

**UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA**

Sergio Genilson Pflieger

**REDUÇÃO DE RUÍDO EM VÍDEOS EM TEMPO REAL
BASEADO NA FUSÃO DO FILTRO DE KALMAN E
FILTRO BILATERAL**

Florianópolis

2016

Sergio Genilson Pflieger

**REDUÇÃO DE RUÍDO EM VÍDEOS EM TEMPO REAL
BASEADO NA FUSÃO DO FILTRO DE KALMAN E
FILTRO BILATERAL**

Dissertação submetida ao Programa de Pós-graduação em Ciência da Computação da Universidade Federal de Santa Catarina para a obtenção do Grau de Mestre em Ciência da Computação.

Orientadora: Prof^a. Dr^a. Patricia Della Mea Plentz

Florianópolis
2016

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Pfleger, Sergio Genilson

Redução de Ruído em Vídeos em Tempo Real Baseado na Fusão
do Filtro de Kalman e Filtro Bilateral / Sergio Genilson
Pfleger ; orientadora, Patricia Della Mea Plentz -
Florianópolis, SC, 2016.

90 p.

Dissertação (mestrado) - Universidade Federal de Santa
Catarina, Centro Tecnológico. Programa de Pós-Graduação em
Ciência da Computação.

Inclui referências

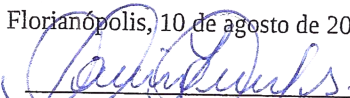
1. Ciência da Computação. 2. Redução de Ruído em Vídeos em
Tempo Real. I. Plentz, Patricia Della Mea. II.
Universidade Federal de Santa Catarina. Programa de Pós
Graduação em Ciência da Computação. III. Título.

Sergio Genilson Pflieger

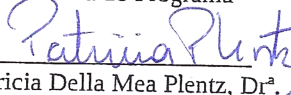
**REDUÇÃO DE RUÍDO EM VÍDEOS EM TEMPO-REAL
BASEADO NA FUSÃO DO FILTRO DE KALMAN E
FILTRRO BILATERAL**

Esta dissertação foi julgada adequada para obtenção do título de mestre e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Florianópolis, 10 de agosto de 2016.

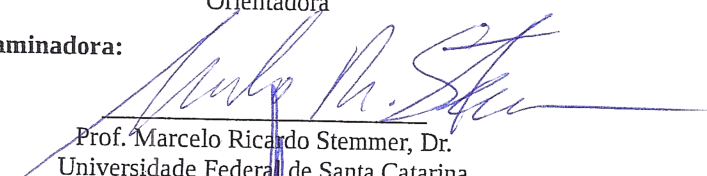


Prof. Carina Friedrich Dorneles, Dr.^a
Coordenadora do Programa

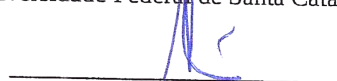


Prof.^a Patricia Della Mea Plentz, Dr.^a
Universidade Federal de Santa Catarina
Orientadora

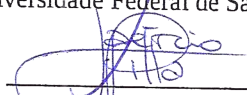
Banca Examinadora:



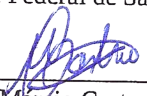
Prof. Marcelo Ricardo Stemmer, Dr.
Universidade Federal de Santa Catarina



Prof. Aldo von Wangenheim, Dr.
Universidade Federal de Santa Catarina



Prof. Laércio Pilla, Dr.
Universidade Federal de Santa Catarina



Prof. Marcio Castro, Dr.
Universidade Federal de Santa Catarina

Este trabalho é dedicado aos meus queridos pais.

AGRADECIMENTOS

À Profa. Dra. Patricia Della Mea Plentz, pela confiança, incentivo e oportunidade de trabalhar ao seu lado.

Ao Prof. Dr. Márcio Bastos Castro, pelo apoio no processo de amadurecimento do trabalho.

À Hylson Viscovi Netto, pela paciência em ouvir e pelos sábios conselhos.

Ao grande amigo Ronan Rasia, pelas cervejas e discussões ao longo do trabalho.

À João R. Rover, Ramon D. Miranda e Alexandre Israel Henckel, por me apoiarem nesta jornada.

Aos amigos e família, pela compreensão pelos momentos em que não pude me fazer presente.

À Maryah E. M. Haertel, pelo carinho, apoio e compreensão.

À CAPES pelo apoio financeiro na execução deste trabalho.

RESUMO

Neste trabalho é proposto um filtro para atenuação de ruído em vídeos, em tempo real, baseado na fusão de uma modificação do filtro de Kalman e do filtro bilateral, de forma a aproveitar características espaciais e temporais das imagens, preservando contornos e características essenciais à visão humana e computacional. O algoritmo proposto, chamado STMKF, mantém as características do filtro de Kalman original para regiões onde não há movimento e aplica o filtro bilateral nas regiões onde ocorre movimento, fazendo o filtro de Kalman convergir mais rápido para os novos valores adquiridos. Os resultados experimentais mostraram que o filtro é competitivo em relação aos demais, principalmente onde o fundo da imagem é estacionário. A avaliação de desempenho em CPUs e GPUs mostrou sua viabilidade em tempo real, com a filtragem de aproximadamente 30 frames FullHD por segundo em um Intel i7 e mais de 1000 FPS para um vídeo 480p em GPU.

Palavras-chave: Filtro de Kalman. Filtro de Kalman Modificado. Redução de Ruído. Ruído Gaussiano. Shot Noise. Video Denoising. Multicore Video Denoising. GPU Video Denoising

ABSTRACT

In this work is proposed a filter to minimize noise in videos, in real-time, based on the fusion of a modified Kalman Filter and a bilateral filter, taking advantage of spatial and temporal characteristics of the images, preserving contours and essential features for human and computer vision. The proposed algorithm, called STMKF, maintains the original Kalman filter characteristics in motionless regions and it applies the bilateral filter in regions with motion, which make the Kalman filter converge faster for the new acquired values. The experimental results show that the proposed filter is competitive in relation to others, mainly in videos with more static backgrounds. The performance evaluation in CPUs and GPUs shows that STMKF is viable in real time, filtering approximately 30 FullHD frames per second in a Intel i7 and over 1000 FPS of a 480p video on a GPU.

Keywords: Kalman Filter. Modified Kalman Filter. Noise reduction. Gaussian Noise. Shot Noise. Spatio-temporal noise reduction. Video Denoising. Multicore Video Denoising. GPU Video Denoising.

LISTA DE FIGURAS

Figura 1	Uma sequência de imagens formando um vídeo.....	26
Figura 2	Ruído Gaussiano.....	27
Figura 3	Imagem com 5% de Ruído do tipo Salt-and-Pepper.....	28
Figura 4	Ruído de quantização.	29
Figura 5	Ruído periódico.	30
Figura 6	<i>Blur</i> gaussiano aplicado sobre a imagem Lena.	31
Figura 7	À Esquerda imagem com ruído. Ao centro, resultado da difusão anisotrópica aplicada sobre a imagem. À direita, resultado dos métodos misto adaptativo.	33
Figura 8	Filtro Bilateral aplicado sobre uma imagem. A esquerda, imagem original. Ao centro, filtro Bilateral aplicado com $d = 6$, $\sigma_c = 100$ e $\sigma_s = 100$. A direita, filtro Bilateral aplicado com $d = 50$, $\sigma_c = 50$ e $\sigma_s = 50$	34
Figura 9	Filtro de Kalman aplicado (linha contínua) sobre um valor lido (linha tracejada e pontilhada) e sua comparação com o valor real (linha tracejada).	35
Figura 10	Filtro de Kalman aplicado a uma sequência de imagens adquiridas por uma câmera de baixo custo em um ambiente de pouca luz. No topo, figura adquirida. Na base, filtro de Kalman aplicado.	37
Figura 11	Filtro de Kalman aplicado (linha contínua) sobre um valor lido (linha tracejada e pontilhada) e sua comparação com o valor real (linha tracejada).	38
Figura 12	No topo, um frame da sequência <i>Garden</i> . Na base, o borramento ocorrido devido ao filtro de Kalman aplicado aos objetos em movimento da sequência.	39
Figura 13	Métricas de qualidade de imagens e suas diferenças em relação a qualidade percebida.....	40
Figura 14	Os três principais grupos de filtros de vídeo.	45
Figura 15	Blocos de referência, em vermelho, e blocos similares encontrados (<i>block-matching</i>).	47
Figura 16	Fragmentos de imagens com ruído gaussiano ($\sigma = 50$) e o resultado do filtro baseado em <i>block-matching</i>	48
Figura 17	Mapa de K_k do frame 20 da sequência Salesman.	52

Figura 18 PSNR para a sequência Salesman em termos de Q e R .	54
Figura 19 SSIM para a sequência Salesman em termos de Q e R .	54
Figura 20 Diagrama do algoritmo STMKF proposto.	56
Figura 21 Ilustração da abordagem paralela adotada em MT-STMKF. Este exemplo considera 4 <i>threads</i> , cada uma encarregada de filtrar 1/4 de cada frame.	58
Figura 22 Diferenças entre as versões com <i>Streams</i> e sem <i>Streams</i> do algoritmo para GPU	61
Figura 23 PSNR para cada um dos frames da sequência Salesman com ruído ($\sigma = 10$). PSNR Médio= 34,98	64
Figura 24 SSIM para cada um dos frames da sequência Salesman com ruído ($\sigma = 10$). SSIM Médio = 0.930	65
Figura 25 Resultados do STMKF para a sequência Salesman com ruído Gaussiano com desvio padrão $\sigma = 10$. Topo: frame original, frame com ruído e frame com filtro bilateral. Base: frame com o filtro de Kalman, frame com STMKF e o mapa de K_k	65
Figura 26 Comparação da detecção de bordas Canny. Topo: frame original, frame com ruído e frame filtrado pelo STMKF. Base: Canny do frame original, Canny do frame com ruído e Canny do frame filtrado pelo STMKF	66
Figura 27 Resultados em termos de PSNR comparados com outros trabalhos. Dados dos algoritmos STA, VBM3D, K-SVD, 3D-Pach, VBM3D, ST-GSM e DNLM foram extraídos de Han e Chen (2012)	68
Figura 28 Resultados em termos de SSIM comparados com outros trabalhos. Dados dos algoritmos WRSTF, SEQWT, 3DWTF, IFSM, 3DSWDCT, VBM3D, ST-GSM and DNLM foram extraídos de Han e Chen (2012)	69
Figura 29 Tempo necessário para processar um frame, para diversas resoluções	73
Figura 30 Escalabilidade do MT-STMKF versus as versões paralelas do filtro Bilateral e STMKF-CORE para um video com resolução 4K no processador Intel i7.....	74
Figura 31 <i>Speedup</i> do MT-STMKF em processadores multicore, variando o número de <i>threads</i> e a resolução dos vídeos.	75
Figura 32 FPS do MT-STMKF em processadores multicore, variando o número de <i>threads</i> e a resolução dos vídeos.	75
Figura 33 Desempenho do GPU-STMKF com (S) e sem (No-S) <i>CUDA Streams</i>	77

Figura 34 Distribuição dos algoritmos em função da velocidade e
qualidade..... 78

LISTA DE TABELAS

Tabela 1	Termos de busca e seus sinônimos de busca.	43
Tabela 2	Trabalhos remanescentes a cada fase da revisão sistemática.	44
Tabela 3	Resultados em PSNR comparados com outros algoritmos. Dados para 3D KNN, Pizurica e IFSM foram extraídos de Rahman, Ahmad e Swamy (2007).....	67
Tabela 4	Especificação das plataformas paralelas.	70
Tabela 5	Especificação das resoluções dos vídeos.....	71
Tabela 6	Tempo de processamento para diversos algoritmos, fornecido pelos respectivos autores, comparado com a versão <i>single thread</i> do STMKF.....	72
Tabela 7	Resultados em termos de PSNR comparados com outros trabalhos. Dados dos algoritmos STA, VBM3D, K-SVD, 3D-Pach, VBM3D, ST-GSM e DNLM foram extraídos de Han e Chen (2012)	89
Tabela 8	Resultados em termos de SSIM comparados com outros trabalhos. Dados dos algoritmos WRSTF, SEQWT, 3DWTF, IFSM, 3DSWDCT, VBM3D, ST-GSM and DNLM foram extraídos de Han e Chen (2012).....	90

LISTA DE ABREVIATURAS E SIGLAS

RGB	Red, Green and Blue.....	26
PSNR	<i>Peak Signal to Noise Ratio</i>	38
MSE	<i>Mean Square Error</i>	38
SSIM	<i>Structural Similarity</i>	39
FPS	Frames por Segundo.....	41
WRSTF	<i>Wavelet Domain Recursive Temporal Filtering</i>	46
BM3D	<i>Block-matching 3D Denoising</i>	47
VBM3D	<i>Video Block-matching 3D Denoising</i>	47
KNN	<i>K-Nearest Neighbour</i>	49
STVF	<i>Spatiotemporal Varying Filter</i>	49
STMKF	<i>Spatio-temporal Modified Kalman Filter</i>	51
OpenCV	<i>Open Source Computer Vision Library</i>	58
MT-STMKF	<i>Multi Thread STMKF</i>	58
API	<i>Application Programming Interface</i>	58
OpenMP	<i>Open Multi-Processing</i>	58
GPU	<i>Graphics Processing Unit</i>	59
GPU-STMKF	<i>Versão para GPU do STMKF</i>	59
CUDA	<i>Compute Unified Device Architecture</i>	59
OpenCL	<i>Open Computing Language</i>	59
CPU	<i>Central Processing Unit</i>	59
SO	Sistema Operacional.....	70
HT	<i>Hyper-Threading Technology</i>	70
RAM	<i>Random Access Memory</i>	71
FLOPS	<i>Floating-point Operations Per Second</i>	77

LISTA DE SÍMBOLOS

z_i	Valor observado de um píxel i	27
v_i	Valor real de um píxel i	27
r_i	Valor do ruído de um píxel i	27
d	Diâmetro dos píxeis utilizados no filtro Bilateral	34
σ_c	Fator de propagação fotométrica do filtro Bilateral	34
σ_s	Fator de propagação geométrica do filtro Bilateral	34
x_k^-	Estimativa do próximo estado	35
P_k^-	Estimativa da covariância do erro do próximo estado	35
K_k	Ganho de Kalman	35
z_k	Valor lido	35
x_k	Valor estimado	35
P_k	Covariância do erro	35
u_k	Sinal de controle	35
Q	Parâmetro do Filtro de Kalman	35
R	Parâmetro do Filtro de Kalman	35
Δ	Diferença entre a média dos vizinhos do píxel nos instantes k e $k - 1$	52
b_k	Frame do instante k , com <i>blur</i>	52
b_{k-1}	Frame do instante $k - 1$, com <i>blur</i>	52
x_b	Frame resultante do Filtro Bilateral	53
X_k	Frame resultante do STMKF	53
X_{k-1}	Frame resultante do STMKF, do instante $k - 1$	55

SUMÁRIO

1	INTRODUÇÃO	21
1.1	MOTIVAÇÃO	21
1.2	CONTEXTUALIZAÇÃO DO PROBLEMA	21
1.3	OBJETIVOS	23
1.3.1	Objetivo Geral	23
1.3.2	Objetivos Específicos	23
1.4	ORGANIZAÇÃO DO TEXTO	23
2	CONCEITOS E DEFINIÇÕES	25
2.1	IMAGEM DIGITAL	25
2.2	VÍDEO DIGITAL	26
2.3	RUÍDO EM IMAGENS	27
2.3.1	Ruído Gaussiano (<i>Gaussian Noise</i>):	27
2.3.2	<i>Salt-and-Pepper Noise</i>	28
2.3.3	Ruído de Disparo (<i>Shot Noise</i>)	28
2.3.4	Ruído de Quantização (<i>Quantization Noise</i>)	29
2.3.5	Ruídos Periódicos	29
2.4	FILTROS PARA RUÍDOS EM IMAGENS E VÍDEOS ...	30
2.4.1	Filtro de suavização linear (<i>Linear Smoothing</i>): <i>Blur</i>	31
2.4.2	Filtro Adaptativo (<i>Adaptive Filter</i>)	31
2.4.3	Filtro de Mediana (<i>Median Filter</i>)	32
2.4.4	Difusão Anisotrópica	32
2.4.5	<i>Non-Local Means</i>	32
2.4.6	Método Misto Adaptativo	33
2.5	FILTRO BILATERAL	33
2.6	FILTRO DE KALMAN	34
2.7	MÉTRICAS DE QUALIDADE EM IMAGEM	37
2.8	TEMPO REAL	41
3	TRABALHOS RELACIONADOS	43
3.1	DEFINIÇÃO DA REVISÃO DA LITERATURA	43
3.2	EXECUÇÃO DA BUSCA	44
3.3	TRABALHOS RELACIONADOS	44
3.3.1	Métodos Baseados em Transformadas	45
3.3.2	Métodos Baseados em Algoritmos de Detecção de Movimento e <i>Block Matching</i>	46
3.3.3	Métodos que Combinam Filtros nos Domínios Tem- porais e Espaciais	48

4	<i>SPATIO-TEMPORAL MODIFIED KALMAN FILTER - STMKF</i>	51
4.1	FILTRO PROPOSTO	51
4.1.1	Convergência Otimizada do Filtro de Kalman para Regiões com Movimento	51
4.1.2	Ponderação entre o Filtro de Kalman e o Filtro Bilateral	53
4.1.3	Automatização da escolha do parâmetro R	53
4.2	VISÃO GERAL DO ALGORITMO STMKF	55
4.3	ALGORITMO SEQUENCIAL	55
4.4	ALGORITMO PARALELO (MT-STMKF)	58
4.5	IMPLEMENTAÇÃO PARA GPU (GPU-STMKF)	59
4.6	SOBRE AS IMPLEMENTAÇÕES DESENVOLVIDAS ...	60
5	RESULTADOS EXPERIMENTAIS	63
5.1	EFICÁCIA DO STMKF	63
5.1.1	Comparação do STMKF com os filtros que o compõem	63
5.1.2	Análise visual dos resultados	64
5.1.3	Comparativo do STMKF com os demais filtros	66
5.2	AVALIAÇÃO DO DESEMPENHO	69
5.2.1	Plataformas e Resoluções dos Vídeos	70
5.2.2	Resultados para Versão Sequencial	71
5.2.3	Resultados para Processadores Multicore	72
5.2.4	Resultados para GPUs	76
6	CONCLUSÃO	79
6.1	TRABALHOS FUTUROS	80
	REFERÊNCIAS	81
	APÊNDICE A – Resultados Complementares	89

1 INTRODUÇÃO

O presente trabalho visa desenvolver um filtro de ruídos gaussiano (*gaussian noise*) e de disparo (*shot noise*) em vídeos, que seja viável de ser processado em tempo real e que seja capaz de preservar detalhes das imagens que se fazem importantes na visão computacional.

1.1 MOTIVAÇÃO

A busca pela reprodução cada vez mais aprimorada de imagens leva pesquisadores a investigar técnicas sempre mais sofisticadas para sua captura e tratamento. No campo da Visão Computacional, a qualidade das imagens implica diretamente sobre os resultados do processo de visão, tornando assim as técnicas de captura e restauração importantes.

1.2 CONTEXTUALIZAÇÃO DO PROBLEMA

O desejo de perpetuar uma imagem no tempo surgiu a muitos anos atrás. Para chegar ao que temos hoje, em termos de captura de imagens, é necessário visitar a história e entender alguns princípios.

O primeiro conceito importante para a fotografia se refere a definição de “câmara escura”. Chamada também de “caixa escura”, ela possui um orifício em uma de suas paredes e na superfície oposta é projetado o objeto exposto a frente do orifício, sob luz, de forma invertida. Os seus princípios óticos já eram conhecidos por Aristóteles. Posteriormente o uso da câmara escura passou a ser um auxílio ao desenho e a pintura (KODAK, 2014a).

Na tentativa de melhorar a qualidade da imagem projetada na caixa preta, o tamanho do orifício era reduzido. No entanto, a quantidade de luz que adentrava pelo orifício era proporcionalmente menor, fazendo com que a imagem projetada escurecesse, tornando-a quase impossível de reconhecer. Em 1550, o físico Girolamo Cadano propôs o uso de uma lente biconvexa junto ao orifício, permitindo assim uma imagem nítida e clara (KODAK, 2014a).

Em 1826 o francês Joseph Nicéphore Niépce conseguiu fazer uma gravura sobre uma placa de estanho utilizando um derivado do petróleo fotossensível em sua cobertura e uma câmara escura, sendo esta consi-

derada a primeira fotografia (KODAK, 2014b).

Desde a primeira fotografia de Niépce houve diversas evoluções, como redução do tamanho das câmeras e fotografias coloridas. Somente em 1975 o engenheiro elétrico da Kodak, Steve Sasson, desenvolveu a primeira câmera digital da história, através da união de equipamentos analógicos e digitais, com uma resolução de 100x100 píxeis. No entanto, muito antes da criação da câmera digital, logo após Morse e Vail desenvolverem o telégrafo elétrico, inventores iniciaram suas explorações quanto ao envio e gravações de imagens (RICHARDS, 2013).

Em meados do século XX deu-se início a chamada *corrida espacial*, que tinha por finalidade, aparentemente, a expansão dos conhecimentos humanos para além dos limites da Terra. A partir de então, com a aquisição de imagens no espaço, sob condições adversas, e seu valor inestimável, fez-se necessário um processamento posterior a sua aquisição, com a finalidade de obter qualidade melhor nas imagens, culminando em um impulso nas pesquisas de restauração de imagens (ZAMPOLO, 2003).

Nos dias atuais, com o avanço da tecnologia e o surgimento de robôs autônomos, a restauração de imagens faz-se importante a medida que os meios visuais são de importância para a navegação destes robôs. Imagens com baixos níveis de ruído tendem a fornecer resultados melhores em algoritmos de interpretação de imagens.

A melhoria da qualidade das imagens também faz-se necessária à apreciação humana. Visto que câmeras de alta qualidade são de custo elevado, a restauração de imagens adquiridas com dispositivos de baixo custo se torna interessante.

Um dos principais fatores que degradam as imagens é o ruído. Ruído é uma deturpação sobre um sinal, geralmente indesejada. O ruído em uma imagem é uma variação nos valores dos píxeis que pode ocorrer durante o processo de captura, transmissão ou compactação, por exemplo. Apesar de indesejável, dificilmente o ruído pode ser evitado, o que torna a redução de ruídos importante para visão computacional.

A literatura apresenta várias técnicas de redução de ruído. Uma das técnicas utilizadas é o filtro de Kalman (KALMAN, 1960), que utiliza medições com incertezas realizadas ao longo do tempo e faz o resultado se aproximar dos valores reais da grandeza medida. Considerando a importância do tratamento de ruídos, este trabalho propõe um filtro de Kalman modificado, que considera características temporais e espaciais de sequências de imagens (vídeos).

1.3 OBJETIVOS

1.3.1 Objetivo Geral

O objetivo geral deste trabalho é reduzir ruídos gaussiano e *shot noise* em vídeos, de modo a preservar os detalhes e bordas, essenciais a visão computacional e/ou para apreciação humana, e em tempo real, ou seja, com um tempo de execução tão pequeno quanto o tempo entre a captura de um quadro e a captura do quadro seguinte.

1.3.2 Objetivos Específicos

Para alcançar o objetivo geral deste trabalho, os seguintes objetivos específicos foram definidos:

- Reduzir ruídos de aquisição que atrapalham o processo de visão computacional: desenvolver um método de redução de ruído gaussiano e de disparo, afim de que algoritmos de visão computacional possam explorar de forma eficiente as imagens adquiridas;
- Manter características essenciais a visão computacional: bordas e detalhes de imagens são importantes para algoritmos de visão computacional. Muitos dos métodos de redução de ruído também removem detalhes e borram as bordas;
- Tempo de computação adequado para ser utilizado em navegação robótica, transmissões “ao vivo” e video-chamadas.

1.4 ORGANIZAÇÃO DO TEXTO

Esta dissertação está organizada como segue: O primeiro capítulo é esta introdução. No segundo capítulo são descritos alguns conceitos e definições essenciais ao entendimento do trabalho. O terceiro capítulo apresenta uma revisão bibliográfica. No quarto capítulo é apresentado o algoritmo proposto. O quinto capítulo exhibe os resultados experimentais. No sexto capítulo são apresentadas as conclusões e trabalhos futuros. Ao final do trabalho encontram-se as referências bibliográficas.

2 CONCEITOS E DEFINIÇÕES

Neste capítulo serão apresentados os conceitos e definições que constituem a base teórica desta pesquisa.

2.1 IMAGEM DIGITAL

Segundo (CONCI; AZEVEDO; LETA, 2008), “*uma imagem pode ser considerada uma representação visual de objetos, podendo ser adquirida (fotos, filmes, cenas, etc.) ou gerada (pinturas, desenhos, esculturas, etc.)*”. Uma imagem digital é...

...uma representação de uma imagem em uma região discreta, limitada através de um conjunto finito de valores inteiros que representam seus pontos. As imagens digitais podem ser unidimensionais, bidimensionais ou tridimensionais. Podem ser binárias, monocromáticas, multibandas ou coloridas, quanto ao conteúdo de cada um de seus pontos. Também podem ser vetoriais ou matriciais quanto à forma de descrição (CONCI; AZEVEDO; LETA, 2008).

Comumente uma imagem digital é representada por uma matriz bidimensional, onde cada elemento representa um ponto (conhecido como píxel) (PEDRINI; SCHWARTZ, 2008).

Em uma imagem digital em tons de cinza (*gray scale*), os valores associados aos píxeis variam entre um mínimo e um máximo (PEDRINI; SCHWARTZ, 2008) de acordo com uma escala. É comum representar uma imagem *gray scale* utilizando 8 bits por píxel, no entanto este valor pode variar de acordo com a aplicação. Em uma imagem *gray scale* de 8 bits o valor do píxel pode variar entre 256 tons de cinza, sendo 0 o tom mais escuro (preto) e 255 o tom mais claro (branco).

Em imagens *multibandas* ou *multiespectrais*, os valores associados a cada píxel são valores vetoriais. Ou seja, cada píxel possui mais de um valor associado, onde cada valor pode seguir uma escala diferente e cada valor pode representar uma grandeza diferente. Uma imagem colorida é uma imagem *multiespectral* ou *multibanda* onde a cor é definida pelas grandezas luminância, matiz e saturação. Dado que a maioria das cores visíveis ao ser humano pode ser representada pela combinação das cores primárias vermelha, verde e azul, uma representação comum é o RGB (Vermelho, Verde e Azul em inglês: *Red*,

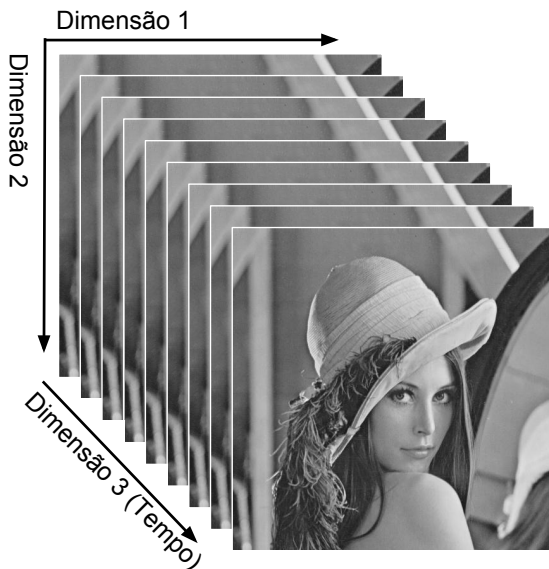
Green and Blue) (PEDRINI; SCHWARTZ, 2008). É comum o uso de 256 tons para cada componente do RGB.

2.2 VÍDEO DIGITAL

Um vídeo é basicamente uma sequência de imagens. Então, um vídeo digital é uma sequência dinâmica de imagens, onde o conteúdo imageado evolui ao longo do tempo, geralmente contendo objetos em movimento e/ou transformações. Desta forma há mais informação contida em um vídeo do que em uma imagem estática (BOVIK, 2009).

Considerando que uma imagem digital é basicamente um sinal multidimensional e que um vídeo digital é uma sequência de imagens digitais, pode-se dizer que um vídeo é também um sinal multidimensional, com a adição de mais uma dimensão: o tempo (BOVIK, 2010). A Figura 1 mostra uma sequência de imagens dispostas ao longo de duas dimensões espaciais e do tempo.

Figura 1 – Uma sequência de imagens formando um vídeo.



Cada imagem disposta em sequência formando um vídeo é chamada de *frame* ou quadro. Tipicamente um vídeo possui em torno de 30 frames por segundo, podendo variar de acordo com a aplicação.

2.3 RUÍDO EM IMAGENS

Ruído é uma deturpação de um sinal, geralmente indesejável, que se faz presente em diversas áreas da ciência. O ruído em imagens pode ser tratado como uma variação (randômica ou não) no brilho ou na cor, que não está presente no objeto imageado (FAROOQUE; SOHANKAR, 2013). As principais fontes de ruído em imagens são erros durante o processo de captura, compressão e/ou transmissão.

O ruído pode ser representado conforme a Equação 2.1, onde z_i representa o valor observado, v_i representa o valor real e r_i representa o valor do ruído, para cada píxel i (BUADES; COLL; MOREL, 2005).

$$z_i = v_i + r_i \quad (2.1)$$

Os principais tipos de ruído em imagens são: gaussiano, *salt-and-pepper*, de *shot noise*, de quantização e periódicos.

2.3.1 Ruído Gaussiano (*Gaussian Noise*):

O ruído Gaussiano é ocasionado por flutuações randômicas na leitura do sinal e é um dos ruídos mais comuns na captura de imagens. Pode ser modelado através de uma distribuição normal (BARDU, 2013), onde o ruído é independente para cada píxel e independente da intensidade do sinal (FAROOQUE; SOHANKAR, 2013). Cada píxel da imagem é modificado do seu valor original por um valor (geralmente) pequeno (MYTHILE; KAVITHA, 2011). A Figura 2 é um exemplo de ruído gaussiano sobre uma imagem.

Figura 2 – Ruído Gaussiano.



2.3.2 *Salt-and-Pepper Noise*

Também conhecido por *impulse noise*, *spike noise*, ruído randômico ou ruído independente, *salt-and-pepper noise* é geralmente causado por erros de conversão de sinal analógico para digital, erros de transmissão, píxeis mortos, entre outros. Este ruído pode ser visualizado como pontos pretos e/ou brancos na imagem, o que deu o nome ao tipo de ruído. Geralmente se apresenta em pequenas quantidades (MYTHILE; KAVITHA, 2011) (FAROOQUE; SOHANKAR, 2013).

A Figura 3 é um exemplo de ruído do tipo *salt-and-pepper* sobre uma imagem.

Figura 3 – Imagem com 5% de Ruído do tipo Salt-and-Pepper.



Fonte: recorte da figura da web¹

2.3.3 Ruído de Disparo (*Shot Noise*)

O ruído de disparo ou ruído de Poisson é tipicamente causado pela variação da quantidade de fótons contados pelo sensor para um nível de exposição, é independente em cada píxel e possui valor quadrático médio proporcional à raiz quadrada da intensidade da imagem. Ou seja, quanto maior a intensidade do valor medido pelo sensor, maior o tamanho do ruído. No entanto, como a intensidade do ruído cresce com a raiz quadrada da intensidade do sinal, a proporção entre os dois aumenta, tornando-o menos perceptível. Assim, o ruído de disparo é mais perceptível em baixas intensidade (imagens mais escuras). O

¹Disponível em: <<http://www.fit.vutbr.cz/~vasicek/imagedb/>> Acesso em mai. 2016

ruído de disparo não difere muito do ruído gaussiano e é um dos ruídos mais comuns em câmeras digitais(GARG; KUMAR, 2012).

2.3.4 Ruído de Quantização (*Quantization Noise*)

O ruído de quantização é causado pela quantização dos píxeis de uma imagem para um número discreto de níveis (MYTHILE; KAVITHA, 2011; FAROOQUE; SOHANKAR, 2013). Ou seja, ocorre quando se representa uma imagem como um número restrito de cores. Assim, a cor de cada píxel é representada pela cor mais próxima entre as disponíveis. Seu surgimento é comum na compactação de certos tipos de imagens.

A Figura 4 mostra o ruído de quantização devido a redução de níveis de intensidade (discretização) em uma imagem.

Figura 4 – Ruído de quantização.

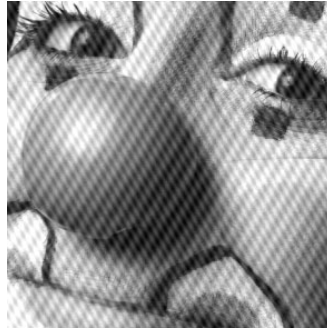


2.3.5 Ruídos Periódicos

Quando a imagem é submetida a uma perturbação periódica, não aleatória, diz-se que é um ruído periódico. O resultado é a formação de linhas sobre a imagem. A Figura 5 mostra um exemplo de uma imagem com ruído periódico.

²Disponível em: <<http://img-service.com/overview/pics/chap8/clown.html>>
Acesso em out. 2014

Figura 5 – Ruído periódico.



Fonte: imagem da web²

2.4 FILTROS PARA RUÍDOS EM IMAGENS E VÍDEOS

No campo de processamento de sinais, filtros são utilizados para remover componentes indesejados ou características do sinal. Neste trabalho, no entanto, um filtro é considerado um processo no qual o ruído é removido (*denoising*), resultando em uma imagem com valores mais próximos dos valores reais, referentes ao objeto imageado.

Apesar de indesejado, tanto em processos de visão computacional quanto para a apreciação humana, ruído em imagens dificilmente pode ser evitado. Assim, métodos de redução de ruído em imagens se tornam importantes.

Para cada tipo de ruído há abordagens mais apropriadas para a sua solução (MYTHILE; KAVITHA, 2011). Este trabalho visa reduzir ruído gaussiano e ruído de disparo, que são os tipos mais comuns de ruído no momento da captura (BARDU, 2013; GARG; KUMAR, 2012). Vários outros trabalhos tratam da remoção de ruídos em imagens e vídeos, e serão apresentados na seção de trabalhos correlatos.

O processo de remoção de ruídos em imagens estáticas ocorre sobre o domínio do espaço. Ou seja, o valor de um determinado píxel é corrigido de acordo com o seu valor e o valor de seus vizinhos. No processo de *denoising* de vídeos, a remoção de ruídos ocorre sobre o domínio do espaço, ou sobre o domínio do tempo, ou ainda no domínio do espaço e do tempo. Portanto, o valor de um determinado píxel pode ser corrigido de acordo com o seu valor e o de seus vizinhos no frame atual, assim como nos frames passados e/ou futuros.

A seguir são apresentados alguns filtros mais comuns importantes

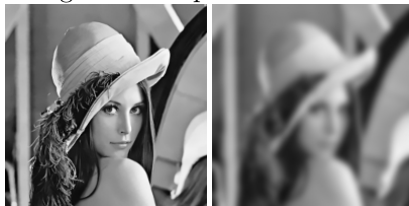
para ruídos em imagens estáticas. Os filtros para vídeos serão tratados no Capítulo 3 - Trabalhos Relacionados.

2.4.1 Filtro de suavização linear (*Linear Smoothing*): *Blur*

Os filtros do tipo *blur* fazem uma média entre píxeis vizinhos. Como resultado, o píxel que possuía um valor elevado de ruído passa a ter seu ruído distribuído entre os vizinhos, reduzindo o seu valor absoluto. No entanto, os píxeis que diferem de seus vizinhos por guardar detalhes da imagem, também passam a ter seu valor distribuído com os vizinhos, causando um borramento na imagem e, conseqüentemente, a perda de detalhes.

A média com os vizinhos pode ser simples, onde todos os píxeis possuem o mesmo peso, ou de média ponderada, onde os píxeis recebem pesos distintos, de acordo com algum padrão (*kernel*), como no caso do *blur* gaussiano, onde os pesos seguem uma distribuição gaussiana em função da distância em relação ao píxel de referência. A Figura 6 exhibe um *blur* gaussiano aplicado sobre uma imagem.

Figura 6 – *Blur* gaussiano aplicado sobre a imagem Lena.



2.4.2 Filtro Adaptativo (*Adaptive Filter*)

Conhecido também como o Filtro de Wiener, é um tipo de filtro linear, que se adapta a variância local da imagem. Nas regiões da imagem onde a variância é elevada (regiões de borda) o filtro faz uma suavização de baixa intensidade e nas regiões onde a variância é reduzida (regiões mais distante das bordas) o filtro faz uma suavização mais rebuscada. Assim, Wiener consegue resultados melhores que os filtros de suavização linear, porém seu custo computacional é maior.

2.4.3 Filtro de Mediana (*Median Filter*)

O filtro de mediana é um filtro não linear que, em vez de fazer a média entre os píxeis da vizinhança, como no caso do *blur*, utiliza a mediana entre os píxeis vizinhos. Ou seja, os valores dos píxeis vizinhos são ordenados em ordem crescente e é escolhido o valor do meio em vez do valor médio. É bastante efetivo contra o ruído *salt-and-pepper* e tende a borrar pouco as bordas (CONCI; AZEVEDO; LETA, 2008).

2.4.4 Difusão Anisotrópica

Perona e Malik (1990) sugerem uma nova definição de escala espacial através da difusão anisotrópica, um processo de difusão baseado na equação parcial diferencial não linear. A difusão anisotrópica tornou-se uma ferramenta muito útil na suavização de imagens, detecção de bordas, segmentação e aprimoramento de imagens. A filtragem através do processo de difusão anisotrópica consegue suavizar o ruído preservando os limites das regiões e pequenas estruturas dentro da imagem, quando seus parâmetros são determinados corretamente. Tsiotsios e Petrou (2013) propõem em seu trabalho um critério de parada automática para o algoritmo de difusão anisotrópica, que leva em consideração a qualidade das bordas preservadas ao invés de somente os níveis de suavização alcançados. A Figura 7, ao centro, mostra o resultado da difusão anisotrópica aplicada sobre uma imagem.

2.4.5 *Non-Local Means*

Buades, Coll e Morel (2005) apresentam um algoritmo para redução de ruídos em imagens baseado em métodos não-locais (*non-local means*). Este método não só compara o valor de um píxel em si, mas a configuração geométrica de sua vizinhança, o que permite ao filtro um resultado mais robusto. No entanto, a análise da configuração geométrica exige muitas operações, fazendo com que a complexidade do algoritmo seja da ordem de $21609 \times N^2$, onde N^2 é o número de píxeis da imagem.

2.4.6 Método Misto Adaptativo

Liu, Fu e Zhang (2011) propõem um método misto adaptativo para a restauração de imagens. Num primeiro momento a imagem é decomposta em dois componentes: uma estrutura geométrica e um padrão de oscilação. Em seguida, para restaurar a parte da estrutura é utilizada uma equação de difusão bidirecional, e para a parte oscilante é aplicado um filtro de métodos não-locais. Desta forma o filtro misto se propõe a reunir vantagens de outros filtros. A Figura 7, à direita, mostra o resultado do método.

Figura 7 – À Esquerda imagem com ruído. Ao centro, resultado da difusão anisotrópica aplicada sobre a imagem. À direita, resultado dos métodos misto adaptativo.



Fonte: Liu, Fu e Zhang (2011)³

2.5 FILTRO BILATERAL

O filtro bilateral é um filtro de imagens que atua no domínio do espaço, proposto por Tomasi e Manduchi (1998). Ele é um filtro de suavização (*smoothing*) não linear, que preserva as bordas e borra (*blur*) as demais regiões. Para isto, cada píxel da imagem é substituído por uma média ponderada dos píxeis vizinhos, considerando a proximidade geométrica e sua similaridade fotométrica. Uma implementação com complexidade $O(n)$, onde n é o número de píxeis de uma imagem, é proposta por Chaudhury (2013).

O resultado do Filtro Bilateral depende basicamente da escolha de três parâmetros: d , que é o diâmetro no qual são considerados os

³Disponível em: <<http://opticalengineering.spiedigitallibrary.org/article.aspx?articleid=1157734>> Acesso em out. 2014

píxeis vizinhos; σ_c , que é o fator de propagação fotométrica, ou seja, é o fator que determina o peso da cor no cálculo e σ_s , que é o fator de propagação geométrica, isto é, o fator que determina o quanto a proximidade do píxel interfere no cálculo. A escolha destes parâmetros depende da aplicação do filtro. Segundo descrito em OpenCV (2016), para uma simplificação, σ_c e σ_s podem assumir o mesmo valor.

Figura 8 – Filtro Bilateral aplicado sobre uma imagem. A esquerda, imagem original. Ao centro, filtro Bilateral aplicado com $d = 6$, $\sigma_c = 100$ e $\sigma_s = 100$. A direita, filtro Bilateral aplicado com $d = 50$, $\sigma_c = 50$ e $\sigma_s = 50$



Apesar do Filtro Bilateral ser computacionalmente eficiente, os seus resultados são insatisfatórios quando utilizado para remoção de ruídos em vídeos. Por se tratar de um filtro que somente atua no domínio do espaço (um filtro espacial), o processo de redução de ruídos é aplicado a cada frame, individualmente, sem levar em conta os frames anteriores e/ou posteriores. Isto resulta na oscilação do brilho em pequenas áreas, conhecido como *local flickering artifacts* (DUFAUX et al., 2016), que é desagradável aos olhos.

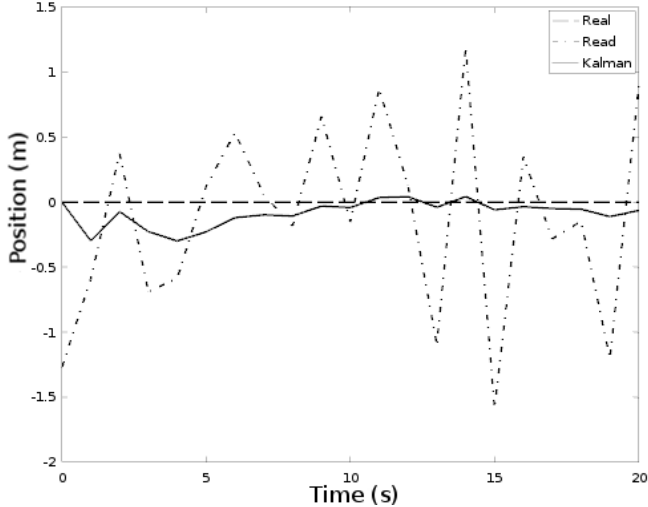
2.6 FILTRO DE KALMAN

O filtro de Kalman, desenvolvido por Rudolf Kalman (KALMAN, 1960), utiliza medições com incertezas realizadas ao longo do tempo e faz o resultado se aproximar dos valores reais da grandeza medida.

O filtro de Kalman é muito usado em diversas áreas que dependem de estimação, como sistemas de orientação e navegação, visão por computador, processamento de sinais e econometria (MathWorks, 2016). A Figura 9 mostra o filtro de Kalman aplicado sobre um sinal referente a posição ao longo do tempo. Pode-se observar que o valor estimado pelo filtro de Kalman (linha contínua) está mais próximo ao valor real

(linha tracejada) do que o valor lido (linha tracejada e pontilhada).

Figura 9 – Filtro de Kalman aplicado (linha contínua) sobre um valor lido (linha tracejada e pontilhada) e sua comparação com o valor real (linha tracejada).



Aplicando o mesmo conceito, píxel a píxel ao longo do tempo, em uma imagem podemos aproximar o valor do píxel ao valor real, reduzindo o erro.

O Filtro de Kalman é desenvolvido em duas fases: predição e correção. Na fase de predição o filtro faz uma estimativa do próximo estado x_k^- (Equação 2.2), assim como a estimativa da covariância do erro P_k^- (Equação 2.3). Na fase de correção ele computa o ganho de Kalman K_k (Equação 2.4) e utilizando os valores lidos z_k faz uma atualização (*update*) do valor estimado para o estado x_k (Equação 2.5) e para a covariância do erro P_k (Equação 2.6). Estas duas fases são executadas para cada nova aquisição de dados.

$$x_k^- = Ax_{k-1} + Bu_k \quad (2.2)$$

$$P_k^- = AP_{k-1}A^T + Q \quad (2.3)$$

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \quad (2.4)$$

$$x_k = x_k^- + K_k(z_k - Hx_k^-) \quad (2.5)$$

$$P_k = (I - K_k H)P_k^- \quad (2.6)$$

As variáveis A , B e H são matrizes, no entanto, considerando que o valor de cada píxel é uma grandeza unidimensional, estas variáveis assumem um valor numérico. Para este problema, podemos considerar A , B e H como constantes de valor 1. u_k representa um sinal de controle. Como nesta aplicação não há sinal de controle, u_k recebe o valor 0. Desta forma, as Equações 2.2, 2.3, 2.4, 2.5 e 2.6 podem ser reescritas, respectivamente, de forma simplificada, pelas Equações 2.7, 2.8, 2.9, 2.10 e 2.11.

$$x_k^- = x_{k-1} \quad (2.7)$$

$$P_k^- = P_{k-1} + Q \quad (2.8)$$

$$K_k = P_k^- (P_k^- + R)^{-1} \quad (2.9)$$

$$x_k = x_k^- + K_k (z_k - x_k^-) \quad (2.10)$$

$$P_k = (I - K_k) P_k^- \quad (2.11)$$

Q e R são valores constantes e a escolha destes valores interfere na rigidez do filtro. Fixando Q , se aumentarmos R , flutuações de z_k interferem menos em x_k . Diminuindo R , flutuações de z_k interferem mais em x_k . Da mesma forma, fixando R e aumentando Q , flutuações de z_k interferem mais em x_k , e reduzindo Q , flutuações em z_k interferem menos em x_k .

A Figura 10 ilustra o resultado do filtro aplicado a capturas realizadas com uma câmera de baixo custo. A captura foi realizada em ambiente com baixa intensidade luminosa, onde o ruído se torna mais evidente.

No entanto, o filtro de Kalman tem como consequência uma resistência a uma mudança abrupta de nível do valor lido. O gráfico da Figura 11 ilustra o resultado do filtro aplicado a um sinal que possuía como valor real 0 e no instante $t = 10s$ passou a ter valor 10, formando uma espécie de degrau no gráfico. Os valores lidos oscilam próximo aos valores reais. O valor estimado pelo filtro até o instante $t = 9s$ é mais próximo do valor real do que o valor lido. A partir do instante $t = 10$, quando ocorre uma mudança abrupta, o valor estimado demora a convergir para próximo de 10.

A Figura 12 mostra esta mesma questão em uma figura capturada, onde o Filtro de Kalman é aplicado. Nota-se a formação de borramentos (fantasmas) quando ocorre movimento. Quando um objeto se movimenta em uma sequência de imagens, o valor do píxel sofre uma variação abrupta, para cima ou para baixo, similar à apresentada pela Figura 11. Os fantasmas formados são devido à resistência que o

Figura 10 – Filtro de Kalman aplicado a uma seqüência de imagens adquiridas por uma câmara de baixo custo em um ambiente de pouca luz. No topo, figura adquirida. Na base, filtro de Kalman aplicado.

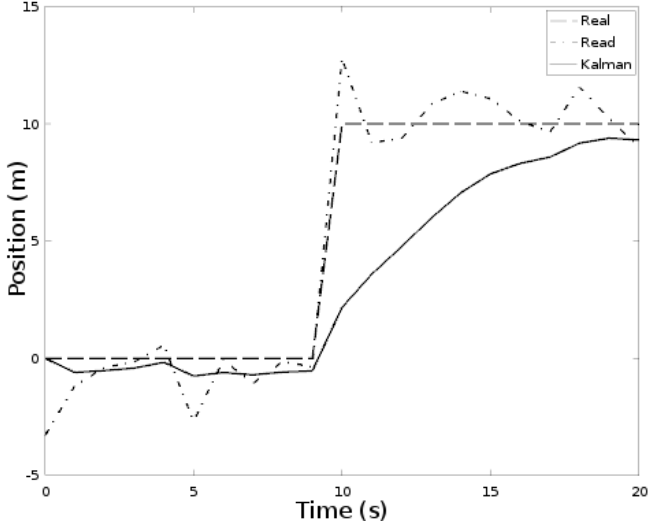


filtro oferece a convergir para o novo valor do píxel.

2.7 MÉTRICAS DE QUALIDADE EM IMAGEM

A eficácia de um filtro de ruído em imagem é avaliado comumente através da Relação de Sinal-Ruído de Pico (*Peak Signal to Noise Ratio* - PSNR). O PSNR é calculado através da Equação 2.12, onde m é o valor máximo de um píxel e MSE é o erro quadrático médio (*Mean Square Error*), calculado através da Equação 2.13 (MAHMOUD; FAHEEM; SARHAN, 2008). O PSNR de uma seqüência de imagens (vídeo) é a média dos PSNRs de cada *frame* da seqüência. Quanto mais elevado o PSNR, maior a semelhança entre a imagem original

Figura 11 – Filtro de Kalman aplicado (linha contínua) sobre um valor lido (linha tracejada e pontilhada) e sua comparação com o valor real (linha tracejada).



sem ruído e a imagem filtrada.

$$PSNR = 10 \times \log \frac{m^2}{MSE} \quad (2.12)$$

$$MSE = \frac{1}{M \times N} \sum_{i=1}^N \sum_{j=1}^M \left(x(i, j) - x'(i, j) \right)^2 \quad (2.13)$$

Na Equação 2.13, M e N representam a largura e a altura da imagem. $x(i, j)$ representa o valor do píxel da imagem original e $x'(i, j)$ representa o valor do píxel corrigido, numa posição i, j da imagem.

Apesar do PSNR ser largamente utilizado, o PSNR difere da qualidade da imagem percebida (WANG; BOVIK, 2009). Outro método comum de avaliação do resultado é através da similaridade estrutural (*structural similarity* - SSIM) (WANG; BOVIK; SIMONCELLI, 2004). O SSIM é apontado como um indicador melhor da qualidade da imagem percebida pelo ser humano (WANG; BOVIK, 2009) e pode ser calculado através da equação 2.14, a seguir

Figura 12 – No topo, um frame da sequência *Garden*. Na base, o borramento ocorrido devido ao filtro de Kalman aplicado aos objetos em movimento da sequência.

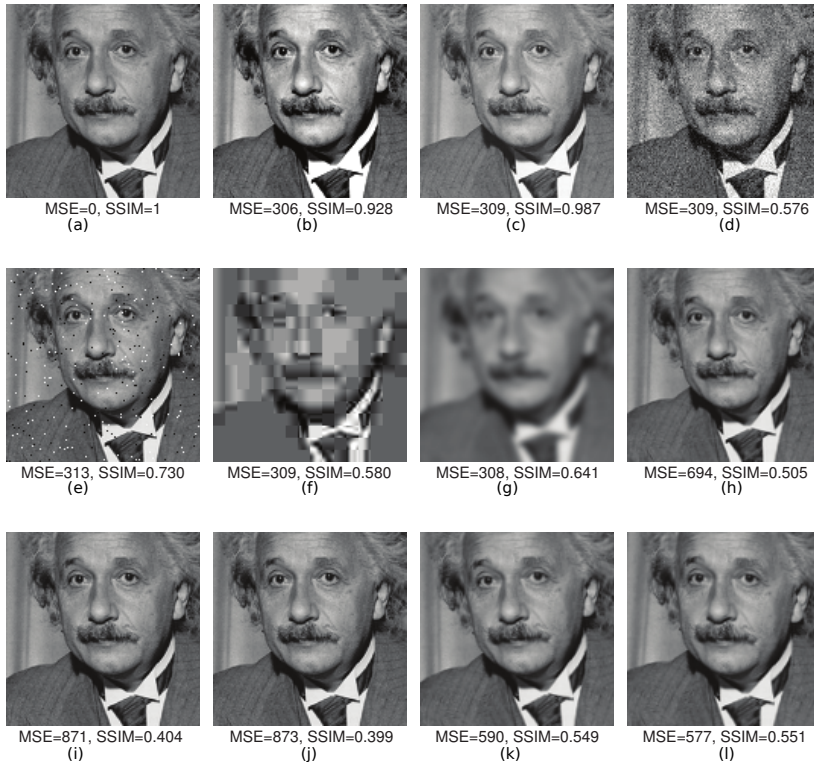


$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \quad (2.14)$$

quando x e y são fragmentos de imagens (*image patches*) extraídos da janela local da imagem original e da imagem distorcida (HAN; CHEN, 2012). μ , σ^2 e σ_{xy} representam respectivamente a média, a variância e a correlação cruzada (*cross-correlation*) computados para a janela local. $c_1 = (k_1G)^2$ e $c_2 = (k_2G)^2$ são duas variáveis para estabilizar a divisão por um denominador fraco, $k_1 = 0.01$, $k_2 = 0.03$ e $G = 2^{\# \text{ bits por pixel}} = 255$. O SSIM de um frame de vídeo é calculado a partir da média dos SSIM locais e o SSIM final de um vídeo é a média entre o SSIM dos frames da sequência completa (HAN; CHEN, 2012). Quanto mais o SSIM se aproxima de 1, maior a semelhança entre a imagem original sem ruído e a imagem filtrada.

A Figura 13, extraída de Wang e Bovik (2009), ilustra o quanto as duas métricas se aproximam da percepção humana. A imagem (a) é a imagem original, e as métricas são computadas entre as demais imagens e esta. O MSE, e logo o PSNR, são idênticos para diversas imagens, enquanto que as imagens apresentam diferenças drásticas quanto a sua qualidade visual [imagens (b)-(g)]. Por outro lado, algumas imagens visualmente idênticas [imagens (h)-(l)] apresentam valores de PSNR e SSIM bem destoantes entre si. Assim, apesar de métricas usuais, elas podem divergir do resultado observado.

Figura 13 – Métricas de qualidade de imagens e suas diferenças em relação a qualidade percebida.



Fonte: Wang e Bovik (2009) .

2.8 TEMPO REAL

O conceito de tempo real pode assumir diversas roupagens, de acordo com a área a qual está associada. Para este trabalho, assumem-se as restrições de tempo da área de processamento de sinais digitais: um sistema de processamento de sinais digitais de tempo real demanda que o tempo para se processar um sinal seja menor que o período de amostragem, de modo que o processamento seja concluído antes de uma nova amostra (KUO; LEE; TIAN, 2006). Ou seja, o tempo adequado para processar um frame (o sinal) é o tempo entre a capturas de um quadro e de outro.

A partir deste conceito de tempo real, podemos elencar duas restrições:

Considerando aplicações de vídeo atuais, como transmissões *ao vivo*, é comum utilizar em torno de 30 frames por segundo (FPS). Neste caso, pode-se assumir que um filtro de redução de ruídos é possível de ser executado em tempo real se conseguir processar mais de 30 FPS. Por outro lado, em aplicações como câmeras de vigilância, é comum o uso de taxas menores, próximo de 4 FPS. Assim sendo, a consideração sobre o que é tempo real ou não depende da aplicação.

A outra restrição a observar é que, dentro deste conceito de tempo real, se dispõe somente de frames já amostrados (os passados e o atual) para computar o frame atual.

Transmissões televisivas, vídeo-chamadas, monitoramento de ambientes e navegação robótica podem ser tomados como exemplo de aplicações onde se faz necessário que o processamento de vídeo seja em tempo real.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta o estado atual em que se encontram as pesquisas relacionadas a remoção de ruídos em vídeos. A análise do estado da arte é realizado segundo o método de revisão sistemática de literatura definido por Kitchenham (2004).

3.1 DEFINIÇÃO DA REVISÃO DA LITERATURA

Uma revisão sistemática da literatura tem por objetivo identificar, avaliar e interpretar todas as pesquisas relevantes a uma área ou questão de pesquisa (KITCHENHAM, 2004). Para nortear esta revisão sistemática a seguinte questão foi levantada: Como remover ruídos em vídeos em tempo real?

As buscas foram realizadas via web nas bases IEEE Xplore¹, Springer², Scopus³, Science Direct⁴, Spie Digital Library⁵ e Google Scholar⁶.

Os termos de busca utilizados foram *real-time* e *video denoising*. A Tabela 1 mostra estes termos juntamente com outros termos sinônimos, cujos também foram utilizados nas buscas.

Tabela 1 – Termos de busca e seus sinônimos de busca.

Termos	Sinônimos
Real time	real-time, fast
Video denoising	spatio-temporal denoising, video filter, noise filter, spatio-temporal filter

Foram considerados principalmente trabalhos que tratam de remoção de ruídos em vídeos. Trabalhos focados em remoção de ruído em imagens também foram analisados, mas de uma forma menos aprofundada.

Foram excluídos da pesquisa os trabalhos cuja remoção de ruídos em vídeos é implementada em Hardware.

¹<http://ieeexplore.ieee.org/>

²<https://www.springer.com>

³<https://www.scopus.com>

⁴<http://www.sciencedirect.com/>

⁵<http://spiedigitallibrary.org/>

⁶<https://scholar.google.com.br/>

3.2 EXECUÇÃO DA BUSCA

As buscas foram realizadas em 4 fases: **Fase 1:** Aplicação das strings de busca, **Fase 2:** Análise de títulos e palavras-chave, **Fase 3:** Leitura do Abstract e **Fase 4:** Identificação de contribuições. A Tabela 2 mostra a quantidade de trabalhos remanescentes após cada uma das fases, para cada uma das bases de dados pesquisadas.

Tabela 2 – Trabalhos remanescentes a cada fase da revisão sistemática.

Bases	Fase 1	Fase 2	Fase 3	Fase 4
IEEE Xplore	16	8	5	3
Springer	53	10	6	3
Scopus	70	11	6	4
Science Direct	18	5	3	1
Spie Digital Library	25	8	4	2
Google Scholar	764	27	10	6

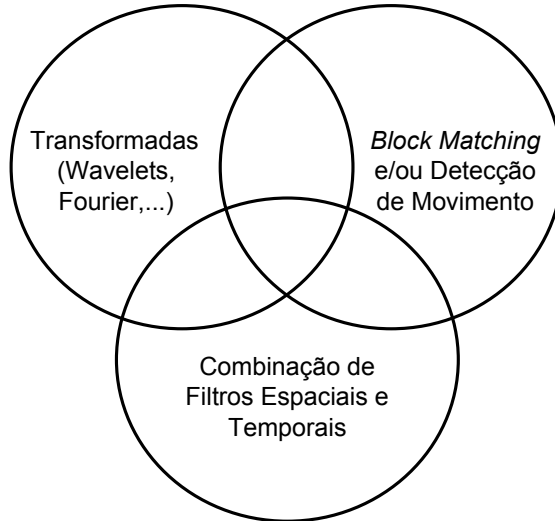
Após esta etapa, foram excluídos os trabalhos duplicados. Foi feita também uma breve análise dos trabalhos referenciados pelos trabalhos remanescentes, afim de evitar a perda de algum trabalho importante da área. Os trabalhos relacionados selecionados estão descritos a seguir.

3.3 TRABALHOS RELACIONADOS

De uma forma geral, os métodos de remoção de ruídos em vídeo podem ser divididos em três grupos principais: métodos baseados em transformadas (PIZURICA; ZLOKOLICA; PHILIPS, 2004; SELESNICK; LI, 2003; MAHMOUD; FAHEEM; SARHAN, 2008; RAHMAN; AHMAD; SWAMY, 2007; ZLOKOLICA; PIZURICA; PHILIPS, 2006), métodos que utilizam algoritmos de detecção de movimento e/ou *block matching* (correspondência entre blocos de imagem) (DABOV et al., 2006; DABOV; FOI; EGIAZARIAN, 2007; LI; ZHENG, 2009; LI; ZHANG; DAI, 2011) e métodos que combinam filtros nos domínios temporais e espaciais (TOMASI; MANDUCHI, 1998; ZLOKOLICA; PHILIPS; VILLE, 2002; ZUO et al., 2013; CHAN et al., 2005; CHEN et al., 2008; HONG-ZHI; LING; SHU-LIANG, 2013). A maioria dos filtros podem ser definidos como pertencentes a pelo menos um destes três grupos, mas muitas vezes mesclam técnicas de mais de um grupo, pertencendo, assim, a mais de um grupo. A Figura 14

ilustra os três grupos.

Figura 14 – Os três principais grupos de filtros de vídeo.



3.3.1 Métodos Baseados em Transformadas

As transformadas, de uma forma geral, possuem diversas aplicações em processamento e análise de imagens. Elas resultam na alteração da representação inicial de uma imagem (PEDRINI; SCHWARTZ, 2008). Ou seja, as imagens são convertidas para um outro domínio, como o da frequência, por exemplo, sobre o qual é feito algum processamento ou análise. No caso da aplicação de transformadas para remoção de ruídos, as transformadas wavelet são bastante comuns. As transformadas wavelets são aplicadas sobre as imagens com ruídos, seguido da recuperação dos coeficientes do sinal original e uma transformada inversa para retornar ao domínio original da imagem.

Pizurica, Zlokolica e Philips (2004) desenvolveram um filtro baseado em Transformadas Wavelet 2D, que faz uma filtragem no domínio do espaço, seguido de um filtro no domínio do tempo, baseado na detecção do movimento. Neste filtro, a detecção de movimento e o filtro temporal são aplicados sobre os frames que já foram filtrados espacialmente.

Rahman, Ahmad e Swamy (2007) propuseram uma função de densidade de probabilidade conjunta dos coeficientes de vídeo wavelet para quaisquer dois quadros vizinhos utilizando a distribuição gaussiana bivariável. Os parâmetros da função de densidade que mede a correlação entre os coeficientes da transformada de wavelet de dois frames é usada como uma medida do movimento que há entre dois frames. Esta função de densidade é então utilizada para filtragem espacial dos coeficientes wavelet ruidosos, por um estimador bivariado de máximo a posteriori (*bivariate maximum a posteriori estimator*). Os coeficientes filtrados espacialmente são então passados a um filtro de média temporal para remoção adicional de ruído.

Selesnick e Li (2003) apresentam o design e a aplicação das transformadas complexas wavelet 3-D *dual-tree* não separáveis para filtragem de vídeo. A decomposição de vídeos baseado em movimento em multi-escala é alcançada com esta transformação que isola em sua sub-banda o movimento ao longo de diferentes direções. Além disso, é explorada a remoção de ruídos utilizando transformadas 2-D e 3-D *dual-tree*, onde a transformada 2-D é aplicada a cada frame individualmente.

O algoritmo WRSTF (*Wavelet Domain Recursive Temporal Filtering*) é proposto por Zlokolica, Pizurica e Philips (2006) como um método de remoção de ruídos baseado em filtragem de banda wavelet não dizimada⁷. No método proposto, a estimativa do movimento e a filtragem adaptativa recursiva temporal é realizada em um *loop* fechado, seguido de um filtro adaptativo espacial intra-frame. O algoritmo é implementado em três passos: estimativa do movimento, compensação do movimento e o esquema de filtragem adaptativo espacial. Toda a computação ocorre no domínio das wavelets.

O processo de remoção de ruído através de transformadas pode ter uma complexidade computacional elevada associada. As transformadas mais simples tendem a ser somente espaciais, necessitando de uma filtragem temporal complementar e/ou a identificação da direção do movimento.

3.3.2 Métodos Baseados em Algoritmos de Detecção de Movimento e *Block Matching*

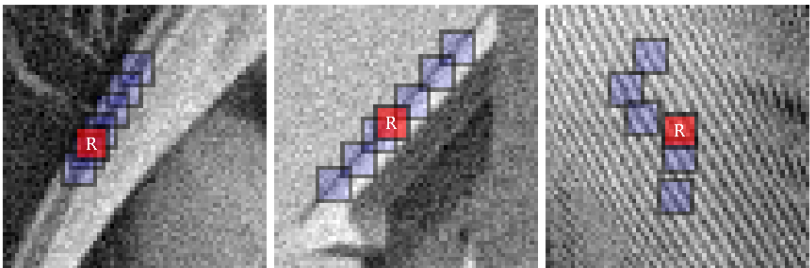
Algoritmos de *block matching* buscam blocos de imagem similares dentro de uma mesma imagem ou de uma sequência de imagens. É

⁷Tradução própria para: *video denoising method based on nondecimated wavelet band filtering*.

comum o uso de algoritmos de *block matching* para executar filtragem de ruído em imagens e vídeos e sua utilização é bastante variada. Existe um grande número de algoritmos de *block matching* e a complexidade dos algoritmos também é bastante variada. Os algoritmos de detecção de movimento (*motion detection*) tentam estimar a direção do movimento em uma sequência de imagens. Uma forma de fazer a detecção de movimento é a localização dos blocos (*block matching*) de imagem ao longo de frames subsequentes.

Dabov et al. (2006) propuseram o BM3D (*Block-matching 3D Denoising*) que faz a redução de ruídos através do processamento de blocos similares dentro de uma imagem. Através de *block matching* o algoritmo encontra blocos idênticos a um bloco de referência, os armazena em uma estrutura de dados 3D e faz a reconstrução dos blocos encontrados através de uma média ponderada. A Figura 15 ilustra o *matching* entre o blocos similares, onde o bloco vermelho (*R*) é o bloco de referência e os azuis os similares. A Figura 16 mostra os resultados deste filtro quando aplicado sobre imagens com ruído.

Figura 15 – Blocos de referência, em vermelho, e blocos similares encontrados (*block-matching*).



Fonte: imagem da web⁸

O algoritmo *Video Block-matching 3D Denoising* (VBM3D), proposto por Dabov, Foi e Egiazarian (2007), estende o algoritmo BM3D para sequências de imagens (vídeos), encontrando blocos não somente no espaço, mas também no tempo, nos frames anteriores e seguintes.

Li e Zheng (2009) e Li, Zhang e Dai (2011) descrevem a redução de ruídos em vídeos de forma similar ao VBM3D, trabalhando com os

⁸Disponível em: <<http://www.cs.tut.fi/~foi/BM-3D-PRESENT/den3d-slide-04.html>> Acesso em out. 2014

⁹Disponível em: <<http://www.cs.tut.fi/~foi/BM-3D-PRESENT/den3d-slide-37.html>> Acesso em out. 2014

Figura 16 – Fragmentos de imagens com ruído gaussiano ($\sigma = 50$) e o resultado do filtro baseado em *block-matching*.



Fonte: imagem da web⁹

blocos, por sua vez, em 3D (*3D-Patch*) e os armazena em uma estrutura 4D.

Estes trabalhos, ao lado de outros, compõem o estado da arte em redução de ruídos em vídeos. No entanto, sua complexidade computacional torna-os proibitivos para aplicações em tempo real.

3.3.3 Métodos que Combinam Filtros nos Domínios Temporais e Espaciais

Métodos que combinam filtros espaciais e temporais tendem a ser algoritmos de complexidade computacional mais reduzida, em relação aos outros dois grupos. Esta característica é fundamental para aplicações em tempo real.

Um filtro não linear que ordena os píxeis dentro de uma janela 3D, considerando suas diferenças com o valor do píxel central, é proposto por Zlokica, Philips e Ville (2002). O próximo passo do filtro é fazer uma média dos píxeis na janela, e uma ponderação de acordo com sua posição no ordenamento. Este filtro é uma extensão do filtro KNN (*K-Nearest Neighbour*) proposto por Mitchell e Mashkit (1992) e descrito inicialmente por Davis e Rosenfeld (1978). Uma das vanta-

gens deste filtro é sua independência em relação ao tipo de ruído (ruído Gaussiano e *impulse noise*, ou alguma combinação dos dois)

O algoritmo *Spatiotemporal Varying Filter* (STVF) é proposto por Chan et al. (2005) e gera o melhor valor candidato para substituir o valor com ruído do píxel atual explorando a correção dos píxeis vizinhos dentro de uma pequena região, utilizando pesos ótimos para eles. O STVF é capaz de gerar resultados ótimos, no sentido de minimizar o erro do método dos mínimos quadrados. Além disto, o STVF retém a nitidez das bordas em limites de objetos e reduz a variância do ruído nas áreas lisas.

Hong-Zhi, Ling e Shu-Liang (2013) utilizam vários métodos para remoção de ruídos. A técnica consiste em detectar regiões sem e com movimento. Sobre as regiões onde não ocorre movimento é aplicado um filtro de Kalman bilateral temporal. Nas regiões onde ocorre movimento é aplicado um filtro espacial bilateral adaptativo de métodos não-locais (*spatial bilateral adaptive Nonlocal means - ANL*). Assim, o filtro consegue remover os fantasmas nas regiões com movimento, oriundas do filtro de Kalman, com bons valores de PSNR. De forma similar, Zuo et al. (2013) aplicam o filtro de Kalman às regiões onde não ocorre movimento. Porém, nas regiões com movimento é aplicado um filtro bilateral. A imagem resultante é gerada a partir de uma ponderação entre os dois filtros. Tanto Hong-Zhi, Ling e Shu-Liang (2013) quanto Zuo et al. (2013) fazem uso de algoritmo de *block matching* para detectar/estimar o movimento e para realizar a ponderação entre as técnicas.

O STMKF, que é proposto a seguir, se utiliza dos filtros *blur*, bilateral e Kalman, para gerar o frame filtrado. Ele não faz uso de técnicas mais apuradas de detecção de movimento, assim como de compensação de movimento, devido a alta complexidade dos algoritmos deste tipo. A complexidade computacional de cada um dos passos do filtro é $O(n)$, onde n é o número de píxeis de cada frame, fazendo o algoritmo possível de ser executado em tempo real.

4 SPATIO-TEMPORAL MODIFIED KALMAN FILTER - STMKF

O algoritmo proposto *Spatio-temporal Modified Kalman Filter* - STMKF é uma fusão entre o filtro bilateral, que opera no domínio do espaço, e do filtro de Kalman, que opera pixel a pixel no domínio do tempo. A seguir são apresentados o algoritmo proposto e sua implementação.

4.1 FILTRO PROPOSTO

O desenvolvimento do STMKF se baseia em três pontos principais, apresentados nas Subseções 4.1.1, 4.1.2 e 4.1.3.

4.1.1 Convergência Otimizada do Filtro de Kalman para Regiões com Movimento

Como visto anteriormente, o filtro de Kalman, quando aplicado em vídeos, pode demorar a convergir para o devido valor em regiões onde ocorre movimento. O filtro de Kalman se baseia na covariância do erro P_k para calcular K_k e estimar os valores, de forma que o valor lido recebe maior peso ou o valor estimado recebe maior peso.

Quando ocorre uma mudança no cenário, que é quando ocorre o surgimento dos fantasmas, a confiança P_k , nos valores z_k lidos, precisa levar em conta também esta mudança de cenário. Analisando a Equação 2.8, vemos que a projeção da covariância P_k é constituída de duas componentes: a covariância no instante anterior P_k^- e Q . Considerando-se uma projeção, é necessário manter o componente referente ao instante anterior P_k^- . Por outro lado, a componente Q é um parâmetro que diz respeito a certeza que se tem em P_k^- e pode ser reescrita conforme a Equação 4.1:

$$P_k^- = P_{k-1} + \Delta^2 Q \quad (4.1)$$

onde Δ é a diferença entre a média dos vizinhos do pixel, nos instantes k e $k-1$, descrita pela Equação 4.2. Em outras palavras, Δ é a diferença entre dois *frames* com *blur* consecutivos.

$$\Delta = b_k - b_{k-1} \quad (4.2)$$

Considerando o ruído Gaussiano e o ruído de disparo, o ruído possui dimensão igual a 1 píxel, é independente dos vizinhos e uniformemente distribuído. Assim, numa condição onde não ocorre nenhuma alteração no cenário, o Δ tende a se manter pequeno. Quando o cenário é modificado, vários píxeis sofrem alteração e a média dos vizinhos sofre uma alteração grande, fazendo assim com que Δ assuma um valor mais elevado. Desta forma, o filtro proposto tende a manter as propriedades do filtro de Kalman nas regiões onde não ocorrem alterações, e passa a assumir valores mais próximos de z_k nos píxeis onde ocorreu movimento.

Resultados experimentais mostraram que a utilização de Δ^2 na Equação 4.1 traz resultados melhores do que utilizar somente Δ . Isto ocorre por que torna as diferenças entre dois frames consecutivos mais expressivas, reduzindo a confiança P_k em regiões de movimento, enquanto que as similaridades ($\Delta \rightarrow 0$), se mantém menos expressivas, mantendo a confiança P_k em regiões sem movimento. Por outro lado, a utilização de expoentes maiores que 2 em Δ não apresentaram vantagens nos experimentos.

A Figura 17 mostra um mapa de K_k do frame 20 de Salesman. Pode-se observar que os valores de K_k são mais elevados (mais brancos) nas regiões onde há movimento na imagem. Ou seja, nas regiões onde ocorre movimento z_k recebe um peso maior, enquanto que nas regiões onde ocorre menos movimento há uma resistência maior para o ruído.

Figura 17 – Mapa de K_k do frame 20 da sequência Salesman.



4.1.2 Ponderação entre o Filtro de Kalman e o Filtro Bilateral

Fazendo com que o filtro proposto seja menos intenso nas regiões com movimento (para convergir mais rapidamente a z_k) o ruído passa a ser, também, mais evidente nestas regiões. Seguindo a ideia de Zuo et al. (2013) podemos aplicar uma filtragem espacial nas regiões onde ocorre movimento.

Zuo et al. (2013) sugerem uma ponderação entre o frame obtido a partir do filtro de Kalman e de um filtro bilateral, com um peso baseado em uma distribuição Gaussiana conforme o movimento, medido através de algoritmos de detecção de movimento. Visto que um algoritmo de detecção de movimento é computacionalmente custoso e que K_k já é um ponderador entre z_k e x_k^- , K_k pode também fazer a ponderação entre o frame oriundo do filtro de Kalman x_k e o frame oriundo do filtro bilateral x_b , conforme a Equação 4.3.

$$X_k = [I - K_k].x_k + K_k.x_b \quad (4.3)$$

Considerando o filtro com as alterações propostas nesta Seção e na anterior (4.1.1), o resultado obtido pelo filtro depende da escolha dos parâmetros Q e R . As figuras 18 e 19 mostram os valores de PSNR e SSIM, respectivamente, para a sequência Salesman, para diversos valores de Q e R . Pode-se notar pelo gráfico que existem valores de Q e R que maximizam o PSNR e o SSIM. No entanto, os mesmos pares Q e R não maximizam simultaneamente o PSNR e o SSIM. Da mesma forma, um mesmo par de valores Q e R , tal que o PSNR seja máximo para uma sequência de imagens, não necessariamente será o máximo para outra sequência de imagem.

Os filtros apresentados na Seção 2.4 e o filtro bilateral foram testados para desempenhar a filtragem espacial. No entanto, o filtro bilateral, em conjunto com o filtro de Kalman, apresentou os melhores resultados, motivando a escolha do bilateral para compor o STMKF.

4.1.3 Automatização da escolha do parâmetro R

De fato, a Equação 4.1 faz o Filtro de Kalman convergir mais rápido. No entanto, ainda faz-se necessária a escolha dos parâmetros Q e R . Para simplificar, a escolha de R pode ser automatizada pela Equação 4.4.

Figura 18 – PSNR para a sequência Salesman em termos de Q e R .

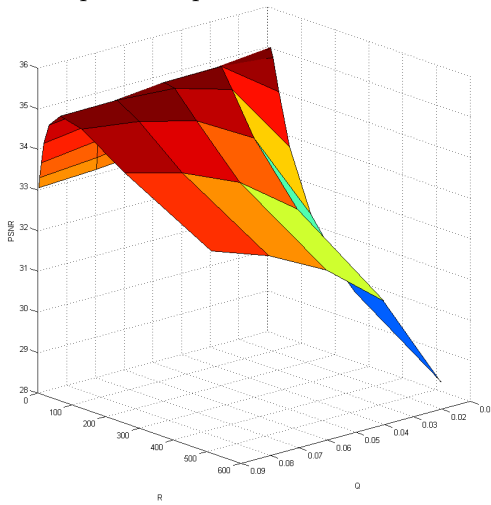
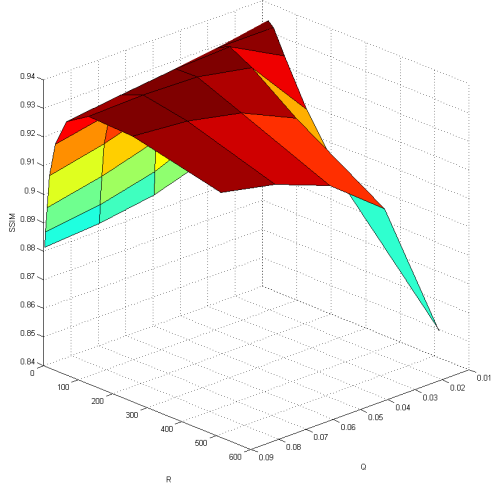


Figura 19 – SSIM para a sequência Salesman em termos de Q e R .



$$R = 1 + R(R + K_k)^{-1} \quad (4.4)$$

Assim, R se ajusta as diferentes regiões, com e sem movimento, baseado em K_k .

4.2 VISÃO GERAL DO ALGORITMO STMKF

A Figura 20 mostra o diagrama das principais operações executadas pelo STMKF a cada frame. A partir de cada novo frame com ruído (Z_k) que é servido de entrada no algoritmo, são computadas cinco operações principais:

Blur Filter: é computada a média da vizinhança, com o filtro *blur*. O frame com *blur* atual b_k é passado para computar Δ e armazenado para ser utilizado no próximo instante $k + 1$;

Delta: a partir do frame com *blur* do instante atual b_k e do frame com *blur* do instante anterior b_{k-1} é computado Δ .

Filtro Bilateral: é computado o Filtro Bilateral a partir do frame com ruído z_k ;

Filtro de Kalman: é computado o Filtro de Kalman, com as devidas modificações, utilizando como entrada o frame com ruído z_k , Δ e o frame resultante do instante anterior X_{k-1} .

Ponderação (*Weighting*): a partir dos resultados do Filtro Bilateral x_b e do Filtro de Kalman modificado x_k , utilizando a constante de Kalman K_k como peso, é computado o frame resultante X_k .

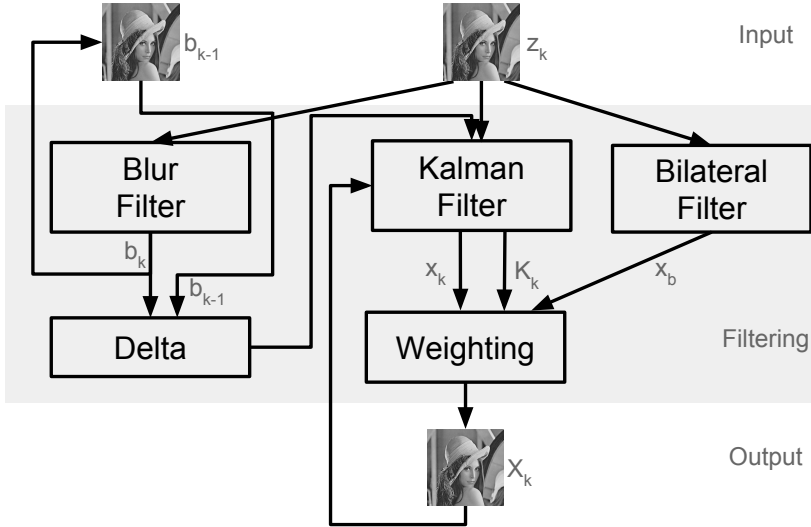
O STMKF não utiliza algoritmos de detecção de movimento por causa da alta complexidade computacional desta família de algoritmos. A complexidade computacional de cada passo do STMKF é $O(n)$, onde n é o número de pixels de um frame. Assim, a complexidade do STMKF é também $O(n)$ para cada frame.

4.3 ALGORITMO SEQUENCIAL

O Algoritmo 1 apresenta o pseudocódigo do STMKF. O processo principal (STMKF-MAIN) inicializa todas as matrizes utilizadas por STMKF e chama a função (STMKF-CORE) para cada novo frame. As variáveis Q , x , x^- , *previousBlured*, K , P e P^- são matrizes bidimensionais com as mesmas dimensões de um frame do vídeo a ser processado. Todas as operações entre as matrizes ocorrem elemento a elemento.

A matriz Q é inicializada com o valor q fornecido pelo usuário (linha 2), as matrizes x e *previousBlured* são inicializadas com 0, a

Figura 20 – Diagrama do algoritmo STMKF proposto.



matriz K é inicializada com 0.5 e as matrizes P^- , P e R são inicializadas com 1.

Após a inicialização, STMKF-MAIN pega o próximo frame (*frameIN*) do vídeo a ser filtrado (linha 6). O frame é utilizado como entrada dos filtros *blur* e bilateral (linhas 8 e 9). O resultado dos filtros blur e Bilateral são armazenados, respectivamente, em duas novas matrizes: *blured* e *bf*. Por fim, o frame original (*frameIN*), *blured* e *bf* são repassados como parâmetros de entrada para a função principal STMKF-CORE, que computa o STMKF em si, e irá retornar o frame resultante, com ruído atenuado. Este frame é armazenado na matriz *frameOUT*, que pode ser gravado como arquivo ou exibido na tela. Este processo é repetido a cada frame do vídeo de entrada.

O algoritmo STMKF foi implementado em C++, com auxílio da biblioteca *Open Source Computer Vision Library* (OpenCV) (PULLI et al., 2012) para processar as operações com matrizes descritas no Algoritmo 1. OpenCV é uma biblioteca que possui abstrações altamente otimizadas e funções que facilitam o desenvolvimento de aplicações de visão computacional em tempo real. Além das funções de computação de matrizes, OpenCV também inclui implementações eficientes dos filtros *blur* e bilateral. Estas implementações foram utilizadas para cal-

Algorithm 1 Sequential Algorithm

Global: *Matrices previousBlured, R, K, x⁻, x, P⁻, P, Q*

```

1: procedure STMKF-MAIN(video, q)
2:    $Q \leftarrow q$ 
3:    $x, \text{previousBlured} \leftarrow 0$ 
4:    $K \leftarrow 0.5$ 
5:    $P^-, P, R \leftarrow 1$ 
6:    $\text{frameIN} \leftarrow \text{GETFRAME}(\text{video})$ 
7:   repeat
8:      $\text{blured} \leftarrow \text{BLUR-FILTER}(\text{frameIN})$ 
9:      $\text{bf} \leftarrow \text{BILATERAL-FILTER}(\text{frameIN})$ 
10:     $\text{frameOUT} \leftarrow \text{STMKF-CORE}(\text{frameIN}, \text{blured}, \text{bf})$ 
11:     $\text{frameIN} \leftarrow \text{GETFRAME}(\text{video})$ 
12:  until  $\text{frameIN} = \text{NULL}$ 
13: end procedure
14: function STMKF-CORE(frame, blured, bf)
15:    $\Delta \leftarrow \text{previousBlured} - \text{blured}$ 
16:    $\text{previousBlured} \leftarrow \text{blured}$ 
17:    $R \leftarrow 1 + R/(R + K)$ 
18:    $x^- \leftarrow x$ 
19:    $P^- \leftarrow P + \Delta^2 \cdot Q$ 
20:    $K \leftarrow P^-/(P^- + R)$ 
21:    $x \leftarrow x^- + K \cdot (\text{frame} - x^-)$ 
22:    $x \leftarrow (1 - K) \cdot x + K \cdot \text{bf}$ 
23:    $P \leftarrow (1 - K) \cdot P^-$ 
24:   return  $x$ 
25: end function

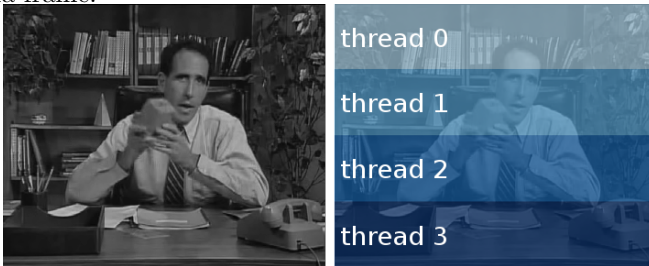
```

cular as matrizes de *blured* e *bf* na implementação do STMKF.

4.4 ALGORITMO PARALELO (MT-STMKF)

Para explorar o máximo dos processadores *multicore* atuais foi implementada uma versão paralela do STMKF (MT-STMKF). Foi adotada uma estratégia simples de repartir virtualmente a imagem em t blocos, onde t representa o número de *threads* disponíveis. Então, cada uma das *threads*, em paralelo, executam o STMKF-CORE sobre o seu respectivo bloco. A Figura 21 mostra um exemplo da estratégia de paralelização proposta utilizando 4 *threads*, onde o frame é dividido em 4 partes.

Figura 21 – Ilustração da abordagem paralela adotada em MT-STMKF. Este exemplo considera 4 *threads*, cada uma encarregada de filtrar 1/4 de cada frame.



Para desenvolver a paralelização foi utilizada a Interface de Programação de Aplicativo (*Application Programming Interface* - API) chamada *Open Multi-Processing* (OpenMP) (OpenMP Architecture Review Board, 2013). O OpenMP permite adicionar diretivas específicas para o compilador (*compiler directives*), no código original, para determinar partes do código que devem ser executadas em paralelo (região paralela).

O OpenMP segue um modelo *fork-join*: a execução inicia com uma *thread* principal (execução sequencial). Em tempo de execução, quando a *thread* atinge uma região paralela, o OpenMP cria um conjunto de novas *threads* que executarão o código da região em paralelo. Então, todas as *threads* sincronizam ao final da região paralela e a *thread* principal reassume a execução sequencial do restante do algoritmo.

O OpenMP permite uma implementação da estratégia de paralelização descrita acima. Foram adicionadas diretivas de compilador

`#pragma omp parallel for private(i,j)`, onde i e j são variáveis que operam sobre dois laços *for* aninhados, sobre a altura e a largura da imagem, respectivamente, de forma que o STMKF-CORE é executado em uma região paralela. Cada bloco da imagem é associado a uma *thread* que executa o STMKF-CORE, utilizando seu bloco da imagem como entrada. A mesma estratégia poderia ser aplicada ao filtro *blur* e bilateral. No entanto, o OpenCV já implementa paralelismo para estes filtros e este retrabalho não se faz necessário.

Para fazer uso da paralelização através do OpenMP, ao compilar, além das flags do próprio OpenCV, foram adicionadas as seguintes *flags* de compilação: `-fopenmp -O3`. A primeira *flag* ativa o próprio OpenMP, enquanto que a segunda *flag* habilita otimizações no código que favorecem o paralelismo.

4.5 IMPLEMENTAÇÃO PARA GPU (GPU-STMKF)

As Unidades de Processamento Gráfico (*Graphics Processing Unit* - GPU) são processadores de muitos núcleos (*manycore*) otimizados para computação paralela. O seu design é focado em processamento de grande volume de dados por um grande número de *threads*. O design orientado a taxa de vazão (*throughput-oriented*) prioriza a vazão (*throughput*) de um grande número de *threads* em relação a *threads* individuais, onde o tempo de execução será potencialmente maior (KIRK; HWU, 2010).

O desenvolvimento para GPUs requer um linguagem de programação paralela específica, como o CUDA (*Compute Unified Device Architecture*), que é específica para GPUs NVIDIA, e OpenCL (*Open Computing Language*), que está disponível para uma vasta gama de processadores e GPUs. No entanto, o desenvolvimento com CUDA ou OpenCL não é trivial, visto que é necessária a atenção a detalhes como sincronização, transferência de dados entre GPU e memória principal e compartilhamento de memória. Convenientemente, o OpenCV disponibiliza módulos para facilitar a utilização de GPUs, provendo um controle explícito sobre a transferência de dados entre a memória principal (dita memória da CPU) e a memória da GPU. Apesar da necessidade de algum código adicional para fazer uso da GPU, esta abordagem é igualmente flexível e permite uma computação mais eficiente.

Os recursos disponíveis no OpenCV foram utilizados para o desenvolvimento da versão para GPU do STMKF (GPU-STMKF). O GPU-STMKF é basicamente uma implementação da função STMKF-CORE para

GPU. Aqui também não se fez necessária a implementação dos filtros *blur* e Bilateral para GPU por já estarem disponíveis no OpenCV. No entanto, alguns passos adicionais são necessários, a cada novo frame, para filtrar, no GPU-STMKF:

1. Transferir o frame da memória principal para a memória da GPU. Este processo é conhecido como *upload*;
2. Processar o *blur*, Bilateral e o STMKF-CORE na GPU;
3. Transferir o frame resultante da memória da GPU de volta para a memória principal. Este processo é conhecido como *download*.

Em uma primeira implementação do GPU-STMKF, as transferências entre a memória principal e a GPU e a computação na GPU foram executadas totalmente de forma síncrona. Ou seja, o frame z_{k+1} somente é transferido para a GPU quando o download do frame X_k for completado.

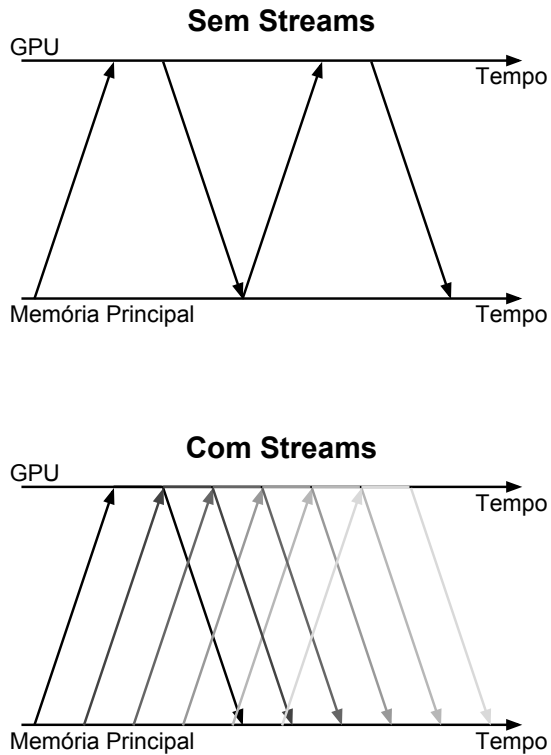
Para esconder os tempos das transferências, foi desenvolvida uma segunda versão do GPU-STMKF, onde elas ocorrem de forma assíncrona, através de CUDA *streams*. Assim é possível transferir z_{k+1} para a GPU simultaneamente ao processamento de X_k e a transferência de X_{k-1} da GPU para a memória principal.

A Figura 22 ilustra a diferença entre as duas implementações para GPU, onde cada seta representa a transferência de um frame entre a memória principal e a GPU. No topo, a primeira implementação, sem *streams*. Na base, a segunda implementação, com *streams*.

4.6 SOBRE AS IMPLEMENTAÇÕES DESENVOLVIDAS

O algoritmo foi modelado, a partir do Filtro de Kalman e Bilateral, para reduzir o ruído em vídeos de maneira eficaz, e desenvolvido, em suas versões sequencial, paralela e para GPU, para explorar a capacidade dos processadores modernos e alcançar velocidade de filtragem compatível com aplicações de tempo real. Os códigos de cada uma das versões está disponível, sob os termos da licença *GNU General Public License*, em <https://github.com/sergiogenilson/STMKF>. Para verificar se está de acordo com os objetivos estipulados, experimentos foram realizados, conforme segue.

Figura 22 – Diferenças entre as versões com *Streams* e sem *Streams* do algoritmo para GPU



5 RESULTADOS EXPERIMENTAIS

Para mostrar as qualidades do STMKF, neste trabalho, considera-se a avaliação do filtro em termos da qualidade dos vídeos resultantes (eficácia) e a velocidade de filtragem (desempenho), nas seções que seguem, respectivamente.

5.1 EFICÁCIA DO STMKF

Para mensurar a eficácia do STMKF utilizou-se as métricas PSNR e SSIM, descritas na Seção 2.7. A análise da qualidade do vídeo resultante foi decomposta em três partes: comparação com os filtros de Kalman e do filtro Bilateral, que compõem o STMKF; a análise visual dos resultados; e a comparação com os demais filtros, descritos no Capítulo 3.

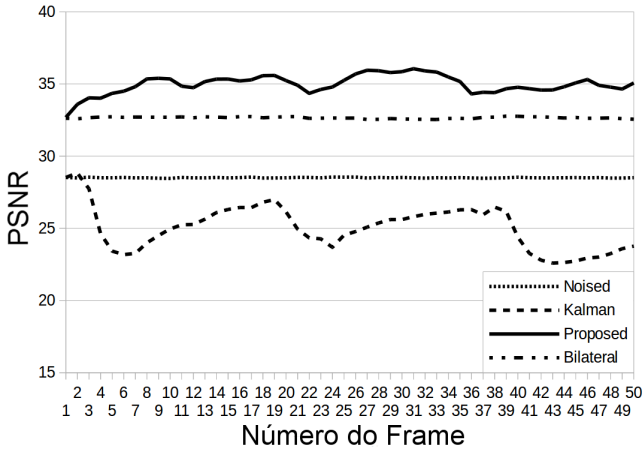
De uma forma geral, a análise da qualidade de um filtro requer que se tenha a imagem sem ruídos, à qual será comparada a imagem resultante do filtro. As métricas PSNR e SSIM tem isto como requisito. A inspeção visual, por ser subjetiva, não tem a necessidade da imagem original sem ruídos, mas tê-la para comparação é interessante. A forma convencional de se obter este par de imagens se faz pela utilização de uma imagem originalmente sem ruídos, à qual é aplicada o ruído desejado, seguido do processo de filtragem.

Para o estudo da eficácia do STMKF aplicou-se, aos vídeos de referência, ruído gaussiano. O ruído gaussiano é o mais utilizado nos trabalhos correlatos e o seu uso facilita nos comparativos com os demais. No entanto, o STMKF é aplicável a qualquer tipo de ruído que satisfaça as condições para o uso do Δ (o ruído possui dimensão igual a 1 píxel, independente dos vizinhos e uniformemente distribuído), como o ruído de disparo.

5.1.1 Comparação do STMKF com os filtros que o compõem

A Figura 23 e a Figura 24 apresentam os valores de PSNR e SSIM, respectivamente, para o STMKF, comparado com o Filtro de Kalman e o Filtro Bilateral, para cada frame da sequência Salesman. À sequência original adicionou-se um ruído Gaussiano de média 0 e desvio padrão $\sigma = 10$. Em seguida aplicou-se o filtro. O STMKF supera

Figura 23 – PSNR para cada um dos frames da sequência Salesman com ruído ($\sigma = 10$). PSNR Médio= 34,98



os resultados dos filtros Bilateral e de Kalman, mostrando que a fusão das duas técnicas traz resultados melhores. É interessante notar que, pela métrica PSNR, o Filtro de Kalman, por si só, estraga a imagem resultante. No entanto, as suas modificações e a combinação com o filtro Bilateral traz resultados melhores em relação aos filtros originais.

5.1.2 Análise visual dos resultados

Para uma análise visual dos resultados do STMKF, a Figura 25 mostra o resultado do STMKF para o 25º frame da sequência Salesman, com $\sigma = 10$. No topo, o frame original, o frame com ruído e o frame resultante do Filtro Bilateral. Na base, o frame resultante do Filtro de Kalman, o frame resultante do STMKF e o mapa K_k . O mapa K_k mostra que o parâmetro K_k varia de acordo com as regiões com (mais claras) e sem (mais escuras) movimento. Nas regiões mais claras, o Filtro Bilateral recebe um peso maior. Nas regiões mais escuras, o Filtro de Kalman recebe um peso maior.

É observável na Figura 25 que o STMKF consegue reunir as qualidades do filtro Bilateral e do filtro de Kalman, deixando as superfícies uniformes da imagem mais homogêneas e mantendo a qualidade das bordas, mesmo nos objetos em movimento.

Um fator visualmente importante ao se filtrar uma imagem,

Figura 24 – SSIM para cada um dos frames da sequência Salesman com ruído ($\sigma = 10$). SSIM Médio = 0.930

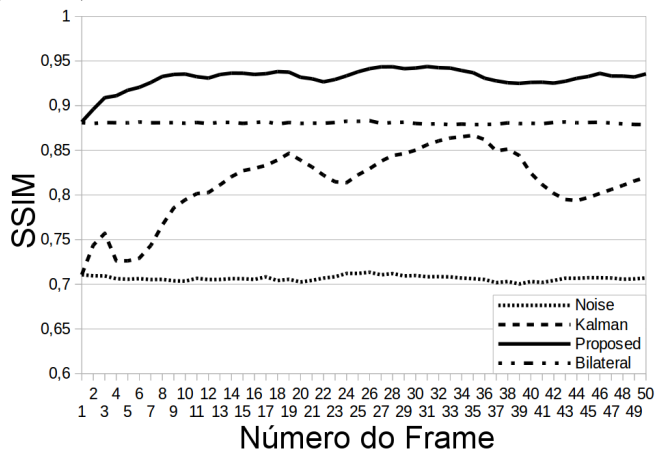


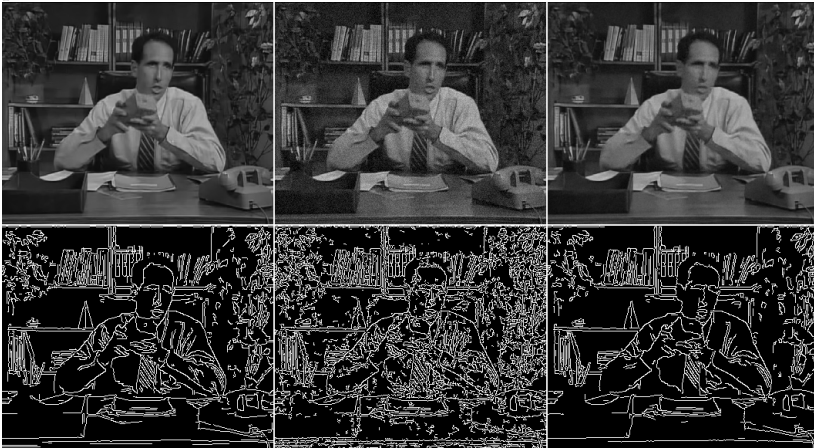
Figura 25 – Resultados do STMKF para a sequência Salesman com ruído Gaussiano com desvio padrão $\sigma = 10$. Topo: frame original, frame com ruído e frame com filtro bilateral. Base: frame com o filtro de Kalman, frame com STMKF e o mapa de K_k



tanto para visão humana quanto para a visão computacional, é a qualidade das bordas na imagem. O ruído tende a deteriorar as bordas.

A Figura 26 mostra o detector de bordas Canny aplicados sobre um frame original da sequência Salesman, ao lado de um frame ruidoso e outro com o STMKF aplicado. Pode-se observar que o STMKF fez com que as bordas do frame filtrado se aproximassem das bordas detectadas no frame original.

Figura 26 – Comparação da detecção de bordas Canny. Topo: frame original, frame com ruído e frame filtrado pelo STMKF. Base: Canny do frame original, Canny do frame com ruído e Canny do frame filtrado pelo STMKF



5.1.3 Comparativo do STMKF com os demais filtros

Para um comparativo com os demais filtros apresentados no Capítulo 3, o STMKF foi aplicado sobre diversos vídeos comuns na literatura e disponíveis em diversos repositórios.

A Tabela 3 mostra os resultados em termos de PSNR. A cada sequência original adicionou-se um ruído Gaussiano de média 0 e desvio padrão $\sigma = 10$ ou $\sigma = 20$. Os resultados são comparados com os algoritmos 3D Rational (COCCHIA; CARRATO; RAMPONI, 1997), 3D KNN (ZLOKOLICA; PHILIPS; VILLE, 2002), Pizurica (PIZURICA; ZLOKOLICA; PHILIPS, 2004) e IFSM (RAHMAN; AHMAD; SWAMY, 2007). Pode-se observar que o STMKF supera os demais filtros para as sequências Salesman e Football. Nas sequências Tennis e Coastguard o STMKF fica

atrás apenas do IFSM.

Tabela 3 – Resultados em PSNR comparados com outros algoritmos. Dados para 3D KNN, Pizurica e IFSM foram extraídos de Rahman, Ahmad e Swamy (2007).

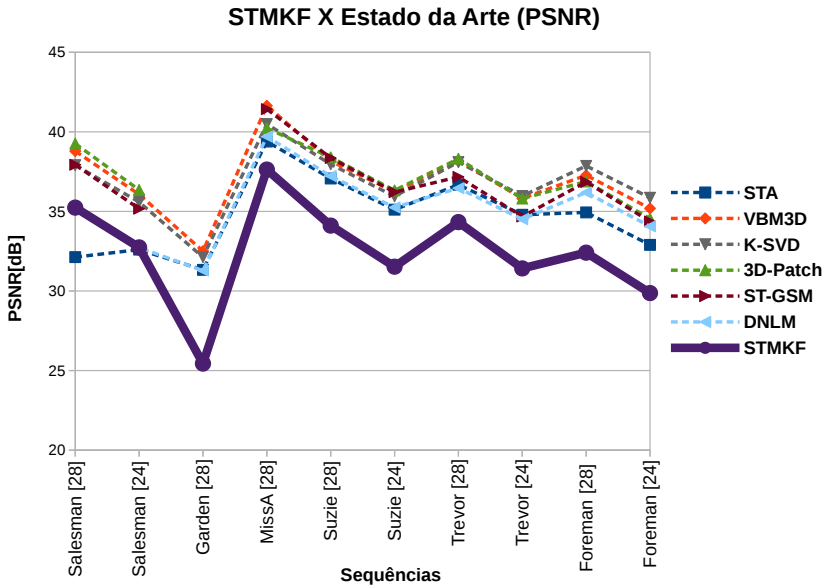
Sequência	σ	3D Rational	3D KNN	Pizurica	IFSM	STMKF
Tenis	10	29.86	29.14	30.76	32.05	31.15
Salesman	10	31.65	32.13	34.11	34.41	35.24
Coastguard	20	27.23	26.62	27.42	28.40	27.55
Football	20	26.39	24.95	26.62	27.06	27.78

A Figura 27 também mostra os resultados em termos de PSNR, no entanto, a cada sequência adicionou-se ruído Gaussiano de média 0 e desvio padrão σ de modo que o PSNR dos frames ruidosos fosse 24 e 28 (*Input* PSNR [dB]). Os resultados são comparados com os algoritmos STA (BOULANGER; KERVRANN; BOUTHEMY, 2007), VBM3D (DABOV; FOI; EGIAZARIAN, 2007), K-SVD (PROTTER; ELAD, 2009), 3D-Patch (LI; ZHENG, 2009), ST-GSM (VARGHESE; WANG, 2010) e DNLM (HAN; CHEN, 2012). Estes filtros compõem o estado da arte em qualidade e observa-se que os resultados do STMKF estão próximos aos demais resultados, mas não os supera. Os valores exatos que compõem o gráfico da Figura 27 podem ser encontrados na Tabela 7 do Apêndice A.

A Figura 28 mostra os resultados experimentais em termos de SSIM. A cada sequência original adicionou-se um ruído Gaussiano de média 0 e desvio padrão $\sigma = 10, 15$ e 20. Os resultados do STMKF são comparados com os algoritmos WRSTF (ZLOKOLICA; PIZURICA; PHILIPS, 2006), SEQWT (PIZURICA; ZLOKOLICA; PHILIPS, 2003), 3DWTF (SELESNICK; LI, 2003), IFSM (RAHMAN; AHMAD; SWAMY, 2007), 3DSWCDT (RUSANOVSKYY; EGIAZARIAN, 2005), VBM3D (DABOV; FOI; EGIAZARIAN, 2007), ST-GSM (VARGHESE; WANG, 2010) e DNLM (HAN; CHEN, 2012). Por esta métrica observa-se que o STMKF se equipara a alguns filtros, vence em alguns casos e perde em outros. Os valores exatos que compõem o gráfico da Figura 28 podem ser encontrados na Tabela 8 do Apêndice A.

De uma forma geral, o STMKF é competitivo com diversos outros filtros encontrados na literatura para ambas as métricas, especialmente para vídeos com a imagem de fundo estáticas. A sequência Salesman é um exemplo de caso onde o STMKF se sobressai em relação a maioria dos filtros, ficando bastante próximo aos filtros que representam o estado da arte, na métrica SSIM. Pela métrica PSNR, o STMKF se distancia um pouco dos resultados dos filtros do estado da arte, mas ainda com resultados satisfatórios.

Figura 27 – Resultados em termos de PSNR comparados com outros trabalhos. Dados dos algoritmos STA, VBM3D, K-SVD, 3D-Pach, VBM3D, ST-GSM e DNLM foram extraídos de Han e Chen (2012)

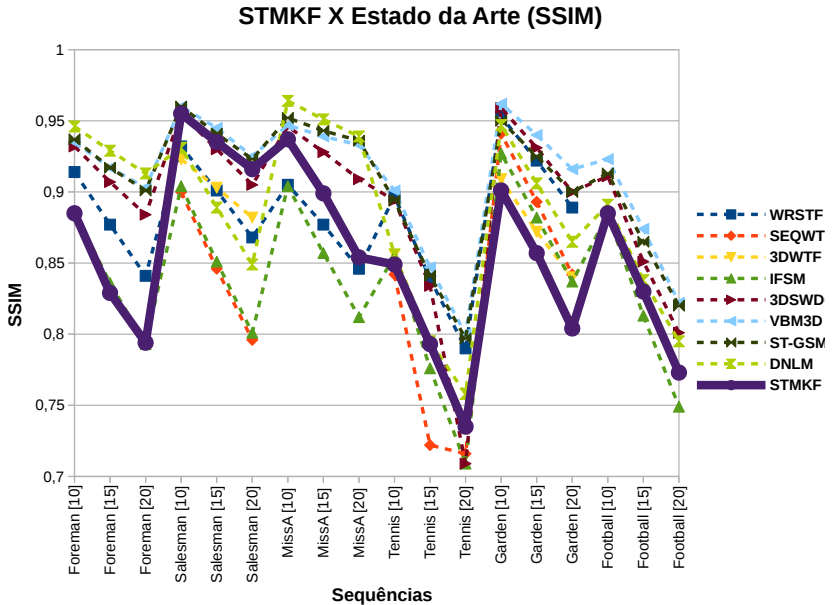


Para vídeos onde há movimento em muitas regiões, como o caso da sequência Garden, o STMKF é superados pelos demais filtros. O principal motivo deste fato são as técnicas de detecção e/ou compensação de movimento. Como mencionado anteriormente, estas técnicas aumentam a complexidade do algoritmo ao mesmo tempo que melhoram os resultados. No entanto, o resultado depende de quão bem estas técnicas podem estimar o movimento. A sequência Football, por exemplo, possui muitas áreas com movimento e o STMKF consegue resultados melhores que os filtros 3D KNN, Pizurica e IFSM. Neste caso específico, o movimento não pode ser predito com facilidade, dificultando sua estimativa e/ou compensação destas outras técnicas.

Os resultados se apresentam diferentes nas duas métricas devido as características das métricas discutidas na Seção 2.7. Assim, a qualidade do vídeo observado pode divergir da qualidade mensurada pelas métricas.

É importante notar que a qualidade dos resultados obtidos com

Figura 28 – Resultados em termos de SSIM comparados com outros trabalhos. Dados dos algoritmos WRSTF, SEQWT, 3DWTF, IFSM, 3DSWDCT, VBM3D, ST-GSM and DNLM foram extraídos de Han e Chen (2012)



o STMKF depende da escolha adequada do parâmetro Q . Um valor para Q que maximize o PSNR não necessariamente maximiza o SSIM. De forma análoga, um valor de Q que maximize os resultados para uma sequência específica não necessariamente maximiza os resultados para outra sequência.

O propósito do STMKF não é prover somente o melhor resultado, mas sim ser um filtro eficaz dentro das restrições de tempo. A avaliação da performance temporal é realizada na Seção a seguir.

5.2 AVALIAÇÃO DO DESEMPENHO

Nesta seção será avaliada a performance das implementações paralelas do STMKF em processadores multicore e GPUs. Inicialmente

serão descritas as plataformas e as resoluções de vídeos utilizadas nos experimentos. Em seguida, serão discutidos os resultados obtidos com o MT-STMKF (em multicores) e o GPU-STMKF (em GPUs).

5.2.1 Plataformas e Resoluções dos Vídeos

O desempenho das implementações do STMKF foram avaliadas em quatro plataformas. Estes processadores representam três diferentes classes: processadores de propósito geral (Intel i7), processadores embarcados (ARM Cortex-A7) e processadores manycores (GPUs). Os processadores ARM são comumente encontrados em sistemas embarcados. No contexto de processamento de imagens em tempo real, sistemas embarcados tem se tornado muito relevantes, com diversas aplicações, incluindo dispositivos portáteis, câmeras inteligentes e robótica (JUNG; SUKHATME, 2004; WOLF; OZER; LV, 2002). A Tabela 4 apresenta as principais especificações de cada um dos processadores considerados.

Tabela 4 – Especificação das plataformas paralelas.

Tipo	Modelo	Cores	Clock	Memória	SO
CPU	Intel Core i7-4710MQ	4 + HT	2.5 GHz	16 GB	Ubuntu 15.04
	ARM Cortex-A7	4	900 MHz	1 GB	Raspbian 7
GPU	NVIDIA Tesla K40	2,880	745 MHz	12 GB	Ubuntu 14.04.3
	NVIDIA GeForce GTX 850M	640	876 MHz	2 GB	Ubuntu 15.04

Considerando que o processador Intel i7 possui *Hyper-Threading Technology* (HT), para este processador, nos experimentos foram utilizados até 8 *threads* (um para cada núcleo virtual). Para o processador ARM Cortex-A7, no entanto, o número de *threads* foi limitado ao número de núcleos físicos (4 núcleos). Para as GPUs, foram utilizados todos os núcleos disponíveis.

Para avaliar o desempenho destas plataformas foram utilizados diversos vídeos. Como o objetivo desta Seção é avaliar a velocidade do STMKF, a principal característica a ser observada é o tamanho do vídeo. Assim, foram produzidos cinco vídeos nas dimensões comumente encontradas atualmente. A Tabela 5 apresenta as resoluções dos vídeos utilizados nos experimentos.

Todos os resultados apresentados nas próximas subseções foram computados baseados nos tempos médios calculados a partir de 30 execuções e apresentam um nível de confiança estatística de 95% na distribuição t de Student e um erro relativo máximo de 0,7%.

Tabela 5 – Especificação das resoluções dos vídeos.

Resolução	Dimensões (px)	Número de píxeis
240p	240 × 360	86.400
480p	480 × 720	345.600 ou ~0.3MP
720p ou HD	720 × 1.280	921.600 ou ~0.9MP
1080p ou FullHD	1.080 × 1.920	2.073.600 ou ~2MP
4k	2.160 × 3.840	8.294.400 ou ~8MP

5.2.2 Resultados para Versão Sequencial

O algoritmo proposto foi implementado em C++ e utiliza o OpenCV, como descrito anteriormente. Utilizando o computador com o Intel i7 e destinando-se apenas uma *thread* ao STMKF, para remover o ruído de Salesman com 352 x 288 e 50 frames são necessários aproximadamente 220 milissegundos. O algoritmo VBM3D, que possui uma implementação em Matlab fornecida pelo autor através do seu *website*, também foi executado na mesma máquina para a mesma sequência, e foram necessários 11.1 segundos para remover o ruído (50,4 vezes mais lento).

Embora a comparação entre os tempos dos algoritmos utilizando linguagens e/ou máquinas diferentes não seja, nem de perto, a ideal, há uma grande dificuldade de se conseguir os demais algoritmos, pois a maioria dos autores não os disponibiliza. Assim, a Tabela 6 mostra um comparativo do tempo de execução, por frame, de vários algoritmos. Os tempos listados na primeira seção da Tabela são os tempos anunciados pelos próprios autores dos respectivos algoritmos. Cada autor utilizou uma máquina diferente e linguagens de programação diferentes. Na segunda seção da Tabela, estão listados os tempos apresentados por outro autor. Na terceira seção, os tempos obtidos por códigos disponibilizados pelos autores, executados no Intel i7. Para fins de comparação, o STMKF foi executado em uma única *thread* do processador Intel i7 e em uma única *thread* no processador ARM Cortex-A7.

Visto que, mesmo no processador ARM Cortex-A7, cujo foco é economia de energia em detrimento ao desempenho, o STMKF é 7× mais rápido que os tempos anunciados pelos demais autores, é possível se extrair da Tabela 6 um indicativo de que o STMKF seja efetivamente mais rápido do que os demais. Comparando o desempenho dos algoritmos IFSM, VBM3D e STMKF sob o Intel i7, o STMKF se mostrou 55.5× mais rápido que o VBM3D e 15.25× mais rápido que o IFSM.

No entanto, devido as diferenças de linguagens e sistemas operacionais, isto não pode ser tomado como uma verdade absoluta.

Tabela 6 – Tempo de processamento para diversos algoritmos, fornecido pelos respectivos autores, comparado com a versão *single thread* do STMKF

Método	Tempo (s/frame)	Tamanho do frame	Linguagem/ Computador
ST-GSM	120s	352 × 288	MATLAB, Intel 2.4GHz
STA	60s	384 × 228	8x3 GHz
3D-Patch	1s	352 × 288	MATLAB/C
WRSTF	0.7s	352 × 288	C++, Linux, Athlon64, 2.4GHz
K-SVD	5-120s	360 × 280	MATLAB, Pentium, 2.4GHz
VBM3D	0.7s	352 × 288	MATLAB/C, Core solo, 1.8GHz
DNLM	200s	176 × 144	MATLAB, Intel 2x2GHz
SEQWT ¹	0.814s	352 × 288	C++, Athlon64, 2.4GHz
WRSTF ¹	0.866s	352 × 288	C++, Athlon64, 2.4GHz
IFSM ²	0.061s	352 × 240	MATLAB, i7-4710mq, 2.5GHz
VBM3D ³	0.222s	352 × 288	MATLAB, i7-4710mq, 2.5GHz
STMFK	0.004s	352 × 288	C++, Linux, i7-4710mq, 2.5GHz
	0.1s	352 × 288	C++, Linux, ARM Cortex-A7, 900MHz

A Figura 29 mostra o tempo requerido pelo STMKF para processar um frame, utilizando uma *thread* do Intel i7, para os diversos vídeos da Tabela 5. Pode se observar que o tempo aumenta linearmente com o número de pixels, conforme o esperado para um algoritmo com complexidade $O(n)$.

5.2.3 Resultados para Processadores Multicore

O desempenho do MT-STMKF foi analisado nos dois processadores multicore apresentados na Subseção 5.2.1 (Intel i7 e ARM Cortex-A7). Para avaliar o desempenho foram avaliadas as métricas *speedup* e *frame rate*.

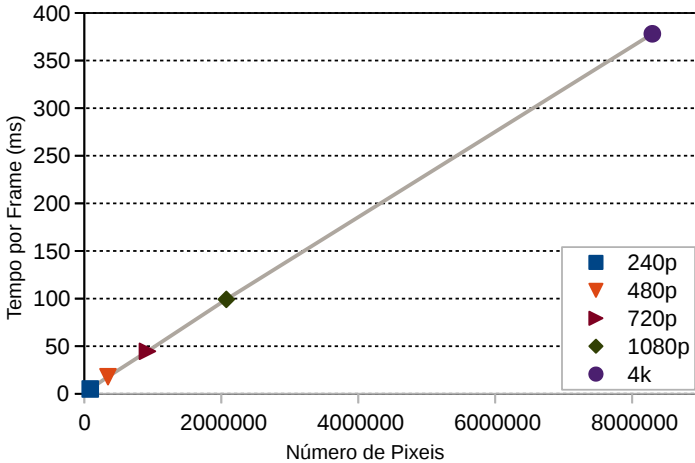
A métrica *frame rate* é a frequência com a qual um algoritmo processa quadros (*frames*) consecutivos e é expressado em *frames* por segundo (FPS).

¹Resultados disponibilizados por Zlokolica, Pizurica e Philips (2006)

²Código fonte disponível em http://teacher.buet.ac.bd/mahbubur/resources/TCSTV_prog.rar

³Código fonte disponível em <http://www.cs.tut.fi/~foi/GCF-BM3D/index.html>

Figura 29 – Tempo necessário para processar um frame, para diversas resoluções



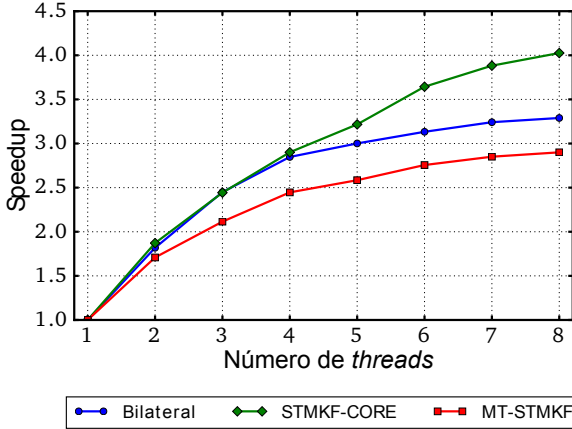
A métrica *speedup*, por outro lado, representa a razão entre o tempo de execução da versão sequencial e a versão multicore, com um número específico de *threads*. É importante salientar que o máximo *speedup* obtido por uma solução paralela é limitado à porção do código que efetivamente pode ser paralelizado. Ou seja, no melhor caso, quando 100% do código sequencial é paralelizado, o *speedup* cresce linearmente com o número de núcleos físicos. No entanto, uma escalabilidade linear é dificilmente alcançada, visto que a maioria das soluções paralelas contam com porções de código sequencial.

A Figura 30 apresenta o *speedup* da versão paralela do filtro Bilateral e do STMKF-CORE ao filtrar um vídeo 4K em um processador Intel i7. É apresentado, também, o *speedup* do MT-STMKF como um todo.

Como pode ser observado, o filtro Bilateral, que é paralelizado pelo OpenCV, apresenta escalabilidade pior do que a implementação do STMKF-CORE. O MT-STMKF, que inclui a versão paralela do Bilateral e do STMKF-CORE, apresenta uma escalabilidade um pouco inferior ao Bilateral e o STMKF-CORE individualmente. Este comportamento é esperado e as principais razões são três:

1. MT-STMKF possui dependências (como o filtro Bilateral e o *blur*, por exemplo) que precisam ser computadas antes do STMKF-

Figura 30 – Escalabilidade do MT-STMKF versus as versões paralelas do filtro Bilateral e STMKF-CORE para um vídeo com resolução 4K no processador Intel i7.



CORE;

- existem porções do MT-STMKF que não podem ser paralelizadas;
- a implementação do filtro *blur* em paralelo, pelo OpenCV, não traz ganhos de desempenho em relação a versão sequencial.

A Figura 31 apresenta o *speedup* e a Figura 32 apresenta o FPS, ambos obtidos pelo MT-STMKF variando-se o número de *threads* e a resolução dos vídeos, nos processadores Intel i7 e ARM Cortex-A7. Não foram incluídos os vídeos com resolução 240p nos resultados do Intel i7 devido a pouca computação exigida para este processador. De forma análoga, os resultados para os vídeos com resolução 4K não estão incluídos, devido a grande quantidade de computação exigida, para os processadores ARM Cortex-A7.

De uma forma geral, o MT-STMKF apresentou um *speedup* significativo quando aumenta-se o número de *threads*. O MT-STMKF atingiu um *speedup* de $2.9\times$ utilizando oito *threads* no processador Intel i7. De forma similar, o MT-STMKF atingiu um *speedup* de $2.3\times$ utilizando quatro *threads* no processador ARM Cortex-A7.

Apesar dos dois processadores apresentarem *speedups* significativos e similares, há uma considerável diferença no que diz respeito ao FPS, devido aos diferentes propósitos de cada processador. Com o Intel

i7, o MT-STMKF atingiu um máximo de 138 FPS para vídeos 480p e um máximo de 7.8 FPS para vídeos 4K. Já no ARM Cortex-A7, cujo foco é mais voltado ao consumo de energia do que desempenho, o MT-STMKF atingiu 18 FPS para vídeos 240p e 1.3 FPS para vídeos 1080p.

5.2.4 Resultados para GPUs

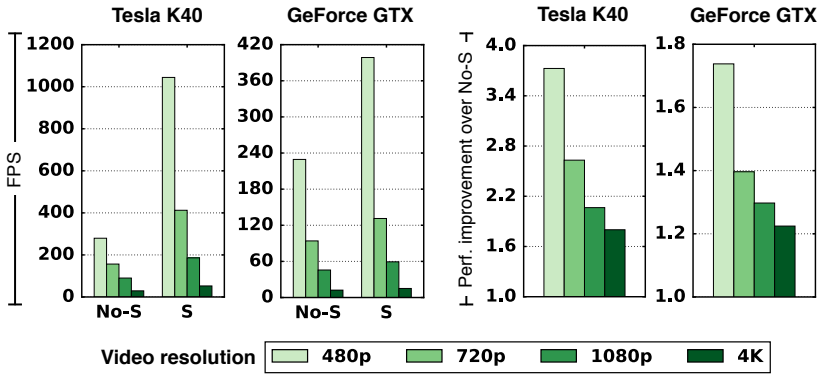
O desempenho da implementação do STMKF para GPU (GPU-STMKF) foi avaliada nas duas GPUs apresentadas na Subseção 5.2.1 (GeForce GTX e Tesla K40). De uma forma geral, são conhecidas pela sua elevada capacidade de processamento de imagens em tempo real (SEILLER; SINGHAL; PARK, 2010; PAUWELS et al., 2012). Assim, espera-se um ganho de desempenho considerável comparado a processadores multicore.

A Figura 33 apresenta o resultado obtido por ambas as GPUs, para diversas resoluções de vídeos. Conforme exposto na Seção 4.5, a primeira implementação de GPU-STMKF (chamada de No-S na Figura 33) faz transferências síncronas de dados entre a memória principal e a GPU. A desvantagem desta abordagem é que um quadro somente pode ser transferido para ser processado quando o quadro anterior já tiver retornado da GPU. A segunda implementação de GPU-STMKF (chamada de S na Figura 33) é uma melhoria do primeiro, onde as transferências de dados entre o hospedeiro e a GPU ocorrem simultaneamente com os processamentos na GPU, através de transferências de dados assíncronas utilizando *CUDA Streams*.

O GPU-STMKF com *CUDA Streams* (S), como se espera, resulta em um desempenho superior, em relação a versão síncrona (No-S), em ambas as GPUs. Usando a resolução 480p como exemplo, o FPS subiu de 279 para 1.044 na Tesla K40 e de 229 para 398 na GeForce GTX. Ou seja, houve um aumento de desempenho de $3,7\times$ e $1,7\times$ na Tesla K40 e na GeForce GTX, respectivamente. Ao aumentar a resolução, no entanto, é possível observar que o ganho de desempenho do GPU-STMKF com *CUDA Streams* reduz. Na resolução de vídeo máxima (4K) é observado um ganho de desempenho de $1,8\times$ e $1,2\times$ na Tesla K40 e na GeForce GTX, respectivamente.

Vale observar que o ganho de desempenho do GPU-STMKF com *CUDA Streams* é significativamente maior na Tesla K40 para todos os vídeos testados. Isto se deve ao fato de que a Tesla K40 possui um poder de processamento¹ aproximadamente $4\times$ maior que a GeForce

Figura 33 – Desempenho do GPU-STMKF com (S) e sem (No-S) *CUDA Streams*.



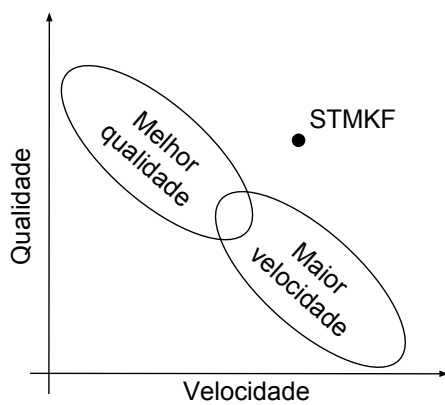
GTX. Em outras palavras, cada *frame* necessita aproximadamente 4× menos tempo para ser computado por GPU-STMKF na Tesla K40 do que na GeForce GTX. Devido a isto, para a versão síncrona do GPU-STMKF (No-S), a razão entre o tempo gasto nas transferências de dados entre o hospedeiro e a GPU e o tempo gasto na computação, na GPU, é muito maior na Tesla K40 do que na GeForce GTX. Assim, os benefícios da transferência de dados assíncrona com *CUDA Streams* são mais expressivos na Tesla K40.

Considerando ambas as abordagens, qualidade e velocidade, o STMKF se posiciona de forma satisfatória entre as duas. Apesar da dificuldade imposta pela não disponibilização dos demais algoritmos para uma avaliação mais precisa, a Figura 34 mostra a distribuição dos filtros relacionando a qualidade e a velocidade, de forma meramente ilustrativa, mostra onde o STMKF se posiciona em relação aos demais.

O STMKF cumpre assim os seus objetivos, apresentando-se rápido ao cumprir as limitações temporais e eficaz ao oferecer qualidade nos vídeos resultantes próxima aos filtros que compõem o estado da arte.

¹O poder de processamento de uma GPU é comumente medido pelo número de operações em ponto flutuante que podem ser computadas em um segundo (*Floating-point Operations Per Second* - FLOPS).

Figura 34 – Distribuição dos algoritmos em função da velocidade e qualidade.



6 CONCLUSÃO

Este trabalho apresentou um filtro para redução de ruídos em tempo real, com resultados qualitativos satisfatórios. O filtro STMKF desenvolvido mostrou-se competitivo em relação aos métodos atuais de filtragem, em relação a qualidade da imagem resultante e mostrou-se mais rápido que os demais filtros.

O STMKF demonstrou ser capaz de reduzir o ruído gaussiano e de disparo em vídeo que atrapalham a visão computacional e que também degradam a experiência do ser humano ao assistir um vídeo, mantendo também as bordas e detalhes da imagem, especialmente em vídeos com pouco movimento. Com a ponderação entre o Filtro Bilateral (que predomina sobre regiões com movimento) e o Filtro de Kalman (que predomina em regiões sem movimento), e dos aprimoramentos no filtro de Kalman para vídeos, o STMKF mostrou-se um filtro mais elaborado em relação as duas técnicas.

Com a capacidade de filtrar mais de 180 frames por segundo (FPS) de um vídeo FullHD (1080p) em GPU ou entre 29 e 30 FPS de um vídeo FullHD em uma CPU de notebook, o STMKF se demonstrou viável em tempo real. O uso de processadores multi-nucleados e GPUs, através de APIs de programação paralela, como o OpenMP e o CUDA, aumenta enormemente o desempenho do filtro.

Os resultados do STMKF se degradam com o aumento da quantidade de movimento no vídeo, afastando o STMKF dos resultados do conjunto de filtros que formam o estado da arte em remoção de ruído em vídeos. No entanto, os resultados estão próximos dos demais filtros de tempo real. O STMKF é adequado para vídeos em que o background é estático e os seus resultados são mais expressivos sempre que houver pouco movimento.

A indisponibilidade dos algoritmos dos outros autores dificulta a comparação entre algoritmos deste gênero. Vale notar, também, que não há na literatura uma padronização nos vídeos a serem utilizados em experimentos deste tipo. Os vídeos referenciados pela literatura estão espalhados por diversos repositórios e com qualidade inapropriada. Muitas vezes (quase sempre), a cada repositório em que um determinado vídeo é encontrado, um ruído diferente (em intensidade ou tipo) o contamina. Ou seja, mesmo se utilizando um vídeo que supostamente é o mesmo, os resultados obtidos através das métricas serão distintos.

De uma maneira geral, pode-se afirmar que o algoritmo proposto

atende de forma satisfatória uma ampla quantidade de vídeos, especialmente em certos tipos de transmissões ao vivo (fundo estático, como telejornais, por exemplo), em vídeo-conferências, robótica e monitoramento de ambientes.

6.1 TRABALHOS FUTUROS

Algumas possíveis melhorias podem ser elencadas para o STMKF:

Melhoria na estimativa de Δ : neste trabalho foi utilizado a diferença entre dois frames consecutivos com *blur* para calcular Δ . É muito provável que existam valores melhores para Δ . Apesar de potencialmente degradar a velocidade do filtro, uma possibilidade é o uso de algoritmos de detecção de movimento para calcular Δ .

Matrizes como variáveis do filtro de Kalman: o filtro de Kalman é deveras complexo e neste trabalho fora utilizado de forma simplificada. As simplificações feitas no filtro de Kalman podem reduzir a qualidade do resultado que este pode apresentar. A versão do filtro de Kalman utilizado neste trabalho assume que suas variáveis e parâmetros sejam valores numéricos. No entanto, a utilização de sua forma matricial pode melhorar os resultados. A complexidade do algoritmo também deve aumentar.

Escolha automatizada de Q : a escolha adequada de Q , de forma automatizada, ainda é um problema em aberto no STMKF. O valor de Q ainda precisa ser escolhido pelo usuário e depende principalmente da quantidade de movimento no vídeo. É possível que exista alguma função $f(K_k)$ capaz de propiciar valores aceitáveis para Q .

Avaliação do desempenho em *many-cores*: o STMKF foi testado em processadores multi-cores e em GPUs. É de interesse a avaliação do desempenho em *many-cores* de baixo consumo de energia como o Mellanox TILE-Gx72 e o Kalray MPPA-256.

Divisão de tarefas para processadores multi-core: a versão multi-core MT-STMKF divide cada frame do vídeo de forma que um bloco é destinado a cada *thread*. Abordagens distintas podem resultar em um algoritmo mais rápido, já que pode otimizar o acesso de cada *thread* a memória.

REFERÊNCIAS

- BARDU, T. Variational image denoising approach with diffusion porous media flow. **Abstract and Applied Analysis**, v. 2013, p. 8, 2013. Disponível em: <<http://dx.doi.org/10.1155/2013/856876>>. Acesso em: 13 out. 2014.
- BOULANGER, J.; KERVRANN, C.; BOUTHEMY, P. Space-time adaptation for patch-based image sequence restoration. **Pattern Analysis and Machine Intelligence, IEEE Transactions on**, IEEE, v. 29, n. 6, p. 1096–1102, 2007.
- BOVIK, A. **The Essential Guide to Video Processing**. [S.l.]: Elsevier Science, 2009. ISBN 9780080922508.
- BOVIK, A. **Handbook of Image and Video Processing**. [S.l.]: Elsevier Science, 2010. (Communications, Networking and Multimedia). ISBN 9780080533612.
- BUADES, A.; COLL, B.; MOREL, J.-M. A non-local algorithm for image denoising. **Proceedings of IEEE Computer Society Conf. on Computer Vision and Pattern Recognition (CVPR'05)**, v. 2, p. 60–65, 2005.
- CHAN, T. et al. A novel content-adaptive video denoising filter. In: **IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)**. Philadelphia, USA: [s.n.], 2005. p. 649–652.
- CHAUDHURY, K. N. Acceleration of the shiftable algorithm for bilateral filtering and nonlocal means. **Image Processing, IEEE Transactions on**, IEEE, v. 22, n. 4, p. 1291–1300, 2013.
- CHEN, T.-Y. et al. The study on video enhancement in the low-light environment by spatio-temporal filtering. In: **International Conference on Intelligent Systems Design and Applications (ISDA)**. Kaohsiung, Taiwan: IEEE, 2008. v. 3, p. 561–564.
- COCCHIA, F.; CARRATO, S.; RAMPONI, G. Design and real-time implementation of a 3-d rational filter for edge preserving smoothing. **IEEE Transactions on Consumer Electronics**, IEEE, v. 43, n. 4, p. 1291–1300, 1997.

CONCI, A.; AZEVEDO, E.; LETA, F. R. **Computação Gráfica: Teoria e Prática**. 1. ed. Rio de Janeiro, Brasil: Elsevier Editora, 2008. ISBN 9788535223293.

DABOV, K.; FOI, A.; EGIAZARIAN, K. Video denoising by sparse 3d transform-domain collaborative filtering. In: **Proc. 15th European Signal Proc. Conf.** [S.l.: s.n.], 2007. v. 1, n. 2, p. 7.

DABOV, K. et al. Image denoising with block-matching and 3d filtering. **Proc. of SPIE-IS&T Electronic Imaging**, v. 6064, 2006.

DAVIS, L.; ROSENFELD, A. Noise cleaning by iterated cleaning. **IEEE Trans. Syst., Man Cybern., SMC-8**, n. 9, p. 705–710, Sept 1978.

DUFAUX, F. et al. **High Dynamic Range Video: From Acquisition, to Display and Applications**. [S.l.]: Elsevier Science, 2016. ISBN 9780128030394.

FAROOQUE, M. A.; SOHANKAR, J. S. Survey on various noises and techniques for denoising the color image. **Int. Journal of Application or Innovation in Engineering & Management (IJAIEEM)**, v. 2, 2013.

GARG, R.; KUMAR, A. Comparison of various noise removals using bayesian framework. **Int. Journal of Modern Engineering Research (IJMER)**, v. 2, 2012.

HAN, Y.; CHEN, R. Efficient video denoising based on dynamic nonlocal means. **Image and Vision Computing**, v. 30, p. 78–85, 2012.

HONG-ZHI, W.; LING, C.; SHU-LIANG, X. Improved video denoising algorithm based on spatial-temporal combination. In: **Image and Graphics (ICIG), 2013 Seventh Int. Conf. on.** [S.l.: s.n.], 2013. p. 64–67.

JUNG, B.; SUKHATME, G. S. Detecting moving objects using a single camera on a mobile robot in an outdoor environment. In: **International Conference on Intelligent Autonomous Systems**. [S.l.: s.n.], 2004. p. 980–987.

KALMAN, R. E. A new approach to linear filtering and prediction problems. **Transactions of the ASME—Journal of Basic Engineering**, v. 82, n. Series D, p. 35–45, 1960.

KIRK, D. B.; HWU, W.-m. W. **Programming Massively Parallel Processors: A Hands-on Approach**. 1st. ed. San Francisco, USA: Morgan Kaufmann Publishers Inc., 2010. ISBN 0123814723.

KITCHENHAM, B. Procedures for performing systematic reviews. **Keele, UK, Keele University**, v. 33, n. 2004, p. 1–26, 2004.

KODAK. **História da Fotografia**: A câmera escura, o princípio da fotografia. [S.l.], 2014. Disponível em: <http://wwwbr.kodak.com/BR/pt/consumer/fotografia_digital_classica/para_uma_boa_foto/historia_fotografia/historia_da_fotografia02.shtml?primeiro=1>. Acesso em: 13 out. 2014.

KODAK. **História da Fotografia**: A química, em auxílio à fotografia. [S.l.], 2014. Disponível em: <http://wwwbr.kodak.com/BR/pt/consumer/fotografia_digital_classica/para_uma_boa_foto/historia_fotografia/historia_da_fotografia03.shtml?primeiro=1>. Acesso em: 13 out. 2014.

KUO, S.; LEE, B.; TIAN, W. **Real-Time Digital Signal Processing: Implementations and Applications**. Wiley, 2006. ISBN 9780470035511. Disponível em: <https://books.google.com.br/books?id=QIj9Pthp_T8C>.

LI, W.; ZHANG, J.; DAI, Q.-h. Video denoising using shape-adaptive sparse representation over similar spatio-temporal patches. **Signal Proc.: Image Communication**, v. 26, n. 4-5, p. 250 – 265, 2011.

LI, X.; ZHENG, Y. Patch-based video proc.: a variational bayesian approach. **Circuits and Systems for Video Technology, IEEE Transactions on, IEEE**, v. 19, n. 1, p. 27–40, 2009.

LIU, R.; FU, S.; ZHANG, C. Adaptive mixed image denoising based on image decomposition. **Optical Engineering**, v. 50, n. 2, 2011. Disponível em: <<http://dx.doi.org/10.1117/1.3542041>>.

MAHMOUD, R.; FAHEEM, M.; SARHAN, A. Intelligent denoising technique for spatial video denoising for real-time applications. In: **Computer Engineering Systems, 2008. ICCES 2008. Int. Conf. on**. [S.l.: s.n.], 2008. p. 407–412.

MathWorks. **Kalman Filter**: Design and use kalman filters in matlab and simulink. [S.l.], 2016. Disponível em: <<http://www.mathworks.com/discovery/kalman-filter.html>>. Acesso em: 06 jul. 2016.

MITCHELL, H.; MASHKIT, N. Noise smoothing by a fast k-nearest neighbour algorithm. **Signal Processing: Image Communication**, Elsevier, v. 4, n. 3, p. 227–232, 1992.

MYTHILE, C.; KAVITHA, V. Efficient technique for color image noise reduction. **The Research Bulletin of Jordan ACM**, II, 2011.

OpenCV. **Image Filtering**: bilateralfilter. [S.l.], 2016. Disponível em: <<http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=bilateralfilter#bilateralfilter>>. Acesso em: 30 mai. 2016.

OpenMP Architecture Review Board. **OpenMP Application Program Interface Version 4.0**. 2013. Disponível em: <<http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf>>.

PAUWELS, K. et al. A comparison of fpga and gpu for real-time phase-based optical flow, stereo, and local image features. **IEEE Transactions on Computers**, v. 61, n. 7, p. 999–1012, July 2012. ISSN 0018-9340.

PEDRINI, H.; SCHWARTZ, W. R. **Análise de Imagens Digitais: Princípios, Algoritmos e Aplicações**. 1st. ed. São Paulo, Brasil: Thomson Learning, 2008. ISBN 987-85-221-0595-3.

PERONA, P.; MALIK, J. Scale-space and edge detection using anisotropic diffusion. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 12, p. 629–639, 1990.

PIZURICA, A.; ZLOKOLICA, V.; PHILIPS, W. Combined wavelet domain and temporal video denoising. In: IEEE. **Advanced Video and Signal Based Surveillance, 2003. Proceedings. IEEE Conf. on**. [S.l.], 2003. p. 334–341.

PIZURICA, A.; ZLOKOLICA, V.; PHILIPS, W. Noise reduction in video sequences using wavelet-domain and temporal filtering. In: INT. SOC. FOR OPTICS AND PHOTONICS. **Photonics Technologies for Robotics, Automation, and Manufacturing**. [S.l.], 2004. p. 48–59.

PROTTER, M.; ELAD, M. Image sequence denoising via sparse and redundant representations. **Image Proc., IEEE Transactions on**, IEEE, v. 18, n. 1, p. 27–35, 2009.

PULLI, K. et al. Real-time computer vision with opencv. **Commun. ACM**, ACM, New York, NY, USA, v. 55, n. 6, p. 61–69, jun. 2012. ISSN 0001-0782.

RAHMAN, S. M.; AHMAD, M. O.; SWAMY, M. Video denoising based on inter-frame statistical modeling of wavelet coefficients.

Circuits and Systems for Video Technology, IEEE Transactions on, IEEE, v. 17, n. 2, p. 187–198, 2007.

RICHARDS, D. **The 30 Most Important Digital Cameras of All Time**. [S.l.], 2013. Disponível em:

<<http://www.popphoto.com/gear/2013/10/30-most-important-digital-cameras>>. Acesso em: 13 out.

2014.

RUSANOVSKYY, D.; EGIAZARIAN, K. Video denoising algorithm in sliding 3d dct domain. In: SPRINGER. **Advanced Concepts for Intelligent Vision Systems**. [S.l.], 2005. p. 618–625.

SEILLER, N.; SINGHAL, N.; PARK, I. K. Object oriented framework for real-time image processing on gpu. In: **IEEE International Conference on Image Processing (ICIP)**. Hong Kong, China: IEEE, 2010. p. 4477–4480.

SELESNICK, I. W.; LI, K. Y. Video denoising using 2d and 3d dual-tree complex wavelet transforms. In: INT. SOCIETY FOR OPTICS AND PHOTONICS. **Optical Science and Technology, SPIE's 48th Annual Meeting**. [S.l.], 2003. p. 607–618.

TOMASI, C.; MANDUCHI, R. Bilateral filtering for gray and color images. In: IEEE. **Computer Vision, 1998. Sixth Int. Conf. on**. [S.l.], 1998. p. 839–846.

TSIOTSIOS, C.; PETROU, M. On the choice of the parameters for anisotropic diffusion in image proc. **Pattern Recog.**, Elsevier Science Inc., New York, NY, USA, v. 46, n. 5, p. 1369–1381, maio 2013. ISSN 0031-3203. Disponível em:

<<http://dx.doi.org/10.1016/j.patcog.2012.11.012>>.

VARGHESE, G.; WANG, Z. Video denoising based on a spatiotemporal gaussian scale mixture model. **Circuits and Systems for Video Technology, IEEE Transactions on**, IEEE, v. 20, n. 7, p. 1032–1040, 2010.

WANG, Z.; BOVIK, A. C. Mean squared error: love it or leave it? a new look at signal fidelity measures. **IEEE signal processing magazine**, IEEE, v. 26, n. 1, p. 98–117, 2009.

WANG, Z.; BOVIK, A. C.; SIMONCELLI, E. P. Image quality assessment: From error visibility to structural similarity. **IEEE Transactions On Image Proc.**, v. 13, n. 4, 2004.

WOLF, W.; OZER, B.; LV, T. Smart cameras as embedded systems. **Computer**, v. 35, n. 9, p. 48–53, Sep 2002.

ZAMPOLO, R. d. F. **Restauração de Imagens via Filtragem de Kalman e Considerações sobre a Avaliação da Qualidade de Imagens Restauradas**. Tese (Tese) — Universidade Federal de Santa Catarina, Florianópolis: Universidade Federal de Santa Catarina, 2003.

ZLOKOLICA, V.; PHILIPS, W.; VILLE, D. V. D. A new non-linear filter for video processing. In: **IEEE Benelux Signal Processing Symposium**. [S.l.: s.n.], 2002. p. 221–224.

ZLOKOLICA, V.; PIZURICA, A.; PHILIPS, W. Wavelet-domain video denoising based on reliability measures. **Circuits and Systems for Video Technology, IEEE Transactions on, IEEE**, v. 16, n. 8, p. 993–1007, 2006.

ZUO, C. et al. Video denoising based on a spatiotemporal kalman-bilateral mixture model. **The Scientific World Journal**, v. 2013, 2013. Disponível em: <<http://dx.doi.org/10.1155/2013/438147>>.

APÊNDICE A – Resultados Complementares

Tabela 7 – Resultados em termos de PSNR comparados com outros trabalhos. Dados dos algoritmos STA, VBM3D, K-SVD, 3D-Patch, VBM3D, ST-GSM e DNLM foram extraídos de Han e Chen (2012)

Sequência	Input PSNR [dB]							
		STA	VBM3D	K-SVD	3D-Patch	ST-GSM	DNLM	STMKF
Salesman	28	35.13	38.79	37.91	39.26	37.93	35.22	35.24
	24	32.60	36.07	35.59	36.35	35.17	32.73	32.74
Garden	28	31.33	32.51	32.13	-	-	31.28	25.44
MissA	28	39.39	41.64	40.49	40.23	41.43	39.70	37.63
Suzie	28	37.07	38.16	37.96	38.40	38.36	37.22	34.11
	24	35.11	36.24	35.95	36.32	36.21	35.25	31.53
Trevor	28	36.68	38.17	38.10	38.31	37.17	36.46	34.33
	24	34.79	35.82	35.97	35.81	34.68	34.51	31.42
Foreman	28	34.94	37.27	37.86	36.88	36.85	36.19	32.41
	24	32.90	35.19	35.86	34.55	34.37	34.06	29.87

Tabela 8 – Resultados em termos de SSIM comparados com outros trabalhos. Dados dos algoritmos WRSTF, SEQWT, 3DWTF, IFSM, 3DSWDCT, VBM3D, ST-GSM and DNLM foram extraídos de Han e Chen (2012)

Sequência	σ	WRSTF	SEQWT	3DWTF	IFSM	3DSWDCT	VBM3D	ST-GSM	DNLM	STMKF
Foreman	10	0.914	-	-	0.886	0.932	0.935	0.937	0.946	0.885
	15	0.877	-	-	0.836	0.907	0.917	0.917	0.929	0.829
	20	0.841	-	-	0.793	0.884	0.903	0.901	0.913	0.794
Salesman	10	0.932	0.900	0.923	0.904	0.955	0.960	0.960	0.931	0.955
	15	0.901	0.846	0.903	0.851	0.930	0.945	0.941	0.889	0.935
	20	0.868	0.796	0.882	0.801	0.905	0.925	0.923	0.849	0.916
MissA	10	0.905	-	-	0.904	0.946	0.947	0.952	0.964	0.937
	15	0.877	-	-	0.857	0.928	0.939	0.943	0.951	0.899
	20	0.846	-	-	0.812	0.909	0.933	0.936	0.939	0.854
Tennis	10	0.897	0.842	0.856	0.855	0.894	0.901	0.894	0.856	0.849
	15	0.839	0.722	0.793	0.776	0.834	0.847	0.841	0.795	0.793
	20	0.790	0.716	0.740	0.709	0.709	0.800	0.797	0.758	0.735
Garden	10	0.953	0.941	0.909	0.927	0.959	0.962	0.950	0.947	0.901
	15	0.922	0.893	0.872	0.882	0.931	0.940	0.925	0.906	0.857
	20	0.889	0.842	0.840	0.837	0.900	0.916	0.900	0.865	0.804
Football	10	-	-	-	0.884	0.911	0.923	0.913	0.891	0.885
	15	-	-	-	0.813	0.851	0.874	0.865	0.838	0.830
	20	-	-	-	0.749	0.801	0.822	0.820	0.795	0.773