UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
Departamento de Engenharia Elétrica

André de Souza Tarallo

**Construção Automática de Mosaicos de Imagens Digitais Aéreas
Agrícolas Utilizando Transformada SIFT e Processamento Paralelo**

Tese apresentada à Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Doutor em Ciências, Programa de Engenharia Elétrica.

Área de Concentração: Processamento de Sinais e Instrumentação

Orientadora: Profa. Dra. Maria Stela Veludo de Paiva

SÃO CARLOS

2013

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTA TRABALHO,
POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS
DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

T176c Tarallo, André de Souza
Construção automática de mosaicos de imagens
digitais aéreas agrícolas utilizando transformada SIFT e
processamento paralelo / André de Souza Tarallo;
orientadora Maria Stela Veludo de Paiva. São Carlos,
2013.

Tese (Doutorado) - Programa de Pós-Graduação em
Engenharia Elétrica e Área de Concentração em
Processamento de Sinais e Instrumentação -- Escola de
Engenharia de São Carlos da Universidade de São Paulo,
2013.

1. Imagens aéreas agrícolas. 2. Mosaico automático.
3. OpenMP. 4. Processamento paralelo. 5. Transformada
SIFT. 6. Speedup. I. Título.

FOLHA DE JULGAMENTO

Candidato: Bacharel **ANDRÉ DE SOUZA TARALLO**.

Título da tese: "Construção automática de mosaicos de imagens digitais aéreas agrícolas utilizando transformada SIFT e processamento paralelo".

Data da defesa: 26/08/2013

Comissão Julgadora:

Resultado:

Profa. Dra. **Maria Stela Veludo de Paiva (Orientadora)**
(Escola de Engenharia de São Carlos/EESC)

APROVADO

Prof. Dr. **Hermes Senger**
(Universidade Federal de São Carlos/UFSCar)

Aprovado

Dr. **Lúcio André de Castro Jorge**
(EMBRAPA)

Aprovado

Prof. Dr. **Almir Olivette Artero**
(Universidade Estadual Paulista "Júlio de Mesquita Filho"/UNESP- Presidente Prudente)

APROVADO

Prof. Dr. **Celso Aparecido de França**
(Universidade Federal de São Carlos/UFSCar)

APROVADO

Coordenador do Programa de Pós-Graduação em Engenharia Elétrica e Presidente da Comissão de Pós-Graduação:

Prof. Titular **Denis Vinicius Coury**

DEDICATÓRIA

Dedico este trabalho a Deus
por me dar a vida e, com isso,
permitir-me cultivar o
conhecimento e a sabedoria.

AGRADECIMENTOS

Agradeço a Deus por ter me dado saúde e disposição para fazer este trabalho e à minha família que sempre me incentivou, principalmente a meus pais.

À minha orientadora, Profa. Dra. Maria Stela Veludo de Paiva, por toda força, esforço e persistência na condução desta tese.

À Universidade de São Paulo por colocar à minha disposição sua estrutura. Aos meus amigos do Laboratório de Visão Computacional (LAVI), em especial ao Dr. Francisco Assis da Silva pelas explicações e incentivos.

À Embrapa - Instrumentação por fornecer toda sua estrutura e em especial ao Engenheiro Dr. Lúcio André de Castro Jorge por apoiar, incentivar e me atender em todas as vezes que necessitei.

Ao Alan Kazuo Hiraga, German Andres Gaviria Martinez, Diego Carlos Pereira da Silva, Roberta Vendramini Gonçalves e Wellington da Rocha Gouveia por toda contribuição, esforço e ajuda no projeto.

Ao professor Dr. Hermes Senger da UFSCar, por contribuir e auxiliar na parte que lhe coube com o projeto.

À Tamiris Trevisan Negri por todo o apoio e suas contribuições na revisão da tese.

A meus queridos (as) amigos (as) que me trazem mensagens de encorajamento todos os dias.

Sou grato também aos funcionários e professores da graduação e pós-graduação do Departamento de Engenharia Elétrica.

Resumo

TARALLO, A.S. **Construção Automática de Mosaicos de Imagens Digitais Aéreas Agrícolas Utilizando a Transformada SIFT e Processamento Paralelo**. 2013. 265f. Tese (Doutorado) – Escola de Engenharia de São Carlos (EESC), Universidade de São Paulo.

A construção automática de grandes mosaicos a partir de imagens digitais de alta resolução é uma área de grande importância, encontrando aplicações em diferentes áreas, como agricultura, meteorologia, sensoriamento remoto e biológicas. Na agricultura, a eficiência no processo de tomada de decisão para controle de pragas, doenças ou queimadas está relacionada com a obtenção rápida de informações. Até o presente momento este controle vem sendo feito de maneira semiautomática, necessitando obter o modelo digital do terreno, fazer a ortorretificação de imagens, inserir marcações manualmente no solo, para que um software possa construir um mosaico de maneira convencional. Para automatizar este processo, o presente projeto propõe três metodologias (1,2,3) baseadas em algoritmos já consolidados na literatura (SIFT, BBF e RANSAC) e processamento paralelo (*OpenMP*), utilizando imagens aéreas agrícolas de alta resolução, de pastagens e plantações de diversas culturas. As metodologias diferem na maneira como os cálculos são realizados para a construção dos mosaicos. Construir mosaicos com este padrão de imagem não é uma tarefa trivial, pois requer alto esforço computacional para processamento destas imagens. As metodologias incluem um pré-processamento das imagens para minimizar possíveis distorções que surgem no processo de aquisição de imagens e contém também algoritmos para suavização das emendas das imagens no mosaico. A base de imagens, denominada base de imagens sem redimensionamento, foi construída a partir de imagens com dimensão de 2336 x 3504 pixels (100 imagens divididas em 10 grupos de 10 imagens), obtidas na região de Santa Rita do Sapucaí - MG, as quais foram utilizadas para validar a metodologia. Outra base de imagens, referida como base de imagens redimensionada, contém 200 imagens de 533 x 800 pixels (10 grupos de 20 imagens) e foi utilizada para avaliação de distorção para comparação com os softwares livres *Autostitch* e *PTGui*, os quais possuem parâmetros pré-definidos para a resolução de 533 x 800 pixels. Os resultados do tempo de processamento sequencial para as três metodologias evidenciaram a metodologia 3 com o menor tempo, sendo esta 63,5% mais rápida que a metodologia 1 e 44,5% do que a metodologia 2. O processamento paralelo foi avaliado para um computador com 2, 4 e 8 *threads* (4 núcleos físicos e 4 núcleos virtuais), reduzindo em 60% o tempo para a construção dos mosaicos de imagens para a metodologia 1. Verificou-se que um computador com 4 *threads* (núcleos físicos) é o mais adequado em tempo de execução e *Speedup*, uma vez que quando se utilizam 8 *threads* são incluídos as *threads* virtuais. Os resultados dos testes de distorção obtidos evidenciam que os mosaicos gerados com a metodologia 1 apresentam menores distorções para 7 grupos de imagens em um total de 10. Foram também avaliadas as distorções nas junções de cinco mosaicos constituídos apenas por pares de imagens utilizando a metodologia 3, evidenciando que a metodologia 3 apresenta menor distorção para 4 mosaicos, em um total de 5. O presente trabalho contribui com a metodologia 2 e 3, com a minimização das distorções das junções do mosaico, com o paralelismo em *OpenMP* e com a avaliação de paralelismo com MPI.

Palavras Chave: Imagens aéreas agrícolas, Mosaico automático, *OpenMP*, Processamento paralelo, Transformada SIFT, *Speedup*

Abstract

TARALLO, A.S. **Automatic Construction of Mosaics from Aerial Digital Images Agricultural using SIFT Transform and Parallel Processing**. 2013. 265p. PhD. Thesis – School of Engineering of São Carlos (EESC), University of São Paulo.

The automatic construction of large mosaics from high resolution digital images is an area of great importance, which finds applications in different areas, especially agriculture, meteorology, biology and remote sensing. In agriculture, the efficiency of decision making is linked to obtaining faster and more accurate information, especially in the control of pests, diseases or fire control. So far this has been done semiautomatically and it is necessary to obtain the digital terrain model, do the orthorectification of images, insert markings on the ground by manual labor, so that software can build a mosaic in the conventional way. To automate this process, this project proposes three methodologies (1,2,3) based on algorithms already well-established in the literature (SIFT, BBF e RANSAC) and parallel processing (OpenMP), using high resolution/size aerial images agricultural of pasture and diverse cultures. The methodologies differ in how the calculations are performed for the construction of mosaics. Build mosaics with this kind of picture isn't a trivial task, as it requires high computational effort for processing these images. The methodologies include a pre-processing of images to minimize possible distortions that arise in the process of image acquisition and also contain algorithms for smoothing the seams of the images in the mosaic. The image database, called image database without scaling, was constructed from images with dimensions of 2336 x 3504 pixels (100 images divided into 10 groups of 10 pictures), obtained in the region of Santa Rita do Sapucaí - MG, which were used to validate the methodology. Another image database, referred to as base images resize, contains 200 images of 533 x 800 pixels (10 groups of 20 pictures). It was used for evaluation of distortion compared to the free softwares Autostitch and PTGui, which have pre-defined parameters for the resolution of 533 x 800 pixels. The results of sequential processing time for the three methodologies showed the methodology 3 with the shortest time, which is 63.5% faster than the methodology 1 and 44.5% faster than the methodology 2. Parallel processing was evaluated for a computer with 2, 4 and 8 threads (4 physical cores and 4 virtual cores), reducing by 60% the time to build the mosaics of images for the methodology 1. It was found that a computer with 4 threads (physical cores) is most appropriate in execution time and Speedup, since when using 8 threads, threads virtual are included. The test results of distortion show that the mosaics generated with the methodology 1 have lower distortion for 7 groups of images in a total of 10. Distortions at the junctions of five mosaics consisting only of pairs of images were also evaluate utilizing the methodology 3, indicating that the methodology 3 has less distortion for 4 mosaics, for a total of 5. Contributions of this work have been the methodologies 2 and 3, with the distortions minimization of the mosaic junction, the parallelism in OpenMP and the assessment of parallelism with MPI.

Key words: aerial images agricultural, automatic mosaic, OpenMP, parallel processing, SIFT Transform, Speedup

Lista de Figuras

Figura 1 - Esquema computacional da visão estéreo.....	32
Figura 2 – Plano epipolar e seus principais elementos.	33
Figura 3 – Representação de linhas epipolares e planos epipolares.	34
Figura 4 – Projeção na retificação cilíndrica.....	36
Figura 5 – Mudança nas coordenadas da imagem.	37
Figura 6 – Vista superior do cilindro e do plano da imagem.....	37
Figura 7 – Representação da obtenção da <i>DoG</i>	42
Figura 8 – Máximos e mínimos das imagens da <i>DoG</i>	43
Figura 9 – Determinação da quantidade de amostra por oitava.....	43
Figura 10 – Determinação do fator sigma.	44
Figura 11 – Exemplo do processo de criação de oitavas.	44
Figura 12 – Exemplo da Diferença da Gaussiana (<i>DoG</i>).	45
Figura 13 – Exemplo da seleção de pontos chave.....	47
Figura 14 – Criação do descritor de pontos chave.	51
Figura 15 – Exemplo de <i>K-d tree</i>	56
Figura 16 – Exemplo de busca BBF.....	58
Figura 17 – Comparação entre <i>kd-tree</i> e BBF.....	59
Figura 18 – Exemplo de ajuste de pontos.	61
Figura 19 – Classificação das arquiteturas de computadores.	66
Figura 20 – Classificação das arquiteturas MIMD.	66
Figura 21 – Compartilhamento de memória de um SMP.....	69
Figura 22 – Modelo de programação <i>OpenMP</i>	71
Figura 23 – Elementos da interface <i>OpenMP</i>	71
Figura 24 – Trechos serial e paralelo de um código.....	76
Figura 25 – Correspondências das características após o RANSAC.....	80
Figura 26 – Composição de duas imagens em um mosaico.	80
Figura 27 – Exemplo de par de imagens sequenciais.....	81
Figura 28 – Sequência de <i>frames</i> do artigo de Liu, Chen e Guo (2008).	82
Figura 29 – Mosaico gerado no artigo de Liu, Chen e Guo (2008).....	83
Figura 30 – Exemplo de imagem usado no projeto de Cho, Bae e Durand (2009). ..	84
Figura 31 – Características SIFT no projeto de Cho, Bae e Durand (2009).	85
Figura 32 – Mosaico gerado no projeto de Cho, Bae e Durand (2009).	85
Figura 33 – Casamento do mosaico com o vídeo, em Cho, Bae e Durand (2009). ..	86
Figura 34 – Exemplo de imagem usado por Bei e Haizhen (2009).	87
Figura 35 – Exemplo de mosaico gerado no trabalho de Bei e Haizhen (2009).....	88
Figura 36 – Mosaico de imagens no trabalho de Li e Geng (2010).....	88
Figura 37 – Mosaico no trabalho de Li e Geng (2010)..	89
Figura 38 – Regiões de sobreposição entre imagens de fundo de olho.....	90
Figura 39 – Etapas do processo de construção de mosaico automático.	91
Figura 40 – Mosaico gerado por Xing e Huang (2010).....	92
Figura 41 – Mosaico no trabalho de Suzuki, Amano e Hashizume (2010)	93

Figura 42 – Comparação do Borramento: (a) SIFT, (b) SURF, (c) PCA-SIFT.....	97
Figura 43 – Resultados de desempenho da SIFT por SMP.	105
Figura 44 – Parâmetros de configuração no sistema CMP.	105
Figura 45 – Implementação utilizando <i>OpenMP</i>	108
Figura 46 – Ganho de desempenho utilizando <i>OpenMP</i>	108
Figura 47 – Divisão dos Dados	110
Figura 48 – Esquema tradicional de mosaico paralelo.....	111
Figura 49 – Metodologia paralela proposta.	113
Figura 50 – A mínima árvore de abrangência.	114
Figura 51 – Diagrama de reamostragem e mistura.	115
Figura 52 – Resultados de <i>Speedup</i> gerados.	115
Figura 53 – Transformada SIFT original.....	116
Figura 54 – Transformada SIFT modificada.....	117
Figura 55 – Processamento das oitavas e seus subníveis.....	118
Figura 56 – Exemplo de redistribuição de pontos chave.....	118
Figura 57 – Esquema geral do cálculo da orientação.	119
Figura 58 – Exemplo de cálculo da orientação e reindexação.	119
Figura 59 – Exemplo de geração do descritor.....	120
Figura 60 – Diagrama do pSIFT	123
Figura 61 – Métodos de divisão dos dados.....	123
Figura 62 – <i>Speedup</i> entre pSIFT e DDP-SIFT.....	125
Figura 63 – Desempenho entre pSIFT e nsSIFT.....	125
Figura 64 – Exemplo de imagem usada no projeto.	130
Figura 65 – Padrão de rota para obtenção das imagens.	130
Figura 66 – Características extraídas pela Transformada SIFT.	133
Figura 67 – Exemplo de região de interesse na “Imagem Mosaico”.	133
Figura 68 – Diagrama da metodologia empregada.	134
Figura 69 – Correspondência das características SIFT (Identificação).	135
Figura 70 – Correspondência após aplicação do RANSAC.	136
Figura 71 – Exemplo de imagem fora da região visível do monitor.....	136
Figura 72 – Geração da matriz homográfica.	137
Figura 73 – Exemplo de janela de imagens.	138
Figura 74 – Exemplo de imagens nas janelas.....	139
Figura 75 – Exemplo da matriz identidade	139
Figura 76 – Exemplo de região de sobreposição.	140
Figura 77 – Exemplo de “Efeito Ghost”.	141
Figura 78 – Exemplo de redução da região de sobreposição.	141
Figura 79 – Imagem panorâmica sem ajuste (Brown; Lowe, 2007).	142
Figura 80 – Imagem panorâmica após ajuste (Brown; Lowe, 2007).	143
Figura 81 – Exemplo de mosaico contendo 20 imagens.....	143
Figura 82 – Diagrama de blocos da metodologia 3.....	145
Figura 83 – Exemplo de informações obtidas com o Gprof.....	147
Figura 84 – Diagrama gerado pelo Gprof para a metodologia 1.	150
Figura 85 – Diagrama gerado pelo Gprof para a metodologia 2 – parte a.	152

Figura 86 – Diagrama gerado pelo Gprof para a metodologia 3 – parte a.	154
Figura 87 – Tempo para criação e fechamento de <i>threads</i>	164
Figura 88 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 1 da metodologia 1.	167
Figura 89 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 2 da metodologia 1.	167
Figura 90 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 3 da metodologia 1.	168
Figura 91 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 4 da metodologia 1.	168
Figura 92 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 5 da metodologia 1.	169
Figura 93 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 6 da metodologia 1.	169
Figura 94 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 7 da metodologia 1.	170
Figura 95 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 8 da metodologia 1.	170
Figura 96 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 9 da metodologia 1.	171
Figura 97 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 10 da metodologia 1.	171
Figura 98 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 1 da metodologia 2.	173
Figura 99 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 2 da metodologia 2.	174
Figura 100 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 3 da metodologia 2.	174
Figura 101 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 4 da metodologia 2.	175
Figura 102 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 5 da metodologia 2.	175
Figura 103 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 6 da metodologia 2.	176
Figura 104 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 7 da metodologia 2.	176
Figura 105 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 8 da metodologia 2.	177
Figura 106 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 9 da metodologia 2.	177
Figura 107 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 10 da metodologia 2.	178
Figura 108 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 1 da metodologia 3.	180
Figura 109 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 2 da metodologia 3.	181
Figura 110 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 3 da metodologia 3.	181
Figura 111 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 4 da metodologia 3.	182
Figura 112 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 5 da metodologia 3.	182
Figura 113 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 6 da metodologia 3.	183
Figura 114 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 7 da metodologia 3.	183
Figura 115 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 8 da metodologia 3.	184
Figura 116 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 9 da metodologia 3.	184
Figura 117 – <i>Speedup</i> máximo e <i>Speedup</i> do grupo 10 da metodologia 3.	185
Figura 118 – Mosaicos gerados em comparação com o <i>Google Earth</i>	190
Figura 119 – Trigonometria de triângulos.	190

Lista de Tabelas

Tabela 1 – Resultados do trabalho de Goltz e Shiguemori (2008).	82
Tabela 2 – Filtragem dos pontos característicos por Bei e Haizhen (2009).	87
Tabela 3 – Desempenho dos resultados obtidos por WANG et. al (2010a).	91
Tabela 4 – Resumo das metodologias utilizando a Transformada SIFT.	95
Tabela 5 – Tempos de processamento em Juan e Gwun (2009).	96
Tabela 6 – Resumo dos resultados obtidos em Juan e Gwun (2009).	97
Tabela 7 – Resultados obtidos por Jiayuan; Yigang e Yun (2010),	100
Tabela 8 – Resumo de outras metodologias aplicadas à geração de mosaicos.	102
Tabela 9 – Comparação entre CPU e GPU em Liu, Chen e Li (2009).	106
Tabela 10 – Comparação do processamento entre CPU e GPU.	109
Tabela 11 – Comparativo dos testes em Liang et. al (2010).	117
Tabela 12 – Resultados obtidos em Vasilyev (2011).	120
Tabela 13 – Comparação dos resultados (tempos em ms) de Huang et. al. (2011).	122
Tabela 14 – Resumo das paralelizações da Transformada SIFT.....	126
Tabela 15 – Funções paralelizadas na metodologia 1.	149
Tabela 16 – Funções paralelizadas na metodologia 2.	151
Tabela 17 – Funções paralelizadas na metodologia 3.	151
Tabela 18 – Tempo de processamento sequencial das metodologias 1,2 e 3.	162
Tabela 19 – Quantidade de características extraídas.	163
Tabela 20 – Tempo de execução com a metodologia 1.....	165
Tabela 21 – <i>Speedup</i> da metodologia 1.....	166
Tabela 22 – <i>Speedup</i> máximo (teórico).....	166
Tabela 23 – Tempo de execução da metodologia 2.....	172
Tabela 24 – <i>Speedup</i> da metodologia 2.....	172
Tabela 25 – <i>Speedup</i> máximo (teórico).....	173
Tabela 26 – Tempo de execução da metodologia 3.....	179
Tabela 27 – <i>Speedup</i> da metodologia 3.....	180
Tabela 28 – <i>Speedup</i> máximo (teórico).....	180
Tabela 29 – Tempo das funções, de acordo com o aumento das iterações.	187
Tabela 30 – Tempo de execução em MPI da metodologia 2.	188
Tabela 31 – Cálculo das proporções para o grupo 1.....	192
Tabela 32 – Cálculo das proporções para o grupo 2.....	192
Tabela 33 – Cálculo das proporções para o grupo 3.....	192
Tabela 34 – Cálculo das proporções para o grupo 4.....	192
Tabela 35 – Cálculo das proporções para o grupo 5.....	192
Tabela 36 – Cálculo das proporções para o grupo 6.....	192
Tabela 37 – Cálculo das proporções para o grupo 7.....	193
Tabela 38 – Cálculo das proporções para o grupo 8.....	193
Tabela 39 – Cálculo das proporções para o grupo 9.....	193
Tabela 40 – Cálculo das proporções para o grupo 10.....	193

Tabela 41 – Cálculo das proporções para o par 1.....	194
Tabela 42 – Cálculo das proporções para o par 2.....	194
Tabela 43 – Cálculo das proporções para o par 3.....	195
Tabela 44 – Cálculo das proporções para o par 4.....	195
Tabela 45 – Cálculo das proporções para o par 5.....	195
Tabela F1 – Valores das marcações do grupo 1.....	248
Tabela F2 – Valores das marcações do grupo 2.....	248
Tabela F3 – Valores das marcações do grupo 3.....	249
Tabela F4 – Valores das marcações do grupo 4.....	249
Tabela F5 – Valores das marcações do grupo 5.....	250
Tabela F6 – Valores das marcações do grupo 6.....	250
Tabela F7 – Valores das marcações do grupo 7.....	251
Tabela F8 – Valores das marcações do grupo 8.....	251
Tabela F9 – Valores das marcações do grupo 9.....	252
Tabela F10 – Valores das marcações do grupo 10.....	252
Tabela G1– Ângulos e segmentos de reta do grupo 1.....	253
Tabela G2– Ângulos e segmentos de reta do grupo 2.....	253
Tabela G3– Ângulos e segmentos de reta do grupo 3.....	253
Tabela G4– Ângulos e segmentos de reta do grupo 4.....	253
Tabela G5– Ângulos e segmentos de reta do grupo 5.....	254
Tabela G6– Ângulos e segmentos de reta do grupo 6.....	254
Tabela G7– Ângulos e segmentos de reta do grupo 7.....	254
Tabela G8– Ângulos e segmentos de reta do grupo 8.....	254
Tabela G9– Ângulos e segmentos de reta do grupo 9.....	254
Tabela G10– Ângulos e segmentos de reta do grupo 10.....	255
Tabela I1 – Valores das marcações do par 1.....	261
Tabela I2 – Valores das marcações do par 2.....	261
Tabela I3 – Valores das marcações do par 3.....	262
Tabela I4 – Valores das marcações do par 4.....	262
Tabela I5 – Valores das marcações do par 5.....	263
Tabela J1 – Ângulos e segmentos de reta do par 1.....	264
Tabela J2 – Ângulos e segmentos de reta do par 2.....	264
Tabela J3 – Ângulos e segmentos de reta do par 3.....	264
Tabela J4 – Ângulos e segmentos de reta do par 4.....	264
Tabela J5 – Ângulos e segmentos de reta do par 5.....	265

Lista de Siglas

API	<i>Application Programmig Interface</i>
BBF	<i>Best Bin First</i>
CMP	<i>Chip Multiprocessor</i>
CUDA	<i>Compute Unified Device Architecture</i>
DDP-SIFT	<i>Distributed Dynamic Parallel - Scale Invariant Feature Transform</i>
DoG	<i>Difference of Gaussian</i>
Embrapa	<i>Empresa Brasileira de Pesquisas Agropecuárias</i>
GPGPU	<i>General Propose Computing on Graphics Processing Unit</i>
GPU	<i>Graphics Processing Unit</i>
GPS	<i>Global Positioning System</i>
LLC	<i>Last-Level-Cache</i>
MIT	<i>Massachussetts Institute of Tecnology</i>
MLESAC	<i>Maximum Likelihood Estimation Sample Consensus</i>
MPI	<i>Message Passing Interface</i>
NUDT	<i>National University of Defense Tecnology</i>
NUMA	<i>Non-Uniform Memory Access</i>
OpenMP	<i>Open Multi-Processing</i>
PCA-SIFT	<i>Principal Component Analysis - Scale Invariant Feature Transform</i>
RANSAC	<i>Random Sample Consensus</i>
SIFT	<i>Scale Invariant Feature Transform</i>
SIMD	<i>Single Instruction Multiple Data</i>
SIMT	<i>Single Instruction Multiple Thread</i>
SM	<i>Multiprocessors-Streaming</i>
SMP	<i>Symmetric Multiprocessing</i>
SP	<i>Streaming-Processor</i>
SURF	<i>Speeded Up Robust Features</i>
UAV	<i>Unmanned aerial vehicle</i>
UTM	<i>Universal Transverse Mercator</i>
UMA	<i>Uniform Memory Access</i>
VANT	<i>Veículo Aéreo Não Tripulado</i>

SUMÁRIO

Resumo	9
Abstract	11
Lista de Figuras	13
Lista de Tabelas	17
Lista de Siglas	19
1. Introdução	25
1.1 Considerações Iniciais	25
1.2 Objetivos	27
1.3 Justificativa	27
1.4 Contribuições	28
1.5 Estrutura do Trabalho	29
2. Técnicas para Construção de Mosaicos de Imagens	31
2.1. Considerações Iniciais	31
2.2. Geometria Epipolar	31
2.3. Retificações	35
2.3.1. Retificação Cilíndrica	36
2.4. A Transformada SIFT	38
2.4.1. Detecção de Extremos no Espaço Escala	40
2.4.2. Localização Exata dos Pontos Chave	45
2.4.2.1. Eliminando Respostas nas Bordas	48
2.4.3. Determinação da Orientação	49
2.4.4. O Descritor da Imagem Local	50
2.4.4.1. Representação do Descritor	50
2.5. Etapas Paralelizáveis da Transformada SIFT	53
2.6. Correspondência das Características	53
2.6.1. Algoritmos de Busca	54
2.6.1.1. O Algoritmo <i>K-d tree</i>	55
2.6.1.2. O Algoritmo BBF	57
2.7. Imagens Correspondentes	59
2.7.1. O RANSAC	60
2.8. Considerações Finais	64

3. Processamento Paralelo.....	65
3.1. Considerações Iniciais.....	65
3.2. Arquitetura de Computadores.....	65
3.3. Processamento Paralelo	67
3.3.1. Processamento Paralelo em SMPs.....	68
3.4. Padrão <i>OpenMP</i>	70
3.5. Padrão MPI.....	72
3.5.1. Operações em MPI	74
3.5.2. Determinismo	75
3.5.3. Operações Globais em MPI	75
3.6. Métricas de Desempenho.....	76
3.6.1. Lei de Amdahl	76
3.7. Considerações Finais	77
4. Trabalhos Relacionados	79
4.1. Considerações Iniciais.....	79
4.2. Transformada SIFT Aplicada à Geração de Mosaicos	80
4.3. Comparação da Transformada SIFT com Outras Técnicas Aplicadas à Geração de Mosaicos.....	95
4.4. Paralelismo Aplicado à Transformada SIFT e a Geração de Mosaicos	102
4.5. Considerações Finais	126
5. Metodologia Desenvolvida.....	129
5.1. Considerações Iniciais.....	129
5.2. A Base de Imagens	129
5.3. Metodologia Desenvolvida.....	131
5.3.1. Metodologia 1	132
5.3.2. Metodologia 2.....	143
5.3.3. Metodologia 3.....	144
5.4. Processamento Paralelo	146
5.4.1. Processamento Paralelo da Metodologia 1	148
5.4.2. Processamento Paralelo da Metodologia 2	151
5.4.3. Processamento Paralelo da Metodologia 3.....	151
5.4.4. Construção de Mosaicos em Paralelo Usando MPI	157
5.5. Considerações Sobre CUDA.....	158
5.6. Considerações Finais	159

6. Resultados e Discussões.....	161
6.1. Resultados do Processamento Sequencial	162
6.2. Resultados do Processamento Paralelo.....	164
6.2.1. Paralelismo na Metodologia 1	165
6.2.2. Paralelismo na Metodologia 2	171
6.2.3. Paralelismo na Metodologia 3	178
6.2.4. Considerações Sobre o Paralelismo das Metodologias 2 e 3	185
6.2.5. Resultados Utilizando MPI	188
6.3. Resultados das Distorções dos Mosaicos	189
6.4. Resultados das Distorções por Pares de Imagens.....	194
7. Conclusões	197
7.1. Trabalhos Futuros	200
7.2. Publicação de Artigos.....	201
7.2.1. Artigos Publicados.....	201
7.2.2. Artigos Aceitos para Publicação.....	201
7.2.3. Artigos a Serem Sumetidos.....	202
Referências Bibliográficas	203
Apêndice A – Base de Imagens Sem Redimensionamento	213
Apêndice B – Mosaicos da Base Sem Redimensionamento.....	218
Apêndice C – Base de Imagens Redimensionadas	223
Apêndice D – Mosaicos da Base de Imagens Redimensionada	233
Apêndice E – Comparação da Distorção Utilizando a Base Redimensionada	238
Apêndice F – Valores das Coordenadas A,B,C para Cálculo da Distorção.....	248
Apêndice G – Valores dos Ângulos e dos Segmentos de Reta.....	253
Apêndice H – Comparação da Distorção por Pares de Imagens	256
Apêndice I – Valores das Coordenadas A,B,C para Cálculo da Distorção.....	261
Apêndice J – Valores dos Ângulos e dos Segmentos de Reta.....	264

1. Introdução

1.1 Considerações Iniciais

Um mosaico de imagens consiste em unir grande quantidade de imagens em uma nova imagem (Shum e Szeliski, 1997). Dentre as várias aplicações com mosaicos de imagens, destacam-se os mosaicos que são construídos a partir de imagens microscópicas (Ma et. al, 2007; Yongwei et. al, 2009), imagens de satélite (Kerschner, 2001; Helmer; Ruefenacht, 2005; Soille, 2006; Bielski et al., 2007) e aqueles que usam imagens aéreas (Wang et al., 2008; Du et al., 2008; Pan et al., 2009; Laliberte et al., 2010; Chon et al., 2010; López et al., 2011).

Um fator importante a ser considerado na aquisição das imagens, para construção de mosaicos, é que estas tem que ter uma área de sobreposição. No caso de imagens aéreas, para minimizar as distorções que surgem na fase de obtenção das imagens, deve ser escolhida a resolução mais adequada ao terreno e à geometria de visualização/iluminação, por meio do ajuste da altura e direção do voo em relação ao solo (Barnsley et al., 1997).

A construção de mosaicos de imagens envolve a identificação de pontos característicos em cada uma das imagens e a correspondência desses pontos característicos entre as imagens (*image matching*). Uma das abordagens para se trabalhar com correspondência de imagens é usar descritores locais para se representar uma imagem. Descritores são vetores de características de uma imagem ou de determinadas regiões de uma imagem que podem ser usados para se comparar regiões entre diferentes imagens.

Na literatura podem ser encontrados diferentes algoritmos que utilizam descritores e fazem a correspondência de imagens, podendo-se citar os seguintes: Transformada SIFT (*Scale Invariant Feature Transform*), proposto por Lowe (2004); o SURF (*Speeded Up Robust Features*), proposto por Bay et. al. (2006a) e suas respectivas variações, como o PCA-SIFT (*Principal Component Analysis Scale Invariant Feature Transform*) proposto por Ke e Sukthankar (2004), U-SURF (Bay et. al., 2008b) e o SURF-128 (Bay et. al., 2008b). Mas grande parte dos trabalhos relacionados à construção de mosaicos utiliza a Transformada SIFT ou SURF por gerarem melhores resultados.

Os trabalhos de Cho, Bae e Durand (2009), Li e Geng (2010), Kaynig et. al. (2010), Liu et. al. (2011), Zhong et. al. (2012), Song e Lu (2013) e Gui et. al. (2013) mostram exemplos de construção de mosaicos baseados na Transformada SIFT ou SURF.

Os artigos de Mikolajczyk e Schmid (2005), Bay et. al. (2008b), Juan e Gwun (2009), Yang et. al. (2011), Khan et. al. (2011), Bai et. al. (2012) e Ali (2012) comparam a Transformada SIFT com outras metodologias e concluem que a Transformada SIFT é superior na extração de características (contribuindo para a boa precisão dos mosaicos), embora possa apresentar um tempo de processamento mais lento em relação às demais metodologias. Além disso, essa transformada é invariante a transformações de rotação, translação, escala e alterações inerentes às condições de iluminação do ambiente (Machado et. al., 2008).

Um dos problemas da construção de mosaicos está na combinação das imagens para gerar um mosaico sem distorção. Essa distorção é causada pelo processo de construção do mosaico. Para minimizá-la, Lowe (2004) propõe um algoritmo que é aplicado no mosaico, no final do processo. Mais recentemente, Silva et. al. (2012b) propôs um algoritmo mais eficiente, que é aplicado ao longo do processo de construção do mosaico.

Dentre as aplicações de mosaicos, que usam imagens aéreas, são de interesse deste trabalho, aquelas voltadas para a área agrícola, com o objetivo de controlar extensas áreas de culturas, pastagens e florestas. Com este objetivo, este trabalho propõe uma metodologia para construção de mosaicos a partir de imagens aéreas agrícolas de alta resolução com minimização das distorções geradas.

1.2 Objetivos

Este trabalho propõe três metodologias para construir automaticamente mosaicos de imagens digitais de alta resolução, a partir de imagens aéreas sequenciais agrícolas, baseando-se na Transformada SIFT. As metodologias diferem na maneira como os cálculos são realizados. O padrão de processamento paralelo *OpenMP* é aplicado nas três metodologias, utilizando sistemas multinúcleos, para minimização de custo computacional.

Como objetivos secundários podem ser citados:

- Retificação das imagens adquiridas para minimizar possíveis distorções do processo de aquisição;
- Suavização das emendas das imagens durante o processo de construção dos mosaicos;
- Avaliação das metodologias desenvolvidas quanto ao custo computacional;
- Avaliação do paralelismo aplicado às metodologias desenvolvidas, utilizando sistemas multinúcleos;
- Avaliação do paralelismo com MPI;
- Avaliação da distorção dos mosaicos gerados, comparando os resultados com dois softwares livres.
- Avaliação da distorção em mosaicos constituídos por pares de imagens, comparando os resultados com dois softwares livres.

1.3 Justificativa

Este trabalho tem por objetivo propor uma metodologia para geração automática de mosaicos para a área agrícola. Os mosaicos aéreos agrícolas permitem direcionar vistorias de campo durante o ciclo do cultivo ou em datas posteriores à colheita, fornecendo um diagnóstico preciso da área de cultivo (Leboeuf, 2000). A partir daí, podem ser elaborados os mapas de recomendações: descompactação, fertilidade e aplicação de insumos em taxa variável, auxiliando nas tomadas de decisões.

A eficiência da tomada de decisão está ligada à obtenção rápida e precisa de

informações e também auxilia no controle de pragas, doenças e queimadas (Wulder et. al, 2004 e Lelong et. al, 2008).

Até o presente momento, a construção de mosaicos na agricultura, vem sendo feita de maneira semiautomática, necessitando obter o modelo digital do terreno, fazer a ortorretificação de imagens e colocação manual de marcadores (bandeirinhas), que são usadas para demarcar a área em questão. Posteriormente um software é utilizado para construir um mosaico desta área. Utilizando este tipo de implementação, a construção de mosaicos é demorada e trabalhosa.

Embora existam trabalhos na literatura para geração de mosaicos (CHO, BAE e DURAND, 2009; LI E GENG, 2010; KAYNIG et. al., 2010; LIU et. al., 2011; ZHONG et. al., 2012; SONG e LU, 2013 e GUI et. al., 2013), o trabalho proposto nesta tese propõe uma metodologia que acelera o processamento da geração de mosaicos em relação às metodologias convencionais. Além disso, inclui técnicas para minimizar distorções e avalia o paralelismo com o *OpenMP* para minimizar o custo computacional.

1.4 Contribuições

Como já mencionado na Seção 1.3, podem ser encontrados na literatura, muitos trabalhos para a construção de mosaicos, sendo que parte deles propõem acelerar a construção de mosaicos com novas metodologias baseadas em novos descritores de características. Diferentemente dessa abordagem, neste trabalho é proposta uma metodologia baseada na Transformada SIFT (metodologia 3) que faz os cálculos iniciais para a construção de mosaicos em imagens de baixa resolução, estimando a área de sobreposição nas imagens de alta resolução a partir destes valores. Além disso, os cálculos para a construção dos mosaicos são determinados na região de sobreposição, minimizando o custo computacional.

Outras contribuições adicionais podem ser mencionadas, tais como a avaliação das três metodologias propostas, a inclusão do algoritmo de Silva et. al. (2012b) para minimizar as distorções nas junções do mosaico ao longo do processo de construção do mesmo, implementação e avaliação do paralelismo com o padrão *OpenMP*, a avaliação da implementação com MPI para minimizar o custo

computacional e finalmente a comparação da distorção dos mosaicos gerados com a metodologia proposta e com os softwares livre, *PTGui* e *Autostitch*.

1.5 Estrutura do Trabalho

Este trabalho está dividido em 7 Capítulos e 10 Apêndices. Neste Capítulo foi apresentado o contexto no qual se insere este trabalho, o objetivo e as justificativas.

O Capítulo 2 apresenta a descrição detalhada das principais técnicas utilizadas neste projeto, que inclui geometria epipolar, retificação de imagens, Transformada SIFT (*Scale Invariant Feature Transform*), BBF (*Best Bin First*) e RANSAC (*Random Sample Consensus*), além de conceitos sobre registro e correspondência de imagens. No Capítulo 0 é apresentada uma revisão bibliográfica com trabalhos científicos para a geração de mosaicos com imagens digitais. O Capítulo 0 apresenta os conceitos sobre processamento paralelo e padrão *OpenMP* e *MPI*. No capítulo 5 são descritas as metodologias empregadas e como foram obtidas as imagens aéreas para a construção de uma base de imagens. O Capítulo 6 apresenta os resultados gerados e a discussão desses resultados. O Capítulo 7 apresenta as conclusões deste projeto de doutorado. No Apêndice A são apresentadas as imagens da base de imagem sem redimensionamento, utilizada para avaliação das metodologias. O Apêndice B mostra os mosaicos construídos pelas imagens do Apêndice A. O Apêndice C apresenta a base de imagens redimensionadas, utilizada para testes de distorções com outros softwares de licença gratuita. O Apêndice D mostra os mosaicos construídos pelas imagens do Apêndice C. O Apêndice E apresenta os mosaicos gerados pelos softwares de licença gratuita. Os demais apêndices apresentam os cálculos detalhados para avaliação das distorções.

2. Técnicas para Construção de Mosaicos de Imagens

2.1. Considerações Iniciais

Neste Capítulo são descritas as principais técnicas para a geração de mosaicos. Inicialmente são apresentados os conceitos sobre geometria epipolar (Seção 2.2) e retificação de imagens (Seção 2.3) para eliminar possíveis distorções nas imagens. Na Seção 2.4 é apresentada a Transformada SIFT para a extração de características, e suas etapas paralelizáveis na Seção 2.5. A BBF é apresentada na Seção 2.6 para acelerar a busca por características correspondentes. Na Seção 2.7 é apresentado o algoritmo RANSAC que exclui possíveis pontos falsos correspondentes entre imagens e estima a matriz homográfica.

2.2. Geometria Epipolar

Na fase de obtenção das imagens podem ocorrer distorções de translação, escala e rotação devido a mudanças de direção ou inclinação do avião por massa de ar. Existe então, a necessidade de ajustar (retificar) estas imagens. Para isto, é necessário entender o conceito de retificações (Seção 2.3). Mas para entender o conceito de retificações é necessário entender o conceito de geometria epipolar, usada para simplificar o processamento durante a busca dos pontos homólogos, pois com ela, a busca não precisa ser feita por toda a imagem, restringindo-se apenas à linha epipolar. A geometria epipolar será apresentada a seguir.

Os seres humanos possuem visão estéreo, fazendo com que consigam distinguir quais objetos estão mais próximos uns dos outros pela percepção. Cada um dos olhos observa o mundo de pontos de vista diferentes e o cérebro consegue extrair várias relações geométricas entre as imagens formadas em cada retina, conseguindo assim reconstruir o ambiente 3D de forma que seja possível perceber as diferenças de profundidade dos objetos que compõem a cena observada.

A simulação computacional do processo de visão que recria o ambiente 3D a partir de duas imagens é baseada na subárea da geometria, denominada geometria epipolar. Essa geometria já foi discutida em vários artigos, como por exemplo, nos trabalhos de Torr e Murray (1997), Zhang (1998), Armangué e Salvi (2003) e Roberto; Teichrieb e Kelner (2009).

Uma das maneiras de modelar a visão estéreo, por meio do conceito da geometria epipolar é através do uso de duas câmeras, como mostrado na Figura 1, sendo possível observar que cada câmera possui o seu próprio centro e orientação e também o seu próprio sistema de coordenadas (POLLEFEYS; KOCH; VAN-COOL, 1999).

Figura 1 - Esquema computacional da visão estéreo.

Fonte: Roberto; Teichrieb; Kelner, 2009.

Uma das relações possíveis entre pares de imagens, e de extrema importância é denominada *baseline*. Ela é uma reta que liga o centro C_1 da primeira câmera com o centro C_2 da segunda e ocorre em todos os casos de visão estéreo. O ponto de intersecção desta reta com o plano de imagem é chamado de epípolo e , cada uma das câmeras possui o seu respectivo epípolo, e_1 e e_2 , como pode ser observado na Figura 1.

Os pontos m_1 e m_2 da Figura 2 são a projeção de um ponto M em coordenadas reais, então pode-se dizer que M , C_1 , C_2 são coplanares e formam o

plano epipolar. Este plano intersecta com o plano de imagem de cada uma das câmeras, formando as linhas epipolares (de e_1 à m_1 e de e_2 à m_2). A Figura 2 ilustra a relação que as linhas epipolares possuem entre uma imagem e outra. Usando o ponto m_1 como referência, é possível definir um raio que parte de C_1 até m_1 . A partir deste raio, pode-se perceber que m_1 na realidade não é apenas a projeção de M , mas sim de todos os pontos que pertencem a este raio. Isto significa que é impossível determinar exatamente a posição espacial de um ponto projetado em uma imagem sem que haja outra imagem, capturada por uma segunda câmera em outra posição. Neste exemplo, m_2 é este segundo ponto de vista de M . Desta forma, a intersecção dos raios que vão de C_1 à m_1 e de C_2 à m_2 ocorre no ponto M (ORAM, 2001 e ROBERTO; TEICHRIEB; KELNER, 2009).

Figura 2 – Plano epipolar e seus principais elementos.
Fonte: Roberto; Teichrieb; Kelner, 2009.

Se o primeiro raio (de C_1 à m_1) for projetado na segunda imagem ele formará uma reta no plano projetivo, que é a linha epipolar de e_1 à m_1 e esta reta conterá o ponto m_2 . O mesmo acontece se o raio de C_2 à m_2 for projetado na primeira imagem. Então, pode-se concluir que para todos os pontos de uma imagem, seu correspondente na outra imagem estará na sua respectiva linha epipolar (ROBERTO; TEICHRIEB; KELNER, 2009).

Todas as linhas epipolares passam pelo epipolo da respectiva imagem e, independente da coordenada espacial do ponto M , todos os planos epipolares passam pela *baseline*, como mostra a Figura 3.

A partir da linha epipolar l_1 da primeira imagem é possível calcular a sua respectiva representação na segunda imagem (l_1 na segunda imagem) através da equação (1) e vice-versa:

$$l_1 = Fm_1 \quad (1)$$

sendo F a matriz fundamental 3x3 (uma representação algébrica da geometria epipolar entre duas imagens). F pode ser determinada a partir da equação (2):

$$m_2^T = F m_1 = 0 \quad (2)$$

Figura 3 – Representação de linhas epipolares e planos epipolares.
Fonte: Roberto; Teichrieb; Kelner, 2009.

Para casos em que o objetivo é calcular a linha epipolar l_2 na primeira imagem, a matriz fundamental também é utilizada, de acordo com a equação (3).

$$l_2 = F^T m_2 \quad (3)$$

Outro papel importante da matriz fundamental é calcular os epipolos da imagem. Eles são definidos como os núcleos da matriz fundamental, de acordo com as equações (4) e (5).

$$F e_1 = 0 \quad (4)$$

$$F^T e_2 = 0 \quad (5)$$

Também é possível encontrar linhas epipolares correspondentes, ou seja, considerando que a equação (1) é conhecida, é possível encontrar a linha epipolar l_2 (que contém o ponto m_1), mesmo sem conhecer o ponto m_2 na segunda imagem. Isto é possível porque existe uma matriz homográfica H que mapeia todos os pontos e retas da primeira imagem na segunda imagem e vice-versa. Esta matriz é encontrada a partir da matriz fundamental F e dos epipolos da imagem de acordo com a equação (6):

$$H = [e_2]_x F + e_2 a^T \quad (6)$$

sendo a variável a um vetor qualquer não nulo usado para garantir que a matriz H tenha uma inversa e, $[e_2]_x$ é a matriz antissimétrica do epipolo da segunda imagem, definida pela equação (7).

$$[e_2]_x = \begin{bmatrix} 0 & -e_2z & e_2y \\ e_2z & 0 & -e_2x \\ -e_2y & e_2x & 0 \end{bmatrix} \quad (7)$$

Conhecendo a matriz homográfica, as linhas epipolares correspondentes podem ser facilmente calculadas usando as equações (8) e (9).

$$l_1 = H^T l_2 \quad (8)$$

$$l_2 = H^T l_1 \quad (9)$$

A geometria epipolar é a base para a compreensão sobre retificações (Seção 2.3), que servem para determinar a reprojeção de um par de imagens a ser retificado.

2.3. Retificações

Apesar dos algoritmos SIFT, BBF e RANSAC serem os principais responsáveis por fazerem a detecção de pontos chave, existem problemas inerentes à junção das imagens propriamente ditas. Com relação a isso, é necessário ajustar as imagens, ou seja, retificá-las para minimizar distorções e tentar suavizar as junções. Há dois tipos de retificações mais facilmente encontradas na literatura: a retificação planar e a retificação cilíndrica. Esses tipos de retificação determinam de qual maneira o par de imagens a ser retificado deve ser reorganizado a partir de uma reprojeção (ROBERTO; TEICHRIEB; KELNER, 2009).

Usando o conceito de geometria epipolar, a retificação deve ajustar as imagens planares à *baseline* antes de uní-las, o que assegura que as linhas relacionadas terão a mesma altura. Observando-se que, se o epipolo da imagem é muito próximo da imagem, a imagem retificada ficará deformada. Quando o epipolo está localizado dentro da imagem, a imagem será esticada até ao infinito (ROY et al., 1997).

A retificação planar consiste em reprojetar as imagens em um plano comum, paralelo à *baseline*. Desta forma, quando a imagem for mapeada em uma região comum com este plano, têm-se a garantia que linhas epipolares correspondentes

estarão na mesma altura (FUSIELLO; TRUCCO; VERRI, 2000). Essa abordagem é relativamente simples de ser implementada, porém quanto mais próximo da imagem o epipolo estiver, maior será o tamanho da imagem retificada, fazendo com que o epipolo fique dentro da imagem, e resultando em uma imagem de tamanho infinito.

A alternativa é utilizar retificação cilíndrica, pois o epipolo nunca se localiza dentro da própria imagem, havendo projeção de cada ponto da imagem em um cilindro de raio Z , de mesma altura da imagem (AGAPITTO, 2001; ORAM, 2001). Isto é feito considerando o plano da imagem em um espaço tridimensional e o cilindro é posicionado tangenciando o plano de imagem, sobre a origem dos eixos, conforme Figura 4.

2.3.1. Retificação Cilíndrica

A ideia da projeção dos pontos da imagem em um cilindro pode ser vista na Figura 4, considerando o cilindro encostado ao plano da imagem. A projeção considera o centro da imagem com coordenadas $(0,0)$, no ponto central do plano da imagem, e cada ponto da imagem terá um sistema de coordenadas global dado como (x,y,z) . Esta posição projetada no cilindro terá as coordenadas (v,θ) , sendo v a altura do ponto de interseção entre o cilindro e a linha reta que liga a origem ao ponto (x,y,z) . O ângulo θ é o ângulo formado entre o eixo e esta linha reta (SZELISKI; SHUM, 1997). Este método é mais complexo de ser implementado e possui um alto custo computacional, pois todos os cálculos são realizados para cada pixel da imagem, em um espaço tridimensional (ROY et al., 1997).

Figura 4 – Projeção na retificação cilíndrica.
Fonte: Adaptado de Hu, 2006.

Considerando a origem dos eixos no centro deste cilindro, e assim coincidente com o centro óptico da lente, o raio do cilindro (Z) será a distância focal da lente. Para a imagem de uma câmera, o eixo Z deverá coincidir com o centro óptico (AGAPITO, 2001; ORAM, 2001). É necessária uma conversão das coordenadas dos pontos. Normalmente considera-se a origem dos eixos da imagem em seu canto superior esquerdo. Para efetuar a projeção, considera-se o centro da imagem, coordenada $(0,0)$, no ponto central do plano da imagem (Figura 5).

Figura 5 – Mudança nas coordenadas da imagem.
Fonte: Adaptado de Hu, 2006.

Cada ponto da imagem terá então uma coordenada no sistema global dada como (X, Y, Z) . Esta posição projetada no cilindro terá a coordenada (v, θ) , sendo que v é a altura do ponto de intersecção entre o cilindro e a reta que liga a origem ao ponto (X, Y, Z) . O ângulo θ é aquele formado entre o eixo Z e esta reta (SZELISKI; SHUM, 1997). Observando o cilindro pela vista superior, vê-se a imagem apresentada na Figura 6.

Figura 6 – Vista superior do cilindro e do plano da imagem.
Fonte: Adaptado de Hu, 2006.

O ângulo θ é calculado de acordo com a equação (10):

$$\theta = \tan^{-1}(x/z) \quad (10)$$

Para calcular v é necessário determinar o valor de X' , que pode ser calculado pela equação (11):

$$X' = Z \sin(\theta) \quad (11)$$

A altura v será calculada pela relação de triângulos, de acordo com a equação (12):

$$\frac{Y}{v} = \frac{X}{X'} \Rightarrow v = \frac{YX'}{X} \quad (12)$$

Substituindo X' obtém-se o valor de v de acordo com a equação (13):

$$v = \frac{YZ \sin(\theta)}{X} \quad (13)$$

Uma vez que as imagens pertencentes ao par de imagens foram retificadas, as mesmas estão prontas para que a Transformada SIFT extraia as respectivas características.

2.4. A Transformada SIFT

A Transformada SIFT foi desenvolvida em 1999 por David G. Lowe, professor do departamento de Ciência da Computação da *University of British Columbia* (Canadá).

A Transformada SIFT é capaz de extrair características de cada uma das imagens, criando uma coleção de descritores de características, sendo que cada um desses descritores é invariante à escala, rotação e parcialmente invariante à mudanças de iluminação e ponto de vista (LOWE, 2004), ou seja, a SIFT busca identificar pontos estáveis ao invés de segmentar regiões ou objetos para produzir um conjunto de descritores.

Os pontos considerados estáveis são pontos na imagem que tendem a ser invariantes à escala e orientação, isto é, pontos que podem ser reconhecidos como

similares independentemente do valor absoluto da escala e orientação que os descrevem (SILVA, 2012).

As características fornecidas pela Transformada SIFT são bem localizadas em ambos os domínios, o da frequência e o do espaço, reduzindo assim a probabilidade de não haver correspondência das características por oclusão ou ruído. As características são altamente distintas, permitindo que uma simples característica seja corretamente correspondida com alta probabilidade diante de um grande banco de dados de características. (LOWE, 2004).

A grande variabilidade de conteúdo apresentada nas imagens aéreas agrícolas obtidas em ambientes não controlados também justifica a escolha da Transformada SIFT. Ambiente não controlado, neste trabalho, refere-se ao local de obtenção das imagens. As imagens podem apresentar diferentes escalas, diferentes inclinações, diferentes tipos de iluminação e reflexo de luminosidade por conta das condições de obtenção das imagens.

Os quatro principais estágios que compõem a Transformada SIFT para gerar a coleção de descritores são (LOWE, 2004):

1. **Deteção de extremos no Espaço Escala:** Neste primeiro estágio é feita a procura por todas as escalas e posições de uma imagem. Para isto é utilizada uma função conhecida como Diferença da Gaussiana, para identificação dos potenciais pontos de interesse, que são invariantes à escala e orientação. Esta é a parte mais custosa do algoritmo.
2. **Localização dos pontos chave:** Para cada local candidato, é determinada a sua posição e escala. Os pontos chave são selecionados baseados nas medidas de sua estabilidade.
3. **Definição da Orientação:** Uma ou mais orientações são atribuídas para cada ponto chave localizado, baseadas em direções do gradiente. Todas as operações posteriores são realizadas sobre os dados da imagem que foram considerados ponto chave e que foram transformados em relação à orientação, escala e localização, proporcionando invariância a estas transformações.
4. **Descritor dos Pontos Chave:** Os gradientes da imagem são mensurados na escala selecionada, na região ao redor de cada ponto chave, sendo criados histogramas de orientações para compor o descritor.

Estes quatro estágios serão detalhados respectivamente nas seções 2.4.1, 2.4.2, 2.4.3 e 2.4.4.

2.4.1. Detecção de Extremos no Espaço Escala

Esta primeira etapa de detecção de pontos chave identifica posições e escalas que podem ser repetidamente atribuídas ao mesmo objeto sob diferentes pontos de vista. A detecção de locais que são invariantes à mudança de escala em uma imagem, pode ser realizada, procurando por características estáveis em todas as escalas possíveis, por meio de uma função denominada Espaço Escala (WITKIN, 1983). Koenderink (1984) e Lindeberg (1994) mostraram que o único *Kernel* (núcleo) possível do espaço escala é a função Gaussiana.

O espaço escala de uma imagem é definido como uma função, $L(x,y,\sigma)$ que é obtida a partir da convolução de uma gaussiana de escala variável, $G(x,y,\sigma)$, com uma imagem de entrada, $I(x,y)$, conforme equação (14).

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y) \quad (14)$$

sendo $*$ a operação de convolução em x e y e, a gaussiana é definida conforme a equação (15):

$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \quad (15)$$

Neste filtro, o parâmetro σ define o fator de suavização de G .

Para uma eficiente detecção de posições de pontos chave estáveis, Lowe (1999) propõe o uso dos extremos do espaço escala na função da diferença da Gaussiana, *DoG* (*Difference of Gaussian*), convoluída com a imagem de entrada $I(x,y)$. A *DoG* é calculada, fazendo-se a diferença de duas imagens filtradas (G) com escalas próximas (separadas por uma constante multiplicativa K), conforme a equação (16).

$$\begin{aligned} DoG(x,y,\sigma) &= (G(x,y,K\sigma) - G(x,y,\sigma)) * I(x,y) \\ &= L(x,y,K\sigma) - L(x,y,\sigma) \end{aligned} \quad (16)$$

Esta convolução é a diferença entre imagens suavizadas por um filtro Gaussiano em escalas σ e $K\sigma$. A função *DoG* consegue detectar variações de intensidade na imagem, como contornos, eliminar detalhes indesejados e realçar características fortes. Variando-se σ , é possível encontrar características em diferentes escalas.

Na Figura 7 é mostrado um esquema eficiente para construir a *DoG*, através de uma pirâmide de imagens. Primeiramente uma dada imagem de entrada é dobrada de tamanho, criando mais pontos de amostragem e possibilitando detectar mais pontos chaves por um fator de quatro; isto é feito através de interpolação linear. Esta imagem é incrementalmente convoluída com Gaussianas para produzir imagens separadas por uma constante K no espaço escala, gerando as imagens suavizadas (L), representadas na coluna esquerda da Figura 7. Na primeira convolução, é utilizado o parâmetro σ (sem fator multiplicativo) na gaussiana. Na segunda convolução é usado $K\sigma$, na terceira é $K^2\sigma$, etc. As imagens de escalas adjacentes da coluna da esquerda da Figura 7 são subtraídas para gerar a *DoG*, como pode ser observado na coluna da direita da Figura 7. O conjunto de imagens L e *DoG* é denominado de oitava.

Uma técnica para gerar quantas *DoG* forem necessárias, é determinar quantas imagens *DoG* entre os extremos, s , deseja-se criar, sendo s referido como intervalo. Na Figura 7 é possível verificar que $s=2$, através das duas figuras destacadas com estrela, na coluna da direita. Para s intervalos são necessárias $s+3$ imagens na coluna da esquerda da Figura 7. Com isso é possível calcular o valor de K através da expressão $K = 2^{1/s}$. Neste trabalho foi adotado $s=2$, de acordo com Lowe (2004) e Hess (2010), uma vez que os autores mencionam que para valores de s maiores do que dois é gerado aumento de custo computacional. O algoritmo da Transformada SIFT em linguagem C encontra-se disponibilizado para pesquisa científica em Hess (2010).

Figura 7 – Representação da obtenção da DoG.
Fonte: LOWE, 2004.

Para o exemplo da Figura 7, sendo $s=2$, $K=\sqrt{2}$. A primeira gaussiana da Figura 7 usa o parâmetro σ , a segunda usa $K\sigma$, ou seja, $(\sqrt{2}\sigma)$, a terceira usa $K^2\sigma$, ou seja, $((\sqrt{2})^2=2\sigma)$, a quarta usa $K^3\sigma$, ou seja, $(K^2*K*\sigma=2K\sigma)$ e a quinta usa $K^4\sigma$, ou seja, $(K^3*K\sigma=2K^2\sigma)$.

Uma vez que uma oitava esteja completa, a imagem suavizada L com 2σ é redimensionada, dividindo a sua dimensão pela metade (escolhendo cada segundo pixel em cada linha e coluna). Esta nova imagem torna-se a primeira imagem da próxima oitava, como pode ser observado na Figura 7. A quantidade de oitavas depende da dimensão da imagem original e os redimensionamentos ocorrem até atingir uma dimensão mínima de aproximadamente 4x4 pixels. Na Transformada SIFT, Lowe definiu a Equação (17), a partir das dimensões da imagem (largura e altura) para calcular a quantidade de oitavas a serem processadas.

$$Oitavas = \lceil \log(\min(largura, altura)) / (\log(2) - 2) \rceil \quad (17)$$

Para detectar o máximo e o mínimo local da DoG, cada ponto desta imagem (identificado por um X na Figura 8) é comparado com seus 8 vizinhos e também com os 9 vizinhos da escala superior e os 9 da escala inferior (Figura 8). As três imagens utilizadas para a detecção dos extremos são da mesma oitava, ou seja, tem a mesma dimensão. Um ponto é selecionado somente se seu valor for maior ou menor do que o de todos os seus 26 vizinhos (LOWE, 2004). O custo dessa verificação é relativamente baixo, devido ao fato que muitos pontos amostrados serão eliminados a partir das primeiras verificações.

Não se deve confundir escala superior e inferior com oitavas. Oitavas contêm imagens com mesma dimensão. Quando se diz escala superior e inferior aqui, está fazendo-se referência à valores diferentes de σ

Figura 8 – Máximos e mínimos das imagens da *DoG*.
Fonte: LOWE, 2004.

Lowe (2004) faz um estudo do número de escalas amostradas por oitava pela porcentagem de repetição de pontos chave detectados na mesma posição e escala em uma imagem transformada, concluindo que a maior porcentagem de repetição de pontos chave detectados na mesma posição é obtida quando são amostradas três escalas por oitava na imagem da *DoG*. Lowe também afirma que o número de pontos chave detectados cresce com o aumento do número de escalas amostradas e, que o total de correspondências corretas também aumenta. Porém, isto tem um custo computacional alto e por esta razão ele faz uso de três amostras de escala por oitava em seu trabalho (Figura 9).

Figura 9 – Determinação da quantidade de amostra por oitava.
Fonte: Lowe, 2004

Lowe (2004) também faz um estudo da determinação do valor prévio de σ , que é o valor mínimo de escala da função Gaussiana aplicado à primeira imagem de

cada oitava, e para cada imagem de cada oitava aplica-se o fator multiplicativo K e novamente conclui que a repetição de pontos chave detectados aumenta com σ , mas que este aumento resulta em um custo computacional alto. Por isso Lowe usa $\sigma=1,6$, o qual provê uma boa taxa de repetição, como pode ser observado na Figura 10.

Figura 10 – Determinação do fator sigma.
Fonte: Lowe, 2004

A seguir, é apresentado de maneira mais intuitiva, um exemplo na Figura 11, do processo de criação de oitavas, contendo 3 oitavas e 5 níveis cada. Cada oitava é constituída por cinco imagens, sendo que cada uma delas possui uma escala diferente (σ) e, portanto, diferentes níveis de detalhes. Neste exemplo foram utilizados os mesmos valores para K e σ , da Figura 7. Cada oitava seguinte usa como referência a imagem da oitava anterior, que possui valor 2σ .

Figura 11 – Exemplo do processo de criação de oitavas.
Fonte: HECKEL, 2008.

Para cada oitava da Figura 11, calcula-se a diferença de imagens em escalas adjacentes. Esta operação produz a pirâmide de Diferença de Gaussiana (*DoG*), conforme ilustra a Figura 12. Nesta pirâmide são determinados os extremos (máximo e mínimo) que serão utilizados na próxima etapa para a localização exata dos pontos chaves.

Figura 12 – Exemplo da Diferença da Gaussiana (*DoG*).
Fonte: HECKEL, 2008.

2.4.2. Localização Exata dos Pontos Chave

Todos os pontos detectados como extremos são possíveis candidatos a pontos chave, e o próximo passo é executar um ajuste detalhado aos pixels próximos em relação à escala, localização, e relação de curvaturas principais. Essa informação permite rejeitar pontos que tenham baixo contraste (são sensíveis ao ruído) ou localizados ao longo da borda. A localização dos pontos chave é especialmente importante para as últimas oitavas, porque o espaçamento amostral destas representa grandes distâncias em relação à imagem base. A implementação inicial desse enfoque (LOWE, 1999) localizava pontos chave em locais e escala de pontos de amostragem centrais.

Como melhoria, Brown e Lowe (2002) desenvolveram um método para ajustar uma função quadrática 3D aos pontos amostrados anteriormente, para determinar a localização interpolada do máximo. Esta técnica mostrou trazer melhorias para a correspondência e estabilidade dos pontos.

Para cada ponto analisado, é usada a expansão de *Taylor* (até os termos quadráticos) da função espaço escala, $D(x,y,\sigma)$, deslocada, para que a origem esteja no ponto de amostragem (LOWE, 2004) – equação (18).

$$D(x) = D + \frac{\partial D^T}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 D}{\partial x^2} x \quad (18)$$

D e suas derivadas (primeira e segunda ordem) são avaliadas no ponto amostrado e $x = (x, y, \sigma)^T$ é o *offset* (deslocamento) deste ponto. Ou seja, D é o valor da função $D(x, y, \sigma)$ no ponto avaliado, x é o deslocamento em relação a este ponto e $D(x)$ é a aproximação do valor de $D(x, y, \sigma)$ interpolado para um ponto transladado com *offset* x .

Os coeficientes quadráticos são computados aproximando-se as derivadas através das diferenças entre pixels das imagens já filtradas. A localização do extremo $\hat{x}$, é determinada ao se fazer a derivada de $D(x)$ em relação à x , igualando o resultado a zero – equação (19).

$$\hat{x} = - \frac{\partial^2 D^{-1}}{\partial x^2} \frac{\partial D}{\partial x} \quad (19)$$

Como sugerido por Brown e Lowe (2002), a matriz *Hessiana* e a Derivada D são aproximadas pelas diferenças de pontos amostrados vizinhos. O sistema linear 3x3 resultante pode ser resolvido com custo mínimo. Se o *offset* for maior que 0,5 em qualquer dimensão, isto significa que o extremo se aproxima mais de outro ponto de amostra. Neste caso, o ponto de amostragem é mudado e a interpolação é realizada para este novo ponto. O *offset* $\hat{x}$ final é adicionado à localização do ponto analisado para se estimar a interpolação para a localização do máximo (LOWE, 2004).

O valor da função no extremo, $D(\hat{x})$, é útil para rejeição de extremos instáveis com baixo contraste. Isto pode ser obtido substituindo-se a equação (19) em (18), resultando na equação (20):

$$D(\hat{x}) = D + \frac{1}{2} \frac{\partial D^T}{\partial x} \hat{x} \quad (20)$$

Lowe (2004) em seus experimentos rejeitou valores de $|D(D(\hat{x}))|$ inferiores a 0,03, assumindo que os pixels da imagem estejam entre [0, 1].

A Figura 13 mostra os efeitos da seleção de pontos chave em uma imagem natural. A imagem é de baixa resolução (233 por 189 pixels) e pontos chave são mostrados como vetores dando a localização, dimensão e orientação de cada ponto chave (atribuição de orientação será descrito na Seção 2.4.3).

A Figura 13a mostra a imagem original. A Figura 13b contém 832 pontos chave referentes a todos os máximos e mínimos da *DoG*. A Figura 13c contém 729 pontos restantes, após remover pontos com $|D(D(\hat{x}))|$ inferior a 0,03. A Figura 13d contém 536 pontos chave após aplicação de r (razão) nas relações das curvaturas principais (que será descrito na Seção 2.4.2.1).

Figura 13 – Exemplo da seleção de pontos chave.

(a) – imagem original (b) – imagem com 832 pontos chaves (c) – imagem com 729 pontos chaves (d) - imagem com 536 pontos chaves

Fonte: LOWE, 2004

2.4.2.1. Eliminando Respostas nas Bordas

Para a estabilidade dos pontos chave não é suficiente apenas rejeitar pontos chave com baixo contraste. A função *DoG* tem boa resposta nas bordas, mesmo se a localização ao longo da borda não for bem determinada e portanto instável até para pequenos valores de ruído.

Um pico mal definido na *DoG* terá uma grande curvatura principal pela extremidade das bordas, mas uma pequena curvatura na direção perpendicular. As curvaturas principais podem ser calculadas através da matriz *Hessiana 2x2* (H) na equação (21), a qual foi determinada na localização e escala dos pontos chave:

$$H(x, y) = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \quad (21)$$

As derivadas são estimadas pelas diferenças de pontos amostrados vizinhos. D_{xx} na equação (22) é a derivada segunda em relação a x ; D_{yy} na equação (23) é a derivada segunda em relação a y ; D_{xy} na equação (24) é a derivada de $D(x, y, \sigma)$ na localização e escala dados em relação a x e então a y (BELO, 2006).

$$D_{xx} = D(x + 1, y, \sigma) - 2D(x, y, \sigma) + D(x - 1, y, \sigma) \quad (22)$$

$$D_{yy} = D(x, y + 1, \sigma) - 2D(x, y, \sigma) + D(x, y - 1, \sigma) \quad (23)$$

$$D_{xy} = \left(\frac{D(x-1, y+1, \sigma) - D(x+1, y+1, \sigma)}{+D(x+1, y-1, \sigma) - D(x-1, y-1, \sigma)} \right) / 4 \quad (24)$$

Os autovalores de H são proporcionais às curvaturas principais de D . Não é necessário calcular os autovalores, pois o que se busca é a razão entre as curvaturas (LOWE, 2004). Sendo α o autovalor de maior magnitude e β o de menor magnitude, pode-se calcular a soma dos autovalores pelo traço (Tr) de H e o produto destes autovalores pelo determinante (Det), de acordo com a equação (25):

$$\begin{aligned} Tr(H) &= D_{xx} + D_{yy} = \alpha + \beta, \\ Det(H) &= D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta. \end{aligned} \quad (25)$$

Caso o determinante seja negativo, as curvaturas têm sinais diferentes e o ponto é descartado, não sendo um extremo. Sendo r a razão entre o maior e o menor autovalor ($r=\alpha/\beta$). Então,

$$\frac{Tr(H)^2}{Det(H)} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r} \approx r, \quad (26)$$

A equação (26) depende apenas da razão entre os autovalores, e não de seus valores individuais. O valor $(r+1)^2/r$ é mínimo quando os dois autovalores são iguais e aumenta com r . É possível verificar se a razão das curvaturas principais está abaixo de um dado *threshold*, usando a equação (27):

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r} \quad (27)$$

Assim, eliminam-se pontos chaves que não são estáveis, apesar de estarem próximos de extremidades.

Após determinação de quais pontos vão ser utilizados como pontos chave, na sequência, na Seção 2.4.3 será apresentado como estabelecer uma ou mais orientações para cada um dos pontos chave e na Seção 2.4.4 é apresentado como construir descritores com relação às orientações definidas.

2.4.3. Determinação da Orientação

Até o estágio anterior do algoritmo foi apresentada a invariância dos pontos chave para escala. Nesta Seção será determinada a orientação para cada ponto chave. Ao se atribuir uma orientação consistente para cada ponto chave baseada nas propriedades da imagem, podem-se representar os descritores em relação a esta orientação, conseguindo-se assim invariância quanto à rotação (LOWE, 2004).

A escala σ do ponto chave é usada para selecionar a imagem suavizada L , que tem a escala mais próxima ao ponto chave e na oitava referente ao ponto

avaliado, de forma que todos os cálculos são executados com invariância à escala. Para cada imagem $L(x,y,\sigma)$, nessa escala σ , são calculadas a magnitude do gradiente $m(x,y)$ e a orientação $\theta(x,y)$, usando a diferença de pixels (LOWE, 2004), conforme mostram as equações (28) e (29).

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \quad (28)$$

$$\theta(x,y) = \tan^{-1}((L(x,y+1) - L(x,y-1)) / (L(x+1,y) - L(x-1,y))) \quad (29)$$

Um histograma de orientações é constituído com as orientações dos gradientes dos pontos amostrados que se encontram em uma região ao redor do ponto chave. Esse histograma possui 36 regiões (*bins*), cobrindo os 360° de orientações. Cada amostra na vizinhança do ponto chave, adicionada ao histograma, é ponderado pela magnitude do seu gradiente e por uma janela Gaussiana circular com um σ que é 1,5 vezes maior do que o σ do ponto chave. A Função Gaussiana é aplicada para reduzir a importância dos gradientes mais distantes do ponto chave central.

O maior pico do histograma é detectado e outros picos que correspondam a 80% do maior pico são usados para criar um ponto chave com aquela orientação. Finalmente, uma parábola é ajustada para os três valores de histograma mais próximos de cada pico, para interpolar a posição do pico, para melhor precisão.

2.4.4. O Descritor da Imagem Local

As operações anteriores atribuíram para cada ponto chave a localização, escala e orientação. Nesta Seção é calculado um descritor para a imagem que seja tão invariante quanto possível às variações restantes, tais como mudança na iluminação ou ponto de vista 3D.

2.4.4.1. Representação do Descritor

Como explicado nas seções anteriores, as magnitudes e orientações dos gradientes da imagem são amostradas ao redor da localização do ponto chave,

usando a escala do ponto chave para selecionar o nível de borramento gaussiano da imagem (LOWE, 2004).

O cálculo do descritor de cada ponto chave é feita em relação à orientação definida na Seção 2.4.3. Primeiramente é selecionada a imagem filtrada L por meio do filtro Gaussiano referente à escala e oitava relativas ao ponto chave. Para se obter invariância à orientação, as coordenadas do descritor e das orientações dos gradientes destes pontos são rotacionadas em relação ao ponto chave. Para maior eficiência, os gradientes são calculados para todos os níveis da pirâmide das imagens convoluídas com a função de diferença de filtros Gaussianos, e esses são ilustrados com pequenas setas de cada local amostrado no lado esquerdo da Figura 14. Uma função Gaussiana é utilizada para dar peso à magnitude do gradiente de cada ponto na vizinhança do ponto chave, σ é escolhido igual à metade da largura da janela em que será calculado o descritor.

O descritor de pontos chave é mostrado no lado direito da Figura 14, e para cada região é construído um histograma de orientação de oito direções, com comprimento de cada seta correspondendo à magnitude de cada entrada do histograma.

Figura 14 – Criação do descritor de pontos chave.
Fonte: Lowe, 2004.

O peso referente à magnitude de cada pixel foi atenuado pela função Gaussiana. A função Gaussiana não é aplicada de modo idêntico ao da Seção anterior. É utilizado um peso t para interpolar a direção relativa no histograma.

O descritor é composto por um vetor contendo os valores de todos os histogramas de orientações de entrada, correspondendo ao tamanho das setas representadas, do lado direito na Figura 14. A referida figura mostra uma matriz 2x2 com 8 orientações em cada posição, ou seja, um vetor característico com 32

elementos para cada ponto chave (LOWE, 2004). Lowe afirma que os melhores resultados foram obtidos quando se usou uma matriz de 4x4 histogramas, com 8 direções em cada, obtendo um vetor com 128 características para cada ponto chave.

O vetor de elementos descritores é modificado para reduzir os efeitos na mudança de iluminação. O primeiro passo é normalizar o vetor, e em seguida fazer uma mudança no contraste da imagem, no qual cada valor de pixel é multiplicado por uma constante. Uma mudança no brilho na qual uma constante é adicionada para cada pixel da imagem não irá afetar os valores dos gradientes, pois eles são computados de diferentes pixels. Dessa forma, o descritor é invariante a pequenas mudanças de iluminação. Mudanças não lineares de iluminação podem ocorrer devido à saturação das câmeras ou devido a mudanças de iluminação que afetem superfícies 3D com diferentes orientações. Esses efeitos podem causar uma grande mudança nas magnitudes relativas para alguns gradientes, mas afetam menos as orientações dos gradientes. Com isso consegue-se reduzir a influência das grandes magnitudes do gradiente com um valor de limiar. Após a normalização, todos os valores acima de um determinado limiar são ajustados para este limiar. Isto é feito para que direções com magnitude muito grande não dominem a representação do descritor. É sugerido usar um valor de limiar 0,2, e então o vetor é normalizado novamente para a unidade de tamanho. De acordo com Lowe (2004) o valor 0,2 foi obtido por meio de testes usando imagens com diferentes iluminações para um mesmo objeto 3D.

Há dois parâmetros que podem ser usados para variar a complexidade do descritor: o número de orientações, q , nos histogramas e o tamanho, n , na matriz $n \times n$ do histograma de orientações. O tamanho do vetor do descritor resultante é qn^2 . Com isto será possível discriminar melhor as características em uma base de dados muito grande, mas os descritores serão mais sensíveis à distorção de forma e oclusão (LOWE, 2004; CRUZ, 2008).

Para cada imagem, são construídos diversos descritores, cada um referente a um ponto chave. Quando se aplica a Transformada SIFT em uma imagem, tem-se como resultado, portanto, um conjunto de descritores. Na Seção 2.6 será visto como encontrar os pontos correspondentes (pontos comuns) entre duas imagens, de acordo com os pontos chaves detectados pela Transformada SIFT.

2.5. Etapas Paralelizáveis da Transformada SIFT

A Transformada SIFT exige alto custo computacional e as quatro etapas descritas nas seções 2.4.1, 2.4.2, 2.4.3 e 2.4.4 podem ser paralelizadas por alguma técnica de processamento paralelo para minimizar o alto custo computacional envolvido nesta transformada, sendo que as etapas de Detecção de Extremos no Espaço Escala e Representação do Descritor são as mais custosas (WARN et. al, 2009; HUANG et. al., 2011).

2.6. Correspondência das Características

Uma vez aplicada a Transformada SIFT sobre as imagens, é possível encontrar a correspondência entre duas imagens de acordo com os pontos chaves detectados. Para isso, é feita a comparação dos descritores da primeira imagem com os da segunda, encontrando os melhores pontos chave candidatos da segunda imagem que correspondam aos da primeira imagem. Após serem encontrados os pontos em comum entre duas imagens, estes pontos são usados para se encontrar uma função (matriz homográfica) que mapeie as posições dos pontos de uma imagem para as posições dos pontos correspondentes na outra imagem; isto será visto na Seção 2.7 (BELO, 2006; LOWE, 2004).

A seguir é mostrado um esquema resumido de como encontrar a correspondência de características entre duas imagens. Sejam duas imagens I_1 e I_2 para as quais se deseja encontrar os pontos correspondentes. Os descritores são respectivamente di_1 e dj_2 , sendo i e j os índices para cada um dos descritores de cada imagem, k é o tamanho de cada descritor e m é a magnitude de cada valor dos vetores (BELO, 2006). As equações (30) e (31) apresentam di_1 e dj_2 .

$$di_1 = (m1i_1, m1i_2, m1i_3, \dots, m1i_k) \quad (30)$$

$$dj_2 = (m2j_1, m2j_2, m2j_3, \dots, m2j_k) \quad (31)$$

A correspondência é realizada encontrando-se o melhor candidato dj_2 para determinado di_1 e, isto é feito procurando-se o vizinho mais próximo de di_1 entre

todos os possíveis candidatos k em dj_2 , incluindo as falsas correspondências, que serão eliminadas em uma etapa posterior. O melhor candidato é definido por dj_2 , que possui a menor distância euclidiana em relação à di_1 , ou seja, deseja-se encontrar j para todo i que minimize a equação (32):

$$|di_1 - dj_2| = \sqrt{(m_1i_1 - m_2j_1)^2 + (m_1i_2 - m_2j_2)^2 + \dots + (m_1i_k - m_2j_k)^2} \quad (32)$$

Isto é feito para todo i de modo a serem encontrados todos os pares de descritores correspondentes nas imagens I_1 e I_2 .

De acordo com Lowe (2004), o método mais eficiente para identificar os vizinhos mais próximos em espaços de grande dimensão é a busca exaustiva. No entanto, a busca exaustiva requer um custo computacional considerável e, devido a isso, é usado o algoritmo BBF (*k-d tree* modificada) no lugar da busca exaustiva, que retorna o vizinho mais próximo com mais rapidez. Este algoritmo será visto na Seção 2.6.1.2.

Muitos dos pares encontrados correspondem a falsas correspondências, portanto, as correspondências serão refinadas e os falsos pares descartados. Isto será visto na Seção 2.7. Na próxima Seção são apresentados os conceitos para se buscar os vizinhos mais próximos com maior rapidez.

2.6.1. Algoritmos de Busca

Como já mencionado na Seção anterior, a busca exaustiva é o método mais eficiente para a identificação dos vizinhos mais próximos em um banco de dados de grande dimensão. Os melhores algoritmos, tal como *K-d tree* (FRIEDMAN et al., 1977) não oferecem aumento de velocidade sobre busca exaustiva acima de 10 dimensões. Para dimensões superiores a 10, Lowe (2004) sugere que seja utilizado o algoritmo denominado *Best-Bin-First* (BBF) (BEIS; LOWE, 1997), que é uma modificação do algoritmo *K-d tree*. Como o algoritmo BBF é uma modificação do *K-d tree* (uma forma de árvore de busca binária balanceada), na próxima Seção serão apresentados os fundamentos deste último.

Uma árvore *k-d tree* (FRIEDMAN et al., 1977) é construída e nela é realizada uma busca exaustiva para encontrar a correspondência dos pontos chave entre as

imagens. O algoritmo BBF usa uma ordenação de busca modificada para o algoritmo da árvore *k-d tree*, de forma que os descritores são procurados para suas distâncias mais próximas do local de consulta. Um bom resultado pode ser conseguido com baixo custo, interrompendo a busca depois que um número específico de vizinhos tenha sido explorado.

2.6.1.1. O Algoritmo *K-d tree*

Em Berg et al. (2008) é descrita a construção de uma árvore *k-d tree* a partir de um conjunto de pontos P bidimensionais (x,y) . Primeiramente o conjunto é dividido na coordenada x , depois na coordenada y . Em seguida, novamente é dividido na coordenada x , e assim por diante. Na raiz, o conjunto P é dividido com uma linha vertical ℓ em dois subconjuntos aproximadamente de tamanhos iguais. A linha de separação é armazenada na raiz. P_{left} , o subconjunto de pontos para a esquerda ou na linha de separação, é armazenado na subárvore esquerda, e P_{right} , o subconjunto à direita, é armazenado na subárvore direita.

No filho esquerdo da raiz, P_{left} é dividido em dois subconjuntos, com uma linha horizontal; os pontos abaixo são armazenados na subárvore esquerda do filho esquerdo, e os pontos acima são armazenados na subárvore direita. O filho esquerdo armazena a linha de divisão. Da mesma forma, o conjunto P_{right} é dividido com uma linha horizontal em dois subconjuntos, que são armazenados na subárvore esquerda e direita do filho direito (SILVA, 2012).

Nos netos da raiz, ocorre novamente uma divisão com uma linha vertical. Em geral, os nodos cuja profundidade é “par” são divididos com uma linha vertical, e os nodos cuja profundidade é “ímpar” são divididos com uma linha horizontal. A Figura 15 ilustra como é feita a divisão (lado esquerdo) assim como a correspondente árvore binária (lado direito). Uma árvore como essa é chamada de *k-d tree*, onde k representa a dimensão. A árvore da Figura 15 é de dimensão 2.

Figura 15 – Exemplo de *K-d tree*.
Fonte: BERG et al., 2008

Pode-se construir a árvore *k-d tree* utilizando um procedimento recursivo, e nele são utilizados dois parâmetros: um conjunto de pontos P e um número inteiro (representando a profundidade utilizada nas chamadas recursivas). O parâmetro de profundidade é zero na primeira chamada. A profundidade é importante, pois determina se a divisão ocorre com uma linha vertical ou uma linha horizontal. O processo retorna a raiz da árvore *k-d tree*.

No primeiro nível da árvore (raiz), os dados são divididos em duas metades por um hiperplano ortogonal para uma dimensão escolhida, com um valor limiar. Geralmente, essa divisão é feita em média na dimensão com a maior variação no conjunto de dados. Ao comparar o vetor da consulta com o valor de particionamento, é fácil determinar a que metade os dados do vetor de consulta pertence. Cada uma das duas metades dos dados é então dividida recursivamente da mesma forma para criar uma árvore binária completamente balanceada. Na parte inferior da árvore, cada nodo da árvore corresponde a um único ponto do conjunto de dados, embora em algumas implementações, os nodos folha podem conter mais de um ponto. A altura da árvore será $\log_2 N$, sendo N o número de pontos no conjunto de dados (HARTLEY, 2003).

A árvore *k-d tree* foi introduzida em Bentley (1975) como uma generalização de uma árvore binária de dimensão elevada. Diversas variações na construção de uma árvore, incluindo a seleção aleatória do valor de particionamento, foram sugeridas. Mais tarde, uma árvore *k-d tree* otimizada, com um tempo logarítmico de busca, foi proposto em Friedman (1977). No entanto, essa busca em tempo

logarítmico não se aplica a árvores de grande dimensão, onde o tempo de busca pode tornar-se quase linear.

Assim, embora as árvores *k-d trees* sejam eficazes e eficientes em baixas dimensões, a sua eficácia diminui para os dados com altas dimensões. Isto porque com dados de alta dimensão, uma árvore *k-d tree* geralmente leva muito tempo para encontrar a solução ideal. Busca-se encontrar o mínimo absoluto com um desempenho probabilístico dentro de limites razoáveis com o algoritmo BBF (SILVA, 2012).

2.6.1.2. O Algoritmo BBF

A modificação apresentada anteriormente estende o domínio das *K-d tree* para uma recuperação rápida do vizinho mais próximo. A pesquisa é realizada de acordo com a estrutura da árvore, que depende somente dos pontos armazenados e não leva em conta a posição do ponto de consulta. Uma ideia para melhorar as pesquisas, é olhar para os *bins* (nodos da árvore) em ordem de aumento das distâncias a partir do ponto de consulta, ou seja, as buscas são feitas na ordem de suas distâncias mais próximas a partir do ponto de consulta. A distância de um *bin* é definida como a mínima distância entre q (Figura 16) e qualquer ponto no limite do *bin*. Todo este processo deu origem ao algoritmo BBF.

Essa busca pode ser facilmente implementada com uma pequena quantidade de *overhead* usando uma fila de prioridade para a determinação eficiente da ordem de busca. Durante a pesquisa do vizinho mais próximo, quando uma decisão é feita em um nó interno para ramificar em certa direção, uma entrada é adicionada à fila de prioridade para armazenar informações sobre as opções não tomadas. Isto inclui a posição atual da árvore e a distância do ponto de consulta a partir do nó. Depois que um nó folha for examinado, a entrada de cima na fila de prioridade é removida e utilizada para continuar a pesquisa no ramo que contém a *bin* mais próxima (BEIS; LOWE, 1997).

O BBF encontra o vizinho mais próximo para uma grande parte das consultas, e encontra um vizinho muito próximo, nos demais casos.

Um exemplo da abordagem BBF pode ser vista na Figura 16 para uma *K-d tree* contendo 8 pontos e $k=2$, que examina primeiro os *bins* que estão mais perto do

ponto de consulta q . Neste caso, a BBF deverá reduzir o número de nós folha examinados desde que o vizinho mais próximo seja o *bin* mais adjacente no espaço (logo abaixo de q) ao invés do bin mais próximo na estrutura da árvore (a esquerda de q).

Figura 16 – Exemplo de busca BBF.
Fonte: Beis e Lowe (1997).

Toda ramificação da árvore pode ser cortada se a região do espaço que ela representa está além do ponto de consulta q para o vizinho mais próximo, contendo uma distância superior a um raio de busca estabelecido. A busca termina quando todas as ramificações tenham sido podadas. Este processo pode ser muito eficiente em espaços com baixas dimensões, mas em altas dimensões há muito mais nodos adjacentes para o nodo central que deve ser examinado, e o desempenho degrada rapidamente. De forma interessante, uma grande porção dessa busca é gasta examinando nodos na qual somente uma pequena fração do seu volume poderia possivelmente prover o vizinho mais próximo (SILVA, 2012).

O BBF organiza os componentes do descritor de cada característica de forma que valores semelhantes fiquem próximos entre si. Ao ser realizada uma busca, primeiramente são verificadas as distâncias referentes a determinadas dimensões do vetor (ARYA; MOUNT, 1993) e, a busca é interrompida quando aquela região de características se mostra distante daquela que está sendo correspondida. O BBF é eficiente em casos que características ambíguas são descartadas, como no caso daquelas geradas pelo plano de fundo da imagem (LOWE; 1999).

Uma razão para o algoritmo BBF funcionar bem é que somente são consideradas correspondências nas quais o vizinho mais próximo é menor do que 0,8 vezes a distância do segundo vizinho mais próximo e, portanto não é necessário resolver os casos mais difíceis, nos quais muitos vizinhos têm muitas distâncias similares (LOWE, 2004). O BBF consegue encontrar a características com menor distância euclidiana em aproximadamente 95% das tentativas com um ganho computacional de duas ordens de grandeza (Figura 17).

A Figura 17 apresenta um gráfico comparativo entre *K-d tree* e BBF, mostrando a quantidade de características versus a quantidade de comparações.

Figura 17 – Comparação entre *kd-tree* e BBF.

Fonte: Arya; Mount, 1993.

De acordo com a Figura 17 é possível observar que o algoritmo BBF é o mais eficaz na busca pelas correspondências das características, uma vez que o mesmo consegue extrair aproximadamente cem mil características com menos de cem comparações.

2.7. Imagens Correspondentes

Às vezes acontece de pares encontrados serem falsas correspondências, sendo necessária a respectiva eliminação. Com isso será reduzido o número de falsas correspondências e possibilitará o cálculo da matriz homográfica (que mapeia as posições dos pontos de uma imagem para as posições dos pontos

correspondentes, na outra imagem, de um par de imagens), responsável por juntar duas imagens sequenciais.

2.7.1. O RANSAC

O algoritmo RANSAC (*Random Sample Consensus*) possibilita encontrar as correspondências características geometricamente consistentes usadas para resolver a homografia (matriz fundamental) entre pares de imagens (FISCHLER; BOLLES, 1981). Esta atividade é realizada para a construção de mosaicos de imagens utilizando descritores invariantes como é o caso dos extraídos das imagens com a Transformada SIFT (BROWN; LOWE, 2007). O RANSAC é geralmente usado em visão computacional, como por exemplo, para resolver simultaneamente o problema de correspondência entre pontos de duas imagens e estimar a matriz fundamental M relacionada ao par de imagens.

O RANSAC é um algoritmo de estimação robusto proposto por Fischler e Bolles (1981) para extração dos *inliers*¹ do conjunto total dos dados de entrada (pontos chaves). É robusto no sentido de que ele pode estimar parâmetros com um alto grau de acerto mesmo quando um número significativo de *outliers*² estiver presente nos dados analisados.

Diferentemente de técnicas tradicionais que utilizam muitos dados para obter uma solução inicial, o RANSAC utiliza apenas um conjunto mínimo e suficiente de pontos necessários para uma primeira estimativa de parâmetros, aumentando o conjunto se necessário.

Uma amostra mínima depende do modelo a ser gerado. Geralmente é o número s de dados que será utilizado para inicializar um modelo. Por exemplo, no caso do modelo a ser estimado ser uma reta, uma amostra mínima consiste de apenas dois elementos.

A Figura 18 mostra um exemplo apresentado por Hartley e Zisserman (2003), que pode ser facilmente visualizado. Neste exemplo é estimada uma reta ajustada para um conjunto de pontos bidimensionais. Isso pode ser feito estimando uma

¹ *inliers*: pontos de dados que se ajustam com um determinado modelo desejado dentro de uma certa tolerância de erro.

² *outliers*: pontos de dados que não se ajustam ao modelo correspondente ao objeto desejado, estão fora de uma certa tolerância de erro.

transformação unidimensional $x'=ax+b$ entre os pontos correspondentes em duas linhas.

O problema ilustrado na Figura 18a é encontrar a reta que minimiza a soma das distâncias a partir de um conjunto de pontos de dados 2D, sujeito à condição que nenhum dos pontos válidos diverge dessa linha por mais de t unidades. Deseja-se então a reta ajustada aos dados e a classificação de dados em *inliers* (pontos válidos) e *outliers*. O limiar t é fixo de acordo com a medida de ruído (por exemplo, $t = 3\sigma$). Esse exemplo mostra que o ajuste dos pontos é severamente afetado pelos *outliers*.

Já com o uso do algoritmo RANSAC (Figura 18b), dois pontos são selecionados aleatoriamente, e esses pontos definem uma reta. Essa seleção aleatória é repetida um número de vezes e a reta com maior número de pontos que ficam dentro de um limiar de distância é considerada como ajuste robusto. Os pontos dentro da distância do limiar são os *inliers* (e constitui o conjunto dos consensos).

Figura 18 – Exemplo de ajuste de pontos.
(a) – pontos dados (b) – pontos entre um determinado limiar
Fonte: Hartley e Zisserman, 2003.

Na Figura 18 os pontos sólidos representam os *inliers*, e os pontos abertos representam os *outliers*. Na Figura 18b as linhas entre os pontos indicam a distância do limiar t , a linha passando pelos pontos a e b constitui a linha ajustada; o ponto c é um *outlier*.

A partir do conjunto mínimo necessário de pontos selecionados aleatoriamente, o RANSAC mede a precisão do modelo para todos os outros pontos, classificando-os em *inliers* ou *outliers*. Para tal, é definido um valor de limiar t , que é a distância máxima do modelo que um dado pode estar para ser considerado um *inlier*. O conjunto de *inliers* associado a um modelo é o conjunto consenso (*consensus set*) do modelo. Assim, com várias iterações, o algoritmo é capaz de

encontrar um modelo preciso para o problema em questão. O objetivo do algoritmo RANSAC é fazer o ajuste robusto de um modelo para um conjunto de dados S (a quantidade de elementos de S é a quantidade de características extraída pela Transformada SIFT e, varia de imagem para imagem) que contêm *outliers*. De acordo com Fischler e Bolles (1981), o algoritmo RANSAC é descrito a seguir, por meio dos seguintes passos:

1: Selecionar aleatoriamente uma amostra mínima de s ($s=4$ correspondências) pontos do conjunto S ($S=[]$). $S_i = 4$ correspondências. Determinado S_i , deve-se calcular a matriz homográfica H (que contém parâmetros de escala, rotação e translação) a partir deste subconjunto.

2: Incluir S_i e todas correspondências de S que possuem um erro de reprojeção de distância menor que t ($t=4$) no conjunto consenso. O conjunto S_i é o conjunto consenso (S_{consenso}) da amostra e define os *inliers* de S .

3: Se o tamanho de S_i (quantidade de *inliers*) for maior que um dado limiar T (quantidade mínima de *inliers*) ($T=0$ inicialmente), os parâmetros da matriz homográfica são reestimados utilizando todos os *inliers* identificados (S_{consenso}) e termina.

4: Se o tamanho de S_i for menor que T , um novo subconjunto é selecionado e os passos anteriores são repetidos.

5: Se após N tentativas, nenhum S_i com T ou mais elementos for selecionado, o S_i com maior quantidade de elementos é selecionado e a matriz homográfica é reestimada usando todos os elementos do subconjunto $S_i(S_{\text{consenso}})$.

Os parâmetros N e T são discutidos a seguir. Segundo Hartley e Zisserman (2003) o número de iterações N é escolhido grande o suficiente para assegurar uma probabilidade p (normalmente estabelecida para 0,99 (0,01)), que pelo menos uma das amostras aleatórias do conjunto de s pontos não inclua um *outlier*. Considere w ($w = 0,25$) a representação da probabilidade de algum dado selecionado ser um *inlier* e $\varepsilon = 1 - w$ a probabilidade de ser um *outlier* ($\varepsilon=0,75$). Pelo menos N iterações (cada uma contendo s pontos) são requeridas, onde $(1-w^s)^N=1-p$. Assim, o número de iterações necessárias é dado pela equação (33).

$$N = \frac{\log(1 - p)}{\log(1 - (1 - \varepsilon)^S)} \quad (33)$$

O tamanho do conjunto consenso é o que determina o quão bom um modelo é. O parâmetro T define o número necessário que deve ser alcançado para um modelo ser considerado interessante (conjunto consenso).

Caso o tamanho de S_i seja maior que T (inicialmente 0), o $S_{\text{consenso}} = S_i$ e $T = \text{tamanho de } S_i$, mas ainda o algoritmo não termina, uma vez que o valor de w vai ser atualizado de acordo com o valor atualizado de T . Então N vai ser recalculado utilizando este novo valor de w e o algoritmo só termina quando $N > 0,01$.

Durante a execução do RANSAC, quando for obtido mais de T *inliers*, o algoritmo finaliza, gerando novamente um modelo utilizando todos os T *inliers* obtidos, que será o modelo final encontrado pelo algoritmo (refinamento do modelo). É necessário definir um número máximo N de iterações que o algoritmo irá realizar, para evitar um número infinito de iterações. Se um determinado modelo não alcançar T *inliers*, o algoritmo gera um novo modelo a partir de uma nova seleção de amostra mínima dos dados de entrada. O algoritmo prossegue até conseguir obter um conjunto consenso, ou alcançar o número máximo de iterações retornando o conjunto com maior quantidade de *inliers*.

Resumindo, dado um modelo com parâmetros, é necessário estimá-los. Para tal, isso pode ser feito a partir de um número S de itens em um conjunto de dados conhecido. Os parâmetros que podem ser alterados de modo a determinar um bom resultado para o RANSAC são:

- t , que indica o limiar de distância que uma determinada entrada pode estar do modelo para ser considerado um *inlier*;
- T , que representa o número de *inliers* a ser alcançado para que um modelo parcial possa ser considerado um bom modelo, sendo que isso depende de cada caso, normalmente determinado empiricamente e;
- N , que representa o número máximo de iterações que devem ser realizadas.

O RANSAC é capaz de tratar com eficiência conjuntos de dados que contêm uma porcentagem significativa de erros. Este paradigma é particularmente aplicável à análise de cena por causa dos detectores de características locais, que

frequentemente cometem erros, que podem ser de dois tipos: erros de classificação e erros de medida. Erros de classificação ocorrem quando um detector de característica identifica uma porção de uma imagem incorretamente como uma ocorrência de uma característica. Erros de medida acontecem quando o detector de característica identifica a característica corretamente, mas comete erros de cálculo em um de seus parâmetros (por exemplo, seu local na imagem). Erros de medida geralmente seguem uma distribuição normal, e então a suposição de suavização é aplicável a eles. Porém, erros de classificação têm um efeito significativo maior que erros de medida (FISCHLER; BOLLES, 1981).

2.8. Considerações Finais

Este Capítulo apresentou as principais técnicas utilizadas neste trabalho para a construção automática de mosaicos. Foi apresentada a geometria epipolar e conceitos sobre retificações cilíndricas para minimizar as distorções das imagens de entrada. A Transformada SIFT foi utilizada para a extração de características, o algoritmo BBF para acelerar o processo de correspondência das imagens. O algoritmo RANSAC tem por objetivo eliminar as falsas correspondências e estimar a matriz homográfica, para união das imagens, sendo o algoritmo mais encontrado na literatura para aplicação de mosaico de imagens.

Neste trabalho de doutorado foram realizados testes, variando-se os parâmetros livres da Transformada SIFT e do algoritmo RANSAC sugerido por Lowe (2004) e Hess (2010), e verificou-se que a alteração desses parâmetros livres para valores superiores gerou maior custo computacional e aproximadamente os mesmos resultados do que com os valores sugeridos pelos autores. Alterando os valores desses parâmetros para valores inferiores, verificou-se que uma menor quantidade de características é extraída, dificultando a composição do mosaico. Devido a isso, foram utilizados neste trabalho os valores de parâmetros livres sugeridos por Lowe (2004) e Hess (2010).

3. Processamento Paralelo

3.1. Considerações Iniciais

Em diversas áreas científicas e de engenharia, há uma grande demanda de desempenho computacional. As aplicações científicas utilizadas nestas áreas ainda são as forças que impulsionam a computação paralela necessária para manipular uma grande quantidade de informações, acelerar a execução de códigos e resolver problemas que consomem tempo excessivo. Por este motivo, nas próximas seções serão apresentados conceitos sobre processamento paralelo e sobre os padrões de paralelização *OpenMP* (*Open Multi Processing*) e *MPI* (*Message-Passing Interface*).

3.2. Arquitetura de Computadores

Devido à existência de uma grande variedade de arquiteturas de computadores, inúmeras classificações foram propostas ao longo dos anos, porém a classificação mais aceita atualmente nos ambientes de hardware até hoje, é a conhecida classificação de Flynn (1972). Essa classificação leva em consideração o número de instruções executadas em paralelo e o conjunto de dados sob os quais as instruções são submetidas. Dessa forma, na Figura 19 são mostrados os tipos de computadores estabelecidos pela classificação de Flynn (1972).

		Dados	
		Simple	Múltiplo
Instrução	Simple	SISD Von Neuman	SIMD Vetores, matrizes
	Múltipla	MISD pipeline	MIMD multiprocessadores multicomputadores

SISD – Simple Instrução Simple Dados

SIMD – Simple Instrução Múltiplos Dados

MISD – Múltiplas Instruções Simple Dados

MIMD – Múltiplas Instruções Múltiplos Dados

Figura 19 – Classificação das arquiteturas de computadores.

Os computadores denominados como arquitetura MIMD possuem múltiplos núcleos de processamento, cada qual podendo executar instruções independente dos demais. Essas máquinas também podem ser classificadas como multiprocessadores (memória compartilhada) ou multicomputadores (memória distribuída). O primeiro grupo pode também ser denominado de ambiente fortemente acoplado enquanto que o segundo fracamente acoplado (Figura 20). As máquinas denominadas multiprocessadas possuem, como principal característica, o compartilhamento da memória entre todos os processadores (TANENBAUM, 2009).

Figura 20 – Classificação das arquiteturas MIMD.

Em arquiteturas de memória compartilhada, outra classificação que também merece destaque é a que define como os processadores acessam a memória. Dessa forma, eles podem ser classificados como de arquitetura UMA (*Uniform Memory Access*) ou NUMA (*Non-Uniform Memory Access*) (TANENBAUM, 2009).

- UMA – Nas máquinas que possuem esse tipo de arquitetura, o tempo de acesso aos dados localizados em qualquer posição da memória é o mesmo para todos os processadores. A forma de interconexão mais comum neste tipo de máquina é o barramento e a memória geralmente é implementada com um único módulo. Um dos padrões mais conhecidos é o SMP (*Symmetric Multiprocessor System*).
- NUMA – Nos computadores que possuem arquitetura NUMA, o tempo de acesso à memória depende da posição em que a mesma se encontra em relação ao processador. Isso acontece devido ao fato de que, nesse tipo de arquitetura, existe um limite quanto ao número de processadores por barramento de memória. Dessa forma, um processador terá acesso mais rápido a uma região de memória localizada no seu barramento do que a uma região localizada em um barramento diferente.

Nos processadores SMP existe os conceitos de *thread* física e de *thread* virtual. O termo “*thread* física” refere-se aos núcleos propriamente ditos presentes no computador enquanto que “*thread* virtual” refere-se aos núcleos simulados (virtuais), que o sistema operacional emula, duplicando a quantidade de núcleos físicos. Nesta emulação há a duplicação do conjunto de registradores, mas não há duplicação da unidade de ponto flutuante, ULA (unidade lógica e aritmética) e barramentos. Com isso, o desempenho das *threads* virtuais é inferior ao das *threads* físicas.

3.3. Processamento Paralelo

A computação de alto desempenho consiste em uma disciplina da computação científica que tem como objetivo aumentar a velocidade de processamento ou até mesmo viabilizar as simulações numéricas de problemas físicos complexos e com elevado grau de discretização.

Uma das técnicas da computação científica de alto desempenho é o processamento paralelo, que consiste na divisão de uma determinada tarefa em tarefas menores e na execução de cada uma dessas tarefas em diferentes processadores. O processamento paralelo pode ser realizado em ambientes de

memória distribuída (MPI) e em ambientes de memória compartilhada (*OpenMP*), de acordo com os trabalhos de Chandra, Dagum e Menon (2001) e Grama et. al (2003).

Processamento paralelo em ambientes de memória distribuída consiste em distribuir as tarefas para serem executadas em diferentes processadores que possuem seus próprios recursos de memória e são interconectados através de uma rede local. Nesse caso, o processamento paralelo é realizado por meio da troca de dados entre os processadores através da rede de interconexão.

Processamento paralelo em ambientes de memória compartilhada consiste na divisão das tarefas entre vários processadores que compartilham o mesmo recurso de memória global. Nesse caso não há necessidade de troca de dados entre os processadores, uma vez que todos têm acesso ao mesmo endereço de memória. Por outro lado, surge a necessidade da sincronização do acesso aos dados na memória compartilhada.

Nas últimas décadas são crescentes as pesquisas no campo da computação de alto desempenho. Diante do desenvolvimento tecnológico das arquiteturas paralelas e das limitações físicas que começam a surgir nos computadores pessoais, a computação de alto desempenho, que inclui a computação paralela, se apresenta como forte alternativa para aumentar o desempenho dos computadores e viabilizar uma grande variedade de aplicações nos meios científicos e industriais (CHANDRA, DAGUM e MENON 2001) e (GRAMA et. al 2003).

3.3.1. Processamento Paralelo em SMPs

SMP é uma máquina de arquitetura paralela MIMD com memória compartilhada, constituída por processadores que na maioria das vezes estão conectados à memória por meio de um barramento. Elas possuem arquitetura UMA, isto significa que todos os processadores têm acesso a todas as posições de memória e que o tempo de acesso é o mesmo para todas as posições. Entretanto, vale ressaltar que nem toda memória é compartilhada em um SMP. A velocidade de processamento dos processadores tem crescido de forma bastante rápida, porém as instruções são realizadas em dados cujo acesso na memória principal do computador não cresce na mesma proporção, de acordo com Tanenbaum (2009).

Para que esse acesso aos dados aconteça de forma mais rápida, cada processador possui uma memória privada, menor e mais rápida, denominada memória cache. Os dados são então trazidos da memória principal para a memória cache para que o processador possa acessá-los. A memória cache, entretanto, é específica de cada processador, somente sendo acessada por eles. A Figura 21 ilustra o modelo SMP. Os processadores multinúcleos atuais são SMP, como por exemplo, o modelo Intel i7.

Figura 21 – Compartilhamento de memória de um SMP.

O processamento paralelo em SMPs é feito através de programas com múltiplos *threads* (linhas de execução) ou múltiplos processos que executam simultaneamente, porém de forma independente, e cooperam para resolver um problema através do compartilhamento da memória. O desenvolvimento de programas paralelos com essa característica requer o uso de ferramentas que possibilitem a criação de *thread* e que implementem mecanismos de sincronização entre elas. As ferramentas de programação paralela mais comuns em SMPs atualmente são aquelas com *threading* explícito e as baseadas em diretivas de compilação (TANENBAUM, 2009).

Programação paralela com *threading* explícito – O programador cria explicitamente múltiplas *threads* dentro de um mesmo processo e divide, também explicitamente, o trabalho a ser realizado pelo programa entre essas *threads*. Assim como a criação, cabe ao programador gerenciar toda a sincronização das *threads*.

Programação paralela baseada em diretivas – O programador utiliza diretivas de compilação inseridas no código sequencial para informar ao compilador quais as regiões devem ser paralelizadas. Com base nestas diretivas o compilador gera um código paralelo. Dessa forma é necessário um compilador “especial”, ou seja, capaz de entender as diretivas e gerar o código *multithread*. Pode-se citar o padrão

OpenMP como uma interface que especifica um conjunto de diretivas de compilação, funções e variáveis de ambiente para a programação paralela em arquiteturas com memória compartilhada.

3.4. Padrão *OpenMP*

O padrão *OpenMP* é desenvolvido e mantido pelo grupo *OpenMP Architecture Review Board* (ARB), formado pelos maiores fabricantes de software e hardware do mundo, tais como SUN Microsystems, SGI, IBM, Intel, dentre outros, que, no final de 1997, reuniram esforços para criar um padrão de programação paralela para arquiteturas de memória compartilhada.

O *Open* significa que é padrão e está definido por uma especificação de domínio público e MP são as siglas de *Multi Processing*. O *OpenMP* consiste em uma API (*Application Programming Interface*) e um conjunto de diretivas que permite a criação de programas paralelos com compartilhamento de memória através da implementação automática e otimizada de um conjunto de *threads* (CHAPMAN; JOST; VAN DER PAS, 2008).

O *OpenMP* não é uma linguagem de programação, ele representa um padrão que define como os compiladores devem gerar códigos paralelos através da incorporação, nos programas sequenciais, de diretivas que indicam como o trabalho será dividido entre os processadores ou *cores*. Dessa forma, muitas aplicações podem tirar proveito desse padrão com pequenas modificações no código. Mesmo podendo ser usada em máquinas mononúcleo, o *OpenMP* foi planejado para satisfazer a implementação em larga escala das arquiteturas SMPs. O sucesso do *OpenMP* é propagado pelas arquiteturas *multicore* e *multithread* e pode ser atribuído a fatores como a robusta estrutura de programação paralela suportada e a facilidade do seu uso.

Diretivas: Consiste em uma linha de código com significado “especial” para o compilador. Por exemplo, nas linguagens C e C++ as diretivas *OpenMP* são identificadas pelo `#pragma_omp`, enquanto que na linguagem Fortran, as diretivas são identificadas pela sentinela `!$omp` (BARNEY, 2012).

No *OpenMP*, a paralelização é explicitamente realizada com múltiplas *threads* dentro de um mesmo processo. A criação de *threads* é uma forma de um processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas

simultaneamente (BARNEY,2012). Cada *thread* possui sua própria pilha de execução, porém compartilha o mesmo endereço de endereçamento de memória com as outras *threads* do mesmo processo (enquanto que cada processo possui seu próprio espaço de endereçamento).

O modelo de programação inicia com uma única *thread* que executa sozinha as instruções até encontrar uma região paralela (identificada pela diretiva). Ao encontrar essa região ela cria um grupo de *threads*, que juntas executam o código dentro dessa região. Quando as *threads* completam a execução do código na região paralela, elas sincronizam-se e somente a *thread* inicial segue na execução do código até que uma nova região paralela seja encontrada ou que o programador decida encerrar essa *thread* (BARNEY,2012). Esse modelo de programação é conhecido na literatura como *fork-join* e está ilustrado na Figura 22.

Figura 22 – Modelo de programação *OpenMP*.
Fonte: adaptado de Barney (2012).

Como citado anteriormente, o *OpenMP* é constituído de variáveis de ambiente, diretivas de compilação e bibliotecas de serviço, que juntas integram todas as funcionalidades disponíveis do padrão *OpenMP*. Na Figura 23 é ilustrada a sintaxe desses elementos para a linguagem C/C++ (CHANDRA; DAGUM; MENON, 2001).

Figura 23 – Elementos da interface *OpenMP*.

É possível citar algumas vantagens do padrão *OpenMP*, entre elas:

- Normalmente são feitas poucas alterações no código sequencial existente
- Possui uma robusta estrutura para suporte a programação paralela
- Fácil compreensão e uso das diretivas
- Suporte a paralelismo aninhado
- Possibilita o ajuste dinâmico do número de *threads*

De acordo com a apresentação anterior dos conceitos de arquitetura de computadores, processamento paralelo e do padrão *OpenMP*, este padrão de paralelismo foi utilizado com maior ênfase neste projeto de Doutorado por conta da possibilidade da aplicação ser portátil desde uma arquitetura mononúcleo até uma arquitetura de n núcleos, por ajuste dinâmico, sem intervenção do usuário final. Outra vantagem do padrão *OpenMP* é que os processadores atuais são multinúcleo SMP e o padrão *OpenMP* encaixa-se bem com este tipo de arquitetura pelo seu dinamismo.

Mais detalhes sobre o padrão *OpenMP* pode ser encontrados em Chandra, Dagum e Menon (2001), Grama et. al (2003), Chapmam, Jost e Van Der Pas (2008) e Barney (2012).

3.5. Padrão MPI

MPI (*Message-Passing Interface*) é uma especificação de uma biblioteca de interface de troca de mensagens que representa um paradigma de programação paralela no qual os dados são movidos de um espaço de endereçamento de um processo para o espaço de endereçamento de outro processo, através de operações de troca de mensagens (MPI, 2013). As principais vantagens do estabelecimento de um padrão de troca de mensagens são a portabilidade e a facilidade de utilização. Um ambiente de comunicação de memória distribuída ter a possibilidade de utilizar rotinas em mais alto nível ou abstrair a necessidade de conhecimento e controle das rotinas de passagem de mensagens é um benefício bastante razoável. Como a comunicação por troca de mensagens é padronizada por um fórum de especialistas (MPI, 2013), é correto assumir que será provido pelo MPI eficiência, escalabilidade e portabilidade.

Além do MPI possibilitar a implementação de programação paralela em memória distribuída em qualquer ambiente, também pode ser utilizado em plataformas de memória compartilhada como as arquiteturas SMPs e NUMA. E ainda, torna-se mais aplicável em plataformas híbridas, como cluster de máquinas NUMA. Todo o paralelismo em MPI é explícito, ou seja, de responsabilidade do programador, e implementado com a utilização dos construtores da biblioteca no código.

No padrão de programação paralela MPI uma coleção de processos executa o programa escrito em uma linguagem sequencial padrão contendo chamadas para uma biblioteca de funções que envia e recebe mensagens. A passagem de mensagens engloba tanto a transmissão da mensagem (via algum canal de comunicação), quanto a sincronização e ordenação das mensagens (MASSETTO, 2007). A especificação do emissor e do receptor no envio de uma mensagem é necessária para garantir a sincronização no envio e recebimento. O programador é responsável pela sincronização das tarefas. Dados transferidos entre processos necessitam da cooperação dos processos, de forma que, cada envio de mensagem deveria ter um recebimento de mensagem correspondente (EL SAIFI, 2006).

Em MPI existe o conceito de Grupos de Processos determinados por um comunicador. Todo o conjunto de processos participantes de uma aplicação faz parte de um comunicador global denominado *MPI_COMM_WORLD* e cada processo participante é identificado por um número inteiro maior ou igual a zero, denominado *rank*. Assim, se uma aplicação tiver n processos participantes do comunicador *MPI_COMM_WORLD*, seus respectivos *ranks* irão variar de 0 a $n-1$. O comunicador global é visível a todos os processos. Entretanto, é possível criar outros comunicadores, agrupando processos em subconjuntos (MPI, 2013).

Um ou mais processos podem ser executados em um mesmo *host*, denominado nodo ou nó. Caso dois ou mais processos em um mesmo nó necessitem trocar mensagens, implementações usam a comunicação através da memória, tornando-a mais eficiente.

No modelo de programação MPI, uma computação é composta por um ou mais processos que se comunicam chamando rotinas de biblioteca para enviar e receber mensagens a outros processos. Na maioria das implementações de MPI, um conjunto fixo de processos é criado para iniciar o programa, e um processo é criado por processador (MASSETTO, 2007). Porém, estes processos podem executar

programas diferentes. Consequentemente, o modelo de programação MPI, às vezes, são chamados de múltiplos programas múltiplos dados (MPMD) que se distingue de modelo SPMD no qual todo processador executa o mesmo programa.

O disparo de uma aplicação MPI em várias máquinas utiliza protocolos de *shell* remoto como o *Remote Shell* ou *Secure Shell*. O aplicativo *mpirun* é responsável por iniciar os processos a partir de uma lista de *hosts* pré-definida pelo usuário, ou por uma lista de *hosts* disponíveis, onde o próprio aplicativo realiza a seleção de onde os processos serão disparados (EL SAIFI, 2006).

Já que o número de processos em uma computação MPI normalmente é fixo, a próxima Seção vai focar no mecanismo de comunicação de dados entre os processos. Processos podem usar operações de comunicação ponto-a-ponto para enviar mensagem de um processo designado a outro. Estas operações podem ser usadas para implementar comunicações locais não estruturadas.

3.5.1. Operações em MPI

Embora MPI seja composta por um conjunto grande de funções, é possível resolver uma gama extensiva de problemas que usa apenas seis de suas funções. Essas seis funções iniciam e terminam uma computação, identificam processos, enviam e recebem mensagens. São elas a saber:

- MPI_INIT: Inicia uma computação MPI
- MPI_FINALIZE: Finaliza uma computação MPI
- MPI_COMM_SIZE: Determina número de processos
- MPI_COMM_RANK: Determina identificador do processo executável
- MPI_SEND: Envia mensagem
- MPI_RECEIVE: Recebe mensagem

O padrão MPI não especifica como uma computação paralela é iniciada, porém, um mecanismo típico poderia ser um argumento de linha de comando que indica o número de processos que seriam criados, por exemplo, *myprog -n 4*, sendo *myprog* o nome do executável. As maiorias das implementações de MPI são feitas para as linguagens C e Fortran (MPI, 2013).

3.5.2. Determinismo

Programação com passagem de mensagem não são determinísticos por padrão. A ordem de chegada de mensagens enviadas de dois processos, A e B, para um terceiro processo C, não é definido. Porém, MPI garante que duas mensagens enviadas de um processo A para outro processo B, chegará na mesma ordem enviada. É a responsabilidade do programador assegurar que uma computação seja determinística quando for o caso (EL SAIFI, 2006).

Em um modelo de programação de canais, determinismo é garantido com a definição de canais separados para comunicações diferentes, e garantido que cada tem um único escritor e um único leitor. Consequentemente, um processo C pode distinguir mensagens recebidas de A ou B quando eles chegam em canais separados. MPI não suporta canais diretamente, mas provê mecanismos semelhantes.

3.5.3. Operações Globais em MPI

Algoritmos paralelos pedem frequentemente operações de comunicação coordenadas que envolvem vários processos. Por exemplo, todos os processos podem precisar cooperar para transpor uma matriz distribuída ou somar um conjunto de números distribuídos, um por processo. Claramente, estas operações globais podem ser implementadas por um programador que usa funções enviar e receber. Para conveniência e para permitir implementações aperfeiçoadas, MPI provê também um conjunto de funções de comunicação coletivas especializadas que executam operações geralmente usadas neste tipo (EL SAIFI, 2006). Algumas destas funções são:

- Barrier: Sincroniza todos os processos
- Broadcast: Envia dados de um processo para todos os processos
- Gather: Junta dados de todos os processos para um processo
- Scatter: Espalha dados de um processo para todos os processos
- Reduction: somas, multiplicação, etc., sobre dados distribuídos.

3.6. Métricas de Desempenho

Ao resolver um problema em um sistema paralelo, o principal interesse é saber o ganho que se tem sobre a implementação sequencial deste problema. Uma das principais métricas que fornecem essa informação é o fator *Speedup* $S(n)$, que representa o ganho de velocidade de processamento de uma aplicação quando executada com n processadores. Quanto maior o *Speedup*, mais rápido se encontra o código paralelo. A equação (34) define essa métrica (GRAMA et. al., 2003).

$$S_n = \frac{T_s}{T_n} \quad \begin{array}{l} T_s = \text{tempo serial} \\ T_n = \text{tempo paralelo} \end{array} \quad (34)$$

Porém, por maior que seja a disponibilidade de processadores a serem utilizados, o fator *Speedup* é limitado por um valor máximo, decorrente da parcela serial do código, isso é ilustrado pela lei de Amdahl.

3.6.1. Lei de Amdahl

Como comentado anteriormente, nem todos os trechos do código são paralelizáveis. Dessa forma pode-se sempre identificar, dentro de um código (C), uma região que sempre será executada em serial (S) e outra que pode ser paralelizada (P) (Figura 24). O aumento do número de processadores apenas influenciará no tempo necessário para executar a região passível de paralelização.

Figura 24 – Trechos serial e paralelo de um código..

Observando a Figura 24, em que n é o número de processadores, é possível combinar estas expressões com a definição do fator *Speedup*, resultando na expressão conhecida como Lei de Amdahl, que representa o *Speedup* teórico

(máximo), de acordo com a equação (35), sendo S a fração serial do código, P a fração paralela do código e n a quantidade de processadores.

$$S_p = \frac{1}{S + \frac{P}{n}} \quad ; \text{ se } \lim_{n \rightarrow \infty} S_p = \frac{1}{S} \quad (35)$$

A idéia da Lei de Amdahl é que existe sempre um limite ao qual a capacidade de ganho pela paralelização estará sujeita. Isso se deve a fatores como entrada e saída, dependência entre os dados e outros fatores intrínsecos à aplicação e à técnica de programação paralela utilizada.

3.7. Considerações Finais

Este Capítulo descreveu de maneira sucinta o padrão *OpenMP* e MPI. A utilização do padrão *OpenMP* em arquiteturas SMP é vantajoso, pois permite a configuração automática da quantidade de *threads* disponível no computador, fazendo com que a utilização do software implementado seja simples para um usuário final. Softwares em *OpenMP* são portáveis para computadores com arquitetura SMP. A programação em *OpenMP* exige poucas alterações no código sequencial, facilitando a implementação. Estes fatores motivaram escolher o padrão *OpenMP* para ser utilizado neste projeto. Apesar dessas vantagens apresentadas neste Capítulo, ele não permite a paralelização de vários mosaicos.

O padrão MPI é mais indicado em aplicações que envolvem *cluster* de computadores, sendo vantajoso para paralelização de vários mosaicos.

4. Trabalhos Relacionados

4.1. Considerações Iniciais

Neste Capítulo, são encontrados os resumos dos artigos encontrados na literatura, para obter conhecimento de trabalhos relacionados ao apresentado nesta tese, envolvendo inclusive abordagens diferentes. Esta busca objetivou a aquisição de conhecimento das técnicas, algoritmos e metodologias para a extração de características das imagens, criação do descritor de características, correspondências das características encontradas, geração do mosaico de imagens e técnicas de paralelismo para acelerar a busca pelas características e geração do mosaico em si. Com base na investigação realizada, são apresentados neste Capítulo, trabalhos que utilizam a Transformada SIFT para realizar a extração de características, que estão mais próximos da metodologia apresentada neste trabalho de doutorado, mas são também apresentados trabalhos que utilizam outras técnicas, como PCA-SIFT e SURF.

A Seção 4.2 apresenta diversas aplicações da Transformada SIFT para auxiliar a geração de mosaicos, a Seção 4.3 apresenta outras técnicas para geração de mosaicos em comparação com a Transformada SIFT e a Seção 4.4 apresenta trabalhos sobre processamento paralelo aplicado junto a Transformada SIFT para construção de mosaicos de imagens.

4.2. Transformada SIFT Aplicada à Geração de Mosaicos

O trabalho de Zhan-Long e Bao-Long (2008) diz respeito à construção de um software para mosaico automático de imagens utilizando Transformada SIFT para extração de características da imagem seguido do algoritmo RANSAC para eliminação dos *outliers* e estimação da matriz homográfica. Os autores basearam-se nos conceitos de Lowe (2004).

O método foi aplicado e testado em pares de imagens que sofreram alterações de rotação, translação, escala e adição de ruído.

A Figura 25 mostra um exemplo de um par de imagens de entrada. Neste exemplo, a quantidade de características encontradas pela Transformada SIFT são 2081 e 2493 respectivamente. Após aplicação do RANSAC somente 570 pontos são *inliers* e, pela grande quantidade de *inliers*, só são mostrados 40 pontos correspondentes altamente estáveis.

Figura 25 – Correspondências das características após o RANSAC.
(a) – imagem da esquerda a ser unida (b) – imagem da direita a ser unida
Fonte: Zhan-Long e Bao-Long (2008).

Na Figura 26 é mostrado um exemplo da geração de mosaico.

Figura 26 – Composição de duas imagens em um mosaico.
Fonte: Zhan-Long e Bao-Long (2008).

Os autores afirmam que pelos testes realizados, o algoritmo proposto é robusto para alterações de translação, rotação, ruído e escala. Os autores mencionam que os algoritmos convencionais lidam bem com pares de imagens com fator de escala até 4, mas que este algoritmo por eles proposto, comporta-se bem com fator de escala até 10. Zhan-Long e Bao-Long (2008) implementaram o algoritmo em MATLAB 6.5 e todos experimentos foram realizados em um PC Pentium IV 2 GHz com 256Mb de RAM e S.O. Windows XP *Professional*.

No trabalho de Goltz e Shiguemori (2008) a Transformada SIFT é usada na busca por correspondências entre sequência de imagens diferentes que contenham partes de mesma cena. A busca é feita através de pontos-chave correspondentes, utilizando-se seus descritores. Neste artigo, a distância euclidiana é utilizada em três abordagens diferentes para avaliar a mais apropriada para navegação de veículos aéreos não tripulados (VANT). Os algoritmos de cada abordagem são: *DistEuclidConvencional* (1), que faz uma busca diretamente, sem tratar seus resultados; *DistEuclidRedundante* (2), que chama a função de busca duas vezes e, a segunda chamada é feita invertendo-se os parâmetros da função; somente as correspondências que ocorram em ambas são guardadas e *DistEuclidEsc* (3), que observa a continuidade de escala entre segmentos de reta traçados entre os pontos pertencentes e às correspondências geradas por esta função.

As imagens utilizadas neste artigo são imagens aéreas sequenciais, obtidas por um VANT. A Figura 27 apresenta um exemplo de pontos correspondentes.

Figura 27 – Exemplo de par de imagens sequenciais.
(a) – imagem da esquerda a ser unida (b) – imagem da direita a ser unida
Fonte: Goltz e Shiguemori (2008).

As três abordagens foram testadas para estimação de altitude, velocidade e direção de um VANT. A Tabela 1 mostra as porcentagens de acerto dos três

algoritmos levando-se em conta as dez primeiras correspondências geradas, para seis pares de imagens diferentes.

Tabela 1 – Resultados do trabalho de Goltz e Shiguemori (2008).

Algoritmo	Par 1	Par 2	Par 3	Par 4	Par 5	Par 6
1	90%	100%	100%	100%	100%	80%
2	100%	100%	100%	100%	100%	90%
3	100%	100%	100%	100%	100%	80%

Os autores concluem que a utilização do cálculo da distância euclidiana entre vetores que representam descritores de pontos gerados pela Transformada SIFT mostra-se bastante eficiente. A abordagem *DistEuclidRedundante* foi a mais precisa na avaliação realizada, no entanto, não foi a que apresentou o menor custo computacional. Os autores não citam a configuração do processador utilizado nos testes e nem resultados envolvendo tempo.

Liu, Chen e Guo (2008) desenvolveram um algoritmo para criar um mosaico com imagens aéreas. O projeto usa um VANT com uma câmera acoplada para criar um vídeo por onde passa. Cada frame de vídeo tem 320x240 pixels de dimensão e, a Figura 28 mostra um exemplo de sequência de *frames* obtidos.

Figura 28 – Sequência de *frames* do artigo de Liu, Chen e Guo (2008).

(a) – primeira imagem (b) – segunda imagem
(c) – terceira imagem (d) – quarta imagem

Para o vídeo obtido, a Transformada SIFT é usada para extrair pares de pontos característicos entre dois *frames* sequenciais. Na sequência, o RANSAC é utilizado para excluir possíveis falsas correspondências e para calcular a matriz

homográfica entre todos os *frames* sequenciais. Feito isso, um *frame* vai ser considerado 'imagem base' e fazendo cálculos em cascata, todas as outras matrizes homográficas são calculadas a partir da matriz homográfica da 'imagem base'. Com isso é possível transladar todo frame ao plano da imagem, em que o 'frame base' se localiza.

Quando o aeromodelo mantém uma rota constante, os *frames* não sofrem interferência de rotação, e a variação entre os *frames* sequenciais é pequena. Quando o aeromodelo retorna não há rotações notáveis entre *frames* sequenciais, contribuindo para uma boa junção das imagens. Na Figura 29 é possível observar o mosaico gerado com as imagens da Figura 28.

Figura 29 – Mosaico gerado no artigo de Liu, Chen e Guo (2008).

O tempo para gerar o mosaico da Figura 29 foi de três segundos. A proposta de Liu, Chen e Guo (2008) se mostra estável e de alta qualidade, uma vez que as imagens de mosaicos são suaves e completas e, com a boa obtenção das imagens feita pelo aeromodelo, os *frames* podem ser juntados sem qualquer quebra, de acordo com os autores. Os autores não citam a configuração do processador utilizado nos testes.

Cho, Bae e Durand (2009) desenvolveram um trabalho para reconstruir ambientes 3D a partir de imagens e vídeos capturados em determinados pontos fixos de uma cidade para permitir propagação automática de conhecimento urbano entre mapas e imagens digitais. Este trabalho foi patrocinado pelo Departamento da Força Aérea dos EUA.

O mosaico de imagens digitais e vídeos são georregistradas por meio da correspondência de algumas de suas características 2D com as suas contrapartes

3D em imagens *Ladar* aéreas. Uma vez que os planos das imagens estão alinhados com os mapas gerados pelo *Ladar*, o conhecimento urbano abstraído pode se propagar de um para o outro. Foram coletadas imagens digitais e vídeos em três locais diferentes do campus do Instituto de Tecnologia de Massachusetts (MIT), onde as câmeras foram programadas para operações de rotação e zoom, mas não foram programadas para translação. Os testes mostram que o conhecimento abstrato pode ser rapidamente projetado a partir do espaço 3D em mosaicos estáticos e subsequentemente em sequências de vídeos dinâmicos.

Inicialmente os autores estudaram técnicas para construir mosaicos 3D a partir de várias imagens 2D de entrada. A técnica para extrair características foi a Transformada SIFT. Cada imagem obtida tem dimensão de 3264x2448 pixels, como exemplificado na Figura 30.

Figura 30 – Exemplo de imagem usado no projeto de Cho, Bae e Durand (2009).

O RANSAC também foi usado para minimizar os falsos pares correspondentes. Nesse projeto, os pontos que são características SIFT, após aplicação do RANSAC, recebem um identificador comum entre cada par de imagens (Figura 31).

Figura 31 – Características SIFT no projeto de Cho, Bae e Durand (2009).

Após o RANSAC é construído o mosaico das imagens. Os autores utilizaram 21 imagens de teste para compor um exemplo de mosaico, como exemplificado na Figura 32.

Figura 32 – Mosaico gerado no projeto de Cho, Bae e Durand (2009).

Após esta tarefa, uma câmera de vídeo é colocada no mesmo lugar em que foram obtidas as imagens, para coletar um vídeo, dentro da área do mosaico formado anteriormente, para corresponder cada frame de vídeo com o mosaico de imagens, tão rápido quanto possível. Para isso, os autores desenvolveram um algoritmo cujo desempenho representa um compromisso entre precisão e velocidade. Nesse algoritmo, para cada frame de vídeo individual, com 1888×1062 pixels, as características SIFT são extraídas e correspondidas com suas homólogas no mosaico de imagens. Se um possível ponto correspondente é encontrado para uma característica de vídeo, um raio 3D é reconstruído a partir da imagem calibrada e associada com a coordenada de vídeo da característica 2D. Um RANSAC iterativo é utilizado para minimizar falsas correspondências entre raios e pares de coordenadas.

A Figura 33a mostra a localização angular instantânea de um frame de vídeo relativo ao mosaico, indicado pelo *view frustum* amarelo. A Figura 33b mostra o vídeo dinâmico alinhado com o panorama estático a partir da perspectiva de fundo da câmera.

Figura 33 – Casamento do mosaico com o vídeo, em Cho, Bae e Durand (2009).
(a) – localização angular instantânea de um frame (b) – vídeo dinâmico alinhado

A posição absoluta, orientação e escala da reconstrução, exemplificada na Figura 32 não pode ser determinada por técnicas convencionais de Visão Computacional. Então, para estabelecer estes parâmetros, as imagens e os vídeos devem ser inseridos em um mapa 3D com informações de posição global.

Na sequência foi feito o georreferenciamento do mosaico com os dados do Ladar, sendo necessário para isso a seleção manual de vários pares de *tiepoints* (representa a relação geográfica entre um ponto de uma cena e o mesmo ponto geográfico em outra cena) embora os demais componentes do *pipeline* de processamento tenha sido completamente automático. Após centenas de *frames* de vídeo terem sido correspondidos com os panoramas, as anotações de estruturas urbanas foram transferidas da imagem estática de fundo para a imagem dinâmica de primeiro plano. Os autores mencionam também que os usuários podem selecionar etiquetas estacionárias de interesse dentro dos *frames* de vídeo e recuperar suas geocoordenadas correspondentes. Os autores não citam a configuração do processador utilizado nos testes e nem resultados envolvendo tempo.

Bei e Haizhen (2009) propõem um novo algoritmo de mosaicos de imagens baseado na Transformada SIFT, combinando vários métodos de correspondência de características para obter alta precisão na construção dos mosaicos. Este novo

método aplica-se à imagens que possuem informações similares de estrutura e textura.

Após obtenção das características SIFT, um método, denominado “*Cosine Similarity*” foi adotado para corresponder os pontos característicos. Neste método, informações similares de textura e estrutura trazem algumas dificuldades para a correspondência das características, gerando resultados instáveis, podendo produzir falsas correspondências. Por conta disto, foi utilizado o conceito de grau de “*Definitude*” e distância euclidiana para obter pontos correspondentes com alta precisão. O valor de “*Definitude*” representa a discriminação entre a intensidade máxima de pontos correspondentes e a próxima intensidade abaixo do máximo. As imagens utilizadas são de 3648x2376 pixels, como exemplificado na Figura 34.

Figura 34 – Exemplo de imagem usado por Bei e Haizhen (2009).
(a) – imagem da esquerda a ser unida (b) – imagem da direita a ser unida

A quantidade de características SIFT detectadas nas Figura 34a e Figura 34b, respectivamente são 2116 e 3429. Com “*Similarity Cosine*” foram obtidos 1376 pontos correspondentes. Após a filtragem por distância euclidiana e “*Definitude*”, somente 21 pares de pontos correspondentes foram obtidos; e estes resultados podem ser observados na Tabela 2. A Figura 35 ilustra o mosaico resultante.

Tabela 2 – Filtragem dos pontos característicos por Bei e Haizhen (2009).

	SIFT	<i>Cosine Similarity</i>	<i>Definitude</i>	Distância Euclidiana	Taxa de Otimização
Figura 34a	2116	1376	156	21	98,5%
Figura 34b	3429

Figura 35 – Exemplo de mosaico gerado no trabalho de Bei e Haizhen (2009).

Os autores concluem que o emprego da Transformada SIFT juntamente com técnicas para minimizar a quantidade de pontos correspondentes torna o algoritmo mais robusto e eficiente. Os autores não citam a configuração do processador utilizado nos testes e nem resultados envolvendo tempo.

Li e Geng (2010) desenvolveram um software para a construção de mosaicos de imagens utilizando a Transformada SIFT para a extração de características e a técnica BBF para acelerar a busca por correspondência de pontos característicos. Na sequência, o algoritmo RANSAC foi usado para remover falsas correspondências entre pares de imagens. Imagens capturadas por uma câmera de vigilância foram usadas como entrada para testar o software. A Figura 36 mostra um exemplo de mosaico gerado pela metodologia proposta.

Figura 36 – Mosaico de imagens no trabalho de Li e Geng (2010).

Os autores adotaram 200 iterações no RANSAC. As imagens têm dimensão de 670x576 pixels e o tempo médio de processamento para detectar as características em cada par de imagens foi de 8,4 segundos. O tempo de processamento para unir duas imagens foi de 9s utilizando uma CPU Pentium 4 de 1.7 GHz com 512Mb de memória RAM.

Li e Geng (2010) concluem que o algoritmo usando Transformada SIFT é capaz de gerar bons mosaicos de imagens sequenciais, mesmo usando câmeras de vigilância, como mostra a Figura 37, mas que é custoso computacionalmente calcular as características SIFT para as várias imagens.

Figura 37 – Mosaico no trabalho de Li e Geng (2010)..

Wang et. al (2010a) desenvolveram um algoritmo para construção automática de mosaicos de imagens de fundo de olho baseado nas características SIFT e na BBF para corresponder os pontos característicos. A escolha da Transformada SIFT se deu porque ela é capaz de reconhecer características de baixo contraste e pela pouca distorção da geometria entre diferentes campos de visão deste tipo de imagem. Os *outliers* são descartados usando o algoritmo MLESAC (*Maximum Likelihood Estimation Sample Consensus*) ao invés do RANSAC. O MLESAC foi desenvolvido a partir de uma alteração no algoritmo RANSAC.

O registro de imagens é completado com a correção das imagens distorcidas utilizando uma transformação espacial e com isso as imagens registradas alinham-se com as imagens de referência, podendo desta forma ser construído o mosaico.

O registro de imagens foi analisado por alguns algoritmos de transformação do espaço da imagem para mosaicos automáticos; estes foram testados e comparados pelos autores. As transformações são as seguintes: afim, perspectiva, polinomial quadrática e polinomial cúbica. Para estas transformações, através de uma análise estatística, foram analisados o tempo de processamento e a precisão de coincidência de pontos característicos do registro de imagens. O erro de registro de imagem foi definido como a média da distância euclidiana entre cada ponto característico na imagem de referência. As imagens têm resolução de 1600x1200 pixels.

Estas várias transformações resultam em regiões de sobreposição, conforme Figura 38. A Figura 38a mostra que a transformação afim para corrigir a distorção entre duas imagens tem um erro residual grande. Usando a transformação em

perspectiva e polinomial quadrática é possível observar que os capilares estão bem alinhados, conforme a Figura 38b e Figura 38c. Embora a transformação polinomial cúbica (Figura 38d) tenha um erro menor, os capilares não estão localizados corretamente. De acordo com estas considerações, os autores optaram por adotar o método polinomial quadrático.

Figura 38 – Regiões de sobreposição entre imagens de fundo de olho.
 (a)– transformação afim (b) – transformação em perspectiva
 (c) – transformação polinomial quadrático (d) - transformação polinomial cúbica
 Fonte: Wang et. al (2010).

A Figura 39a mostra duas imagens de fundo de olho usadas como exemplo para a construção de um mosaico. Na Figura 39b as linhas verdes representam pares de pontos característicos correspondentes já filtrados após o algoritmo MLESAC; há 74 pares de pontos corretamente correspondidos. A Figura 39c mostra as duas imagens unidas após o cálculo da matriz de transformação. A Figura 39d mostra os resultados da fusão de duas imagens na região de sobreposição e a Figura 39e mostra o resultado de sobreposição após o registro das imagens. É possível observar que os capilares estão consistentemente ligados na região de sobreposição e que as imagens de fundo de olho podem ser precisamente juntadas automaticamente.

Figura 39 – Etapas do processo de construção de mosaico automático.
 (a) – exemplo de imagens (b) – pontos correspondentes (c) – imagens unidas (d) fusão de imagens na região de sobreposição (e) sobreposição das imagens após o registro
 Fonte: Wang et. al (2010).

Os autores testaram a metodologia com imagens de alta resolução (1600x1200 pixels) e baixa resolução (640 x 480 pixels) utilizando uma CPU Intel E8400 3.0 Ghz, 2Gb de RAM e ambiente de desenvolvimento Matlab R2009a. O custo computacional em segundos é apresentado na Tabela 3.

Tabela 3 – Desempenho dos resultados obtidos por WANG et. al (2010a).

Algoritmo	Extração de características	Correspondência dos pontos característicos	Filtragem dos pares de pontos	Registro e mosaico
Alta resolução (s)	2,537	0,459	0,431	2,897
Baixa resolução (s)	0,631	0,236	0,399	0,951

Os autores concluem que o algoritmo proposto tem bom desempenho, de acordo com os resultados apresentados. Quando se obtém pares de pontos correspondentes, o MLESAC é usado para rejeitar *outliers*, e na sequência o melhor modelo de registro de imagem é determinado e, finalmente, é construído um mosaico automático com imagens de fundo de olho.

Xing e Huang (2010) desenvolveram um método a partir da Transformada SIFT para construir mosaicos com imagens sequenciais de alta resolução obtidas por um VANT. No entanto, o propósito dos VANT's não pode ser alcançado com precisão, o que leva à diminuição da precisão e eficiência da correspondência automática. Existem algumas características da Transformada SIFT, tais como a

invariância de escala, rotação e luminosidade que são muito úteis para aumentar a precisão da correspondência das imagens obtidas por um VANT.

Foi utilizada a Transformada SIFT para detectar e corresponder características entre imagens e a transformação em perspectiva juntamente com o RANSAC foram usados para filtrar possíveis pontos falsos (*outliers*). A novidade aqui empregada é o uso do *algoritmo L-M* para otimizar os parâmetros, conseguindo construir o mosaico de imagens. A otimização é feita por um método não linear nos oito parâmetros da matriz de transformação, por conta dos erros na etapa de correspondência das características e a instabilidade da projeção em perspectiva, que é linear.

As imagens deste projeto têm 2000x3000 pixels e, na Figura 40 é possível observar o resultado de um mosaico gerado. O tempo gasto para gerar este mosaico exemplo foi de nove minutos, utilizando catorze imagens, de acordo com os autores. A configuração da máquina utilizada não foi descrita pelos autores. Os autores concluem que mesmo a Transformada SIFT demorando um tempo excessivo para extrair características é conveniente usá-la, pois obtém pares correspondentes estáveis, necessários para a criação de um mosaico preciso. Os autores não citam a configuração do processador utilizado nos testes.

Figura 40 – Mosaico gerado por Xing e Huang (2010).

No trabalho de Suzuki, Amano e Hashizume (2010) é descrita uma maneira de gerar mosaicos de imagens usando um pequeno e leve VANT para um sensoriamento remoto efetivo. Limites de carga é a fonte da maioria dos erros neste tipo de VANT para a geração de mosaicos precisos. Para superar essas desvantagens é integrado um GPS e sensores inerciais, além da própria câmera digital para a geração de um mosaico grande. A posição e altitude do VANT são estimados usando os sensores e correlação de imagem a partir das características SIFT entre imagens sobrepostas. Um algoritmo de otimização não linear é utilizado

para otimizar a posição e altitude do VANT. A metodologia apresenta baixo custo. O resumo da metodologia proposta pode ser descrita como uma maneira de estimar a posição e altitude dos dados, integrando operações do sensor de imagens para gerar um mosaico de imagens preciso, como segue:

- A área da imagem projetada no DEM (modelo de elevação digital) é calculada usando os dados de posição e atitude. A taxa de sobreposição é calculada usando a área da imagem projetada e avaliação de imagens sobrepostas combinadas.
- Os pontos chave nas imagens são identificados pela Transformada SIFT e correspondidos para formar imagens sobrepostas.
- Dados de posição e atitude são estimados utilizando correlação de imagens para minimizar o somatório do erro quadrado do custo da função.
- O mosaico é gerado usando estimação de posição e atitude baseada na imagem projetada no DEM.

Os autores concluem o artigo fazendo uma observação que o objetivo foi atingido, que é de construir mosaicos a partir de informações de posição e atitude a partir de uma UAV pequeno e leve com baixo custo, como pode ser observado na Figura 41.

Figura 41 – Mosaico no trabalho de Suzuki, Amano e Hashizume (2010) .

O trabalho de Hiraga e Silva (2011) propõe uma metodologia para construção de imagens panorâmicas utilizando a Transformada SIFT, algoritmo BBF e algoritmo RANSAC, mas ainda havendo uma preocupação de minimizar posições distorções nas junções, utilizando para isso retificação cilíndrica. Os autores não citam a

configuração do processador utilizado nos testes e nem resultados envolvendo tempo.

Em Silva et. al (2012) é apresentado um novo algoritmo para a construção de imagens panorâmicas (incluindo panoramas em 360 graus), que tem como principal característica evitar a distorção que ocorre pela junção de várias imagens sucessivas. Foram utilizados os algoritmos SIFT e RANSAC para encontrar áreas de sobreposição entre os pares de imagens, bem como um algoritmo de *Blend* para suavizar as junções. O algoritmo proposto não causa distorções durante a junção das imagens, assim, uma correção posterior não é necessária, o que contribui para um melhor desempenho. Os resultados dos experimentos usando softwares comerciais e também o algoritmo proposto foram comparados através de uma análise quantitativa, utilizando medidas numéricas calculadas sobre uma imagem panorâmica, gerada a partir de uma sequência de vinte imagens de uma região mapeada e georreferenciada pelo *Google Earth* e com isso ficou comprovado que a nova metodologia desenvolvida apresentou menores distorções do que os outros softwares comerciais testados, comparados com o *Google Earth*. Os autores não citam a configuração do processador utilizado nos testes e nem resultados envolvendo tempo.

A Tabela 4 apresenta um resumo das metodologias utilizadas por cada um dos trabalhos descritos nesta Seção.

Tabela 4 – Resumo das metodologias utilizando a Transformada SIFT.

Autor(es)	Metodologia	Plataforma	Configuração	Dimensão de imagem / frame	Tempo de processamento
Zhan-Long e Bao-Long (2008)	SIFT+ RANSAC	Matlab	Pentium IV 2Ghz, 256Mb de RAM	Não cita	Não cita
Goltz e Shiguemori (2008)	SIFT+ RANSAC	Não cita	Não cita	Não cita	Não cita
Liu, Chen e Guo (2008)	SIFT+ RANSAC	Não cita	Não cita	320 x 240	3s (04 imagens)
Cho, Bae e Durand (2009)	SIFT+RANSAC	Não cita	Core 2 Duo 2,2 Ghz, 4Gb de RAM	3264 x 2448 (imagem) 1888 x 1062 (frame)	10s por frame
Bei e Haizhen (2009)	SIFT + RANSAC + modificação na correspondência	Não cita	Não cita	3648 x 2376	Não cita
Li e Geng (2010)	SIFT+RANSAC	Não Cita	Pentium IV 1,7Ghz, 512 Mb de RAM	670 x 576	9s (par de imagens)
Wang et. al.(2010)	SIFT+MLESC	Matlab	Pentium E8400 3Ghz, 2Gb de RAM	1600 x 1200 640 x 480	2,9s 1s
Xing e Huang (2010)	SIFT+RANSAC+ otimização de parâmetros	Não Cita	Não Cita	2000 x 3000	9min (14 imagens)
Hiraga e Silva (2011)	SIFT+RANSAC	Não Cita	Não Cita	2144 x 1424	Não Cita
Silva et. al. (2012)	SIFT+RANSAC modificado	C	Não Cita	Não Cita	Não Cita

4.3. Comparação da Transformada SIFT com Outras Técnicas Aplicadas à Geração de Mosaicos

No trabalho de Mikolajczyk e Schmid (2005) foi realizado um levantamento sobre os seguintes detectores de pontos de interesse e descritores: *shape context*, *steerable filters*, *PCA-SIFT*, *differential in variants*, *spin images*, *SIFT*, *complex filters*, *moment in variants* e *cross-correlation*. Neste trabalho foi constatada que a Transformada SIFT apresenta um desempenho superior em relação aos demais algoritmos mencionados em termos de invariância.

Os autores do SURF (*Speeded Up Robust Features*), Bay et al. (2008b), afirmam que o seu algoritmo é superior ao SIFT em termos de eficiência de tempo de processamento e que o mesmo apresenta um bom desempenho no processo de reconhecimento de objetos, considerando a qualidade dos pontos de características.

O trabalho de Juan e Gwun (2009) apresenta uma comparação entre três métodos de detecção de características: Transformada SIFT, *Principal Component Analysis* (PCA)-SIFT e SURF. Os autores utilizaram KNN (*K-Nearest Neighbor*) para determinar as correspondências e o RANSAC para eliminar os *outliers* nas correspondências. Os três métodos de detecção foram comparados para: mudanças de escala, rotação, borrimento, mudanças de iluminação e transformações afim.

Os autores utilizaram a base de imagens “Graffiti” para testar as três metodologias. Esta base contém imagens com dimensões de 320x240 pixels. Os autores afirmam que o tempo de processamento é um resultado relativo, pois só mostra a tendência das três abordagens em relação ao custo computacional. Há fatores que influenciaram nos resultados, tais como, tamanho e qualidade da imagem, tipo de imagem e parâmetros do algoritmo. O tempo de processamento é determinado para o processamento completo, que inclui detecção de características e detecção das correspondências. A Tabela 5 mostra que SURF é o mais rápido, a Transformada SIFT é a mais lenta, porém é a que detecta mais correspondências.

Tabela 5 – Tempos de processamento em Juan e Gwun (2009).

Itens	SIFT	PCA-SIFT	SURF
Total de correspondências	271	18	186
Tempo total (ms)	2,15378e+007	2,13969e+007	3362,86
Tempo de 10 correspondências (ms)	2,14806e+007	2,09696e+007	3304,97

O segundo experimento refere-se a performance de invariância à escala. De acordo com os autores quando a mudança de escala é maior, SIFT ou SURF são melhores que PCA-SIFT, sendo que os resultados mostram que PCA-SIFT não é estável como SIFT e SURF para a invariância de escala, além do que PCA-SIFT detecta poucas correspondências.

O terceiro experimento mostra a influência da rotação nos três métodos. As imagens foram rotacionadas em passos de 5 em 5 graus ou de 10 em 10 graus. De acordo com os resultados, a Transformada SIFT é a técnica mais estável, SURF apresentou o pior resultado, com as menores quantidades de correspondências.

O experimento quatro avalia o desempenho dos algoritmos para imagem com borrimento Gaussiano, com raio de mudança de borrimento de 0,5 a 9,0. Os autores afirmam que SURF e PCA-SIFT apresentam bom desempenho quando o raio é grande. SURF detecta poucas características e PCA-SIFT detecta mais

correspondências, mas poucas corretas. A Transformada SIFT apresenta o melhor desempenho, conforme a Figura 42.

Figura 42 – Comparação do Borramento: (a) SIFT, (b) SURF, (c) PCA-SIFT.
Fonte: Juan e Gwun (2009).

O quinto experimento avalia os efeitos das mudanças de iluminação para os três métodos, com a iluminação sendo reduzida gradativamente. Para este experimento, na média, SURF teve o melhor desempenho.

O último experimento trata sobre a estabilidade das transformações afim, que é importante na construção de panoramas. Segundo os autores, a Transformada SIFT e SURF tiveram bons resultados quando a mudança no ponto de vista é pequena; na média PCA-SIFT teve o melhor desempenho.

A Tabela 6 mostra um resumo dos resultados dos experimentos para cada algoritmo, evidenciando que não há um melhor método para as operações de testes realizados. A Transformada SIFT mostra sua estabilidade em todos os experimentos, exceto para o tempo, sendo a técnica mais lenta; não apresentou bons resultados para mudança de iluminação. SURF tem um desempenho parecido com SIFT, mas não é estável para rotação e mudanças de iluminação. PCA-SIFT necessita melhorar seu desempenho para borramento e mudança de escala.

O que se pode concluir é que cada metodologia possui sua particularidade e a escolha de uma delas, depende do objetivo do projeto em questão.

Tabela 6 – Resumo dos resultados obtidos em Juan e Gwun (2009).

Método	Tempo	Escala	Rotação	Borramento	Iluminação	Transformação afim
SIFT	Ruim	Melhor	Melhor	Melhor	Ruim	Bom
PCA-SIFT	Bom	Ruim	Bom	Ruim	Bom	Bom
SURF	Melhor	Bom	Ruim	Bom	Melhor	Bom

No trabalho de Hong et.al. (2009) os autores propõem um algoritmo para construção de mosaicos de imagens baseado na extração de características por SURF (*Speeded Up Robust Feature*). Posteriormente é aplicado o RANSAC para

eliminação dos *outliers* e obtenção da matriz de transformação entre os pares de imagens. Finalmente as imagens são unidas por um algoritmo *multi-band blending*.

A extração de características pelo SURF é baseada na matriz Hessiana e o espaço escala é construído usando “*box filters*” ao invés de filtros gaussianos usados na Transformada SIFT. O espaço escala é analisado alterando o tamanho do filtro ao invés de reduzir iterativamente o tamanho da imagem, reduzindo desta forma o tempo de cálculo. O descritor SURF pode ser dividido em duas tarefas distintas. A primeira etapa consiste em definir uma orientação em uma região circular em volta do ponto de interesse, usando a Wavelet de Haar. A segunda etapa consiste de construir uma região quadrada alinhada com a orientação selecionada e extrair o descritor SURF a partir dela.

A correspondência das características SURF é diferente da Transformada SIFT, pois seleciona apenas parte das características e com isso há um processo de verificação de um subconjunto V1 de características com o conjunto total de características F1. Após este passo, é aplicado RANSAC para a exclusão dos *outliers*. O processo de fusão das imagens é ajustado aplicando o algoritmo “*multi-band blending*”; a ideia é misturar as baixas frequências em uma grande faixa espacial e as altas frequências em uma pequena faixa espacial. Isto é feito em múltiplas bandas de frequência usando uma pirâmide laplaciana.

Os testes foram realizados em um microcomputador Intel dual core de 2.83 GHz com 2Gb de RAM. Foram utilizados dois conjuntos de imagens com dimensões de 320x240 pixels. Os autores compararam a metodologia por eles desenvolvida (SURF), com a Transformada SIFT. Usando SIFT, gastou-se 1063 ms para juntar um par de imagens e usando SURF, 507 ms. A SIFT tem um custo computacional mais elevado do que SURF, mas detecta correspondências com mais eficácia.

No trabalho proposto por Azad et al. (2009) é feita a combinação do detector de cantos de Harris (HARRIS; STEPHENS, 1988) com a Transformada SIFT (LOWE, 2004), que determina descritores com alta repetibilidade e boas propriedades de correspondência com um tempo de processamento menor em relação ao da execução da Transformada SIFT original. Segundo os autores, a SIFT consome um tempo de 500 a 600 milissegundos para calcular os descritores SIFT de uma imagem com resolução de 640x480 pixels em uma CPU de 3 GHz, enquanto que usando a abordagem proposta por eles, a combinação do detector de

cantos de Harris com a Transformada SIFT, consome um tempo de processamento de apenas 20 milissegundos. Para esse ganho de tempo, os autores substituíram o estágio mais custoso computacionalmente da Transformada SIFT, a detecção de pontos chave estáveis no espaço escala, pelo detector de cantos de Harris.

Jiayuan; Yigang e Yun (2010) desenvolveram um algoritmo para reduzir ou eliminar a quantidade de falsas correspondências de características detectadas pela transformada SIFT, baseado no algoritmo RANSAC, através de um algoritmo de cancelamento de erros (geometria triangular) pelo cálculo de uma “correspondência base”. Para isto, primeiro tem-se que encontrar três pares de pontos correspondentes considerando a distância euclidiana mínima; segundo, determinam-se os três pares de pontos correspondentes como “correspondência base”; e verificar se estes pares satisfazem um critério de limiar, senão encontrar outros três pares de pontos correspondentes e repetir o processo. Finalmente, checam-se todos os outros pares correspondentes e possível eliminação dos falsos pares correspondentes. Os resultados mostraram que o algoritmo proposto tem baixo custo computacional quando herda a invariância à rotação da Transformada SIFT. Ao mesmo tempo, isto não exclui os pares correspondentes corretos com forte robustez, fazendo com que a precisão do mosaico de imagens seja melhorada.

É usada a menor distância euclidiana para calcular as características correspondentes SIFT, então os pontos correspondentes são obtidos pela euclidiana padrão, da menor para a maior, e então são extraídos n pares das menores distâncias euclidianas. Extraí-se um par da “correspondência base” a partir dos pontos característicos filtrados e usa-se o RANSAC para checar a “correspondência base” e usá-la para filtrar outros pontos característicos. Isto é capaz de eliminar completamente as falsas correspondências e reter as verdadeiras correspondências.

Os resultados apresentados pelos autores mostraram-se melhores do que usando somente RANSAC. Para um dado grupo de imagens, a SIFT detectou 91 pontos corretos. O RANSAC removeu 15 pontos corretos, com 4 pontos errados restantes, restando no total 80 dos 91 pontos. Com o uso da técnica proposta pelos autores, houve remoção de 5 pontos corretos e nenhum erro restante, restando no total 86 dos 91 pontos. Os resultados comparativos utilizando RANSAC, Distribuição Espacial e o método proposto pelos autores podem ser verificados na Tabela 7.

Tabela 7 – Resultados obtidos por Jiayuan; Yigang e Yun (2010),

Experimentos		<i>Frog blood</i>	<i>Lung</i>	<i>Sunflower cells 1</i>	<i>Sunflowers cells 2</i>	<i>Kidney cells</i>	<i>Spleen cells</i>
SIFT	Pontos corretos	91	129	103	34	1033	2001
	Total de pontos	204	145	177	93	1118	2134
RANSAC	Pontos restantes	80	118	89	27	930	1811
	Pontos corretos removidos	15	12	18	8	103	190
	Erros restantes	4	1	4	1	0	0
Distribuição Espacial	Pontos restantes	204	145	177	93	1118	2134
	Pontos corretos removidos	-	-	-	-	-	-
	Erros restantes	-	-	-	-	-	-
Método Proposto	Pontos restantes	86	128	94	34	1023	1986
	Pontos corretos removidos	5	1	9	0	10	15
	Erros restantes	0	0	0	0	0	0

O método RANSAC tem alto custo computacional quando duas imagens contêm uma grande quantidade de estruturas similares, sendo provável que remova alguns pontos correspondentes corretos e que permaneçam alguns falsos pontos correspondentes, afetando o mosaico em si. O método de Descritor de Distribuição Espacial não tem invariância à rotação, e só pode ser aplicado a um número muito pequeno de pontos característicos, não tendo valor prático. Se for utilizado, em primeiro lugar é necessário usar outros métodos (por exemplo, iterativo) para capturar a parte semelhante das duas imagens e, em seguida usar o algoritmo para remover o erro. O algoritmo proposto pelos autores melhora a precisão da correspondência com baixo custo computacional, fazendo os autores concluírem que o algoritmo proposto é melhor do que os testados e, que os resultados do algoritmo proposto por eles têm alto valor prático.

O artigo de Yang et. al. (2011) compara a transformada SIFT com SURF e o detector de cantos de Harris para o requisito de velocidade e precisão exigido na área de mosaicos de imagens. Os autores afirmam que o algoritmo de Harris teve o pior desempenho para extrair pontos característicos e para achar a correspondência entre esses pontos. A Transformada SIFT apresentou vantagens para transformação

afim, rotação e invariância à escala, embora tenha alto custo computacional. A SURF é a mais rápida, mas não apresenta bom desempenho para mudanças de cor e iluminação.

O Trabalho de Khan et. al (2011) compara as Transformada SIFT e SURF para classificação de cenas em ambientes internos e externos. Os autores modificaram a dimensão do descritor SIFT de 128 para 32, 64 e 96, gerando resultados tão rápidos quanto SURF e os autores concluem que SURF gera bons resultados, exceto para escala, borramento e invariância à ponto de vista.

Bai et. al (2012) comparam os algoritmos de Harris, Transformada SIFT e SURF para detecção de características e correspondência das mesmas na área de exploração de superfícies planetárias. Os autores afirmam que o algoritmo de Harris vem sendo bem empregado na área de engenharia, mas com resultados nada surpreendentes. A Transformada SIFT é indicada para imagens com muitas mudanças de escala, possui alta robustez e precisão na detecção das características e correspondência. Os autores afirmam que a Transformada SIFT pode ser utilizada muito bem para a necessidade deles, mas que essa transformada é complicada de se entender e lenta. Bai et. al (2012) dizem que a SURF é a versão rápida da Transformada SIFT, com resultados similares.

O trabalho de Ali (2012) compara a Transformada SIFT com SURF para área de construção de imagens panorâmicas. As imagens deste artigo variam em alguns fatores, como iluminação, ponto de vista, rotação, escala, borramento e ruído e estes fatores afetam na detecção das características e na união das imagens. Os autores afirmam que a Transformada SIFT é indicada para aplicações que exigem alta precisão e fidelidade da união. O SURF é um meio termo entre precisão e custo computacional.

A Tabela 8 apresenta um resumo das metodologias utilizadas por cada um dos trabalhos descritos nesta Seção.

Tabela 8 – Resumo de outras metodologias aplicadas à geração de mosaicos.

Autor(es)	Metodologia	Tempo de Processamento	Dimensão das Imagens	Conclusões
Mikolajczyk e Schmid (2005)	Comparação da SIFT com diversas metodologias	Não cita	Não cita	SIFT tem desempenho superior quanto à invariância
Bay et al. (2008)	SURF, SIFT	SURF: 1036 ms SIFT: 354 ms (construir 1 descritor)	320 x 240	SURF é superior em tempo de processamento, comparado à SIFT
Juan e Gwun (2009)	SIFT, PCA-SIFT e SURF	SIFT: 2,15e+007ms SURF: 3363ms PCA-SIFT: 2,13e+007 ms (correspondência de um par de imagens)	320 x 240	Cada metodologia possui a sua particularidade, sendo mais indicada para cada tipo de necessidade
Hong et al. (2009)	SURF, SIFT	SIFT: 1063ms SURF: 507 ms (unir 1par de imagens)	320 x 240	SURF é superior em tempo de processamento, mas a SIFT detecta características com mais eficácia
Azad et al. (2009)	SIFT + detector de cantos de Harris	SIFT: 600 ms SIFT modificada: 20 ms (cálculo de 1 descritor)	640 x 480	Com esta modificação, o custo computacional da Transformada SIFT é reduzido
Jiayuan et al. (2010)	SIFT + RANSAC modificado	Não cita	Não cita	Com esta modificação a eliminação de <i>outliers</i> se torna mais eficaz
Suzuki et. al.(2010)	SIFT+ coordenadas GPS	Não cita	Não cita	Possibilitou a construção de mosaicos à partir de informações de posição e altitude
Yang et. al. (2011)	Comparação da SIFT, SURF e Harris	SIFT: 768ms SURF: 198ms Harris: 1411ms	Não cita	A SIFT apresenta vantagens, apesar do alto custo computacional
Khan et. al (2011)	Comparação da SIFT e SURF	SIFT: 59s SURF: 80s	<i>David Nister Dataset</i>	SURF não é tão bom quanto SIFT
Bai et. al (2012)	Comparação da SIFT, SURF e Harris	SIFT: 54s SURF: 18s Harris: 10s	1392x1040	SIFT é indicada para aplicações que exigem muitos detalhes da imagem
Ali (2012)	Comparação da SIFT e SURF	SIFT: 9s SURF: 4,5s	576 x 576	SIFT é melhor quando é exigido precisão e fidelidade

4.4. Paralelismo Aplicado à Transformada SIFT e a Geração de Mosaicos

O trabalho de Feng et. al.(2008) propõe paralelizar a Transformada SIFT por SMP (*Symmetric Multiprocessing*) e por CMP (*Chip Multiprocessor*), conhecido também por processadores com vários núcleos. Os resultados mostraram que a Transformada SIFT teve 11x de *Speedup* em um sistema SMP de 16 núcleos.

Utilizando um simulador de 64 núcleos CMP, foi possível conseguir um *Speedup* de 52x.

O desequilíbrio de carga limita significativamente a escalabilidade e a Transformada SIFT sofre de intensa exigência de largura de banda de memória de ruptura no sistema SMP de 16 núcleos. Já no simulador CMP de 64 núcleos a exigência sobre a memória não é alta devido ao último nível de cache compartilhado (LLC - *last-level cache*), que acomoda grande compartilhamento de leitura/escrita na Transformada SIFT.

Os autores escolheram paralelizar a Transformada SIFT de maneira menos global ("*fine-grain thread level*"), pois reduz o conjunto de trabalho e provê capacidade de processamento *on-line*, explorando cada frame individual, no caso de processamento de vídeo. Já a estratégia global ("*coarse-grain*") processa vários *frames* em paralelo, porém a excessiva carga de trabalho faz a memória *cache* ficar sobrecarregada, limitando a escalabilidade.

Para a estratégia "*fine-grain*", as etapas mais custosas da Transformada SIFT foram selecionadas para a paralelização. As quatro etapas mais custosas são: construção do espaço escala gaussiano, detecção e localização dos pontos chaves, determinação da orientação e construção do descritor. Como há várias linhas e pixels paralelizáveis durante o processamento das imagens, é simples para cada *thread* processar diferentes blocos de dados para as três primeiras etapas da SIFT e para a última etapa, conforme os cálculos variam, os principais são atribuídos dinamicamente, em paralelo.

O desempenho da Transformada SIFT é melhorado através de instruções SIMD (*Single Instruction Multiple Data*) através do conjunto de instruções SSE-2 presente em várias arquiteturas de processadores e, com isso há 1,9x de *Speedup* utilizando um processador *Intel Core2Quad*.

Também é otimizado o desempenho da cache/memória para a Transformada SIFT, uma vez que a cache é compartilhada por todos os núcleos coletivamente. Na Transformada SIFT, os autores aplicam fissão de laço e laço intercambiável para melhorar a localidade da cache. Para eliminar operações de cópia de memória redundante, mudanças na estrutura dos dados e controle de fluxo são realizadas e, com isso se economiza o uso da banda da memória. Outra otimização na cache é mudar cuidadosamente o arranjo dos dados a fim de evitar falsos compartilhamentos em programas paralelos.

O mecanismo de afinidade de *thread* pode anexar uma *thread* em um núcleo específico no SMP ou CMP. Quando um grupo de *threads* tem compartilhamento pesado de dados, é usada a afinidade de *thread* para ligá-las aos núcleos ao mesmo chip, utilizando compartilhamento LLC para comunicação rápida e sincronização. Na Transformada SIFT, o mecanismo de afinidade tenta escalonar as *threads* em um mesmo socket e até mesmo sobre o mesmo chip, trazendo mais benefícios, pois a Transformada SIFT tem muitos dados compartilhados nas *threads* e, isto traz um melhoramento de 10% comparado com o escalonamento tradicional dos sistemas operacionais.

Os resultados sobre o sistema real SMP foram gerados utilizando servidor QuadCore (HP ProLiant DL580 G5) que contém 4 processadores *Intel Core2Quad* (processador que contém quatro núcleos internamente rodando a 1.6 GHz com 64Kb de L1 e 16MB de L2), totalizando 16 núcleos. A implementação com *OpenMP* foi feita pelo compilador Intel C/C++ versão 9.1 em Linux.

Foram utilizadas duas bases de dados para os testes. Uma utilizando imagens MPEG-2 (720x576 pixels contendo 700 pontos chaves) e outra utilizando imagens de alta definição (1920x1080 pixels contendo 1038 pontos chave).

A Figura 43 apresenta os resultados de desempenho da Transformada SIFT por SMP. O rótulo “original” representa a Transformada SIFT sem SIMD e sem otimização da cache. Os *Speedup* podem ser verificados na Figura 43. Para imagens MPEG-2 o sistema rodou a 30 fps e para as imagens de alta definição em 10 fps. Pela Figura 43, é possível verificar que através de SIMD o *Speedup* foi de 1,85x para imagens de alta definição e de 1,72x para imagens MPEG-2. Também é possível observar que a otimização da cache melhorou de 1,2x a 1,25x o desempenho. Totalizando todas otimizações é possível verificar que houve de 1,8x a 2,2x de *Speedup* na implementação SIFT desenvolvida pelos autores.

Mais detalhes sobre análise de desempenho do sistema SMP do trabalho de Feng. et. al. (2008) podem ser encontrados no próprio artigo dos autores.

Já nos testes do sistema simulado CMP de 64 núcleos é utilizada LLC compartilhada e alta largura de banda de memória. Possui cache L1 privada e cache L2 compartilhada. Os parâmetros de configuração adotada podem ser vistos na Figura 44.

Figura 43 – Resultados de desempenho da SIFT por SMP.

Fonte: Feng. et. al. (2008)

Processor Parameters	
# Cores	1-64
Frequency	2GHz
Core width	2
Branch predictor	gshare, 2k entries
Memory Hierarchy Parameters	
Private (L1) cache	32KB, 8-way, 64B line
Shared L2 cache	16MB, 16 banks, 16-way (inclusion)
Coherence Protocol	MSI
Interconnect network	Bi-directional ring (40s tops)
Contentionless Memory Latencies	
L1 hit	3 cycles
L2 hit	18-58 cycles
Main memory access	298-338 cycles

Figura 44 – Parâmetros de configuração no sistema CMP.

Fonte: Feng. et. al. (2008)

Os resultados mostram que as imagens de alta definição tiveram melhor escalabilidade (52x) do que as imagens MPEG-2 (39x), fazendo com que as imagens de alta definição possam rodar a 33fps (tempo real) com o sistema de 64 *threads* CMP. Mais detalhes sobre análise de desempenho do sistema CMP do trabalho de Feng. et. al. (2008) podem ser encontrados no próprio artigo dos autores.

Os autores concluem afirmando que as técnicas por eles testadas para otimizar a Transformada SIFT conseguem atingir até 85% de ganho de desempenho.

Liu, Chen e Li (2009) apresentam uma metodologia paralelizável da transformada SIFT através de GPU (*Graphic Processing Unit*) aplicada em mosaicos de imagens coloridas ou mosaicos de vídeos.

Inicialmente os *frames* dos vídeos são decodificados a fim de se escolher uma sequência de 10 *frames*. Após isso, as características SIFT são extraídas por GPU.

As imagens são processadas na GPU como texturas. As etapas custosas da Transformada SIFT são otimizadas ao aplicar processamento em GPU, com o filtro gaussiano sendo executado na vertical e horizontal separadamente e em paralelo para a construção do espaço escala. Também foram processados em GPU o algoritmo RANSAC e o cálculo da matriz homográfica, seguindo o mesmo esquema de Lowe(2004). Para conseguir algumas características a mais, dada uma quantidade pequena de características, a invariância das características é calculada separadamente pelos três canais RGB e somente um dos pontos chave é selecionado nos pontos extremos, que mostra o maior valor significativo.

O experimento realizado para a geração de mosaicos de imagens utilizou imagens de 400x300 pixels, um microcomputador da Dell modelo D830 com uma GPU NVIDIA GeForce 135M. A Tabela 9 apresenta os resultados comparativos entre o processamento em CPU e GPU.

Tabela 9 – Comparação entre CPU e GPU em Liu, Chen e Li (2009).

Imagem n°.	Qtde. de caract.		Tempo para extrair as caract. SIFT (s)		Tempo total (s)	
	GPU	CPU	GPU	CPU	GPU	CPU
1	625	494	0,317	0,512	2,985	4,796
2	582	524	0,215	0,431		
3	659	496	0,234	0,472		
4	699	621	0,238	0,458		
5	932	572	0,251	0,493		

Os autores também realizaram um experimento para a geração de mosaicos baseados em vídeo; foram selecionados 4 *frames*. Os resultados mostraram que o processamento por CPU levou 51,5s e por GPU, 24,6s.

O trabalho de Warn et. al (2009) trata da aceleração da Transformada SIFT (que é uma das etapas mais custosas do mosaicamento de imagens) por paralelização utilizando *OpenMP* (*Open Multi-Processing*) em uma arquitetura SMP e também por CUDA (*Compute Unified Device Architecture*), uma tecnologia GPGPU

(*General Purpose Computing on Graphics Processing Unit*), determinando a eficácia destas duas técnicas, aplicadas ao mesmo código base da Transformada SIFT tradicional.

Segundos os autores, por testes realizados, as partes da Transformada SIFT que se aplica paralelização são:

- Orientação dos pontos chave
- Construção do espaço escala
- Geração do descritor

A vantagem de usar *OpenMP* é o seu custo benefício, segundo os autores, uma vez que foi possível aplicar *OpenMP* nas três etapas da Transformada SIFT que cabe paralelismo, utilizando menos de 20 novas linhas de código entre a SIFT tradicional e a Transformada SIFT paralelizada por *OpenMP*. O *OpenMP* transforma seções de códigos seriais em regiões em paralelo através de *threads*.

Acelerar uma aplicação com CUDA é uma tarefa mais complexa. A NVIDIA descreve a sua arquitetura como SIMT (*Single Instruction Multiple Thread*), contendo uma hierarquia complexa de memória.

A implementação *OpenMP* foi testada em um Intel Quad core Xeon com 2.66GHz, 8Mb de L3 compartilhada pelos 4 núcleos e com 256Kb de L2 para cada núcleo. A GPU usada para o CUDA é do modelo NVIDIA FX5800 com 4Gb de memória, testada em um Intel E6550 com 2.33GHz.

A Figura 45 mostra o tempo de processamento da implementação *OpenMP*, funcionando de uma a 8 *threads*. As linhas no gráfico representam as escalas de imagem em relação a original. A linha superior é a imagem original e as outras são as imagens reduzidas pela metade, um quarto e um oitavo em relação a original.

Figura 45 – Implementação utilizando *OpenMP*.
Fonte: Warn et. al (2009)

A Figura 46 mostra o ganho de desempenho obtido com *OpenMP*, baseado na Figura 45. Um ganho de desempenho ideal é mostrado e os resultados demonstraram que só foi obtido um ganho de desempenho igual a metade do ideal.

Figura 46 – Ganho de desempenho utilizando *OpenMP*.
Fonte: Warn et. al (2009)

Quanto aos resultados envolvendo CUDA, há uma comparação entre a execução feita somente utilizando a CPU descrita com a implementação feita em CUDA, utilizando a mesma configuração de máquina. A Tabela 10 apresenta os resultados; na coluna “tempo de processamento” é descrito quanto tempo se gasta para realizar a convolução com um *Kernel* de dimensão 15 em uma imagem de 8272x2848 pixels. A coluna “tempo de comunicação” mostra os resultados referentes a transferência da imagem de entrada para a placa e a cópia da imagem de saída da placa; isto só se aplica a GPU. O tempo total de processamento com a GPU foi de 168ms e com a CPU foi de 2180ms. A GPU é 13x mais rápido do que a CPU.

Tabela 10 – Comparação do processamento entre CPU e GPU.
Fonte: Warn et. al (2009)

	Tempo de Processamento	Tempo de Comunicação	Tempo total
GPU	10.4 ms	158 ms	168 ms
CPU	2180 ms	-	2180 ms

Os autores concluem que uma implementação em *OpenMP* é mais simples que em GPU, mas que o desempenho não é tão elevado quanto a implementação em GPU.

O trabalho de Jiang et. al. (2010) apresenta uma maneira de paralelizar a Transformada SIFT, denominada pelos autores de DDP-SIFT (*Distributed Dynamic Parallel SIFT*), utilizando *clusters*. A etapa de construção do espaço escala foi ajustada para aumentar a proporção de paralelismo e reduzir o tempo de comunicação por redução de dados alocados. A metodologia envolve os seguintes passos:

- **Introdução dos Dados Paralelos:** divide o espaço escala gaussiano por oitavas objetivando obter grandes blocos de imagens, que é de grande importância para grandes descritores de características. A Figura 47 mostra duas abordagens de blocos de dados.
- **Otimização da Alocação dos Dados:** Primeiramente segue a mesma ideia proposta no artigo de Lowe (2004) no que diz respeito a construção de oitavas. Então a pirâmide de oitavas é dividida em tarefas e cada uma alocada em cada processador, com isso os blocos de dados são construídos em paralelo.

Figura 47 – Divisão dos Dados

(a) por oitavas - (b) por divisão da imagem de entrada.

Fonte: Jiang (2010)

- **Algoritmo Paralelo Dinamicamente Distribuído:** As tarefas são dinamicamente alocadas no respectivo nó, conforme são solicitadas a partir de uma lista de tarefas geral.
- **Otimização para Passagem de Dados:** Às vezes o comprimento da linha do bloco de dados é muito pequeno e imagens são bloqueadas e, os blocos de imagens devem ser colocados em *buffer* no espaço de memória antes de serem enviados para os nós, gerando diminuição de desempenho comparado com o envio de dados por linha.
- **Otimização para Equilíbrio de Carga:** Diminuição do tamanho do bloco, dividindo as oitavas em $M \times N$ blocos.

O algoritmo DDP-SIFT foi desenvolvido em linguagem C com OpenCV e GSL e funciona em 32 nós, interligados por uma placa de rede gigabit no Sistema Operacional Linux. As imagens utilizadas tem dimensão de 1024x768 pixels.

Os resultados gerados pelos autores mostram que quando o número de processadores é menor do que 14 a eficiência é maior do que 0,5, mas diminui rapidamente com o aumento do número de processadores.

O trabalho de Wang et. al. (2010b) apresenta uma metodologia para acelerar o esquema tradicional de mosaicos paralelos, que não otimiza a passagem do processamento serial e ainda leva em conta que o registro e o “*blending*” das imagens são custosos, não havendo para estes casos uma estratégia paralela para acelerar o processamento. E mais, para grande quantidade de dados, há problemas

de Entrada/Saída (E/S) que afetam todo o desempenho. Para resolver estes problemas, os autores propuseram um algoritmo otimizado de mosaico de imagens com paralelização de E/S e estratégia paralela dinâmica agrupada baseada na mínima árvore de abrangência.

Na Figura 48 é possível observar o esquema tradicional de mosaicos paralelos, que é parcialmente similar ao esquema do mosaico de imagens serial. Por esta figura se observa os possíveis gargalos citados anteriormente.

Figura 48 – Esquema tradicional de mosaico paralelo.

Fonte: Wang et. al. (2010b)

Para resolver estes problemas de desempenho do algoritmo tradicional de mosaicos paralelos, os autores propõem a metodologia citada anteriormente, com distribuição dinâmica de tarefas para balanceamento de carga na parte de registro e também reamostragem e distribuição dos dados no sistema de arquivo paralelo da E/S do *cluster multithread* paralelos. Além disso, operações *blending* sequenciais evitam a imprecisão e com isso a eficiência do algoritmo é melhorada.

A ideia principal do algoritmo proposto pelos autores é que a parte de registro é realizada pelo nó mestre, fazendo um pré-processamento que calcula o escopo da imagem resultante, obtém a lista da área de sobreposição entre as imagens, a qual é gerada na mínima árvore de abrangência. Com isso, o algoritmo divide a efetiva área de sobreposição por linha e então distribui as tarefas para os respectivos nós escravos com a estratégia paralela dinâmica agrupada. Cada nó escravo obtém a

sua própria tarefa e faz um registro para obter pontos de controle, então o nó mestre recolhe estes pontos de controle dos diferentes nós escravos, depois de todas tarefas terem sido finalizadas.

Na parte de reamostragem, o nó mestre obtém a lista de tarefas para reamostragem e para *blending*, divide a efetiva área e distribui as tarefas para o respectivo nó escravo, que obtém a sua própria tarefa, faz a reamostragem, escreve o bloco da imagem resultante novamente e mantém a área de sobreposição da imagem reamostrada reservada para ser usada. Quando o par de imagens está pronto, as regiões de sobreposição são divididas, misturadas e escritas de volta. A Figura 49 apresenta o esquema de mosaico paralelo proposto pelos autores.

O algoritmo é paralelizável em duas camadas: o algoritmo é paralelo entre os nós e o algoritmo é paralelo *multithread*, com as *threads* de leitura, escrita e processamento em cada nó. E mais, a pré-busca de dados e os dados assíncronos de E/S também levam considerável tempo por conta da sobreposição juntamente com o tempo de construção do mosaico. Por fim, o algoritmo evita distribuir os dados repetidamente e os dados de leitura, para redução do tempo de processamento, alternando a reamostragem.

A ideia principal da estratégia paralela dinâmica agrupada baseada na mínima árvore de abrangência é gerar uma lista de subtarefas de registro e reamostragem/junção, baseada na mínima árvore de abrangência para distribuir a lista de subtarefas para os grupos os quais distribui as subtarefas dentro do grupo, dinamicamente. O algoritmo é paralelo entre os nós e também paralelo em cada nó. O nó mestre distribui uma área sobreposta para um grupo de nós, os quais podem compartilhar dados da imagem sobreposta para garantir a realização do registro da imagem. Então o nó mestre distribui a subtarefa que corresponde a cada bloco de busca, dividido em linha, para todo nó em cada grupo. O esquema da mínima árvore de abrangência pode ser visto na Figura 50.

Figura 49 – Metodologia paralela proposta.

Fonte: Wang et. al. (2010b)

Figura 50 – A mínima árvore de abrangência.

Fonte: Wang et. al. (2010b)

A parte de reamostragem é alternada com a parte de junção no algoritmo. O algoritmo tem equilíbrio de carga pela distribuição dinâmica de tarefa. As tarefas são alocadas para cada nó, um por um e, alocados de acordo com a condição anterior do nó. Cada nó escreve na primeira imagem de um par de imagens sobrepostas e lê na área sobreposta para uso posterior, depois de a primeira imagem ter sido reamostrada. Então o nó vai fazer a reamostragem da segunda imagem do par, lê a área de sobreposição a qual é usada para se juntar com a área de sobreposição da primeira imagem e então escreve de novo na área de sobreposição, na imagem de destino. Todo este processo pode ser verificado na Figura 51.

Para lidar com o problema do tempo de espera de E/S, uma estratégia oculta é apresentada. O esquema paralelo apresentado, baseado na distribuição dinâmica de tarefas e *multithreads* paralelas de E/S elimina o tempo de espera de E/S e calcula por sobreposição, o bloco de imagem inicial e a escrita do bloco da imagem de destino. Há três *threads* neste esquema: ‘*thread* de leitura de E/S’ a qual é responsável por obter a área do bloco da imagem de entrada para o próximo bloco e então ler os dados naquela área do bloco, ‘*thread* de escrita de E/S’ que é responsável pela saída dos dados resultantes do bloco anterior e a ‘*thread* de processamento’ o qual trata os dados de reamostragem/junção do bloco atual.

Os resultados do artigo de Wang et. al. (2010b) foram gerados em um *cluster* do modelo Dawning 3000 Blade Engine HPC, que possui dez nós com oito CPU Intel Xeon X5530 com 8GB DDR3 e 11Tb de HD. Todos os nós são interligados com uma estrutura de redes de 20Gb/s e utiliza S.O. Centos 5.3. Foram utilizadas 16 imagens Landsat, com o escopo de 10000*10000 no *cluster*. Os resultados gerados podem ser observados na Figura 52.

Figura 51 – Diagrama de reamostragem e mistura.

Fonte: Wang et. al. (2010b)

Figura 52 – Resultados de *Speedup* gerados.

Fonte: Wang et. al. (2010b)

O artigo de Liang et. al (2010) diz respeito a modificação da Transformada SIFT (detecção de pontos chave em cada oitava) para detectar pontos característicos mais rapidamente. A parte mais custosa da Transformada SIFT é a detecção dos extremos e descrição das características.

A Figura 53 apresenta o diagrama original da Transformada SIFT. A Figura 54 apresenta o diagrama da Transformada SIFT modificada. A etapa de construção do

espaço escala e da pirâmide de *DoG* foram mantidas conforme a metodologia original, então quando se inicia a detecção dos pontos chaves em cada oitava, uma lista é criada. Cada ponto chave detectado é adicionado a esta lista até que todos pontos, de todas escalas e oitavas sejam detectados.

O próximo passo é descrever os pontos chaves. Para cada lista de pontos chaves, são escolhidos os pontos chaves para as *threads*, para a extração das características. Após completar o cálculo de extração de características para todas as listas, então é obtido o descritor de características.

Figura 53 – Transformada SIFT original.

Fonte: Liang et. al. (2010)

Figura 54 – Transformada SIFT modificada.
Fonte: Liang et. al. (2010)

Os testes efetuados pelos autores medem o tempo de processamento em imagens de diferentes resoluções para a Transformada SIFT original e modificada. A Tabela 11 mostram os resultados comparativos.

Tabela 11 – Comparativo dos testes em Liang et. al (2010).

	160x240 pixels	240x320pixels	480x720 pixels
Transformada SIFT original	16,58 ms	27,09 ms	52,11 ms
Transformada SIFT modificada	10,32 ms	19,51 ms	36,2 ms

O trabalho de Vasilyev (2011) implementa, utilizando CUDA uma maneira de paralelizar a Transformada SIFT. O esquema da metodologia adotada pelos autores pode ser verificado da Figura 55 à Figura 59.

A Figura 55 apresenta como é feita a extração das oitavas e seus subníveis (que é realizado sequencialmente para economizar o uso de memória da GPU).

Blocos verdes e azuis são chamados GPU-memória para iteração atual, sendo que os blocos verdes usam representação matricial dos dados da GPU.

Figura 55 – Processamento das oitavas e seus subníveis.
Fonte: Vasilyev (2011)

Para a construção do espaço escala, há alocação de memória na GPU. A matriz geradora do filtro gaussiano é feita somente na GPU, bem como a matriz parcial da *DoG* e a detecção dos extremos. A redistribuição dos pontos chave é realizada na CPU e GPU, de acordo com a Figura 56.

Figura 56 – Exemplo de redistribuição de pontos chave.
Fonte: Vasilyev (2011)

A “*base grid*” é o mapa de extremos, o “*indexed grid*” é usado para converter um mapa para uma matriz linear e “*integral sum*” é usado para formar uma matriz integral.

A determinação da orientação pode ser verificada na Figura 57 e Figura 58, após todas oitavas terem sido processadas. Para isto, o esquema da representação do espaço escala é usado e o processamento é todo realizado na GPU.

Figura 57 – Esquema geral do cálculo da orientação.
Fonte: Vasilyev (2011)

Se várias orientações são determinadas para um *blob* (estrutura de dados binária, uma imagem por exemplo), a redistribuição é realizada afim de construir um conjunto de *blobs*, tendo uma orientação, sendo que este processo é realizado na GPU e CPU. A geração final do descritor é realizada na GPU e um exemplo pode ser visto na Figura 59.

Figura 58 – Exemplo de cálculo da orientação e reindexação.
Fonte: Vasilyev (2011)

Figura 59 – Exemplo de geração do descritor.
Fonte: Vasilyev (2011)

Outros detalhes sobre a metodologia do trabalho de Vasilyev (2011) podem ser encontrados no próprio artigo do autor.

Os testes foram realizados utilizando uma GPU do modelo GeForce GTX 260 em imagens de 800 x 600 pixels. Os resultados são apresentados na Tabela 12, a partir da execução em uma única imagem.

Tabela 12 – Resultados obtidos em Vasilyev (2011).

Extração dos pontos chave	37,5ms / 975 blobs
Determinação da orientação	5,1ms / 1139 blobs
Geração do descritor	43,89 ms
Tempo total	87,2 ms

Os autores terminam o artigo concluindo que a metodologia por eles desenvolvida contribui para diminuir o tempo de processamento.

O trabalho de Huang et. al. (2011) apresenta uma maneira de otimizar a Transformada SIFT através de GPU, utilizando o *framework* CUDA (*Compute Unified Device Architecture*). Os resultados demonstraram que esta metodologia atingiu 57,8 vezes de *Speedup*.

Das três etapas do processo de cálculo do espaço escala (pirâmide gaussiana) da SIFT (filtragem Gaussiana, aplicação de escalas na imagem e *DoG*), a mais custosa é a filtragem gaussiana. Na pirâmide de escala *DoG* há três passos para verificar se um pixel é um ponto extremo da *DoG* ou não:

1. Comparação do pixel central com os oito pixels vizinhos na imagem *DoG*. Enquanto isso, verificar se o limiar de contraste é alcançado.

2. Comparação com os dezoito pixels adjacentes nas duas vizinhanças das imagens *DoG* no espaço escala.
3. Verificar o efeito da borda.

Os passos 1 e 2 podem ser simplificados para comparação com os 26 pixels vizinhos ao mesmo tempo. Este processo pode ser atribuído a um *kernel thread* para computação paralela, fazendo uso de vários núcleos na GPU.

Para calcular a direção dos pontos característicos, o histograma do gradiente deve ser colocado na memória compartilhada do bloco de *threads*. Memória compartilhada é a cache interna da GPU, cuja largura de banda e tempo de latência são menores do que na memória global. Uma vez que cada segmento terá acesso ao histograma, e considerando que as estatísticas do histograma são de acesso aleatório e de grande volume (difícil de otimizar), é possível então utilizar a memória compartilhada do CUDA eficiente.

O cálculo do descritor no bloco de *threads* é gerado por cada *thread* processa pontos de amostragem. Informação do cálculo do gradiente é adicionado aos descritores armazenados na memória compartilhada. Por fim, os descritores de características são copiados da memória compartilhada para a memória global e saída.

O registro das características SIFT é feita por correlação através da multiplicação de matrizes. Coeficiente de correlação é produto do ponto de descritores de recursos. Isso é feito por uma quantidade de *threads* e procurando o ponto característico mais próximo de um dada imagem J para todo ponto característico em uma dada imagem I, chamada de linha de registro. Similiarmente, abrindo n *threads* e procurando o ponto característico mais próximo na imagem I para todo ponto característico na imagem J, denominado de coluna de registro.

Os testes foram realizados utilizando uma *CPU Intel Xeon E5506* de 2.13 GHz com 24GB de RAM com uma GPU *Nvidia Tesla C2050* no Sistema Operacional Windows 7 64 bits com CUDA 3.2. A Tabela 13 apresenta os resultados comparativos de desempenho, medidos em milisegundos para a CPU e para a GPU.

Tabela 13 – Comparação dos resultados (tempos em ms) de Huang et. al. (2011).

Etapas	Par de Imagens 1		Par de Imagens 2	
	SiftCPU	SiftCUDA	SiftCPU	SiftCUDA
1. Cópia dos dados da imagem e inicialização da textura	xxx	65	xxx	101
2. Construção do espaço escala	592	23	2044	24
3. Extração dos pontos chaves.	374	32	1295	33
4. Cálculo da direção	437	20	1513	20
5. Cálculo do descritor	2231	31	7644	33
6. Registro das características	328	31	562	52
Tempo total	3962	202	13058	226
Pontos Registrados	79	86	234	257
Taxa de aceleração	19,6x		57,8x	

Na Tabela 13, a taxa de aceleração de todas etapas é superior a 10. Em particular, a grande dimensão das imagens SAR tem o benefício do alto desempenho da GPU utilizada (que possui 448 núcleos), fazendo o algoritmo ser otimizado em 57,8 vezes, sendo possível concluir que a otimização é maior em GPU do que em clusters.

O trabalho de Zheng et. al. (2012) propõe paralelizar a Transformada SIFT utilizando um *cluster* de computadores. A técnica por eles desenvolvida foi denominada de pSIFT. Armazenamento redundante e comunicação são utilizados ao limite dos dados para melhorar o desempenho e posterior representação do descritor de características. Realocação de dados é adotado para manter o equilíbrio de carga no pSIFT.

A Figura 60 apresenta o diagrama do pSIFT. Após abrir a imagem de entrada, a mesma é particionada e distribuída para todos os nós; a pirâmide da Gaussiana é gerada e os extremos locais são detectados usando dados locais em cada nó. Todos os nós enviam o número de pontos chaves para o nó mestre (nó mestre é o nó com identificador 0 na Figura 60). O nó mestre calcula o valor médio de acordo com o número de pontos chave e nós e, distribui para todos os nós. Todos os nós enviam os pontos chaves com uma pequena região em torno, que excede o valor médio para o nó mestre e o nó mestre realoca este dado para os nós cujo número de pontos chave é menor do que o valor médio. Todos os nós calculam o descritor usando dados locais em paralelo.

Figura 60 – Diagrama do pSIFT
Fonte: Zheng et. al. (2012)

Durante a geração da pirâmide da gaussiana, a imagem base da oitava atual é dada a partir da última oitava, de modo que o procedimento da geração de oitavas não pode ser completamente paralelizada. De acordo com o número de nós, a imagem de entrada é alocada para todos os nós de forma uniforme e todos os nós calculam a pirâmide da gaussiana nos dados da imagem local. A divisão da imagem de entrada é diversa, mas somente duas são de uso comum, como mostrado na Figura 61.

Figura 61 – Métodos de divisão dos dados
Fonte: Zheng et. al. (2012)

Se o limite dos dados é redundante, dois métodos de divisão são mostrados na Figura 61 e não tem nenhum efeito no algoritmo de paralelização, mas para a comunicação dos dados, os dois métodos terão diferente impacto no desempenho. Para a Figura 61a, um nó se comunicará com dois nós adjacentes ou mais. Para a Figura 61b, um nó se comunicará com até quatro nós, o que trará sobrecarga de

comunicação adicional e, adiciona complexidade ao programa. Os autores adotaram o esquema da Figura 61a.

O principal problema gerado pela divisão dos dados é o processamento do limite de dados. Após a divisão das imagens de entrada, a parte do meio das imagens de entrada deverá ser o limite de dados de um nó e pontos chave deverão ser extraídos a partir desta área.

Há dois métodos para o limite dos dados; um é o armazenamento redundante e o outro é a comunicação. Na geração da pirâmide da gaussiana, a imagem base da oitava atual é dada a partir da última oitava e por isso que se o armazenamento redundante é usado, haverá grande volume de dados para ser redundante e cálculos adicionais deverão ser interpostos. Maiores detalhes sobre o processamento do limite de dados pode ser encontrado no artigo do Zheng et. al. (2012).

Há necessidade da sincronização dos dados antes do cálculo do descritor, pois o número de pontos chave em cada nó não é igual e o equilíbrio do balanço fica variável com diferentes imagens de entrada. Para isso, todos ‘nós’ enviam o número de pontos chave para o nó mestre, que calcula e distribui o valor médio baseado no número total de nós e pontos chave. Todos ‘nós’ comparam o número de pontos chave com o valor médio e o ponto chave que exceder o valor médio é enviado para o nó mestre contendo posição, escala, orientação do gradiente e uma pequena região ao redor. O nó mestre distribui estes pontos chaves para o nó cujos números de pontos chave é menor do que a média. Cada nó calcula o descritor em paralelo.

Os testes foram realizados no *cluster* desenvolvido pela Universidade Nacional de Defesa Tecnológica (NUDT), com a seguinte configuração: 48GB de memória RAM, 6 núcleos *Xeon CPU X5670* com 2,93 GHz de *clock* e 12MB de *cache* L3, sistema de arquivos *luster* e taxa de transferência da rede de 160 Gbps. As imagens utilizadas para os teste possuem dimensão de 2048x2048 pixels.

A Figura 62 mostra o *Speedup* do pSIFT em comparação com a DDP-SIFT e pode-se observar que a pSIFT foi superior. A Figura 63 mostra os resultados comparativos entre pSIFT e nsSIFT em clusters de 32 núcleos e a pSIFT apresentou um desempenho superior.

Figura 62 – *Speedup* entre pSIFT e DDP-SIFT.
Fonte: Zheng et. al. (2012)

Figura 63 – Desempenho entre pSIFT e nsSIFT.
Fonte: Zheng et. al. (2012)

A Tabela 14 apresenta um resumo das metodologias utilizadas por cada um dos trabalhos descritos nesta Seção.

Tabela 14 – Resumo das paralelizações da Transformada SIFT.

Autor(es)	Metodologia	Configuração	Dimensão das Imagens	Speedup / tempo
Feng et al. (2008)	<i>OpenMP</i> C/C++ Processador SMP e CMP	SMP: HP ProLiant DL 580 G5 CMP: simulador de 64 núcleos 2Ghz	720 x 576 1920 x 1080	SMP: 11 CMP: 52
Liu, Chen e Li (2009)	GPU	Dell Latitude D830, GeForce 135M	400 x 300	CPU: 4,8s GPU: 2,9s (SIFT)
Warn et al. (2009)	<i>OpenMP</i> SMP CUDA	SMP: Quad Core Xeon, 2,66 Ghz CUDA: GPU FX 5800, Intel E6550 2,33Ghz	8272 x 2848	CPU: 2180ms GPU: 168ms (convolução)
Jiang et al. (2010)	Cluster C, OpenCV MPI	32 núcleos	1024 x 768	11
Wang et al. (2010)	cluster	8 CPU Intel Xeon X5530, 8Gb RAM, interconectados por rede com 20Gb/s	10000 x 10000	33
Liang et al. (2010)	<i>threads</i>	Não cita	480 x 720	SIFT: 52,1 ms SIFT modificada: 36,2 ms
Vasilyev (2011)	CUDA	GeForce GTX 260	800 x 600	87,2 ms (descritor)
Huang et al. (2011)	CUDA	Xeon E5506 2.13 Ghz, 24Gb RAM Nvidia Tesla C2050	Imagens SAR	57,8
Zheng et al. (2012)	Cluster	6 núcleos Xeon X5670 2,93Ghz, 48Gb RAM, interconectados por rede com 160Gbps	2048 x 2048	33

4.5. Considerações Finais

Dentre os trabalhos relacionados a algoritmos de extração de características, pode-se citar o trabalho de Juan e Gwun (2009) que apresenta vários testes envolvendo a Transformada SIFT, SURF e PCA-SIFT. Os autores concluem que cada metodologia possui sua particularidade e a escolha de uma delas, depende do objetivo do projeto em questão.

No trabalho de Mikolajczyk e Schmid (2005) foi constatada que a Transformada SIFT apresenta um desempenho superior em relação aos algoritmos mencionados no artigo em termos de invariância à rotação e escala.

Os autores do SURF, Bay et al. (2008), afirmam que o algoritmo que desenvolveram é superior ao da Transformada SIFT em termos de eficiência de tempo de processamento e que o mesmo apresenta um bom desempenho no processo de reconhecimento de objetos, considerando a qualidade dos pontos de características.

Os artigos de Yang et. al. (2011), Khan et. al (2011), Bai et. al (2012) e Ali (2012) comparam o desempenho da Transformada SIFT com SURF, concluindo que a Transformada SIFT possui maior custo computacional que SURF, mas apresenta melhor desempenho na detecção de características e correspondências das mesmas.

De acordo com os trabalhos apresentados na Seção 4.2 sobre construção de mosaicos, é possível comprovar a eficiência dos algoritmos SIFT, RANSAC e BBF para a aplicação de mosaicos, sendo por esta razão adotados no presente trabalho. Os autores mencionados no início desta seção mencionam que a Transformada SIFT é indicada quando se tem muitos detalhes na imagem e quando é exigido precisão na junção dos mosaicos.

A respeito da Seção 3.4, que trata do paralelismo aplicado à Transformada SIFT, são apresentados trabalhos com diferentes técnicas de paralelismo. O trabalho de Feng et. al.(2008) apresenta uma maneira de paralelizar a Transformada SIFT por SMP e CMP. De acordo como os detalhes apresentados neste artigo, a implementação SMP é mais simples que a CMP.

No trabalho de Liu, Chen e Li (2009) a Transformada SIFT foi paralelizada por GPU, utilizando CUDA, e teve um bom ganho de desempenho em relação à implementação em CPU.

No trabalho de Warn et. al (2009) a Transformada SIFT foi paralelizada por *OpenMP* e por CUDA. A implementação CUDA teve melhor desempenho em relação a *OpenMP*, porém é mais custosa de se implementar.

O trabalho de Jiang (2010) apresenta uma maneira de paralelizar a Transformada SIFT por *cluster* de computadores, fazendo um estudo detalhado dessa Transformada, que torna a implementação da paralelização complexa.

No trabalho de Wang et. al. (2010) é apresentada uma metodologia para acelerar a construção de mosaicos utilizando um *cluster* de computadores, e o artigo mostra que não é trivial fazer esta implementação.

Neste projeto de doutorado, no que se diz respeito a processamento paralelo, o intuito é diminuir o tempo de execução para a construção do mosaico de imagens de maneira simples, prática e funcional, principalmente para o usuário final.

Das técnicas citadas nos artigos, destaca-se a *OpenMP* que mostra ser mais simples de se implementar e mais adequada para um usuário final, uma vez que os processadores atuais (SMP) estão sendo fabricados para tirar proveito do padrão

OpenMP e os softwares implementados por este padrão de programação paralela são facilmente portáveis para qualquer tipo de computador SMP, sem necessidade de nenhum pré-requisito ou configuração específica. Além disso, o padrão *OpenMP* tem a capacidade de configurar automaticamente a quantidade de *threads* disponíveis em um computador SMP, fazendo com que o usuário final utilize o software de maneira prática e simples. Por estes motivos e fatores citados, neste trabalho foi adotado este padrão de programação paralela.

A técnica CUDA que utiliza processamento paralelo em GPU possui a necessidade de adaptação do código para cada modelo de GPU (placa de vídeo), dificultando seu uso pelo usuário final.

5. Metodologia Desenvolvida

5.1. Considerações Iniciais

Este Capítulo apresenta na Seção 5.2 a descrição da base de imagens e nas seções seguintes as metodologias para a implementação dos mosaicos de imagens agrícolas aéreas utilizando processamento sequencial e paralelo.

5.2. A Base de Imagens

As imagens aéreas utilizadas no presente projeto foram fornecidas pela EMBRAPA (Empresa Brasileira de Pesquisas Agropecuárias) Instrumentação, situada em São Carlos – SP. As imagens contêm áreas de pastagens, lavouras e áreas urbanas, obtidas na região de Santa Rita do Sapucaí – MG em setembro de 2007, com o auxílio de um avião tripulado de pequeno porte, contendo uma câmera acoplada a um suporte externo.

As imagens possuem 24 bits por pixel, com 72 dpi (pontos por polegada) no formato JPG e foram obtidas em 8 megapixels. A câmera utilizada é do modelo “Canon EOS 30D” com 12 Megapixels de resolução, com as seguintes características:

- Velocidade do obturador e tempo de exposição: 1/5000 s;
- Abertura da lente e abertura do diafragma: f/3,5;
- Comprimento focal: 18 mm.

A Figura 64 ilustra um exemplo de imagem usada no projeto.

Figura 64 – Exemplo de imagem usada no projeto.

As imagens foram obtidas seguindo um padrão de aquisição, que inclui os seguintes itens:

- A câmera foi acoplada ao suporte externo do avião;
- O avião percorreu sempre a mesma distância, em trajetórias horizontais na área demarcada, como ilustrada na Figura 65;
- Após obter uma imagem, a câmera foi configurada para aguardar um tempo fixo (3s) antes de obter a próxima.

Figura 65 – Padrão de rota para obtenção das imagens.

Para a construção de um mosaico sem muita distorção, é de extrema importância que seja seguido o padrão apresentado para a obtenção das imagens, garantindo dessa maneira que haja uma área comum entre as imagens obtidas em sequência o que permite que elas sejam sobrepostas.

Quando esta trajetória é alterada devido a problemas de massa de ar e dificuldade do piloto em manter o avião sem grandes inclinações e com pouca variação de altitude, as imagens obtidas tem a qualidade alterada para fins de

construção de mosaicos, podendo ocasionar a interrupção da construção do mesmo quando não há área de sobreposição entre a imagem anterior e a seguinte. Devido aos problemas citados, algumas imagens possuem problema de inclinação e de iluminação, esta última causada principalmente pelas condições climáticas.

A partir das imagens adquiridas foram criadas duas bases de imagens: uma base de imagens sem redimensionamento e outra com redimensionamento. A base de imagens sem redimensionamento é referida neste trabalho como 'base de imagens sem redimensionamento e contém 100 imagens divididas em 10 grupos de 10 imagens sendo cada imagem com dimensão de 2336 x 3504 pixels. Os Apêndices A e B apresentam a base de imagem sem redimensionamento bem como os mosaicos gerados, respectivamente.

A outra base de imagens é referida como base de imagens redimensionada e contém 200 imagens divididas em 10 grupos de 20 imagens, tendo cada imagem dimensão de 533 x 800 pixels. A base de imagens redimensionada foi utilizada para comparação de distorção com outros softwares de construção de mosaicos, os quais possuem parâmetros pré-definidos para a resolução das imagens de 533 x 800 pixels. Os Apêndices C e D apresentam a base de imagens redimensionada e os mosaicos gerados, respectivamente.

A base de imagens sem redimensionamento foi utilizada para avaliação de tempo de processamento/custo computacional. Apesar de o software suportar uma quantidade maior do que 10 imagens por grupo, o valor 10 foi definido para redução do custo computacional da construção do mosaico, uma vez que vários testes foram realizados para testar as metodologias desenvolvidas e também pela necessidade de excluir imagens sem área comum. Quando não se tem área comum, o valor do parâmetro T (número mínimo de *inliers*) do algoritmo RANSAC (Seção 2.7.1) não é atingido, fazendo com que o processo de construção de mosaicos se interrompa inesperadamente.

5.3. Metodologia Desenvolvida

Inicialmente foi desenvolvida uma metodologia, denominada Metodologia 1, para compor o mosaico. Nessa metodologia, os parâmetros de cada técnica são

determinados sequencialmente para cada par de imagens e no final é composto o mosaico.

Com o objetivo de reduzir o custo computacional foram criadas posteriormente, duas versões modificadas da metodologia 1, denominadas respectivamente de Metodologia 2 e Metodologia 3.

Na metodologia 2, o algoritmo calcula todos os parâmetros necessários para compor o mosaico, para todas as imagens que farão parte do mosaico, antes de haver a união das mesmas; ou seja, o software faz todos os cálculos antecipadamente para posteriormente unir todas as imagens.

Na metodologia 3 as imagens de alta resolução são inicialmente redimensionadas em 70% para calcular os parâmetros de composição do mosaico. Esses valores são então estimados para as imagens de alta resolução e aplicada a metodologia 1 para determinar a região de sobreposição e construir o mosaico.

Em seguida, o algoritmo de cada metodologia foi paralelizado, utilizando o padrão *OpenMP* e *MPI*. Os resultados de construção dos mosaicos para as versões sequenciais e paralelas foram avaliados.

Foi também avaliada a distorção para cada mosaico gerado, utilizando a base de imagens redimensionada. Os softwares *Autostitch* e *PTGui* foram utilizados para gerar esses mesmos mosaicos, com o objetivo de avaliar a distorção por eles gerada e comparar com a distorção da metodologia 1.

5.3.1. Metodologia 1

Na metodologia 1, para a construção do mosaico é feita a junção das imagens por pares sequenciais, ou seja, 1ª imagem com a 2ª imagem, formando uma imagem denominada “imagem mosaico”, que na sequência é juntada à 3ª imagem e assim por diante. A “imagem mosaico” é a imagem que vai crescendo conforme as junções vão sendo realizadas até formar o mosaico completo. Neste trabalho toda imagem a ser unida à “imagem mosaico” é denominada “imagem2”.

A Figura 68 mostra um diagrama contendo as principais técnicas empregadas neste trabalho para a construção do mosaico.

A primeira etapa consiste em retificar (ajustar) as imagens para minimizar possíveis distorções de translação, rotação e escala que podem ocorrer na fase de aquisição das imagens.

De acordo com a Figura 68, o processo se inicia pela aplicação da retificação cilíndrica para minimizar possíveis distorções de translação, rotação e escala que podem ocorrer na fase de aquisição das imagens. Neste trabalho foi utilizada a retificação cilíndrica, por possuir vantagens em relação à retificação planar, conforme discutido no Capítulo 2. Essa retificação foi feita de acordo com a Seção 2.3, baseando-se nas equações de (10) à (13). Após, são extraídas as características do par de imagens pela Transformada SIFT (Figura 66) e são criados os respectivos descritores (128 descritores para cada ponto chave), utilizando as quatro etapas que a compõe, de acordo com a Seção 2.4.

Após as imagens do primeiro par serem unidas, não há necessidade de extrair todas as características do mosaico parcial gerado, na etapa seguinte de união do mesmo com a próxima imagem (imagem2). Para reduzir o custo computacional, é utilizada somente a parte do mesmo que tenha dimensões similares (RoI) à da imagem2 (Figura 67).

Figura 66 – Características extraídas pela Transformada SIFT.

Figura 67 – Exemplo de região de interesse na "Imagem Mosaico".

Figura 68 – Diagrama da metodologia empregada.

Uma vez que as características SIFT foram extraídas, os descritores constituídos são utilizados para fazer a correspondência das características do par de imagens a ser unido (BBF), de acordo com a Seção 2.6.1.2 (LOWE, 2004). Um exemplo de características comuns entre um par de imagens sequencial pode ser visualizado na Figura 69. Nesta figura, marcadores de mesmo número em ambas as imagens são consideradas características em comum.

Figura 69 – Correspondência das características SIFT (Identificação).

Seguindo a sequência do diagrama da Figura 68, é utilizado o algoritmo RANSAC para estimar a matriz homográfica (matriz 3x3 que contém os parâmetros de escala, rotação e translação) e também eliminar falsas correspondências (*outliers*), conforme ilustrado na Figura 70. Nessa figura, as linhas representam as correspondências das características entre duas imagens sequenciais, após aplicação do RANSAC. No algoritmo desenvolvido a matriz homográfica é referente à “imagem2”, calculada com base na “imagem mosaico”.

Os algoritmos da Transformada SIFT, do RANSAC e do BBF utilizados nesta metodologia foram obtidos de Hess (2010), de acesso livre para pesquisa científica. Os valores dos parâmetros livres desses algoritmos foram utilizados de acordo com Lowe (2004) e Hess (2010), uma vez que testes foram realizados e estes valores se mostraram adequados. Os parâmetros livre do algoritmo RANSAC são:

- S (vetor de descritores correspondidos). $S = []$. A quantidade de S é variável, pois a quantidade de características extraída de cada imagem é variável.
- s (quantidade mínima de amostras de S). $s=8$ (4 correspondências).
- t (limiar de distância para ser considerado *inlier*). $t=4$ pixels.

- T (número mínimo de *inliers* a ser alcançado para que um modelo possa ser considerado um bom modelo). T=3

Figura 70 – Correspondência após aplicação do RANSAC.

Com a estimação da matriz homográfica, antes da sua aplicação na “imagem2”, as imagens a serem unidas têm de ser posicionadas em suas respectivas janelas do mosaico parcial, sendo necessário verificar os valores de translação da matriz homográfica. Se esses valores forem negativos, deve ser feito um ajuste de translação das imagens, pois um valor negativo em X e/ou Y significa que a “imagem2” está fora da região visível do monitor de vídeo ou mesma sobreposta à “imagem mosaico” (Figura 71), impossibilitando a união das mesmas.

Figura 71 – Exemplo de imagem fora da região visível do monitor.

Para realizar este ajuste, são geradas duas matrizes derivadas da matriz homográfica, uma para cada imagem (“imagem2” e “imagem mosaico”), onde cada matriz tem o valor de translação alterado para cada imagem. A Figura 72 apresenta um exemplo de como acontece este ajuste, baseando-se nos valores da matriz homográfica dada como exemplo.

Na Figura 72, as duas matrizes geradas são identificadas como “matriz imagem mosaico” e “matriz imagem 2”, e a “matriz homográfica” é apenas para exemplificar a localização das variáveis de translação X e Y na “matriz homográfica (real)”.

Para o cálculo são feitas as seguintes considerações:

- Se o X da “matriz homográfica (real)” for negativo então $X_{im} = -X$ e $X_{i2} = 0$; caso contrário $X_{im} = 0$ e $X_{i2} = X$;
- Se o Y da “matriz homográfica (real)” for negativo então $Y_{im} = -Y$ e $Y_{i2} = 0$; caso contrário $Y_{im} = 0$ e $Y_{i2} = Y$.

Quando o X é negativo, é necessário deslocar a “Imagem Mosaico” para a direita, pois se isso não for feito, a “imagem 2” não terá espaço suficiente para a junção; o mesmo acontece quando o Y é negativo, e neste caso a “Imagem Mosaico” deve ser deslocada para baixo.

Figura 72 – Geração da matriz homográfica.

Para o exemplo da Figura 72 é possível observar que a “imagem mosaico” não será transladada (matriz identidade) e que a “matriz imagem 2 (real)” final ficou com os mesmos valores da “matriz homográfica (real)”, uma vez que os valores de X

e Y são positivos e a translação somente acontece para casos em que X e/ou Y são negativos.

A partir das matrizes geradas na etapa anterior, são criadas duas janelas de imagem, sem conteúdo (Figura 73). De acordo com a Figura 72, o X e o Y da matriz homográfica não são negativos, sendo assim, a janela da “imagem mosaico” continua do mesmo tamanho e a janela da “imagem 2” sofreu um deslocamento em X (231,752819) e um pequeno deslocamento em Y (25,142461), aumentando tanto em altura como em largura para poder se encaixar à “imagem mosaico”. Este processo é ilustrado na Figura 73.

Figura 73 – Exemplo de janela de imagens.

Na “imagem mosaico” e na “imagem 2” são aplicadas as matrizes homográficas e essas imagens são apresentadas nas suas respectivas janelas (Figura 74). Nessa figura, a “imagem mosaico” não sofre alteração, pois sua matriz homográfica resultante é a matriz identidade (Figura 75). Já para a “imagem 2” a Matriz Homográfica é aplicada proporcionalmente até os 50% (na horizontal) dessa imagem, de modo a evitar a propagação de erros na sequência de imagens unidas. Nos 50% restantes da imagem é aplicada somente a matriz identidade (condições ideais de rotação e escala), deixando apenas o parâmetro de translação em x e y. Experimentalmente verificou-se que o valor de 50% da porção da imagem no eixo horizontal representa a faixa suficiente para a união das mesmas e minimiza as distorções na imagem resultante (mosaico parcial).

Figura 74 – Exemplo de imagens nas janelas.

$$\begin{bmatrix} 1 & 0 & Xi2 \\ 0 & 1 & Yi2 \\ 0 & 0 & 1 \end{bmatrix}$$

Figura 75 – Exemplo da matriz identidade

Para controlar a aplicação da matriz homográfica (H) na imagem a ser unida (imagem2), é utilizada uma matriz aditiva (F), sendo cada elemento obtido de acordo com a equação (36), de acordo com Silva (2012b).

$$f_{i,j} = \frac{\lambda - h_{i,j}}{0.5w} \quad (36)$$

sendo: $\lambda=1$ se $i=j$ e $\lambda=0$ se $i \neq j$ e w é a largura da imagem.

As cores de pixel da imagem resultante $I_r(x_r, y_r)$ são obtidas a partir das cores da imagem2 $(x,y,1)$, onde as coordenadas x_r e y_r são obtidos com a equação (37). Esta multiplicação é realizada ponto a ponto. O a representa um fator de degradação, que controla a aplicação da matriz homográfica, sendo que na diagonal da matriz degrada o dobro do que nos outros parâmetros, mas deixando livres os parâmetros de translação em x e y . A degradação começa mínima na última coluna e conforme vai se aproximando do 50% da imagem I_r , vai aumentando até o seu valor total.

$$(x_r, y_r, 1) = (x, y, 1) \cdot \begin{pmatrix} f_{1,1} \cdot a + h_{1,1} & f_{1,2} \cdot a + h_{1,2} & h_{1,3} \\ f_{2,1} \cdot a + h_{2,1} & f_{2,2} \cdot a + h_{2,2} & h_{2,3} \\ f_{3,1} \cdot a + h_{3,1} & f_{3,2} \cdot a + h_{3,2} & 1 \end{pmatrix} \quad (37)$$

A aplicação da matriz homográfica pode ocasionar o surgimento de riscos pretos na imagem (Figura 74), devido a problemas inerentes do cálculo. Devido a isso, após a aplicação da matriz homográfica, é aplicado a interpolação dos 4 vizinhos mais próximos, somente nos riscos verticais pretos, para a sua eliminação.

Ainda de acordo com a Figura 74, o ponto de máximo X e o ponto de mínimo X são calculados quando se aplica a matriz homográfica nas imagens. A região de sobreposição (RS) é calculada pela diferença entre o ponto de máximo e o ponto de mínimo no eixo X (Figura 76).

Figura 76 – Exemplo de região de sobreposição.

Algumas vezes a região de sobreposição RS é extensa (próximo de 50% da imagem), podendo gerar um efeito denominado “*Ghost*”, (Figura 77) que são sombras geradas na região de sobreposição. Para minimizar este efeito é definido o tamanho de RS em 100 pixels. A Figura 78 apresenta uma alteração da região de sobreposição da Figura 76 dada como exemplo, gerando consequentemente um novo ponto de mínimo X e um novo ponto de máximo X.

A escolha do valor de RS igual a 100 pixels se deu por testes realizados, sendo que este valor se mostrou adequado para que haja a união das imagens que compõem a base de imagens deste trabalho. Diminuindo o valor de RS observou-se

que a quantidade de características não é suficiente para a junção das imagens. Aumentando o valor de RS aumenta o custo computacional e a melhoria na junção das imagens é visualmente imperceptível.

Figura 77 – Exemplo de “Efeito Ghost”.

Figura 78 – Exemplo de redução da região de sobreposição.

A partir do “Novo Ponto de Mínimo X” e “Novo Ponto de Máximo X”, são calculados os pesos dos pixels (w) da região RS, de acordo com a equação (38). O valor de RS igual a 100 pixels é fixado para possibilitar que ‘ w ’ varie de 0% a 100%.

$$w = (X - \text{NPM})/\text{RS}$$

(38)

sendo X a coordenada X do pixel da imagem 2, NPM o Novo Ponto de Mínimo X e RS a Região de Sobreposição.

A região de junção, geralmente apresenta uma diferença na iluminação da cena, gerando um mosaico com um aspecto de corte nas junções. Para minimizar este efeito, é aplicado o algoritmo de "*Blend Feathering*" (UYTTENDAELE et al., 2001) (LEVIN et al., 2006), que faz uma mistura de cor dos pixels correspondentes nas duas imagens do par (I_1 - mosaico e I_2 - imagem2). A mistura de cor dos pixels é dada pela equação (39):

$$I_f(i,j) = (1-w) \cdot I_1(i,j) + w \cdot I_2(i,j) \quad (39)$$

onde I_1 é a imagem mosaico e I_2 a imagem a ser unida, w é o peso do pixel e I_f é a imagem resultante.

De acordo com a equação (39), quanto maior o valor do peso (w), maior a contribuição dos pixels da imagem a ser unida (I_2) para os pixels correspondentes do mosaico.

Como já mencionado nesta Seção, no processo de junção das imagens, após aplicação da matriz homográfica, ocorrem distorções que são causadas pela sucessiva união de imagens. Sem as correções que são realizadas pelas equações (36) e (37) essas distorções são propagadas à medida que a matriz homográfica é aplicada a cada junção, gerando uma distorção final acentuada, como ilustra a Figura 79, transformando o horizonte numa linha sinusoidal.

Figura 79 – Imagem panorâmica sem ajuste (Brown; Lowe, 2007).

Brown e Lowe (2007) sugerem que esta distorção deva ser corrigida no final do processo de construção de mosaicos, utilizando um algoritmo de ajuste denominado "*Bundle Adjustment*" (Brown; Lowe, 2007), a fim de se gerar um mosaico em que o horizonte volte a se tornar uma linha horizontal. O resultado de tal correção é mostrado na Figura 80.

Figura 80 – Imagem panorâmica após ajuste (Brown; Lowe, 2007).

No presente projeto, diferentemente do trabalho de Brown e Lowe (2007), a correção das distorções causadas pela sucessiva união de imagens é feita ao longo do processo de construção de mosaicos, minimizando as distorções geométricas e melhorando a posição do horizonte do mosaico gerado, na forma de uma linha horizontal (SILVA et. al., 2012b).

Após todas essas etapas, pares de imagens são unidos, até formar um mosaico completo, contendo as imagens do seu respectivo grupo. Isso pode ser observado na Figura 81.

Figura 81 – Exemplo de mosaico contendo 20 imagens.

5.3.2. Metodologia 2

Construir mosaicos de imagens de alta resolução demanda alto custo computacional e devido a isso foi implementada uma nova versão do software, a partir da metodologia 1 para melhoria de custo computacional. A metodologia 2 faz todos os cálculos (cálculo dos pontos característicos, correspondência dos pontos característicos, eliminação dos *outliers* e estimação da matriz homográfica) de todas as imagens antecipadamente, para então depois fazer a união de todas as imagens de uma vez só.

A desvantagem é que há acúmulo de informações na memória, uma vez que todos os cálculos são feitos antecipadamente. Com isso, o funcionamento desta

versão de software fica limitada pela quantidade de memória do computador em uso e pela quantidade de imagens a serem unidas, uma vez que se a capacidade máxima de memória for atingida o processamento será interrompido por estouro de memória.

Para esta metodologia, os mosaicos gerados podem ter uma variação mínima em relação à metodologia 1, devido à estimação randômica de pontos pelo algoritmo RANSAC.

5.3.3. Metodologia 3

Nesta nova abordagem as imagens de alta resolução (3504 x 2336 pixels), que são utilizadas para a construção do mosaico, são redimensionadas em 70% (1052 x 700 pixels), criando as respectivas imagens em resolução mais baixa. Por testes realizados, o valor de 70% mostrou-se adequado para as imagens que compõem a base de imagens deste trabalho. Abaixo do valor de 70%, para algumas imagens, não é obtida a quantidade de características adequada para a construção do mosaico. Acima do valor de 70%, ocorre o aumento de custo computacional, não justificando o uso desses valores, uma vez que a junção das imagens já é feita com o valor de 70%.

A partir das imagens de baixa resolução, são calculados os seguintes parâmetros: pontos característicos, correspondência entre eles, eliminação dos *outliers* e estimação da matriz homográfica. Com isso é possível determinar a dimensão da região de sobreposição (RS) em cada imagem de baixa resolução; mas ainda não é feita a união das imagens. Com base nesse valor de RS é estimado o valor desse mesmo parâmetro para as imagens de alta resolução. Em seguida, com base na metodologia 1, é necessário calcular à cada par de imagens de alta resolução, somente na faixa da região de sobreposição (RS), os seguintes parâmetros: pontos característicos, correspondência entre eles, eliminação dos *outliers* e estimação da matriz homográfica. Como esta última etapa é aplicada a cada par de imagens de alta resolução, o mosaico vai sendo construído progressivamente como na metodologia 1. A Figura 82 apresenta o diagrama em blocos da metodologia 3.

Os blocos X e Y em verde da Figura 82 mostram a entrada (X) e saída (Y) do sistema e os demais blocos (em azul) mostram a contribuição e inovação da metodologia 3 em relação às metodologias tradicionais de construção de mosaicos e em relação à metodologia 1.

Para esta metodologia, os mosaicos gerados podem ter uma variação mínima em relação à metodologia 1, devido à estimação randômica de pontos pelo algoritmo RANSAC.

Figura 82 – Diagrama de blocos da metodologia 3.

Na metodologia 3 a dimensão da RS estimada fica limitada a um quinto da dimensão da imagem a ser unida (20%). O valor de RS igual à 20% foi determinado em testes realizados para a base de imagens deste trabalho, mostrando-se um valor adequado para que houvesse a junção das imagens. Abaixo dos 20% ocorreu falhas nas junções e aumento do efeito *ghost*, e acima deste valor há aumento de custo computacional sem melhoria visíveis na junção.

Além disso, é possível ao usuário reduzir a dimensão da RS estabelecida, para possíveis testes com outras bases de imagens. A diminuição da RS pelo usuário fica limitada pela qualidade das imagens adquiridas, uma vez que se estas imagens não possuírem resolução adequada, ao reduzir a RS, provavelmente haverá falhas em algumas junções.

5.4. Processamento Paralelo

O padrão *OpenMP* foi utilizado para paralelizar as metodologias descritas na Seção 5.3. Este padrão é adequado para processadores SMP multinúcleo, sendo de fácil utilização por usuários finais.

Os principais construtores e cláusulas do padrão *OpenMP* utilizados a fim de ganhar desempenho, foram:

- *Nowait*: retira a sincronização das *threads* nos casos em que é necessária independência de execução em determinado momento.
- *Omp_critical*: determina que somente uma *thread* vai executar a sessão do código, assegurando estabilidade de execução por evitar inconsistência de dados em tempo de execução.
- *Omp_flush*: responsável por atualizar o valor de determinada variável em todas as *threads*.
- *Omp_sections*: faz a divisão das tarefas entre as *threads*, em blocos de código onde não existem iterações (laços).
- *For – Schedule Dynamic*: controla a forma como as iterações são distribuídas entre as *threads*. Nesse tipo as iterações são distribuídas entre as *threads* à medida que as mesmas solicitam mais iterações. Cada *thread* executa um bloco de iterações e em seguida solicita outro bloco, e dessa forma o acesso não é necessariamente sequencial.

Para paralelizar um código sequencial há a necessidade de identificar as partes desse código que podem ser paralelizadas; geralmente o código sequencial não é completamente paralelizável, somente partes dele.

Para facilitar a identificação das partes paralelizáveis, foi utilizada a ferramenta de *Profiling Gprof* (Graham, 2004), que faz análise dinâmica de códigos sequenciais, por meio da coleta de informações e identificação de trechos de código mais constantemente solicitados por um programa em execução, além da porcentagem de tempo consumido por cada rotina de um programa.

Figura 83 – Exemplo de informações obtidas com o Gprof.

O Gprof, após sua execução, cria um arquivo que contém informação relevante ao custo computacional e gera um diagrama de hierarquia das chamadas das funções. Um exemplo de resultados gerado pelo Gprof é mostrado na .

De acordo com a Figura 83, em ‘a’ é apresentado o nome da função analisada, em ‘b’ a porcentagem do tempo total gasto pela função analisada, em ‘c’ a porcentagem de tempo para a função realizar sua tarefa, e em ‘d’ a quantidade de vezes que a função foi chamada. Em ‘e’ e ‘f’ são apresentadas as porcentagens de tempo usadas na chamada de outras funções ($b = c + e + f$).

Na Figura 83, a porcentagem de tempo total empregado pela função exemplo ‘MatrizHomografica’, está distribuída em 3 partes: ‘c’, ‘e’, ‘f’. Quando uma função é chamada, há duas possibilidades: executar tarefas contidas nela ou passar o trabalho para outras funções, fazendo novas chamadas. A porcentagem de tempo ‘c’ representa que ela fez tarefas sem passar trabalho para outras funções; e ‘e’ e ‘f’ representam as porcentagens de tempo gasto pelas tarefas executadas por outras funções, chamadas pela função no nível superior.

As metodologias 1,2 e 3 foram inicialmente desenvolvidas para plataforma Windows, implementadas em linguagem C na IDE Dev-C, utilizando a biblioteca de processamento digital de imagens *OpenCV*. Os códigos, inicialmente sequenciais, foram posteriormente adaptados para a plataforma Linux, utilizando o compilador GCC e *OpenCV* versão Linux, para fins de paralelização, pelo fato do *OpenMP* para Linux apresentar melhor desempenho.

A versão do software para o Linux gerou os mesmos resultados de mosaico que no Windows, com a vantagem que o Linux administra melhor a memória principal, sem gerar mensagens de erro por estouro de memória.

Nas próximas seções serão apresentados os detalhes da ferramenta *Gprof* para cada metodologia.

5.4.1. Processamento Paralelo da Metodologia 1

A Figura 84 mostra os resultados gerados pelo *Gprof* para o código da metodologia 1. A partir desses resultados, foi possível identificar as porcentagens de tempo total gasto por cada função, bem como a quantidade de vezes que cada função foi chamada. Após a análise do diagrama, foram escolhidas funções que permitem paralelização, iniciando pelas de alto custo computacional. É possível notar que um dos maiores custo computacional identificado pelo *Gprof* está na Transformada SIFT, e como já mencionado nos artigos de Feng et. al.(2008), Warn et. al (2009) e Huang et. al. (2011) é possível paralelização.

Uma consideração a ser feita a respeito da escolha das funções a serem paralelizadas é que a paralelização se torna mais simples para as funções que se localizam em níveis mais baixos da hierarquia. Outra consideração a ser feita é que nem todas as funções permitem paralelização devido a motivos como:

- Algumas funções possuem variáveis locais e globais, gerando dependências em tempo de execução e com isso, impossível de serem executadas em paralelo. Exemplo de funções que apresentam este tipo de problema: *MatrizHomográfica* (estima a matriz homográfica) e *calc_feature_cris* (localização dos pontos chave).
- A quantidade de chamadas de determinadas funções é inversamente proporcional à porcentagem de esforço computacional (milhões de chamadas para uma baixa porcentagem de esforço computacional). Pode-se citar como exemplo as funções *cvRound* e *cvFloor*.

A Tabela 15 mostra as funções selecionadas para a paralelização.

Tabela 15 – Funções paralelizadas na metodologia 1.

Função	Porcentagem de tempo (%)
aplicaHomografia	7,56
interpolacao	5,30
<i>compute_descriptors</i>	41,45
<i>space_scale_extrema</i>	20,44
Total	74,75

A função ‘aplicaHomografia’ é responsável por fazer a junção de um par de imagens, após cálculo das respectivas matrizes homográficas. A função ‘interpolacao’ é responsável por realizar a interpolação na respectiva imagem para eliminar os riscos pretos que podem surgir durante o processo de construção dos mosaicos. A função ‘*compute_descriptors*’ é responsável por calcular os descritores de características da Transformada SIFT. A função ‘*space_scale_extrema*’ é responsável por determinar os extremos no espaço escala, a primeira etapa da Transformada SIFT.

Com as devidas funções selecionadas e paralelizadas, o código ficou com 74,75% paralelizado do seu total e com 25,25% sequencial. Com esta porcentagem de paralelização o custo computacional foi reduzido.

Figura 84 – Diagrama gerado pelo Gprof para a metodologia 1.

5.4.2. Processamento Paralelo da Metodologia 2

A Figura 85 mostra os resultados gerados pelo *Gprof* para o código da metodologia 2. A partir desses resultados, foi seguido o mesmo procedimento utilizado para a metodologia 1 para a escolha das funções a serem paralelizadas.

A Tabela 16 mostra as funções selecionadas para a paralelização.

Tabela 16 – Funções paralelizadas na metodologia 2.

Função	Porcentagem de tempo (%)
SIFT	45,12
RANSAC	8,61
Total	53,73

Com as devidas funções selecionadas e paralelizadas, o código ficou com 53,73% paralelizado do seu total e com 46,27% sequencial, conseguindo minimizar o custo computacional da metodologia 2.

5.4.3. Processamento Paralelo da Metodologia 3

A Figura 86 mostra os resultados gerados pelo *Gprof* para o código da metodologia 3. A partir desses resultados, foi seguido o mesmo procedimento utilizado para a metodologia 1 para a escolha das funções a serem paralelizadas.

A Tabela 17 mostra as funções selecionadas para a paralelização.

Tabela 17 – Funções paralelizadas na metodologia 3.

Função	Porcentagem de tempo (%)
SIFT	40,98
RANSAC	21,63
Total	62,61

Com as devidas funções selecionadas e paralelizadas, o código ficou com 62,61% paralelizado do seu total e com 37,39% sequencial, conseguindo minimizar o custo computacional da metodologia 3.

Figura 85 – Diagrama gerado pelo Gprof para a metodologia 2 – parte a.

Figura 87 - Diagrama gerado pelo Gprof para a metodologia 2 – parte b.

Figura 86 – Diagrama gerado pelo Gprof para a metodologia 3 – parte a.

b

Figura 88 - Diagrama gerado pelo Gprof para a metodologia 3 – parte b.

Figura 88 - Diagrama gerado pelo Gprof para a metodologia 3 – parte c.

5.4.4. Construção de Mosaicos em Paralelo Usando MPI

O padrão de processamento paralelo MPI foi utilizado para criar mosaicos independentes em paralelo, baseado na versão sequencial da Metodologia 2. A implementação foi realizada numa máquina que contém 4 *threads* físicas e 4 *threads* virtuais (Capítulo 6). A quantidade de mosaicos que podem ser criados em paralelo fica limitada pela quantidade de processadores físicos disponíveis na arquitetura em questão. Por exemplo, se um sistema possui 4 núcleos de processamento, é possível construir até 4 mosaicos com diferentes imagens, gastando-se o tempo de construção de um mosaico sequencial. Essa implementação possibilita reduzir o tempo para a construção de vários mosaicos.

Esse mesmo padrão foi utilizado para paralelizar a construção dos mosaicos, de forma a se ter um nó responsável por cada uma das seguintes etapas do algoritmo: calcular as características SIFT, correspondência dos pontos característicos, eliminação de *outliers*, estimação da matriz homográfica, cálculo da região de sobreposição de cada imagem selecionada para ser unida, e por fim, o retorno desses cálculos em um nó mestre capaz de fazer a união das imagens sequenciais.

No entanto, neste projeto é utilizada a biblioteca de processamento de imagens OpenCV, já citada anteriormente, que faz uso de muitas variáveis, com diferentes tipos de dados (exemplo *struct*) e, ainda, com muitas funções fazendo a chamada de outras funções. Foi constatado que o OpenCV não possui o suporte adequado para trabalhar com MPI, uma vez que possui variáveis do tipo *struct* próprias, que não estão definidas de maneira conveniente para o MPI. O padrão MPI aceita somente como passagem de parâmetro, funções e variáveis simples (*int*, *char*, *double* etc) ou variáveis do tipo *struct* definidas em termos de variáveis simples. Como consequência, a comunicação entre os processadores fica limitada, impedindo que as informações dos outros processadores sejam reunidas para a construção dos mosaicos. Para solucionar este problema, seria necessário ser criada uma versão do OpenCV apropriada para MPI.

5.5. Considerações Sobre CUDA

A arquitetura CUDA foi considerada para paralelizar as etapas de construção do mosaico. No entanto verificou-se que essa arquitetura exige modificações no código para cada modelo de GPU (placa de vídeo) utilizada, sendo uma limitação deste padrão de paralelismo (HALFHILL, 2008). Como o intuito deste projeto é criar uma ferramenta de fácil uso pelo usuário final, optou-se pela não utilização dessa arquitetura.

De acordo com a disponibilidade de diversos modelos e versões de GP-GPUs, e sendo uma característica destas a execução de diversos *threads* paralelamente via hardware, é necessário o ajuste do software aos recursos disponíveis pelo tipo de GPU a ser utilizada, de modo a otimizar o tempo de execução do sistema (SILVA, 2012). Os dois principais recursos a serem especificados pelo programa são:

- o número máximo de *threads* que podem ser executados simultaneamente;
- a quantidade disponível, tipo, e endereçamento da memória disponível na GPU, otimizando a transferência de dados entre os circuitos de *threads* e as memórias.

Os *threads* em GPUs geralmente são multi-dados, executando o mesmo código em diversos *threads*, mas utilizando e produzindo dados diferentes. Os dados a serem processados podem ser vistos como elementos em matrizes de *threads*, e o processamento como operações matriciais na qual todos os elementos de dados são processados ao mesmo tempo (HALFHILL, 2008). Caso o volume de dados ultrapasse a capacidade máxima da GPU, um novo processamento deve ser efetuado, resultando no *thread swap*, que consiste na troca de dados entre *device* e *host*. Este processo demanda tempo de execução e deve ser reduzido ao máximo (SILVA, 2012). Normalmente a memória disponível na GPU é reduzida em relação ao sistema do *host*, e o *thread swap* sempre ocorre. As diversas cópias realizadas entre uma memória e outra podem causar algum impacto no desempenho geral das aplicações, mas isso varia de acordo com o barramento do sistema (HALFHILL, 2008).

5.6. Considerações Finais

Neste Capítulo foram apresentadas três metodologias sequenciais e o padrão *OpenMP* utilizado para paralelizá-las. Após a implementação dessas metodologias, as mesmas foram submetidas a testes para verificação de desempenho e qualidade das junções dos mosaicos construídos, dando subsidio para o Capítulo de resultados e discussões. Foi realizada também uma implementação em MPI para paralelização de vários mosaicos, onde cada mosaico é construído de forma sequencial.

6. Resultados e Discussões

Este Capítulo apresenta os resultados dos experimentos realizados pelas metodologias propostas, utilizando as duas bases de imagens descritas na Seção 5.2. Na Seção 6.1 são apresentados os resultados e comparações de custo computacional para a base de imagens sem redimensionamento, para as três metodologias, utilizando somente processamento sequencial. Na Seção 6.2 são apresentados os resultados de custo computacional para as metodologias paralelizadas, sendo comparados os resultados das três metodologias.

Na Seção 6.3 são apresentados os resultados da avaliação da distorção para cada mosaico gerado, utilizando a base de imagens redimensionada. Os softwares *Autostitch* e *PTGui* foram utilizados para gerar esses mesmos mosaicos, com o objetivo de avaliar a distorção por eles gerada e comparar com a distorção da metodologia 1.

Na seção 6.4 são apresentados os resultados da avaliação da distorção para os cinco primeiros pares de imagens do grupo 7 da base de imagens redimensionada. A avaliação foi realizada da mesma maneira que na seção 6.3, utilizando a metodologia 3.

Para todos os resultados gerados foi utilizado um microcomputador com as seguintes configurações: *notebook Dell Vostro 3560 (i7 3612QM de 3a geração - 2.1GHz – 8 threads (4 threads físicas e 4 threads virtuais) – 6MB de Cache e 8GB*

de RAM DDR3 1600MHz) e todos os valores de tempos são apresentados em segundos.

Para avaliar as metodologias 1, 2 e 3 sequencial e paralela, foram utilizados os 10 grupos de imagens sem redimensionamento, constituídos por 10 imagens distintas cada um, que podem ser observados no Apêndice A. Os mosaicos gerados pela metodologia 1 são apresentados no Apêndice B.

6.1. Resultados do Processamento Sequencial

A Tabela 18 apresenta os tempos de processamento sequencial para gerar cada um dos mosaicos utilizando as metodologias 1, 2 e 3.

Tabela 18 – Tempo de processamento sequencial das metodologias 1,2 e 3.

	Metodologia 1 (s)	Metodologia 2 (s)	Metodologia 3 (s)
Grupo1	988	843	294
Grupo2	772	653	214
Grupo3	724	598	227
Grupo4	828	673	345
Grupo5	445	383	162
Grupo6	425	378	312
Grupo7	622	461	193
Grupo8	469	395	177
Grupo9	329	224	135
Grupo10	466	359	155
Média	606,8	496,7	221,4

De acordo com a média dos tempos apresentados na Tabela 18, é possível observar que a metodologia 1 é 18,15% (1,2x) mais lenta que a metodologia 2 e 63,5% (2,74x) mais lenta que a metodologia 3. O fato da construção dos mosaicos ser mais lenta para as metodologias 1 e 2, deve-se à utilização de toda a área da Imagem2 para se calcular os parâmetros necessários à união com o mosaico parcial.

A diferença entre as metodologias 1 e 2, deve-se ao fato da metodologia 2 calcular todos os parâmetros de todas as imagens para então construir o mosaico, e com isso ocorre redução de acesso a memória e redução da quantidade de chamadas feitas para as principais funções do código (Transformada SIFT, BBF, RANSAC). No entanto, a metodologia 2 faz uso de toda área da Imagem2 para o

cálculo dos parâmetros e por isso não teve uma redução maior de custo computacional.

A metodologia 3 somente calcula os parâmetros na região de sobreposição (RS), que foi estimada baseada nas imagens de baixa resolução, otimizando o custo computacional. Devido a isso ela tem uma redução de tempo em 2,74x em relação à metodologia 1. Essa metodologia representa uma inovação em relação à metodologia convencional, a qual é implementada neste trabalho como metodologia 1. A metodologia 3 mostrou ser a melhor entre as três metodologias no que se refere ao tempo de processamento.

Observa-se também na Tabela 18 que existe diferença de tempo para a construção de mosaico para cada grupo de imagens. Essa diferença é justificada pela quantidade de características extraídas de cada uma das 10 imagens que compõem o respectivo grupo e, isso se deve à quantidade de detalhes pertencentes a cada uma das imagens. A Tabela 19 apresenta a quantidade de características extraídas pela Transformada SIFT para cada um dos 10 grupos de imagens do Apêndice A (base de imagens sem redimensionamento).

Essa comparação também é apresentada na Tabela 19 para a base de imagens redimensionadas. É possível observar entre a base sem redimensionamento e a redimensionada, que há uma diferença média de 11,9 vezes maior de quantidade de características.

Tabela 19 – Quantidade de características extraídas.

	Base Redimensionada	Base sem redimensionamento
Grupo1	12390	144259
Grupo2	8456	116594
Grupo3	7890	86520
Grupo4	9385	119378
Grupo5	5761	68824
Grupo6	5623	60740
Grupo7	6878	83285
Grupo8	6052	71316
Grupo9	4993	53393
Grupo10	5945	70976
Média	7337,3	87528,5

6.2. Resultados do Processamento Paralelo

Para verificar o desempenho das três metodologias paralelas propostas neste projeto, foram realizados testes comparativos dos tempos de execução do código sequencial ($n=1$) e paralelo para gerar cada um dos mosaicos. A análise do tempo de execução dos códigos paralelo foi realizada utilizando $n=2$, $n=4$ e $n=8$ *threads* e os resultados são apresentados a seguir. Até $n=4$ *threads* são utilizados núcleos físicos e acima inclui os quatro núcleos virtuais.

Um detalhe a ser considerado ao longo das paralelizações deste trabalho, é que, muitas vezes, aumentando-se a porcentagem de código paralelizável não houve benefícios de desempenho, devido a fatores como:

- O número de chamadas de algumas funções é da ordem de milhões de chamadas, fazendo com que a paralelização destas funções não gere muita eficiência, porque o tempo necessário para criar e fechar as *threads* (Figura 87) aumenta linearmente.
- Os índices do laço '*for*' mais usados neste trabalho variam somente de 1 até 3. Os cálculos efetuados dentro destes laços muitas vezes são simples, fazendo com que a paralelização tenha pouco benefício.

Figura 87 – Tempo para criação e fechamento de *threads*.

6.2.1.Paralelismo na Metodologia 1

Na Tabela 20 é possível observar os tempos de execução para cada grupo de imagens utilizando a metodologia 1 para valores de *threads* iguais a $n=1$, $n=2$, $n=4$ e $n=8$. Também é possível observar o tempo médio de execução para cada valor de n .

Tabela 20 – Tempo de execução com a metodologia 1.

	n=1 (s)	n=2 (s)	n=4 (s)	n=8 (s)
grupo1	988	655	456	421
grupo2	772	498	342	301
grupo3	724	498	324	289
grupo4	828	567	367	293
grupo5	445	281	203	168
grupo6	425	302	187	165
grupo7	622	410	288	221
grupo8	469	323	230	213
grupo9	329	259	179	177
grupo10	466	305	234	202
Média	606,8	409,8	281	245

Os resultados apresentados evidenciam uma alteração significativa no tempo de execução à medida que se aumenta a quantidade de *threads*. Comparando-se os resultados da Tabela 20 para $n=1$ e $n=4$, o tempo se reduz quase pela metade (54% ou 2,15x). No entanto, nota-se que quando se altera a quantidade de *threads* de $n=4$ para $n=8$, a redução do tempo de processamento é de aproximadamente 13% (1,14x). Isso se deve ao fato de estar utilizando quatro núcleos lógicos para $n=8$, e além disso, como já mencionado no início da Seção 6.2, ao fato que o número de chamadas de algumas funções é da ordem de milhões de chamadas e assim, o tempo necessário para abrir e fechar as *threads* tende ao tempo de execução sequencial (Figura 87). Outro fator que contribui para este resultado são os índices dos ciclos ‘for’ mais usados que variam no máximo até 3.

De acordo com a Tabela 20 e com os conceitos de métricas de desempenho da Seção 3.6, foi possível calcular o *Speedup* para cada grupo, que mede o ganho das implementações paralelizáveis sobre a implementação sequencial. A Tabela 21 apresenta os *Speedup* gerados.

De acordo com a Seção 5.4.1, a parcela do código paralelizado é de 74,75% e a do código sequencial é de 25,25%. De acordo com os resultados da Tabela 21,

determinados pela lei de Amdahl (Seção 3.6), é possível observar que a metodologia proposta atingiu um nível de desempenho diferente para cada grupo de imagens, que se deve às quantidades diferentes de características extraídas pelo algoritmo SIFT.

Tabela 21 – *Speedup* da metodologia 1.

	<i>Speedup</i> n=8	<i>Speedup</i> n=4	<i>Speedup</i> n=2
grupo1	2,35	2,17	1,51
grupo2	2,56	2,26	1,55
grupo3	2,51	2,23	1,45
grupo4	2,83	2,26	1,46
grupo5	2,65	2,19	1,58
grupo6	2,58	2,27	1,41
grupo7	2,81	2,16	1,52
grupo8	2,20	2,04	1,45
grupo9	1,86	1,84	1,27
grupo10	2,31	1,99	1,53
média	2,46	2,14	1,47

Comparando-se os valores de *Speedup* da Tabela 21 com os valores de *Speedup* máximo da Tabela 22, observa-se que nenhum valor da Tabela 21 ultrapassou os valores da Tabela 22, calculados pela lei de Amdahl. Pela avaliação do *Speedup médio*, pode-se concluir que à medida que *n* aumenta, a diferença entre os *Speedup* tende a diminuir, levando-se a concluir que o aumento do *Speedup* quando se aumenta o número de *threads* de *n*=4 para *n*=8 é pouco significativa.

Tabela 22 – *Speedup* máximo (teórico).

n=8	n=4	n=2
2,89	2,28	1,60

Da Figura 88 a Figura 97 são mostrados os gráficos de *Speedup*, separados por grupo, dos dados comparativos da Tabela 21 e Tabela 22.

Figura 88 – *Speedup* máximo e *Speedup* do grupo 1 da metodologia 1.

Figura 89 – *Speedup* máximo e *Speedup* do grupo 2 da metodologia 1.

Figura 90 – *Speedup* máximo e *Speedup* do grupo 3 da metodologia 1.

Figura 91 – *Speedup* máximo e *Speedup* do grupo 4 da metodologia 1.

Figura 92 – *Speedup* máximo e *Speedup* do grupo 5 da metodologia 1.

Figura 93 – *Speedup* máximo e *Speedup* do grupo 6 da metodologia 1.

Figura 94 – *Speedup* máximo e *Speedup* do grupo 7 da metodologia 1.

Figura 95 – *Speedup* máximo e *Speedup* do grupo 8 da metodologia 1.

Figura 96 – *Speedup* máximo e *Speedup* do grupo 9 da metodologia 1.

Figura 97 – *Speedup* máximo e *Speedup* do grupo 10 da metodologia 1.

6.2.2. Paralelismo na Metodologia 2

As considerações feitas no início da Seção 6.2 também são utilizadas para analisar o paralelismo da metodologia 2.

Na Tabela 23 são mostrados os tempos de execução para cada grupo de imagens. Observa-se que à medida que o valor de n é aumentado, o tempo de

execução para construção dos mosaicos não é reduzido de forma significativa, evidenciando que o paralelismo é pouco eficiente para esta metodologia.

Tabela 23 – Tempo de execução da metodologia 2.

	n=1 (s)	n=2 (s)	n=4 (s)	n=8 (s)
grupo1	843	821	802	791
grupo2	653	620	598	574
grupo3	598	589	577	561
grupo4	673	651	638	625
grupo5	383	376	362	353
grupo6	378	364	355	348
grupo7	461	452	444	436
grupo8	395	383	375	363
grupo9	224	217	208	199
grupo10	359	347	334	321
média	496,7	482	469,3	457,1

Quando se usa $n=4$ threads, o tempo médio é 5,5% (1,05x) menor, comparado com o processamento sequencial ($n=1$); e quando se usa $n=8$ threads o tempo médio é 8% (1,08x) menor, não havendo, portanto, uma redução de tempo significativa, comparado com $n=1$ threads.

De acordo com a Tabela 23 e com os conceitos de métricas de desempenho da Seção 3.6, foi possível calcular o *Speedup* para cada grupo, que mede o ganho das implementações paralelizáveis sobre a implementação sequencial. A Tabela 24 apresenta os *Speedup* gerados.

Tabela 24 – *Speedup* da metodologia 2.

	<i>Speedup</i> n=8	<i>Speedup</i> n=4	<i>Speedup</i> n=2
grupo1	1,07	1,05	1,03
grupo2	1,14	1,09	1,05
grupo3	1,07	1,04	1,02
grupo4	1,08	1,05	1,03
grupo5	1,08	1,06	1,02
grupo6	1,09	1,06	1,04
grupo7	1,06	1,04	1,02
grupo8	1,09	1,05	1,03
grupo9	1,13	1,08	1,03
grupo10	1,12	1,07	1,03
média	1,09	1,06	1,03

De acordo com a Seção 5.4.1, a parcela do código paralelizado é de 53,73% e a do código sequencial é de 46,27%. Com os resultados da Tabela 24, determinados pela lei de Amdahl (Seção 3.6), é possível observar que os valores de *Speedup* são similares para os três valores de *n*, enfatizando não haver benefício no paralelismo aplicado a esta metodologia. A justificativa será apresentada na Seção 6.2.4. A Tabela 25 mostra os valores de *Speedup* máximo, podendo-se constatar que há diferença considerável entre os *Speedup* médio e os *Speedup* máximo e que nenhum valor da Tabela 24 ultrapassou os valores de *speedup* máximo calculados pela lei de Amdahl.

Tabela 25 – *Speedup* máximo (teórico).

n=8	n=4	n=2
1,89	1,67	1,37

Da Figura 98 à Figura 107 são mostrados os gráficos de *Speedup*, separados por grupo, dos dados comparativos da Tabela 24 e Tabela 25, enfatizando o pouco benefício do paralelismo para esta metodologia.

Figura 98 – *Speedup* máximo e *Speedup* do grupo 1 da metodologia 2.

Figura 99 – *Speedup* máximo e *Speedup* do grupo 2 da metodologia 2.

Figura 100 – *Speedup* máximo e *Speedup* do grupo 3 da metodologia 2.

Figura 101 – *Speedup* máximo e *Speedup* do grupo 4 da metodologia 2.

Figura 102 – *Speedup* máximo e *Speedup* do grupo 5 da metodologia 2.

Figura 103 – *Speedup* máximo e *Speedup* do grupo 6 da metodologia 2.

Figura 104 – *Speedup* máximo e *Speedup* do grupo 7 da metodologia 2.

Figura 105 – *Speedup* máximo e *Speedup* do grupo 8 da metodologia 2.

Figura 106 – *Speedup* máximo e *Speedup* do grupo 9 da metodologia 2.

Figura 107 – *Speedup* máximo e *Speedup* do grupo 10 da metodologia 2.

Pelos resultados obtidos pela metodologia 1 e 2, de acordo com a Tabela 21 e Tabela 24, fazendo-se a comparação para $n=8$, é possível verificar que houve um aumento de *Speedup* de apenas 1,09 vezes para a metodologia 2 e de 2,46 vezes para a metodologia 1.

Esta metodologia tem a vantagem de abrir menos as *threads*, porém é limitada pela quantidade de imagens a ser unida e pela memória disponível no computador em uso.

Outra consideração a ser feita sobre esta metodologia é que ela acumula informação na memória, uma vez que primeiramente realiza todos os cálculos para que posteriormente haja a junção par a par das imagens. Esse processo desencadeia lentidão na junção das imagens à medida que aumenta a quantidade de imagens que já foram unidas.

6.2.3. Paralelismo na Metodologia 3

As especificações apresentadas na Seção 6.2 também foram aplicadas para testar o paralelismo da metodologia 3.

Na Tabela 26 são mostrados os tempos de execução para cada grupo de imagens. Pelos resultados apresentados, fica evidente que o paralelismo não foi

eficiente para reduzir o tempo de execução, da mesma forma que para a metodologia 2. E assim, as mesmas considerações realizadas para a metodologia 2 aplicam-se para a metodologia 3. A diferença a ser ressaltada é que o tempo de execução sequencial para a metodologia 3 é muito menor do que o da metodologia 2 e 1, já discutidos na Seção 6.1.

Tabela 26 – Tempo de execução da metodologia 3.

	n=1 (s)	n=2 (s)	n=4 (s)	n=8 (s)
grupo1	294	288	281	274
grupo2	214	207	201	198
grupo3	227	222	214	211
grupo4	345	338	332	327
grupo5	162	158	155	149
grupo6	312	304	291	286
grupo7	193	187	180	173
grupo8	177	171	165	159
grupo9	135	131	122	117
grupo10	155	149	141	137
média	221,4	215,5	208,2	203,1

Quando se usa $n=4$ *threads*, o tempo médio é 6% (1,06x) menor, comparado com o processamento sequencial ($n=1$); e quando se usa $n=8$ *threads* o tempo médio é 8,5% menor (1,09x), não havendo, portanto, uma redução de tempo significativa, comparado com $n=1$ *threads*.

De acordo com a Tabela 26 e com os conceitos de métricas de desempenho da Seção 3.6, foi possível calcular o *Speedup* para cada grupo, que mede o ganho das implementações paralelizáveis sobre a implementação sequencial. A Tabela 27 apresenta os *Speedup* gerados.

De acordo com a Seção 5.4.1, a parcela do código paralelizado é de 62,61% e a do código sequencial é de 37,39%. Com os resultados da Tabela 27, determinados pela lei de Amdahl (Seção 3.6), é possível observar que os valores de *Speedup* são similares para os três valores de n , enfatizando não haver benefício no paralelismo aplicado a esta metodologia. A justificativa será apresentada na Seção 6.2.4. A Tabela 28 mostra os valores de *Speedup* máximo, podendo-se constatar que nenhum valor da Tabela 27 ultrapassou os valores de *Speedup* máximo calculados pela lei de Amdahl.

Tabela 27 – *Speedup* da metodologia 3.

	<i>Speedup</i> n=8	<i>Speedup</i> n=4	<i>Speedup</i> n=2
grupo1	1,07	1,05	1,02
grupo2	1,08	1,06	1,03
grupo3	1,08	1,06	1,02
grupo4	1,06	1,04	1,02
grupo5	1,09	1,05	1,03
grupo6	1,09	1,07	1,03
grupo7	1,12	1,07	1,03
grupo8	1,11	1,07	1,04
grupo9	1,15	1,11	1,03
grupo10	1,13	1,10	1,04
média	1,10	1,07	1,03

Tabela 28 – *Speedup* máximo (teórico).

n=8	n=4	n=2
2,21	1,89	1,46

Da Figura 108 a Figura 117 são mostrados os gráficos de *Speedup*, separados por grupo, dos dados comparativos da Tabela 27 e Tabela 28, enfatizando o pouco benefício do paralelismo para esta metodologia.

Figura 108 – *Speedup* máximo e *Speedup* do grupo 1 da metodologia 3.

Figura 109 – *Speedup* máximo e *Speedup* do grupo 2 da metodologia 3.

Figura 110 – *Speedup* máximo e *Speedup* do grupo 3 da metodologia 3.

Figura 111 – *Speedup* máximo e *Speedup* do grupo 4 da metodologia 3.

Figura 112 – *Speedup* máximo e *Speedup* do grupo 5 da metodologia 3.

Figura 113 – *Speedup* máximo e *Speedup* do grupo 6 da metodologia 3.

Figura 114 – *Speedup* máximo e *Speedup* do grupo 7 da metodologia 3.

Figura 115 – *Speedup* máximo e *Speedup* do grupo 8 da metodologia 3.

Figura 116 – *Speedup* máximo e *Speedup* do grupo 9 da metodologia 3.

Figura 117 – *Speedup* máximo e *Speedup* do grupo 10 da metodologia 3.

Pelos resultados obtidos pela metodologia 2 e 3, de acordo com a Tabela 24 e Tabela 27, fazendo-se a comparação para $n=8$, é possível verificar que houve um aumento de *Speedup* de apenas 1,09 vezes para a metodologia 2 e de 1,10 vezes para a metodologia 3.

É interessante destacar também que o tempo médio de execução sequencial da metodologia 3 (221s) é ainda menor do que a metodologia 1 paralela para $n=8$ (245s), comprovando que a metodologia 3 destaca-se como a melhor metodologia entre as três implementadas neste trabalho de doutorado apesar do paralelismo não ser eficiente para esta metodologia.

6.2.4. Considerações Sobre o Paralelismo das Metodologias 2 e 3

Como evidenciado pelos resultados das metodologias 2 e 3, a paralelização não obteve resultados satisfatórios. As funções paralelizadas foram: SIFT e RANSAC, resultando em uma média de 58,17% de paralelização e, mesmo assim, a redução do tempo para a construção de mosaicos não chegou à 10%.

Para avaliar estes resultados, foram realizados testes nos códigos das metodologias 2 e 3, com o auxílio do *Gprof* e da função *gettimeofday()* do C++, para medir o tempo (em s) gasto por cada uma das funções citadas anteriormente.

A seguir seguem as considerações obtidas a partir dessa análise, que justificam os resultados obtidos.

Inicialmente foi considerado o paralelismo dentro do código principal, com as rotinas sendo chamadas dentro de um *loop*, como no exemplo a seguir:

```
for( i=0;i<N;i++){
    SIFT(ImagensAux[0]);
    SIFT(ImagensAux[1]);
    RANSAC(ImagensAux[0],ImagensAux[1],escala);
    UnirlImagens(Imagens[0], Imagens[1],Blend,TMeioImagem);
}
```

A paralelização dentro do código principal fica limitada por certas particularidades do código sequencial. A cada iteração, as variáveis 'ImagensAux[0]' e 'ImagensAux[1]' contêm somente as partes da imagem que são necessárias para criar o mosaico. Se esse *loop* fosse paralelizado, a variável 'ImagensAux' deveria ser compartilhada (*shared*) por requerimento do *OpenMP*. No código sequencial, a variável 'ImagensAux' é definida como uma estrutura de dados composta por vários tipos de variáveis (*struct*). Neste caso, se uma das *threads* usasse a função SIFT em alguns elementos da variável 'ImagensAux', as outras *threads* precisariam esperar para fazer a sincronização e, por esta razão, a paralelização não pôde ser implementada no código principal. Além disso, a variável 'ImagensAux' contém somente dois elementos(0,1), o que limita a paralelização a duas *threads*.

Por esta razão, a paralelização é mais eficiente e foi implementada dentro das funções de maior custo computacional (SIFT, RANSAC). De acordo com os testes realizados e, utilizando a função *gettimeofday()*, foi possível fazer algumas considerações, como segue:

- O tempo gasto pelas funções RANSAC e SIFT permanece próximo de 4 e 9 segundos (Tabela 29) por iteração, respectivamente, e esse tempo não varia mesmo com um aumento de esforço computacional notável (aumento da quantidade de imagens), comparado com o tempo total de execução. Isto acontece devido à minimização de trabalho exigido pela metodologia 3, que realiza os cálculos dos parâmetros somente em uma pequena região de sobreposição, fazendo com que cada *thread* fique com pouco trabalho a ser realizado.
- Mesmo aumentando a quantidade de imagens, o esforço computacional, por iteração, só é crescente na função 'UnirlImagens'. Porém esta

função pertence ao código principal e, como descrito anteriormente, não cabe paralelização para esta função.

A Tabela 29 apresenta os tempos gastos (s) pelas funções SIFT, RANSAC e 'Unirlmagens', de acordo com o aumento das iterações, para mosaicos constituídos por 8, 10 e 12 imagens respectivamente.

Tabela 29 – Tempo das funções, de acordo com o aumento das iterações.

8 imagens				
Qtde. de iterações	SIFT (s)	RANSAC (s)	Unirlmagens (s)	Total (s)
2	9	4	6	19
3	8	4	7	19
4	8	3	8	19
5	8	4	10	22
6	8	5	12	25
7	8	4	12	24
total	49	24	55	128
10 imagens				
Qtde. de iterações	SIFT (s)	RANSAC (s)	Unirlmagens (s)	Total (s)
2	9	5	6	20
3	9	4	6	19
4	8	3	8	19
5	7	4	10	21
6	8	5	11	24
7	9	3	12	24
8	10	4	14	28
9	8	9	16	33
total	68	37	83	188
12 imagens				
Qtde. de iterações	SIFT (s)	RANSAC (s)	Unirlmagens (s)	Total (s)
2	10	4	6	20
3	8	3	6	17
4	7	4	8	19
5	8	3	9	20
6	9	5	11	25
7	9	3	12	24
8	10	4	14	28
9	9	8	16	33
10	8	6	18	32
11	9	2	20	31
Total	87	42	120	249

Na SIFT, as funções internas de maior custo são '*compute_descriptors*' e '*scale_space_extrema*'. Essas funções possuem vários *loops*, inclusive com alguns *loops* aninhados, facilitando a paralelização. Mesmo assim, o tempo de execução não foi reduzido para um valor significativo. A principal razão é que o tempo da SIFT usado por iteração é baixo (10 segundos no máximo), e consequentemente as

funções *'compute_descriptors'* e *'scale_space_extrema'* gastam um tempo de execução ainda menor, o que diminui o benefício da paralelização, podendo ocorrer acréscimo do tempo de execução devido ao tempo necessário para criar e sincronizar as *threads*.

A função interna de maior custo dentro do RANSAC é *'ransac_xform'*, cujas características dificultam a paralelização, por ter *loops* do tipo *while* e condicionais *if-else*. Neste caso, foram paralelizadas partes dela, utilizando as funções *sections* e *single* do *OpenMP*. Novamente não foi obtido um ganho em tempo, e às vezes acréscimo do tempo de execução pelas mesmas razões explicadas no SIFT. Além disso, o tempo de execução dessa função, por iteração, é pequeno (9 segundos no máximo).

Concluindo, a única função que permitiria obter ganho considerável com paralelismo seria a *'Unirlmagens'*, porque o seu tempo de execução por iteração aumenta com a quantidade de imagens. No entanto, esta função pertence ao código principal e a sua estrutura interna contém várias condicionais *if-else*, impossibilitando a sua paralelização.

6.2.5. Resultados Utilizando MPI

A estrutura de construção de mosaicos em paralelo com MPI utilizou a metodologia 2. Com a quantidade de *threads* limitada pelas características do computador utilizado neste trabalho, foi possível construir até quatro mosaicos (quantidade máxima de *threads* físicas) no mesmo tempo de construção de um mosaico sequencial. A Tabela 30 apresenta os resultados para construir 10 mosaicos, utilizando duas *threads*. Nesta tabela é mostrado o maior tempo de execução para cada duas *threads*.

Tabela 30 – Tempo de execução em MPI da metodologia 2.

Grupo 1/2	843
Grupo 3/4	673
Grupo 5/6	383
Grupo 7/8	461
Grupo 9/10	359
média	272

6.3. Resultados das Distorções dos Mosaicos

Como já mencionado na Seção 5.3.1, os mosaicos gerados apresentam distorções nas junções que são ocasionados pelo processo de construção do mosaico e nesta Seção são avaliadas as distorções geradas pela metodologia 1. Para a avaliação foi utilizado o banco de imagens redimensionadas (Apêndice C) para gerar os mosaicos (Apêndice D).

Na avaliação, cada mosaico gerado foi comparado com a sua mesma localização no *Google Earth*, utilizado como referência. Além disso, os grupos de imagens do Apêndice C também foram submetidos a dois programas de licença gratuita (*PTGui Trial* e *Autostitch Demo*) para construção dos respectivos mosaicos e ambos também tiveram os seus resultados avaliados no aspecto de distorção. Esses softwares possuem parâmetros pré-definidos para imagens com a resolução de 533 x 800 pixels. O Autostitch foi desenvolvido por Lowe (2004), que foi quem desenvolveu a Transformada SIFT.

Foram avaliadas as distorções nos 10 mosaicos gerados, correspondentes a cada grupo de imagens. Por exemplo, para o grupo 10 de imagens foi gerado um mosaico com a metodologia 1, um mosaico com o *PTGui* e um mosaico com o *Autostitch* (Figura 118), e feitas três marcações facilmente identificáveis em cada um dos mosaicos. No *Google Earth*, na mesma região dos mosaicos gerados anteriormente, foram feitas as mesmas marcações.

As três marcações realizadas nos mosaicos são denominadas de A, B e C de acordo com a Figura 119 e formam um triângulo. Dessa forma podem ser calculados os segmentos de reta que se formam entre os vértices (AB, AC, BC) e os valores dos ângulos formados entre cada um desses segmentos (α , β e γ). Os segmentos de reta são calculados a partir da equação (40) e os ângulos a partir das equações (41) e (39).

Com os valores dos segmentos de reta é possível calcular a proporção entre cada um deles (AB/BC, AB/AC e BC/AC) e fazer uma comparação com os valores do Google Earth, determinando assim qual programa distorce menos as imagens para construir um mosaico.

Google Earth

Metodologia Desenvolvida

Autostitch Demo

PTGui Trial

Figura 118 – Mosaicos gerados em comparação com o Google Earth.

Figura 119 – Trigonometria de triângulos.

$$D = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} \quad (40)$$

$$m = \frac{(y_1 - y_0)}{(x_1 - x_0)} \quad (41)$$

$$\theta = \tan^{-1} \left| \frac{m_1 - m_0}{1 + m_1 \cdot m_0} \right| \quad (42)$$

Todos os mosaicos gerados pelos três programas mais o *Google Earth*, separados por grupos, podem ser visualizados no Apêndice E.

Os valores das marcações feitas no *Google Earth* são baseados em coordenadas GPS de latitude e longitude (GPS - *Global Positioning System*). As coordenadas GPS foram convertidas para o padrão UTM (*Universal Transverse Mercator*) (SILVA, 2012a), para que os cálculos pudessem ser realizados e comparados com os das outras metodologias.

Os valores das marcações feitas no programa da metodologia 1, no *Autostitch* e no *PTGui* são baseadas nas coordenadas da própria imagem (x,y). No Apêndice F, da Tabela F1 à Tabela F10, são apresentados os valores das coordenadas A, B e C para cada um dos 10 grupos de imagens e para cada metodologia empregada.

A partir das coordenadas obtidas são calculados os ângulos e os respectivos segmentos de reta AB, BC e AC, formados pelos vértices. Esses valores encontram-se no Apêndice G (Tabelas G1 à G10).

Com os resultados das Tabelas G1 à G10 é possível verificar que as somas de β , α e γ sempre dão 180 graus, comprovando a validade dos resultados. Com os valores dos segmentos de reta são cálculos as proporções, os quais são apresentados da Tabela 31 à Tabela 40.

Para determinar a menor distorção entre as metodologias avaliadas é calculado o Erro Quadrático Médio (EQM) em relação ao *Google Earth*, identificando qual metodologia apresenta menor valor de EQM, e portanto menor distorção. Da Tabela 31 à Tabela 40, os menores valores de EQM são destacados em negrito.

Tabela 31 – Cálculo das proporções para o grupo 1.

Proporção	Metodologia 1	<i>Autostitch</i>	<i>PTGui</i>	<i>Google Earth</i> (Referência)
AB/BC	0,976	1,170	0,959	1,025
AB/AC	0,501	0,545	0,495	0,513
BC/AC	0,513	0,466	0,516	0,500
EQM	0,052	0,153	0,070	0,000

Tabela 32 – Cálculo das proporções para o grupo 2.

Proporção	Metodologia 1	<i>Autostitch</i>	<i>PTGui</i>	<i>Google Earth</i> (Referência)
AB/BC	0,721	0,756	0,765	0,435
AB/AC	0,422	0,434	0,450	0,307
BC/AC	0,586	0,574	0,588	0,705
EQM	0,331	0,369	0,379	0,000

Tabela 33 – Cálculo das proporções para o grupo 3.

Proporção	Metodologia 1	<i>Autostitch</i>	<i>PTGui</i>	<i>Google Earth</i> (Referência)
AB/BC	0,930	1,003	0,907	0,941
AB/AC	0,484	0,503	0,476	0,500
BC/AC	0,521	0,501	0,525	0,531
EQM	0,022	0,069	0,043	0,000

Tabela 34 – Cálculo das proporções para o grupo 4.

Proporção	Metodologia 1	<i>Autostitch</i>	<i>PTGui</i>	<i>Google Earth</i> (Referência)
AB/BC	0,893	0,906	0,861	1,163
AB/AC	0,476	0,481	0,479	0,545
BC/AC	0,533	0,531	0,557	0,469
EQM	0,287	0,272	0,322	0,000

Tabela 35 – Cálculo das proporções para o grupo 5.

Proporção	Metodologia 1	<i>Autostitch</i>	<i>PTGui</i>	<i>Google Earth</i> (Referência)
AB/BC	1,236	1,314	1,099	1,187
AB/AC	0,555	0,576	0,527	0,556
BC/AC	0,449	0,438	0,479	0,468
EQM	0,053	0,132	0,093	0,000

Tabela 36 – Cálculo das proporções para o grupo 6.

Proporção	Metodologia 1	<i>Autostitch</i>	<i>PTGui</i>	<i>Google Earth</i> (Referência)
AB/BC	0,971	0,766	0,689	1,040
AB/AC	0,498	0,439	0,415	0,514
BC/AC	0,512	0,573	0,603	0,494
EQM	0,073	0,294	0,380	0,000

Tabela 37 – Cálculo das proporções para o grupo 7.

Proporção	Metodologia 1	<i>Autostitch</i>	<i>PTGui</i>	<i>Google Earth</i> (Referência)
AB/BC	0,800	0,825	0,806	0,778
AB/AC	0,452	0,463	0,460	0,451
BC/AC	0,565	0,562	0,570	0,579
EQM	0,026	0,052	0,031	0,000

Tabela 38 – Cálculo das proporções para o grupo 8.

Proporção	Metodologia 1	<i>Autostitch</i>	<i>PTGui</i>	<i>Google Earth</i> (Referência)
AB/BC	0,853	0,941	1,204	0,891
AB/AC	0,474	0,495	0,570	0,483
BC/AC	0,555	0,526	0,474	0,542
EQM	0,041	0,054	0,332	0,000

Tabela 39 – Cálculo das proporções para o grupo 9.

Proporção	Metodologia 1	<i>Autostitch</i>	<i>PTGui</i>	<i>Google Earth</i> (Referência)
AB/BC	0,928	0,949	1,115	1,011
AB/AC	0,489	0,493	0,532	0,507
BC/AC	0,527	0,519	0,477	0,502
EQM	0,088	0,066	0,110	0,000

Tabela 40 – Cálculo das proporções para o grupo 10.

Proporção	Metodologia 1	AutoStitch	PTGUI	<i>Google Earth</i> (Referência)
AB/BC	0,680	0,704	0,655	0,747
AB/AC	0,409	0,420	0,399	0,431
BC/AC	0,601	0,597	0,609	0,578
EQM	0,074	0,048	0,103	0,000

Com os resultados de EQM destacados, pode-se concluir que a metodologia 1 apresenta os menores valores (menor distorção) para 7 grupos e o *Autostitch* para 3 grupos. Além disso, observa-se que o *PTGui* distorce mais as imagens em relação as duas outras metodologias, provavelmente por não tratar questões de retificação e distorção das imagens de forma adequada.

Os resultados evidenciam que de uma maneira geral a metodologia 1 apresentou os melhores resultados quando comparados com os dos softwares *Autostitch* e *PTGui*. O *Autostitch* de uma maneira geral é um programa que gera melhores resultados do que o *PTGui* no que se refere à distorção.

6.4. Resultados das Distorções por Pares de Imagens

Os cinco primeiros pares sequenciais do grupo 7 de imagens, da base de imagens redimensionadas foram testados para distorção, para avaliação da área da junção, que é a área mais crítica do mosaico. Foram avaliados os resultados de distorção da metodologia 3 e comparados com os do *PTGui* e *Autostitch*.

No Apêndice H podem ser visualizados todos os pares de imagens gerados pelos três programas mais o *Google Earth*.

Os valores das marcações feitas nos mosaicos gerados pela metodologia 3, no *Autostitch* e no *PTGui* são baseados nas coordenadas da própria imagem (x,y). No Apêndice I, da Tabela I1 à Tabela I5, são apresentados os valores das coordenadas A, B e C para cada um dos 5 pares de imagens e para cada metodologia utilizada.

A partir das coordenadas obtidas são calculados os ângulos e os respectivos segmentos de reta AB, BC e AC, formados pelos vértices. Esses valores encontram-se no Apêndice J (Tabelas J1 à J5).

Com os resultados das Tabelas J1 à J5 é possível verificar que as somas de β , α e γ sempre dão 180 graus, comprovando a validade dos resultados. Com os valores dos segmentos de reta são calculadas as proporções, as quais são apresentadas nas Tabela 41 à Tabela 45.

Tabela 41 – Cálculo das proporções para o par 1.

Proporção	Metodologia 3	AutoStitch	PTGUI	Google Earth (Referência)
AB/BC	1,028	1,019	1,020	1,326
AB/AC	0,558	0,556	0,561	0,648
BC/AC	0,543	0,546	0,550	0,488
EQM	0,316	0,326	0,324	0,000

Tabela 42 – Cálculo das proporções para o par 2.

Proporção	Metodologia 3	AutoStitch	PTGUI	Google Earth (Referência)
AB/BC	0,968	0,978	0,982	0,776
AB/AC	0,691	0,697	0,692	0,531
BC/AC	0,713	0,713	0,705	0,684
EQM	0,252	0,263	0,262	0,000

Tabela 43 – Cálculo das proporções para o par 3.

Proporção	Metodologia 3	AutoStitch	PTGUI	Google Earth (Referência)
AB/BC	1,789	1,794	1,791	1,795
AB/AC	0,937	0,916	0,955	0,944
BC/AC	0,524	0,511	0,533	0,526
EQM	0,009	0,032	0,014	0,000

Tabela 44 – Cálculo das proporções para o par 4.

Proporção	Metodologia 3	AutoStitch	PTGUI	Google Earth (Referência)
AB/BC	1,388	1,387	1,400	1,330
AB/AC	0,720	0,707	0,732	0,715
BC/AC	0,518	0,510	0,523	0,538
EQM	0,062	0,065	0,074	0,000

Tabela 45 – Cálculo das proporções para o par 5.

Proporção	Metodologia 3	AutoStitch	PTGUI	Google Earth (Referência)
AB/BC	0,957	0,980	0,983	0,894
AB/AC	0,514	0,523	0,520	3,009
BC/AC	0,537	0,534	0,529	3,366
EQM	3,772	3,769	3,775	0,000

Com os resultados de EQM destacados, pode-se concluir que a metodologia 3 apresenta os menores valores (menor distorção) para 4 pares e o *Autostitch* para 1 par.

Para os testes de distorção realizados com a metodologia 3, os resultados evidenciam que de uma maneira geral a metodologia 3 apresentou os melhores resultados, que se deve aos algoritmos aplicados para minimizar a distorção dos mosaicos.

7. Conclusões

Este trabalho apresentou a proposta de uma metodologia para construir mosaicos automáticos de imagens aéreas agrícolas sequenciais utilizando processamento paralelo, a partir de uma base de imagens obtidas na região de Santa Rita do Sapucaí – MG. As imagens foram obtidas a partir de uma câmera de alta resolução acoplada a um suporte externo de um avião. As imagens contêm áreas de pastagens, matas e área urbana. Para que o mosaico possa ser construído, há necessidade de haver uma área de sobreposição entre as imagens sequenciais e devido a isso, algumas normas foram estabelecidas para a aquisição das imagens, conforme descrito na Seção 5.2. A partir das imagens adquiridas foram criadas duas bases de imagens, sendo uma delas não redimensionada (2336 x 3504 pixels) e outra redimensionada (533 x 800 pixels).

A primeira base, contendo 10 grupos de 10 imagens, foi utilizada para construir mosaicos de alta resolução. A segunda base de imagens, contendo 10 grupos de 20 imagens, foi utilizada para comparar as distorções dos mosaicos gerados com outros dois softwares livre de construção de mosaicos.

Para a implementação dos mosaicos foram propostas três metodologias, sendo que as três baseiam-se na Transformada SIFT (LOWE, 2004) para a extração dos descritores de pontos chave, no algoritmo BBF para acelerar a busca por correspondência dos pontos chave e no algoritmo RANSAC para eliminar falsas correspondências e estimar a matriz homográfica. A diferença entre as metodologias está na maneira como os cálculos são realizados para a construção dos mosaicos.

Na metodologia 1 os cálculos são realizados na sequencia de chamada dos

algoritmos. Na metodologia 2 são realizados todos os cálculos e posteriormente composto o mosaico. Na metodologia 3 são determinados cálculos em imagens com resolução 70% menor para minimizar o custo computacional e esses valores servem para estimar a região de junção nas imagens de alta resolução para posterior cálculo dos parâmetros nos pares de imagens e junção das mesmas. Além disso, nessa metodologia é possível ao usuário reduzir o tamanho da região de junção, reduzindo dessa forma o tempo de construção dos mosaicos.

O tempo de construção dos mosaicos com a base de imagens não redimensionada para as três metodologias foi determinado, destacando-se a metodologia 3 com o menor tempo, sendo 63,5% mais rápida que a metodologia 1 e 44,5% mais rápida que a metodologia 2. A metodologia 3 destaca-se como uma das contribuições deste trabalho de doutorado.

Outros algoritmos que devem ser destacados, presentes nas três metodologias, estão relacionados aos ajustes realizados para minimizar as distorções dos mosaicos, que são realizados durante e após o processo de junção das imagens, minimizando o efeito de corte nas junções. Estes algoritmos baseiam-se no trabalho de Silva et. al. (2012b) e diferenciam do algoritmo proposto por Lowe (2004), que é aplicado somente após a construção do mosaico.

O algoritmo de cada metodologia foi paralelizado, utilizando o padrão *OpenMP*. Para avaliar o resultado do paralelismo, foi determinado o tempo de construção dos mosaicos para número de *threads* igual a $n=1$, $n=2$, $n=4$ e $n=8$ e com estes valores foi determinado o *Speedup* para cada grupo de imagens.

Para a metodologia 1, o tempo de processamento reduziu de 54% para $n=4$ em comparação com $n=1$. Para $n=8$, o ganho de tempo foi menos significativo, de aproximadamente 60% em comparação com $n=1$. Os valores de *Speedup* comprovam esses resultados, ou seja, quando se aumenta n de 4 para 8, o aumento do *Speedup* é pouco significativo. Isso se deve ao fato que para $n=4$ são utilizados somente 4 *threads* físicas e para $n=8$ são utilizados 4 *threads* virtuais e 4 *threads* físicas e portanto o desempenho tende a cair. Outro fator que contribui para essa pequena diferença de *Speedup* é que o número de chamadas de algumas funções é da ordem de milhões de chamadas e assim, o tempo necessário para abrir e fechar as *threads* tende ao tempo de execução sequencial.

Para as metodologias 2 e 3 o paralelismo não apresentou resultados satisfatórios. Isso deve ao fato de se utilizar as *threads* para paralelizar as funções de maior custo e não as etapas do mosaico em si. Como são realizados todos os cálculos inicialmente, é realizado o cálculo de uma imagem por vez e o tempo total gasto é a soma dos tempos para uma imagem. Utilizando a função *gettimeofday()* do C++ foi também verificado que a maioria das funções possui um tempo de execução com pouca variação por iteração, ou seja, o aumento da quantidade de imagens não interfere na duração do tempo de execução da função, mas sim no do mosaico. A única função cujo tempo aumenta com a quantidade de imagens é a 'Unirmagens', mas não há benefício em paralelizá-la devido à quantidade de condicionais em sua estrutura interna.

É interessante destacar também que o tempo médio de execução sequencial da metodologia 3 (221s) é ainda menor do que a metodologia 1 paralela para $n=8$ (245s), comprovando que a metodologia 3 destaca-se como a melhor metodologia entre as três implementadas neste trabalho de doutorado.

Foi também implementado o paralelismo de construção de quatro mosaicos com MPI utilizando a metodologia 2. A quantidade de mosaicos foi limitada pela quantidade de *threads* do computador utilizado neste trabalho. O tempo gasto foi o mesmo tempo de construção de um mosaico sequencial. O MPI também foi avaliado para a construção das etapas de um mosaico, tendo sido constatada a impossibilidade de realizar essa implementação pelo fato da biblioteca OpenCV utilizar estrutura de dados não desenvolvidas para MPI.

Finalmente foram avaliadas as distorções dos mosaicos gerados utilizando os grupos de imagens da base de imagens redimensionada, com a metodologia 1. Foram também avaliadas as distorções para os cinco primeiros pares de imagens do grupo 7 de imagens com a metodologia 3. Com o objetivo de comparação, utilizando esse mesmo grupo de imagens, foram gerados mosaicos com outros dois softwares livre: *Autostitch* e *PTGui*. As distorções desses mosaicos foram comparadas utilizando o *Google Earth* como referência. Para determinar essas distorções, foram utilizados pontos de referências e calculados os erros quadráticos médios (EQM) para cada software em relação ao *Google Earth*. Dos dez mosaicos gerados, sete apresentaram um menor valor de EQM para a metodologia 1 deste projeto e três para o *Autostitch* (BROWN; LOWE, 2007). Para os pares de imagens, com a

metodologia 3, dos cinco mosaicos gerados, quatro apresentaram um menor valor de EQM para a metodologia 3 quando comparados com o *Autostitch* e o *PTGui*. Esses resultados também representam contribuição, evidenciando a qualidade dos mosaicos construídos.

O software desenvolvido neste trabalho foi idealizado para construção de mosaicos de forma automática, não sendo necessária qualquer intervenção do usuário durante o processamento. Há, porém, a necessidade do usuário seguir as instruções da Seção 5.2 para aquisição das imagens, garantindo a região de sobreposição das imagens.

Sobre a metodologia, a escolha da Transformada SIFT (LOWE, 2004) baseou-se nos trabalhos de Mikolajczyk e Schmid (2005), Juan e Gwun (2009), Bai et. al (2012) e Ali (2012), que constatarem em seus trabalhos que esta transformada apresenta um desempenho superior em qualidade da descrição dos pontos chave em relação a outros algoritmos para a mesma finalidade.

As principais contribuições deste trabalho que podem ser destacadas são as seguintes:

- Utilização dos algoritmos de Silva (2012b) para reduzir a distorção das junções do mosaico;
- A metodologia 3, que realiza os cálculos de forma mais rápida, utilizando imagens de resolução reduzida para estimar os parâmetros nas imagens de alta resolução;
 - A implementação e avaliação do paralelismo com OpenMP.
 - A implementação e avaliação do paralelismo com MPI.
 - Avaliação e comparação dos mosaicos gerados pela metodologia 1 e 3, com os mosaicos gerados por dois softwares gratuitos (*PTGui* e *Autostitch*).

7.1. Trabalhos Futuros

Dentre os seguimentos que podem ser dados a este trabalho, no que diz respeito a trabalhos futuros, é possível citar:

- Avaliar a metodologia com outros descritores no lugar da Transformada SIFT;

- Implementação da metodologia desenvolvida utilizando CUDA;
- Criação de uma versão do OpenCV que seja compatível com MPI;
- Implementação da metodologia desenvolvida em *cluster* de computadores;
- Eliminação mais eficiente do Ghosting das emendas das imagens conforme sugerido no artigo de Uyttendaele et al.(2001).

7.2. Publicação de Artigos

7.2.1. Artigos Publicados

Poster: *Parallel Processing Applied to Image Mosaic Generation* – IX Workshop de Visão Computacional – Rio de Janeiro - RJ, 2013.

Capítulo de Livro: (DOI: 10.7436/2012.avc.15 ISBN 978-85-64619-09-8) Mosaicos de Imagens Aéreas Sequenciais Construídos Automaticamente. Livro: Avanços em Visão Computacional. Ed. Omnipax, 2012.

Poster: Construção de Mosaicos de Imagens Aéreas Agrícolas e Comparação com Outras Metodologias – VIII Workshop de Visão Computacional – Goiânia - GO, 2012.

Capítulo de Livro: Construção Automática de Mosaico de Imagens Agrícolas Aéreas Sequenciais - 1º Livro da Rede Agricultura de Precisão – Embrapa Instrumentação (CNPDIA), São Carlos - SP, 2011.

Artigo Completo Internacional: Construção Automática de Mosaico de Imagens Agrícolas Aéreas Sequenciais – X Congreso Internacional de la Sociedad Peruana de Computación – Pucallpa, Peru, 2011.

Poster: Mosaico Automático de Imagens Agrícolas Aéreas – Anais do VII Workshop de Visão Computacional – Curitiba - PR, 2011.

Poster: Mosaico Automático de Imagens Agrícolas através da Transformada SIFT – Anais do VI Workshop de Visão Computacional – Presidente Prudente - SP, 2010.

Poster: Mosaico Automático de Imagens Aéreas Agrícolas – I Jornada Científica – Embrapa Pecuária Sudeste/Embrapa Instrumentação (CNPDIA), São Carlos - SP, 2009.

7.2.2. Artigos Aceitos para Publicação

Capítulo de Livro: Uso de Mosaico de Imagens Aéreas como Ferramenta de Auxílio ao Diagnóstico de Diversas Culturas – 2º Livro da Rede Agricultura de Precisão – Embrapa Instrumentação (CNPDIA), São Carlos - SP, 2013.

Poster: Construção de Mosaicos de Imagens Aéreas Agrícolas de Alta Resolução – IV Escola Regional de Alto Desempenho de São Paulo (ERAD-SP) – São Carlos - SP, 2013.

7.2.3. Artigos a Serem Sumetidos

Journal: Computers and Electronics in Agriculture (Elsevier)

Referências Bibliográficas

AGAPITTO, L.; HAYMAN, E.; REID, I.D. **Self-Calibration of Zooming Cameras from a Single Viewpoint**. In: BENOSMAN, R.; KANG, S. Panoramic Vision. SpringerVerlag p.269-289, 2001.

ALI, S.; HUSSAIN, M. **Panoramic Image Construction using Feature based Registration Methods**. In: IEEE Multitopic Conference (INMIC), 2012.

ARMANGUÉ X.; SALVI, J. **Overall view regarding fundamental matrix estimation**. Image and Vision Computing, Guildford, v. 21, p. 205–220, 2003.

ARYA, S.; MOUNT, D. **Approximate Nearest Neighbour Queries in Fixed Dimensions**. Fourth Annual Symposium on Discrete Algorithms (SODA). Pags: 271-280, 1993.

AZAD, P.; ASFOUR, T.; DILLMANN, R. **Combining Harris Interest Points and the SIFT Descriptor for Fast Scale-Invariant Object Recognition**. In: IEEE International Conference On Intelligent Robots And Systems, St. Lois, USA, 2009.

BAI, X.; NING, X; WANG, L. **Analysis and Comparison of Feature Detection and Matching Algorithms for Rovers Vision Navigation**. In: IEEE International Symposium on Instrumentation and Control Technology (ISICT), 2012

BARNEY, B. **OpenMP por Lawrence Livermore National Laboratory**, disponível em: <https://computing.llnl.gov/tutorials/OpenMP/>. Acessado em 25/09/2012, 2012.

BARNESLEY, M., BOXALL, S. R., MATTHEWS, A. M., & WHITE, S. J. **Editorial to special issue**. International Journal of Remote Sensing, 18, 1861–1863, 1997.

BAY, H.; TUYTELAARS, T.; GOOL, L. V. **SURF: Speeded Up Robust Features**, volume 3951 of Lecture Notes in Computer Science, chapter Computer Vision - ECCV 2006. Springer Berlin/ Heidelberg , pp. 404–417, 2006a.

BAY, H.; ESS, A.; TUYTELAARS, T.; GOOL, L. V. **SURF: Speeded Up Robust Features**. CVIU, Vol. 110, No. 3, pp. 346–359, 2008b.

BEI, L; HAIZHEN, Z. **An algorithm of fabric image mosaic based on SIFT feature matching**. IEEE International Conference on Artificial Intelligence and Computational Intelligence. p. 435 – 438, 2009

BEIS, J.; LOWE, D.G.. **Shape indexing using approximate nearest-neighbour search in highdimensional spaces**. Conference on Computer Vision and Pattern Recognition, Puerto Rico, pp. 1000-1006, 1997.

BELO, F. A. W. **Desenvolvimento de Algoritmos de Exploração e Mapeamento Visual para Robôs Móveis de Baixo Custo**. Rio de Janeiro, 276 p. Dissertação (Mestrado). *Pontifícia Universidade Católica do Rio de Janeiro*, 2006.

BENTLEY, J. L. **Multidimensional Binary Search Trees Used For Associative Searching**. Commun. ACM 18, 9, 509–517. (pp. 7, 10, 21), 1975.

BERG, M.; CHEONG, O.; KREVELD, M.; OVERMARS, M. **Computational Geometry: Algorithms and Applications**. Springer-Verlag. 3rd. ed. 2008.

BIELSKI, C., GRAZZINI, J., SOILLE, P. **Automated morphological image composition for mosaicing large image data sets**. IEEE International Symposium on Geoscience and Remote Sensing (IGARSS), 23–28 July, New York. pp. 4068–4071, 2007.

BROWN, M. AND LOWE, D.G. **Invariant features from interest point groups**. In British Machine Vision Conference, Cardiff, Wales, pp. 656-665, 2002.

BROWN, M.; LOWE, D. G. **Automatic Panoramic Image Stitching using Invariant Features**. *International Journal of Computer Vision*, 2007.

CHANDRA, R., DAGUM, L., MENON, R. **Parallel Programming in OpenMP**. San Francisco: Morgan Kaufmann Publishers, 2001.

CHAPMAN, B., JOST G., VAN DER PAS, R. **Using OpenMP**. Massachusetts: Massachusetts Institute of Technology, 2008.

CHO, P.; BAE, S.; DURAND, F. **Image-based querying of urban knowledge databases**. XVIII Signal Processing, Sensor Fusion, and Target Recognition (*The International Society for Optical Engineering – SPIE*), 2009.

CHON, J., KIM, H., LIN, C.S. **Seam-line determination for image mosaicking: a technique minimizing the maximum local mismatch and the global cost**. ISPRS Journal of Photogrammetry and Remote Sensing 65 (1), 86–92, 2010.

CRUZ, B. F. **Aplicação da Técnica SIFT na Identificação de Olhos Humanos**. Dissertação (Mestrado). Universidade do Estado do Rio de Janeiro, 2008.

DU, Q., RAKSUNTORN, N., ORDUYILMAZ, A., BRUCE, L.M. **Automatic registration and mosaicking for airborne multispectral image sequences.** Photogrammetric Engineering & Remote Sensing 74 (2), 169–181, 2008.

EL SAIFI, M.M. PMPI: uma Implementação MPI Multi-plataforma, Multi-linguagem. Dissertação de Mestrado, USP São Paulo, 2006.

FENG, H.; YURONG, E.L.; ZHANG, C. Y. **Parallelization and Characterization of SIFT on Multi-Core Systems.** IEEE International Symposium on Workload Characterization (II SWC), p. 14-23, 2008.

FISCHLER, M. A.; BOLLES, R. C. **Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography.** Comm. of the ACM, v. 24, p. 381- 395, 1981.

FLYNN, M. J. **Some Computer Organizations and Their Effectiveness.** IEEE Transactions on Computers, VOL. c-21, NO. 9, 1972.

FOSTER, C. H. Q. **Alinhamento Imagem-Modelo Baseada na Visão Estéreo de Regiões Planares Arbitrárias.** Tese (Doutorado). *Universidade de Campinas*, 2004.

FRIEDMAN, J.H.; BENTLEY, J.L.; FINKEL, R.A. **An algorithm for finding best matches in logarithmic expected time.** ACM Transactions on Mathematical Software, 3(3):209-226, 1977.

FUSIELLO, A; TRUCCO, E.; VERRI, A. A compact algorithm for rectification of stereo pairs. In Machine Vision and Applications. Vol. 12, p16-22, 2000.

GOLTZ, G. A. M.; SHIGUEMORI, E. H. **Aplicação do Algoritmo SIFT em Imagens de Navegação Autônoma.** Workshop Anual de Pesquisa e Desenvolvimento do IEAv, 2008.

GRAHAM, S.L.; KESSLER, P.B.; MCKUSICK, M.K. **Gprof: a call graph execution profiler.** *SIGPLAN Not.* 39, 4, 49-57, 2004.

GRAMA, A.; GUPTA, A.; KARYPIS, G.; KUMAR, V. **Introduction to Parallel Computing** 2a. edição. Ed. Addison Wesley, 856p - ISBN: 0-201-64865-2, 2003.

GUI, Y.; SU, A.; DU, J. **Point-pattern matching method using SURF and Shape Context.** Optik, 2013.

ORAM, D. **Rectification for Any Epipolar Geometry.** British Machine Vision Conference. Páginas 653-662, 2001.

HALFHILL, T. R. **Parallel Processing With Cuda**. In: Microprocessor report, 2008.

HARRIS, C.; STEPHENS, M. **A combined corner and edge detector**. In: 4th ALVEY VISION CONFERENCE. Manchester, p. 147-151, UK, 1988.

HARTLEY, R.; ZISSERMAN, A.. **Multiple View Geometry in Computer Vision**. Cambridge University Press, 2ª. ed, 2003.

HECKEL, Y. D. **Método Híbrido de Correspondência para Pares Estereoscópicos de Imagens Aéreas e de Satélite de Alta Definição**. Rio de Janeiro, 115 p. Dissertação (Mestrado). *Pontifícia Universidade Católica do Rio de Janeiro*, 2008.

HELMER, E.H., RUEFENACHT, B., **Cloud-free satellite image mosaics with regression trees and histogram matching**. *Photogrammetric Engineering & Remote Sensing* 71 (9), 1079–1089, 2005.

HESS, R. An Open-Source SIFT Library. ACM Multimedia, 2010. Disponível em: <http://robwhess.github.com/opensift/>, acessado em 19/11/2012.

HIRAGA, A.K.; SILVA, F. A. **Algoritmo Para a Construção de Panoramas Contemplando a Minimização das Distorções Causadas nas Junções das Imagens**. 20º Simpósio Internacional de Iniciação Científica da USP, 2011.

HONG, J.; LIN, W.; ZHANG, H; LI, L. **Image Mosaic Based on SURF Feature Matching**, IEEE - The 1st International Conference on Information Science and Engineering ICISE, 2009

HUANG, Y.; LIU, J.; TU, M.; LI, S.; DENG, J. **Research on CUDA-based SIFT Registration of SAR Image**. IEEE Fourth International Symposium on Parallel Architectures, Algorithms and Programming, p. 100-104, 2011.

HU, O. R. T. **Contribuições ao desenvolvimento de um sistema de telepresença por meio da aquisição, transmissão e projeção em ambientes imersivos de vídeos panorâmicos**. Tese de doutorado. Universidade de São Paulo, 2006.

IRANI, M., ANANDAN, P., HSU, S. **Mosaic Based Representations of Video Sequences and Their Applications**. *Proceeding of Fifth International Conference on Computer Vision (ICCV'95)*, 1995.

JIANG, G.; ZHANG, G.; ZHANG, D. **A Distributed Dynamic Parallel Algorithm for SIFT Feature Extraction**. IEEE 3rd International Symposium on Parallel Architectures, Algorithms and Programming, p. 381 – 385, 2010.

JIAYUAN, R.; YIGANG, W.; YUN, D. **Study on Eliminating Wrong Match Pairs of SIFT**. IEEE International Conference on Signal Processing, 2010.

JUAN, L.; GWUN, O., **A Comparison of SIFT, PCA-SIFT and SURF**. International Journal of Image Processing (IJIP) Volume(3), Issue(4), 2009

KAYNIG, V.; FISCHER, B; MULLER, E.; BUHMANN, J. M. **Fully automatic stitching and distortion correction of transmission electron microscope images**. Journal of Structural Biology (171), p. 163–173, 2010.

KE, Y.; SUKTHANKAR, R., **PCA-SIFT: A more distinctive representation for local image descriptors**. In IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), vol. 2, pages 506–513, 2004.

KERSCHNER, M. **Seamline detection in colour orthoimage mosaicking by use of twin snakes**. ISPRS Journal of Photogrammetry and Remote Sensing 56 (1), 53-64, 2001.

KHAN, N.B.; McCANE, B.; WYVILL, G. **IFT and SURF Performance Evaluation Against Various Image Deformations on Benchmark Dataset**. In: IEEE International Conference on Digital Image Computing: Techniques and Applications, 2011

KOENDERINK, J.J. **The structure of images**. Biological Cybernetics, 50:363-396, 1984.

LALIBERTE, A.S., HERRICK, J.E., RANGO, A., WINTERS, C. **Acquisition, orthorectification, and object-based classification of unmanned aerial vehicle (UAV) imagery for rangeland monitoring**. Photogrammetric Engineering & Remote Sensing 76 (6), 661–672, 2010.

LEBOEUF, J., **“Practical applications of remote sensing technology—an industry perspective,”** HortTechnology 10, pp. 475-480, 2000.

LELONG, C.C.D., BURGER, P., JUBELIN, G., ROUX, B.; LABBÉ, S., BARET, F., **“Assessment of Unmanned Aerial Vehicles Imagery for Quantitative Monitoring of Wheat Crop in Small Plots,”** Sensors 2008, 8, pp. 3557-3585, 2008.

LEVIN, A.; ZOMET, A.; PELEG, S.; WEISS, Y., **Seamless Image Stitching in the Gradient Domain**, Proc. 9th European Conference on Computer Vision, Austria, 2006.

LIANG, F.; TINGFA, X.; QIAN, H.; XIAOYAN, W.; PING, W. **SIFT Implementation Based on Parallel Computation**. IEEE 6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM), p. 1-4, 2010

LI, L.; GENG; N. **Algorithm for Sequence Image Automatic Mosaic based on SIFT Feature**. IEEE – International Conference on Information Engineering. P. 203 -206, 2010.

LINDBERG, T. **Scale-space theory: A basic tool for analysing structures at different scales**. Journal of Applied Statistics, 21(2):224-270, 1994.

LIU, J.; CHEN, Z.; GUO, R. **A Mosaic Method for Aerial Image Sequence by R/C Model**. International Conference on Computer Science and Software Engineering – IEEE, 2008.

LIU, J.; CHEN, L.; LI, D. **Invariant feature extraction for color image mosaic by graph card processing**. In Multispectral Image Processing & Pattern Recognition. Proceedings of SPIE, 2009.

LIU, G-H.; LI, Z-Y.; ZHANG, L.; XU, Y. **Image retrieval based on micro-structure descriptor**. *Pattern Recognition* (44), p. 2123–2133, 2011.

LÓPEZ, D.H., GARCÍA, B.F., PIQUERAS, J.G., ALCÁZAR, G.V. An approach to the radiometric aerotriangulation of photogrammetric images. *ISPRS Journal of Photogrammetry and Remote Sensing* 66 (6), 883–893, 2011.

LOWE D. G. **Distinctive Image Features from Scale Invariant**. International Journal of Computer Vision, 2004.

LOWE D. G. **Object Recognition from Local Scale-Invariant Features**. Proceeding of Ninth International Conference on Computer Vision (ICCV'99), 1999.

MA, B.; ZIMMERMANN, T.; ROHDE, M.; WINKELBACH, S.; HE, F.; LINDENMAIER, W.; DITTMAR, K.E.J. **Use of Autostitch for automatic stitching of microscope images**. In: ELSEVIER: The International Research and Review Journal for Microscopy (Micron), ed. 38, p. 492-499, 2007

MACHADO, W.R.S.; LOURO, A.H.F.; GONZAGA, A.; BOAVENTURA, M. **Estudo e proposta de adaptação do algoritmo SIFT em relação ao**

problema de iluminação em imagens. In: IV Workshop de Visão Computacional. Anais do WVC 2008, Bauru: Canal 6 Projetos Editoriais, 2008.

MASSETTO, F. I. Hybrid MPI – Uma Implementação MPI para Ambientes Distribuídos Híbridos. Tese de Doutorado. USP São Paulo, 2007

MIKOLAJCZYK, K.; SCHMID, C. **A Performance Evaluation of Local Descriptors.** IEEE Trans. on Pattern Analysis and Machine Intelligence. Volume 27, Issue 10, Pages: 1615-1630, 2005.

MPI: A Message-Passing Interface Standard Version 3.0. Disponível em: <http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf>. Acessado em: 21 de Maio de 2013.

ORAM, D. **Rectification for Any Epipolar Geometry.** British Machine Vision Conference. Páginas 653-662, 2001.

PAN, J., WANG, M., LI, D.R., LI, J., 2009. **Automatic generation of seamline network using area Voronoi diagrams with overlap.** IEEE Transactions on Geoscience and Remote Sensing 47 (6), 1737–1744.

POLLEFEYS M.; KOCH R.; VAN-GOOL L. **A simple and efficient rectification method for general motion,** Proc. Seventh IEEE Intern. Conf. on Computer Vision, v. 1, p. 496–501, 1999.

ROBERTO, R. A.; TEICHRIEB, V.; KELNER, J. **Retificação Cilíndrica: um método eficiente para retificar um par de imagens,** Sibgrapi, 2009.

ROY S.; MEUNIER J.; COX I. **Cylindrical rectification to minimize epipolar distortion,** Proceedings of the Int. Conf. on Computer Vision and Pattern Recognition, p. 393–399, 1997.

SHUM, H-Y.; SZELISKI, R. **Construction of Panoramic Image Mosaic.** In: BENOSMAN, R., KANG, S. Panoramic Vision. New York: Springer-Verlag, p.227-268, 2001.

SILVA, F. A. **Georreferenciamento Automático de Placas de Sinalização com Imagens Obtidas com um Sistema Móvel de Mapeamento.** Tese de Doutorado apresentada na EESC-USP, 2012a.

SILVA, F. A.; HIRAGA, A.K.; ARTERO, A.O.; PAIVA, M.S.V.; BARBOSA, R.L. **Um Novo Algoritmo Para a Construção de Imagens Panorâmicas Usando os Algoritmos Sift e Ransac.** Anais do III Simpósio Brasileiro de Geomática, Presidente Prudente - SP, 25-27 de julho, v.1, 2012b.

SILVA, L. M. C. **Geração De B-Splines Via Fpga**. Tese de Doutorado apresentada na EESC/USP, 2012.

SOILLE, P. **Morphological image compositing**. IEEE Transactions on Pattern Analysis and Machine Intelligence 28 (5), 673–683, 2006.

SONG, F.; LU, B. **An Automatic Video Image Mosaic Algorithm Based on SIFT Feature Matching**. Proceedings of the 2013 International Conference on Communication, Electronics and Automation Engineering Advances in Intelligent Systems and Computing. Springer: Volume 181, pp 879-88, 2013

SZELISKI, R.; SHUM, H.-Y. **Creating Full View Panoramic Image Mosaics and Environment Maps**. In: Proceedings of 24th International Conference on Computer Graphics and Interactive Techniques. Anais. ACM, P.251-258, 1997.

SUZUKI, T.; AMANO, Y.; HASHIZUME, T. **Vision Based Localization of a Small UAV for Generating a Large Mosaic Image**. In: SICE Annual Conference, p 2960-2964, 2010.

TANENBAUM, A.S. – **Sistemas Operacionais Modernos**, São Paulo: Ed. Pearson, 2009.

TORR, P.; MURRAY, D. **The development and comparison of robust methods for estimating the fundamental matrix**. International Journal of Computer Vision, Dordrecht, v. 24, n. 3, p. 271–300, 1997.

UYTTENDAELE M.; EDEN A.; SZELISKI R.; **Eliminating ghosting and exposure artifacts in image mosaics**. Proc. Conference on Computer Vision and Pattern Recognition (CVPR 2001), 2001.

VASILYEV, A. I.; BOGUSLAVSKIY, A. A.; SOKOLOV, S. M. **Parallel SIFT-detector implementation for images matching**. 21nd International Conference on Computer Graphics and Vision (GraphiCon'2011), 2011.

XING, C.; HUANG, J. **An improved mosaic method based on SIFT algorithm for UAV sequence images**. IEEE International Conference on Computer Design and Applications (ICCD), p. 414-417, 2010.

YANG, L.; WU, X.; ZHAI, J.; LI, H. **A Research of Feature-based Image Mosaic Algorithm**. In: IEEE International Congress on Image and Signal Processing, 2011.

YONGWEI, MA; XINKE, G., XIANJUN, D.; JIAYIN, X. **A New Method of Microscopic Images' Automatic Mosaicing**. IEEE International Conference on Bioinformatics and Biomedical Engineering (ICBBE), 2009.

WANG, C.X., STEFANIDIS, A., CROITORU, A., AGOURIS, P. **Map registration of image sequences using linear features**. Photogrammetric Engineering & Remote Sensing 74 (1), 25–38, 2008.

WANG, Y.; SHEN, J.; LIAO, W.; ZHOU, L. **Automatic Fundus Images Mosaic Based on SIFT Feature**. International Congress on Image and Signal Processing, 2010a.

WANG, Y.; MA, Y.; LIU, P.; LIU, D.; XIE, J. **An Optimized Image Mosaic Algorithm with Parallel I/O And Dynamic Grouped Parallel Strategy Based On Minimal Spanning Tree**. Ninth International Conference on Grid and Cloud Computing, p. 501-506, 2010b.

WARN, S.; EMENEKER, W.; COTHREN, J.; APON, A. **Accelerating SIFT on Parallel Architectures**. IEEE International Conference on Cluster Computing and Workshops 2009 (CLUSTER '09), 2009.

WITKIN, A.P. **Scale-space filtering**. International Joint Conference On Artificial Intelligence, 1983

WULDER, M.A. et al., “**Comparison of Airborne and Satellite High Spatial Resolution Data for the Identification of Individual Trees with Local Maxima Filtering**,” International Journal of Remote Sensing, Vol.25, No.11, pp. 2225–2232, 2004.

ZHANG, Z. **Determining the epipolar geometry and its uncertainty: A review**. The International Journal of Computer Vision, v. 27, n. 2, p. 161–195, August 1998.

ZHAN-LONG, Y.; BAO-LONG, G. **Image Mosaic Based On Sift**. International Conference on Intelligent Information Hiding and Multimedia Signal Processing - IEEE, 2008.

ZHENG, M.; SONG, Z.; XU, K.; LIU, H.- **Parallelization and Optimization of SIFT Feature Extraction on Cluster System**. World Academy of Science, Engineering and Technology 2012 (WASET) p. 273-277, 2012.

ZHONG, M.; ZENG, J.; XIE, X. **Panorama Stitching Based on SIFT Algorithm and Levenberg-Marquardt Optimization**. In: International Conference

on Medical Physics and Biomedical Engineering. Physics Procedia (33), p. 811 – 818, 2012.

Apêndice A – Base de Imagens Sem Redimensionamento

Grupo1

Grupo 2

Grupo 3

IMG_2232.JPG

IMG_2233.JPG

IMG_2234.JPG

IMG_2235.JPG

IMG_2236.JPG

IMG_2237.JPG

IMG_2238.JPG

IMG_2239.JPG

IMG_2240.JPG

IMG_2241.JPG

Grupo 4

IMG_2242.JPG

IMG_2243.JPG

IMG_2244.JPG

IMG_2245.JPG

IMG_2246.JPG

IMG_2247.JPG

IMG_2248.JPG

IMG_2249.JPG

IMG_2250.JPG

IMG_2251.JPG

Grupo 5

IMG_2266.JPG

IMG_2267.JPG

IMG_2268.JPG

IMG_2269.JPG

IMG_2270.JPG

IMG_2271.JPG

IMG_2272.JPG

IMG_2273.JPG

IMG_2274.JPG

IMG_2275.JPG

Grupo 6

IMG_2276.JPG

IMG_2277.JPG

IMG_2278.JPG

IMG_2279.JPG

IMG_2280.JPG

IMG_2281.JPG

IMG_2282.JPG

IMG_2283.JPG

IMG_2284.JPG

IMG_2285.JPG

Grupo 7

IMG_2299.JPG

IMG_2300.JPG

IMG_2301.JPG

IMG_2302.JPG

IMG_2303.JPG

IMG_2304.JPG

IMG_2305.JPG

IMG_2306.JPG

IMG_2307.JPG

IMG_2308.JPG

Grupo 8

IMG_2309.JPG

IMG_2310.JPG

IMG_2311.JPG

IMG_2312.JPG

IMG_2313.JPG

IMG_2314.JPG

IMG_2315.JPG

IMG_2316.JPG

IMG_2317.JPG

IMG_2318.JPG

Grupo 9

IMG_2334.JPG

IMG_2335.JPG

IMG_2336.JPG

IMG_2337.JPG

IMG_2338.JPG

IMG_2339.JPG

IMG_2340.JPG

IMG_2341.JPG

IMG_2342.JPG

IMG_2343.JPG

Grupo 10

IMG_2344.JPG

IMG_2345.JPG

IMG_2346.JPG

IMG_2347.JPG

IMG_2348.JPG

IMG_2349.JPG

IMG_2350.JPG

IMG_2351.JPG

IMG_2352.JPG

IMG_2353.JPG

Apêndice B – Mosaicos da Base Sem Redimensionamento

Grupo 1

Grupo 2

Grupo 3 **Grupo 4**

Grupo 5 **Grupo 6**

Grupo 7 **Grupo 8**

Grupo 9 **Grupo 10**

Apêndice C – Base de Imagens Redimensionadas

Grupo1

IMG_2204.JPG

IMG_2205.JPG

IMG_2206.JPG

IMG_2207.JPG

IMG_2208.JPG

IMG_2209.JPG

IMG_2210.JPG

IMG_2211.JPG

IMG_2212.JPG

IMG_2213.JPG

IMG_2214.JPG

IMG_2215.JPG

IMG_2216.JPG

IMG_2217.JPG

IMG_2218.JPG

IMG_2219.JPG

IMG_2220.JPG

IMG_2221.JPG

IMG_2222.JPG

IMG_2223.JPG

Grupo 2

IMG_2232.JPG

IMG_2233.JPG

IMG_2234.JPG

IMG_2235.JPG

IMG_2236.JPG

IMG_2237.JPG

IMG_2238.JPG

IMG_2239.JPG

IMG_2240.JPG

IMG_2241.JPG

IMG_2242.JPG

IMG_2243.JPG

IMG_2244.JPG

IMG_2245.JPG

IMG_2246.JPG

IMG_2247.JPG

IMG_2248.JPG

IMG_2249.JPG

IMG_2250.JPG

IMG_2251.JPG

Grupo 3

IMG_2266.JPG

IMG_2267.JPG

IMG_2268.JPG

IMG_2269.JPG

IMG_2270.JPG

IMG_2271.JPG

IMG_2272.JPG

IMG_2273.JPG

IMG_2274.JPG

IMG_2275.JPG

IMG_2276.JPG

IMG_2277.JPG

IMG_2278.JPG

IMG_2279.JPG

IMG_2280.JPG

IMG_2281.JPG

IMG_2282.JPG

IMG_2283.JPG

IMG_2284.JPG

IMG_2285.JPG

Grupo 4

IMG_2299.JPG

IMG_2300.JPG

IMG_2301.JPG

IMG_2302.JPG

IMG_2303.JPG

IMG_2304.JPG

IMG_2305.JPG

IMG_2306.JPG

IMG_2307.JPG

IMG_2308.JPG

IMG_2309.JPG

IMG_2310.JPG

IMG_2311.JPG

IMG_2312.JPG

IMG_2313.JPG

IMG_2314.JPG

IMG_2315.JPG

IMG_2316.JPG

IMG_2317.JPG

IMG_2318.JPG

Grupo 5

IMG_2334.JPG

IMG_2335.JPG

IMG_2336.JPG

IMG_2337.JPG

IMG_2338.JPG

IMG_2339.JPG

IMG_2340.JPG

IMG_2341.JPG

IMG_2342.JPG

IMG_2343.JPG

IMG_2344.JPG

IMG_2345.JPG

IMG_2346.JPG

IMG_2347.JPG

IMG_2348.JPG

IMG_2349.JPG

IMG_2350.JPG

IMG_2351.JPG

IMG_2352.JPG

IMG_2353.JPG

Grupo 6

IMG_2270.JPG

IMG_2271.JPG

IMG_2272.JPG

IMG_2273.JPG

IMG_2274.JPG

IMG_2275.JPG

IMG_2276.JPG

IMG_2277.JPG

IMG_2278.JPG

IMG_2279.JPG

IMG_2280.JPG

IMG_2281.JPG

IMG_2282.JPG

IMG_2283.JPG

IMG_2284.JPG

IMG_2285.JPG

IMG_2286.JPG

IMG_2287.JPG

IMG_2288.JPG

IMG_2289.JPG

Grupo 7

Grupo 8

IMG_2380.JPG

IMG_2381.JPG

IMG_2382.JPG

IMG_2383.JPG

IMG_2384.JPG

IMG_2385.JPG

IMG_2386.JPG

IMG_2387.JPG

IMG_2388.JPG

IMG_2389.JPG

IMG_2390.JPG

IMG_2391.JPG

IMG_2392.JPG

IMG_2393.JPG

IMG_2394.JPG

IMG_2395.JPG

IMG_2396.JPG

IMG_2397.JPG

IMG_2398.JPG

IMG_2399.JPG

Grupo 9

IMG_2310.JPG

IMG_2311.JPG

IMG_2312.JPG

IMG_2313.JPG

IMG_2314.JPG

IMG_2315.JPG

IMG_2316.JPG

IMG_2317.JPG

IMG_2318.JPG

IMG_2319.JPG

IMG_2320.JPG

IMG_2321.JPG

IMG_2322.JPG

IMG_2323.JPG

IMG_2324.JPG

IMG_2325.JPG

IMG_2326.JPG

IMG_2327.JPG

IMG_2328.JPG

IMG_2329.JPG

Grupo 10

IMG_2280.JPG

IMG_2281.JPG

IMG_2282.JPG

IMG_2283.JPG

IMG_2284.JPG

IMG_2285.JPG

IMG_2286.JPG

IMG_2287.JPG

IMG_2288.JPG

IMG_2289.JPG

IMG_2290.JPG

IMG_2291.JPG

IMG_2292.JPG

IMG_2293.JPG

IMG_2294.JPG

IMG_2295.JPG

IMG_2296.JPG

IMG_2297.JPG

IMG_2298.JPG

IMG_2299.JPG

Apêndice D – Mosaicos da Base de Imagens Redimensionada

Grupo 1

Grupo 2

Grupo 3 **Grupo 4**

Grupo 5 **Grupo 6**

Grupo 7 **Grupo 8**

Grupo 9 **Grupo 10**

Apêndice E – Comparação da Distorção Utilizando a Base Redimensionada Grupo 1

Google Earth

Metodologia Desenvolvida

Autostitch Demo

PTGui Trial

Grupo 2

Google Earth

Metodologia Desenvolvida

Autostitch Demo

PTGui Trial

Grupo 3

Google Earth

Metodologia Desenvolvida

Autostitch Demo

PTGui Trial

Grupo 4

Google Earth

Metodologia Desenvolvida

Autostitch Demo

PTGui Trial

Grupo 5

Google Earth

Metodologia Desenvolvida

Autostitch Demo

PTGui Trial

Grupo 6

Google Earth

Metodologia Desenvolvida

Autostitch Demo

PTGui Trial

Grupo 7

Google Earth

Metodologia Desenvolvida

Autostitch Demo

PTGui Trial

Grupo 8

Google Earth

Metodologia Desenvolvida

Autostitch Demo

PTGui Trial

Grupo 9

Google Earth

Metodologia Desenvolvida

Autostitch Demo

PTGui Trial

Grupo 10

Google Earth

Metodologia Desenvolvida

Autostitch Demo

PTGui Trial

Apêndice F – Valores das Coordenadas A,B,C para Cálculo da Distorção.

Tabela F1 – Valores das marcações do grupo 1.

Grupo1 - Google Earth				
Coordenadas GPS		X (UTM)	Y (UTM)	Ponto
-22,25951	-45,718961	425922,623363654	7538272,408379230	A
-22,244179	-45,699091	427962,139939887	7539979,075911690	B
-22,235598	-45,675659	430372,533371206	7540939,914245720	C
Grupo1 - Autostitch Demo				
Coordenadas da Imagem		X	Y	Ponto
		270	189	A
		721	141	B
		1102	212	C
Grupo1 - PTGui Trial				
Coordenadas da Imagem		X	Y	Ponto
		2998	1930	A
		4120	1648	B
		5324	1716	C
Grupo1 - Metodologia 1				
Coordenadas da Imagem		X	Y	Ponto
		356	283	A
		980	204	B
		1610	339	C

Tabela F2 – Valores das marcações do grupo 2.

Grupo2 - Google Earth				
Coordenadas GPS		X (UTM)	Y (UTM)	Ponto
-22,225071	-45,663128	431658,807044870	7542110,904036480	A
-22,236035	-45,677977	430133,869324447	7540890,472768360	B
-22,249491	-45,71909	425904,059689832	7539381,405303090	C
Grupo2 - Autostitch Demo				
Coordenadas da Imagem		X	Y	Ponto
		211	136	A
		628	99	B
		1174	192	C
Grupo2 - PTGui Trial				
Coordenadas da Imagem		X	Y	Ponto
		1036	812	A
		2128	268	B
		3715	415	C
Grupo2 - Metodologia 1				
Coordenadas da Imagem		X	Y	Ponto
		279	199	A
		918	119	B
		1803	241	C

Tabela F3 – Valores das marcações do grupo 3.

Grupo3 - Google Earth				
Coordenadas GPS		X (UTM)	Y (UTM)	Ponto
-22,26658	-45,760331	421663,869897110	7537468,937863210	A
-22,25256	-45,732608	424512,738873028	7539034,997520170	B
-22,244139	-45,699091	427966,426056635	7539051,338904760	C
Grupo3 - Autostitch Demo				
Coordenadas da Imagem		X	Y	Ponto
		362	121	A
		862	158	B
		1347	278	C
Grupo3 - PTGui Trial				
Coordenadas da Imagem		X	Y	Ponto
		1251	1099	A
		2640	1289	B
		4157	1586	C
Grupo3 - Metodologia 1				
Coordenadas da Imagem		X	Y	Ponto
		519	271	A
		1274	305	B
		2061	509	C

Tabela F4 – Valores das marcações do grupo 4.

Grupo4 - Google Earth				
Coordenadas GPS		X (UTM)	Y (UTM)	Ponto
-22,238459	-45,716901	426123,840940722	7540603,668515670	A
-22,228567	-45,680037	429917,864815360	7541716,188911440	B
-22,210808	-45,653129	432682,518874472	7543694,211539380	C
Grupo4 - Autostitch Demo				
Coordenadas da Imagem		X	Y	Ponto
		83	137	A
		561	81	B
		1082	184	C
Grupo4 - PTGui Trial				
Coordenadas da Imagem		X	Y	Ponto
		561	1179	A
		1954	646	B
		3663	930	C
Grupo4 - Metodologia 1				
Coordenadas da Imagem		X	Y	Ponto
		113	212	A
		850	123	B
		1672	250	C

Tabela F5 – Valores das marcações do grupo 5.

Grupo5 - Google Earth				
Coordenadas GPS		X (UTM)	Y (UTM)	Ponto
-22,253553	-45,755482	422156,275048388	7538913,484375710	A
-22,241557	-45,73617	424139,773951287	7540251,202748050	B
-22,238697	-45,716858	426128,396984508	7540577,343986080	C
Grupo5 - Autostitch Demo				
Coordenadas da Imagem		X	Y	Ponto
		398	172	A
		782	146	B
		1064	225	C
Grupo5 - PTGui Trial				
Coordenadas da Imagem		X	Y	Ponto
		1823	1264	A
		2790	1202	B
		3637	958	C
Grupo5 - Metodologia 1				
Coordenadas da Imagem		X	Y	Ponto
		537	237	A
		1128	294	B
		1588	432	C

Tabela F6 – Valores das marcações do grupo 6.

Grupo6 - Google Earth				
Coordenadas GPS		X (UTM)	Y (UTM)	Ponto
-22,252401	-45,715656	426259,440265789	7539060,958803880	A
-22,242669	-45,702739	427585,442035033	7540144,484785800	B
-22,236512	-45,68819	429081,614539270	7540832,922648860	C
Grupo6 - Autostitch Demo				
Coordenadas da Imagem		X	Y	Ponto
		819	249	A
		1021	209	B
		1288	239	C
Grupo6 - PTGui Trial				
Coordenadas da Imagem		X	Y	Ponto
		3505	2083	A
		4023	1751	B
		4901	1587	C
Grupo6 – Metodologia 1				
Coordenadas da Imagem		X	Y	Ponto
		1170	363	A
		1554	296	B
		1953	339	C

Tabela F7 – Valores das marcações do grupo 7.

Grupo7 - Google Earth				
Coordenadas GPS		X (UTM)	Y (UTM)	Ponto
-22,246919	-45,746641	423063,651805151	7539652,365156900	A
-22,234129	-45,726042	425179,532529241	7541078,490824560	B
-22,231149	-45,69437	428442,017125407	7541423,671884050	C
Grupo7 - Autostitch Demo				
Coordenadas da Imagem		X	Y	Ponto
		204	126	A
		595	100	B
		1036	277	C
Grupo7 - PTGui Trial				
Coordenadas da Imagem		X	Y	Ponto
		2269	1074	A
		3315	698	B
		4680	893	C
Grupo7 - Metodologia 1				
Coordenadas da Imagem		X	Y	Ponto
		264	221	A
		863	195	B
		1573	436	C

Tabela F8 – Valores das marcações do grupo 8.

Grupo8 - Google Earth				
Coordenadas GPS		X (UTM)	Y (UTM)	Ponto
-22,22817	-45,721407	425654,056748911	7541740,408001460	A
-22,240524	-45,743165	423418,368757457	7540362,029080970	B
-22,243305	-45,771618	420487,789756439	7540039,510845870	C
Grupo8 - Autostitch Demo				
Coordenadas da Imagem		X	Y	Ponto
		204	130	A
		601	81	B
		1009	201	C
Grupo8 - PTGui Trial				
Coordenadas da Imagem		X	Y	Ponto
		268	200	A
		754	106	B
		1134	263	C
Grupo8 - Metodologia 1				
Coordenadas da Imagem		X	Y	Ponto
		327	277	A
		957	166	B
		1673	390	C

Tabela F9 – Valores das marcações do grupo 9.

Grupo9 - Google Earth				
Coordenadas GPS		X (UTM)	Y (UTM)	Ponto
-22,229679	-45,702653	427587,630753385	7541582,456071030	A
-22,244616	-45,72896	424884,403785157	7539916,180538780	B
-22,25252	-45,758228	421872,746414990	7539026,418140430	C
Grupo9 - Autostitch Demo				
Coordenadas da Imagem		X	Y	Ponto
		135	156	A
		563	99	B
		1011	178	C
Grupo9 - PTGui Trial				
Coordenadas da Imagem		X	Y	Ponto
		640	475	A
		2184	272	B
		3567	464	C
Grupo9 - Metodologia 1				
Coordenadas da Imagem		X	Y	Ponto
		186	235	A
		829	120	B
		1521	247	C

Tabela F10 – Valores das marcações do grupo 10.

Grupo10 - Google Earth				
Coordenadas GPS		X (UTM)	Y (UTM)	Ponto
-22,252361	-45,715699	426254,988472403	7539065,365668260	A
-22,241199	-45,700936	427770,487387304	7540308,068291260	B
-22,231745	-45,67759	430171,626217169	7541365,531622880	C
Grupo10 - Autostitch Demo				
Coordenadas da Imagem		X	Y	Ponto
		170	166	A
		436	114	B
		815	181	C
Grupo10 - PTGui Trial				
Coordenadas da Imagem		X	Y	Ponto
		1214	558	A
		2135	306	B
		3593	276	C
Grupo10 - Metodologia 1				
Coordenadas da Imagem		X	Y	Ponto
		252	269	A
		703	188	B
		1372	267	C

Apêndice G – Valores dos Ângulos e dos Segmentos de Reta

Da Tabela G1 à Tabela G10, são apresentados os valores dos ângulos e dos comprimentos dos segmentos de reta, separados por grupos, sendo importante ressaltar que as medidas obtidas pelo Google Earth são em metros e as obtidas pela metodologia desenvolvida, *Autostitch* e *PTGui* são em pixels.

Tabela G1– Ângulos e segmentos de reta do grupo 1.

	ângulos			Medidas		
	β	α	γ	AB	AC	BC
Metodologia 1	9,772	160,690	9,538	628,981	1255,250	644,302
<i>Autostitch</i>	7,659	163,369	8,973	453,547	832,318	387,559
<i>PTGui</i>	8,852	162,659	8,489	1156,896	2335,824	1205,919
<i>Google Earth</i>	8,982	161,811	9,207	2659,388	5188,187	2594,842

Tabela G2– Ângulos e segmentos de reta do grupo 2.

	ângulos			Medidas		
	β	α	γ	AB	AC	BC
Metodologia 1	8,715	165,015	6,270	643,988	1524,579	893,369
<i>Autostitch</i>	8,399	165,263	6,338	418,638	964,627	553,864
<i>PTGui</i>	18,052	148,227	13,721	1220,000	2708,256	1593,794
<i>Google Earth</i>	13,296	160,964	5,740	1953,174	6369,245	4490,944

Tabela G3– Ângulos e segmentos de reta do grupo 3.

	ângulos			Medidas		
	β	α	γ	AB	AC	BC
Metodologia 1	6,196	168,047	5,758	755,765	1560,259	813,010
<i>Autostitch</i>	4,824	170,335	4,841	501,367	997,434	499,625
<i>PTGui</i>	1,724	176,712	1,564	1401,935	2946,524	1545,800
<i>Google Earth</i>	14,704	151,473	13,823	3250,938	6498,170	3453,726

Tabela G4– Ângulos e segmentos de reta do grupo 4.

	ângulos			Medidas		
	β	α	γ	AB	AC	BC
Metodologia 1	8,282	164,331	7,387	742,354	1559,463	831,753
<i>Autostitch</i>	9,376	162,135	8,489	481,269	1000,105	531,084
<i>PTGui</i>	16,349	149,627	14,024	1491,489	3111,978	1732,437
<i>Google Earth</i>	8,888	160,760	10,352	3953,773	7250,359	3399,395

Tabela G5– Ângulos e segmentos de reta do grupo 5.

	ângulos			Medidas		
	β	α	γ	AB	AC	BC
Metodologia 1	5,002	168,810	6,188	593,742	1068,937	480,254
<i>Autostitch</i>	8,423	160,477	11,100	384,879	668,106	292,857
<i>PTGui</i>	5,906	167,598	6,495	968,986	1839,628	881,445
<i>Google Earth</i>	11,269	155,317	13,414	2392,438	4306,528	2015,190

Tabela G6– Ângulos e segmentos de reta do grupo 6.

	ângulos			Medidas		
	β	α	γ	AB	AC	BC
Metodologia 1	8,142	163,952	7,907	389,801	783,368	401,310
<i>Autostitch</i>	9,979	162,388	7,632	205,922	469,107	268,680
<i>PTGui</i>	13,097	157,923	8,980	615,263	1481,497	893,185
<i>Google Earth</i>	7,130	165,455	7,415	1712,399	3332,345	1646,960

Tabela G7– Ângulos e segmentos de reta do grupo 7.

	ângulos			Medidas		
	β	α	γ	AB	AC	BC
Metodologia 1	11,813	158,766	9,422	599,564	1326,539	749,787
<i>Autostitch</i>	14,091	154,327	11,582	391,863	845,592	475,195
<i>PTGui</i>	15,478	152,098	12,423	1111,527	2417,785	1378,858
<i>Google Earth</i>	15,752	152,059	12,189	2551,624	5662,538	3280,694

Tabela G8– Ângulos e segmentos de reta do grupo 8.

	ângulos			Medidas		
	β	α	γ	AB	AC	BC
Metodologia 1	14,791	152,635	12,573	639,704	1350,735	750,221
<i>Autostitch</i>	12,077	156,574	11,349	400,012	808,125	425,281
<i>PTGui</i>	15,108	146,605	18,288	495,007	868,289	411,156
<i>Google Earth</i>	13,432	154,625	11,943	2626,448	5439,059	2948,273

Tabela G9– Ângulos e segmentos de reta do grupo 9.

	ângulos			Medidas		
	β	α	γ	AB	AC	BC
Metodologia 1	10,655	159,460	9,885	653,203	1335,054	703,557
<i>Autostitch</i>	9,025	162,413	8,562	431,779	876,276	454,912
<i>PTGui</i>	7,275	164,606	8,119	1557,288	2927,021	1396,264
<i>Google Earth</i>	7,553	164,809	7,638	3175,517	6260,450	3140,343

Tabela G10– Ângulos e segmentos de reta do grupo 10.

	ângulos			Medidas		
	β	α	γ	AB	AC	BC
Metodologia 1	10,080	163,083	6,837	458,216	1120,002	673,648
<i>Autostitch</i>	12,393	158,914	8,693	271,035	645,174	384,877
<i>PTGui</i>	8,542	165,876	5,581	954,853	2395,655	1458,309
<i>Google Earth</i>	8,927	164,417	6,656	1959,859	4542,116	2623,680

Apêndice H – Comparação da Distorção por Pares de Imagens

Par 1

Google Earth

Metodologia 3

AutoStitch Demo

PTGui Trial

Par 2

Google Earth

Metodologia 3

AutoStitch Demo

PTGui Trial

Par 3

Google Earth

Metodologia 3

AutoStitch Demo

PTGui Trial

Par 4

Google Earth

Metodologia 3

AutoStitch Demo

PTGui Trial

Par 5

Google Earth

Metodologia 3

AutoStitch Demo

PTGui Trial

Apêndice I – Valores das Coordenadas A,B,C para Cálculo da Distorção.

Tabela I1 – Valores das marcações do par 1.

Google Earth				
Coordenadas GPS		X (UTM)	Y (UTM)	Ponto
-22,214066	-45,647342	433280,539086627	7543336,129701730	A
-22,219986	-45,651008	432763,981356775	7542680,807415690	B
-22,21939	-45,658454	432137,785954045	7542741,859511080	C
AutoStitch Demo				
Coordenadas da Imagem		X	Y	Ponto
		201	335	A
		738	125	B
		1236	394	C
PTGui Trial				
Coordenadas da Imagem		X	Y	Ponto
		479	893	A
		1735	347	B
		2917	983	C
Metodologia 3				
Coordenadas da Imagem		X	Y	Ponto
		425	833	A
		1733	315	B
		2945	951	C

Tabela I2 – Valores das marcações do par 2.

Google Earth				
Coordenadas GPS		X (UTM)	Y (UTM)	Ponto
-22,217304	-45,655965	432763,981356775	7542680,807415690	A
-22,223701	-45,66002	431978,463296705	7542263,954264450	B
-22,216748	-45,66826	431125,828057048	7543029,888889500	C
AutoStitch Demo				
Coordenadas da Imagem		X	Y	Ponto
		215	367	A
		877	207	B
		1045	883	C
PTGui Trial				
Coordenadas da Imagem		X	Y	Ponto
		533	1045	A
		2081	693	B
		2477	2261	C
Metodologia 3				
Coordenadas da Imagem		X	Y	Ponto
		521	941	A
		2161	577	B
		2561	2265	C

Tabela I3 – Valores das marcações do par 3.

Google Earth				
Coordenadas GPS		X (UTM)	Y (UTM)	Ponto
-22,216728	-45,668238	431128,085796037	7543032,112776140	A
-22,226164	-45,672895	430652,735966858	7541985,475605770	B
-22,222896	-45,678021	430122,823362236	7542344,869586600	C
AutoStitch Demo				
Coordenadas da Imagem		X	Y	Ponto
		255	763	A
		959	265	B
		1193	685	C
PTGui Trial				
Coordenadas da Imagem		X	Y	Ponto
		657	2101	A
		2425	769	B
		2965	1881	C
Metodologia 3				
Coordenadas da Imagem		X	Y	Ponto
		597	2041	A
		2385	701	B
		2969	1805	C

Tabela I4 – Valores das marcações do par 4.

Google Earth				
Coordenadas GPS		X (UTM)	Y (UTM)	Ponto
-22,225369	-45,672487	430694,394333541	7542073,664499770	A
-22,229699	-45,679139	430010,974466548	7541591,297828930	B
-22,226045	-45,683815	429527,244205334	7541993,607831500	C
AutoStitch Demo				
Coordenadas da Imagem		X	Y	Ponto
		207	283	A
		859	177	B
		1087	595	C
PTGui Trial				
Coordenadas da Imagem		X	Y	Ponto
		389	829	A
		2077	493	B
		2609	1601	C
Metodologia 3				
Coordenadas da Imagem		X	Y	Ponto
		366	809	A
		1981	485	B
		2537	1533	C

Tabela I5 – Valores das marcações do par 5.

Google Earth				
Coordenadas GPS		X (UTM)	Y (UTM)	Ponto
-22,229143	45,681413	428303,173958930	7541651,791353470	A
-22,230673	-45,687872	429111,449431318	7541479,418968760	B
-22,226601	-45,695704	428302,224829136	7541926,481491660	C
AutoStitch Demo				
Coordenadas da Imagem		X	Y	Ponto
		193	317	A
		845	535	B
		1237	1117	C
PTGui Trial				
Coordenadas da Imagem		X	Y	Ponto
		473	721	A
		1873	1241	B
		2733	2493	C
Metodologia 3				
Coordenadas da Imagem		X	Y	Ponto
		421	709	A
		1813	1217	B
		2685	2497	C

Apêndice J – Valores dos Ângulos e dos Segmentos de Reta

Da Tabela J1 à Tabela J5, são apresentados os valores dos ângulos e dos comprimentos dos segmentos de reta, separados por pares, sendo importante ressaltar que as medidas obtidas pelo Google Earth são em metros e as obtidas pela metodologia desenvolvida, *Autostitch* e *PTGui* são em pixels.

Tabela J1 – Ângulos e segmentos de reta do par 1.

	ângulos			Medidas		
	β	α	γ	AB	AC	BC
Metodologia 3	24,286	130,707	25,007	1406,836	2522,761	1368,737
AutoStitch	24,621	130,265	25,114	576,601	1036,680	566,008
PTGUI	25,609	128,221	26,169	1369,544	2439,661	1342,244
Google Earth	24,277	122,678	33,045	834,433	1288,038	629,165

Tabela J2 – Ângulos e segmentos de reta do par 2.

	ângulos			Medidas		
	β	α	γ	AB	AC	BC
Metodologia 3	45,498	90,817	43,684	1679,910	2431,990	1734,746
AutoStitch	45,456	90,369	44,175	681,061	977,321	696,563
PTGUI	44,837	91,363	43,800	1587,516	2292,988	1617,232
Google Earth	39,983	110,113	29,904	889,272	1674,934	1146,143

Tabela J3 – Ângulos e segmentos de reta do par 3.

	ângulos			Medidas		
	β	α	γ	AB	AC	BC
Metodologia 3	31,167	81,029	67,804	2234,400	2383,711	1248,948
AutoStitch	30,522	83,849	65,629	862,334	941,237	480,787
PTGUI	31,549	78,908	69,543	2213,605	2318,462	1236,181
Google Earth	31,216	80,280	68,504	1149,525	1217,726	640,290

Tabela J4 – Ângulos e segmentos de reta do par 4.

	ângulos			Medidas		
	β	α	γ	AB	AC	BC
Metodologia 3	29,787	106,603	43,610	1647,180	2288,540	1186,356
AutoStitch	28,756	109,376	41,868	660,560	933,672	476,139
PTGUI	30,433	104,390	45,177	1721,116	2350,401	1229,100
Google Earth	31,291	105,035	43,674	836,505	1169,893	629,165

Tabela J5 – Ângulos e segmentos de reta do par 5.

	ângulos			Medidas		
	β	α	γ	AB	AC	BC
Metodologia 3	18,251	144,314	17,435	1481,799	2884,899	1548,801
AutoStitch	18,975	142,449	18,576	687,479	1315,270	701,704
PTGUI	17,723	144,862	17,416	1493,452	2871,861	1518,915
Google Earth	102,236	16,880	60,883	826,451	274,692	924,505