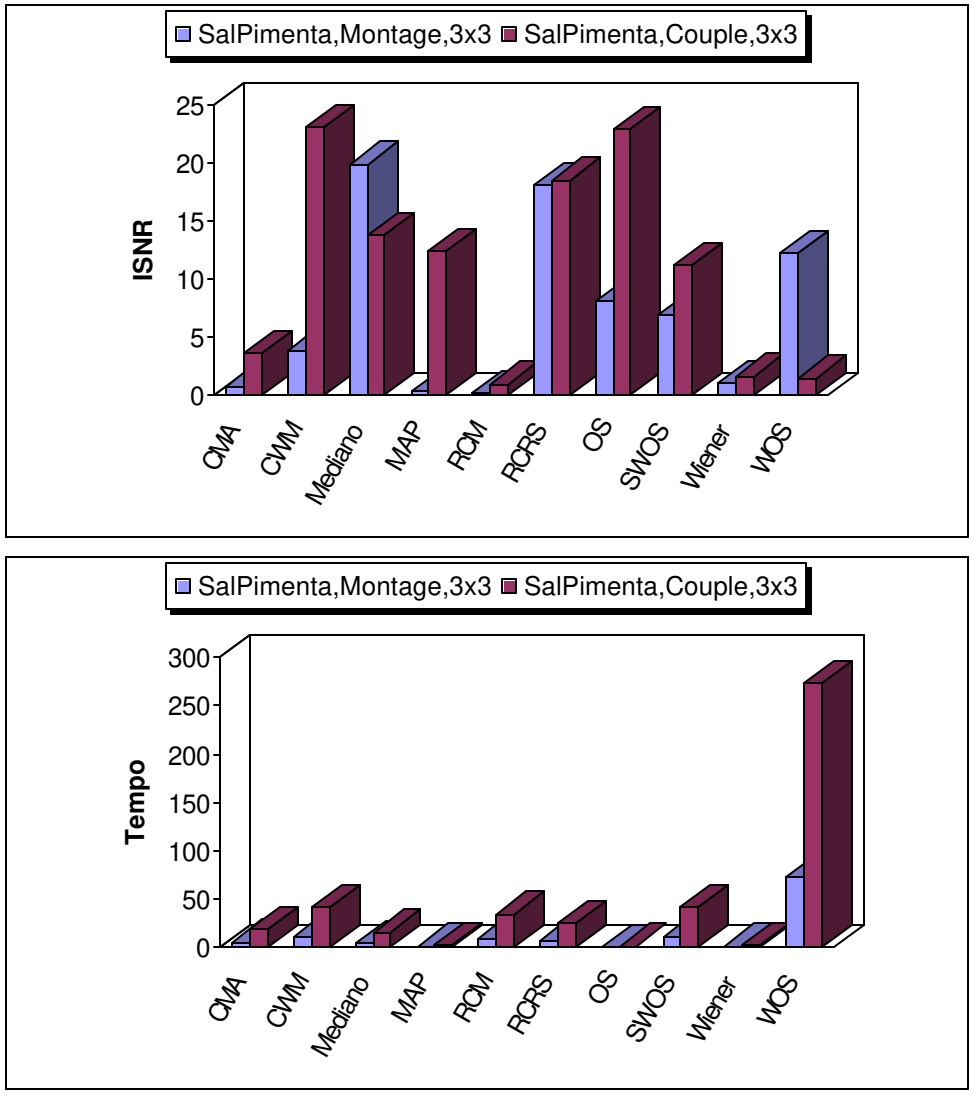


Figura 4.35 Ruído sal e pimenta, imagens montage e couple.

Os filtros: mediano, RCRS, WOS, OS e SWOS tiveram melhores respostas para a imagem montage. Para a imagem couple os melhores filtros foram CWM, OS, RCRS, mediano e MAP. O filtro RCM apresentou a pior resposta para as duas imagens.

O tempo de processamento apresentado pela imagem couple foi maior, o que já era de se esperar por a mesma ser maior.

Aqui temos ruído gaussiano, janela 3x3 e imagens montage e couple.

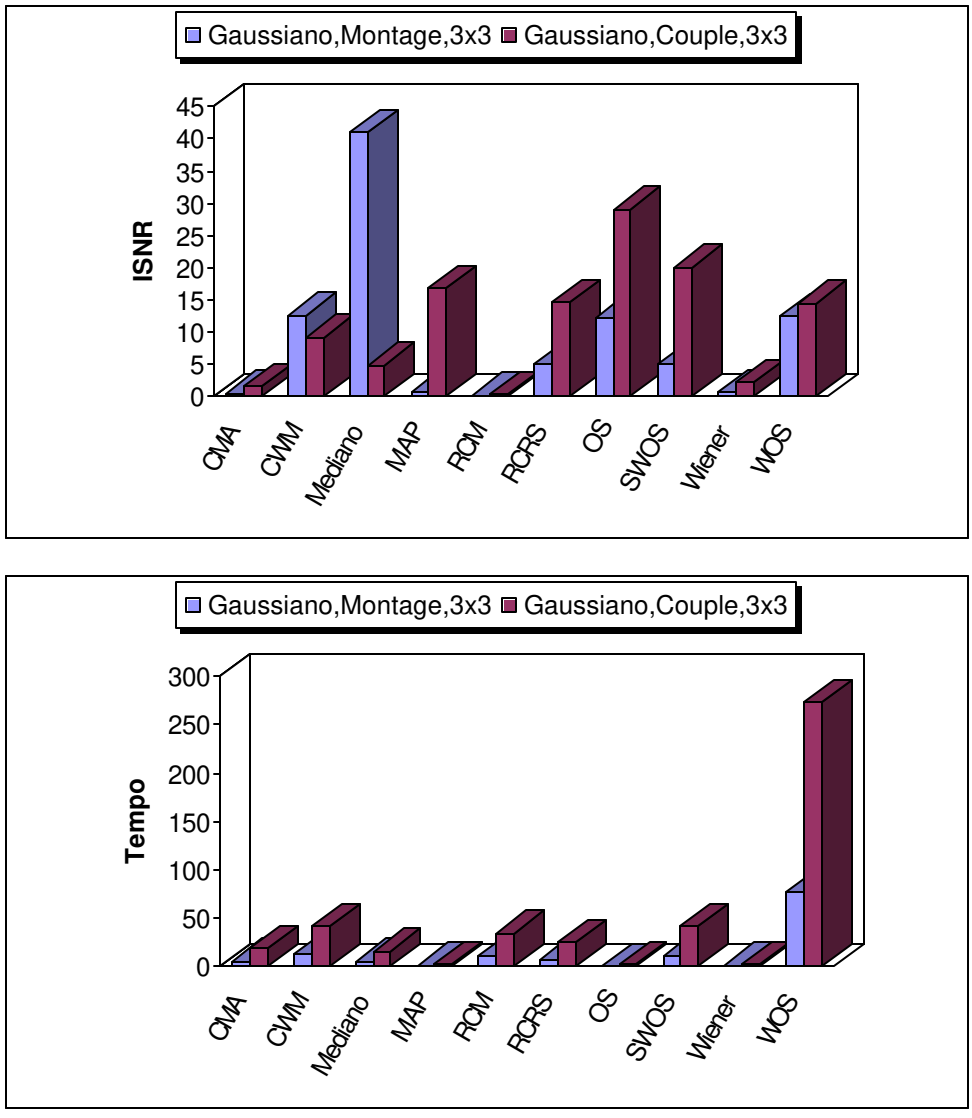


Figura 4.36 Ruído gaussiano, imagens montage e couple.

Os filtros: mediano, CWM, WOS, OS e RCRS tiveram melhores respostas para a imagem montage. Para a imagem couple os melhores filtros foram OS, SWOS, MAP, RCRS e WOS. O filtro RCM apresentou a pior resposta para as duas imagens.

O tempo de processamento apresentado pela imagem couple foi maior, o que já era de se esperar por a mesma ser maior.

Aqui temos ruído sal e pimenta, janelas 3x3 e 7x7, imagem couple.

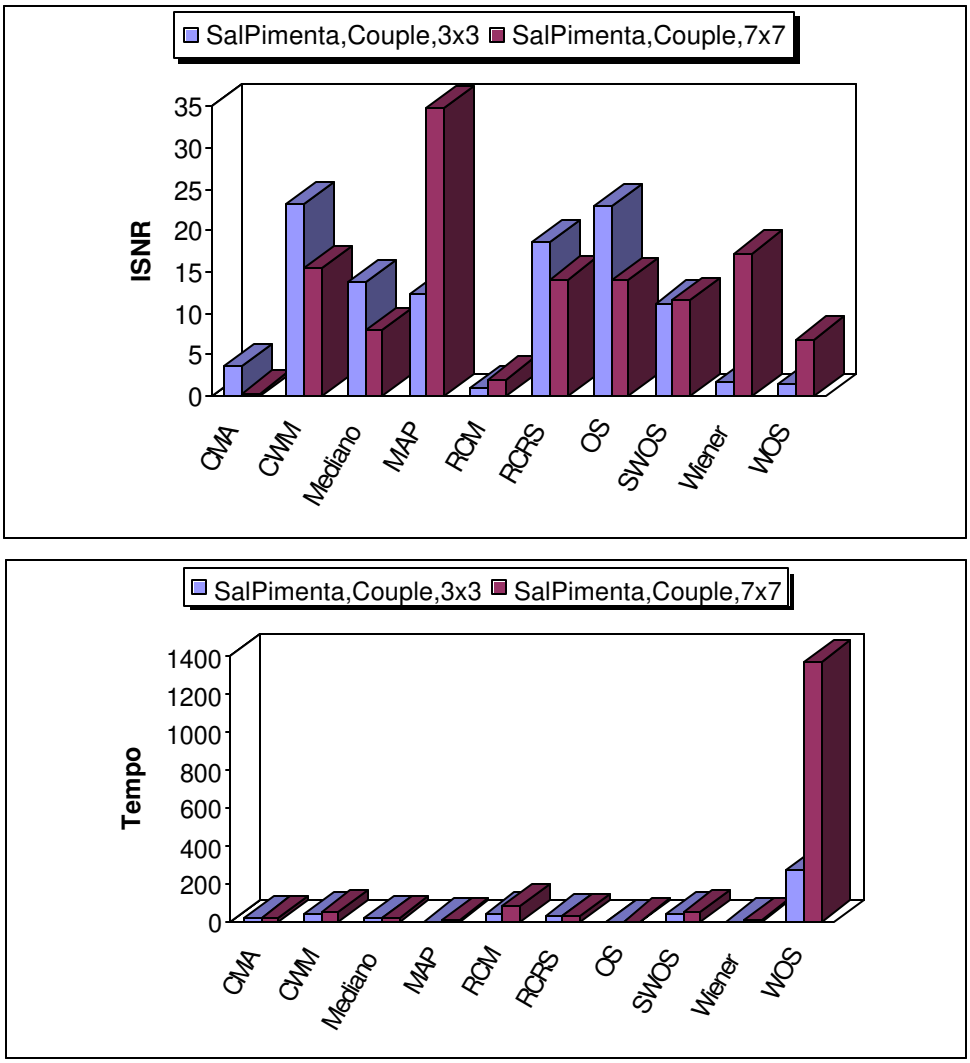


Figura 4.37 Ruído sal e pimenta, janela 3x3 e 7x7, couple.

Os filtros: CWM, OS, RCRS, mediano e MAP tiveram melhores respostas para janela de observação 3x3. Para janela de observação 7x7 os melhores filtros foram

MAP, Wiener, mediano, OS e RCRS. Os filtros RCM e CMA apresentaram as piores respostas para janelas de observação 3x3 e 7x7 respectivamente.

O tempo de processamento foi maior para janela de observação 7x7.

Aqui temos ruído gaussiano, janelas 3x3 e 7x7, imagem couple.

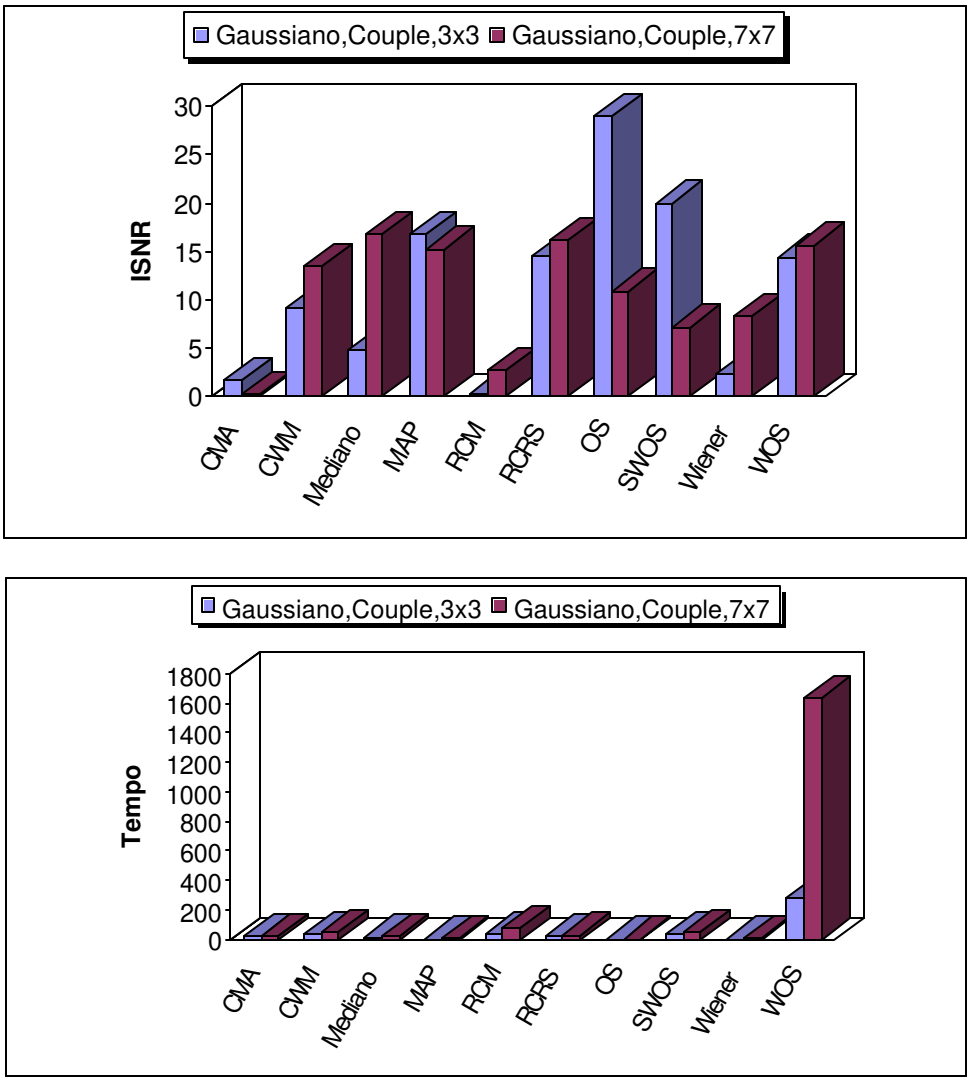


Figura 4.38 Ruído gaussiano, janela 3x3 e 7x7, couple.

Os filtros: OS, SWOS, MAP, RCRS e WOS tiveram melhores respostas para janela de observação 3x3. Para janela de observação 7x7 os melhores filtros foram:

mediano, RCRS, WOS, MAP e CWM. Os filtros RCM e CMA apresentaram as piores respostas para janelas de observação 3x3 e 7x7 respectivamente.

O tempo de processamento foi maior para janela de observação 7x7.

Aqui temos ruído sal e pimenta e gaussiano, janela 3x3, imagem couple.

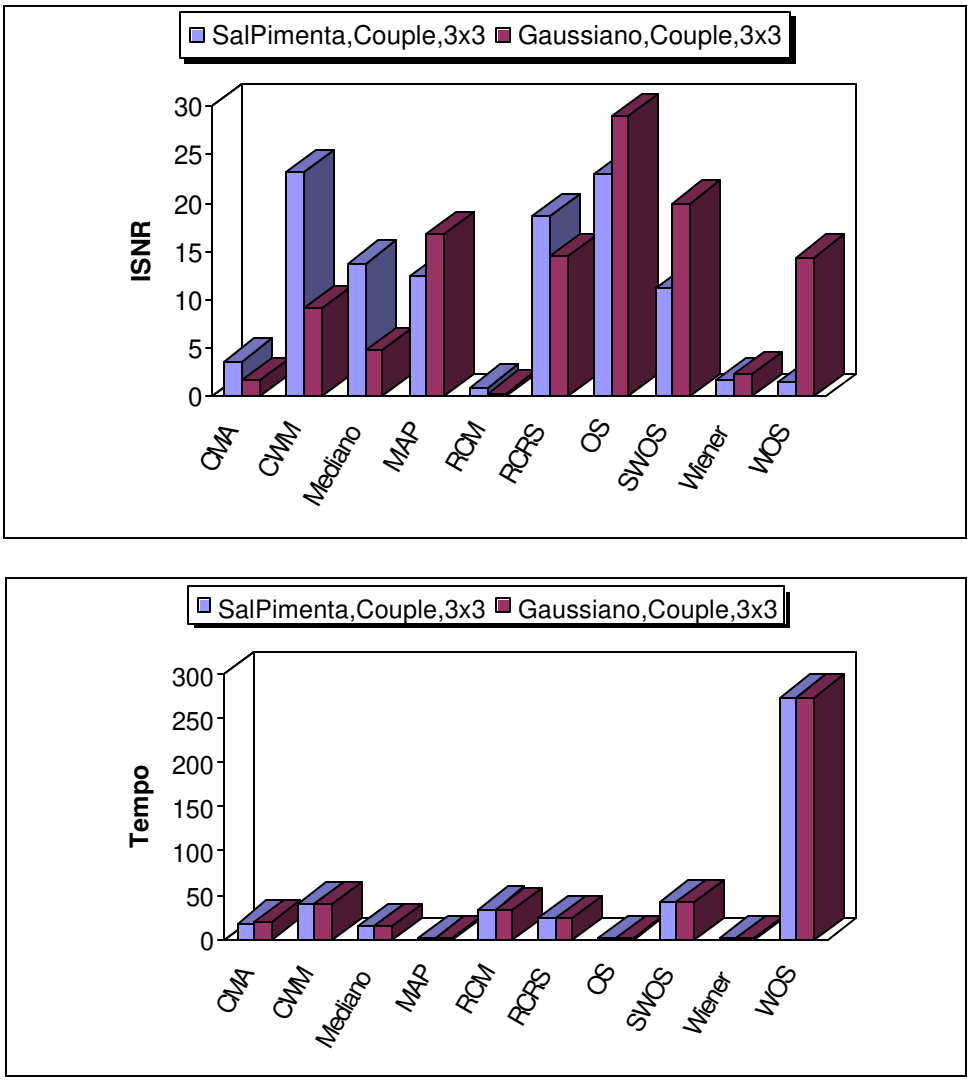


Figura 4.39 Ruído sal e pimenta e gaussiano, janela 3x3, couple.

Os filtros: CWM, OS, RCRS, mediano e MAP tiveram melhores respostas para ruído sal e pimenta. Para ruído gaussiano os melhores filtros foram: OS, SWOS, MAP,

RCRS e WOS. O filtro RCM apresentou a pior resposta para ruído sal e pimenta e gaussiano.

Aqui temos ruído sal e pimenta e gaussiano, janela 7x7, imagem couple.

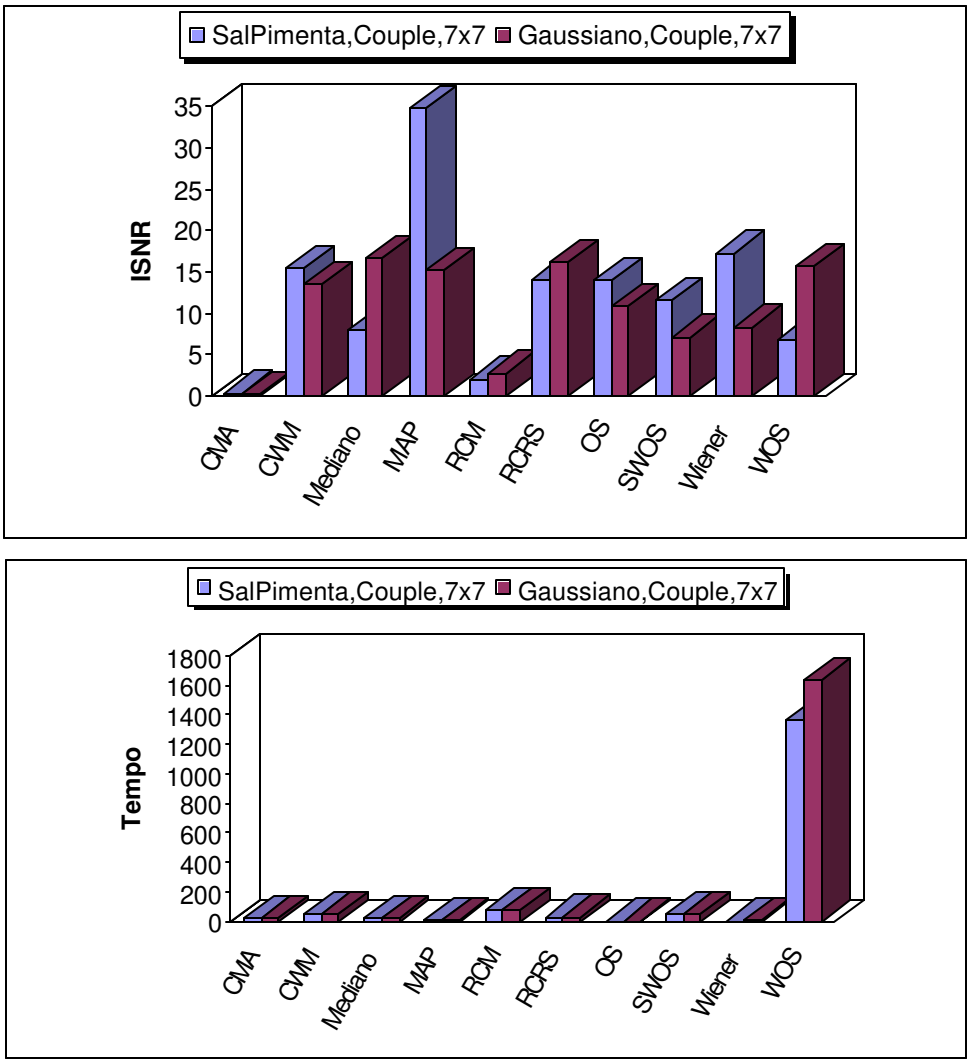


Figura 4.40 Ruído sal e pimenta e gaussiano, janela 7x7, couple.

Os filtros: MAP, Wiener, mediano, OS, e RCRS tiveram melhores respostas para ruído sal e pimenta. Para ruído gaussiano os melhores filtros foram: mediano, RCRS, WOS, MAP e CWM. O filtro CMA apresentou a pior resposta para ruído sal e pimenta e gaussiano.

4.2 Desempenho visual dos Filtros

No capítulo 3 mostramos o comportamento de cada filtro com relação à imagem lena em com dados de entrada iguais, a partir desse momento faremos o mesmo com a imagem boat.

4.2.1 Imagem boat, erro MAE.

A seguir temos a Figura 4.24, que é a imagem boat corrompida por ruído do tipo sal e pimenta densidade 0.12.

Figura 4.41 Imagem Corrompida boat, 512x512.

O erro MAE apresentado pela imagem degradada foi aproximadamente 15.

Para restauração da imagem utilizaremos uma janela de observação de tamanho 5x5.

Imagem CMA – MAE 14.7

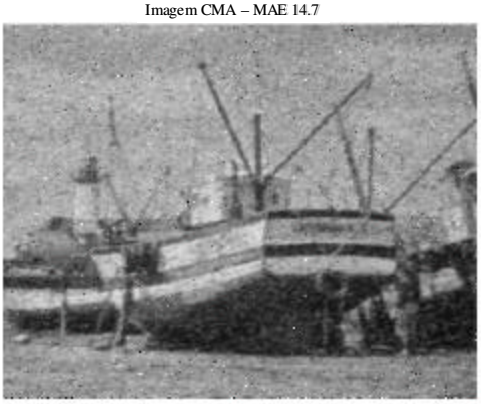


Imagem CWM – MAE 7.9

Imagem Mediano – MAE 8.3

Imagem MAP – MAE 16.8

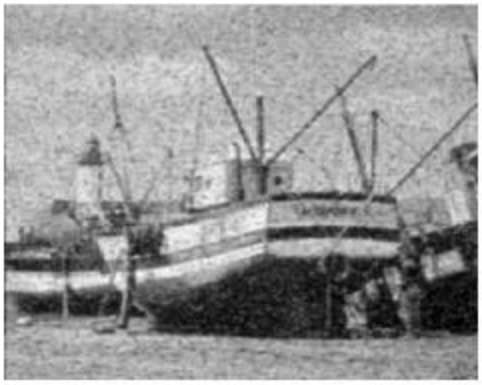


Imagem RCM – MAE 14.1

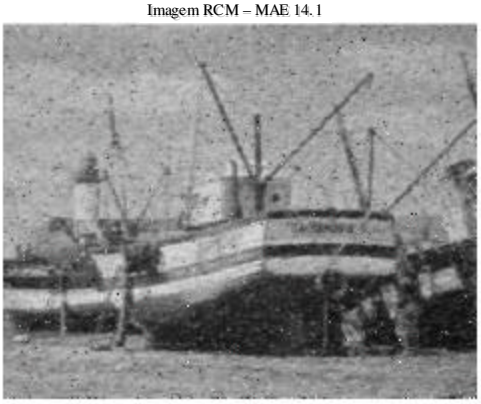


Imagem RCRS – MAE 2.0

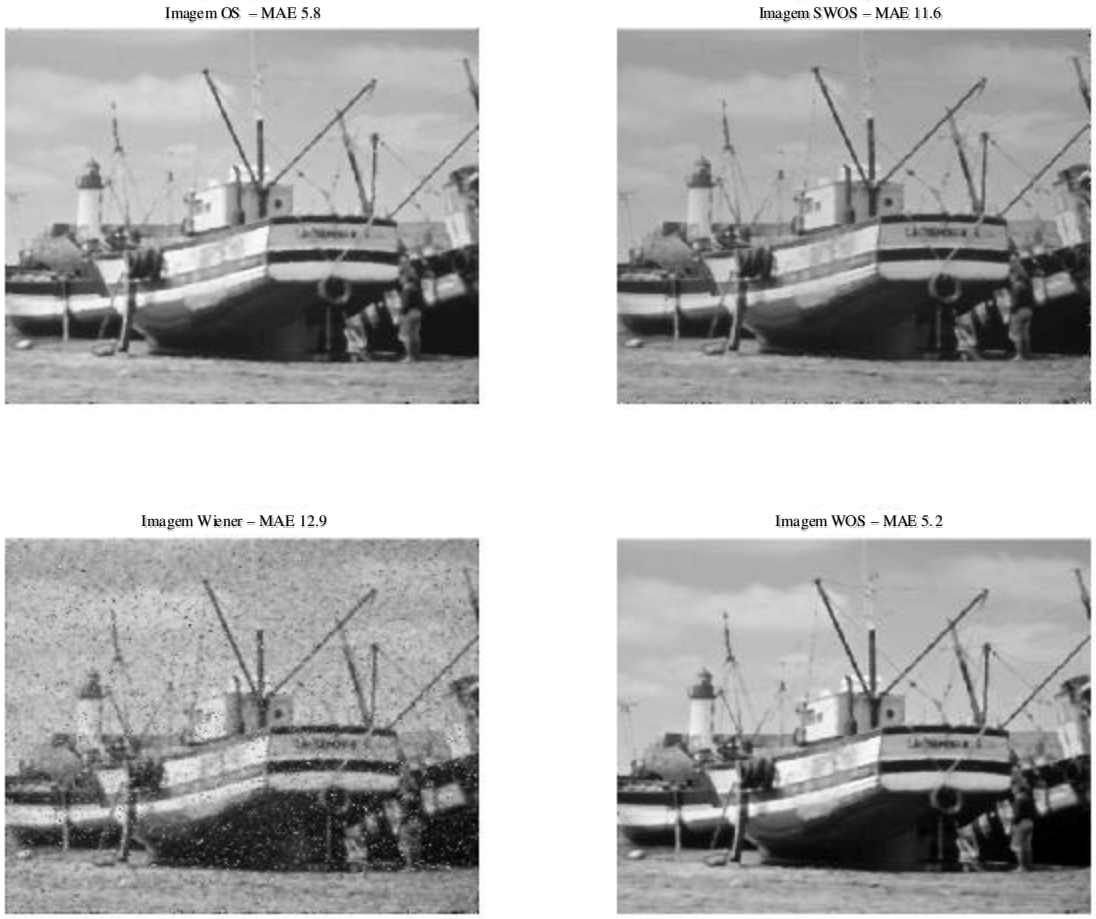


Figura 4.42 Imagens na saída dos filtros

A seguir temos os gráficos dos erros e tempos gastos.

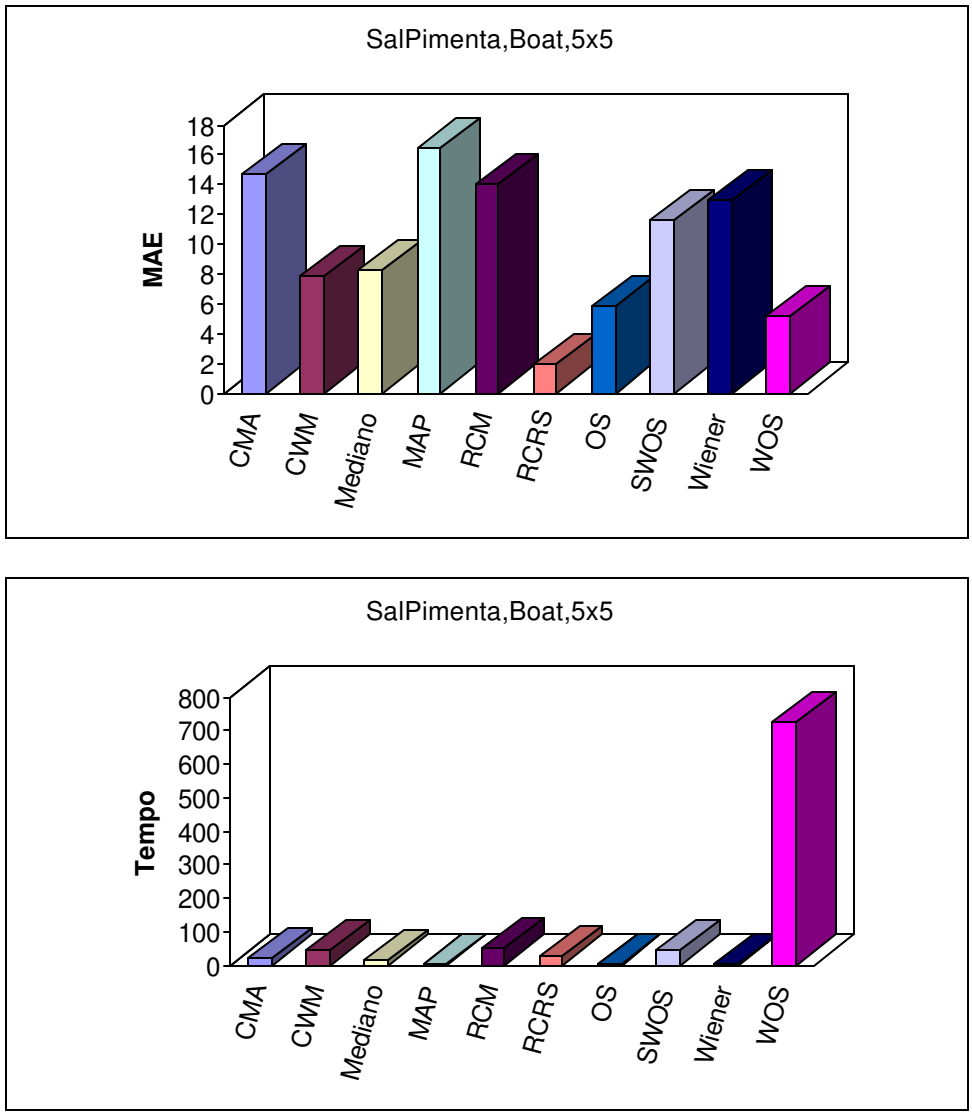


Figura 4.43 Vetores erro e tempo gasto

O erro MAE na saída de cada filtro é: 14.7 (CMA), 7.9 (CWM), 8.3 (mediano), 16.8(MAP), 14.1 (RCM), 2.0 (RCRS), 5.8 (OS), 11.6 (SWOS), 12.9 (Wiener), 5.2 (WOS). Somente o filtro MAP não apresentou um erro MAE menor do que o apresentado pela imagem de entrada (erro MAE na entrada do filtro 15). Logo, com relação ao erro, os filtros RCRS, WOS e OS tiveram melhores desempenhos. O tempo de processamento de cada filtro é: 19 (CMA), 44 (CWM), 17 (Mediano), 4 (MAP), 54 (RCM), 28 (RCRS), 1 (OS), 45 (SWOS), 2 (Wiener), 750 (WOS) segundos. Logo, com relação ao tempo gasto, o filtro OS um desempenho melhor.

4.2.2 Imagem montage, erro ISNR.

Nesta seção apresentamos testes realizados com a imagem montage e as respectivas medidas ISNR (eq. 3.2-3, pág. 35)

Primeiramente apresentamos testes com a janela de observação 5x5 e ruído sal e pimenta de densidade 0.04.

Imagem corrompida

Imagem CMA – ISNR 2.2

Imagem CWM – ISNR 14.7

Imagem Mediano – ISNR 11.9

Imagem MAP – ISNR 3.2

Imagem RCM – ISNR 1.7

Imagem RCRS – ISNR 26.8

Imagem OS – ISNR 14.9

ImagemSWOS – ISNR 11.0

Figura 4.44 Imagem montage, sal e pimenta, 5x5.

O erro ISNR na saída de cada filtro é: 2.2 (CMA), 14.7 (CWM), 11.9 (Mediano), 3.2 (MAP), 1.7 (RCM), 26.8 (RCRS), 14.9 (OS), 11.0 (SWOS), 1.9 (Wiener), 15.7 (WOS). Nesse caso, com relação ao erro os filtros RCRS, CWM e SWOS tiveram os melhores desempenhos respectivamente. O tempo de processamento de cada filtro é: 5 (CMA), 11 (CWM), 4 (Mediano), 0.9 (MAP), 15 (RCM), 7 (RCRS), 0.2 (OS), 11 (SWOS), 0.7 (Wiener), 183 (WOS) segundos. Com relação ao tempo gasto, o filtro OS, teve o melhor desempenho.

A seguir apresentamos os testes com janela de observação 5x5, ruído gaussiano com média 0 e variância 0.04.

Imagem CMA – ISNR 2.2

Imagem CWM – ISNR 9.7

Imagem Mediano – ISNR 4.9

Imagem MAP – ISNR 4.4

Imagem RCM – ISNR 1.7

Imagem RCRS – ISNR 11.8

Figura 4.45 Imagem montage, gaussiano, 5x5.

O erro ISNR na saída de cada filtro é: 2.2 (CMA), 9.7 (CWM), 4.9 (Mediano), 4.4 (MAP), 1.7 (RCM), 11.8 (RCRS), 9.9 (OS), 4.9 (SWOS), 12.6 (Wiener), 10.1 (WOS). Nesse caso, com relação ao erro os filtros CWM, SWOS e RCRS tiveram os melhores desempenhos respectivamente. O tempo de processamento de cada filtro é: 5 (CMA), 11 (CWM), 4 (Mediano), 0.9 (MAP), 15 (RCM), 7 (RCRS), 0.2 (OS), 11 (SWOS), 0.6 (Wiener), 184 (WOS) segundos. Com relação ao tempo gasto, o filtro OS, teve o melhor desempenho.

A seguir apresentamos os testes com janela de observação 3x3, ruído sal e pimenta densidade 0.04.

Imagem corrompida

Imagem CMA – ISNR 0.8

Imagem CWM– ISNR 3.8

Imagem Mediano – ISNR 19.9

Imagem MAP – ISNR 0.3

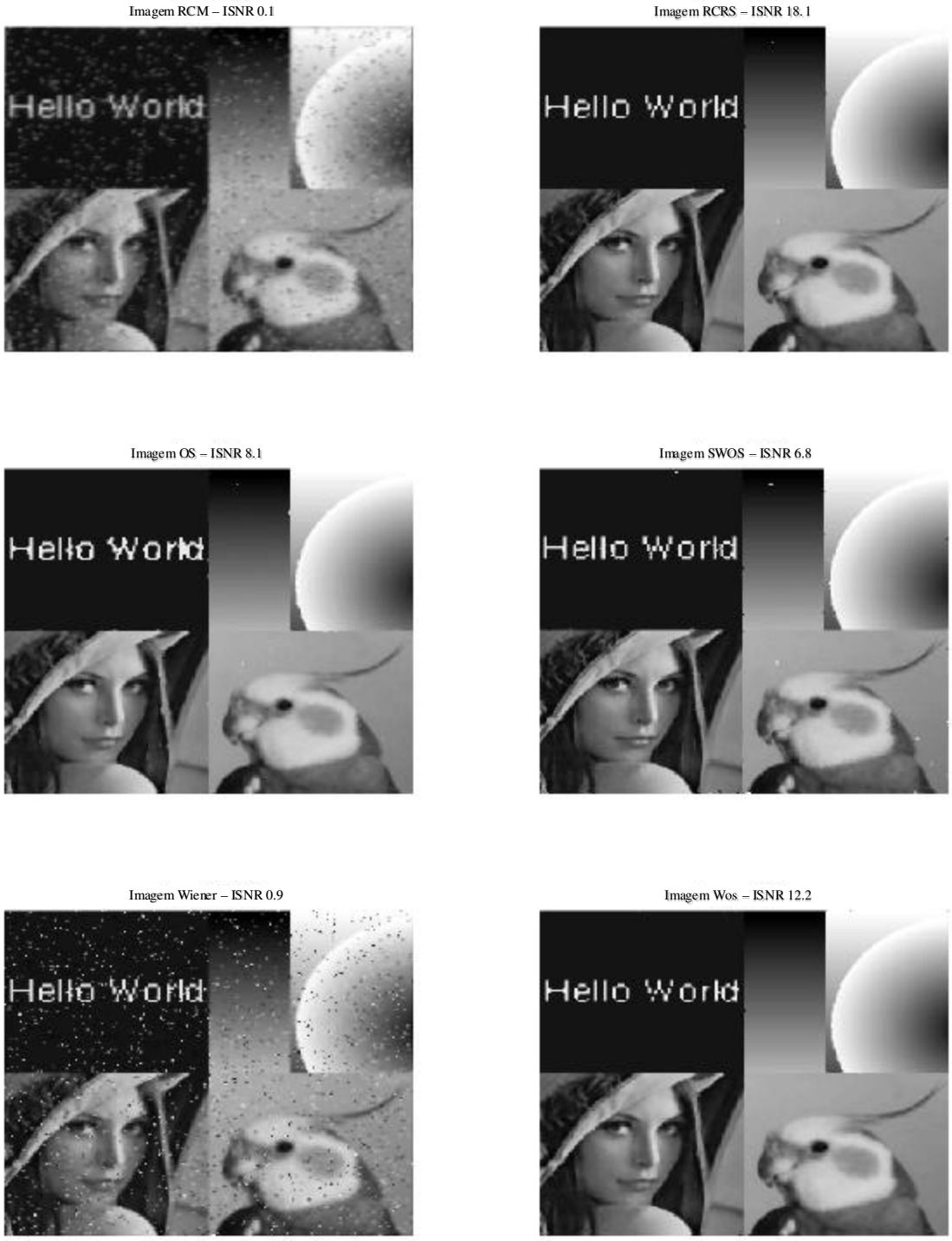


Figura 4.46 Imagem montage, sal e pimenta, 3x3.

O erro ISNR na saída de cada filtro é: 0.8 (CMA), 3.8 (CWM), 19.9 (Mediano), 0.3 (MAP), 0.1 (RCM), 19.1 (RCRS), 8.1 (OS), 6.8 (SWOS), 0.9 (Wiener), 12.2 (WOS).

Nesse caso, com relação ao erro, os filtros RCRS e WOS tiveram os melhores desempenhos respectivamente. O tempo de processamento de cada filtro é: 5 (CMA), 11 (CWM), 4 (Mediano), 0.5 (MAP), 9 (RCM), 7 (RCRS), 0.1 (OS), 11 (SWOS), 0.5 (Wiener), 73 (WOS) segundos. Com relação ao tempo gasto, o filtro OS, apresentou o melhor desempenho.

Para o próximo conjunto de testes consideraremos janela 3x3, ruído gaussiano com média 0 e variância 0.04.

Imagem Mediana

Imagem MAP

Imagem RCM

Imagem RCRS

Imagem OS

Imagem SWOS

Figura 4.47 Imagem montage, gaussiano, 3x3.

O erro ISNR na saída de cada filtro é: 0.4 (CMA), 12.8 (CWM), 40.9 (Mediano), 0.8 (MAP), 0.04 (RCM), 4.9 (RCRS), 12.5 (OS), 4.8 (SWOS), 0.4 (Wiener), 12.4 (WOS). Nesse caso, com relação ao erro os filtros Mediano, WOS, CWM e OS tiveram os melhores desempenhos respectivamente. O tempo de processamento de cada filtro é: 5 (CMA), 12 (CWM), 5 (Mediano), 0.5 (MAP), 9 (RCM), 7 (RCRS), 0.1 (OS), 11 (SWOS), 0.4 (Wiener), 76 (WOS) segundos. Com relação ao tempo gasto o filtro OS, teve o melhor desempenho.

Abaixo temos os vetores de erro ISNR e tempo apresentados pelos filtros nos últimos 4 testes (item 4.1.7.2), para melhor visualizarmos a resposta dos filtros quando aumentamos a janela de observação, ou inserimos ruídos diferentes.

Aqui temos ruído sal e pimenta e gaussiano com janela 5x5.

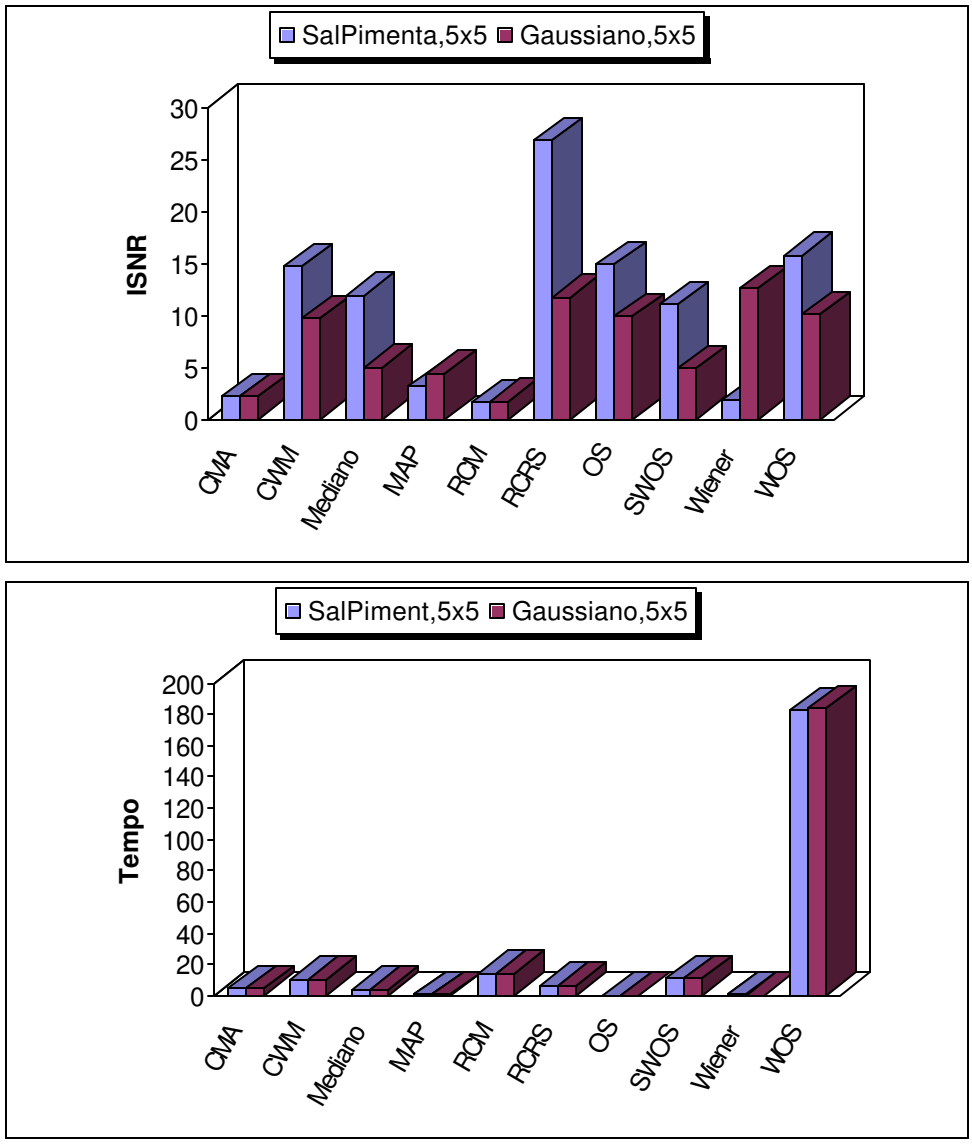
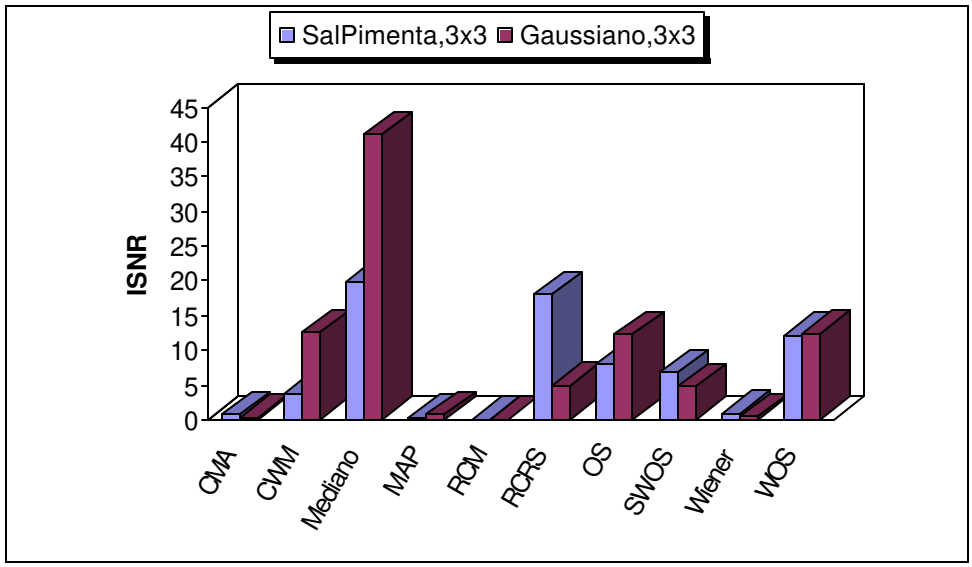


Figura 4.48 Ruído sal e pimenta e gaussiano, janela 5x5

Para ruído sal e pimenta os filtros: RCRS, WOS, OS, CWM e mediano tiveram uma melhor performance. Já para ruído gaussiano temos os filtros: Wiener, RCRS, WOS, OS e CWM com melhor performance. O filtro RCM apresentou a pior performance para ruído sal e pimenta e gaussiano. Os vetores de tempo, nesses dois casos se confundem, o que já era de se esperar por ser a mesma imagem.

Aqui temos ruído sal e pimenta e gaussiano com janela 3x3.



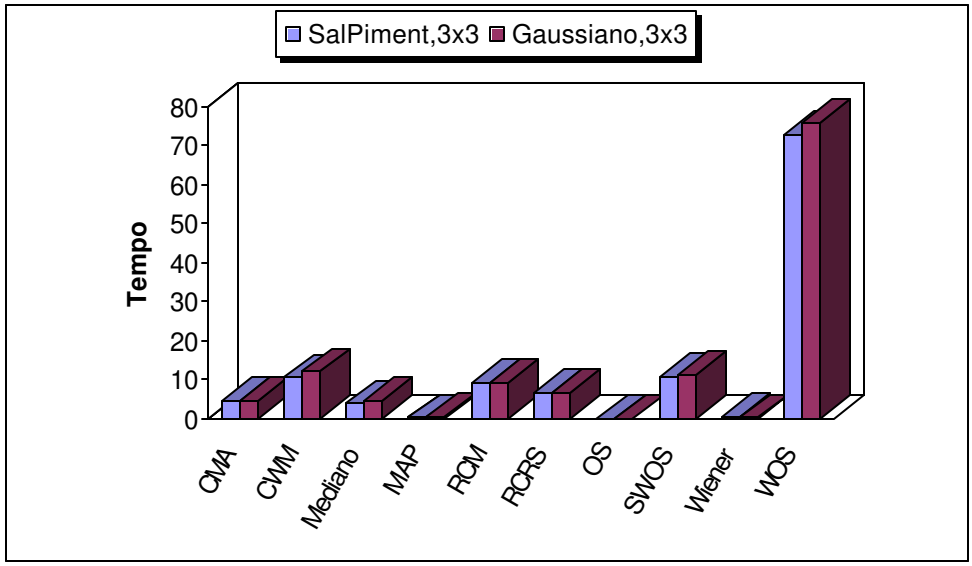


Figura 4.49 Ruído sal e pimenta e gaussiano, janela 3x3

Para ruído sal e pimenta os filtros: mediano, RCRS, WOS, OS e SWOS tiveram uma melhor performance. Já para ruído gaussiano temos os filtros: mediano, CWM, WOS, OS e RCRS com melhor performance. O filtro RCM apresentou a pior performance para ruído sal e pimenta e gaussiano. Os vetores de tempo, novamente se confundem.

Aqui temos ruído sal e pimenta, com janelas 3x3 e 5x5.

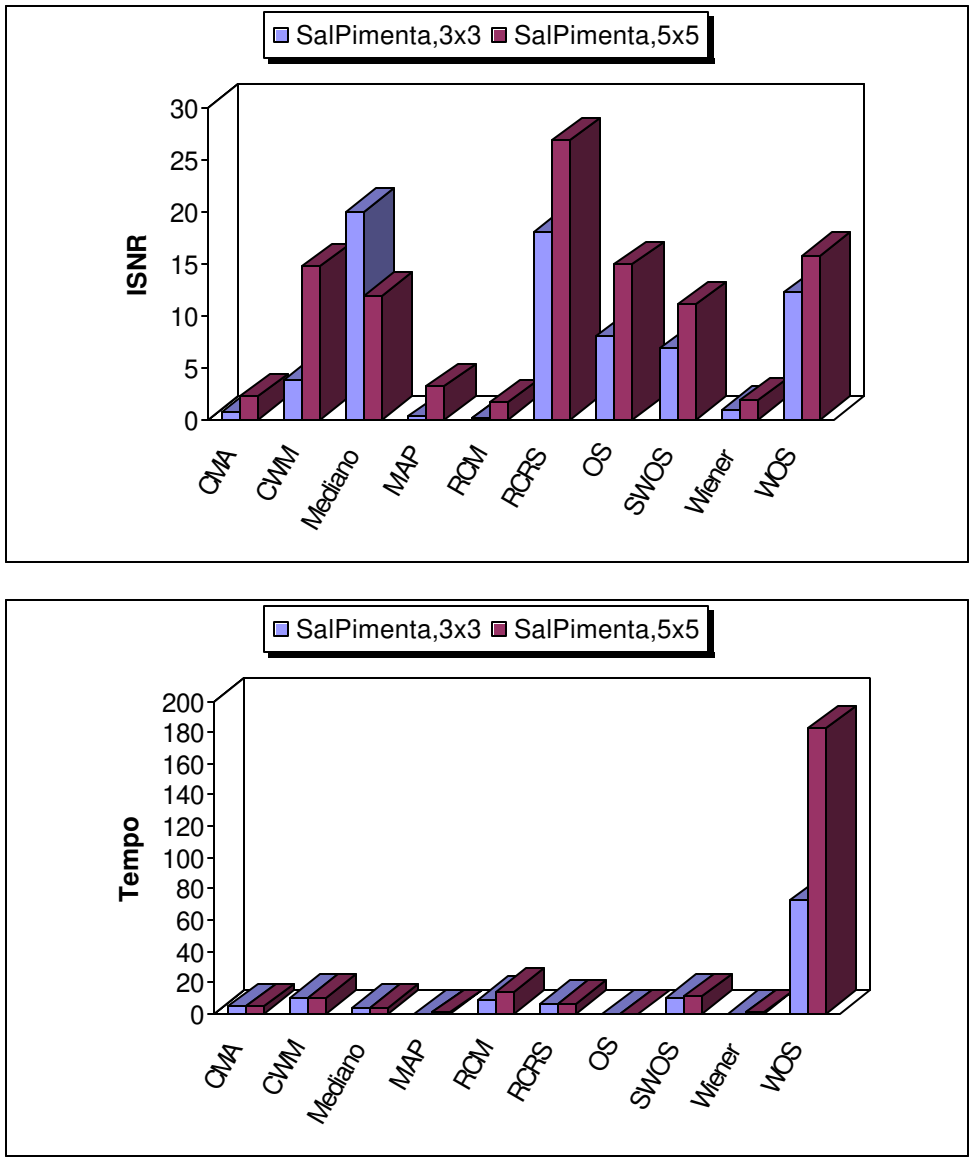


Figura 4.50 Ruído sal e pimenta, janelas 3x3 e 5x5.

Para janela de observação 3x3 os filtros: mediano, RCRS, WOS, OS e SWOS tiveram uma melhor performance. Já para janela de observação 5x5 temos os filtros: RCRS, WOS, OS, CWM e mediano com melhor performance. O filtro RCM apresentou a pior performance para ruído sal e pimenta e gaussiano. Os vetores de tempo, novamente se confundem.

Aqui temos ruído gaussiano, com janelas 3x3 e 5x5.

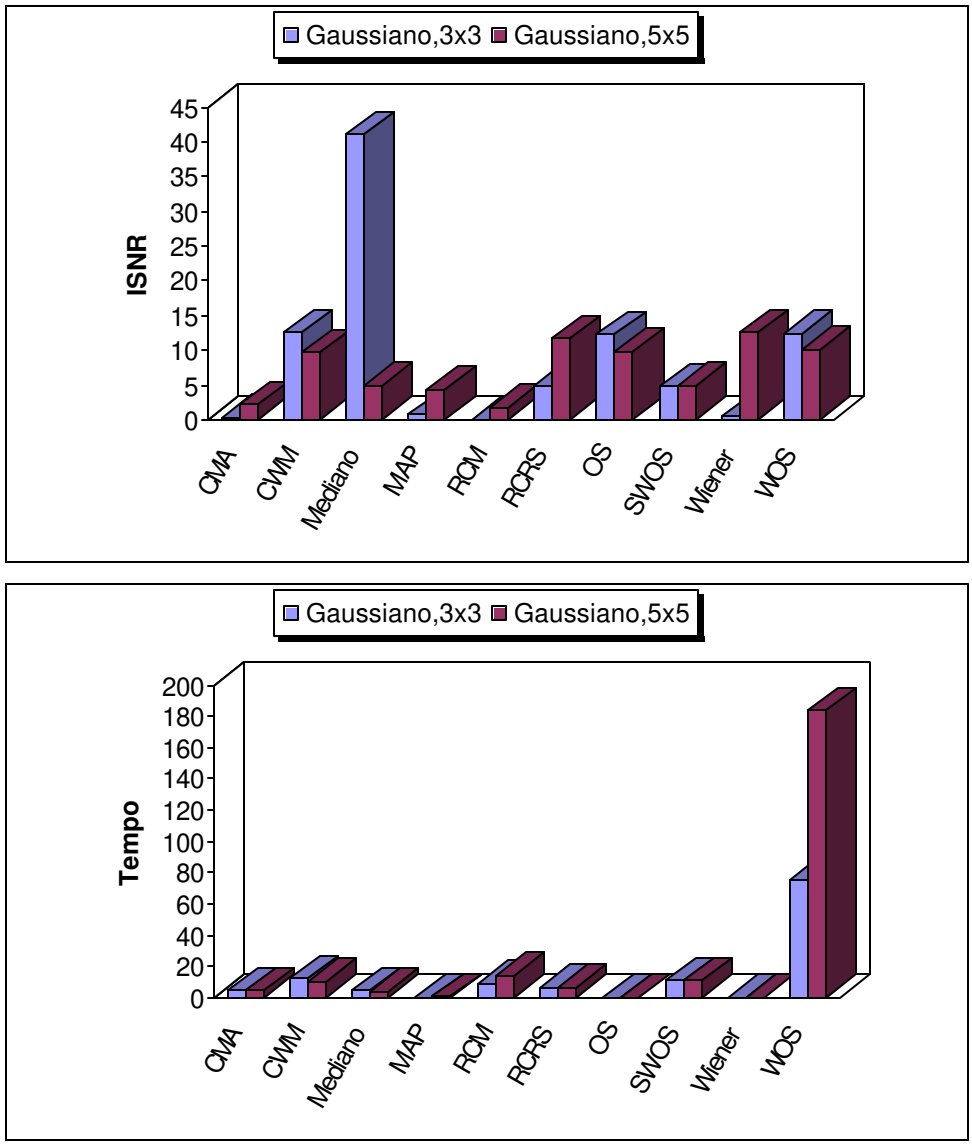


Figura 4.51 Ruído gaussiano, janelas 3x3 e 5x5

Para janela de observação 3x3 os filtros: mediano, CWM, WOS, OS e RCRS tiveram uma melhor performance. Já para janela de observação 5x5 temos os filtros: Wiener, RCRS, WOS, OS e CWM com melhor performance. O filtro RCM apresentou a pior performance para ruído sal e pimenta e gaussiano. Os vetores de tempo, novamente se confundem.

4.3 Algumas conclusões

Os filtros RCRS, OS, CWM, SWOS e WOS tem um melhor desempenho para janela de observação maiores que 3x3, isto para ruído do tipo sal e pimenta e gaussiano. O que também ocorre para o filtro MAP somente para ruído gaussiano.

Os filtros CMA (ruído sal e pimenta e gaussiano), Mediano (ruído sal e pimenta e gaussiano) e RCM (ruído gaussiano) quando aumentamos a janela de observação o erro também aumenta, ou seja, estes filtros tendem a serem mais eficientes com janelas de observação menores.

O filtro RCM de uma maneira geral foi o que apresentou pior desempenho.

O desempenho dos filtros considerando janela de observação e tipo de ruído é o seguinte:

✍✍Para janela de observação 3x3:

- ruído sal e pimenta os filtros: mediano, RCRS, WOS, OS e CWM tendem a ter uma resposta melhor que os demais;
- ruído gaussiano os filtros: mediano, CMA, MAP, Wiener e RCRS tendem a ter uma resposta melhor que os demais.

✍✍Para janela de observação 5x5:

- ruído sal e pimenta os filtros: RCRS, WOS, OS, CWM e mediano tendem a ter uma resposta melhor que os demais;
- ruído gaussiano os filtros: Wiener, RCRS, WOS, OS e CWM tendem a ter uma resposta melhor que os demais.

✍✍Para janela de observação 7x7:

- ruído sal e pimenta os filtros: RCRS, WOS, OS, CWM e mediano tendem a ter uma resposta melhor que os demais;

- ruído gaussiano os filtros: MAP, Wiener, RCRS, CMA e mediano tendem a ter uma resposta melhor que os demais

✍✍Para janela de observação 9x9:

- ruído sal e pimenta os filtros: RCRS, WOS, OS, CWM e mediano tendem a ter uma resposta melhor que os demais;

- ruído gaussiano os filtros: MAP, RCRS, WOS, OS e Wiener tendem a ter uma resposta melhor que os demais

Quanto ao tempo de processamento quanto maior a janela de observação maior o tempo. O filtro WOS trabalha com vetores bem maiores do que os demais filtros, o mesmo, requer mais memória é mais demorado. Uma sugestão para melhorar a performance do filtro WOS é implementá-lo em C ou C++ por exemplo.

CAPÍTULO 5

CONSIDERAÇÕES FINAIS E SUGESTÕES DE TRABALHOS FUTUROS

Neste capítulo faremos uma revisão geral do trabalho e algumas sugestões para trabalhos futuros. Este trabalho de conclusão de mestrado teve como principal objetivo a utilização de filtros para a recuperação de imagens degradadas.

Com relação ao filtro Passa-Baixas, notamos que o mesmo tende a minimizar ruídos e apresentar o efeito de borramento da imagem. Já no caso do filtro Passa-Altas, que realçam detalhes, atenuam ou eliminam os componentes de baixa-freqüência, notamos que a imagem recuperada ficou pior do que a corrompida, realça as diferenças (ruído). Logo esse filtro não foi utilizado para efeito de comparação.

Observamos também filtros não-lineares, que são bem mais eficientes em extrair ruído sem distorcer as características da imagem. Após análise dos testes, verificamos que filtros não-lineares apresentam um desempenho superior quando comparados aos filtros lineares.

Dentre os filtros não-lineares temos alguns que se destacam pelo desempenho, bem como outras características.

- Filtro condicional de média aritmética que algumas vezes tem uma resposta até melhor do que o seu antecessor, porém temos a dificuldade de encontrar o limiar (U), para que isso ocorra.

- Filtro de média aritmética ponderada que, dependendo da aplicação funciona bem.

- Filtro condicional mediano que depende de parâmetros, o que dificulta o trabalho e faz com que o sucesso fique ainda mais dependente da aplicação.

- O Filtro de ordem estatística que tem como saída uma ordem estatística de um conjunto de amostras observadas em sua entrada, mostrou-se muito eficiente, no que se refere a recuperar a imagem e preservar as bordas da mesma, o que não ocorria nos anteriores. Esse também apresenta algumas variações bem interessantes, as quais diferem unicamente no tipo de informação utilizada para decidir a ordem estatística (amostra), a ser selecionada.

- O Filtro Mediano apresenta um baixo nível de complexidade computacional, e recupera a imagem com relativo sucesso.

- Filtro mediano ponderado central tem como saída o pixel mediano do conjunto de entradas ordenado, depois de aplicado o peso à amostra central.

- Filtro de ordem estatística ponderada simples difere do anterior (Filtro mediano ponderado central), pela forma de escolha da saída do filtro, no conjunto ordenado. Neste caso a saída é uma das amostras do conjunto ordenado.

- Filtro de ordem estatística ponderada: neste caso o peso pode ser aplicado a todas as amostras, estes filtros foram definidos devido à limitação dos filtros SWOS. O problema aqui é encontrar uma máscara de ponderação que funcione satisfatoriamente.

- Filtros condicionais seletores de posição utilizam a ordem das amostras escolhidas como base para a seleção da posição, oferece uma considerável flexibilidade de implementação e os filtros de baixa ordem são relativamente simples de implementar e otimizar. O aumento da ordem do filtro tem como conseqüência o aumento da complexidade computacional. Mostrou-se muito eficiente, nos testes feitos.

Em particular observamos que os filtros RCRS tiveram um desempenho superior, quando comparado aos demais. Isto se dá, principalmente, pela capacidade dos filtros RCRS de minimizarem o ruído sem alterarem a estrutura da imagem.

Como sugestão para trabalhos futuros, poderíamos utilizar uma linguagem de programação que nos desse maior flexibilidade e desempenho, como o C ou C++, por exemplo. Alguns filtros têm um tempo de processamento bem alto, como podemos perceber pelos testes exibidos. Como já foi dito, em processamento de imagens nem sempre uma solução é boa para todas as aplicações e em alguns casos talvez seja necessário a utilização de algum filtro não implementado. Poderia ser testado também outro tipo de ruído e até mesmo outras dimensões de janela de observação. O que se percebe quando trabalhamos nessa área é que ainda há muito a se pesquisar/desenvolver. Reiterando, temos em mãos uma área relativamente nova e por isso um campo vasto para pesquisa.

INTERNET LINKS

Waterloo BragZone http://links.uwaterloo.ca/bragzone.base.html.

The USC-SIPI Image Database - http://sipi.usc.edu/services/database/Database.html

Lotufo, Roberto de Alencar. Visão Computacional.
http://www.dca.fee.unicamp.br/~lotufo/

Veldhuizen, Todd. Grid Filters for Local Nonlinear Image Restoration. Janeiro 1998.
http://osl.iu.edu/~tveldhui/papers/MAScThesis/node1.html

Chakrabarti, Chaitali; Lucke, Lori. VLSI Architectures for Weighted Order Statistic (WOS) Filters.
http://www.minnetronix.com/assets/pubs%20images%20and%20pdfs/VLSI%20for%20WOS%20pdf/WOSFilters.pdf.

Introdução ao Processamento e à Análise Digital de Imagens - Departamento de Ciência dos Materiais e Metalurgia da PUC-Rio - http://www.dcmm.puc-rio.br/Cursos/IMAGEM/

Reunião Anual da SBPC - Goiânia, GO - Julho/2002 – Utilização de Operações Morfológicas na Eliminação de Ruído e Realce de Bordas em Imagens de Raios-X - http://www.cbpf.br/cat/download/sbpc/Morph.pdf

BIBLIOGRAFIA

[Astola, 1994] Astola, J., Cheikh-Alaya, F., Gabbouj M., "When are two Weighted Order Statistic Filter Identical", Proc. Of SPIE, vol. 2180, pp. 45-54, May 1994.

[Arce, 1996] Arce, Gonzalo R. and Tian, Mu, "Order Statistic Filter Banks", IEEE Transactions on Image Processing, vol. 5, no. 6, pp. 827-837, June 1996.

[Banham, 1997] Banham, Mark R. and Katsaggelos, Aggelos k., "Digital Image Restoration", IEEE Signal Processing Magazine, pp. 24-41, March 1997.

[Barner, 1994] Barner, Kenneth E. and Arce, Gonzalo R., "Permutation Filters: A Class of Nonlinear Filters Based on Set Permutations", IEEE Transactions on Signal Processing, vol. 42, no. 4, pp. 782-798, April 1994.

[Barner, 1997] Barner, Kenneth E. and Arce, Gonzalo R., "Design of Permutation Order Statistic Filters Through Group Colorings", IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing, vol. 44, no. 7, pp. 531-547, July 1997.

[Blume, 1999] Blume, Holger, "Nonlinear Vector Error Tolerant Interpolation of Intermediate Video Images by Weighted Medians", Signal Processing: Image Communication, no. 14, pp. 851-868, 1999.

[Chakrabarti, 1997] Chakrabarti, Chaitali and Lucke, E. Lori, "VLSI Architectures for Weighted Order Statistic (WOS) Filters", Signal Processing, vol 80, pp. 1419-1433, July 1997.

[Davila, 1994] Davila, Carlos E., "An Algorithm For Efficient, Unbiased, Equation-Error Infinite Impulse Response Adaptive Filtering", IEEE Transactions on Signal Processing, vol. 42, no. 2, pp. 415-418, February, 1994.

[Flaing, 1998] Flaing, Alexander, Arce, Gonzalo R. and Barner, Kenneth E., "Affine Order-Statistic Filters: Medianization of Linear FIR Filters", IEEE Transactions on Signal Processing, vol. 46, no. 8, pp. 2101-2112, August 1998.

[Gonzalez, 1992] Gonzalez, Rafael C, Woods Richard, "Digital Image Processing", 1992.

[Gonzalez, 2000] Gonzalez, Rafael C, Woods Richard, "Processamento de Imagens Digitais", 2000.

[Guillon, 1998] Guillon, Sébastien, Baylou, Pierre, Najim, Mohamed, "Adaptive nonlinear filters for 2D and 3D image enhancement", Signal Processing, vol. 67, pp. 237-254, February 1998.

[Hakami, 1994] Hakami, M. Reza, Warter, Peter J. and Boncelet, Charles G. Jr., "A New VLSI Architeture Suitable for Multidimensional Order Statistic Filtering", IEEE Transactions on Signal Processing, vol. 42, no. 4, pp. 991-993, April 1994.

[Hardie, 1994] Hardie, Russell C. and Barner, Kenneth E., "Rank Conditioned Rank Selection Filters for Signal Restoration", IEEE Transactions on Image Processing, vol. 3, no. 2, pp.192-206, March 1994.

[Hardie, 1996] Hardie, Russell C. and Barner, Kenneth E., "Extended Permutation Filters and Their Application to Edge Enhancement", IEEE Transactions on Image Processing, vol. 5, no. 6, pp.855-867, June 1996.

[Himayat, 1994] Himayat, Nageen and Kassam, Sallem A., "A Structure for Adaptive Order Statistics Filtering", IEEE Transactions on Image Processing, vol. 3, no. 3, pp. 265-280, May 1994.

[Ko, 1991] Ko Sung-Jea and Lee, Yong Hoon, "Center Weighted Median Filters and Their Applications to Image Enhancement", IEEE Transactions on Circuits and Systems, vol. 38, no. 9, pp. 984-993, September 1991.

[Lin, 1996] Lin, Ching C., Kuo, Chung J., "A Note on 'On the VLSI Implementation of Real-Time Order Statistic Filters'", IEEE Transactions on Signal Processing, vol. 44, no. 5, pp. 1314-1315, May 1996.

[Lopes, 1996] Lopes, Eymar, Namikawa, Laércio Massaru, Bruno, Regina Lúcia de Souza. SPRING: Tutorial de Geoprocessamento. Divisão de Processamento de Imagens, Outubro 1996. http://www.dpi.inpe.br/spring/teoria/.

[Muneyasu, 1999] Muneyasu, Mitsuji, Nishi, Nobutaka and Hinamoto, Takao, "A New Adaptive Center Weighted Median Filter Using Counter Propagation Networks", Jornal of the Franklin Institute, no. 337, pp. 631-639, August 1999.

[Neuvo, 1991] Neuvo, Yrjö, Yli-Harja, Olli and Astola, Jaakko, "Analysis of the Properties of Median and Weighted Median Filters Using Threshold Logic and Stack Filter Representation", IEEE Transactions on Signal Processing, vol. 39, no. 2, pp. 395-409, February 1991.

[Nuevo, 1994] Neuvo, Yrjö and Yin, Lin, "Fast Adaptation and Performance Characteristics of FIR-WOS Hybrid Filters", IEEE Transactions on Signal Processing, vol. 4, no. 7, pp. 1610-1628, July 1994.

[Oppenheim, 1975] Oppenheim, A.; Schafer R., "Digital Signal Processing", Prentice Hall, International Editions, New Jersey, USA 1975.

[Paez, 1994] Paez-Borralho, José M., Zazo-Bello, Santiago, "On the Joint Statistical Characterization of the Input and Output of an Order Statistic Filter", IEEE Transactions on Signal Processing, vol.42, no. 2, February 1994.

[Pérez, 2000] Pérez, Wilson Lobaiana, "Uma Análise dos Filtros Condicionais Seletores de Posição e sua Aplicação na Restauração de Imagens Digitais", 2000. Dissertação (Mestrado em Engenharia Elétrica) - Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas – SP.

[Pitas, 1991] Pitas, I. and Venetsanopoulos, A. N., "Adaptive Filters Based on Order Statistics", IEEE Transactions on Signal Processing, vol. 39, no. 2, pp. 518-522, February 1991.

[Pratt, 1978] Pratt W. K., "Digital Image Processing", John Wiley & Sons, USA 1978.

[Roopert, 1996] Roopert, Michael, Saint-Martin, François Moreau and Pelé, Danielle, "A new Representation of Weighted Order Statistic Filters", Signal Processing, vol 54, Issue 6, pp. 201-206, September 1996.

[Scuri, 1999] Scuri, Antonio Escaño. Fundamentos da Imagem Digital. Tecgraf/PUC-Rio, Mestrado em Informática, Outubro 1999. (http://www.tecgraf.puc-rio.br/~scuri/pub/fid.pdf)

[Yu, 1994] Yu, Pao-Ta and Liao, Wei-Hsiang, "Weighted Order Statistics Filters – Their Classification Some Properties, and Conversion Algorithm", IEEE Transactions on Signal Processing, vol. 2, no. 10, pp. 2678-2691, October 1994.

ANEXO 1

EXEMPLO NUMÉRICO DO PROBLEMA DE BORDAS

A seguir ilustraremos o problema de bordas através de um exemplo numérico, para uma melhor compreensão.

Quando utilizamos janelas de observação no processamento de imagens alguns pixels (os pertencentes as bordas da imagem) não vão ter a mesma quantidade de vizinhos que os demais. Isto é um problema, principalmente se considerarmos que o algoritmo vai tratar todos os pixels de maneira igual. A Figura A1.1 mostra uma imagem qualquer que será utilizada para melhor explicar as possíveis soluções para esse problema.

Figura A1.1 Imagem utilizada.

A matriz abaixo foi retirada do centro da imagem acima, a partir de agora vamos considerá-la como sendo a imagem a ser tratada.

53	55	51	53	45	43
55	47	51	48	46	44
47	43	54	49	50	41
43	47	61	60	45	40
50	60	79	79	48	38
50	57	94	93	76	47

Figura A1.2 Pixels da imagem observada.

A máscara utilizada neste caso será 3x3, por ser mais fácil a representação.

Como já foi dito temos várias maneiras de resolver o problema dos pixels da borda da figura, ou seja, aqueles que não possuem o mesmo número de vizinhos.

Uma solução seria tratar os pixels da borda de maneira diferente. Inicialmente determina-se quantos pixels serão observados, no caso de uma janela 3x3, cada pixel terá 8 vizinhos. Faz-se o laço (loop) de processamento com uma decisão dentro se o pixel observado pertence a alguma das bordas, se pertence significa que o mesmo não vai ter o mesmo número de vizinhos dos outros pixels. Neste caso teremos duas possibilidades: o pixel vai ter 3 vizinhos (os 4 pixels dos cantos) ou 5 vizinhos (demais pixels da borda).

53	55	51	53	45	43
55	47	51	48	46	44
47	43	54	49	50	41
43	47	61	60	45	40
50	60	79	79	48	38
50	57	94	93	76	47

Figura A1.3 Pixels de acordo com vizinhança

Observando a Figura A1.3, notamos que os pixels cinza escuro têm 3 vizinhos, os pixels cinza claro 5 vizinhos e os brancos 8 vizinhos.

Quando analisamos os filtros, principalmente aqueles que vão utilizar média aritmética ponderada, podemos ver claramente que esta é uma forma bem coerente de proceder e eliminar o problema.

Agora detalharemos o caso em que a imagem é aumentada, processada e logo após despreza-se as linhas e colunas acrescidas. Que teve um desempenho melhor, nas simulações feitas.

No caso de uma máscara genérica NxN, o número de linhas e colunas acrescidas seria (N-1)/2, devemos lembrar sempre que N vai ser sempre um número ímpar. Retornando ao nosso caso onde temos uma máscara 3x3 (N = 3), teríamos que aumentar 1 linha e 1 coluna. A partir deste ponto temos três formas de resolver o problema como mostra os itens a seguir.

1. Zeros fora da imagem o que ocorre neste caso é que, acrescentamos zeros fora da imagem.

0	0	0	0	0	0	0	0
0	53	55	51	53	45	43	0
0	55	47	51	48	46	44	0
0	47	43	54	49	50	41	0
0	43	47	61	60	45	40	0
0	50	60	79	79	48	38	0
0	50	57	94	93	76	47	0
0	0	0	0	0	0	0	0

Figura A1.4 Exemplo, zeros fora da imagem.

A Figura A1.4 nos mostra pixels brancos que são na realidade a imagem e pixels cinza são os pixels de valor zero acrescidos. Agora o processamento pode ser feito normalmente, sem se preocupar com os pixels da borda. Quando aplicamos esta solução para filtros que tem como saída, a média aritmética ponderada dos pixels de entrada, notamos que esta solução não é nada viável, as bordas da imagem ficam irreconhecíveis.

2. Repetir linhas e colunas, o que ocorre neste caso é a duplicação das primeiras e últimas, linhas e colunas da imagem.

53	53	55	51	53	45	43	43
53	53	55	51	53	45	43	43
55	55	47	51	48	46	44	44
47	47	43	54	49	50	41	41
43	43	47	61	60	45	40	40
50	50	60	79	79	48	38	38
50	50	57	94	93	76	47	47
50	50	57	94	93	76	47	47

Figura A1.5 Exemplo, repetir linhas e colunas.

A Figura A1.5 nos mostra pixels brancos que são na realidade a imagem e pixels cinza são os pixels duplicados. Agora o processamento pode ser feito normalmente, sem se preocupar com os pixels da borda.

3. Repetir linhas e colunas como se a imagem fosse circular repetir linhas e colunas agora duplicaremos linhas e colunas imagem, só que desta vez vamos inverter a posição. É como se a imagem fosse repetida de forma circular.

47	50	57	94	93	76	47	50
43	53	55	51	53	45	43	53
44	55	47	51	48	46	44	55
41	47	43	54	49	50	41	47
40	43	47	61	60	45	40	43
38	50	60	79	79	48	38	50
47	50	57	94	93	76	47	50
43	53	55	51	53	45	43	53

Figura A1.6 Exemplo, repetir linhas e colunas circular.

A Figura A1.6 nos mostra pixels brancos que são na realidade a imagem e pixels cinza são os pixels duplicados. Após esse pequeno ajuste o processamento pode ser feito normalmente, sem se preocupar com os pixels da borda.

ANEXO 2

FUNÇÃO IMNOISE DO MATLAB

IMNOISE - é uma função do Matlab 6.0.0.88, Release 12, 22 de Setembro de 2000, que insere um ruído a uma imagem dada.

ImagemRuido=imnoise(ImagemOriginal,Tipo, ...)

O Tipo é que determinará os outros valores a serem acrescentados na função acima. Podemos ter cinco tipos de ruído, que são descritos a seguir, somente utilizaremos as duas primeiras.

- 'gaussian' - insere ruído Gaussiano branco, necessita de dois parâmetros adicionais, a média (M) e a variância(V). Temos então, neste caso, ImagemRuido = imnoise(ImagemOriginal,'gaussian',M,V). Quando não informamos média e/ou variância a função assume os valores 0 e 0.01 respectivamente.

- 'salt & pepper' - insere ruído do tipo sal e pimenta, necessita de um par parâmetro adicional que é a densidade (D). Temos então, neste caso

ImagemRuido = imnoise(ImagemOriginal,'salt & pepper',D). Quando não informamos a densidade a função assume o valor 0.05.

- 'speckle' - insere ruído multiplicativo, necessita de um par parâmetro adicional que é a variância (V). Temos então, neste caso ImagemRuido = imnoise(ImagemOriginal,'speckle',V). Quando não informamos a densidade a função assume o valor 0.04.

- 'localvar' – insere ruído Gaussian branco de média zero com intensidade dependente da variância . Temos então, neste caso ImagemRuido = imnoise(ImagemOriginal,'localvar',V). Quando não informamos a densidade a função assume o valor 0.01.

- 'poisson' - insere ruído de Poisson. Temos então, neste caso ImagemRuido = imnoise(ImagemOriginal,'poisson').

Exemplo

```
% leitura de uma imagem qualquer
        I = imread('eight.tif');
% insere ruído a imagem lida
        J = imnoise(I,'salt & pepper', 0.02);
% mostra na tela a imagem original e a corrompida.
         imshow(I), figure, imshow(J)
```

Abaixo temos a função IMNOISE.

```
function b = imnoise(varargin)
%IMNOISE Add noise to image.
%   J = IMNOISE(I,TYPE,...) Add noise of a given TYPE to the intensity image
%   I. TYPE is a string that can have one of these values:
%
%       'gaussian'      Gaussian white noise with constant
%                       mean and variance
%
%       'localvar'      Zero-mean Gaussian white noise
%                       with an intensity-dependent variance
%
%       'poisson'       Poisson noise
%
%       'salt & pepper'  "On and Off" pixels
%
%       'speckle'       Multiplicative noise
```

```
%
%   Depending on TYPE, you can specify additional parameters to IMNOISE. All
%   numerical parameters are normalized; they correspond to operations with
%   images with intensities ranging from 0 to 1.
%
%   J = IMNOISE(I,'gaussian',M,V) adds Gaussian white noise of mean M and
%   variance V to the image I. When unspecified, M and V default to 0 and
%   0.01 respectively.
%
%   J = imnoise(I,'localvar',V) adds zero-mean, Gaussian white noise of
%   local variance, V, to the image I.  V is an array of the same size as I.
%
%   J = imnoise(I,'localvar',IMAGE_INTENSITY,VAR) adds zero-mean, Gaussian
%   noise to an image, I, where the local variance of the noise is a
%   function of the image intensity values in I.  IMAGE_INTENSITY and VAR
%   are vectors of the same size, and PLOT(IMAGE_INTENSITY,VAR) plots the
%   functional relationship between noise variance and image intensity.
%   IMAGE_INTENSITY must contain normalized intensity values ranging from 0
%   to 1.
%
%   J = IMNOISE(I,'poisson') generates Poisson noise from the data instead
%   of adding artificial noise to the data. In order to respect Poisson
%   statistics, the intensities of uint8 and uint16 images must correspond
%   to the number of photons (or any other quanta of information).
%   Double-precision images are used when the number of photons per pixel
%   can be much larger than 65535 (but less than 10^12); the intensities
%   values vary between 0 and 1 and correspond to the number of photons
%   divided by 10^12.
%
%   J = IMNOISE(I,'salt & pepper',D) adds "salt and pepper" noise to the
%   image I, where D is the noise density.  This affects approximately
%   D*PROD(SIZE(I)) pixels. The default for D is 0.05.
%
%   J = IMNOISE(I,'speckle',V) adds multiplicative noise to the image I,
%   using the equation J = I + n*I, where n is uniformly distributed random
%   noise with mean 0 and variance V. The default for V is 0.04.
%
%   Class Support
%   -------------
%   I can be of class uint8, uint16, or double.  The output image J is of
%   the same class as I.  If I has more than two dimensions it is treated
%   as a multidimensional intensity image and not as an RGB image.
%
%   Example
%   -------
%       I = imread('eight.tif');
%       J = imnoise(I,'salt & pepper', 0.02);
```

```
%       imshow(I), figure, imshow(J)
%
%   See also IMNSTATS, RAND, RANDN.

%   Copyright 1993-2001 The MathWorks, Inc.
%   $Revision: 5.18 $  $Date: 2001/03/21 20:07:37 $

[a, code, classIn, classChanged, p3, p4, p5, msg] = ParseInputs(varargin{:});
if (~isempty(msg))
   error(msg);
end

clear varargin;
sizeA = size(a);

switch code
  case 'gaussian' % Gaussian white noise
    b = a + sqrt(p4)*randn(sizeA) + p3;

  case 'localvar_1' % Gaussian white noise with variance varying locally
    % imnoise(a,'localvar',v)
    % v is local variance array
       b = a + sqrt(p3).*randn(sizeA); % Use a local variance array

  case 'localvar_2' % Gaussian white noise with variance varying locally
    % Use an empirical intensity-variance relation
    intensity = p3(:); % Use an empirical intensity-variance relation
    var       = p4(:);
    minI  = min(intensity);
    maxI  = max(intensity);
    b     = min(max(a,minI),maxI);
    b     = reshape(interp1q(intensity,var,b(:)),sizeA);
    b     = a + sqrt(b).*randn(sizeA);

  case 'poisson' % Poisson noise
    switch classIn
      case 'uint8'
        a = round(a*255);
      case 'uint16'
        a = round(a*65535);
      case 'double'
        a = round(a*10^12); % Recalibration
    end

    a = a(:);

    %  (Monte-Carlo Rejection Method) Ref. Numerical
```

```
    %  Recipes in C, 2nd Edition, Press, Teukolsky,
    %  Vetterling, Flannery (Cambridge Press)

    b=zeros(size(a));
    idx1=find(a<50); % Cases where pixel intensities are less than 50 units
    if (~isempty(idx1))
       g=exp(-a(idx1));
       em=-ones(size(g));
       t=ones(size(g));
       idx2=[1:length(idx1)]';
       while ~isempty(idx2)
          em(idx2)=em(idx2)+1;
          t(idx2)=t(idx2).*rand(size(idx2));
          idx2=idx2(find(t(idx2)>g(idx2)));
       end
       b(idx1)=em;
    end

    % For large pixel intensities the Poisson pdf becomes
    % very similar to a Gaussian pdf of mean and of variance
    % equal to the local pixel intensities. Ref. Mathematical Methods
    % of Physics, 2nd Edition, Mathews, Walker (Addison Wesley)
    idx1=find(a>=50); % Cases where pixel intensities are more than 49 units
    if (~isempty(idx1))
       b(idx1)=round(a(idx1)+sqrt(a(idx1)).*randn(size(idx1)));
    end

    a = reshape(a,sizeA);

  case 'salt & pepper' % Salt & pepper noise
    b = a;
    x = rand(sizeA);
    d = find(x < p3/2);
    b(d) = 0; % Minimum value
    d = find(x >= p3/2 & x < p3);
    b(d) = 1; % Maximum (saturated) value

  case 'speckle' % Speckle (multiplicative) noise
    b = a + sqrt(12*p3)*a.*(rand(sizeA)-.5);

end

% Truncate the output array data if necessary
if strcmp(code,{'poisson'})
    switch classIn
      case 'uint8'
        b = uint8(b);
```

```
      case 'uint16'
        b = uint16(b);
      case 'double'
        b = min(b/10^12,1);
    end
else
    b = max(0,min(b,1));
    % The output class should be the same as the input class
    if classChanged,
        b = changeclass(classIn, b);
    end
end


%%%
%%% ParseInputs
%%%
function [a, code, classIn, classChanged, p3, p4, p5, msg] = ParseInputs(varargin)

% Initialization
a            = [];
code         = 'gaussian';
classIn      = '';
classChanged = [];
p3           = [];
p4           = [];
p5           = [];
msg = '';

% Check the number of input arguments.
if nargin < 1
    msg = 'Too few inputs.';
    return;
end

% Check the input-array type.
a = varargin{1};
if ~(isa(a,'uint8') | isa(a,'uint16') | isa(a,'double'))
    msg = 'Invalid input array class: must be uint8, uint16 or double.';
    return;
else
    % Change class to double
    classIn = class(a);
    classChanged = 0;
    if ~isa(a, 'double')
        a = im2double(a);
        classChanged = 1;
```

```
    end
end

% Check for valid image I
if ~(isNonnegativeReal(a(:)) & all(a(:)<=1))
    msg = 'A double-class image must have values between zero and one.';
    return;
end

% Check the noise type.
if nargin > 1
    if ~ischar(varargin{2})
        msg = 'Invalid noise type: must be a character string.';
        return;
    end

    % Preprocess noise type string to detect abbreviations.
    allStrings = {'gaussian', 'salt & pepper', 'speckle',...
                  'poisson','localvar'};
    idx = strmatch(lower(varargin{2}), allStrings);
    switch length(idx)
      case 0
        msg = sprintf('Unknown noise type: "%s".', varargin{2});
        return;
      case 1
        code = allStrings{idx};
      otherwise
        msg = sprintf('Ambiguous noise type: "%s".', varargin{2});
        return;
    end
else
    code = 'gaussian';  % default noise type
end

switch code
  case 'poisson'
    if nargin > 2
        msg = 'Too many inputs for "poisson" noise.';
        return;
    end

  case 'gaussian'
    p3 = 0;     % default mean
    p4 = 0.01;  % default variance

    if nargin > 2
        p3 = varargin{3};
```

```
        if ~isRealScalar(p3)
            msg = 'For "gaussian" noise, M must be a real scalar.';
            return;
        end
    end

    if nargin > 3
        p4 = varargin{4};
        if ~isNonnegativeRealScalar(p4)
            msg = 'For "gaussian" noise, V must be a real nonnegative scalar.';
            return;
        end
    end

    if nargin > 4
        msg = 'Too many inputs for "gaussian" noise.';
        return;
    end

  case 'salt & pepper'
    p3 = 0.05;   % default density

    if nargin > 2
        p3 = varargin{3};
        if ~isNonnegativeRealScalar(p3) | (p3 > 1)
            msg = 'For "salt & pepper" noise, D must be a real nonnegative scalar less than or
equal to 1.';
            return;
        end

        if nargin > 3
            msg = 'Too many inputs for "salt & pepper" noise.';
            return;
        end
    end

  case 'speckle'
    p3 = 0.05;    % default variance

    if nargin > 2
        p3 = varargin{3};

        if ~isNonnegativeRealScalar(p3)
            msg = 'For "speckle" noise, V must be a nonnegative real scalar.';
            return;
        end
    end
```

```
   if nargin > 3
      msg = 'Too many inputs for "speckle" noise.';
      return;
   end

  case 'localvar'
   if nargin < 3
      msg = 'Too few inputs for "localvar" noise.';
      return;

   elseif nargin == 3
      % IMNOISE(a,'localvar',v)
      code = 'localvar_1';
      p3 = varargin{3};
      if ~isNonnegativeReal(p3) | ~isequal(size(p3),size(a))
         msg = sprintf('%s\n%s.', 'For the IMNOISE(A,"localvar",V) syntax, ', ...
                  'V must contain only nonnegative real values and be the same size as
A.');
         return;
      end

   elseif nargin == 4
      % IMNOISE(a,'localvar',IMAGE_INTENSITY,NOISE_VARIANCE)
      code = 'localvar_2';
      p3 = varargin{3};
      p4 = varargin{4};

      if ~isNonnegativeRealVector(p3) | (any(p3) > 1)
         msg = 'For "localvar" noise, IMAGE_INTENSITY must be a nonnegative real
vector less than or equal to 1.';
         return;
      end

      if ~isNonnegativeRealVector(p4)
         msg = 'For "localvar" noise, NOISE_VARIANCE must be a nonnegative real
vector.';
         return;
      end

      if ~isequal(size(p3),size(p4))
         msg = sprintf('%s\n%s.', 'For the
IMNOISE(A,"localvar",IMAGE_INTENSITY,NOISE_VARIANCE) syntax, ', ...
                  'IMAGE_INTENSITY and NOISE_VARIANCE must be the same
size.');
         return;
      end
```

```
   else
      msg = 'Too many inputs for "localvar" noise.';
      return;
   end

end

%%%
%%% isReal
%%%
function t = isReal(P)
%   isReal(P) returns 1 if P contains only real
%   numbers and returns 0 otherwise.
%
isFinite  = all(isfinite(P(:)));
t         = isreal(P) & isFinite & ~isempty(P);


%%%
%%% isNonnegativeReal
%%%
function t = isNonnegativeReal(P)
%   isNonnegativeReal(P) returns 1 if P contains only real
%   numbers greater than or equal to 0 and returns 0 otherwise.
%
t = isReal(P) & all(P(:)>=0);


%%%
%%% isRealScalar
%%%
function t = isRealScalar(P)
%   isRealScalar(P) returns 1 if P is a real,
%   scalar number and returns 0 otherwise.
%
t = isReal(P) & (prod(size(P))==1);


%%%
%%% isNonnegativeRealScalar
%%%
function t = isNonnegativeRealScalar(P)
%   isNonnegativeRealScalar(P) returns 1 if P is a real,
%   scalar number greater than 0 and returns 0 otherwise.
%
t = isReal(P) & all(P(:)>=0) & (prod(size(P))==1);
```

```
%%%
%%% isVector
%%%
function t = isVector(P)
%   isVector(P) returns 1 if P is a vector and returns 0 otherwise.
%
t = ((prod(size(P)) >= 2) & ((size(P,1) == 1) | (size(P,2) == 1)));


%%%
%%% isRealVector
%%%
function t = isRealVector(P)
%   isRealVector(P) returns 1 if P is a real,
%   vector and returns 0 otherwise.
%
t = isReal(P) & isVector(P);


%%%
%%% isNonnegativeRealVector
%%%
function t = isNonnegativeRealVector(P)
%   isNonnegativeRealVector(P) returns 1 if P is a real,
%   vector greater than 0 and returns 0 otherwise.
%
t = isReal(P) & all(P(:)>=0) & isVector(P);
```

ANEXO 3

FUNÇÃO WIENER2 DO MATLAB

Esta é uma função do Matlab 6.0.0.88, Release 12, 22 de Setembro de 2000, que recupera uma imagem degradada por algum tipo de ruído. A chamada para a mesma é feita da seguinte forma: ImgRecWiener = wiener2(ImgRuido, Janela). Esta função requer dois parâmetros que são ImgRuido (imagem corrompida) e Janela (janela de observação). E nos retorna a imagem recuperada (ImgRecWiener).

```
% function [f,noise] = wiener2(varargin)
%       WIENER2 Perform 2-D adaptive noise-removal filtering.
%       WIENER2 lowpass filters an intensity image that has been degraded by constant
% power additive noise. WIENER2 uses a pixel-wise adaptive Wiener method based on
% statistics estimated from a local neighborhood of each pixel.
%       J = WIENER2(I,[M N],NOISE) filters the image I using pixel-wise adaptive
% Wiener filtering, using neighborhoods of size M-by-N to estimate the local image mean
% and standard deviation. If you omit the [M N] argument, M and N default to 3. The
% additive noise (Gaussian white noise) power is assumed to be NOISE.
%       [J,NOISE] = WIENER2(I,[M N]) also estimates the additive noise power before
% doing the filtering. WIENER2 returns this estimate as NOISE.
%       Os lowpass WIENER2 filtram uma imagem da intensidade que seja degradada pelo
% ruído do aditivo da potência constante. WIENER2 usa um método adaptável pixel-sábio
% do wiener baseado nos statistics estimados de uma vizinhança local de cada pixel.
```

```
%       [ J, ruído ] = WIENER2(I, %[m N ]) estima também a potência de ruído aditiva
% antes de fazer filtrar.
%       WIENER2 retorna esta estimativa como o RUÍDO.
% Class Support
%       The input image I can be of class uint8, uint16, or double.
%       The output image J is of the same class as I.

%       Example
%               I = imread('saturn.tif');
%               J = imnoise(I,'gaussian',0,0.005);
%               K = wiener2(J,[5 5]);
%               imshow(J), figure, imshow(K)

%       See also FILTER2, MEDFILT2.

%       The following syntax is grandfathered:

%       J = WIENER2(I,[M N],[MBLOCK NBLOCK],NOISE) or [J,NOISE] =
% WIENER2(I,[M N],[MBLOCK NBLOCK]) processes the intensity image I as above but
% in blocks of size MBLOCK-by-NBLOCK.     Use J = WIENER2(I,[M N],SIZE(I),NOISE) % to process the matrix all at once.
=================================================================
==
[g, nhood, block, noise, msg] = ParseInputs(varargin{:});
if (~isempty(msg))
            error(msg);
end
classin = class(g);
classChanged = 0;
if ~isa(g, 'double')
            classChanged = 1;
            g = im2double(g);
end

% Estimate the local mean of f.
localMean = filter2(ones(nhood), g) / prod(nhood);

% Estimate of the local variance of f.
localVar = filter2(ones(nhood), g.^2) / prod(nhood) - localMean.^2;

% Estimate the noise power if necessary.
if (isempty(noise))
            noise = mean2(localVar);
end
```

```
% Compute result
% f = localMean + (max(0, localVar - noise) ./ ... max(localVar, noise)) .* (g -
localMean);
% Computation is split up to minimize use of memory for temp arrays.
f = g - localMean;
g = localVar - noise;
g = max(g, 0);
localVar = max(localVar, noise);
f = f ./ localVar;
f = f .* g;
f = f + localMean;

if classChanged,
        f = changeclass(classin, f);
end

%%%
%%% Subfunction ParseInputs
%%%
function [g, nhood, block, noise, msg] = ParseInputs(varargin)

g = [];
nhood = [3 3];
block = [];
noise = [];
msg = '';

switch nargin
case 0
        msg = 'Too few input arguments';
        return;
case 1
   % wiener2(I)
        g = varargin{1};
case 2
        g = varargin{1};
        switch prod(size(varargin{2}))
case 1
           % wiener2(I,noise)
           noise = varargin{2};
case 2
           % wiener2(I,[m n])
           nhood = varargin{2};
otherwise
           msg = 'Invalid input syntax';
           return;
end
```

```
case 3
        g = varargin{1};
        if (prod(size(varargin{3})) == 2)
                % wiener2(I,[m n],[mblock nblock])  GRANDFATHERED
                nhood = varargin{2};
                block = varargin{3};
        else
                % wiener2(I,[m n],noise)
                nhood = varargin{2};
                noise = varargin{3};
         end
case 4
        % wiener2(I,[m n],[mblock nblock],noise)  GRANDFATHERED
        g = varargin{1};
        nhood = varargin{2};
        block = varargin{3};
        noise = varargin{4};
otherwise
        msg = 'Too many input arguments';
        return;
end

if (isempty(block))
        block = bestblk(size(g));
end
```

ANEXO 4

IMAGENS ORIGINAIS UTILIZADAS

Temos aqui várias imagens originais utilizadas para teste.

Figura A4.1 Imagem original lena.tiff

A Figura A4.1 nos mostra a imagem original lena.tiff, tamanho 512x512.

Figura A4.2 Imagem original montage.gif

A Figura A4.2 nos mostra a imagem original montage.gif, tamanho 256x256.

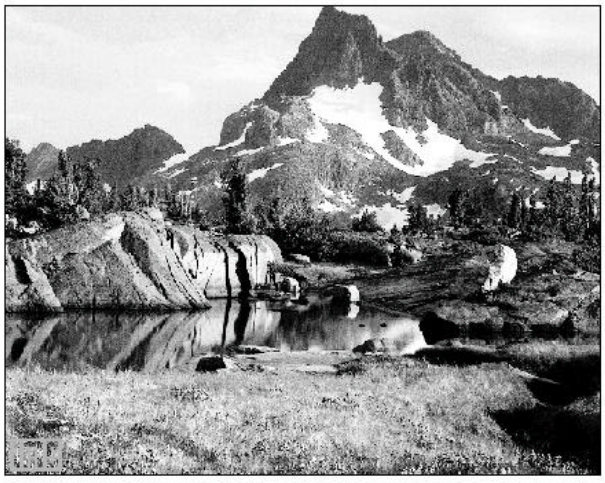


Figura A4.3 Imagem original mountain.gif

A Figura A4.3 nos mostra a imagem original mountain.gif, tamanho 480x640.

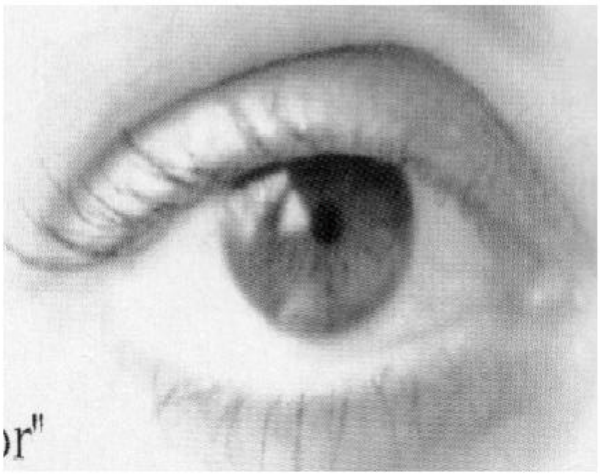


Figura A4.4 Imagem original olhodigital.tiff

A Figura A4.4 nos mostra a imagem original olhodigital.tiff, tamanho 273x442.

Figura A4.5 Imagem original flor.jpg

A Figura A4.5 nos mostra a imagem original flor.jpg, tamanho 256x250.

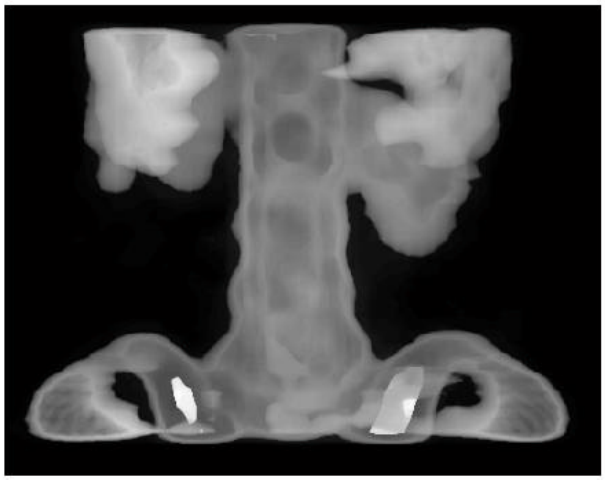


Figura A4.6 Imagem original spine.tiff

A Figura A4.6 nos mostra a imagem original spine.tiff, tamanho 367x490.

Figura A4.7 Imagem original tire.tiff

A Figura A4.7 nos mostra a imagem original tire.tiff, tamanho 205x232.

Figura A4.8 Imagem original boat.gif

A Figura A4.8 nos mostra a imagem original boat.gif, tamanho 512x512.

Figura A4.9 Imagem original cameraman.tiff

A Figura A4.9 nos mostra a imagem original cameraman.tiff, tamanho 256x256.

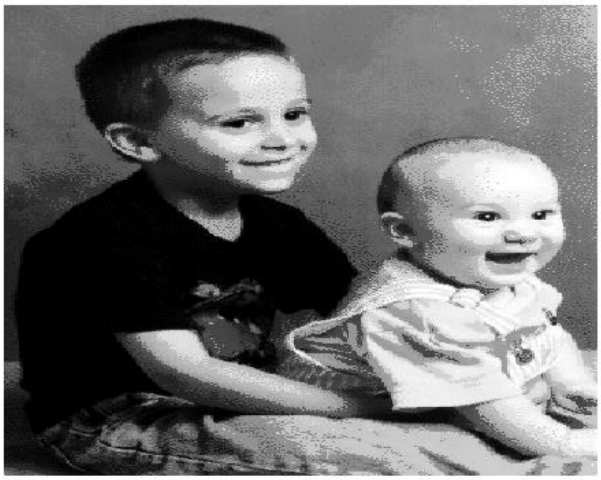


Figura A4.10 Imagem original kids.tiff

A Figura A4.10 nos mostra a imagem original kids.tiff, tamanho 400x318.

Figura A4.11 Imagem original trees.tiff

A Figura A4.11 nos mostra a imagem original trees.tiff, tamanho 258x350.

Figura A4.12 Imagem original couple.tiff

A Figura A4.12 nos mostra a imagem original couple.tiff, tamanho 494x512.

Figura A4.13 Imagem original moon.tiff

A Figura A4.13 nos mostra a imagem original moon.tiff, tamanho 537x358.

Figura A4.14 Imagem original elaine.tiff

A Figura A4.14 nos mostra a imagem original elaine.tiff, tamanho 512x512.

ANEXO 5

Sopt para o Filtro RCRS

A imagem utilizada na geração do Sopt foi a lena.tiff.

Figura A5.1 Imagem original lena.tiff

A Figura A5.1 nos mostra a imagem original lena.tiff, tamanho 512x512.

A Figura A5.2 nos mostra a função ótima (Sopt) gerada pelo filtro RCRS, sendo que a imagem foi corrompida por ruído do tipo sal e pimenta, densidade entre [0.05, 0.5], a janela de observação utilizada foi 3x3.

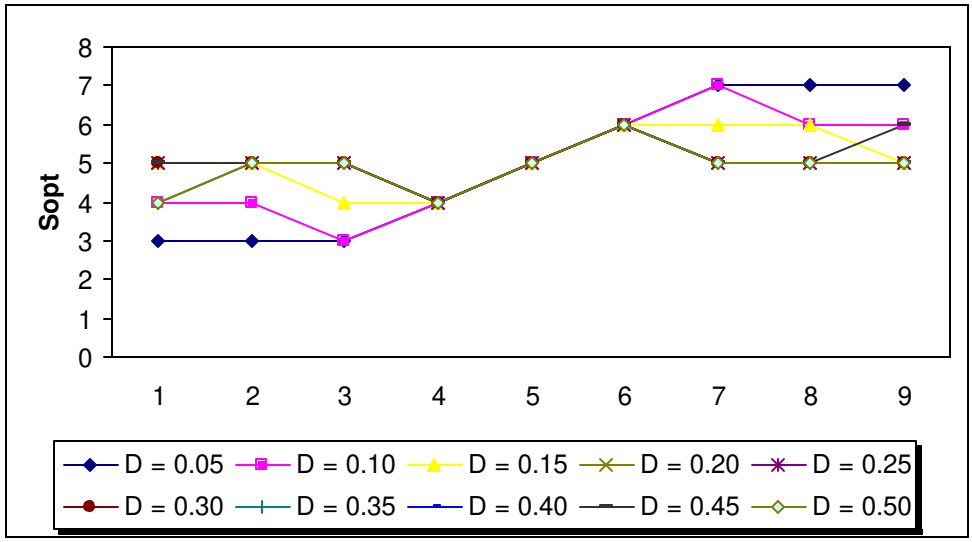


Figura A5.2 Sopt, ruído SalPimenta, 3x3

A Figura A5.3 nos mostra a função ótima (Sopt) gerada pelo filtro RCRS, sendo que a imagem foi corrompida por ruído do tipo sal e pimenta, densidade entre [0.05, 0.5], a janela de observação utilizada foi 5x5.

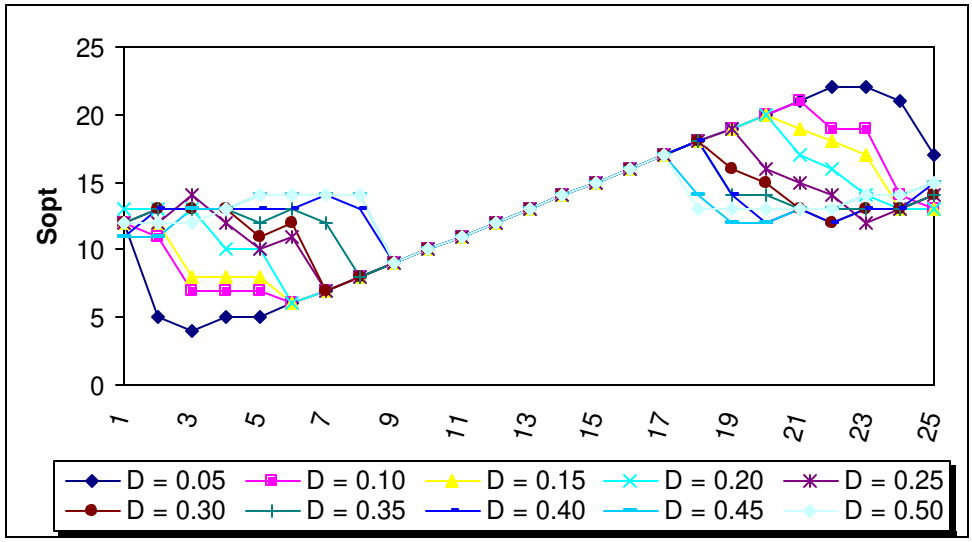


Figura A5.3 Sopt, ruído SalPimenta, 5x5

A Figura A5.4 nos mostra a função ótima (Sopt) gerada pelo filtro RCRS, sendo que a imagem foi corrompida por ruído do tipo sal e pimenta, densidade entre [0.05, 0.5], a janela de observação utilizada foi 7x7.

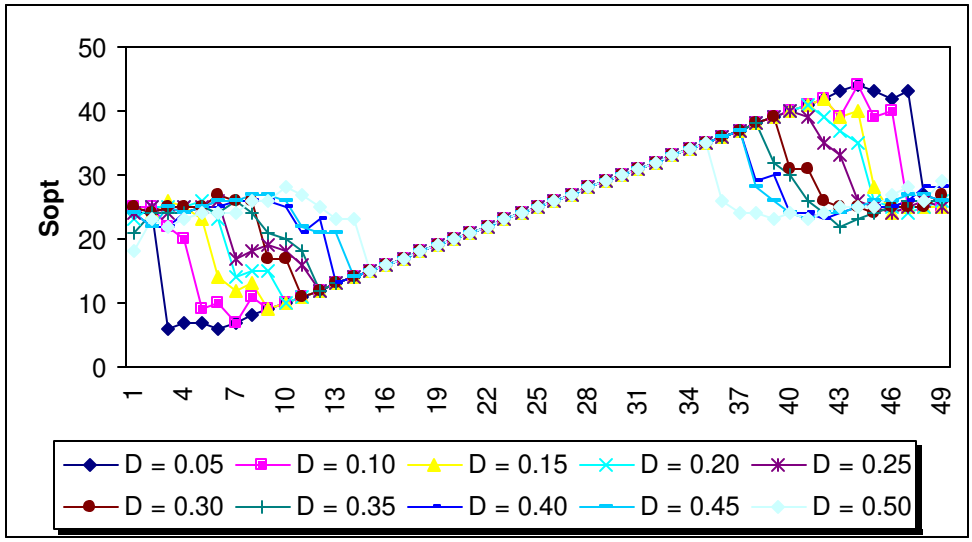


Figura A5.4 Sopt, ruído SalPimenta, 7x7

A Figura A5.5 nos mostra a função ótima (Sopt) gerada pelo filtro RCRS, sendo que a imagem foi corrompida por ruído do tipo sal e pimenta, densidade entre [0.05, 0.5], a janela de observação utilizada foi 9x9.

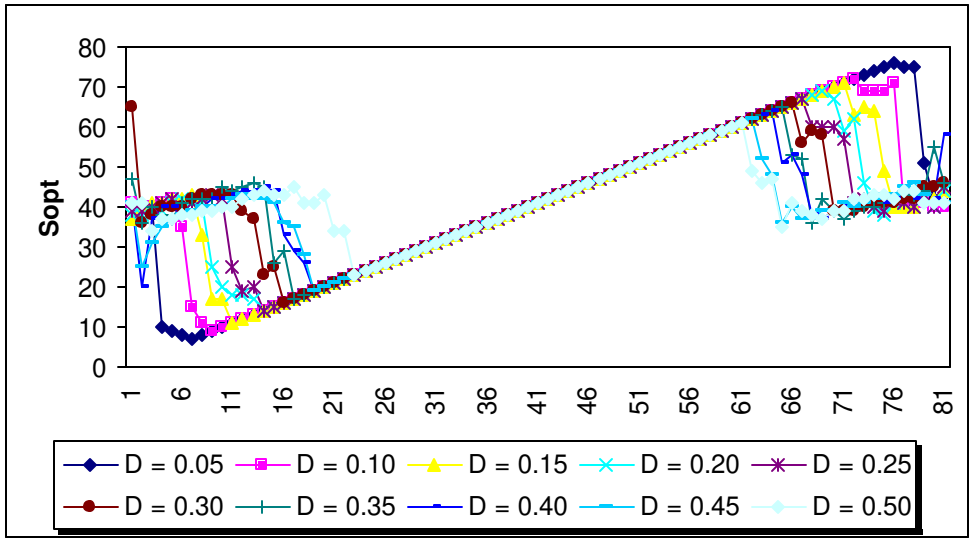


Figura A5.5 Sopt, ruído SalPimenta, 9x9

A Figura A5.6 nos mostra a função ótima (Sopt) gerada pelo filtro RCRS, sendo que a imagem foi corrompida por ruído do tipo gaussiano, média 0 e variância entre [0.05, 0.5], a janela de observação utilizada foi 3x3.

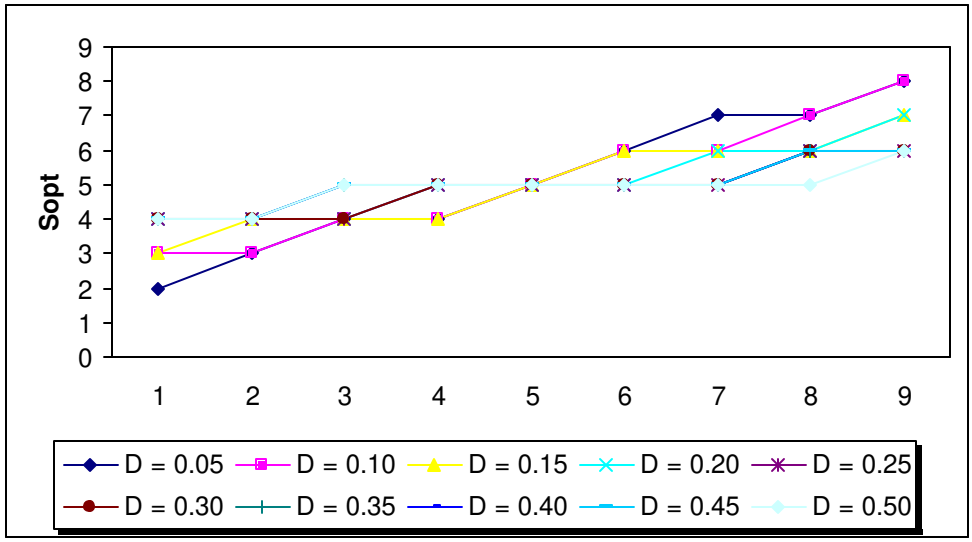


Figura A5.6 Sopt, ruído gaussiano, 3x3

A Figura A5.7 nos mostra a função ótima (Sopt) gerada pelo filtro RCRS, sendo que a imagem foi corrompida por ruído do tipo gaussiano, média 0 e variância entre [0.05, 0.5], a janela de observação utilizada foi 5x5.

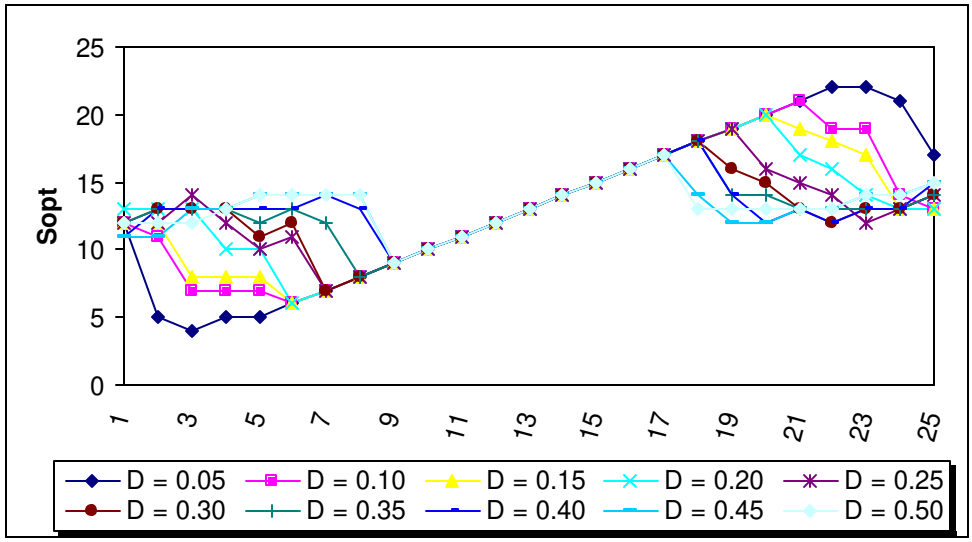


Figura A5.7 Sopt, ruído gaussiano, 5x5

A Figura A5.8 nos mostra a função ótima (Sopt) gerada pelo filtro RCRS, sendo que a imagem foi corrompida por ruído do tipo gaussiano, média 0 e variância entre [0.05, 0.5], a janela de observação utilizada foi 7x7.

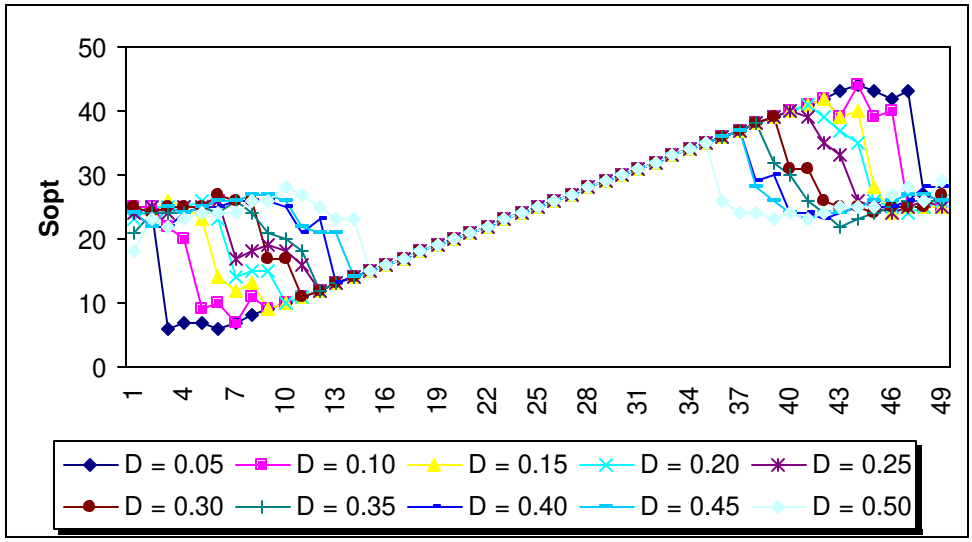


Figura A5.8 Sopt, ruído gaussiano, 7x7

A Figura A5.9 nos mostra a função ótima (Sopt) gerada pelo filtro RCRS, sendo que a imagem foi corrompida por ruído do tipo gaussiano, média 0 e variância entre [0.05, 0.5], a janela de observação utilizada foi 9x9.

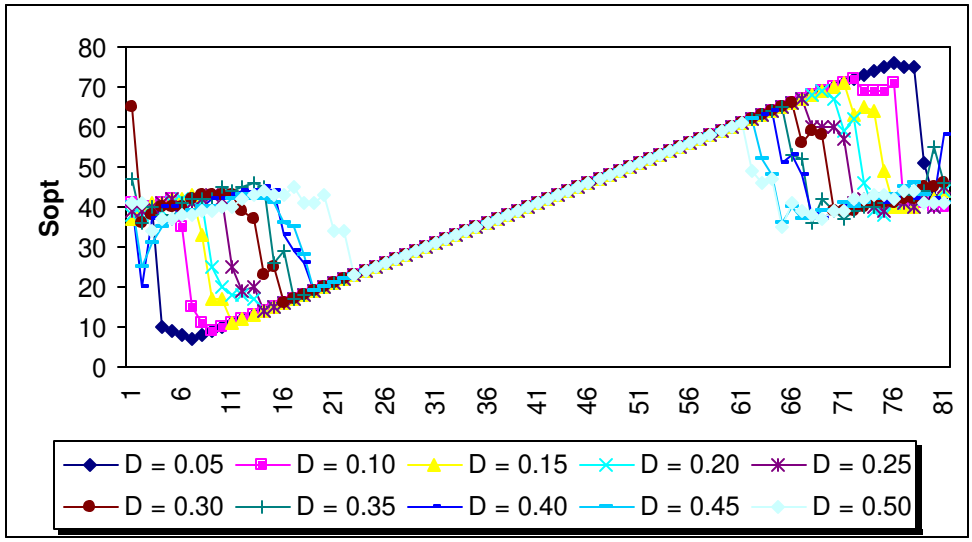


Figura A5.9 Sopt, ruído gaussiano, 9x9