ESTE EXEMPLAR CORRESPONDE A REDAÇÃO FINAL DA TESE DEFENDIDA POR GIOVANI BERNARDES VITOR E APROVADA PELA COMISSÃO JULGADORA EM 13/08/2010

ORIENTADOR

UNICAMP

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
COMISSÃO DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA

Giovani Bernardes Vitor

Rastreamento de alvo móvel em mono-visão aplicado no sistema de navegação autônoma utilizando GPU

Campinas, 2010

85/2010

Giovani Bernardes Vitor

Rastreamento de alvo móvel em mono-visão aplicado no sistema de navegação autônoma utilizando GPU

Dissertação de mestrado acadêmico apresentada à comissão de Pós Graduação da Faculdade de Engenharia Mecânica, como requisito para obtenção do título de Mestre em Engenharia Mecânica.

Área de Concentração: Mecânica Dos Sólidos E Projeto Mecânico

Orientador: Janito Vaqueiro Ferreira

Campinas
2010

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

V833r

Vitor, Giovani Bernardes
Rastreamento de alvo móvel em mono-visão aplicado no sistema de navegação autônoma utilizando GPU. / Giovani Bernardes Vitor. --Campinas, SP: [s.n.], 2010.

Orientador: Janito Vaqueiro Ferreira.
Dissertação de Mestrado - Universidade Estadual de Campinas, Faculdade de Engenharia Mecânica.

1. Visão por computador. 2. Processamento de imagens. 3. Rastreamento automático. 4. Computação de alto desempenho. 5. Navegação de robôs móveis. I. Ferreira, Janito Vaqueiro. II. Universidade Estadual de Campinas. Faculdade de Engenharia Mecânica. III. Título.

Título em Inglês: Tracking of target moving in monocular vision system applied to autonomous navigation using GPU.
Palavras-chave em Inglês: Computer vision, Image processing, Automatic tracking, High performance computing, Autonomous robot´s navigation
Área de concentração: Mecânica dos Sólidos e Projeto Mecânico
Titulação: Mestre em Engenharia Mecânica
Banca examinadora: Clésio Luis Tozzi, Ely Carneiro de Paiva
Data da defesa: 13/08/2010
Programa de Pós Graduação: Engenharia Mecânica

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA
DEPARTAMENTO DE MECÂNICA COMPUTACIONAL

DISSERTAÇÃO DE MESTRADO ACADEMICO

Rastreamento de alvo móvel em mono-visão aplicado no sistema de navegação autônoma utilizando GPU

Autor: Giovani Bernardes Vitor
Orientador: Janito Vaqueiro Ferreira

A Banca Examinadora composta pelos membros abaixo aprovou esta Dissertação:

Prof. Dr. Janito Vaqueiro Ferreira, Presidente
DMC/FEM/UNICAMP

Prof. Dr. Clésio Luis Tozzi
DCA/FEEC/UNICAMP

Prof. Dr. Ely Carneiro de Paiva
DPM/FEM/UNICAMP

Campinas, 13 de agosto de 2010

Dedicatória:

A meus pais,
José Maria e Maria de Lourdes (in memorian),
pelo apoio e incentivo desmedido de sempre
as decisões por mim tomadas.

Agradecimentos

À DEUS, sempre;

Agradeço ao Prof. Dr. Janito Vaqueiro Ferreira, pela confiança, apoio, paciência e amizade;

Aos meus pais, José Maria e Maria de Lourdes (in memorian), meus irmãos Gilson Bernardes Vitor, Gislene Bernardes Vitor e Giomara Bernardes Vitor, pelo amor, incentivo, companheirismo e compreensão;

Ao prof. Dr. Clésio Luiz Tozzi, tanto pela disponibilidade quanto pelas valiosas sugestões;

Aos professores Dr. Roberto de Alencar Lotufo, Dr. Douglas Eduardo Zampieri, Dr. Luiz Otávio Saraiva Ferreira, pela ajuda em diversos momentos, sugestões e discussões proveitosas;

Aos meus colegas do Departamento de Mecânica Computacional da FEM e do Laboratório de Computação e Automação da FEEC, em especial para meu amigo e colega de publicações André Korbes, e também para os amigos Greice Martins de Freitas, Fernando Paoliere Neto, Danilo Pagano, Wendell Diniz, Josué Labaki, Rolando Perez, Marcelo Cavaguti, Ruben Dario, Camilo Gordillo, pelos diversos e agradáveis momentos de convívio em Campinas;

Aos meus colegas do Laboratório de Mobilidade, em especial para Arthur de Miranda Neto, Luiz Gustavo Turatti e Justo Emilio Alvarez Jácobo no qual erros foram mapeados, sugestões foram apresentadas e muitas idéias surgiram;

Aos meus amigos Rodrigo Braga de Oliveira e Antônio Paulo Belo Junior pelo eterno companheirismo, paciência em todos os momentos, especialmente na divisão do apartamento e nas festas;

À Beth Viana, secretária do DMC, pelas instruções, atenção e ajuda nos momentos que precisei;

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pela bolsa que me foi concedida;

"Não estejais inquietos por coisa alguma;
Antes as vossas petições sejam em tudo
conhecidas diante de Deus pela oração
e súplica, com ação de graças."
Filipenses 4:6

Resumo

O sistema de visão computacional é bastante útil em diversas aplicações de veículos autônomos, como em geração de mapas, desvio de obstáculos, tarefas de posicionamento e rastreamento de alvos. Além disso, a visão computacional pode proporcionar um ganho significativo na confiabilidade, versatilidade e precisão das tarefas robóticas, questões cruciais na maioria das aplicações reais.

O presente trabalho tem como objetivo principal o desenvolvimento de uma metodologia de controle servo visual em veículos robóticos terrestres para a realização de rastreamento e perseguição de um alvo. O procedimento de rastreamento é baseado na correspondência da região alvo entre a seqüência de imagens, e a perseguição pela geração do movimento de navegação baseado nas informações da região alvo.

Dentre os aspectos que contribuem para a solução do procedimento de rastreamento proposto, considera-se o uso das técnicas de processamento de imagens como filtro KNN, filtro Sobel, filtro HMIN e transformada Watershed que unidas proporcionam a robustez desejada para a solução. No entanto, esta não é uma técnica compatível com sistema de tempo real. Deste modo, tais algoritmos foram modelados para processamento paralelo em placas gráficas utilizando CUDA.

Experimentos em ambientes reais foram analisados, apresentando diversos resultados para o procedimento de rastreamento, bem como validando a utilização das GPU´s para acelerar o processamento do sistema de visão computacional.

Palavras Chaves: Rastreamento, GPU, GPGPU, CUDA, Visão por Computador, Robôs Móveis, Navegação de Robôs Móveis, Sistema Paralelo, Sistema Distribuído

Abstract

The computer vision system is useful in several applications of autonomous vehicles, such as map generation, obstacle avoidance tasks, positioning tasks and target tracking. Furthermore, computer vision can provide a significant gain in reliability, versatility and accuracy of robotic tasks, which are important concerns in most applications.

The present work aims at the development of a visual servo control method in ground robotic vehicles to perform tracking and follow of a target. The procedure for tracking is based on the correspondence between the target region sequence of images, and persecution by the generation of motion based navigation of information from target region.

Among the aspects that contribute to the solution of the proposed tracking procedure, we consider the use of imaging techniques such as KNN filter, Sobel filter, HMIN filter and Watershed transform that together provide the desired robustness for the solution. However, this is not a technique compatible with real-time system. Thus, these algorithms were modeled for parallel processing on graphics cards using CUDA.

Experiments in real environments were analyzed showed different results for the procedure for tracking and validating the use of GPU's to accelerate the processing of computer vision system.

Keywords: Tracking, GPU, GPGPU, CUDA, Computer Vision, Mobile Robots, Navigation, Parallel System, Distributed System

Lista de Figuras

Figura 1.1: Distribuição de verbas por área científica. FONTE (SILVEIRA, 2010) 3
Figura 1.2: Cenário de operação dos Veículos autônomos 4
Figura 1.3: Sistema de Rastreamento. (a) Protótipo do equipamento (b) Fase de calibração da câmera 6
Figura 1.4: Modelo de sistema em camadas – Adaptado (MIRANDA NETO e RITTNER, 2006) 8
Figura 2.1: Taxonomia dos métodos de rastreamento FONTE (YILMAZ, JAVED e SHAH, 2006) 14
Figura 2.2: Tipos de Tracking: (a) Points Tracking (b) Kernel Tracking (c) Contour Tracking (d) Shape Tracking FONTE (YILMAZ, JAVED e SHAH, 2006) 15
Figura 2.3: Floating-point operation per Second and memory bandwidth for the CPU and GPU. FONTE(NVIDIA, 2010) 17
Figura 2.4: Arquitetura das CPU's e GPU's 18
Figura 2.5: Robô móvel utilizado no projeto - SRV-1 22
Figura 3.1: Forma da distribuição Gaussiana com média zero e desvio σ em 2-D 25
Figura 3.2: Determinação da reconstrução morfológica, sendo extraídos seus cumes. 29
Figura 3.3: Características do componente conexo: a) Sinal de entrada b) atributo de altura c) atributo de área d) atributo volume Fonte: Adaptado (DOUGHERTY e LOTUFO, 2003)... 30
Figura 3.4: Máximos Regionais de uma imagem em escala de cinza 31
Figura 3.5: Sinal 1D de entrada em azul e o sinal após o filtro reconstrutivo H-MIN em verde. 32
Figura 3.6: Interpretação física do Watershed: (a) Imagem Original (b) Linhas Watershed com os mínimos regionais (c) Representação topográfica da imagem original (d) sinal 1D entrada (e) resultado Watershed 1D 34
Figura 3.7: Áreas de duas superfícies 35
Figura 3.8: Discretização da superficie 36
Figura 3.9: Representação do centróide de uma Superfície 37
Figura 4.1: Relação das Streams com o Kernel. 41
Figura 4.2: Abstração entre "host" e "device", e estruturação dos dados para execução em paralelo 43
Figura 4.3: Simples comparativo de programação seqüencial e paralela 44
Figura 4.4: Arquitetura de memória no GPU 46
Figura 4.5: Janela de Vizinhança para um pixel 47
Figura 4.6: Fluxograma do algoritmo KNN paralelo em GPU 49
Figura 4.7: Modelagem KNN do Grid em Blocos (a)Imagem (b) Blocos do Grid 50
Figura 4.8: Exemplo do processamento paralelo para o filtro KNN 52
Figura 4.9: Fluxograma do algoritmo Sobel paralelo em GPU 54
Figura 4.10: Modelagem Sobel do Grid em Blocos (a)Imagem (b) Blocos do Grid 55
Figura 4.11: Processamento paralelo do filtro Sobel 56
Figura 4.12: Vizinhança do Pixel 57
Figura 4.13: Fluxograma do algoritmo HMIN paralelo em GPU (parte 1) 59
Figura 4.14: Fluxograma do algoritmo HMIN paralelo em GPU (parte 2) 60
Figura 4.15: Modelagem da borda do bloco pelo host 62
Figura 4.16: Estruturação de leitura e escrita do bloco com borda dentro do device 64

Figura 4.17: Calculo da Indexação dos blocos com borda 65
Figura 4.18: Cálculo do kernel RECONSTRUCAO 67
Figura 4.19: Resolução da lista de equivalência..... 69
Figura 4.20: Processo de influência de propagação do pixel mínimo 70
Figura 4.21: Execução do filtro morfológico reconstrutivo HMIN..... 71
Figura 4.22: Etapas do algoritmo Watershed 73
Figura 4.23: Fluxograma do algoritmo WATERSHED paralelo em GPU (parte 1)..... 75
Figura 4.24: Fluxograma do algoritmo WATERSHED paralelo em GPU (parte 2)..... 76
Figura 4.25: : Fluxograma do algoritmo WATERSHED paralelo em GPU (parte 3)..... 77
Figura 4.26: Imagem para aplicação do Watershed 78
Figura 4.27: Processamento do kernel INDEXDESCIDA a) Modelagem do grid e transferência dos dados b) Processamento do kernel c) transferência da memória registrador para a memória global. 79
Figura 4.28: Processamento do kernel PROPAGADESCIDA a) Modelagem do grid e transferência dos dados b) Processamento do kernel c) transferência da memória registrador para a memória global. 81
Figura 4.29: Passos da união dos mínimos conexos. (a) Determinação do vizinho com índice menor (b) Propagação das relações de vizinhança (c) União dos valores negativos e positivos que foram separados 82
Figura 4.30: Resultado do Watershed. (a) Ajustando os valores todos para positivo (b) Propagando o índice representativo pelos caminhos gerados. 84
Figura 5.1: Fluxograma da Camada de visão Computacional para o processo i..... 85
Figura 5.2: Resultado filtro KNN paralelo (a) Imagem de entrada (b) Imagem resultante do filtro KNN 87
Figura 5.3: Fluxograma do Kernel de conversão RGB para Escala de Cinza..... 88
Figura 5.4: Conversão RGB para Escala de Cinza. (a) Imagem RGB (b) Imagem Escala de Cinza 88
Figura 5.5: Resultado do Filtro Sobel Paralelo (a) imagem de entrada (b) Imagem resultante do filtro Sobel 89
Figura 5.6: Resultado do Filtro reconstrutivo HMIN paralelo (a) Imagem do filtro Sobel (b) Imagem resultante do filtro Hmin 89
Figura 5.7: Resultado da aplicação do filtro reconstrutivo HMIN para 90
Figura 5.8: Resultado Watershed contendo a média dos valores de intensidade RGB (a) Imagem de entrada (b) Imagem segmentada pela transformada Watershed 91
Figura 5.9 : Influência da alteração do pixel mínimo na transformada Watershed..... 92
Figura 5.10 : Resultado da influência do Filtro HMIN na constituição da segmentação com diferentes valores de H. (a) H=10 (b) H=30 (c) H=50 (d) H=100 93
Figura 5.11 : Centróide e média RGB das regiões (a) Imagem de entrada (b) Imagem contendo a média e centróide de cada região..... 95
Figura 5.12 : Transição de Domínios 96
Figura 5.13 : Fluxograma do Algoritmo de Rastreamento..... 97
Figura 5.14 : Processo iterativo de Rastreamento 98
Figura 6.1: Comandos para deslocamentos horizontais 101
Figura 6.2: Áreas de pertinência para a variável VP.x 101
Figura 6.3: referencial de câmera, eixo óptico não é paralelo ao eixo Z do ambiente 102
Figura 6.4: Área de pertinência para a variável VP.y..... 102

Figura 6.5: Base de regras para movimentação. (a) Relação das áreas de pertinência. (b) Tabela de movimentos associadas às áreas. 103
Figura 6.6: Extração da informação de Profundidade (a) Projeção perspectiva para diferentes distâncias em Z (b) Modelo perspectiva..... 104
Figura 6.7: Área de pertinência para a variável VA, abordado sobre o eixo Z 105
Figura 6.8: Arquitetura Funcional do Sistema..... 106
Figura 6.9: Fluxograma do Software desenvolvido..... 109
Figura 6.10: Funcionamento do sistema Concorrente e Paralelo 110
Figura 6.11: Execução do sincronismo entre imagens 111
Figura 7.1: Imagens de entrada para calculo do tempo de processamento da GPU. a) Imagem cameraman b) Imagem lena c) imagem baboon d) imagem peppers 113
Figura 7.2: Representação das regiões para análise. 118
Figura 7.3: Resultado da influência do parâmetro H no rastreamento. 120
Figura 7.4: Resultado da influência do parâmetro LimiarSimilaridadeCor no rastreamento..... 122
Figura 7.5: Resultado da influência do parâmetro LimiarDistancia no rastreamento. 123
Figura 7.6: Rastreamento da mão com parâmetros devidamente ajustados. 124
Figura 7.7: Simulação em ambiente interno, “Visão” do robô..... 125
Figura 7.8: Simulação em ambiente interno, corredor influenciado pelo sol..... 126
Figura 7.9: Simulação em ambiente externo, com fundo semelhante ao alvo. 127

Lista de Tabelas

Tabela 2.1: Categorias de Rastreamento FONTE: Adaptado(YILMAZ, JAVED e SHAH, 2006) 16
Tabela 4.1: Tipos de acesso na memória do GPU 46
Tabela 6.1: Relação dos movimentos com os comandos do robô 108
Tabela 7.1: Tempo de transferência dos dados na CPU para a GPU. 113
Tabela 7.2: Tempo de processamento do filtro KNN em GPU 114
Tabela 7.3: Tempo de processamento para converter RGB para Escala de Cinza 114
Tabela 7.4: Tempo de processamento para o filtro Sobel 114
Tabela 7.5: Tempo de processamento para o filtro reconstrutivo HMIN 115
Tabela 7.6: Tempo de processamento para a transformada Watershed 115
Tabela 7.7: Tempo total de processamento na GPU 116
Tabela 7.8: Análise qualitativa do parâmetro H. 119
Tabela 7.9: Análise qualitativa do parâmetro LimiarSimilaridadeCor 121

SUMÁRIO

INTRODUÇÃO **1**
1.1 Motivação 2
1.2 Proposta do Trabalho 7
1.3 Organização da dissertação 9

REVISÃO BIBLIOGRÁFICA **10**
2.1 Sistema de Visão Computacional 10
2.2 Rastreamento de Objeto 12
2.3 GPGPU 16
2.4 Sistema Robótico 20

MÉTODOS DE PROCESSAMENTO EM VISÃO **23**
3.1 Filtro Gaussiano KNN 24
3.2 Filtro Gradiente Sobel 26
3.3 Reconstrução Morfológica "H-MIN" 28
3.4 Transformada Watershed 32
3.5 Cálculo de Características 35

PROCESSAMENTO PARALELO PARA TRATAMENTO DE IMAGENS **38**
4.1 Paradigma de Programação Paralela 39
4.2 CUDA 41
4.3 Filtro Gaussiano KNN 47
4.4 Filtro Gradiente Sobel 53
4.5 Reconstrução Morfológica "H-MIN" 57
4.6 Transformada Watershed 72

ESTRATÉGIA DA VISÃO COMPUTACIONAL **85**
5.1 Pré-Processamento de Imagem 86
5.2 Segmentação de Imagem 90
5.3 Extração de Característica 93
5.4 Correspondência do alvo entre Imagens 95

INTEGRAÇÃO DO SISTEMA DESENVOLVIDO **99**
6.1 Estratégia de Controle e Navegação 99
6.2 Plataforma de Desenvolvimento 106
6.2.1 Servidor *107*
6.2.2 Comunicação *107*
6.2.3 Eletrônica embarcada *108*
6.3 Comandos e Transmissão 108

6.4 Software Desenvolvido 109

EXPERIMENTOS E RESULTADOS 112

7.1 Tempo de processamento na GPU 112

7.2 Rastreamento do Sistema de Visão Computacional 117

7.3 Simulações em Ambiente real 124

CONCLUSÃO E TRABALHOS FUTUROS 129

8.1 Comentários 129

8.2 Conclusão 130

8.3 Perspectivas Futuras 131

REFERÊNCIAS BIBLIOGRÁFICAS 133

APÊNDICE A - INFORMAÇÕES DA GPU 139

Capítulo 1

INTRODUÇÃO

Em áreas da pesquisa como robótica autônoma, o objetivo central é obter informações do ambiente através de sensores, interpretar os dados colhidos e executar ações através dos atuadores de modo a cumprir a tarefa que lhe foi designada. Nas palavras de SHIROMA (2004) a capacidade de um sistema robótico em realizar uma missão está intimamente associada ao grau de autonomia que este possui em relação a seu poder sensorial, a complexidade do ambiente onde está inserida, a capacidade em gerar decisões próprias bem como a necessidade de interação (intervenção) humana.

Observando tais aspectos, ao se desenvolver uma aplicação em tempo real onde o próprio ambiente em si já se caracteriza bastante complexo, a utilização de um ou mais sensores que proporcione um melhor detalhamento do ambiente é fundamental. Neste sentido, o uso da visão se mostra bem apropriado e promissor, devido à robustez (confiabilidade) na obtenção das informações caracterizando o ambiente desconhecido.

Desde o surgimento da visão computacional, muitos pesquisadores estão reunindo esforços para proporcionar à máquina a mesma habilidade que possui um sistema de visão humano, referenciando esta pesquisa como "visão de Máquina". Este termo é utilizado para descrever algum mecanismo que usa a visão como principal sensor para algum computador que funciona em tempo real (AWCOCK e THOMAS, 1996).

Em seu livro JAIN et al. (1995) coloca a visão de máquina como sendo caracterizada pelo somatório dos aspectos geométricos, de mensuração e de interpretação a partir de imagens. Modelando a visão como:

VISÃO = GEOMETRIA + MENSURAÇÃO + INTERPRETAÇÃO

Objetivando relacionar a visão computacional com a visão de máquina sobre o contexto da robótica, HARALIK e SHAPIRO (1993) descrevem a visão computacional como sendo

resultado da combinação de técnicas de processamento de imagens, reconhecimento de padrões e inteligência artificial aplicado na análise de uma ou mais imagens por meio do computador.

Um dos elementos chave no processo de automação em visão computacional para realizar a missão de seguir um objeto móvel, é seu rastreamento ao longo da seqüência de imagens.

O crescente interesse por sistemas de rastreamento para alvos móveis complexos em ambiente real, nos últimos anos, tem motivado grandes pesquisadores devido a sua natureza multidisciplinar, representando um enorme desafio. Muitas propostas estão sendo estudadas utilizando-se das mais variadas técnicas e tecnologias que podem ser usadas em extensas aplicações como veículos autônomos inteligentes, sistemas aeronáuticos, sistema antimíssil, videoconferência, conforme (BALKENIUS e KOPP, 1996), (LUO e CHEN, 2000), (OZYILDIZ, KRAHNSTOVER e SHARMA, 2002) e (CHEN, LUO e HSIAO, 1999). Dentre as diversas aplicações existentes, é notável o interesse das forças armadas brasileiras em apoiar um projeto com este foco.

1.1 Motivação

Em um noticiário publicado no jornal Valor Econômico por Virgínia Silveira (2010) evidenciou um estudo inédito realizado pelo Instituto de Pesquisa Econômica Aplicada (IPEA), apresentando o Brasil como um país que está investindo mais em projetos de Inovação na Área de Defesa. Esta pesquisa aponta que, nos últimos oito anos, a participação do setor de defesa nos desembolsos dos fundos setoriais cresceu cerca de 10%, tendo 258 de um total de 13.433 projetos analisados pelo IPEA que receberam apoio dos fundos, relacionados a este setor, conforme Figura 1.1.

Figura 1.1: Distribuição de verbas por área científica. FONTE (SILVEIRA, 2010)

Em entrevista com Fernanda De Negri, a qual ocupa o cargo de diretora-adjunta da Diretoria de Estudos Setoriais do IPEA responsável pela pesquisa sobre a participação do setor de defesa nos fundos setoriais, demonstra que os mecanismos de apoio à ciência, tecnologia e inovação no Brasil vem registrando um importante crescimento nos últimos quatro anos, obtendo um salto nos orçamentos para inovação de R$ 300 milhões para R$ 2 bilhões por ano. Este elevado crescimento orçamentário é motivado pelo novo interesse por parte do governo em avaliar tal importância para o desenvolvimento do país. Nas palavras de (Fernanda De Negri) *"O aumento dos investimentos do governo em projetos de defesa é resultado de uma nova percepção da importância desse setor para o desenvolvimento do país, pois várias dessas tecnologias têm aplicações que podem gerar importantes efeitos de transbordamento para o setor produtivo brasileiro"*.

Neste sentido o Ministério da Ciência e Tecnologia (MCT) elabora uma série de ações nesta área de defesa que enfatizam vários estudos relacionados com sistemas computacionais complexos, tecnologia de sensoriamento remoto, fabricação e emprego de propelentes e explosivos, veículos autônomos, estruturas resistentes e eficientes, sensores, ações de defesa

química, biológica e nuclear, e integralização de sistemas. Tais linhas de pesquisa poderão compor o plano de ação em Ciência, Tecnologia e Inovação.

Para melhor entendimento do potencial de um sistema de rastreamento de alvos complexos embasado no contexto acima, considere o seguinte cenário de aplicação militar e/ou para polícia federal.

Suponha-se que em uma dada região de fronteira, a união de interesses de segurança, econômico e de gestão política e ambiental se faz necessário seu monitoramento intensivo em tempo real. O emprego de veículos aéreos não tripulados (VANT) para realizar esta tarefa já se encontra em fase de testes, sendo controlados remotamente por operadores em bases terrestres (MENDES e FADEL, 2009). Tais VANT´s são dotados de potentes câmeras que permitem a visualização em grandes altitudes da movimentação de veículos e pedestres, e em conjunto com os veículos terrestres não tripulados (VTNT), podem constituir uma excelente ferramenta de controle da região de fronteira. A Figura 1.2 apresenta tal cenário:

Figura 1.2: Cenário de operação dos Veículos autônomos

Na configuração do ambiente atual, este tipo de tecnologia servirá para o monitoramento de fronteira do Brasil com seus vizinhos, onde auxiliará nos trabalhos de repressão a atos ilícitos ao longo da faixa. Nesta perspectiva, caso haja uma movimentação suspeita a ser rastreada, existe a necessidade de o operador controlar e seguir tal alvo, podendo ser qualquer objeto móvel como veículos ou pedestres. Com isso, as informações do VANT poderiam servir como estratégia de

rota para um dado comboio militar que teria um veículo autônomo “mestre” a efetuar a missão de se dirigir para o local desejado, juntamente com outros veículos “escravos” que o seguem. Nas palavras de PEREIRA (2005), enquanto que alguns módulos do sistema de operação autônomo são claramente aplicáveis atualmente neste cenário, por exemplo, outras envolvendo reconhecimento de objetos, percepção de situações, replanejamento autônomo e coordenação de tarefas até este momento estão em fase de estudos. Para ele, estas áreas ainda são objeto de I&D[1].

A partir da análise do exemplo acima citado, destacam-se duas aplicações com o mesmo objetivo. A primeira é caracterizada pelo VANT possuir a necessidade de um nível de autonomia maior ao ponto do operador simplesmente selecionar o alvo de interesse na imagem e este o seguir. De forma análoga, a segunda aplicação diz respeito ao comboio, onde cada veículo teria a capacidade de rastrear e seguir o outro veículo a sua frente, de maneira a não colidir.

Outro propósito o qual poderia ser citado é o carro autônomo controlado pelos olhos do motorista. Estudos recentes publicado pela “Freie Universitat Berlin” em 23 de abril de 2010, apresenta um grupo de pesquisadores liderados pelo prof. RAÚL ROJAS (2010) que desenvolveram um sistema de rastreamento do movimento dos olhos. Este software é capaz de dizer onde o condutor está focando seu olhar, e a partir disso inferir os sinais de controle ao sistema de direção do veículo, sendo que ao olhar para a direita o volante irá virar para direita, e ao olhar para esquerda, este irá virar para esquerda.

O sistema é equipado com 2 câmeras, e um infravermelho acoplado em um capacete (vide Figura 1.3.a). Uma das câmeras está virada para captar o movimento do olho, e a outra apontada para mesma direção que os olhos estão enxergando. Existe um espelho transparente infravermelho para auxiliar a captura da imagem sem limitar o campo de visão do condutor.

[1] I&D: ”Investigação & Desenvolvimento”

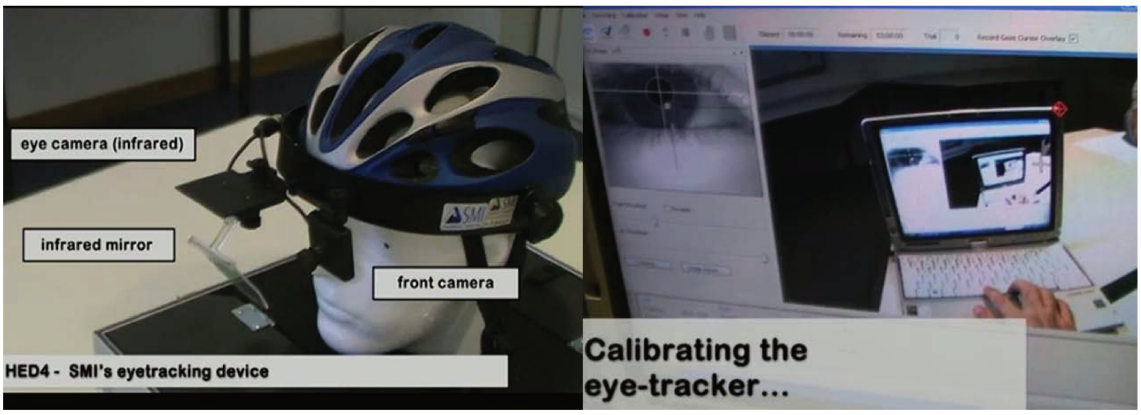


Figura 1.3: Sistema de Rastreamento. (a) Protótipo do equipamento (b) Fase de calibração da câmera

Após uma breve calibração apresentada na Figura 1.3.b, o rastreamento se faz pela pupila, onde o ângulo de direção para controle está referenciado a partir do centro da imagem. O equipamento foi inserido em um veículo "Dodge Grand Caravan 2000" batizado como espírito de Berlim. Este por sua vez possui complexos equipamentos para navegação autônoma. A idéia deste estudo é proporcionar um nível de liberdade ainda maior para pessoas debilitadas por algum motivo.

A partir da análise dos exemplos apresentados, independente do tipo de aplicação, destacamos que, para um veículo autônomo realizar uma tarefa, se faz necessário a coleta de informações sobre o ambiente através de sensores e atuadores, os quais não fornecem medidas totalmente exatas, sendo susceptíveis a falhas e ruídos. Deste modo, ao utilizar vários sensores para compor a "imagem" do ambiente, pode-se gerar maior confiabilidade ao se realizar o cruzamento dos dados, aumentando o grau de autonomia e "inteligência" do veículo. Em contrapartida, o tipo e a quantidade de sensores determinam o volume de processamento destas informações, impondo na maioria dos casos, um alto custo computacional que pode inviabilizar a aplicação do sistema em tempo real (MIRANDA NETO, 2007).

1.2 Proposta do Trabalho

A abordagem deste projeto é então desenvolver um sistema de rastreamento para alvos complexos[2] em ambiente desconhecido[3] gerando o controle de navegação para o robô autônomo terrestre seguir seu objetivo, sendo extraídas as informações do ambiente apenas com sistema de visão monocular.

Deve-se ressaltar que o sistema desenvolvido não necessita de nenhuma calibração prévia da câmera nem alguma forma geométrica específica para rastreamento do alvo, e utiliza somente sistema de visão local, desconsiderando informações globais do ambiente.

Partindo do sistema de rastreamento em mono-visão, é proposto o desenvolvimento das técnicas de processamento de imagem envolvendo o filtro de suavização KNN para eliminar ruídos na imagem proporcionada pela captura da câmera. A aplicação do filtro derivativo Sobel para detectar as bordas, descontinuidades dos valores de intensidade luminosa da imagem, visualizando os contornos dos objetos. A utilização do filtro reconstrutivo morfológico para eliminar os pixels mínimos da imagem Sobel, controlando a característica de segmentação de uma imagem. E a transformada Watershed para segmentação da imagem em partes constituintes.

Notando algumas exigências como o custo computacional para realização em tempo real, e apesar de trabalhar com um único sensor, este por si só caracteriza um custo computacional bastante elevado ao se trabalhar com tais filtros descritos anteriormente. Neste sentido, devido à sua arquitetura, as GPUs (graphics processing units) são excelentes alternativas para implementação do processamento de imagens, porque permitem tratar adequadamente e em paralelo, frames e pixels como entidades independentes. Além disso, como são dispositivos dedicados a processamento gráfico, tem recursos em hardware, altamente otimizados, para lidar com imagens. Por esta razão, este trabalho contempla essa nova tecnologia de processamento em hardware gráfico.

Avaliando o grau de autonomia deste sistema, o objetivo é integrar em um único software, técnicas de programação concorrente para gerenciamento de processos, programação paralela para gerenciamento dos coprocessadores GPU´s e programação paralela para modelagem SIMD.

---

[2] *Alvos complexos:* Caracteriza-se por algum objeto, corpo não rígido, que não possui um padrão geométrico específico, bem como é constituído por diversas características distintas.
[3] *Ambiente desconhecido:* Caracteriza-se por um ambiente não estruturado de natura dinâmica.

De forma geral, baseado nas áreas de processamento de imagem, programação paralela, visão computacional e sistema distribuído. Os componentes são:

- Monovisão;
- Lógica clássica para controle e Navegação;
- Multiprocessado;
- Arquitetura Cliente-Servidor e de Sistema Distribuído;
- Multicamadas independentes entre si;
- De tempo real.

Para melhor organização da complexidade do software, será aplicado um modelo de abstração de sistema em camadas (visto na Figura 1.4), permitindo a implementação de maneira a adequar nas novas gerações de computadores multiprocessados, uma arquitetura de software voltada para sistema distribuído e de Cliente-Servidor, proposta por MIRANDA NETO e RITTNER (2006).

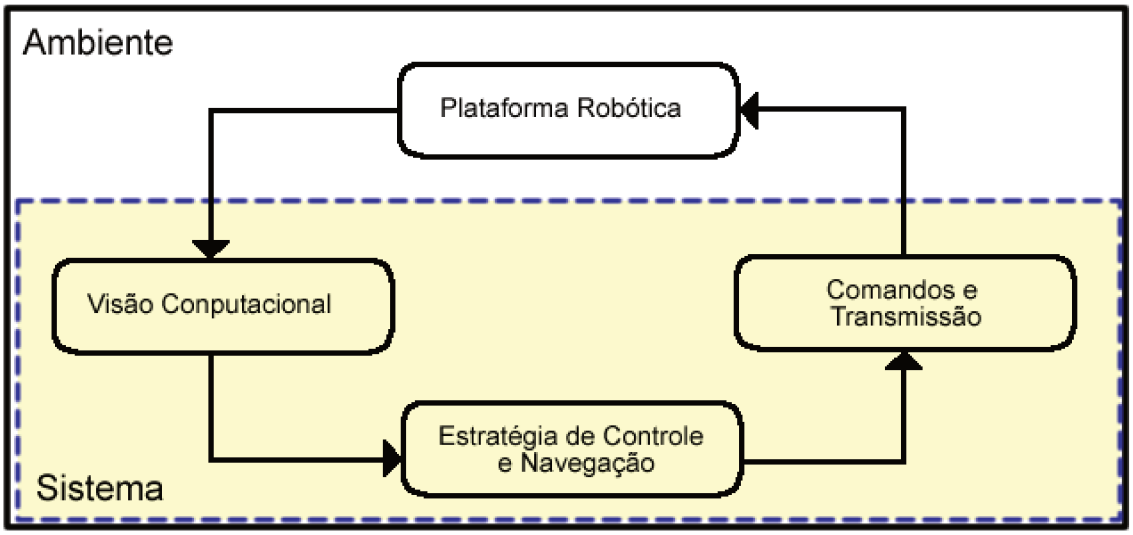


Figura 1.4: Modelo de sistema em camadas – Adaptado (MIRANDA NETO e RITTNER, 2006)

1.3 Organização da dissertação

A dissertação encontra-se dividida da seguinte forma:

No capítulo 2, é apresentada a revisão da literatura que fundamenta este trabalho, tendo como principais temas: Rastreamento de Alvos Complexos, Visão computacional, programação paralela em GPU, e sistema robótico.

Logo a seguir, no capítulo 3, são apresentadas as ferramentas de processamento de imagens aqui utilizadas no sistema de visão computacional, trazendo seus conceitos e definições;

Adiante, no capítulo 4, são abordadas as técnicas de processamento de imagens que foram desenvolvidas em paralelo nos processadores gráficos, apresentando o raciocínio e a lógica de programação paralela aplicada.

O capítulo 5 apresenta-se o sistema de Visão Computacional desenvolvido, onde são estruturadas as fases de pré-processamento, segmentação, extração de características e correspondência da região alvo entre a seqüência de imagens.

O capítulo 6 apresenta-se a integração do Software, abordando a modelagem da camada de Estratégia de Controle e Navegação, a camada de Comandos e Transmissão, bem como a arquitetura do sistema físico e a concepção de desenvolvimento do software.

Posteriormente no capítulo 7, apresentam-se os experimentos e resultados obtidos, avaliando as influências de alguns parâmetros na determinação do rastreamento.

Por fim, no capítulo 8, são apresentadas as conclusões deste trabalho, bem como algumas propostas de trabalhos futuros.

Capítulo 2

REVISÃO BIBLIOGRÁFICA

Neste capítulo é apresentada a fundamentação teórica para o rastreamento de alvos complexos em imagens, objetivando buscar na literatura os principais métodos de solução nas áreas de segmentação e tracking em visão computacional, programação paralela com GPU e a plataforma robótica utilizada.

2.1 Sistema de Visão Computacional

De acordo com a definição de SHAPIRO e STOCKMAN (2001), a visão computacional é útil para tomar decisões sobre objetos físicos reais e cenas baseadas em captura de imagens. De forma a tomar decisões a respeito de objetos reais, quase sempre se faz necessário construir alguma descrição ou modelo deste objeto a partir da imagem. Por causa disto, muitos especialistas dizem que o objetivo da visão computacional é a construção de descrições de cenas a partir de imagens.

Em um ponto de vista bastante semelhante, para SEBE et al. (2005), o objetivo da visão computacional é prover aos computadores uma capacidade de percepção semelhante à humana, de forma que estes possam perceber o ambiente, entender os dados percebidos, tomar ações apropriadas e aprender com estas experiências, de forma a aprimorar desempenhos futuros.

Como citado anteriormente, o sucesso de uma tarefa é caracterizado pela capacidade de percepção que o sistema possui em mapear o ambiente em que se encontra, modelando a "imagem" do ambiente com os diversos tipos de sensores incorporados a plataforma móvel. Nesta perspectiva, onde o foco é atuar em tempo real e obter satisfatório desempenho em ambiente dinâmico, o custo computacional está intimamente relacionado ao número de sensores utilizados para mapeamento do ambiente, podendo comprometer a execução da aplicação em tempo real.

Observando as diversas aplicações para controle de plataforma robótica móvel, BERTOZZI et. al. (2000) no contexto de veiculo autônomo, caracteriza o sistema de visão computacional como parte integrante do conjunto de sensores, apresentando bons resultados em ambientes indoors e outdoors. A fim de atender aos requisitos de tempo real, optou-se por utilizar somente uma câmera monocular para compor o sistema de sensoriamento. Apesar de se utilizar um único sensor, este por si só vai demandar significativo tempo de processamento movido pela alta taxa de captura/leitura de dados.

Para MIRANDA NETO (2007), a quantidade de imagens geradas por segundo, submetidas ao sistema de visão, pode desencadear prejuízos como a demanda no tempo de processamento para análise da visão. No sentido de contornar esta questão, ele propôs um método de descarte redundante de imagens baseado na correlação de Pearson. Este método fornece um valor de similaridade entre duas imagens, variando o valor entre -1 (para imagens oposta, negada) e 1 (para imagens totalmente similares). Através de um limiar escolhido, é feito o descarte da imagem. A técnica apresentada é bastante robusta, no entanto, sua utilização em ambientes com alta variação da cena em um instante de tempo se torna inviável. Outros métodos robustos para serem aplicados em tempo real, podem ser encontrados como descarte aleatório de imagens e bases de armazenamento. O conceito abordado neste trabalho para eliminar este problema é aumentar o poder de processamento do computador nos casos em que não é possível realizar o descarte das imagens. Para isso, usa-se o GPU, onde será discutido mais adiante na seção 2.3.

Conforme o sistema de visão biológico, a maioria dos sistemas de visão computacional tem evoluído para lidar com o mundo em mudanças. Neste sentido, um sistema deve ser habilitado a lidar com alterações do ambiente como o movimento e variação dos objetos, iluminação e variações do ponto de vista da câmera. Tais mudanças são fatores essenciais em diversos tipos de aplicações que envolvam a análise dinâmica de cenas. (JAIN, KASTURI e SCHUNCK, 1995).

O sistema de análise em cenas dinâmicas é modelado sobre a seqüência de frames extraídos do ambiente em movimento, onde a câmera também pode estar em movimento. Cada frame representa uma imagem da cena em um dado instante de tempo. Neste instante, existem várias mudanças que podem ocorrer como o movimento da câmera, movimento dos objetos, variação da iluminação, variação na estrutura, tamanho e forma dos objetos. De fato, todas essas mudanças estão sendo projetadas na imagem da cena, que por sua vez depende da posição da

câmera. Em função dessa relação de câmera e ambiente, JAIN et.al. (1995), apresenta 4 possibilidades para natureza dinâmica da visão:

1 – Câmera Parada, Objeto Parado. (CPOP)
2 – Câmera Parada, Objeto em Movimento. (CPOM)
3 – Câmera em Movimento, Objeto Parado. (CMOP)
4 – Câmera em Movimento, Objeto em Movimento. (CMOM)

Para cada caso apresentado anteriormente, existem diferentes técnicas que melhor se adaptam a resolução da tarefa. Basicamente, este estudo contempla a natureza dinâmica embasada no CMOM, pois o rastreio do alvo se dará pela câmera anexada na plataforma móvel, onde a câmera e o objeto estarão em movimento. Para JAIN et al. (1995) este caso é o mais geral e, possivelmente a situação mais complicada na análise de cena dinâmica. Esta é a área da visão computacional que ao longo do tempo está recebendo mais desenvolvimento.

2.2 Rastreamento de Objeto

A visão é o principal mecanismo de detecção dos seres humanos (HORN, 1986). De forma biológica, o rastreamento visual de um objeto em movimento por um observador humano caracteriza-se pelo acompanhamento do movimento do objeto pelo seu sistema de visão, sendo que o cérebro processa as imagens do objeto que foram captadas pelo sistema de visão de modo a seguir a trajetória descrita (VIDAL, 2009).

A tarefa de rastreamento em si, consiste em gerar e acompanhar a trajetória de um objeto em movimento, computando seu deslocamento na seqüência de imagens (YILMAZ, 2007).

Para MOESLUND e GRANUM (2001), a área de pesquisa voltada para o rastreamento pode ser definida a partir de vários pontos de vista dependendo do contexto da aplicação, sendo esta área já bem estabelecida atualmente. No entanto, para YILMAZ et al. (2006), muitos destes rastreamentos foram desenvolvidos com propósitos de rastrear objetos em tempo real em cenários simples, assumindo alguns fatores como suavidade do movimento, quantidade mínima de

oclusão, iluminação constante, alto contraste do objeto com o plano de fundo, etc., os quais são violados em muitos cenários realísticos e, portanto limitam eficientes rastreamentos em aplicações de monitoramento de tráfico, navegação de veículos, vigilância automatizada, interação homem-máquina, etc. Seguindo o raciocínio de YILMAZ et al. (2006), rastreamento e seus problemas associados à seleção de características, representação de objetos, dinâmica da forma e estimação do movimento ainda são áreas ativas da pesquisa onde a cada dia novas soluções estão sendo propostas.

Nas palavras de YILMAZ et al. (2006) o nível de complexidade do problema de rastreamento é gerado pela:

- Perda de informação causada pela projeção do mundo 3D em imagem 2D;
- Ruído na imagem;
- Complexidade do movimento do objeto;
- A não rigidez ou natureza articulada dos objetos;
- Parcial ou completa oclusão do objeto;
- Forma complexa do objeto;
- Variação na luminosidade da cena;
- A necessidade de processamento em tempo real.

Naturalmente, esta complexidade vem sendo tratada conforme a evolução das novas tecnologias de computadores que dão suporte a processamentos mais pesados e intensos que por sua vez, proporciona a incorporação de variáveis no modelo de rastreamento objetivando eliminar tais problemas e aproximar a aplicação para o cenário real, tornando o processo robusto e aplicável. Adicionalmente, dependendo do domínio, o rastreamento pode gerar informações relevantes para posterior controle como orientação, área, forma e centróide do objeto em questão.

Para modelar um processo de rastreamento, muitos autores utilizam diferentes mecanismos como Kalman Filter (BROIDA e CHELLAPA, 1986), Mean Shift (COMANICIU, RAMESH e MEER, 2003), Histograma (KANG, COHEN e MEDIONI, 2004), etc. Todos estes métodos possuem características particulares que são apropriadas para rastreamento de objetos rígidos e não-rígidos.

Em seu trabalho YILMAZ et al. (2006) avaliando e estudando as principais técnicas para rastreamento de objetos propostas até então, constituiu uma taxonomia sobre os métodos existentes a fim de classificar e categorizar cada técnica, vista na Figura 2.1:

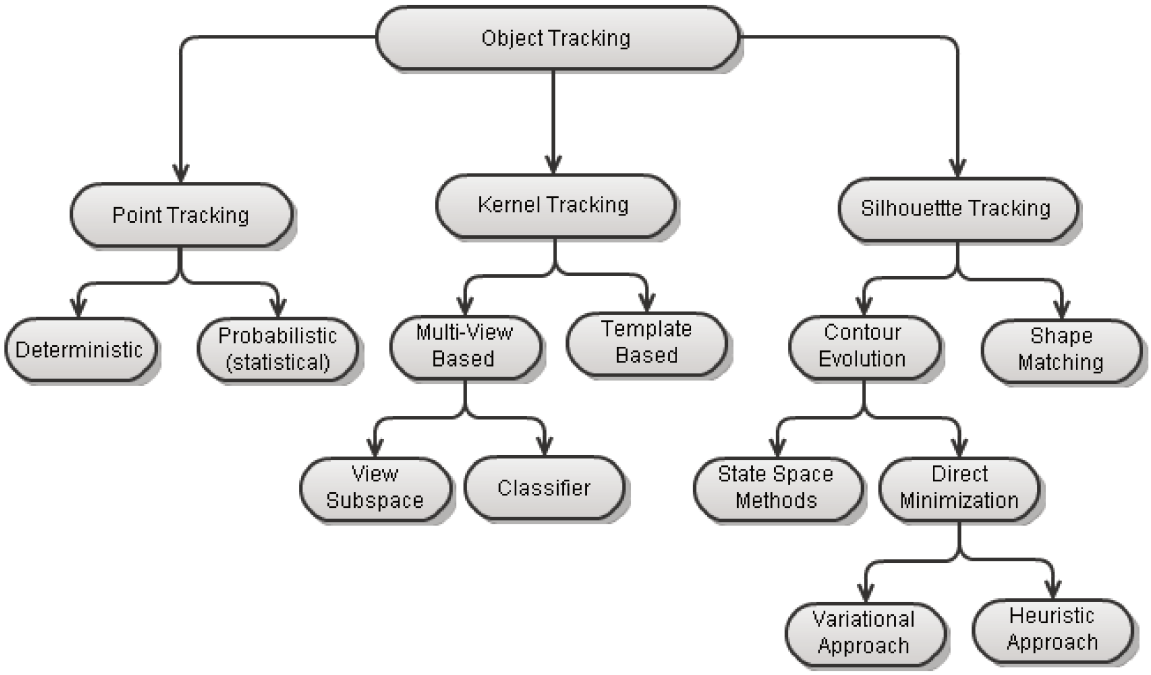


Figura 2.1: Taxonomia dos métodos de rastreamento FONTE (YILMAZ, JAVED e SHAH, 2006)

Deste modo, foram constituídas 3 principais categorias abrangendo todos os trabalhos relatados. De maneira resumida, YILMAZ et al. (2006) apresenta:

- *Point Tracking:* Os objetos detectados são representados por pontos, onde a associação dos pontos é baseada no estado do objeto anterior, incluindo a posição e movimento deste. Tal método exige um mecanismo externo para estimar a próxima posição, podendo ser aplicado o filtro de Kalman ou filtro de partículas. (vide Figura 2.2(a)).

- *Kernel Tracking:* Kernel é caracterizado como uma determinada forma e aparência do objeto, por exemplo um template retangular ou uma forma elipsoidal com o histograma associado. O rastreamento do objeto é computado pelo movimento do kernel na seqüência de imagens, tendo este movimento em forma de transformações paramétricas como translação, rotação e escala. (Vide Figura 2.2(b)).

- *Silhouette Tracking:* Neste caso, o rastreamento é feito por estimar a região do objeto em cada frame. O método usa alguma informação contida na região do objeto. Tal informação pode ser a densidade de aparência e forma do modelo, como exemplo a borda da região. Dado um modelo de objeto, sua silueta é rastreada tanto por correspondência de forma quanto por evolução de contorno. Estes 2 métodos podem ser considerados como segmentação de objetos aplicados em um domínio temporal usando a informação gerada no frame anterior. (Vide Figura 2.2(e) e Figura 2.2(d)).

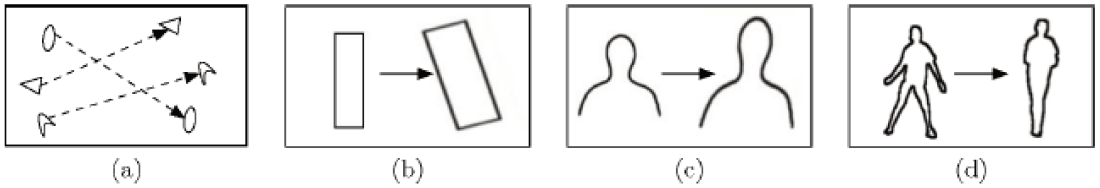


Figura 2.2: Tipos de Tracking: (a) Points Tracking (b) Kernel Tracking (c) Contour Tracking (d) Shape Tracking FONTE (YILMAZ, JAVED e SHAH, 2006)

Considerando os aspectos apresentados nesta seção e com base no estudo realizado por (YILMAZ, JAVED e SHAH, 2006) onde apresenta alguns representativos trabalhos para cada categoria (vide Tabela 2.1), nota-se que tais categorias não surgiram de maneira heurística. Ao observar as épocas que os trabalhos foram desenvolvidos e publicados, existe ali uma evolução nítida do interesse em buscar soluções que não se restringe a problemas específicos e estruturados, objetivando alcançar sistemas reais, robustos e confiáveis. Uma das causas prováveis para proporcionar este fenômeno, gerado pelo aprimoramento das técnicas, mas também de maneira bem significativa, é o avanço da tecnologia voltada para o processamento computacional. Isto possibilita a utilização de modelos de rastreamento mais precisos com alcance mais generalizado.

Tabela 2.1: Categorias de Rastreamento FONTE: Adaptado(YILMAZ, JAVED e SHAH, 2006)

Categorias	Trabalho Representativo
Point Tracking	
Deterministic methods	MGE Tracker (Salari and Sethi 1990) GOA Tracker (Veenman et al. 2001)
Statistical methods	Kalman Filter (Broida and Chellappa 1986) JPDAF(Bar-Shlom and Foreman 1988) PMHT (Streit and Luginbuhl 1994)
Kernel Tracking	
Template and density based appearance models	Mean-Shift (Comaniciu et al. 2003) KLT (Shi and Tomasi 1994) Layering (Tao et al. 2002)
Multi-view appearance models	Eigentracking (Black and Jepson 1998) SVM traker (Avidan 2001)
Silhouette Tracking	
Contour evolution	State space models (Isard and Blake 1998) Variational methods (Bertalmio et al. 2000) Heuristic methods (Ronfard 1994)
Matching shapes	Hausdorff (Huttenlocher et al. 1993) Hough Transform (Sato and Aggarwal 2004) Histogram (Kang et al. 2004)

Ainda dentro deste escopo de evolução, existem métodos que está relacionado com a união de técnicas, como por exemplo, em CAVALLARO et al. (2002) que utiliza um modelo híbrido entre rastreamento por região (region-based) e por características (feature-based) e ainda vincula uma técnica de estimativa linear para determinar o rastreamento no frame seguinte.

Outro método que está recebendo grande atenção de vários pesquisadores refere-se aos métodos diretos. Esta técnica trabalha diretamente com o valor de intensidade luminosa do pixel, sem extrair características da imagem para realizar o tracking (IRANI e ANANDAN, 2000) (SILVEIRA e MALIS, 2010).

2.3 GPGPU

Os recentes avanços tecnológicos e a natureza paralela das operações envolvidas no processamento de imagens em tempo real transformaram as placas gráficas atuais em máquinas com grande poder computacional paralelo. Os hardwares de processamento gráfico, chamados

Graphics Processing Units - GPU´s , são provavelmente as placas com a melhor relação entre custo e desempenho da atualidade (BAGGIO, 2007) e (AMORIM, 2009).

Estudo realizado pela NVIDIA (2010), apresenta um comparativo entre os processadores fabricados por ela frente aos processadores da Intel, apresentando a evolução do desempenho em FLOPS (floating point operation per second) durante os anos de 2003 a 2008, bem como seu Bandwidth quantificando a taxa de transferência de dados, conforme Figura 2.3.

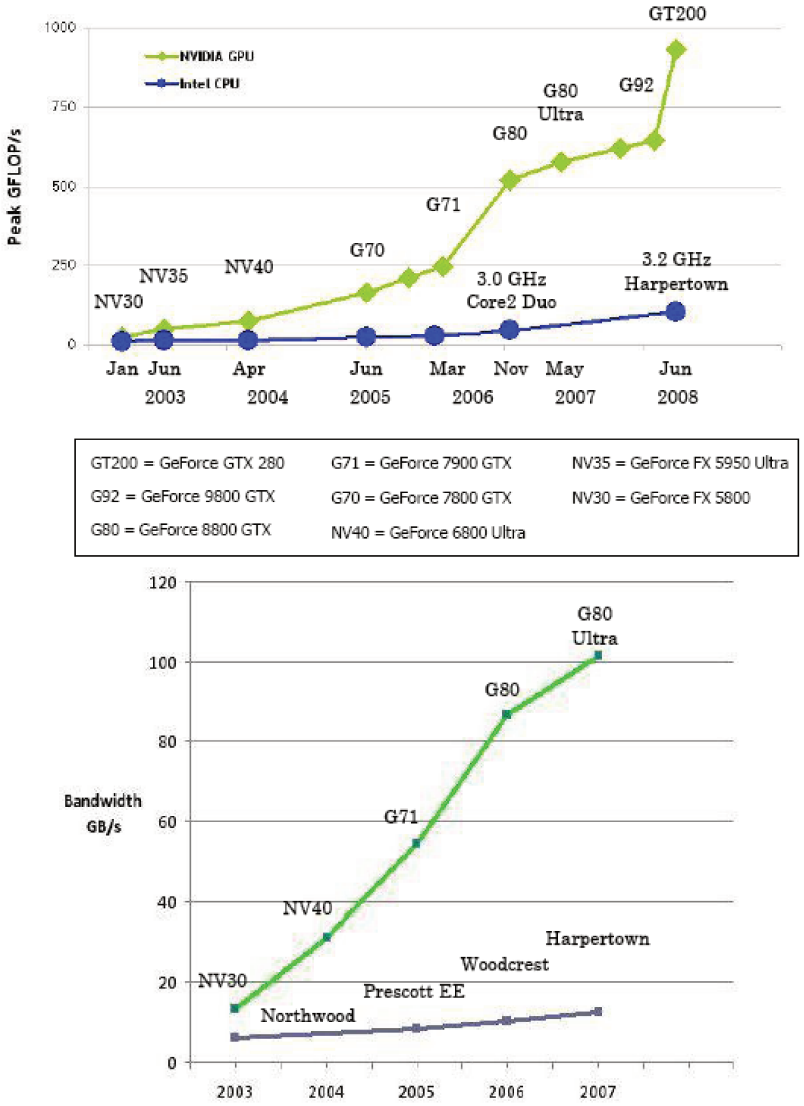


Figura 2.3: Floating-point operation per Second and memory bandwidth for the CPU and GPU. FONTE(NVIDIA, 2010)

Visando entender a discrepância no desempenho observado no gráfico, OWENS et al. (2005), explica que tal fato pode ser atribuído a diferença na arquitetura, onde o propósito das CPU´s são otimizadas para alta performance e código serial, tendo muitos dos seus transistores dedicados a suportar tarefas não computacionais. Por outro lado, a natureza altamente paralela da computação gráfica habilita as GPU´s usarem adicionais transistores para processamento, alcançando um desempenho melhor no processamento. Deve-se considerar também, que, os processadores gráficos surgiram da demanda por processamento em tempo real e por gráficos 3D de alta definição gerada principalmente pela indústria de jogos eletrônicos, processamento de vídeos e animações computacionais. A Figura 2.4 mostra uma análise comparativa da arquitetura entre CPU e GPU.

Figura 2.4: Arquitetura das CPU's e GPU's

Observe que a área do chip da GPU destinada aos cálculos matemáticos (ALU) é muito maior do que a encontrada na CPU. Entretanto, o controle de fluxo e a memória cache da GPU são extremamente pequenos, existindo somente para cumprir as tarefas básicas de gerenciamento das tarefas no chip. Com isto, pode se dizer que a CPU e a GPU são tecnologias complementares, cada uma destinada a um fim diferente.

Em virtude de sua promessa de computação altamente paralela, de maneira SIMD (Single Instruction Multiple Data), a programação de GPU´s para computação de propósito geral está se tornando bastante popular, conhecida como GPGPU (General Purpose programming using Graphic Processing Unit).

Ainda que esta idéia apresenta-se como nova, em seu trabalho OWENS et. al. (2007), menciona algumas pesquisas que foram desenvolvidas de 20 anos pra cá, como planejamento de movimentação de robô feito por Lengyel et. al. em 1990, implementação de redes neurais por

Bohn em 1998, cálculo de diagramas de voronoi realizado por Hoff et. al. em 1999 entre outros. Estes são considerados os precursores da idéia GPGPU, ressaltando que nesta época as GPU´s não possuíam hardware programável, e mesmo assim eles conseguiram modelar seus problemas para processar em placas gráficas.

Enquanto os designs no hardware de GPU evoluíam em direção a unificação de mais processadores, eles aumentavam a relação de semelhança com os computadores de alta performance. Esta notícia motivou pesquisadores a explorar o uso das GPUs para resolver problemas de engenharia e problemas com intensivo processamento (KIRK e HWU, 2010).

Durante a evolução dos hardwares, houve várias implementações desenvolvidas na área de processamento de imagens mais específico na segmentação de imagens, motivos pela necessidade de obter aplicações mais rápidas voltadas para análise de imagens médicas. Seguindo esta perspectiva, têm-se alguns dos principais projetos desenvolvidos focados nas áreas de processamento de imagens e visão computacional.

Dentre eles, YANG e WELCH (2003), usaram a combinação de registradores para calcular o thresholding e uma convolução sobre uma imagem colorida. Thresholding é uma maneira de determinar se um dado pixel está contido ou não em uma dada região à segmentar, baseada no valor da intensidade luminosa do pixel. Sua implementação demonstrou um ganho de 30% na velocidade, comparando a GPU NVIDIA GeForce4 com o CPU Pentium 4 2.2Ghz Intel.

Em seu trabalho, GRIESSER et. al. (2005), concentrou especificamente em segmentação de foreground-background em GPU, utilizando um teste de similaridade de cores no pixel e seus vizinhos, integrado dentro de um estimador bayesiano. O tempo de processamento do experimento apresentou-se na ordem de 4ms para imagens de resolução 640x480 sobre a NVIDIA GeForce 6800GT.

Uma importante biblioteca focada em visão computacional e processamento de imagens portado para GPU é a GPUCV (FARRUGIA, HORAIN, *et al.*, 2006). Esta biblioteca é uma extensão da biblioteca OpenCV da Intel com intuito de acelerar as aplicações escritas com OpenCV sem modificação do código. A GPUCV trabalha com programação em OpenGL e GLSL. Dentre suas funções, possuí conversão de cor, histograma e operações morfológicas como erosão e dilatação.

Outros trabalhos nesta área são, a correção de distorção radial de uma imagem, rastreamento da mão e cálculo do vetor de características de imagens os quais podem ser vistos

em (PHARR e FERNANDO, 2005) tendo estas técnicas encapsuladas em uma biblioteca chamada OpenVidia. Outros trabalhos compreendem a rotulação de componentes conexos (HAWICK, LEIST e PLAYNE, 2009) e detector de bordas Canny (LUO e DURAISWAMI, 2008). Tais trabalhos exploram de maneira satisfatória a utilização desta arquitetura.

Acompanhando o desenvolvimento da tecnologia das GPU´s, sentiu-se a necessidade de desenvolver algum tipo de API (Application Programming Interface), ferramenta que viabilizasse a programação nesta arquitetura de forma mais simples, além da linguagem de propósito gráfico como OpenGL e DirectX. Pensando nisso, a NVIDIA Corporation lançou em 2006 a Compute Unified Device Architecture – CUDA. Trata-se de um framework de desenvolvimento de software voltada à computação massiva, paralela e de alto desempenho, utilizando o hardware contido nas placas gráficas da própria NVIDIA.

O presente projeto contempla esta abordagem CUDA para desenvolvimento dos códigos (onde um melhor entendimento do paradigma de programação será apresentado mais adiante na seção 14.2). No entanto, deve-se ressaltar que existem outras plataformas de programação, as quais não serão discutidas aqui, porém valem a pena serem citadas, como a linguagem BROOCK (BUCK, FOLEY, *et al.*, 2004) desenvolvida para ser portável para todos os hardwares gráficos programáveis, e mais recentemente o OpenCL (KHRONOS, 2008) bastante similar ao CUDA, que é o primeiro padrão aberto, prerrogativa livre para uso geral de programação paralela de sistemas heterogêneos. Além destes, BAGGIO (2007), apresenta de forma mais detalhada outras possibilidades como CTM, Sh, RapidMind.

2.4 Sistema Robótico

Segundo Tsai (1999), robôs podem ser classificados de acordo com vários critérios, como seus graus de liberdade, estrutura cinemática, tecnologia de acionamento, geometria do espaço de trabalho e características de movimento. Entretanto, SHIROMA (2004) diz que atualmente as classificações são compreendidas entre dois grupos fundamentais:

- **Robô Manipulador Industrial:** Definição ISO: “Robôs manipuladores Industriais são máquinas controladas automaticamente, reprogramáveis, multi-propósitos com diversos graus de

liberdade, que podem ser tanto fixas no local ou móvel para uso em aplicações de automação industrial".

- **Robô Móvel:** "Um robô capaz de se locomover sobre uma superfície através da atuação de rodas montadas no robô em contato com a superfície (Muir, 1988). Embora hoje seja possível encontrar arquiteturas robóticas submergíveis, aéreas (Maeta, 2001) e até bípedes, esta definição continua válida para grande maioria de robôs móveis existentes."

Realizando uma analogia na classificação de Shiroma e Tsai, antigamente os robôs eram voltados mais para aplicações industriais e, portanto, sua classificação era focada nas características apresentadas por Tsai, sendo que atualmente todas elas foram atribuídas a uma mesma classe. Provavelmente este fato se deve a evolução da robótica móvel, enfatizando novas concepções de classificação proporcionada principalmente por três atributos ou razões apresentadas na literatura pelos "3 D´s" – Dull, Dirty, Dangerous – que liberam os seres humanos de eventuais tarefas de riscos. Neste sentido, cada vez mais robôs móveis dotados de inteligência estão sendo pesquisados e desenvolvidos com capacidades autônomas e de propósito gerais.

Para SHIROMA (2004), apesar das diferenças entre as classes, um sistema robótico podendo ele ser um braço manipulador, um veículo autônomo ou semi-autômono, é constituído das seguintes partes:

- Sistema mecânico: A parte física do robô. Este encontra-se imerso em um dado ambiente real, podendo interagir com o meio.

- Sensores: Sentidos do robô. São aparelhos de medida que provêem as informações a partir do qual o estado do robô é determinado. Alguns tipos de sensores são: câmeras, lasers, GPS, etc.

- Inteligência: "O cérebro do robô. Responsável pela inteligência do sistema".

Compreendido o sistema robótico, deve-se entender que sua autonomia, isto é, que o sistema autônomo caracteriza-se por possuir capacidade de coleta de dados e informações, de interpretação destes e de síntese de estratégias de atuação que permitem assegurar a execução de

atividades de complexidade típicas daquelas que requerem a participação do ser humano, mas sem a intervenção deste. (PEREIRA, 2005).

Ressalta-se por CAZANGI (2004) e MIRANDA NETO (2007) que, ao afirmar que o sistema é autônomo, não necessariamente ele será completamente autônomo, mas sim que ele detém certo nível de autonomia, sendo que, quanto maior for sua autonomia, menos auxílio externo ele necessitará.

Para desenvolvimento deste projeto, utilizou-se o robô móvel SRV-1, caracterizado por ser um sistema de robótica móvel avançado cujos componentes principais compreendem sistemas de controle, comunicação, sensoriamento e tecnologias de programação. A Figura 2.5 apresenta o robô.

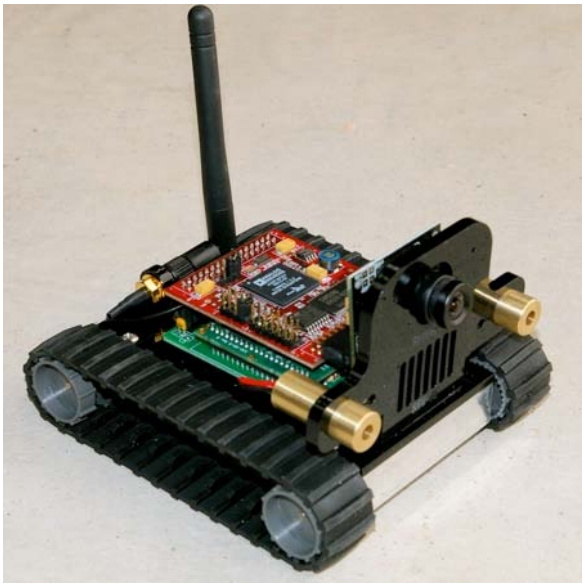


Figura 2.5: Robô móvel utilizado no projeto - SRV-1

Capítulo 3

MÉTODOS DE PROCESSAMENTO EM VISÃO

Neste capítulo é apresentada a fundamentação teórica de processamento de imagens aplicado na visão computacional, sendo abordados as técnicas e métodos utilizados neste projeto. De acordo com GONZALEZ e WOODS (2000), o objetivo da visão computacional é processar dados de imagens para percepção de máquina, além da informação visual.

Deste modo, diversas operações sobre a imagem, como processamento de cores, redução de ruídos, realce de detalhes na imagem, segmentação, extração de informações, classificação dentre outras, podem compor os algoritmos da visão computacional. As principais atividades que compreende o sistema de processamento são determinadas pelas etapas de aquisição, pré-processamento, segmentação e classificação (GONZALEZ e WOODS, 2000).

Na etapa de aquisição de imagens, realizada por algum dispositivo de visão como a câmera, por exemplo, dois elementos são fundamentais: um dispositivo físico que produza um sinal elétrico de saída proporcional a um nível de energia percebida, e um digitalizador que converta a saída elétrica deste sensoriamento para a forma digital (MIRANDA NETO, 2007).

O trabalho de melhoramento das imagens é determinado pelo pré-processamento, onde os filtros podem ser aplicados tanto no domínio espacial quanto no domínio da freqüência. Alguns dos principais métodos destacam-se os filtros de suavização de imagens, utilizados geralmente para eliminação de ruídos, os filtros conhecidos como passa-baixa e passa-alta para abrandar e enfatizar respectivamente as mudanças de níveis de cinza associados às freqüências, sendo utilizados para realce de imagens.

O processo de segmentação tem por finalidade dividir uma imagem digital em partes constituintes que servem para facilitar a análise da imagem. Tais partes podem ser conjuntos de regiões ou objetos que referenciam alguma característica similar, tais como cor, intensidade luminosa, textura ou continuidade.

A etapa de classificação consiste em dizer qual das regiões segmentadas faz parte do objeto e qual faz parte do fundo. De forma generalizada, a classificação se dá em função de

algum objetivo que se quer atingir, como classificação das partes que constitui uma cadeira, pessoas, etc.

Observando a contextualização das etapas de processamento, nos tópicos seguintes serão focados de maneira mais detalhada, os métodos utilizados na fase de pré-processamento e segmentação.

3.1 Filtro Gaussiano KNN

O uso de filtros digitais no domínio espacial tem como conseqüência a variação no valor digital de um pixel da imagem original, segundo a influência de seus pixels vizinhos. Segundo GONZALEZ e WOODS (2002), o processo de redução de ruídos em imagens pode ser suavizado pela alteração dos pixels em função da média dos níveis de intensidade luminosa de seus vizinhos, definido por uma máscara, janela de vizinhança.

Filtros de suavização têm por conseqüência a redução das freqüências altas, produzindo uma homogeneização geral da imagem, podendo ter diferentes resultados, dependendo dos pesos usados na janela de convolução e principalmente do tamanho desta janela. Para BUADES et al (2005), tais filtros, também conhecidos como filtros de vizinhança, são baseados na idéia de que todos os pixels ao longo de um mesmo objeto tem seus valores de níveis de cinza similares. Portanto, na existência de ruídos, a normalização pela média é uma boa alternativa.

Diversos tipos de máscaras podem ser empregados para realizar esta filtragem, neste trabalho utiliza-se a distribuição gaussiana para modelagem da janela de convolução, tendo como média zero e desvio padrão σ em duas dimensões é ilustrada na Figura 3.1 e descrita pela equação (3.1.1).

$$G(s,t) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{s^2+t^2}{2\sigma^2}} \tag{3.1.1}$$

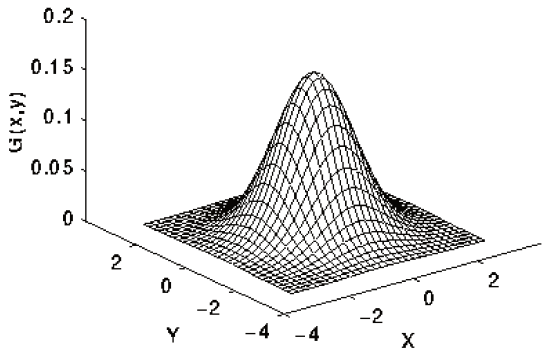


Figura 3.1: Forma da distribuição Gaussiana com média zero e desvio σ em 2-D

Uma vez que a máscara adequada tenha sido definida, a operação de suavização Gaussiana pode ser desenvolvida utilizando métodos de convolução. Esta operação pode ser realizada pela equação (3.1.2), sendo *R(x,y)* o resultado da convolução para o ponto *(x,y)* da imagem, *f(x+s,y+t)* o valor da intensidade luminosa da imagem no ponto vizinho de *(x,y)* e *G(s,t)* sendo o peso de ponderação dado pela equação 3.1.1.

$$R(x,y) = \frac{\sum_{s=-a}^{a}\sum_{t=-b}^{b} f(x+s,y+t).G(s,t)}{\sum_{s=-a}^{a}\sum_{t=-b}^{b} G(s,t)} \tag{3.1.2}$$

Um problema levantado por GONZALEZ e WOODS (2002) aponta o borramento das bordas, pois em quase toda análise de imagem a borda é uma característica importante, que também é caracterizada como alta freqüência. Deste modo, ao aplicar o filtro, talvez possa perder informações nestas regiões.

Neste sentido, a fim de solucionar o problema, é proposto à seguinte equação (3.1.3) conhecida como SUSAN filter (SMITH e BRADY, 1997) ou Bilateral filter (TOMASI e MANDUCHI, 1998), que consiste em um filtro gaussiano mais complexo.

$$KNN_{h,r}u(x) = \frac{1}{C(x)}\int_{\Omega(x)} u(y)e^{-\frac{|y-x|^2}{r^2}}e^{-\frac{|u(y)-u(x)|^2}{h^2}}dy \tag{3.1.3}$$

Onde:

$$C(x) = \int_{\Omega(x)} e^{-\frac{|y-x|^2}{r^2}}e^{-\frac{|u(y)-u(x)|^2}{h^2}}dy$$

Tem-se u(y) sendo a imagem com ruídos, $KNN_{h,r}$ u(x) é a imagem filtrada com os parâmetros *r* representando o sigma da gaussiana e o *h* o parâmetro do filtro espacial. O $\Omega(x)$ é a janela de vizinhança ao redor do pixel central *x*. A motivação desta equação mais elaborada é ponderar a influência de um pixel pela distância espacial, bem como pelo valor da intensidade luminosa deste pixel em relação ao pixel referência. Em contraste da borda separando duas regiões, se a diferença da intensidade luminosa entre dois pixels for maior que h, o algoritmo enfatizará somente o cálculo da média dos pixels pertencentes à mesma região que o pixel de referência. Assim, o filtro pode não borrar as bordas, que é seu objetivo principal (BUADES, COLL e MOREL, 2005).

3.2 Filtro Gradiente Sobel

Ao contrário do filtro de suavização, os filtros de aguçamento procuram realçar as diferenças locais, associadas às altas freqüências. Desde que uma borda é definida por uma mudança no nível de cinza, quando ocorre uma descontinuidade na intensidade, ou quando o gradiente da imagem tem uma variação abrupta, um operador que é sensível a estas mudanças operará como um detector de bordas. Como abordado por MORTENSEN e BARRETT (1995), uma maneira útil de encontrar bordas em imagens é utilizar o operador gradiente.

Um operador de derivada pode ser interpretado como a taxa de mudança de uma função, sendo que em imagens, a taxa de variação dos níveis de cinza são maiores próximos à borda. Se calcular a derivada nos valores da intensidade da imagem encontrando o ponto de máximo, este será a borda. Dado que as imagens são em duas dimensões, é importante considerar variações nos níveis de cinza em muitas direções. Neste sentido, usam-se as derivadas parciais nas direções X e Y, obtendo através da soma do vetor a estimativa da direção atual da borda. Visto a imagem como uma função de duas variáveis *f(x,y)*, então o gradiente é definido pela equação 3.2.1:

$$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) \tag{3.2.1}$$

Um bom operador para calcular o gradiente é o operador de Sobel descrito em (ABDOU e PRATT, 1979). Tecnicamente, este filtro é um operador diferencial discreto, computando a aproximação do gradiente em função da intensidade luminosa da imagem. Para cada ponto na imagem, o resultado do filtro Sobel corresponde ao vetor gradiente ou a normal deste vetor. O operador Sobel é baseado na convolução da imagem com um filtro pequeno, separável e inteiro nas direções horizontais e verticais (BAGGIO, 2007).

Matematicamente, o filtro usa duas janelas 3x3 que são convoluídas com a imagem original segundo a equação 3.2.2. Se definirmos A como sendo a imagem, Gx e Gy serão duas imagens que em cada ponto contém a aproximação derivada horizontal e vertical. Os cálculos são os seguintes:

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * A \; e \; G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A \tag{3.2.2}$$

onde (*) defini a operação de convolução 2D.

A coordenada x está aqui definida como incremento na direção “direita” e a coordenada y é definida como incremento na direção “descida”. Para cada ponto na imagem, o resultado aproximado do gradiente pode ser combinado para dar a magnitude do gradiente, usando a equação 3.2.3:

$$G = \sqrt{{G_x}^2 + {G_y}^2} \tag{3.2.3}$$

Usando esta informação, pode se também calcular a direção do gradiente dado pela equação 3.2.4:

$$\Theta = arctan\left(\frac{G_y}{G_x}\right) \tag{3.2.4}$$

Onde, por exemplo, Θ é 0 para a borda vertical que é escura do lado esquerdo.

3.3 Reconstrução Morfológica "H-MIN"

A morfologia matemática é uma teoria de conjuntos aplicada ao processamento de imagem. Classicamente a imagem é uma função da amplitude ou intensidade luminosa e de suas coordenadas no espaço. A morfologia matemática trata a imagem como um conjunto de pixels, associando as operações de união, interseção e complemento a um elemento estruturante. Maior aprofundamento pode ser visto em BARRERA et al. (1994).

Um dos conceitos mais importantes em morfologia é a reconstrução morfológica, que está associado ao conceito de conexidade e componente conexo (DOUGHERTY e LOTUFO, 2003). O conceito de conexidade está relacionado com a adjacência de um pixel, e o componente conexo pela união dos pixels adjacentes. Embora possa ser facilmente definida em si mesmo, é muitas vezes apresentada como parte de um conjunto de operadores conhecidos como geodésicos. A transformação reconstrutiva é relativamente bem conhecida no caso binário, onde simplesmente extrai os componentes conectados da imagem na qual estão marcados por outra imagem (VINCENT, 1993).

Esta reconstrução também pode ser definida para imagens em tons de cinza. Para VICENT (1993), a idéia acaba sendo particularmente interessante para vários filtros, segmentação e ferramentas de extração, onde surpreendentemente, tem atraído pouca atenção na comunidade de análise de imagens.

A reconstrução morfológica em nível de cinza pode ser obtida por sucessivas dilatações geodésicas. Este processo emprega duas imagens chamadas de imagem marcador e imagem máscara. Ambas as imagens devem ter o mesmo tamanho, e a imagem máscara deve ter valores de intensidade maiores ou iguais à imagem marcador (AREFI e HAHN, 2005).

Na dilatação geodésica o marcador é dilatado por um elemento estruturante elementar, onde a imagem resultado é forçada a ficar abaixo da máscara. Para Arefi e Hahn (2005) significa que a máscara age como uma barreira, limitando a dilatação no marcador. Para representação matemática, a imagem marcador será denotado por M e a imagem máscara por I, observando que ambas possuem o mesmo tamanho e $M \leq I$

A notação clássica da dilatação em nível de cinza da imagem M com o elemento estruturante B é dada por:

$$\delta(M) = M \oplus B \tag{3.3.1}$$

Onde o símbolo $\oplus$ caracteriza a operação de dilatação. No caso da dilatação geodésica de tamanho 1 na imagem marcador *M* com respeito à imagem mascara *I* é definido como:

$$\delta_I^{(1)}(M) = (M \oplus B) \wedge I \tag{3.3.2}$$

Na equação 3.3.2 o operador $\wedge$ denota o mínimo, a interseção entre a imagem marcador dilatada e a imagem máscara. Deste modo, a dilatação geodésica de tamanho *n* da imagem marcador *M* com respeito à imagem máscara *I* é obtida processando *n* sucessivas dilatações geodésicas de amplitude 1 de *M* com respeito a *I* (AREFI e HAHN, 2005).

$$\delta_I^{(n)}(M) = \underbrace{\delta_I^{(1)}(M) \circ \delta_I^{(1)}(M) \circ \cdots \circ \delta_I^{(1)}(M)}_{n\ vezes} \tag{3.3.3}$$

A reconstrução morfológica por dilatação geodésica é definida pela equação 3.3.3. Na descrição de VICENT (1993), a desejada reconstrução é determinada pela realização de várias dilatações geodésicas, até a imagem se estabilizar. A Figura 3.2 apresenta a reconstrução morfológica para 1D tendo um sinal *I* e seu marcador obtido por *M* = *I* – *h*.

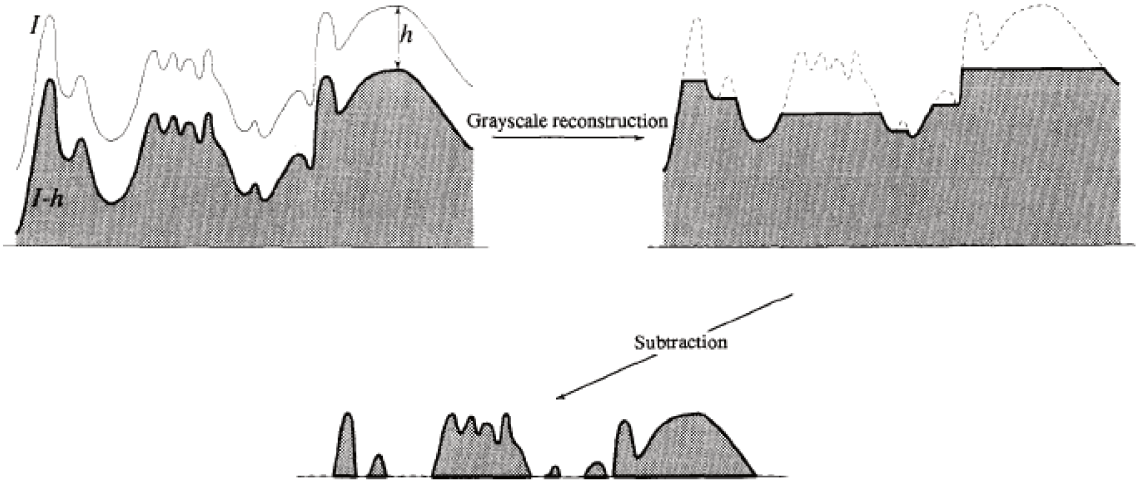


Figura 3.2: Determinação da reconstrução morfológica, sendo extraídos seus cumes.
FONTE (VICENT, 1993)

Em seu livro, DOUGHERTY e LOTUFO (2003) demonstram que ao atribuir para imagem marcador *M*, a imagem máscara *I* subtraída de um valor *h*, defini-se então a reconstrução morfológica por atributo de altura (*h*), sendo chamado de *h-maxima*.

$$HMAX_{h,D}(I) = I\Delta_D(I - h) \tag{3.3.4}$$

Este operador como visto na Figura 3.2, remove algum cume com altura menor que *h*(vide Figura 3.3), onde *h* é o valor do atributo altura, *D* representa a conexidade da vizinhança, *I* a imagem original, e o operador Δ representando a reconstrução morfológica.

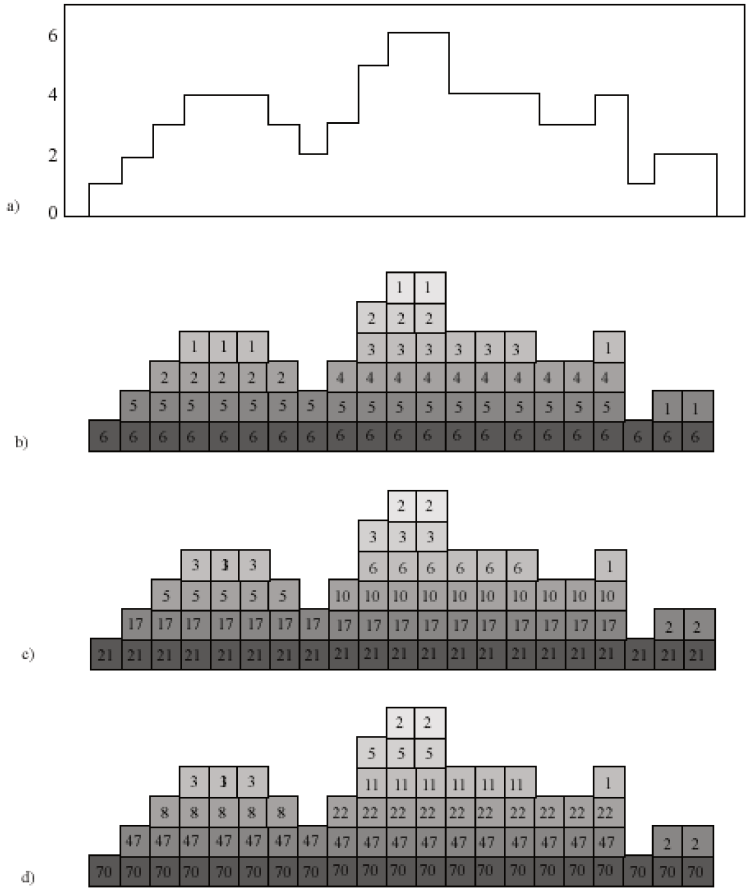


Figura 3.3: Características do componente conexo: a) Sinal de entrada b) atributo de altura c) atributo de área d) atributo volume Fonte: Adaptado (DOUGHERTY e LOTUFO, 2003)

Observa-se que quando o valor do atributo altura é igual a 1, este cume é definido como sendo o máximo regional. Segundo VICENT (1993), um máximo regional *MR* de uma imagem em escala de cinza *I* é o conjunto dos pixels com um dado valor, sendo o atributo de *h=1*, de forma que todo pixel na vizinhança de *MR,* tem que possuir valor restritamente menor. Conforme Figura 3.4.

Máximo regional não deve ser confundido com máximo local. Todo pixel contido em um máximo regional necessariamente é um máximo local, pois não existe nenhum vizinho a este pixel que tenha valor maior. Em outras palavras, a união dos pixels de máximo local conexos forma o máximo regional. Tais definições também são válidas para caracterizar os mínimos regionais.

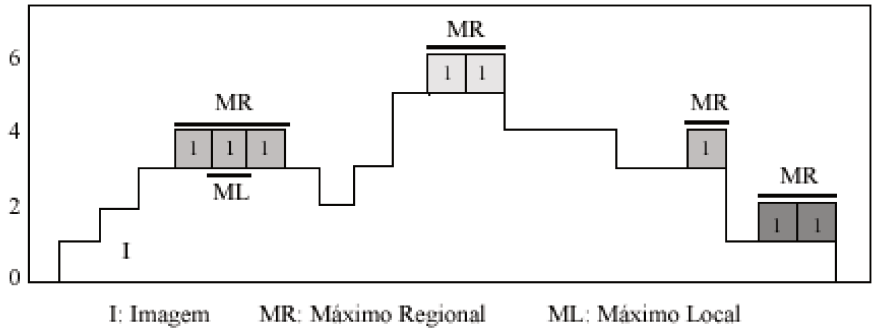


Figura 3.4: Máximos Regionais de uma imagem em escala de cinza

A forma dual do operador *h-maxima* é chamado de *h-minima*:

$$HMIN_{h,D}(I) = (I^C \Delta_D (I - h)^C)^C \tag{3.3.5}$$

Onde *c* caracteriza a imagem negada ou invertida. Esta equação define a reconstrução morfológica "H-MIM". Apresenta se na Figura 3.5 o resultado do processamento do filtro H-MIM para 1D.

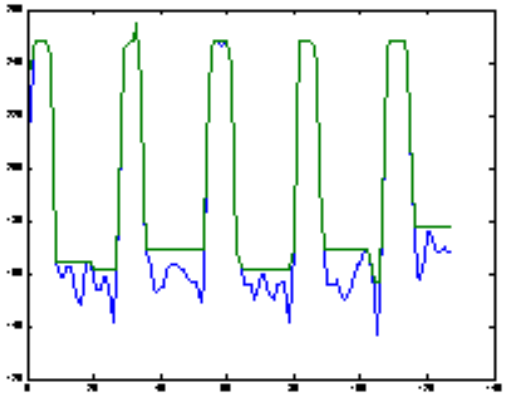


Figura 3.5: Sinal 1D de entrada em azul e o sinal após o filtro reconstrutivo H-MIN em verde.

A motivação e justificativa para utilização deste filtro se da pela remoção dos mínimos regionais encontrados em uma imagem.

3.4 Transformada Watershed

Como apresentado, o processo de segmentação em uma imagem digital tem como objetivo subdividí-la em suas partes ou objetos constituintes. Segundo Lopes (2003), tal tarefa pode ser extremamente sofisticada, e a partir de filtros bem elaborados, pode-se chegar a resultados muito eficientes.

Uma das técnicas para se segmentar uma imagem é a partir da transformada Watershed. Para VITOR et al (2009), esta transformada é amplamente utilizada na segmentação em aplicações de visão computacional. Muitas definições para transformada Watershed existem na literatura (VINCENT e SOILLE, 1991), (MEYER, 1994), (LOTUFO e FALCÃO, 2000), (FALCÃO, STOLFI e LOTUFO, 2004), (BIENIEK e MOGA, 1998), (COUSTY, BERTRAND, *et al.*, 2009) que tomam diferentes propósitos para o problema, cujo definindo componentes conexos via zonas de influência, floresta de caminhos mínimos com a função de custo para a distância, localmente por seguir o caminho da descida mais íngreme. Da maneira abordada neste trabalho, seu conceito é baseado na interpretação topográfica da imagem e sua inundação.

Considera-se que os mínimos regionais de uma imagem em nível de cinza sejam perfurados e que a superfície seja imersa na água. Desta forma a água penetra com velocidade vertical uniforme pelos furos até que fluxos provenientes de mínimos regionais diferentes possam se unir. Como não se quer a união de águas, diques podem então ser construídos para evitar esta fusão. E ao terminar o processo, isto é, quando toda superfície estiver imersa sobre as águas, somente terão os diques emersos, sendo considerado como linhas divisoras que contornam as bacias hidrográficas, contendo cada qual apenas um dos mínimos regionais iniciais (BEUCHER e MEYER, 1993). A Figura 3.6(d)-(e) apresenta este conceito aplicado a um sinal.

Para melhor visualização em outra concepção, considere uma imagem em nível de cinza f, sendo interpretada como uma superfície topográfica S. Ao imaginar gotas de água caindo sobre esta superfície, elas irão escoar dos picos para os vales, onde são denominados mínimos regionais da superfície S. Todas as gotas que escoarem para um mesmo mínimo regional são rotuladas como uma mesma região (vide Figura 3.6(a-b-c)). Neste sentido, a transformada Watershed pode ser classificada como uma segmentação por região ("region-based") (KHIYAL, KHAN e BIBI, 2009). A Figura 3.6 apresenta a interpretação física do Watershed.

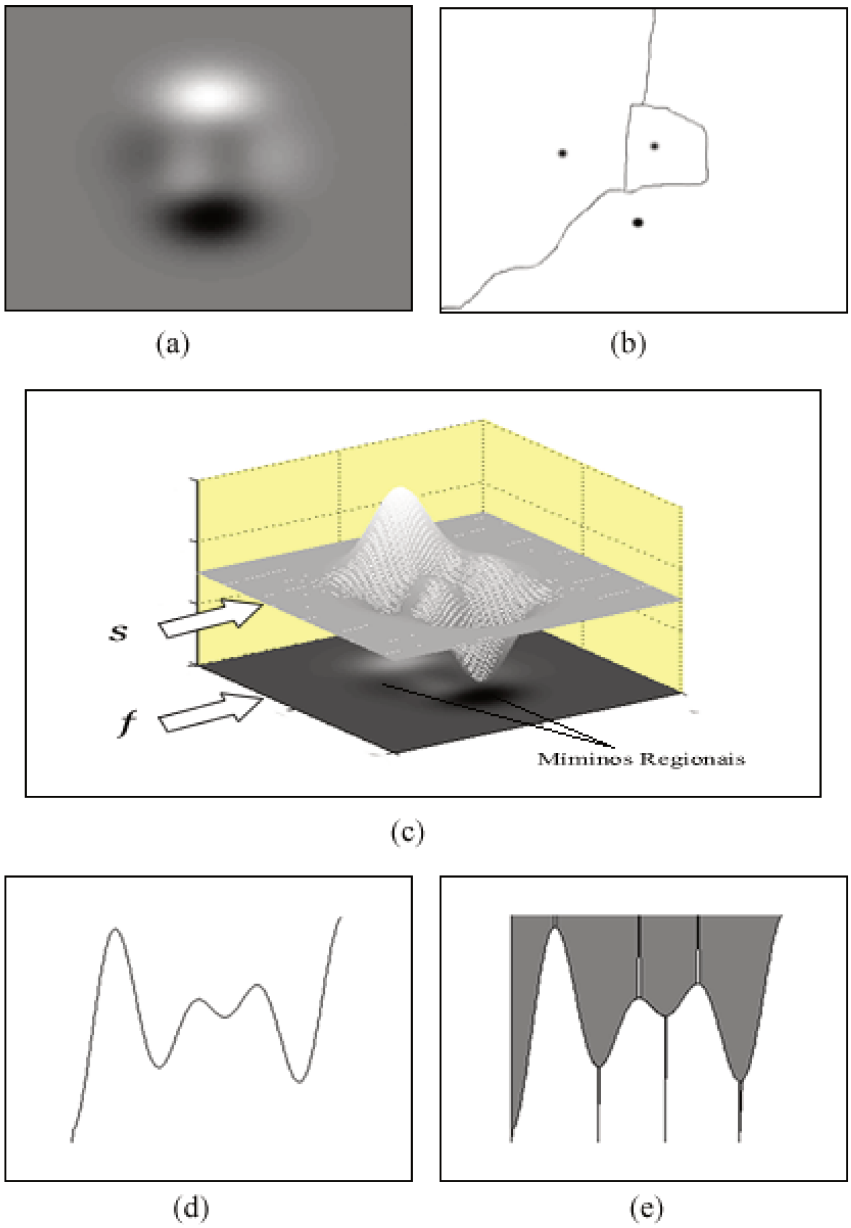


Figura 3.6: Interpretação física do Watershed: (a) Imagem Original (b) Linhas Watershed com os mínimos regionais (c) Representação topográfica da imagem original (d) sinal 1D entrada (e) resultado Watershed 1D

Visando obter resultados úteis do processo de segmentação, muitas vezes o Watershed é computado em cima da imagem gradiente (RONDINA, 2001).

3.5 Cálculo de Características

Esta seção traz alguns conceitos que serão utilizados na fase de visão computacional como área, centróide e média dos valores de intensidade luminosa, sendo apresentadas suas definições.

Área de Superfície Plana

Nas palavras de DOLCE e POMPEO (2005), a área de uma superfície é um número real positivo associado à superfície de forma que:

Às superfícies equivalentes estão associadas áreas iguais (números iguais) e reciprocamente.

$$A \approx B \Leftrightarrow (Área\ de\ A = Área\ de\ B)$$

A uma soma de superfícies está associada uma área (número) que é a soma das áreas das superfícies parcelas.

$$(C = A + B) \Rightarrow \left(Área\ de\ C = Área\ de\ A + Área\ de\ B\right)$$

Se uma superfície está contida em outra, então sai área é menor (ou igual) que a área da outra.

$$B \subset A \Rightarrow Área\ de\ B \leq Área\ de\ A$$

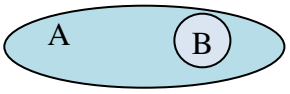


Figura 3.7: Áreas de duas superfícies

Ao calcular a medida de uma superfície, verifica-se que esta pode ser regular ou irregular em sua forma. São regulares as que possuem formas geométricas conhecidas. As irregulares têm forma indefinida. Para medir as superfícies de formas irregulares, é preciso utilizar um procedimento aproximado, ao contrário de formas geométricas conhecidas, que pode utilizar

fórmulas para realizar esses cálculos. Para quantificar a medida da superfície, é necessário estabelecer uma unidade. Em geral, como se trabalha com a imagem discretizada, a unidade definida é um quadrado conhecido como pixel. A Figura 3.8 apresenta a superfície discretizada

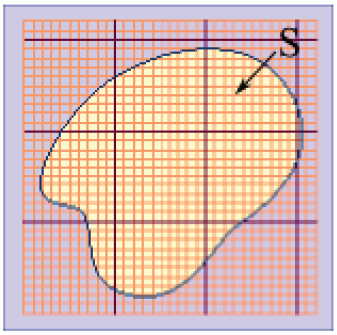


Figura 3.8: Discretização da superficie

Para calcular a área de cada superfície, basta utilizar a seguinte equação 3.5.1.

$$A_S = \sum_{x,y} S(x,y) \tag{3.5.1}$$

Onde

$$S(x,y) = \begin{cases} 1. Se\ p_{x,y} \subset S \\ 0. Se\ p_{x,y} \not\subset S \end{cases}$$

Centróide

Em geometria, o centróide é o ponto no interior de uma forma geométrica que define o seu centro geométrico. Seja uma superfície plana genérica S conforme Figura 3.9.

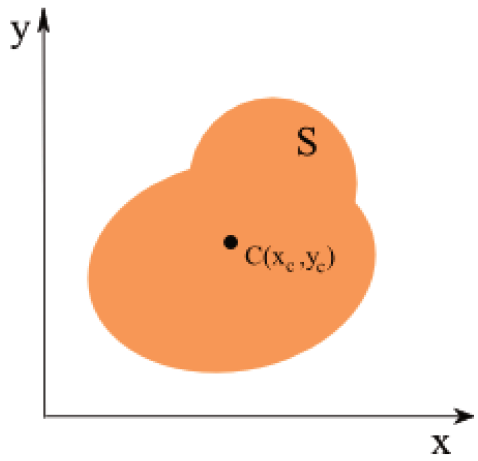


Figura 3.9: Representação do centróide de uma Superfície

Seu centróide é dado por:

$$x_C = \frac{\sum_{x,y} x.S(x,y)}{A_S} \qquad y_C = \frac{\sum_{x,y} y.S(x,y)}{A_S} \tag{3.5.2}$$

O ponto C(xc, yc) é denominado centróide da superfície.

Média dos valores de Intensidade luminosa da Imagem

Considerando uma imagem I tendo os valores da intensidade luminosa em cada ponto I(x,y) representando a cor de uma dada superfície Is(x,y), o valor médio ou esperado da cor nesta superfície $\overline{m_s}$ é obtido pela seguinte equação:

$$\overline{m_S} = \frac{1}{A_s} \sum_{x,y} I_S(x,y) \tag{3.5.3}$$

Capítulo 4

PROCESSAMENTO PARALELO PARA TRATAMENTO DE IMAGENS

A grande motivação de conciliar o poder do paralelismo em algoritmos de visão computacional é proporcionar um desempenho ao ponto da aplicação poder ser operada com a robustez desejada e em tempo real.

O paralelismo em GPU consiste em uma programação SIMD, isto é, possuir um conjunto de dados ordenados e independentes de tal forma que necessite de uma única instrução de processamento para todo conjunto, onde cada elemento executará a operação independente uns dos outros em paralelo.

Para PHARR e FERNANDO (2005) a capacidade da GPU em processar de maneira SIMD é adequada para rodar tarefas de visão computacional, onde envolve um mesmo cálculo operando sobre toda a imagem. Neste sentido, a tarefa de visão computacional mais específica no processamento de imagem é bastante adequada para utilizar os benefícios da arquitetura GPU, pois o conjunto de dados representado pela imagem sofrerá o mesmo cálculo para todos os seus elementos, sendo que cada elemento do conjunto é definido como pixel.

A proposta de solução abordada para modelagem da computação paralela neste capítulo demanda o conhecimento de um conjunto de métodos de processamento em imagens, apresentados no capítulo 3. Além destes métodos, é necessário o conhecimento de alguns conceitos sobre o paradigma de programação paralela e a idéia de programação genérica chamada CUDA. Com o objetivo de facilitar o entendimento da solução proposta, será apresentada a seguir uma revisão destes conceitos seguidos pelo desenvolvimento dos algoritmos de filtro KNN, filtro Sobel, reconstrução morfológica HMIN e a transformada Watershed, exemplificando cada passo.

4.1 Paradigma de Programação Paralela

As aplicações que utilizarão e/ou utilizam os benefícios da arquitetura GPU possuem três características em comum abordadas por Owens et al. (2008), apresentam um domínio amigável a formulações paralelas, necessitam de grande volume de recursos computacionais relacionados a computações sobre conjunto de dados ordenados, e o "throughput" caracterizado pela quantidade de informação baixa processada em um intervalo de tempo. Para explorar as potencialidades da GPU, consideráveis reformulações de algoritmos e estruturas de dados são necessárias (OWENS, LUEBKE, *et al.*, 2007) (VASCONCELOS, 2009).

A construção fundamentada do pensamento computacional para o paradigma de computação paralela, o ganho de maturidade, o conhecimento são adquiridos com a evolução de simples percepções sobre particulares modelos de programação que, ao se consolidar, proporciona um nível de conhecimento capaz de generalizar e adaptar um particular modelo de programação para diversos contextos. Para KIRK e HWU (2010) várias habilidades são necessárias para um eficiente raciocínio computacional paralelo, sendo sumarizadas em quatro fundamentais habilidades em:

- **Arquitetura computacional:** Envolve o conhecimento da organização da memória, caching e localização; largura de banda das memórias; a execução das arquiteturas (SIMT) Simples Instrução, Múltiplos Threads; (SPMD) Simples Programa, Múltiplos Dados; (SIMD) Simples Instrução, Múltiplos Dados; além da exatidão e precisão dos pontos flutuantes. Estas concepções são fundamentais na compreensão das trocas entre os algoritmos.
- **Compiladores e modelos de programação:** Esta seção envolve a compreensão dos modelos de execução paralela, tipos de memórias disponíveis, o layout de arranjos dos dados bem como as transformações dos laços de repetição, para alcançar melhores performances.
- **Técnicas de algoritmos:** existem técnicas modeladas para algoritmos paralelos que são superiores como tiling, cutoff, binning visto em (KIRK e HWU, 2010). O

entendimento das implicações de escalabilidade, eficiência e largura de banda das técnicas são essenciais no pensamento computacional paralelo.

- **Domínio do problema:** Entendem-se pelos métodos numéricos, modelos, propriedades matemáticas que proporcionam desenvolvimentos mais criativos aplicados nas técnicas dos algoritmos.

O primeiro passo para vencer os diferentes desafios em transformar tradicionais algoritmos seqüenciais em algoritmos coerentes a arquitetura de processadores paralelos, é compreender que as GPU´s utilizam o modelo de programação STREAM – "Stream Programming model" discutidos por (OWENS, LUEBKE, *et al.*, 2005) e (LEFOHN, KNISS e OWENS, 2005).

A "stream" é uma coleção ordenada de registros de um mesmo tipo a serem processados por computações similares (maneira SIMD), efetuada por um operador chamado "kernel" (PHARR e FERNANDO, 2005). Este operador é similar a uma função, onde um comando será aplicado a toda stream que representa a operação paralela para cada elemento da stream. Pode ser visto na Figura 4.1 que dentro do kernel, ao executar uma operação como, por exemplo, a soma, a operação é vetorizada, onde cada elemento é processado em paralelo. Para BUCK et al. (2004) o processador de "stream" é um "kernel" que executa uma operação em todos os elementos dos "streams" de entrada, devolvendo o resultado em uma "stream" de saída. Deve ser ressaltado que neste trabalho, a stream está relacionada com os dados de uma imagem.

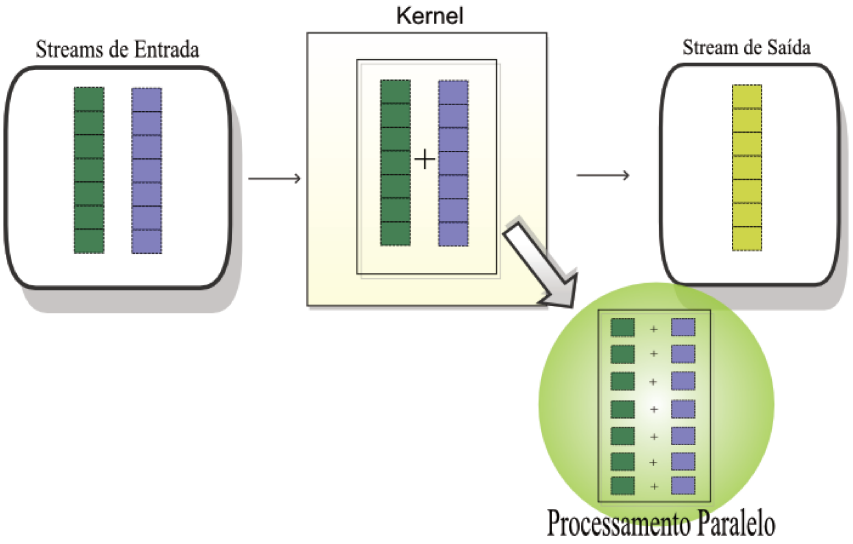


Figura 4.1: Relação das Streams com o Kernel.

A manipulação de memória para codificação de um programa é de fundamental importância no desempenho do processamento paralelo, tendo em vista os vários tipos existentes na GPU como global, constante, textura, compartilhada e os registradores. Suas particularidades são apresentadas na próxima seção e também discutidas em (NVIDIA, 2010).

4.2 CUDA

CUDA é uma API de programação genérica em GPU, onde o conhecimento para mapear as aplicações através de pipeline gráfico e das API´s gráficas não é necessário. Sua proposta é oferecer um ambiente de programação similar à linguagem de programação C, que permita a execução de grandes quantidades de operações em paralelo através de uma única instrução.

A similaridade à linguagem C proporcionada pelo CUDA se traduz em duas abstrações, a abstração do CPU e da GPU. A abstração do CPU é caracterizada como sendo um dispositivo "host", o qual é visto como um processador de controle responsável por configurar parâmetros, inicializar parâmetros e executar as partes seriais do programa. A abstração da GPU é denominada como dispositivo "device", vista como co-processador composto de conjuntos de

multiprocessadores focados na aceleração da computação de dados formulados de maneira paralela (NVIDIA, 2010).

Observando que no modelo de programação seqüencial, os procedimentos que tradicionalmente eram descritos como um laço de repetição percorrendo todo o domínio de processamento, na organização paralela fornecida por CUDA, os elementos do domínio da aplicação devem ser divididos em modelos ou formas independentes. Como visto anteriormente, também em CUDA, a função que processará os elementos de um domínio (stream) é denominada de "kernel", sendo caracterizada como a função que fará a execução em todos os elementos do domínio. No sentido de repartir o domínio em partes paralelas, CUDA utiliza uma modelagem de estruturação em grades conhecida como "grid". Cada "grid" possui subdivisões denominadas de blocos "blocks", que por sua vez possuem os elementos caracterizados como threads. O thread e definido como sendo a execução da instrução do kernel para cada elemento e em paralelo. A Figura 4.2 apresenta à estruturação de um domínio em paralelo modelado em CUDA, bem como a abstração entre "host" e "device" para programação em C.

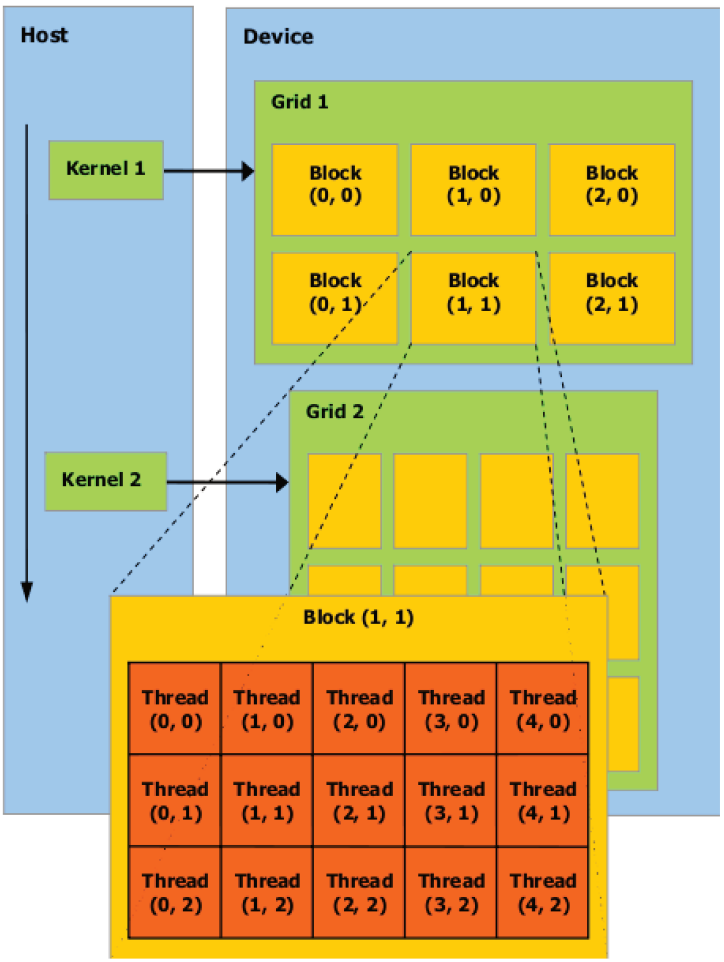


Figura 4.2: Abstração entre "host" e "device", e estruturação dos dados para execução em paralelo

O processamento no "device" se dá pela alocação dos vários blocos de um grid nos diferentes multiprocessadores. A dinâmica de modelagem do número de bloco no grid, bem como o número de threads no bloco deve atender alguns requisitos quanto à arquitetura de hardware utilizada e o domínio do problema referenciado em (KIRK e HWU, 2010). A indexação de acesso a um thread respeita a hierarquia cartesiana no interior dos blocos e os blocos no interior do grid (vide Figura 4.2). Dentro de um kernel, a indexação é fornecida dinamicamente através de variáveis pré-estabelecidas, que servem para o programador manipular mapeamentos relacionados à distribuição dos elementos do domínio, em uma dada lógica de programação. A fim de demonstrar a indexação dentro do kernel, a modelagem de um grid e um comparativo entre a programação seqüencial e paralela, utilizando a linguagem C e CUDA, tem-se na Figura 4.3 o código de uma simples operação de soma entre dois vetores, sendo a primeira em CPU e a segunda em GPU.

```
CPU C program

void add_matrix_cpu
                (float *a, float *b, float *c, int N)
{
    int i, j, index;
    for (i=0;i<N;i++) {
     for (j=0;j<N;j++) {
        index =i+j*N;
        c[index]=a[index]+b[index];
        }
    }
}
void main()
{
   .....
       add_matrix(a,b,c,N);
}
```

```
CUDA C program

__global__ void add_matrix_gpu
                (float *a, float *b, float *c, int N)
{
    int i=blockIdx.x*blockDim.x+threadIdx.x;
    int j=blockIdx.y*blockDim.y+threadIdx.y;
    int index =i+j*N;
    if( i <N && j <N) c[index]=a[index]+b[index];
}


void main()
{
    dim3 dimBlock (blocksize,blocksize);
    dim3 dimGrid (N/dimBlock.x,N/dimBlock.y);
    add_matrix_gpu<<<dimGrid,dimBlock>>>(a,b,c,N);
}
```

Figura 4.3: Simples comparativo de programação seqüencial e paralela

Pode ser percebido que na Figura 4.3, a similaridade entre uma programação C e CUDA é bastante evidente. No entanto, existem algumas particularidades como a diretiva __global__ que serve para especificar que a função será processada no GPU, algumas variáveis pré-definidas como “blockidx”, ”threadidx” especificando os índices dos blocos no grid e os threads no bloco, bem como a variável “Dim3” que serve para modelar o grid em blocos e threads.

Sobre uma visão geral do desenvolvimento do código, tem-se a rotina *main* sendo executada no host, responsável por modelar a dimensão do grid e dos blocos, e iniciar o kernel *add_matrix_gpu*, passando os parâmetros bem como as streams de entrada. O kernel é executado na GPU, tendo as variáveis *i* e *j* recebendo a indexação no sentido horizontal e vertical, a variável *index* recebendo tal indexação para acesso às posições da memória global, e finalmente processando toda a matriz restringindo a faixa de dados na dimensão N. Este é um simples código que utiliza somente a memória global para processamento.

CUDA suporta vários tipos de memória que podem ser usados pelo programador para proporcionar maior velocidade de processamento dos “kernels” (KIRK e HWU, 2010). Dentre essas memórias, existe a de acesso à memória DRAM da placa Gráfica (“onboard memory”) nos tipos constante, textura e global. Os valores contidos nessas memórias são persistentes entre chamadas de “kernels” de uma mesma aplicação e acessível ao “host”, podendo por ele realizar algumas manipulações de inicialização, copia de streams de entrada e saída entre os dispositivos

host-device, e também no controle de finalização de um processamento do “kernel”, onde se necessite avaliar se as respostas de cada bloco não terá influência nos demais.

CUDA também oferece alguns tipos de memória armazenados no chip de cada multiprocessador, sendo caracterizadas pelo rápido acesso, como os registradores, a memória local, e a memória compartilhada (“shared memory”). Os registradores e a memória local são associados aos threads, onde cada thread possui seu próprio espaço. A memória compartilhada é bastante utilizada, pois permite que os threads de um mesmo bloco compartilhem dados através dessa, em alta velocidade (NVIDIA, 2010). Uma restrição encontrada nestes tipos de memória de rápido acesso é o espaço de armazenamento dos dados, o qual é limitado.

Na DRAM, a memória global possui amplo espaço de armazenamento, permite operações de leituras e escritas realizadas pelos “kernels”, aumentando a flexibilidade de utilização desta memória. No entanto, sua restrição está voltada para o tempo demandado na transferência de dados, sendo que para completar uma instrução de escrita ou leitura na memória global, são necessários aproximadamente 400 a 800 ciclos de clock (NVIDIA, 2010).

Uma alternativa poderosa para contornar estes problemas é a utilização da memória de textura. Segundo NVIDIA (2010), esta memória possui cache para otimizar a transferência de dados dentro da GPU, possui otimização para localizar rapidamente os dados em um espaço 2D na memória, normalização dos dados em hardware, tratamento de borda, e também possui interpolação em hardware para aquisição de dados entre pontos discretos da imagem. Como exposto, a memória de textura é bastante apropriada para manipulação de imagens. A Figura 4.4 apresenta a arquitetura das memórias no GPU.

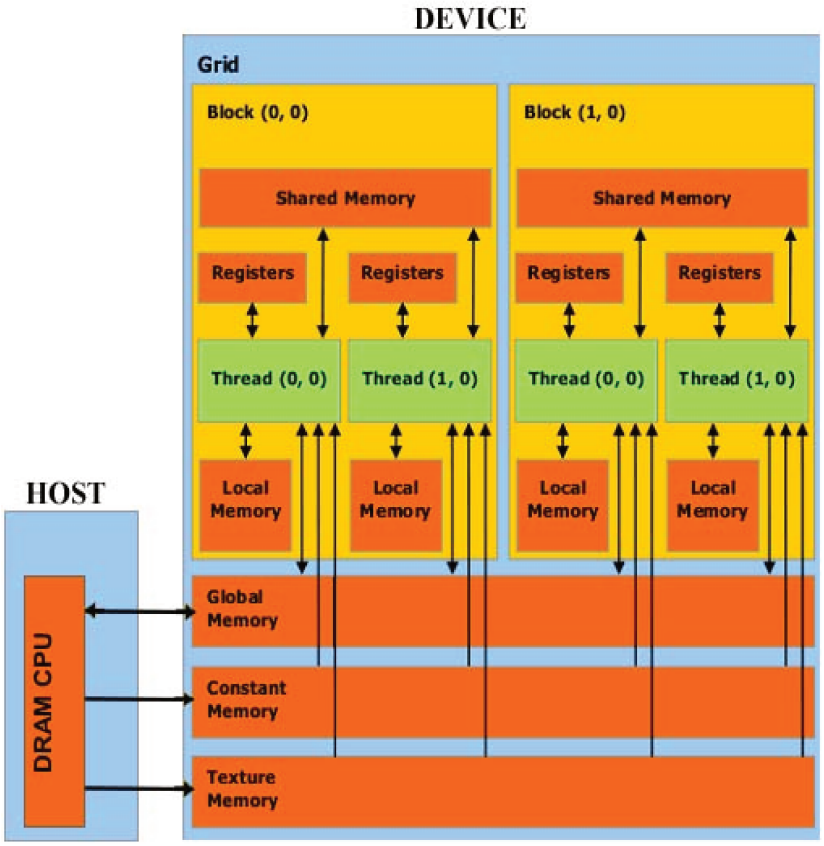


Figura 4.4: Arquitetura de memória no GPU

Objetivando apresentar a relação existente com os diversos tipos de memória e seus acessos por parte do "host" e "device", a Tabela 4.1: Tipos de acesso na memória do GPU trás tais informações:

Tabela 4.1: Tipos de acesso na memória do GPU

	Tipo	Device	Host
No chip	Registradores por thread	Leitura/Escrita	-
	Memória Local por thread	Leitura/Escrita	-
	Memória Compartilhada por bloco	Leitura/Escrita	-
DRAM	Memória Global por grid	Leitura/Escrita	Leitura/Escrita
	Memória Constante por grid	somente Leitura	somente Escrita
	Memória de Textura por grid	somente Leitura	somente Escrita

Um fator interessante a respeito da sincronização necessária quando se programa em paralelo, é que o CUDA oferece um mecanismo de sincronização por barreiras para threads de

um mesmo bloco denominado *__syncthreads()*. No caso da sincronização dos blocos, o mecanismo deve ser feito pelo comando denominado de *cudaThreadSynchronize()*, o qual garante que toda a tarefa dos blocos no "kernel" foi finalizada, isto é, o comando aguarda o "kernel" ser concluído.

Conforme mencionado anteriormente, inicialmente o programador deve conhecer as restrições impostas pelas placas gráficas para assim desenvolver o algoritmo de maneira correta e otimizada. Tais restrições são caracterizadas pela divisão máxima de blocos no grid, o número máximo de threads suportados em um bloco, o fator limitante de espaço a se utilizar nas memórias do chip (memória compartilhada e registradores), bem como os tipos de acesso nas memórias visto na Tabela 4.1. No caso específico deste trabalho, a placa gráfica utilizada é uma GTX 295 com 1792Mb, contendo 2 GPU´s com 240 núcleos de processamento cada. Maiores detalhes sobre as características e limites impostos por esta placa pode ser visto no apêndice A.

4.3 Filtro Gaussiano KNN

A principal idéia do filtro KNN é calcular a intensidade luminosa de cada pixel cujo valor é dependente da intensidade luminosa de uma dada janela de vizinhança visto na Figura 4.5. A proposta do algoritmo é baseado na solução apresentada por (KHARLAMOV e PODLOZHNYUK, 2007) que é inspirada na equação 3.1.3 modelado paralelamente.

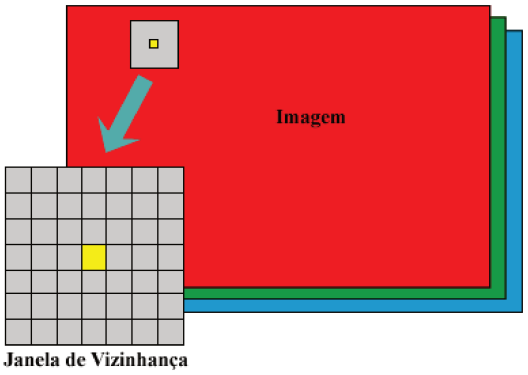


Figura 4.5: Janela de Vizinhança para um pixel

O fluxograma da Figura 4.6, composto por vários blocos, apresenta o algoritmo que executa o filtro KNN. O fluxograma está modelado de forma a demonstrar a execução do código no CPU (host) e GPU (device). O bloco retangular com duas faixas nas extremidades, visto no fluxo do host, denota a chamada do kernel na GPU, sendo que este espera a finalização de todo o processamento sendo realizado no device para então, após receber o resultado do processamento, seguir o fluxo no host. O bloco paralelogramo representa toda transferência de dados entre os diferentes tipos de memória do device e host. Os demais blocos são para mostrar o processamento e controle do fluxo. O algoritmo é realizado em um único kernel, sendo que à imagem de entrada RGB denotada por *I* é armazenada na memória de textura. O acesso a imagem é definido pelos índices *p,q* no sentido horizontal e vertical respectivamente. A imagem de saída chamada por *dst* é armazenada na memória global do device.

No algoritmo, KHARLAMOV e PODLOZHNYUK (2007) considera que a janela de vizinhança, chamada de janela de convolução, é de tamanho NxN, onde N depende do raio da janela chamado R_JANELA, que é caracterizado pelo número de pixels adjacentes conectados entre a borda e o pixel central da janela. Deste modo a dimensão N será N = 2 * R_JANELA + 1. Como a janela de convolução é NxN, então a área da janela e sua inversa é denotada por AREA_JANELA = N*N e INV_AREA_JANELA = 1.0f / AREA_JANELA. Considera-se também que o desvio da gaussiana é σ = N.

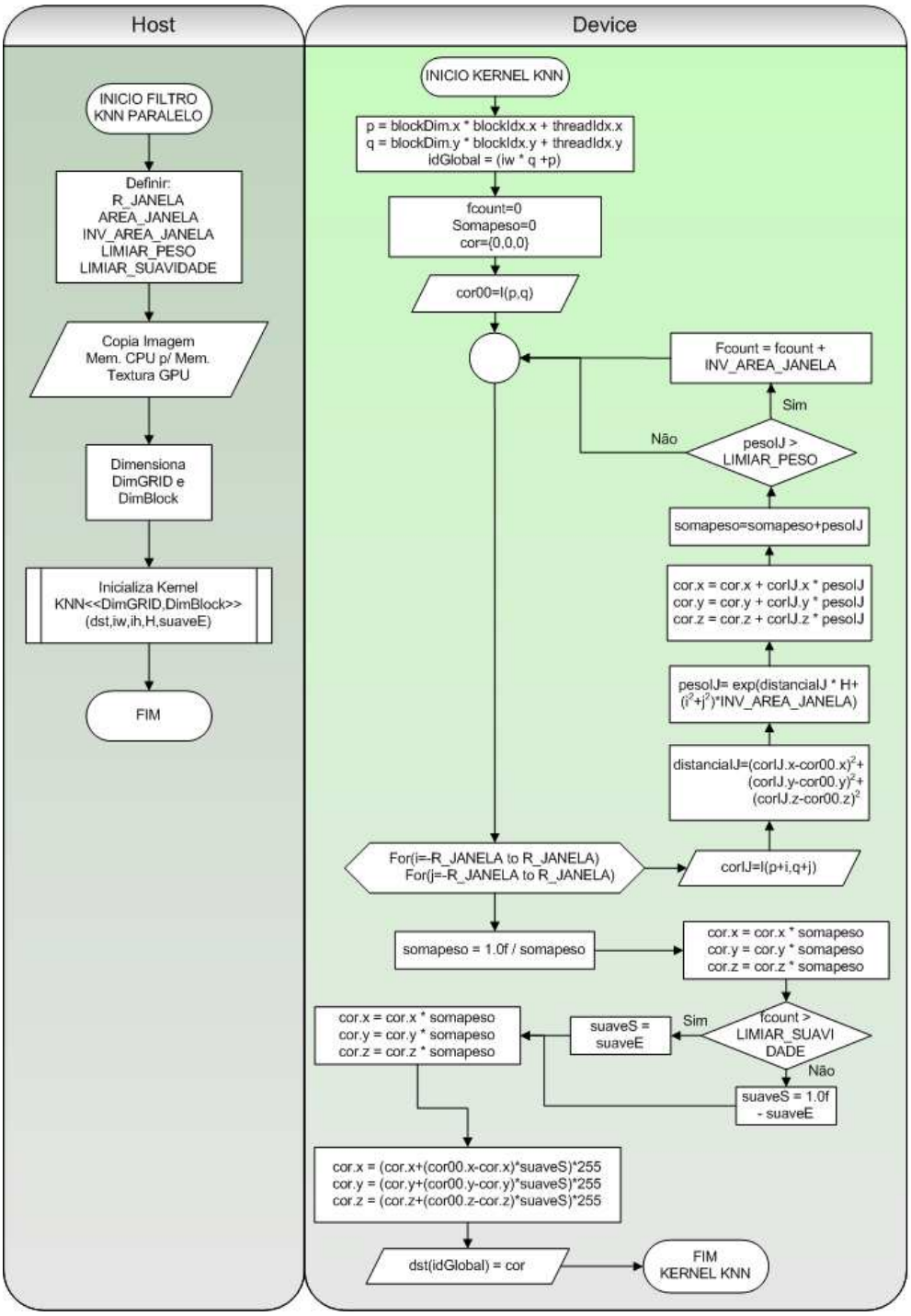


Figura 4.6: Fluxograma do algoritmo KNN paralelo em GPU

Existem ainda alguns parâmetros no algoritmo KNN que podem ser alterados pelo usuário, onde vários valores destes parâmetros foram sugeridos por (KHARLAMOV e PODLOZHNYUK, 2007). Dois destes chamados *h* e *suaveE,* servem para ponderação do filtro

espacial e ponderação de suavidade da janela. A escolha do valor de *h* é subjetivo, devendo o usuário estimar qual o melhor valor para boa qualidade da imagem, e para a variável *suaveE*, é sugerido uma escolha entre [*0.00f,0.33f*]. Três outros parâmetros como o R_JANELA variando entre [5,7], o LIMIAR_PESO que é posto para classificar se a contribuição do peso de cada valor do pixel da janela é significativo ou não, sendo sugerida uma escolha variando entre [*0.66f, 0.95f*], e por fim o parâmetro LIMIAR_SUAVIDADE tendo um valor típico de 66%, com a finalidade de verificar se um pixel central de uma dada janela de convolução foi suavizada, em relação a variável normalizada *fcount*. Caso *fcount* esteja acima do LIMIAR_SUAVIDADE, o algoritmo não altera a variável de ponderação suaveS.

Segundo KHARLAMOV e PODLOZHNYUK (2007) este procedimento descrito anteriormente serve para ponderar a imagem ruidosa com a imagem suavizada na constituição da imagem resultante, presumindo o fator de ponderação para não borrar localidades que contenha borda ou pequenas características.

Para exemplificar como é realizado o filtro KNN paralelo em uma imagem, será apresentado passo a passo a execução do algoritmo. Inicialmente, seja uma imagem caracterizada com dimensão 6x10, possuindo 3 camadas RGB conforme Figura 4.7(a). Aplicando os critérios para modelagem do grid (visto na seção 4.2), dividi-se a imagem em 4 blocos de dimensão 3x5 conforme Figura 4.7(b). Deste modo, o número de threads no bloco representado por DimBlock é 3x5, e o número de blocos no grid representado por DimGRID é 2x2.

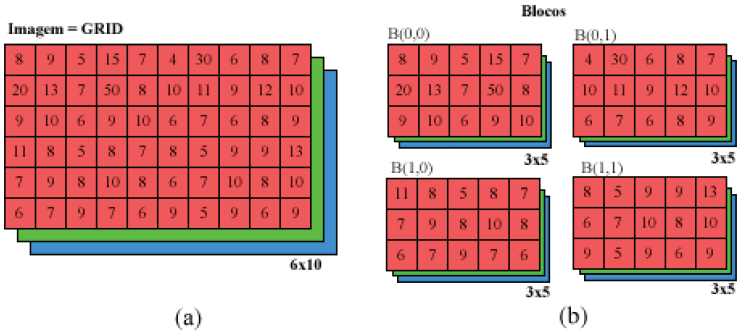


Figura 4.7: Modelagem KNN do Grid em Blocos (a)Imagem (b) Blocos do Grid

Definindo as variáveis macros como *R_JANELA 1,* para por simplicidade, *LIMIAR_PESO 0.02f*, *LIMIAR_SUAVIDADE 0.79f.* conforme explicação apresentada anteriormente. Seguindo o

fluxo do algoritmo apresentado, posterior as definições, tem-se a imagem sendo copiado da memória do CPU - host, para memória de textura da GPU – device.

A próxima etapa é então iniciar o kernel passando os parâmetros para o device, aonde as variáveis *iw* e *ih* representa a dimensão da imagem. No exemplo tais variáveis assumem os valores *iw = 10* e *ih = 6*. Os outros parâmetros passados ao device foram *H 9.77f*, *suaveE 0.40f*, escolhidos subjetivamente, segundo sugestões dos autores. É importante notar que a alocação de memória para o ponteiro *dst*, é feita antes da inicialização do kernel.

Dentro do device, as variáveis *p,q* recebem a indexação composta dos threads nos blocos e dos blocos no grid. Algumas variáveis são inicializadas na memória registrador como fcount = 0, cor = {0,0,0} e somapeso= 0 na memória registrador.

A atribuição *cor00 = I(p,q)*, significa que a memória registrador de cada multiprocessador irá receber os dados de uma fatia da imagem da memória de textura, correspondente a dimensão de um dado bloco(bx,by). Cada ponto desta fatia de imagem corresponde a um pixel central do processamento da janela de convolução. A atribuição corIJ = I(p+i, q+j) representa também o carregamento de uma fatia de imagem nos multiprocessadores, onde cada ponto desta fatia representa o vizinho ij de cada ponto da fatia central.

Os dois loops de (i,j) que variam na dimensão [–N,N], isto é, i = [-N,N] e j[-N,N], servem para varrer toda janela de convolução, que no caso deste exemplo foi definido uma janela de dimensão 3x3. De posse do pixel central e seu respectivo vizinho na janela de convolução, é então calculada a *distanciaIJ* , o *pesoIJ*, a influência do valor do pixel vizinho na constituição do novo valor do pixel central dada por *cor*, a soma dos pesos da janela de convolução dada por *somapeso* e a soma para normalização dado pelo *fcount*.. Conforme demonstrada na Figura 4.8.

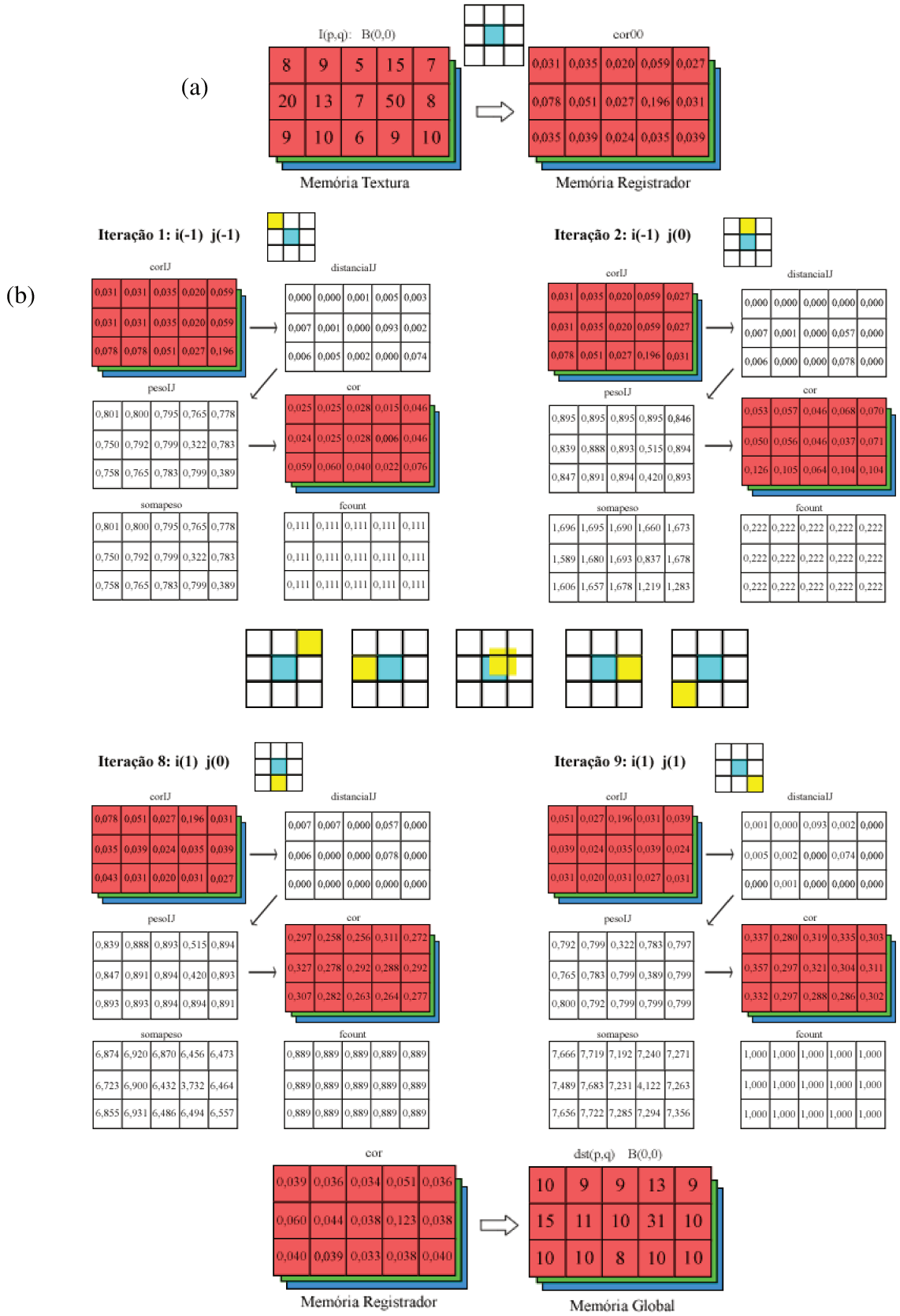


Figura 4.8: Exemplo do processamento paralelo para o filtro KNN

Ao carregar os dados da memória de textura para o registrador, existe a opção de carregar os dados já normalizados (visto na Figura 4.8(a)).

Como visto, o processamento paralelo está implícito em calcular vários blocos nos diversos multiprocessadores. Dentro de cada bloco os threads estão sendo processados também em paralelo. Como a placa GTX 295 têm 30 multiprocessadores, pode se dizer que em um instante de tempo *t*, cada multiprocessador executará um bloco, e cada bloco executará Dimesão_Bloco_X x Dimesão_Bloco_Y threads.

4.4 Filtro Gradiente Sobel

A motivação para aplicação do filtro Sobel se dá pela detecção de contornos em uma imagem. A proposta é baseada na solução desenvolvida por (CUDA, 2007) onde se aplica a formulação descrita na seção 3.2.

Do modo apresentado na seção anterior (4.3), a modelagem paralela para o filtro Sobel poderia seguir a mesma linha de raciocínio descrito no algoritmo KNN paralelo, calculando a resposta do pixel central percorrendo a janela de vizinhança. No entanto, como o filtro Sobel deve calcular o gradiente nas direções X e Y, deveria então possuir no mínimo dois kernels para processamento deste. Entretanto a lógica adotada possui somente 1 kernel, onde se monta a janela de convolução para cada pixel central, processando as duas matrizes de uma só vez, retornando o valor já somado dos gradientes nos dois sentidos.

O fluxograma da Figura 4.9 apresenta o algoritmo que executa o filtro Sobel. As definições quanto ao significado de cada bloco do fluxograma podem ser vistas na seção (4.3). De maneira geral, a idéia do algoritmo é calcular o gradiente na intensidade luminosa do número máximo possível de pixel de um bloco ao mesmo tempo denominado maxThread. Este número está limitado a principio pelo número de threads que podem ser processados ao mesmo tempo por um multiprocessador, bem como pelo número de blocos sendo processados em paralelo pelos multiprocessadores. Nota se que o numero de threads no bloco denominado nThread não necessariamente possui a mesma dimensão do maxThread. No fluxograma do algoritmo, a imagem de entrada em escala de cinza é denotada por *I* e a imagem de saída por *dst*.

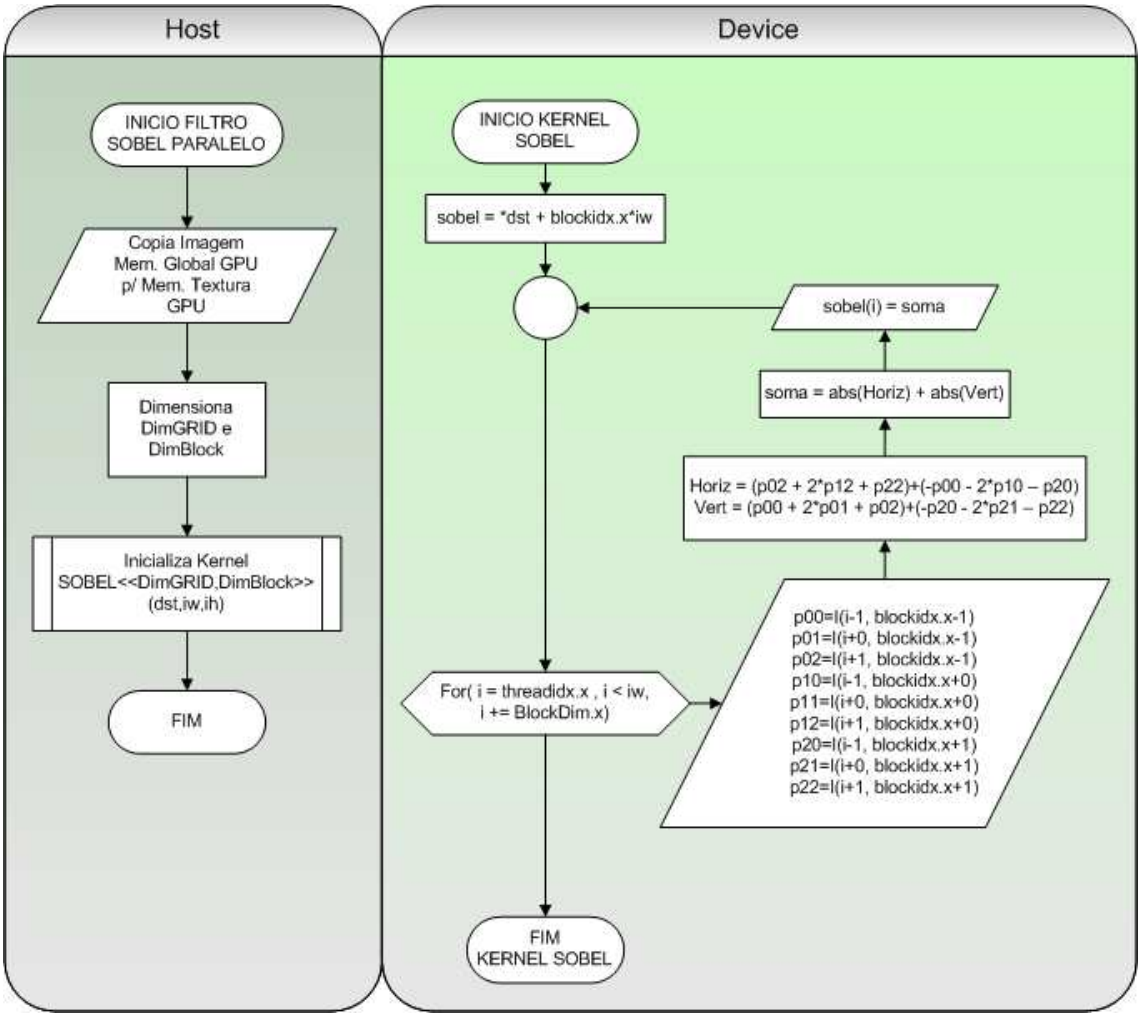


Figura 4.9: Fluxograma do algoritmo Sobel paralelo em GPU

A fim de proporcionar melhor compreensão da execução do algoritmo filtro Sobel, segue a demonstração de um exemplo. A imagem que será aplicada ao filtro possui a dimensão de 6x10 conforme Figura 4.10(a), sendo esta imagem em escala de cinza. A divisão do grid em blocos desta imagem é caracterizada como 1D, onde cada bloco representa uma linha da imagem. O número máximo de threads no bloco é definido pela melhor performance segundo as restrições apresentadas na seção (4.2), para efeito de visualização de maxThread menor que nThread, neste caso, assumiremos o maxThread com valor 3 e nThread igual a 10.. Desta maneira, a modelagem dos blocos no exemplo é determinada por DimGrid(6,1) conforme Figura 4.10(b).

Imagem = GRID

10	10	20	25	20	21	20	25	30	30
10	10	18	29	35	40	41	42	40	40
10	10	20	30	36	46	50	46	45	44
10	10	21	32	39	46	50	70	70	70
10	10	22	30	38	44	50	80	80	80
10	10	19	31	36	40	70	80	80	80

6x10

(a)

Blocos

B(0)	10	10	20	25	20	21	20	25	30	30
B(1)	10	10	18	29	35	40	41	42	40	40
B(2)	10	10	20	30	36	46	50	46	45	44
B(3)	10	10	21	32	39	46	50	70	70	70
B(4)	10	10	22	30	38	44	50	80	80	80
B(5)	10	10	19	31	36	40	70	80	80	80

(b)

Figura 4.10: Modelagem Sobel do Grid em Blocos (a)Imagem (b) Blocos do Grid

Seguindo o fluxograma do algoritmo, transfere-se então os dados da imagem contidos na memória global da GPU para memória de Textura da GPU. Em seguida é alocado espaço de memória na GPU para variável *dst*, para então convocar o kernel.

No kernel, o primeiro processamento é indexar a variável chamada *sobel* seguindo o endereçamento apontado por *dst*. Esta indexação é necessária para armazenar o resultado do filtro de forma coerente com as posições dos dados processados. Conforme o dimensionamento do bloco e dos threads apresentado anteriormente, observa-se na Figura 4.11 que, para cada interação do laço de repetição (iteração i), é transferido da memória de textura para o registrador todos os dados da janela de vizinhança referentes a cada um dos pixels correntes nos threads (maxThread), de cada um dos blocos sendo processados. A partir do momento que a janela foi carregada para o registrador, é então computado o valor do gradiente multiplicando cada elemento dessa janela ponto a ponto com o operador Sobel, tendo a soma deste produto armazenada para variável *Horiz* (o gradiente horizontal) e armazenada para variável *Vert* (o gradiente vertical). O resultado final é obtido somando o valor absoluto das duas variáveis armazenando na variável *sobel*. O passo do laço de repetição é performado pelo maxThread, sendo concluído após os threads percorrerem todos os dados do bloco, que no caso é a largura da imagem.

Figura 4.11: Processamento paralelo do filtro Sobel

Uma observação a ser notada é que na montagem da janela de vizinhança para um pixel corrente de borda, a memória de Textura replica o valor da borda, tendo este tratamento realizado em hardware (NVIDIA, 2010).

4.5 Reconstrução Morfológica "H-MIN"

A idéia do filtro reconstrutivo morfológico HMIN visto anteriormente na seção 3.3, é eliminar os mínimos regionais da imagem. Os míninos regionais são definidos pela união dos pixels conexos que apresentam o mesmo mínimo local. A proposta do algoritmo paralelo aqui desenvolvido é baseado na solução do algoritmo seqüencial apresentado por (VINCENT, 1993).

O algoritmo paralelo desenvolvido segue a equação 4.5.1 que tem a variável *IR* caracterizado pela imagem reconstruída, sendo este o resultado da reconstrução HMIN, a variável *I* sendo a imagem de entrada e a variável *M* caracterizado pela imagem marcador também visto na seção 3.3. Os índices *(p,q)* representam as posições de todos os pontos da matriz imagem e *(i,j)* representam a posição do ponto vizinho de cada ponto *(p,q)*. $N_{MG}(p,q)$ representa o conjunto de pontos vizinhos do ponto central *(p,q)* da imagem *M*, tendo a vizinhança definida por *G*, podendo assumir os valores 4 ou 8, isto é, vizinhança 4 ou vizinhança 8 conforme Figura 4.12.

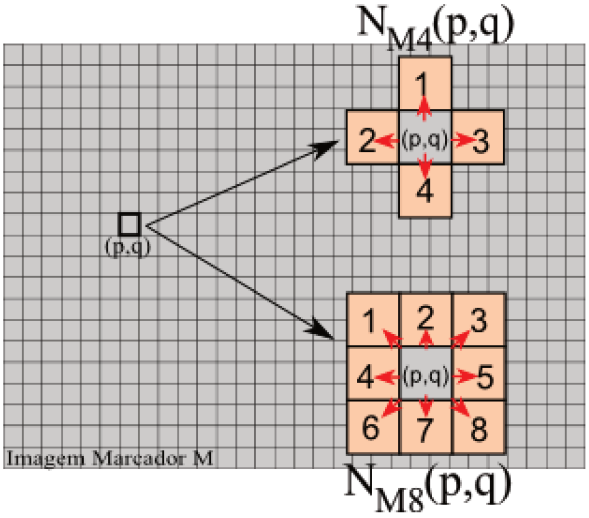


Figura 4.12: Vizinhança do Pixel

Uma vez compreendido as variáveis que compõem a equação 4.5.1, a interpretação é feita tendo cada ponto *p,q* da imagem reconstruída {*IR(p,q)*}, receber o valor mínimo do ponto *M(i,j)* da imagem marcador pertencente a vizinhança $N_{MG}(p,q)$, tal que o valor contido no ponto vizinho *M(i,j)* da imagem marcador tem que ser menor que o valor contido no ponto corrente *M(p,q)* também da imagem marcador. O valor *M(i,j)* de saída deste processo é comparado com o valor contido no ponto *I(p,q)* da imagem de entrada, tendo o sinal $\vee$ caracterizado pelo valor máximo entre estes dois valores.

$$IR(p,q) \leftarrow \Big(MIN\big(M(i,j)\big),\ M(i,j) \in N_{MG}(p,q)\ \forall\ M(i,j) < M(p,q)\Big) \vee I(pq) \qquad (4.5.1)$$

O filtro de reconstrução HMIN é obtido aplicando-se sucessivas vezes a equação acima, até que a imagem reconstruída esteja estabilizada, ou seja, no momento em que se aplica a equação, não houver nenhuma alteração dos valores da imagem de entrada, esta é considerada como estabilizada.

Os fluxogramas da Figura 4.13 e Figura 4.14 demonstram o algoritmo que executa o filtro HMIN paralelo. Ressaltando o que foi dito na seção 4.3, o fluxograma modelado apresenta o código no host e no device, observando que o bloco retangular com duas faixas nas extremidades denota uma chamada do kernel na GPU. O algoritmo desenvolvido possui 2 etapas, tendo a primeira etapa 1 kernel, e a segunda 4 kernels. A primeira etapa tem a principal finalidade de construir a imagem *M* em função do valor H (atributo altura). A segunda etapa, de posse da imagem de entrada *I*, fará o processo de reconstrução na imagem marcador *M* seguindo a equação 4.5.1 apresentada anteriormente. Como esta segunda etapa faz uso da memória compartilhada, existe um kernel para de fato fazer a reconstrução *IR(p,q)* no domínio de cada bloco, e 3 outros kernels para propagar a reconstrução entre os blocos. Isto se faz necessário devido a não visibilidade entre os blocos quando se trabalha com memória compartilhada, e também pelo ganho de performance na convergência da estabilização.

A fim de detalhar o que os kernels desempenham na realização do algoritmo, será apresentado adiante à concepção de cada kernel. Iniciando com o kernel *MAKE*, o qual executa a soma do valor da imagem *I* por um escalar *H*, é modelado o grid em blocos 2D tendo cada bloco (dim_X x dim_Y) threads para processamento. No código, a atribuição *img = I(p,q)* significa que a memória registrador de cada multiprocessador irá receber os dados de uma fatia da imagem da memória de textura, correspondente a dimensão de um dado bloco(bx,by). Este irá processar a

soma pelo escalar *H* e atribuir o resultado na variável *mk*. A variável marker irá receber o resultado do processamento de truncagem que limitará o valor máximo de *mk* em 255. Em seguida, os dados de *marker* são transferidos para imagem marcador *M* na memória global.

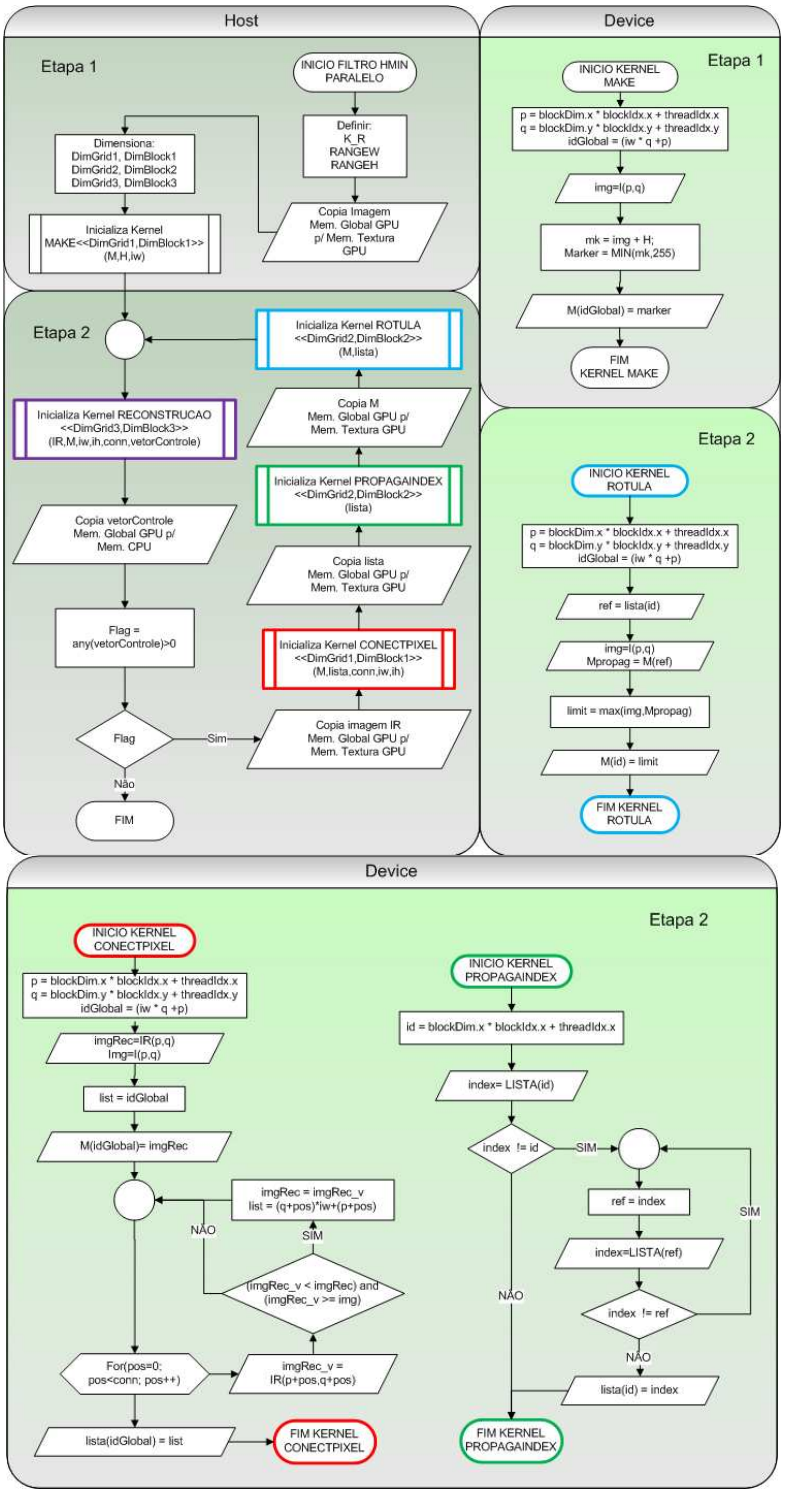


Figura 4.13: Fluxograma do algoritmo HMIN paralelo em GPU (parte 1)

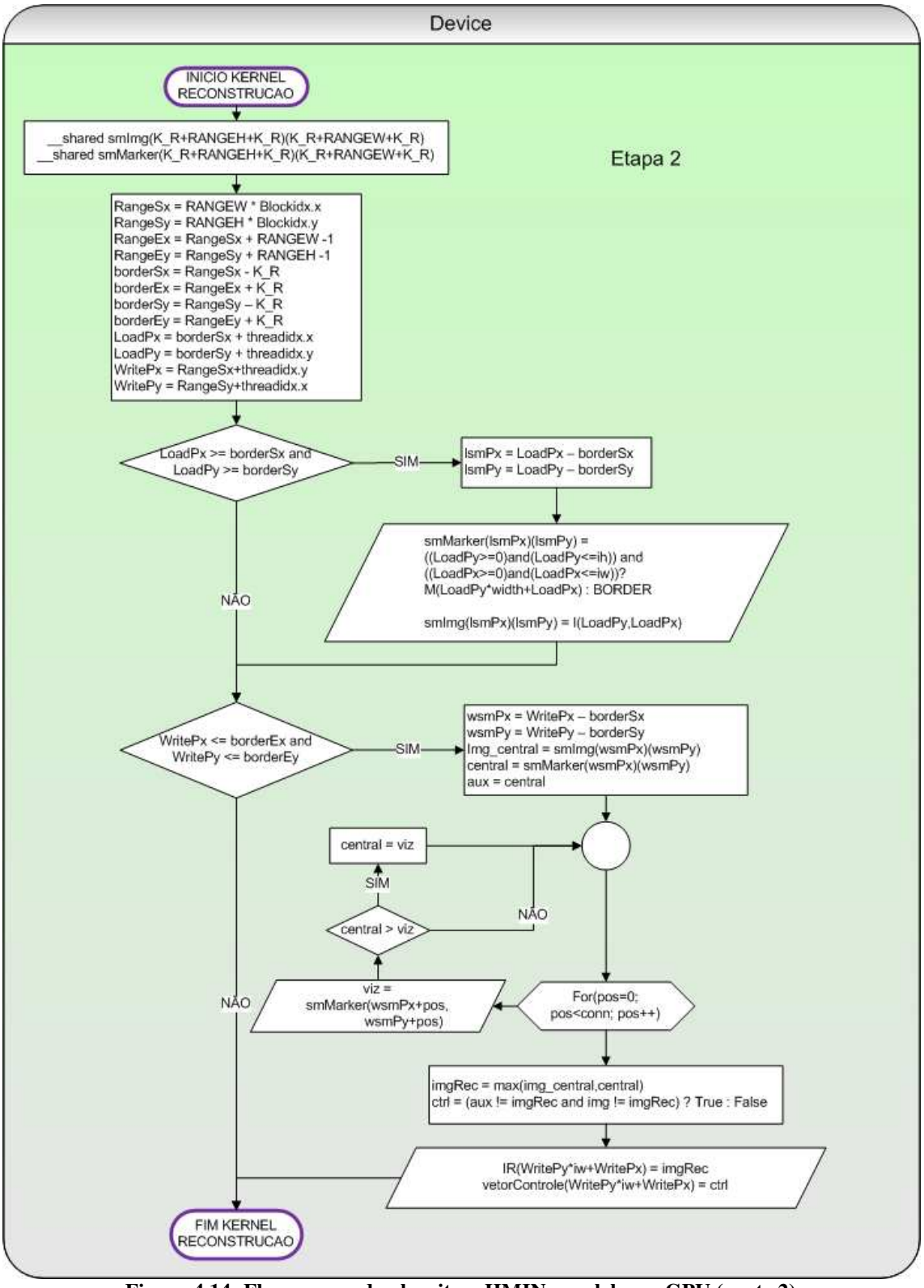


Figura 4.14: Fluxograma do algoritmo HMIN paralelo em GPU (parte 2)

O segundo kernel a ser abordado seguindo o fluxo do algoritmo é o kernel RECONSTRUCAO. A aplicação da equação 4.5.1 está implementada neste kernel, que por sua vez utiliza a memória compartilhada. Como a natureza do problema depende da relação de dados vizinhos no processamento de um dado, e ao mesmo tempo os dados contidos na memória compartilhada de cada bloco não são visíveis aos demais blocos, surge com isto o problema de informações contidas nas fronteiras do bloco conhecido como borda do bloco. Para contornar este problema, é adotada uma estratégia diferente das vistas até o momento, no dimensionamento dos blocos do grid, cuja idéia é adicionar uma borda na constituição do bloco de tal forma a introduzir em cada bloco os valores necessários dos blocos fronteiras, e garantir também que tais bordas não sejam processadas.

A estratégia adotada para modelagem dos blocos com borda no algoritmo inclui 3 novas variáveis, que são *RANGEW*, *RANGEH* e *K_R*. As variáveis *RANGEW* e *RANGEH* servem para especificar qual a faixa de dados dentro do bloco que serão processados, e a variável *K_R* para especificar o raio da borda, que no caso deste algoritmo assume o valor 1 devido à relação de vizinhança envolvida no processamento, conforme Figura 4.15. Neste sentido, o número de threads no bloco é dimensionado como *DimBlock((K_R+RANGEW+K_R),(K_R+RANGEH+K_R))*, e o número de blocos no grid por *DimGrid((w/RANGEW),(h/RANGEH))*, sendo *w* e *h* as dimensões da imagem de entrada.

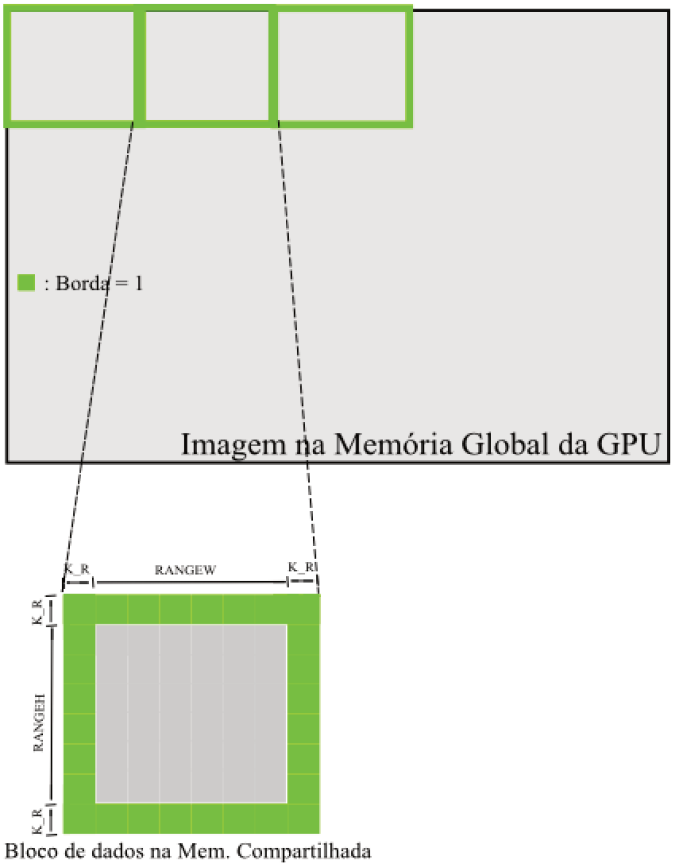


Figura 4.15: Modelagem da borda do bloco pelo host

Visto esta questão da modelagem no host, iremos agora demonstrar a modelagem do device. Para controle de leitura dos dados da imagem para a memória compartilhada de cada multiprocessador bem como para processamento e escrita do resultado dos multiprocessadores na memória global, deve-se calcular os índices de acesso à memória global e os índices de acesso à memória compartilhada dos blocos. Esta leitura e escrita dentro do kernel é realizado em duas fases.

A primeira fase possui a finalidade de transferir os dados dos blocos da imagem modelados com as bordas da memória global para a memória compartilhada. Para isto os índices de acesso chamados de *(loadPx,loadPy)* e *(lsmPx,lsmPy)* foram utilizados. A atribuição *smMarker(lsmPx)(lsmPy)=M(loadPy*w+loadPx)* significa que a variável *smMarker* nas posições apontadas pelos índices *(lsmPx,lsmPy)* irá receber todos os dados do bloco paralelamente referente a variável *M* nas posições apontadas pelos índices *(loadPx,loadPy)*. No entanto, como a modelagem do bloco é composta de informações contidas nas fronteiras do bloco, tais índices precisam ser construídos de tal forma a incluir estas bordas em cada bloco

paralelamente. Portanto, as variáveis índices *(loadPx,loadPy)* são constituídas então por *loadPx = borderSx + threadIdx.x* e *loadPy = borderSy + threadIdx.y*, e as variáveis *(lsmPx,lsmPy)* por *lsmPx = loadPx – borderSx* e *lsmPy = loadPy – borderSy*, Como se percebe, esta envolve a criação de variáveis auxiliares chamadas de *(rangeSx,rangeSy)*, *(rangeEx,rangeEy)*, *(borderSx,borderSy)*, *(borderEx,borderEy)* para modelagem destes índices. As variáveis auxiliares *(rangeSx,rangeSy)* caracterizam o inicio da posição do dado que será transferido em todos os blocos, formado por *rangeSx = RANGEW * blockidx.x* e *rangeSy = RANGEH * blockidx.y*. Por sua vez, as variáveis auxiliares *(rangeEx,rangeEy)* caracterizam o final da posição do dado que será processado em todos os blocos, modelados como *rangeEx = rangeSx + RANGEW – 1* e *rangeEy = rangeSy + RANGEH – 1*. As variáveis auxiliares *(borderSx,borderSy)* caracterizam o inicio da posição do dado de borda que será carregado em todos os blocos dado por *borderSx = rangeSx – K_R* e *borderSy = rangeSy – K_R*. Por fim, as variáveis auxiliares *(borderEx,borderEy)* caracterizam o final da posição do dado de borda que será carregado em todos os blocos, obtido por *borderEx = rangeEx + K_R* e *borderEy = rangeEy + K_R*.

A segunda fase tem a finalidade de realizar o processamento das informações dos blocos sem as bordas e armazenar o resultado de forma coerente na memória global. Para esta fase é preciso de índices diferentes dos criados na primeira fase, pois eles devem acessar somente as posições da faixa de dados que realmente serão processados dentro dos blocos bem como as posições dos dados resultantes do processamento para a memória global. Tais índices foram chamados de *(wsmPx,wsmPy)* e *(writePx, writePy)*. A atribuição *IR(writePy*w+writePx) = imgRec(wsmPx)(wsmPy)* significa que a variável *IR* nas posições apontadas pelos índices *(writePy*w+writePx)* da memória global irá receber o resultado do processamento da variável *imgRec* nas posições apontadas pelos índices *(wsmPx,wsmPy)* da memória compartilhada de todos os blocos. Para isto, as variáveis índices *(writePx, writePy)* são constituídas por *writePx = rangeSx + threadIdx.x* e *writePy = rangeSy + threadIdx.y*, bem como as variáveis índices *(wsmPx,wsmPy)* são constituídas por *wsmPx = writePx – borderSx e wsmPy = writePy – borderSy*. O processamento é realizado manipulando os índices *(wsmPx,wsmPy)* dentro dos blocos, sendo este igual a todos os outros processamentos visto até agora. Um exemplo seria a atribuição *central = smMarker(wsmPx)(wsmPy)* (visto na Figura 4.14) significando que a memória registrador de cada multiprocessador irá receber os dados de uma fatia do bloco da memória compartilhada, correspondente a dimensão definida por (RANGEW,RANGEH). Cada

ponto desta fatia do bloco corresponde a um pixel central *(p,q)* do processamento denominado de bloco central. A atribuição *viz = smMarker(wsmPx+pos)(wsmPy+pos)* representa também o carregamento de uma fatia do bloco nos multiprocessadores, onde cada ponto desta fatia representa o vizinho *(i,j)* de cada ponto da fatia central. Esta fatia do bloco que representa o vizinho *(i,j)* foi denominado de bloco vizinho. A Figura 4.16 apresenta o fluxo das fases de leitura e escrita abordando o papel de cada variável na constituição dos índices.

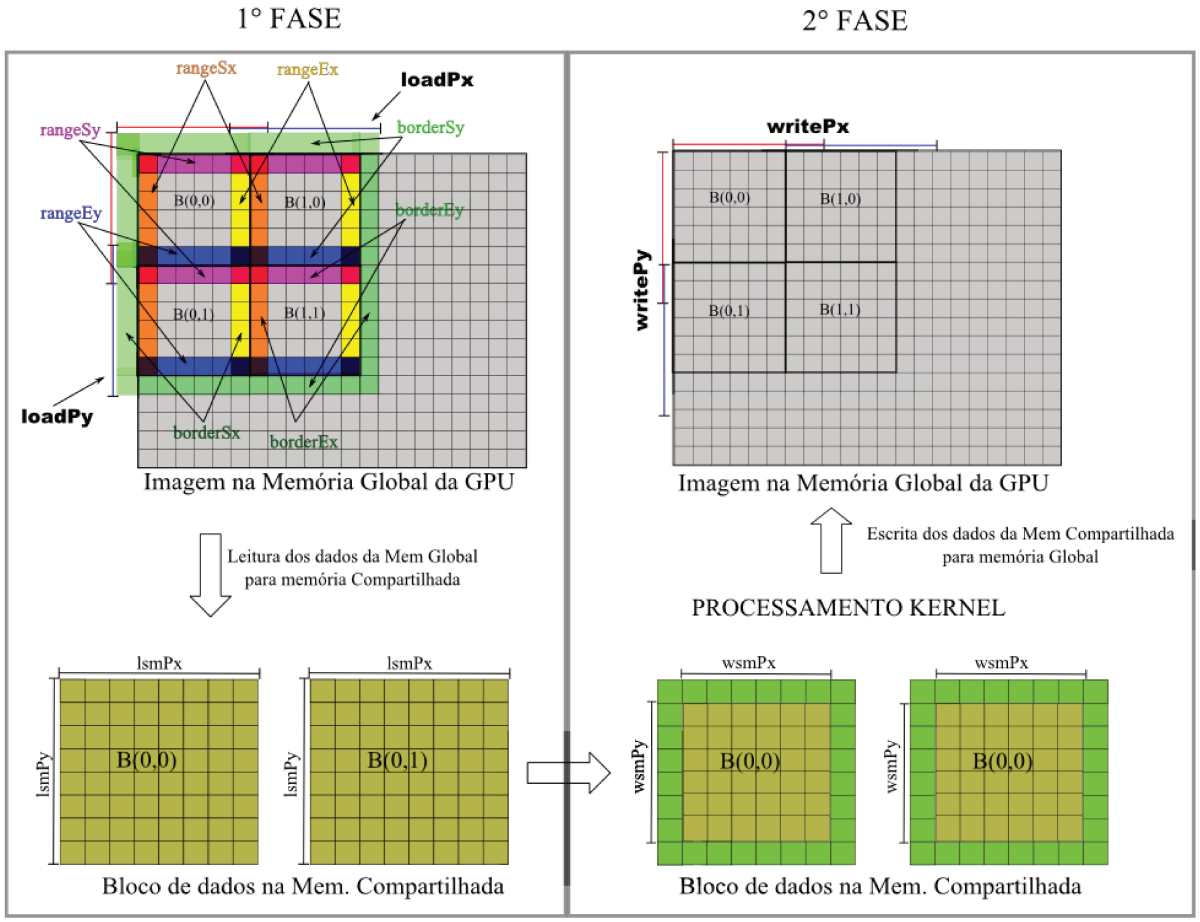


Figura 4.16: Estruturação de leitura e escrita do bloco com borda dentro do device

Para demonstrar a manipulação das variáveis auxiliares no cálculo dos índices dentro do kernel, a Figura 4.17 apresenta um exemplo dessas variáveis para a dimensão X da imagem, visto que as duas dimensões (*X,Y*) obedecem às mesmas condições e que para efeito de demonstração, defini-se RANGEW=6 e K_R=1, tendo dimGrid(3,3) e dimBlock(8,8) conforme a modelagem

dos blocos por parte do host vista anteriormente. Observa-se que apesar do *writePx* envolver a borda, existe a instrução *if* que inibe o processamento envolvendo esta borda.

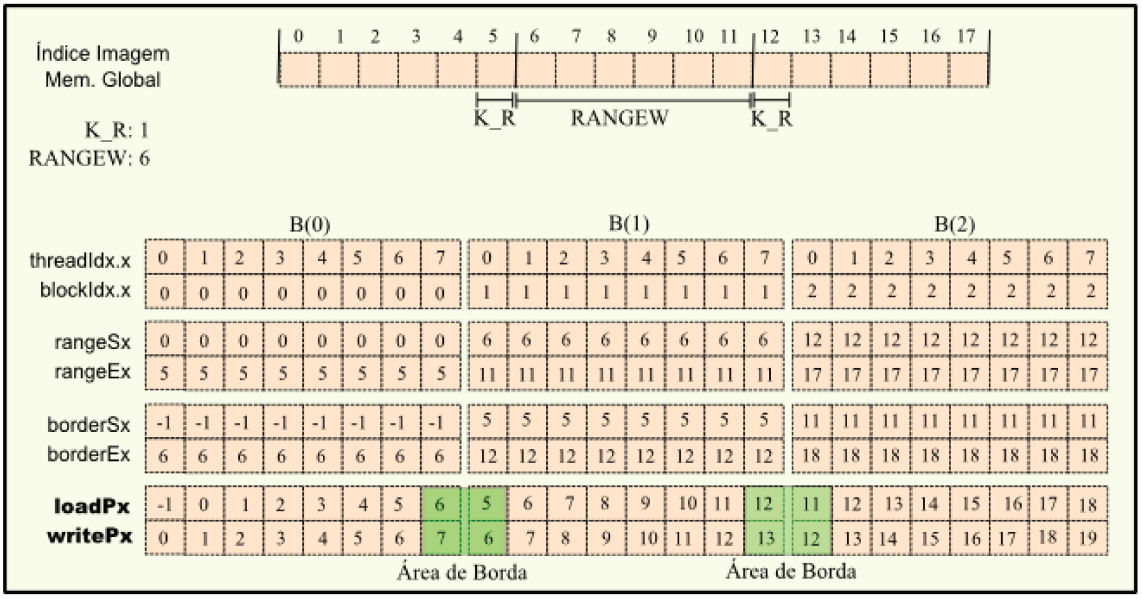


Figura 4.17: Calculo da Indexação dos blocos com borda

O detalhamento da reconstrução descrita anteriormente pela equação 4.5.1, é demonstrada na Figura 4.18. A primeira fase contempla a transferência dos valores da imagem (*I*) contido na memória de textura, bem como os valores da imagem marcador (*M*) contido na memória global para a memória compartilhada, representados pelas variáveis *smImg* e *smMarker*. Na segunda fase, já se encontra disponível os dados na memória compartilhada, deste modo, a Figura 4.18 exemplifica o processamento para um bloco, relembrando que o número de blocos processados em paralelo é relativo ao número de multiprocessadores que cada placa gráfica possui. Neste sentido, é transferido para a variável *central* o bloco central, caracterizado pelo bloco sem as informações da borda. Para cada iteração (i,j) contida dentro do conjunto $N_{MG}(p,q)$, cada ponto do bloco central é comparado com o respectivo ponto do bloco vizinho (i,j) armazenando o menor valor desta comparação novamente na variável *central*. Pode se observar que todos estes pontos estão sendo executados em paralelo pelos threads contidos nos blocos. Ressalta-se que na iteração (i,j) seguinte, as informações contidas no bloco central foram atualizadas pelo resultado do processamento da iteração (i,j). Como resultado deste procedimento descrito anteriormente, o bloco central possuirá o menor valor em relação a seus vizinhos. Este bloco é em seguida

comparado com o bloco da imagem dado por *img_central* e extraído o valor máximo ponto a ponto, tendo o resultado atribuído para a variável *imgRec*. O resultado é transferido para a variável *IR* na memória global. É importante frisar que para controle da estabilização do algoritmo pelo CPU, foi criada uma variável chamada *vetorControle* que tem a finalidade de armazenar no device quais dados dos pixels da imagem reconstruída foram alterados em relação a imagem marcador, para então esta informação ser transferida da memória global da GPU para memória da CPU onde será realizada a verificação das alterações controlando a estabilização a cada interação.

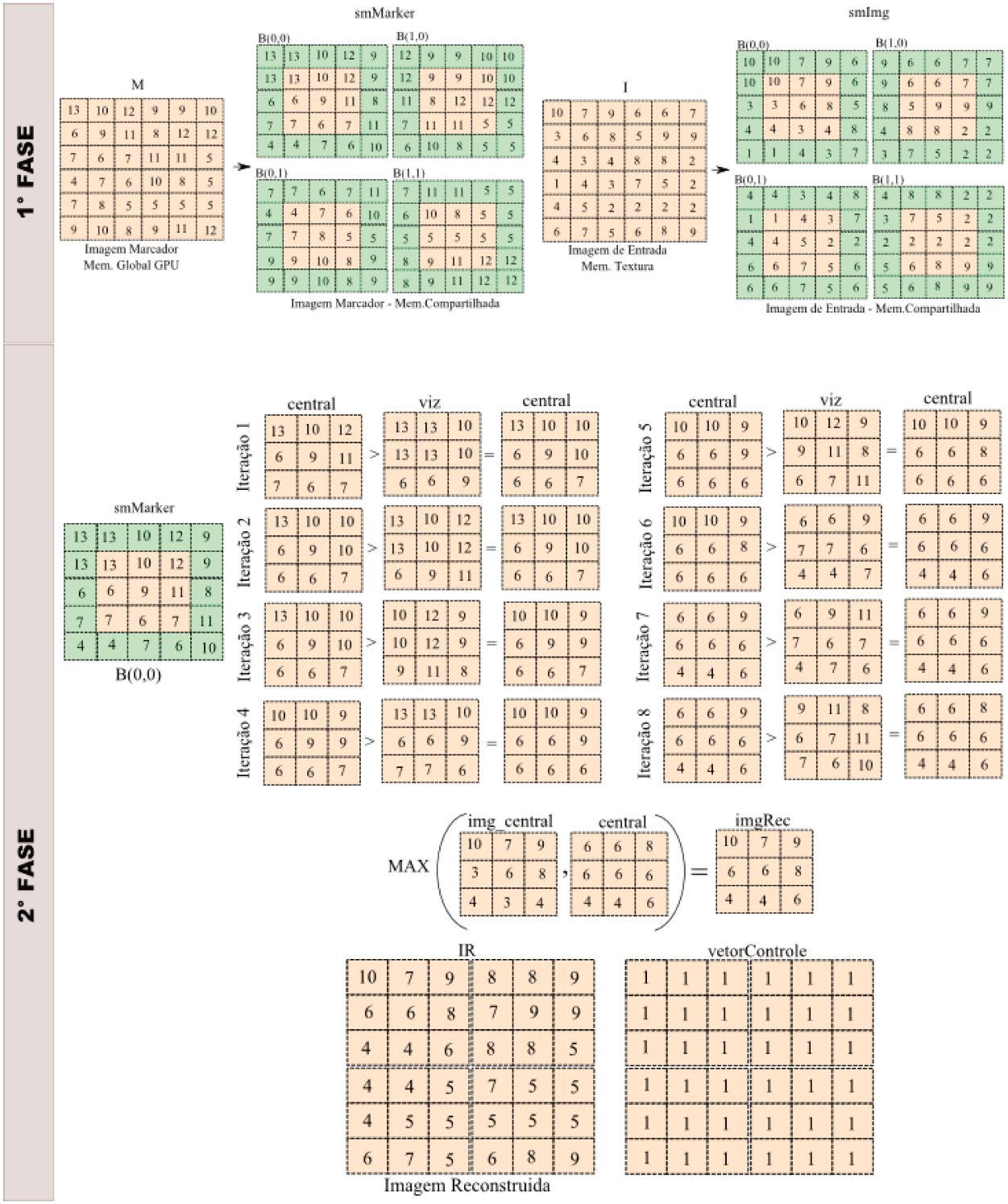


Figura 4.18: Cálculo do kernel RECONSTRUCAO

Seguindo com o detalhamento dos kernels, os próximos 3 kernels chamados CONECTPIXEL, PROPAGAINDEX e ROTULA são incorporados ao código com o propósito de propagar na imagem reconstruída a influência dos pixels conexos ($N_{MG}(p,q)$) conforme a equação 4.5.2, ressaltando que a operação é feita em toda a imagem incorporando portanto também a propagação entre os blocos. O método de propagação de rótulo desenvolvido é baseado no algoritmo implementado por (HAWICK, LEIST e PLAYNE, 2009). O efeito deste procedimento de propagação é reduzir o número de interações realizadas para estabilização da reconstrução, conseqüentemente aumentando a performance do algoritmo.

Portanto, a propagação utiliza o kernel CONECTPIXEL para construir a ligação de pixels que podem ser influenciado pelo mesmo pixel mínimo. Desta forma, após o kernel RECONSTRUCAO retornar a imagem reconstruída, esta é aplicada ao kernel CONECTPIXEL que examinará os pixels em relação a seus vizinhos, montando uma lista de equivalência entre os pixels conforme a seguinte condição:

$$lista(u) = p * w + q$$
$$lista(u) \leftarrow \left(i * w + j, MIN\big(M(i,j)\big) \in N_G(p,q) \forall \begin{cases} M(i,j) < M(p,q) \\ e \\ M(i,j) \geq I(p,q) \end{cases} \right) \tag{4.5.2}$$

Observe que u é denotado como a indexação da lista de equivalência que vai de *0* até *w*h* (tamanho da imagem). A divisão do DimGrid e DimBlock deste kernel, bem como a transferência de dados entre as memórias é igual a modelagem apresentada no Kernel MAKE.

O kernel PROPAGAINDEX vem contribuir na resolução da lista de equivalência, que contém os índices das posições relacionadas, como por exemplo, uma posição i desta lista está apontando para a posição j que aponta para a posição k, tendo a resolução de equivalência apontar as posições i e j para posição k, como mostra a Figura 4.19. Pelo fato de trabalhar com acessos de memória randômicos, o desempenho deste kernel é extremamente dependente da utilização da memória de textura para sua implementação, afetando o desempenho do algoritmo caso não a utilize.

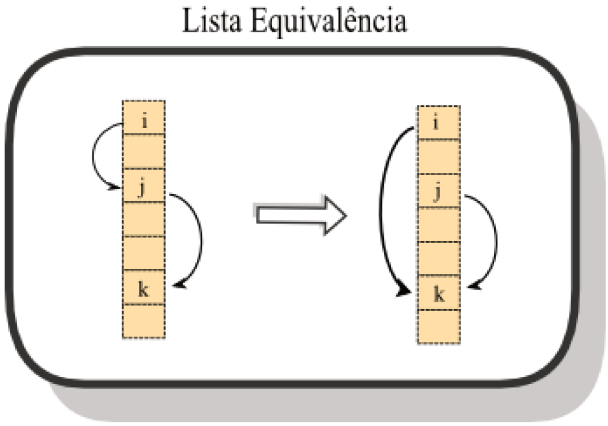


Figura 4.19: Resolução da lista de equivalência

O modo de resolver esta lista de equivalência é gerar um laço de repetição dentro do kernel que irá propagar os índices relacionados na lista, onde a finalização se dá quando toda equivalência estiver resolvida (Vide Figura 4.14). Como este kernel processa uma lista, o DimBlock é dado por DimBlock(*nThread*) e DimGrid dado por DimGrid($u \div nThread$).

Após a propagação das posições que estão referenciadas pela posição dos pixels mínimos, o kernel ROTULA irá transferir o máximo valor entre o valor contido na posição dos pixels mínimos obtido pelo kernel PROPAGAINDEX, e o valor do pixel respectivo da imagem de entrada. Para efeito de demonstração, a Figura 4.20 apresenta um exemplo dos passos da influência de propagação do pixel mínimo para uma iteração do laço de repetição.

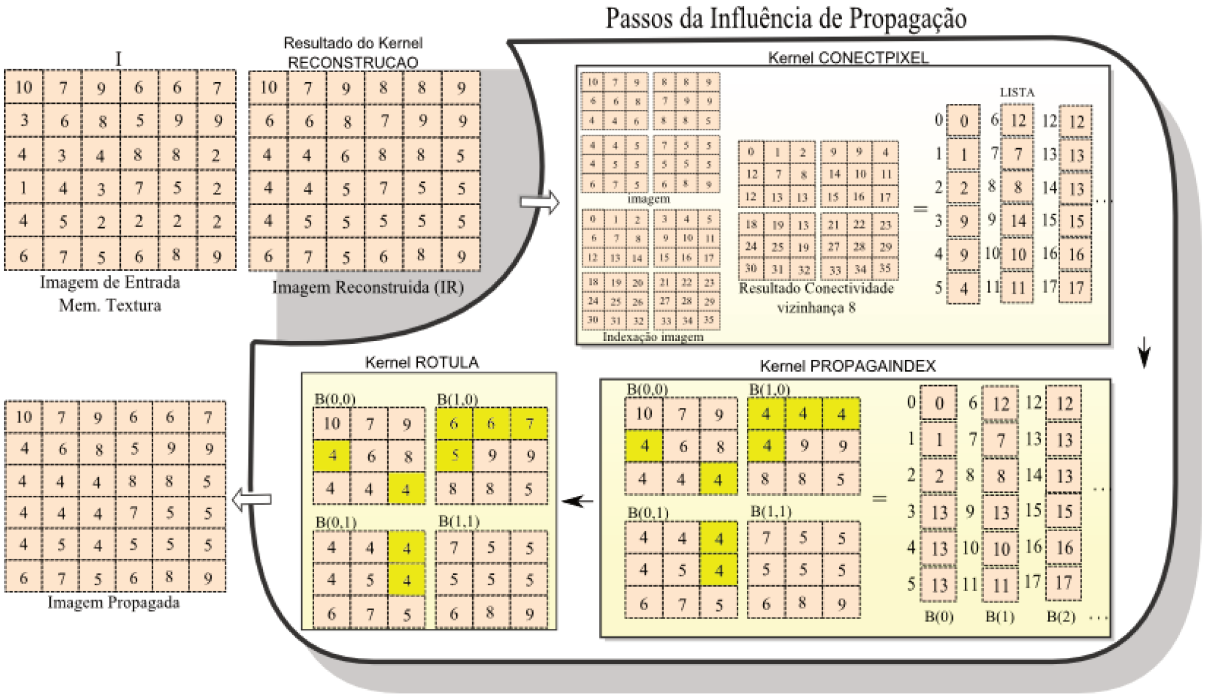


Figura 4.20: Processo de influência de propagação do pixel mínimo

Exposto de forma detalhada cada kernel que compõe o algoritmo morfológico reconstrutivo Hmin, de maneira a ilustrar o algoritmo sob uma visão completa de seu processamento, a Figura 4.21 apresenta o laço de repetição contendo todos os kernels sendo executados, onde depois de construído a imagem marcador na primeira etapa, caracterizado pelo MAKE, as iterações i demonstram o resultado do processamento em cada kernel sendo executado na segunda etapa, bem como a alteração do valor do pixel decorrente deste processamento, destacado pela cor amarela. A variável vetorControle determina se ouve alguma alteração no kernel RECONSTRUCAO, assumindo o valor false indicando que a imagem já está totalmente reconstruída, neste caso finalizando o algoritmo HMIN.

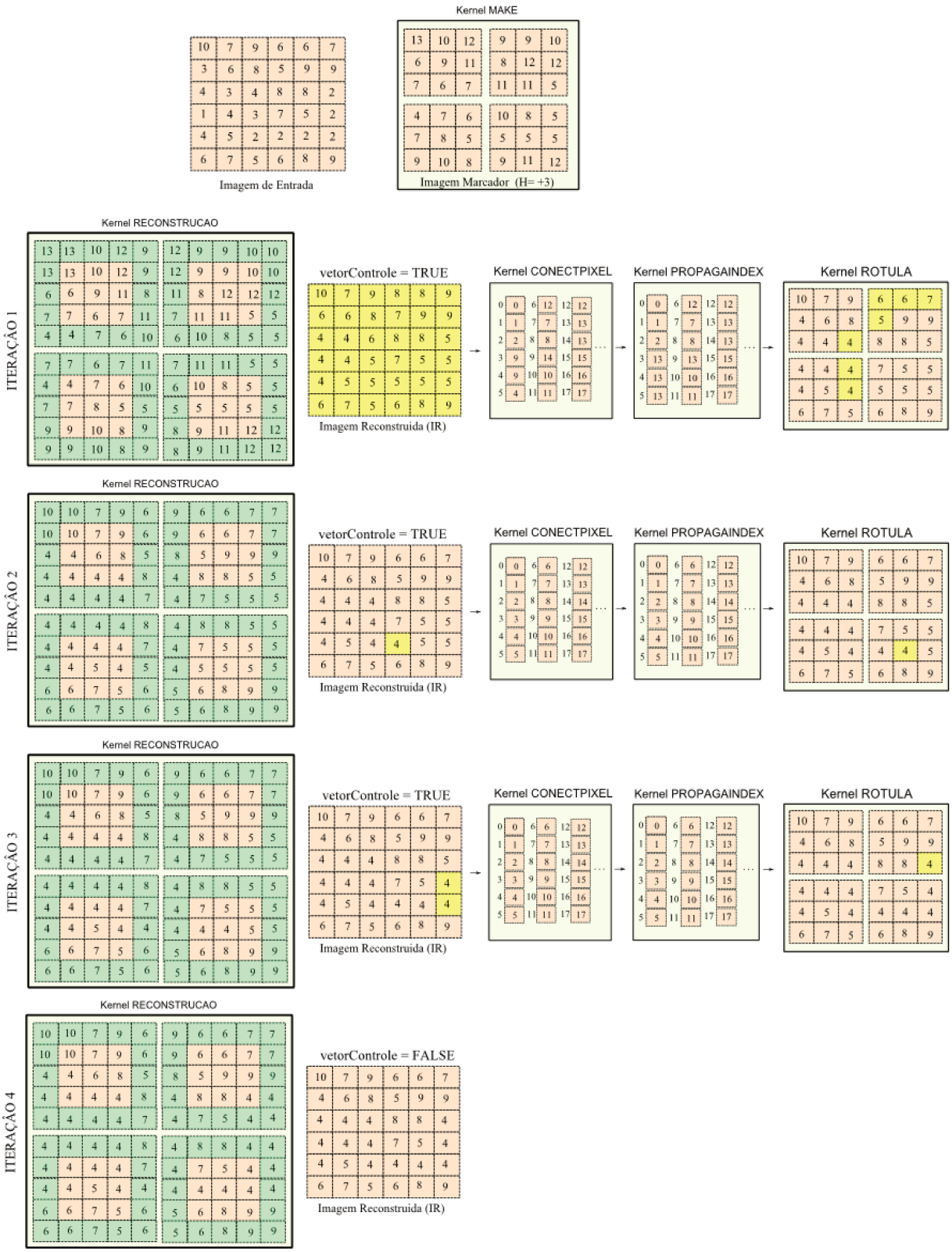


Figura 4.21: Execução do filtro morfológico reconstrutivo HMIN

4.6 Transformada Watershed

A transformada Watershed é utilizada para segmentação de imagem, vista na seção 3.4. A proposta do algoritmo paralelo aqui desenvolvido é uma evolução da solução apresentada por (VITOR, FERREIRA e KÖRBES, 2009).

O algoritmo paralelo desenvolvido possui 4 etapas, conforme a Figura 4.22. A primeira etapa consiste em encontrar a descida mais íngreme do pixel corrente, realizando um caminho até um dado pixel mínimo, sendo o caminho de cada pixel apontado pelo índice da posição onde o valor do pixel vizinho é menor. Caso existam valores iguais, ele irá apontar para o primeiro que encontrar. Observa-se que o resultado desta etapa pode ser visto pelas setas vermelhas. A segunda etapa consiste em eliminar os *plateaus*, caracterizado pelos pixels que não possuem apontamento de descida (todos os valores dos pixels vizinhos são maiores ou iguais ao valor do pixel corrente); e que não são mínimos regionais (conjunto de pixels conexos onde os vizinhos deste, apontam para algum pixel deste conjunto). Para atribuir o apontamento nos pixels do *plateaus*, será propagado o apontamento dos pixels da vizinhança do *plateaus* que estejam apontando para alguma descida. Esta é representada na Figura 4.22 pelas setas azuis. A terceira etapa tem a função de unir os pixels mínimos conexos atribuindo a cada mínimo um rótulo distinto, podendo ser visto pelos rótulos em verde e azul. E por fim, a quarta etapa é responsável por propagar os rótulos para os pixels percorrendo o caminho gerado por cada pixel, segmentando as duas regiões.

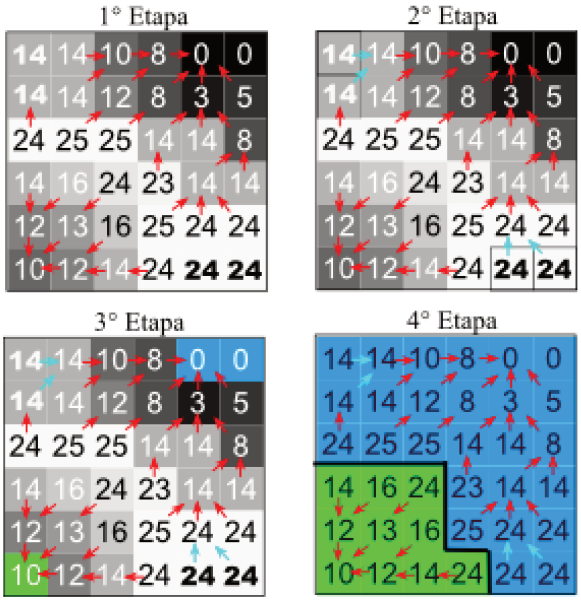


Figura 4.22: Etapas do algoritmo Watershed

Observado a idéia geral do algoritmo, neste momento serão apresentadas as formulações matemáticas abordando de maneira detalhada cada etapa deste processo.

A solução da primeira etapa consiste em resolver a equação 4.6.1, tendo o L caracterizado pela imagem rotulada e I definido como a imagem de entrada. Os índices (p,q) representam as posições de todos os pontos da matriz imagem e (i,j) representam a posição do ponto vizinho de cada ponto (p,q). $N_{XG}(p,q)$ representa o conjunto de pontos vizinhos do ponto central (p,q) da imagem X, tendo X assumindo as diretivas I ou L, e G definido a vizinhança do ponto, podendo assumir os valores de vizinhança 4 ou 8, conforme seção anterior. A expressão $u = (a*w+b)$ representa a transformação do índice da posição 2D (a,b) para 1D (u), sendo w a dimensão horizontal da matriz imagem. Nesta etapa é aplicado 1 kernel para processamento da equação que será exposta adiante.

$$\begin{gathered} L(p,q) \leftarrow q * w + p \\ L(p,q) \leftarrow \left(j * w + i,\ MIN\big(I(i,j)\big) \in N_{IG}(p,q) \forall\, I(i,j) < I(p,q)\right) \end{gathered} \tag{4.6.1}$$

A solução da segunda etapa é obtida conforme a equação 4.6.2. O resultado desta etapa consiste em aplicar sucessivas vezes à equação 4.6.2, até que todas as alterações nas posições dos apontadores estejam concluídas, caracterizando sua estabilização. Este processo de estabilização é controlado por dois laços de repetição, sendo um laço de repetição para estabilização dentro dos

blocos e o outro para estabilização entre os blocos. Observa-se que esta etapa utiliza apenas 1 kernel para resolver a equação e também faz uso da memória compartilhada para ganho de performance.

$$L(p,q) \leftarrow \left(L(i,j),\ L(i,j)\ \in\ N_{LG}(p,q)\ \forall \begin{cases} I(p,q) \geq 0\ e \\ L(i,j) < 0\ e \\ I(p,q) = I(i,j) \end{cases} \right) \tag{4.6.2}$$

Logo adiante na terceira etapa, os pixels que ainda não foram apontados para nenhuma posição são considerados como mínimos regionais, tendo todos os pixels mínimos conexos recebendo o mesmo rótulo. A estratégia utilizada nesta etapa foi aplicar o algoritmo de rotulação proposto por HAWICK et al. (2009), baseado em uma lista de referência que irá percorrer o caminho e propagar o pixel representante. Este algoritmo em si é inspirado na proposta "union-find" (TARJAN, 1983), embora aplicado em paralelo. Este processo também é iterativo, aplicando sucessivas vezes à equação 4.6.3, até a sua estabilização, possuindo 3 kernels que executa esta tarefa.

$$L(p,q) \leftarrow (L(i,j), L(i,j)\ \in N_{LG}(p,q)\ \forall\ L(p,q) \geq 0\ e\ L(i,j) \geq 0) \wedge L(p,q) \tag{4.6.3}$$

Por fim na quarta etapa que possui a finalidade de propagar o rotulo pelo caminho gerado pelos pixels, é aplicado o mesmo kernel de propagação utilizado na terceira etapa, sendo necessário antes, um kernel auxiliar para ajustar as informações na matriz rotulada *L*.

Os fluxogramas das Figura 4.23 à Figura 4.25 demonstram o algoritmo que executa a transformada Watershed. De maneira geral, a seqüência de eventos que executa o algoritmo é modelada montando a matriz imagem *L* contendo as informações dos caminhos de cada pixel, em função dos valores da imagem de entrada *I*, tendo os pixels dos *plateaus* e os mínimos regionais apontados para si mesmo com sinal positivo, e os demais apontados para suas descidas com o sinal do endereço negativo. Este está sendo executado no kernel INDEXDESCIDA. O kernel PROPAGADESCIDA soluciona a questão dos *plateaus*, os kernels AGRUPAPIXEL, PROPAGAINDEX e LABEL são para desempenhar o papel da terceira etapa, unindo os pixels mínimos conexos e elegendo o rotulo do grupo em relação ao pixel com endereço do menor

índice. Após esta etapa, foi feito um kernel chamado AJUSTAVETOR que serve para inverter os sinais negativos para positivos dos apontadores, que em seguida, é aplicado novamente no Kernel PROPAGAINDEX para propagar os rótulos pelos caminhos formados. Estes 2 kernels caracterizam a quarta etapa.

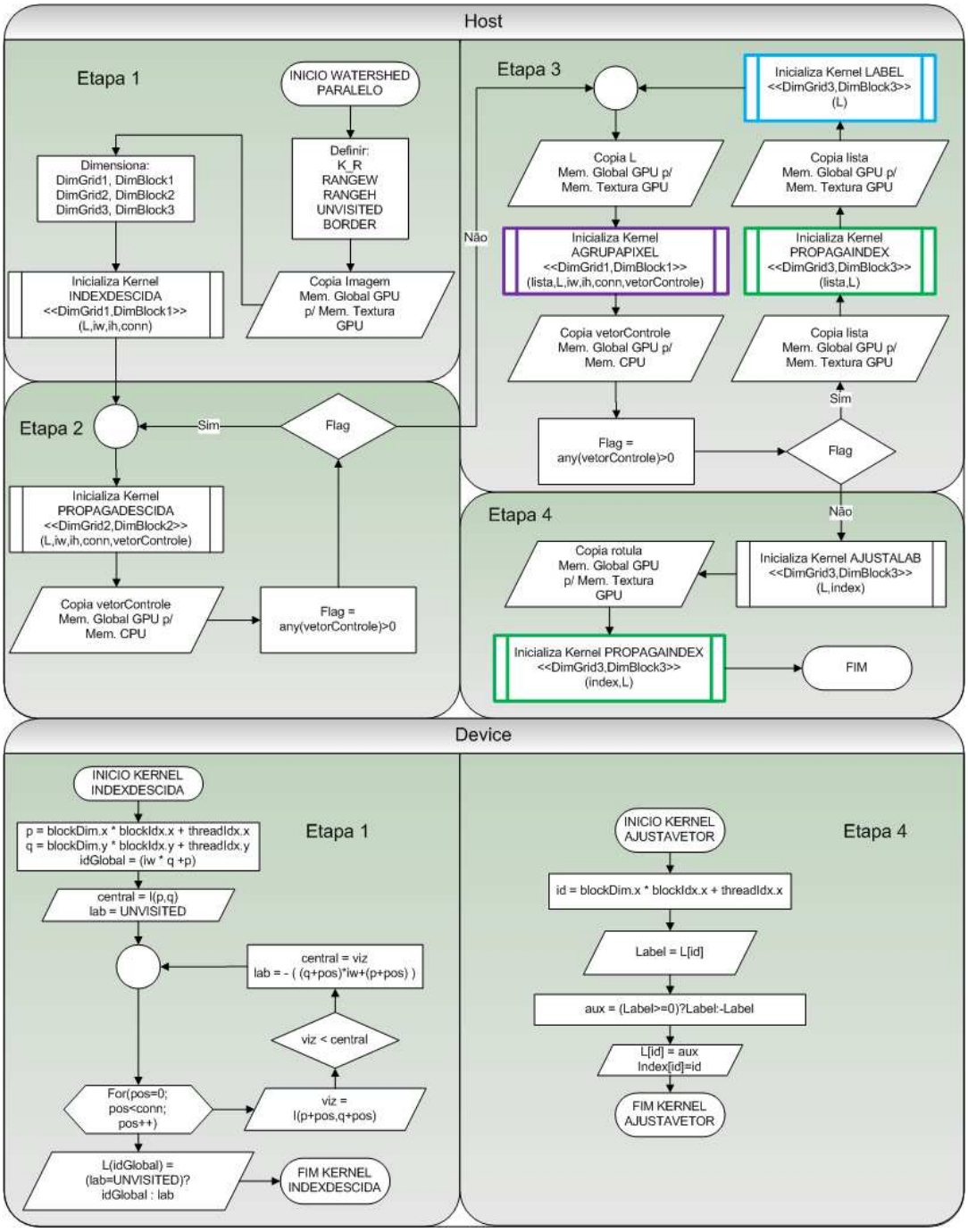


Figura 4.23: Fluxograma do algoritmo WATERSHED paralelo em GPU (parte 1)

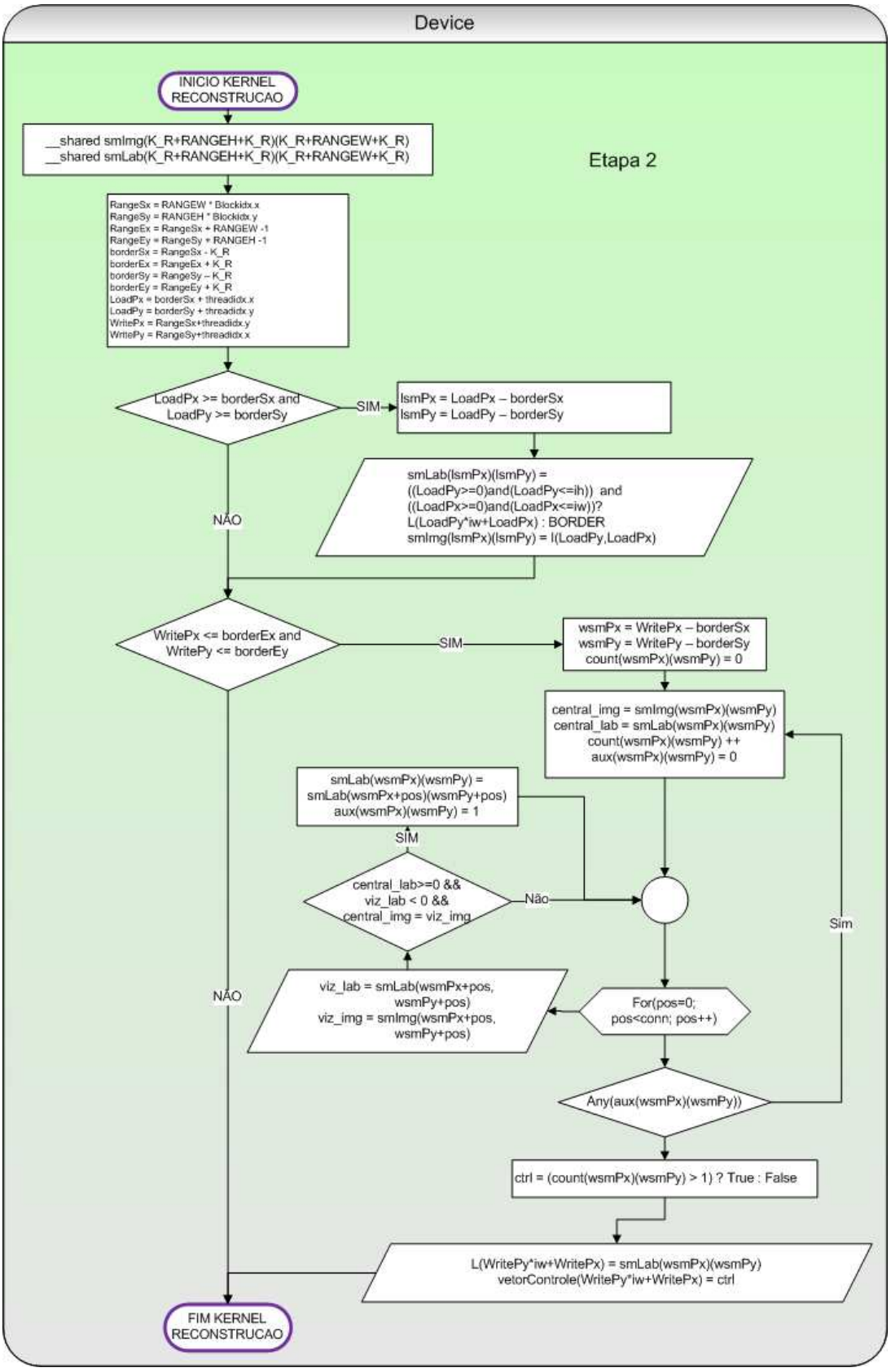


Figura 4.24: Fluxograma do algoritmo WATERSHED paralelo em GPU (parte 2)

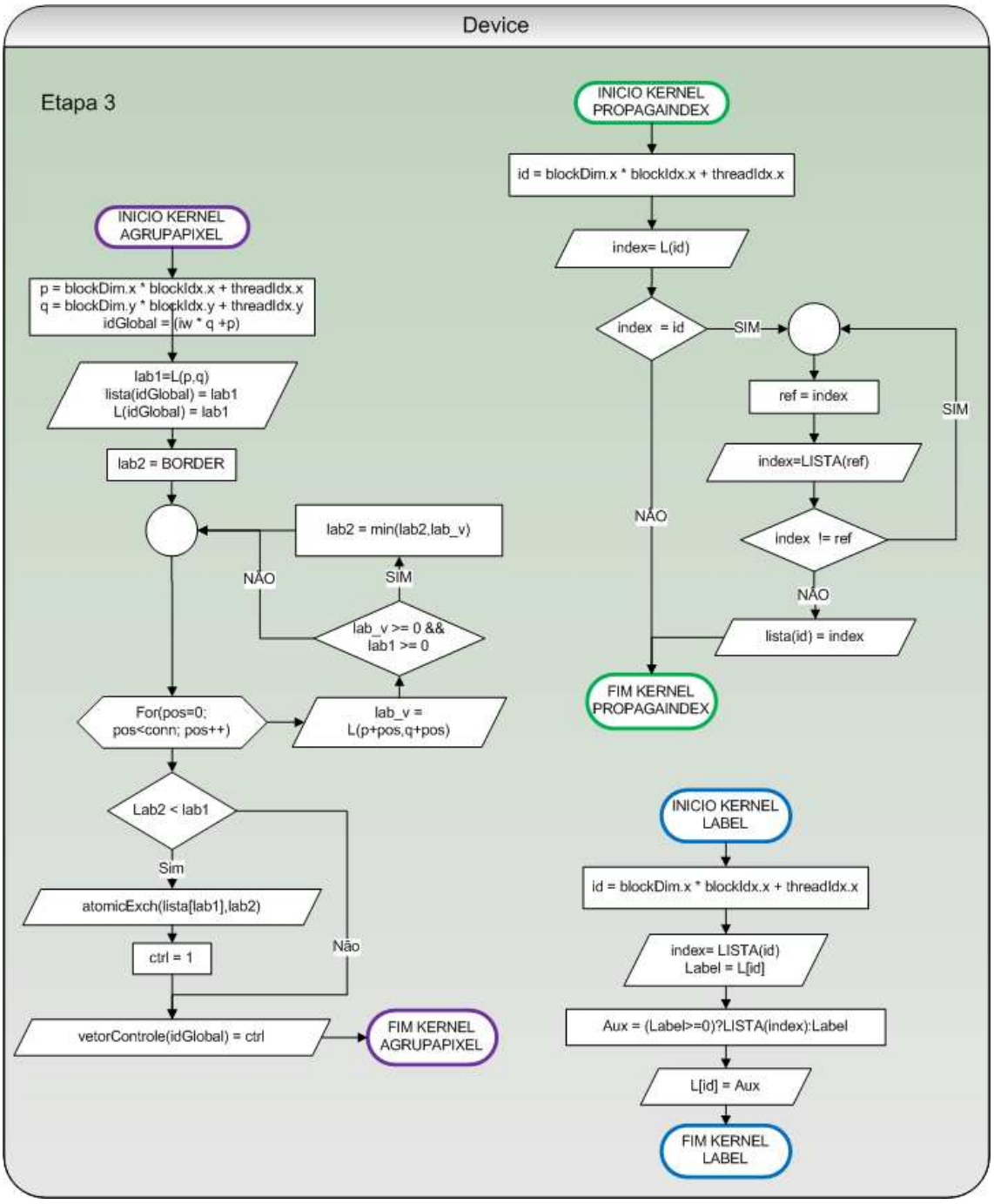


Figura 4.25: : Fluxograma do algoritmo WATERSHED paralelo em GPU (parte 3)

Embasado no algoritmo exposto, abordando as concepções da lógica proposta, as equações que envolvem os kernels, bem como o que cada kernel desempenha neste processo, será então demonstrado através de um exemplo o funcionamento do algoritmo.

A imagem de exemplo foi personalizada como uma matriz 8x10 como mostra a Figura 4.26. É importante ressaltar que a modelagem do grid imagem é realizada conforme as tarefas de cada kernel.

I

10	10	9	8	7	6	6	11	10	6
9	9	5	5	5	5	7	10	9	6
11	11	5	5	5	5	8	9	9	6
11	10	10	5	5	5	4	4	4	6
10	9	8	6	7	7	8	4	4	5
12	13	11	6	7	7	8	3	3	3
12	5	10	11	8	12	5	3	3	6
11	5	10	10	11	10	9	9	3	7

Matriz Imagem 8x10

Figura 4.26: Imagem para aplicação do Watershed

A matriz imagem *I* é transferida para memória de textura. O kernel INDEXDESCIDA, que é o primeiro a ser executado, irá atribuir os dados da imagem *I* na memória de textura para a variável *central* na memória registrador. A modelagem do grid para este kernel seguindo o exemplo foi dividida em 4 blocos de dimensão 4x5, sendo o número de threads alocados para processamento igual ao número de elementos do bloco. Portanto a variável DimBlock é (4,5) e DimGrid é (2,2), (vide Figura 4.27(a)). O processamento é dado pela iteração (i,j) contida dentro do conjunto $N_{I8}(p,q)$, tendo cada ponto do bloco da variável *central* comparado com o respectivo ponto do bloco vizinho (i,j), caracterizado pela variável *viz*, armazenando o menor valor desta comparação novamente na variável *central* e também criando os dados do endereçamento do caminho na variável *lab*, sendo esta atribuída com valor *"U"* representando um pixel não visitado e *"B"* contido na variável *viz* representando o pixel de borda, (vide Figura 4.27(b)). Após a iteração ser concluída, os pixels com valor *"U"* são apontados para si mesmos, e em seguida transferidos para a variável *L* na memória global, conforme Figura 4.27(c).

a)

I

10	10	9	8	7	6	6	11	10	6
9	9	5	5	5	5	7	10	9	6
11	11	5	5	5	5	8	9	9	6
11	10	10	5	5	5	4	4	4	6
10	9	8	6	7	7	8	4	4	5
12	13	11	6	7	7	8	3	3	3
12	5	10	11	8	12	5	3	3	6
11	5	10	10	11	10	9	9	3	7

Imagem de Entrada Mem. Textura

central

B(0,0)

10	10	9	8	7
9	9	5	5	5
11	11	5	5	5
11	10	10	5	5

B(1,0)

6	6	11	10	6
5	7	10	9	6
5	8	9	9	6
5	4	4	4	6

B(0,1)

10	9	8	6	7
12	13	11	6	7
12	5	10	11	8
11	5	10	10	11

B(1,1)

7	8	4	4	5
7	8	3	3	3
12	5	3	3	6
10	9	9	3	7

Imagem de Entrada Mem.Registrador

b)

B(0,0)

10	10	9	8	7
9	9	5	5	5
11	11	5	5	5
11	10	10	5	5

Interação 1

central

10	10	9	8	7
9	9	5	5	5
11	11	5	5	5
11	10	10	5	5

>

viz

B	B	B	B	B
B	10	10	9	8
B	9	9	5	5
B	11	11	5	5

=

central

10	10	9	8	7
9	9	5	5	5
11	9	5	5	5
11	10	10	5	5

lab

U	U	U	U	U
U	U	U	U	U
U	-10	U	U	U
U	U	U	U	U

Interação 2

10	10	9	8	7
9	9	5	5	5
11	9	5	5	5
11	10	10	5	5

>

B	B	B	B	B
10	10	9	8	7
9	9	5	5	5
11	11	5	5	5

=

10	10	9	8	7
9	9	5	5	5
9	9	5	5	5
11	10	5	5	5

U	U	U	U	U
U	U	U	U	U
-10	-10	U	U	U
U	U	-22	U	U

Interação 7

10	9	8	5	5
9	5	5	5	5
9	5	5	5	5
10	5	5	5	5

>

9	9	5	5	5
11	11	5	5	5
11	10	10	5	5
10	9	8	6	7

=

9	9	5	5	5
9	5	5	5	5
9	5	5	5	5
10	5	5	5	5

-10	-2	-12	-12	-13
U	-12	U	U	U
-10	-12	U	U	U
-31	-22	-22	U	U

Interação 8

9	9	5	5	5
9	5	5	5	5
9	5	5	5	5
10	5	5	5	5

>

9	5	5	5	5
11	5	5	5	5
10	10	5	5	5
9	8	6	7	7

=

9	5	5	5	5
9	5	5	5	5
9	5	5	5	5
9	5	5	5	5

-10	-12	-12	-12	-13
U	-12	U	U	U
-10	-12	U	U	U
-41	-22	-22	U	U

c)

lab

B(0,0)

-10	-12	-12	-12	-13
U	-12	U	U	U
-10	-12	U	U	U
-41	-22	-22	U	U

B(1,0)

-14	-15	-6	-9	U
U	-15	-6	-9	U
-36	-36	-36	-37	-38
-36	U	U	U	-38

B(0,1)

-41	-42	-33	-33	-33
-61	-61	-61	U	-43
-61	U	-61	-53	-53
-61	U	-61	-64	-64

B(1,1)

-36	-57	-57	-57	-58
U	-57	U	U	U
-54	-57	U	U	-58
-64	-67	-67	U	-68

Endereço dos apontadores de saída Mem.Registrador

L

-10	-12	-12	-12	-13	-14	-15	-6	-9	9
10	-12	12	13	14	15	-15	-6	-9	19
-10	-12	22	23	24	-36	-36	-36	-37	-38
-41	-22	-22	33	34	-36	36	37	38	-38
-41	-42	-33	-33	-33	-36	-57	-57	-57	-58
-61	-61	-61	53	-43	55	-57	57	58	59
-61	61	-61	-53	-53	-54	-57	67	68	-58
-61	71	-61	-64	-64	-64	-67	-67	78	-68

Endereço dos apontadores de saída Mem.Global

Figura 4.27: Processamento do kernel INDEXDESCIDA a) Modelagem do grid e transferência dos dados b) Processamento do kernel c) transferência da memória registrador para a memória global.

A segunda etapa sendo executada no kernel PROPAGADESCIDA, possui dois laços de repetição para controlar a estabilização da propagação, tendo um laço controlado pelo host através da variável *vetorControle*, e outro dentro do kernel controlado pela variável *aux*. Pelo fato de existir tais controles para estabilização, foi adotada a estratégia do dimensionamento dos blocos conforme apresentado na seção anterior, isto é, os blocos que contém as bordas dos blocos adjacentes. Conforme a estratégia, a fase de leitura do kernel irá transferir tanto os dados da variável *L* contida na memória global quanto os dados da variável *I* contida na memória compartilhada para a memória compartilhada das respectivas variáveis *smLab* e *smImg* (vide Figura 4.28(a)). Como visto no fluxograma da Figura 4.25, que dentro do kernel PROPAGADESCIDA possui a iteração de estabilização e também a iteração dos vizinhos *(i,j)* do conjunto $N_{L8}(p,q)$, a Figura 4.28(b) apresenta somente os resultados das 8 iterações da vizinhança para cada iteração de estabilização dentro do kernel. Tais resultados são vistos pela variável *smLab* contendo os dados da propagação do endereçamento, a variável *aux* demonstrando quais dados dentro do bloco *smLab* foram alterados, bem como a variável *count* armazenando o numero de alterações + 1 de cada posição. A estabilização dentro do kernel é finalizada quando a variável *aux* estiver zerada. Deste modo, *smLab* irá transferir o resultado parcial da propagação realizada dentro dos blocos para a variável *L* na memória global, e também a variável *count* irá atribuir a variável *vetorControle* nas posições que foram alteradas(vide Figura 4.28(c)). A Figura 4.28 apresenta somente a estabilização dentro do kernel, visto que o procedimento fora do kernel irá se repetir até que a variável *vetorControle* esteja zerada.

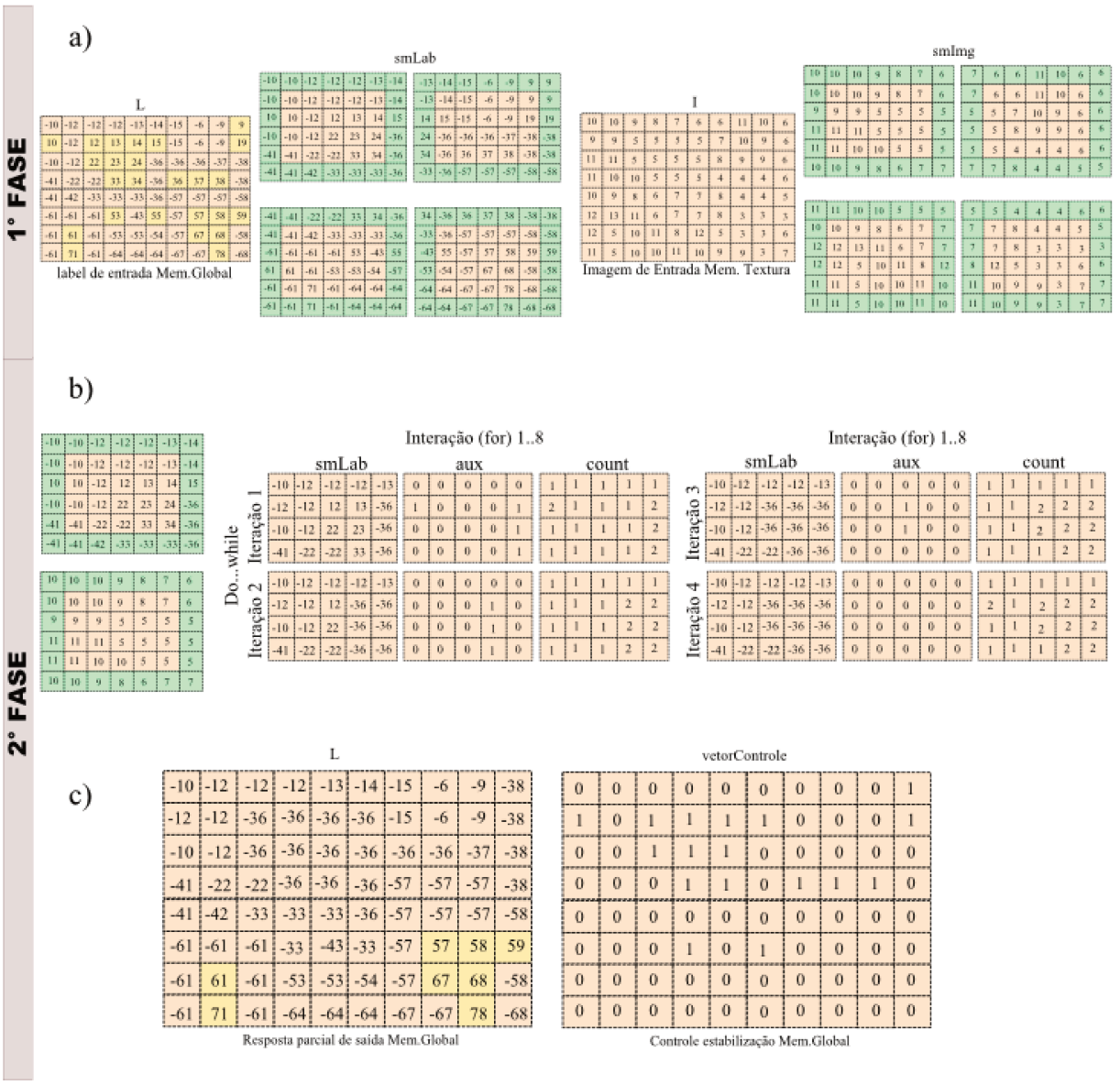


Figura 4.28: Processamento do kernel PROPAGADESCIDA a) Modelagem do grid e transferência dos dados b) Processamento do kernel c) transferência da memória registrador para a memória global.

De posse da matriz *L* contendo neste instante os apontamentos dos caminhos de todos os pixels para os mínimos, a terceira etapa unirá os pixels conexos mínimos caracterizados pelos valores maiores ou iguais a zero. O kernel AGRUPAPIXEL tem a finalidade de percorrer os vizinhos (i,j) do conjunto $N_{L4}(p,q)$ e estabelecer dentre os valores índices do bloco central e seu respectivo bloco vizinho(i,j) qual é o menor, armazenando este valor índice no bloco central. Com o resultado deste processo é então montado a lista de equivalência onde todos os valores de índices negativos são separados dos valores positivos associados, substituindo tais valores negativos pelos índices das próprias posições como visto na Figura 4.29(a). Todo cálculo da

modelagem do grid para este kernel segue o princípio do kernel PROPAGADESCIDA. O segundo passo desta terceira etapa é resolver a lista de equivalência, propagando assim o menor índice que representa o rótulo dos pixels agrupados. Para maiores detalhes do funcionamento deste kernel veja a Figura 4.30(b) da quarta etapa. Por se tratar de lista, o dimensionamento do grid para o kernel no exemplo, foi modelado em 1D possuindo cada bloco 8 threads, sendo assim *DimBlock* é 8 e *DimGrid ((w*h)/DimBlock)*(Vide Figura 4.29(b)). O kernel LABEL tem a finalidade de agrupar os dados negativos com o resultado parcial do agrupamento dos mínimos, como mostra a Figura 4.29(c). É importante notar que existe uma iteração de estabilização que envolve os 3 kernels, onde o kernel AGRUPAPIXEL é responsável por atribuir o valor 1 a variável *vetorControle* para finalizar esta estabilização conforme explicado anteriormente na segunda etapa do algoritmo.

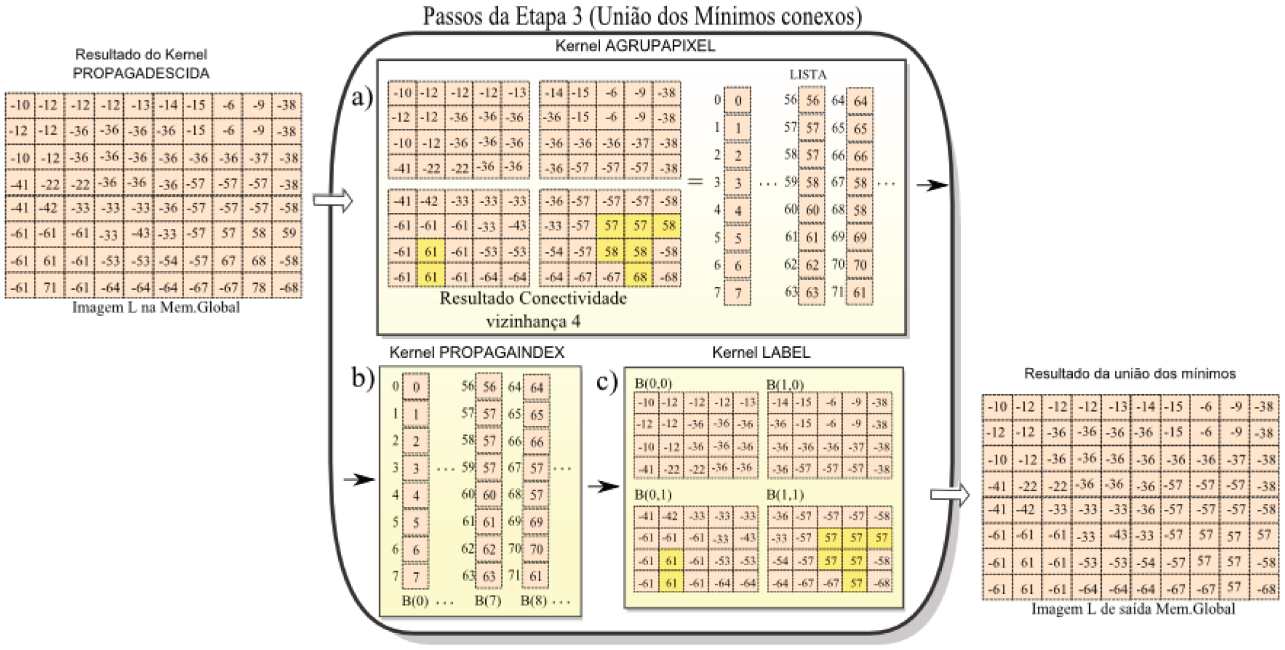


Figura 4.29: Passos da união dos mínimos conexos. (a) Determinação do vizinho com índice menor (b) Propagação das relações de vizinhança (c) União dos valores negativos e positivos que foram separados

A última etapa do algoritmo Watershed é propagar o rótulo contido nas posições dos pixels mínimos para os demais pixels. Neste ponto do algoritmo é utilizado o kernel AJUSTAVETOR para trocar o sinal negativo para positivo dos valores contidos na variável *L*, observando que o dimensionamento do grid para este kernel segue o modelo do Kernel PROPAGAINDEX, visto na Figura 4.30(a). Nota-se que a matriz *L* contém a associação de todos

os pixels, e como a finalidade do kernel PROPAGAINDEX é resolver as equivalências entre os apontadores dos pixels, propagando o índice base desta associação, é então aplicado novamente o kernel para processar a quarta etapa. A matriz *L* é transferida para memória de textura, para acelerar o processo de leitura dos dados dentro do laço de estabilização no kernel. O kernel possui duas variáveis principais chamadas de *ref* e *label*. A variável *label* serve para receber o índice da variável *L* nos índices apontados por *ref*, isto é, *label* = *L(ref)*. Sua estabilização se dá quando as duas variáveis *ref* e *label* forem iguais, indicando a não alteração das equivalências. Como resultado da estabilização desta iteração, surge à imagem Watershed, contendo a segmentação das regiões da imagem de entrada *I*. Para melhor visualização deste processo veja a Figura 4.30(b).

a)

Resultado da união dos mínimos

-10	-12	-12	-12	-13	-14	-15	-6	-9	-38
-12	-12	-36	-36	-36	-36	-15	-6	-9	-38
-10	-12	-36	-36	-36	-36	-36	-36	-37	-38
-41	-22	-22	-36	-36	-36	-57	-57	-57	-38
-41	-42	-33	-33	-33	-36	-57	-57	-57	-58
-61	-61	-61	-33	-43	-33	-57	57	57	57
-61	61	-61	-53	-53	-54	-57	57	57	-58
-61	61	-61	-64	-64	-64	-67	-67	57	-68

Matriz L de entrada Mem.Global

Kernel AJUSTAVETOR

B(0)	10	12	12	12	13	14	15	6	B(5)	41	42	33	33	33	36	57	57
B(1)	9	38	12	12	36	36	36	36	B(6)	57	58	61	61	61	33	43	33
B(2)	15	6	9	38	10	12	36	-36	B(7)	57	57	57	57	61	61	61	53
B(3)	36	36	36	36	37	38	41	22	B(8)	53	54	57	57	57	58	61	61
B(4)	22	36	36	36	57	57	57	38	B(9)	61	64	64	64	67	67	57	68

b)

Kernel PROPAGAINDEX

	iteração 1			iteração 2			iteração 3			iteração 4			iteração 5		
	ID	ref	label	ID	ref	label	ID	ref	label	ID	ref	label	ID	ref	label
B(0)	0	10	12	0	12	36	0	36	57	0	57	57			
	1	12	36	1	36	57	1	57	57	1	57	57			
	2	12	36	2	36	57	2	57	57	2	57	57			
	3	12	36	3	36	57	3	57	57	3	57	57			
	4	13	36	4	36	57	4	57	57	4	57	57			
	5	14	36	5	36	57	5	57	57	5	57	57			
	6	15	36	6	36	57	6	57	57	6	57	57			
	7	6	15	7	15	36	7	36	57	7	57	57			
	⋮			⋮			⋮			⋮			⋮		
B(5)	40	41	42	40	42	33	40	33	36	40	36	57	40	57	57
	41	42	33	41	33	36	41	36	57	41	57	57	41	57	57
	42	33	36	42	36	57	42	57	57	42	57	57	42	57	57
	43	33	36	43	36	57	43	57	57	43	57	57	43	57	57
	44	33	36	44	36	57	44	57	57	44	57	57	44	57	57
	45	36	57	45	57	57	45	57	57	45	57	57	45	57	57
	46	57	57	46	57	57	46	57	57	46	57	57	46	57	57
	47	57	57	47	57	57	47	57	57	47	57	57	47	57	57
	⋮			⋮			⋮			⋮			⋮		
B(7)	56	57	57	64	57	57	64	57	57	64	57	57			
	57	57	57	65	57	57	65	57	57	65	57	57			
	58	57	57	66	57	57	66	57	57	66	57	57			
	59	57	57	67	57	57	67	57	57	67	57	57			
	60	61	61	68	61	61	68	61	61	68	61	61			
	61	61	61	69	61	61	69	61	61	69	61	61			
	62	61	61	70	61	61	70	61	61	70	61	61			
	63	53	33	71	33	36	71	36	57	71	57	57			

Resultado watershed

57	57	57	57	57	57	57	57	57	57
57	57	57	57	57	57	57	57	57	57
57	57	57	57	57	57	57	57	57	57
57	57	57	57	57	57	57	57	57	57
57	57	57	57	57	57	57	57	57	57
61	61	61	57	57	57	57	57	57	57
61	61	61	57	57	57	57	57	57	57
61	61	61	57	57	57	57	57	57	57

matriz L de saída Mem.Global

Figura 4.30: Resultado do Watershed. (a) Ajustando os valores todos para positivo (b) Propagando o índice representativo pelos caminhos gerados.

Capítulo 5

ESTRATÉGIA DA VISÃO COMPUTACIONAL

Neste capítulo é apresentada a estruturação da camada de visão computacional, abordando a aplicação dos métodos de processamento de imagens visto no capítulo 3, contextualizando as técnicas de processamento paralelo em GPU vistos no capítulo 4, bem como a execução do fluxo de dados que ocorre entre o CPU e GPU dentro desta camada. Nesta concepção, a Figura 5.1 apresenta o fluxograma contextualizado com o desenvolvimento da camada de visão computacional. Nota-se que o *PROCESSO i* será abordado na seção 16.4.

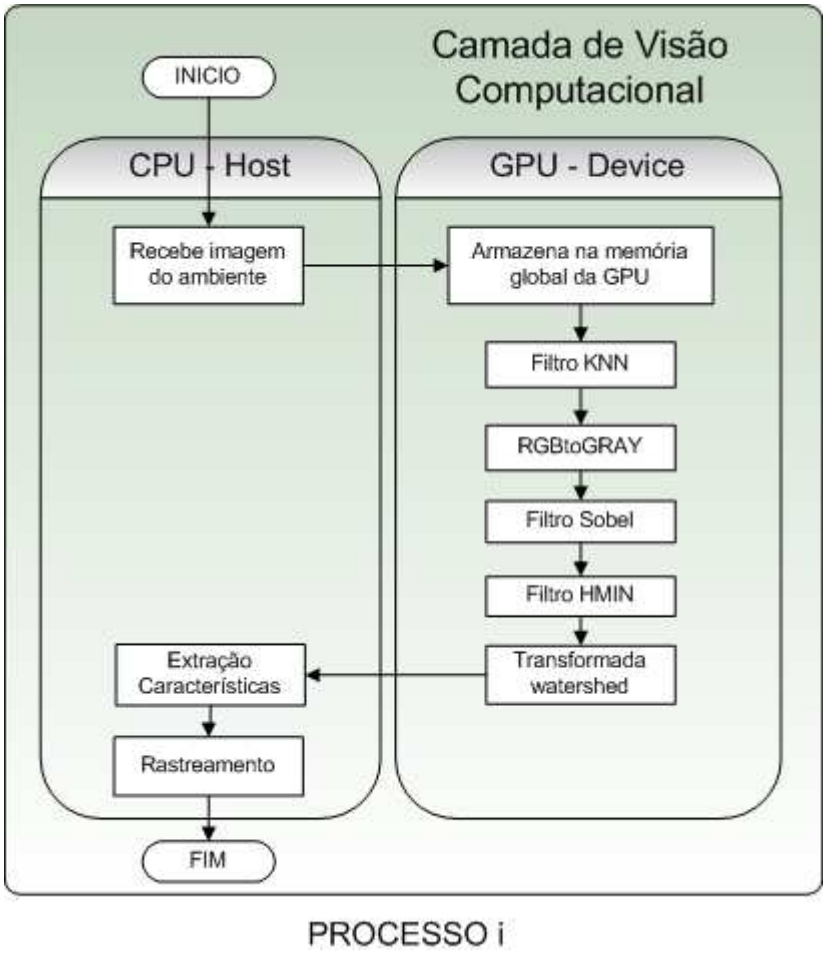


Figura 5.1: Fluxograma da Camada de visão Computacional para o processo i.

A camada de Visão Computacional deste projeto possui dois principais objetivos dentro do sistema, que é alimentar a camada de Estratégia de Controle e Navegação que será apresentada na seção 16.1 e realizar o rastreamento do alvo apresentada na seção 15.4. O primeiro objetivo é extrair e transmitir as informações do alvo entre a camada de visão e a camada de estratégia de controle e navegação, como o ponto representativo e a área, para proporcionar meios de elaborar a movimentação do robô. Nota-se que o ponto representativo denominado de *PR* e a área são obtidos através do rastreamento que será apresentado na seção 15.3. O segundo objetivo é garantir o foco no alvo entre as imagens do ambiente adquiridas pelo robô ao longo do tempo. Para isso a camada de visão computacional deve assegurar que o alvo seguido na imagem seguinte seja o mesmo alvo da imagem anterior, estabelecendo assim a relação do alvo entre as imagens, consolidando o processo de rastreamento. Como visto anteriormente no capítulo 2, esta não é uma tarefa trivial.

Diversos métodos na área de visão computacional podem ser aplicados para solucionar os dois objetivos acima expostos. No caso deste trabalho, a metodologia selecionada para o processo de visão computacional será apresentada a seguir, obtido através da união das seguintes atividades: pré-processamento, segmentação, extração de características e correspondência entre imagens.

5.1 Pré-Processamento de Imagem

Como dito anteriormente, a etapa de pré-processamento serve para o melhoramento da imagem. Este melhoramento deve ser modelado de tal forma a proporcionar uma boa segmentação. Deste modo, para se alcançar uma desejada segmentação em relação ao tipo de imagem adquirida do ambiente como, por exemplo, de uma estrada ou de uma sala onde o nível de detalhamento do ambiente entre estas duas imagens são diferentes, esta etapa de pré-processamento irá influenciar no resultado da segmentação. O motivo desta influência será exposto mais adiante.

Inicialmente, de posse da imagem colorida adquirida do ambiente no instante de tempo *t*, denominada de imagem de entrada I_t, é transferida para memória global de uma das GPU´s, relembrando que a placa GTX295 possui 2 GPU´s. O gerenciamento de distribuição das imagens para os GPU´s será abordado no capítulo 6. Com a imagem na memória global da GPU é então aplicado o filtro de suavização KNN conforme o procedimento descrito na seção 4.3. O resultado pode ser conferido na Figura 5.2.

(a) (b)

Figura 5.2: Resultado filtro KNN paralelo (a) Imagem de entrada (b) Imagem resultante do filtro KNN

Em um passo seguinte, na equação 5.5.1 é feito a conversão da imagem RGB para escala de cinza, pois todo o processamento até a segmentação é realizado na imagem em escala de cinza. Para isso foi feito um kernel chamado RGBtoGRAY para processar a equação 5.1.1. O fluxograma do código deste kernel é visto na Figura 5.3, observando que o dimensionamento do grid foi modelado em blocos de (16x16) threads.

$$(p,q) = \frac{R(p,q) + G(p,q) + B(p,q)}{3} \tag{5.1.1}$$

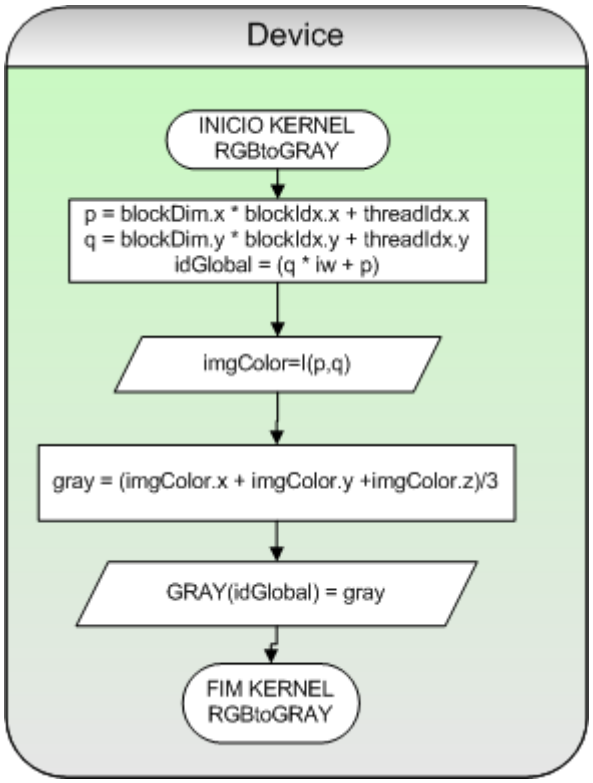


Figura 5.3: Fluxograma do Kernel de conversão RGB para Escala de Cinza.

O resultado da aplicação do kernel RGBtoGRAY é visto na Figura 5.4.

(a) (b)

Figura 5.4: Conversão RGB para Escala de Cinza. (a) Imagem RGB (b) Imagem Escala de Cinza

Após tal conversão, a imagem em escala de cinza é submetida ao filtro Sobel para detecção das bordas, realizando o procedimento descrito na seção 4.4. O resultado desta etapa pode ser visto na Figura 5.5.

(a) (b)

Figura 5.5: Resultado do Filtro Sobel Paralelo (a) imagem de entrada (b) Imagem resultante do filtro Sobel

O último passo da etapa de pré-processamento é aplicar o filtro reconstrutivo HMIN, conforme apresentado na seção 4.5. O resultado é demonstrado pela Figura 5.6.

(a) • (b)

Figura 5.6: Resultado do Filtro reconstrutivo HMIN paralelo (a) Imagem do filtro Sobel (b) Imagem resultante do filtro Hmin

Como é difícil visualizar o resultado deste filtro, a Figura 5.7, demonstra o gráfico do sinal gerado pelo filtro extraído da linha 62 da imagem, para diferentes valores de H (parâmetro do filtro).

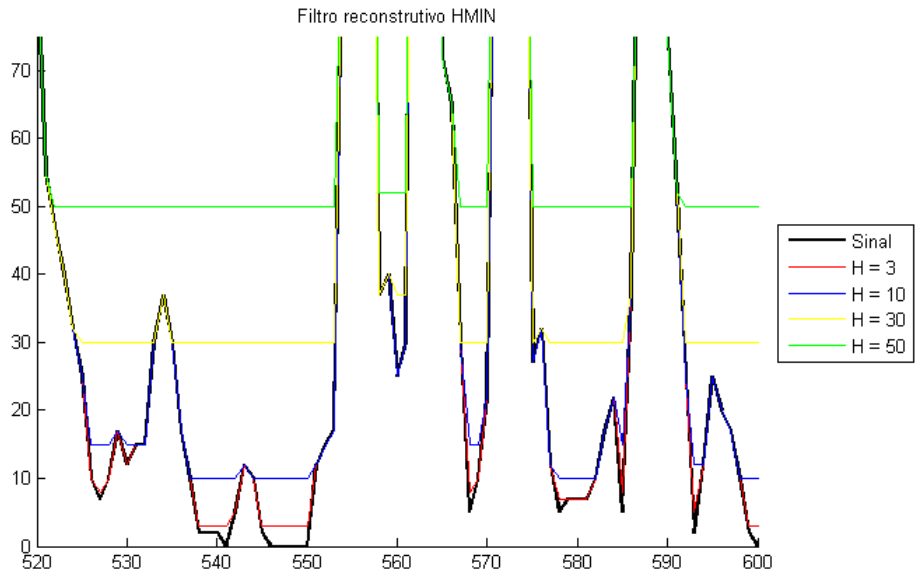


Figura 5.7: Resultado da aplicação do filtro reconstrutivo HMIN para diferentes valores do atributo altura

É importante ressaltar que o parâmetro H deste filtro altera de forma direta o resultado da segmentação.

5.2 Segmentação de Imagem

Segundo GONZALEZ e WOODS (2000) a segmentação de imagens pode ser considerada como a partição de imagens digitais em conjuntos de pixels considerando-se o objetivo do trabalho.

Dentre os métodos de segmentação existentes, optou-se por trabalhar com a transformada Watershed, pelo fato desta ser bastante robusta, quando aplicada de maneira correta.

Considerando a lógica apresentada na seção 4.6 para realizar a transformada Watershed, seu resultado é apresentado na Figura 5.10 contendo a média dos valores de intensidade RGB nas regiões particionadas.

(a) (b)

Figura 5.8: Resultado Watershed contendo a média dos valores de intensidade RGB (a) Imagem de entrada (b) Imagem segmentada pela transformada Watershed

Mencionado anteriormente na seção 5.1, a influência do pré-processamento na determinação da segmentação, mais específico na relação existente entre o filtro reconstrutivo HMIN e a transformada Watershed, é determinado pelos pixels mínimos. Como a transformada Watershed é construída sobre os pixels mínimos, qualquer alteração realizada nos valores destes pixels influenciará o resultado da segmentação. É justamente para este fim que o filtro reconstrutivo está sendo aplicado, para controlar o número de regiões segmentadas em função da imagem gradiente (resultado do filtro Sobel). Para efeito de demonstração, temos na Figura 5.9(a), um sinal de entrada 1D o qual é aplicado à transformada Watershed. Observando os mínimos, o resultado da transformada para este sinal pode ser visto na Figura 5.9(b), tendo gerado quatro regiões. Se antes de aplicar a transformada Watershed, for processado um filtro reconstrutivo para eliminar alguns mínimos locais, como apresentado na Figura 5.9(c), observe que o resultado da transformada foi modificado possuindo agora 3 regiões. Esta operação é útil para controlar o nível de detalhamento da sua segmentação.

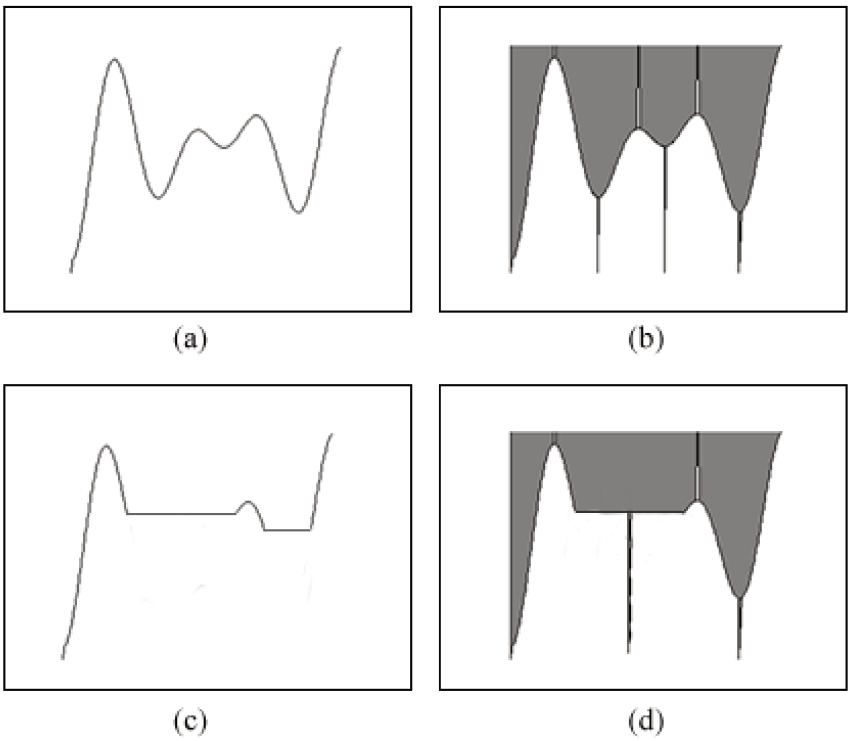


Figura 5.9 : Influência da alteração do pixel mínimo na transformada Watershed.

Embasado nesta técnica, o controle do detalhamento desejado para a segmentação é determinado pelo parâmetro H passado ao filtro reconstrutivo HMIN. Para exemplificar tais níveis de detalhamento de uma imagem, a Figura 5.10 demonstra o resultado da transformada Watershed influenciado pela escolha do parâmetro H.

(a) (b)

(c) (d)

Figura 5.10 : Resultado da influência do Filtro HMIN na constituição da segmentação com diferentes valores de H. (a) H=10 (b) H=30 (c) H=50 (d) H=100

5.3 Extração de Característica

A extração de características determina um passo importante para o processo da visão computacional, pois as informações de identificação de um objeto são obtidas neste passo que servirá de base para alimentar o processo de rastreamento.

Conforme visto todo o procedimento realizado no pré-processamento e segmentação, neste ponto a imagem encontra-se totalmente segmentada na GPU. A imagem RGB suavizada e a

imagem segmentada são transferidas de volta para o CPU onde o processo de extração de características será realizado.

De posse das duas imagens, vários atributos podem ser extraídos como, por exemplo, os atributos de similaridade dados pelas intensidades médias dos valores RGB em cada região, o coeficiente de correlação entre a região alvo da imagem I_t e as demais regiões da imagem I_{t+1}, e também os atributos de compatibilidade dados pela relação de distância entre a região e as regiões vizinhas, bem como a orientação angular associada a cada região vizinha. Segundo GALO (2003), a utilização de várias métricas para representar as características de primitivas, constitui um robusto procedimento para determinação da correspondência destas primitivas entre as imagens.

A princípio, este projeto optou-se por trabalhar com duas métricas para representar as características das regiões, sendo que estas não restringem uma possível incorporação de outras métricas ao modelo em um trabalho futuro. As duas métricas para as regiões são dadas pelas médias dos valores de intensidade RGB, e a posição relativa desta região na imagem, conhecida como centróide. É importante relembrar que a camada de visão computacional possui o objetivo de realizar o rastreamento do alvo e também alimentar a camada de estratégia de controle e navegação. Neste sentido, as duas informações acima definidas são para atender o primeiro objetivo. O segundo objetivo é atendido com a informação da área das regiões alvo e seu ponto representativo *(PR)*.

A determinação dos cálculos para obter as informações de área, centróide e média da região, foi apresentada na seção 3.5, utilizando a equação 3.5.1 para obter a área do alvo, a equação 3.5.2 para encontrar o centróide e a equação 3.5.3 para obter as médias dos valores de intensidade RGB das regiões. A Figura 5.11 expõe o resultado na imagem referente a estes cálculos, notando que as cores das regiões apresentadas são formadas pelas médias RGB, e os pontos em preto referenciando o centróide de cada região.

(a) (b)

Figura 5.11 : Centróide e média RGB das regiões (a) Imagem de entrada (b) Imagem contendo a média e centróide de cada região

5.4 Correspondência do alvo entre Imagens

Este passo dentro da camada de visão configura o procedimento de rastreamento por correspondência de métricas similares entre a região alvo da imagem I_{t-1} e as regiões da imagem I_t.

Depois de extraídos os atributos de todas as regiões segmentadas da imagem, uma lista é feita, onde cada linha desta lista representa as informações de uma região da imagem I_t denominada de região candidata. De posse das informações da região alvo provenientes do processamento anterior da camada de visão, é então iniciado um processo iterativo para determinar as regiões de I_t que mais se assemelham a região alvo obtida de I_{t-1}. É importante frisar que esta transição do domínio da Imagem para o domínio das regiões é caracterizada por descrição relacional. Segundo GALO (2003), esta representação do domínio imagem para o domínio das primitivas, em termos gerais, podem ser denominadas de descrição relacional. A Figura 5.12 demonstra esta representação.

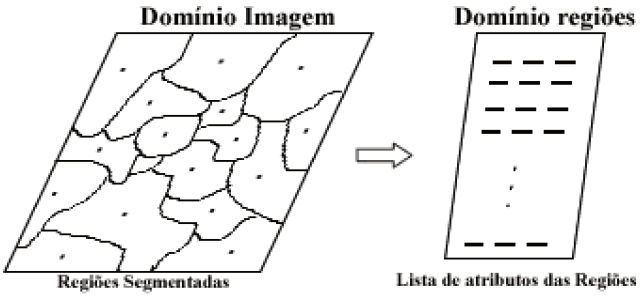


Figura 5.12 : Transição de Domínios

O procedimento de correspondência proposto possui dois parâmetros chamados de *LimiarDistancia* e *LimiarSimilaridadeCor*. O *LimiarDistancia* estabelece qual será o raio de distância máximo onde provavelmente a região alvo possa ter se deslocado, levando em consideração um dado deslocamento do alvo no instante de tempo entre I_{t-1} e I_t. Este parâmetro foi incorporado ao modelo em função da relação existente entre a taxa de captura das imagens (aproximadamente 10fps) e o deslocamento do alvo, que no caso é uma região pertencente a um alvo humano. O segundo parâmetro *LimiarSimilaridadeCor* possui a finalidade de garantir a similaridade do atributo cor entre duas regiões, isto é, dado o valor do ângulo resultante da aplicação do produto escalar, este é comparado com o *LimiarSimilaridadeCor* para permitir que somente as regiões com atributos de cor semelhantes sejam selecionados. O produto escalar entre duas cores RGB é adquirido através da equação 5.4.1, sendo que, quanto mais próximo de 0 for o resultado do ângulo obtido, mais similares serão as cores analisadas. O θ representa o ângulo, VC_A e VC_C representam respectivamente o vetor cor do alvo e o vetor cor do candidato, tendo tais vetores formados pelas médias RGB do atributo cor.

$$\theta = arccos\left(\frac{VC_A.VC_C}{\|VC_A\|.\|VC_C\|}\right) \tag{5.4.1}$$

Onde

$$VC_A.VC_C = R_A R_C + G_A G_C + B_A B_C$$
$$\|VC_x\| = \sqrt{{R_x}^2 + {G_x}^2 + {B_x}^2}$$

Visto os dois parâmetros que influencia o procedimento de correspondência, a Figura 5.13 apresenta o fluxograma do algoritmo proposto. O rastreamento da região alvo é iniciado fornecendo ao sistema qual a região que se pretende seguir na imagem. Na imagem seguinte é

obtida a lista de regiões candidatas. Para cada iteração, o algoritmo irá verificar se o candidato atende as condições de correspondência e caso o candidato corrente seja similar, o algoritmo irá ordenar tal candidato com relação a seu ângulo de similaridade. No final da interação, quando todos os candidatos forem avaliados, as melhores 10 regiões candidatas, se houver, farão parte da constituição das informações na atualização dos dados da região alvo. A atualização dos atributos do alvo é dada pela média aritmética dos atributos das regiões selecionadas.

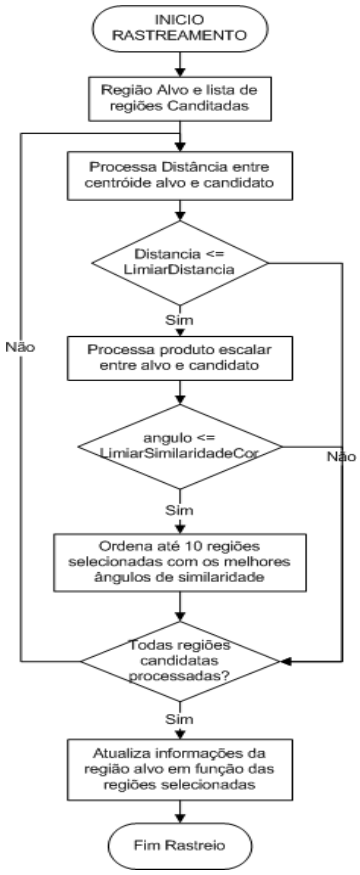


Figura 5.13 : Fluxograma do Algoritmo de Rastreamento

Para ilustrar o processo de rastreamento seguindo o algoritmo proposto, a Figura 5.14 apresenta as imagens segmentadas ao longo do tempo *t*. O rastreamento da região alvo inicia quando o sistema recebe a informação de qual região seguir. No primeiro momento, a informação contida na região selecionada irá automaticamente compor as informações da região alvo. Nos frames seguintes, o algoritmo desenvolvido determina as regiões correspondentes observando que o ponto em amarelo foi caracterizado pelo Ponto Representativo (*PR*) que representa o centróide

dos centróides. No caso do frame *t*, o *PR* representa o próprio centróide com o atributo cor RGB da região selecionada, entretanto nos demais frames, este *PR* representa a média dos centróides das regiões selecionadas, como mencionado anteriormente. Obtido o *PR*, ressalta-se que não necessariamente ele precise coincidir com algum dos centróides das regiões determinadas, tão pouco estar contida na área das regiões.

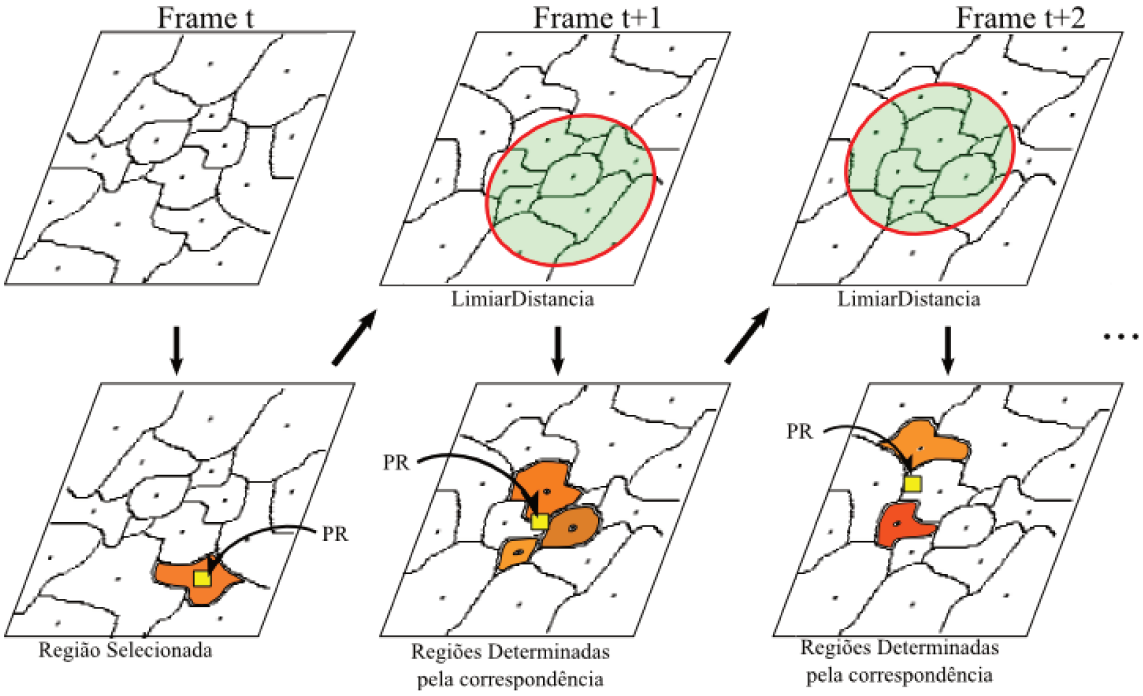


Figura 5.14 : Processo iterativo de Rastreamento

É importante apresentar que nos experimentos realizados, ocorre uma variação freqüente da posição do *PR* e também do valor da área denominada de *A*. Apesar do alvo e o robô estarem em repouso, indicando necessariamente que o valor de *PR* e de *A* teriam também que estarem constante, isto não ocorre devido à oscilação da luminosidade que afeta o sistema de visão, refletindo nestas duas informações que serão passadas para a camada de Estratégia de Controle e Navegação.

Capítulo 6

INTEGRAÇÃO DO SISTEMA DESENVOLVIDO

Neste capítulo, são descritos a camada de Estratégia de Controle e Navegação objetivando apresentar a geração de movimentos para o robô a partir das informações vindas da camada de Visão Computacional, a plataforma do sistema abordando a arquitetura modelada, a camada de Comandos e Transmissão apresentando a codificação dos movimentos para comandos interpretados pelo robô, e finalizando com o software desenvolvido apresentando as ferramentas utilizadas, a distribuição e controle dos processos que envolvem as duas GPU´s.

6.1 Estratégia de Controle e Navegação

Visto anteriormente, esta camada tem a finalidade de decidir para onde o robô deve seguir e gerar sua movimentação. Como o objetivo final do sistema em termos gerais é acompanhar um alvo em movimento, a estratégia de geração do movimento para o robô deve ser tal que este consiga cumprir sua tarefa atingindo o objetivo proposto. Deve ressaltar que o sistema não possui mecanismos de retro-alimentação para validar e garantir a execução correta dos movimentos enviados ao robô. Desta forma a camada de Controle e Navegação estrutura as estratégias de movimentos em função de novas informações fornecidas pela camada de visão computacional, alimentando o robô com novos movimentos.

A estratégia de controle para geração dos movimentos foi modelada com lógica clássica. Esta escolha foi determinada pelo fato do robô apresentar restrições que impedem o envio e recebimento simultâneo de informações. Nos experimentos realizados, observou-se que no robô, existe um tempo de atraso para executar um dado comando com intensidades de velocidade variada ao mesmo tempo em que este transmite os dados da imagem capturada, isto é, ao enviar o comando, o robô irá primeiro executar esta ação para depois realizar o envio da imagem. Em

outras palavras seria como dar passos com os olhos fechados e depois ver quais as mudanças que ocorreram no ambiente durante o deslocamento. Para solucionar esta restrição, optou-se por fixar uma intensidade de velocidade constante para realizar o movimento do robô. Esta escolha afetou o sistema de visão, discutida na seção 17.2.

O princípio básico para modelar a estratégia de controle para este projeto foi fundamentado no deslocamento de profundidade e deslocamento lateral. A profundidade referencia as movimentações de "para frente" e "para trás" caracterizando a relação de distância entre o robô e o alvo, e o deslocamento lateral referenciando as movimentações de "para esquerda" e "para direita". Para elaborar tais comandos e constituir uma base de regras, foram necessários alguns cálculos com as informações recebidas da camada de Visão Computacional, que serão apresentadas neste momento.

De posse das variáveis de entrada da camada de Controle e Navegação que são a área do alvo *(A)* e do ponto representativo *(PR)*, o primeiro passo rumo à modelagem da estratégia é definir o ponto central. Este ponto central servirá de base para verificar qual a variação nos eixos (X, Y) do alvo referente ao centro da imagem. Este ponto central denominado de PC é obtido pela equação 6.1.1, sendo *iw* e *ih* as dimensões horizontais e verticais da imagem.

$$PC = \left(\frac{iw}{2}, \frac{ih}{2}\right) \tag{6.1.1}$$

Definido o ponto central *PC*, a variação de PR referente à PC foi denominada de VP que é obtido por:

$$VP = PR - PC = (PR.x - PC.x\,,\ PR.y - PC.y) \tag{6.1.2}$$

Com a obtenção dos valores de VP em relação aos dois eixos, a base de regras é então criada para gerar o movimento do robô. Com relação ao deslocamento horizontal, é possível determinar os movimentos de "Para Esquerda" e "Para Direita" seguindo as informações de VP.x. Se o valor de *VP.x* ser negativo, o comando será "Para Esquerda", caso seja positivo, então "Para Direita", conforme Figura 6.1.

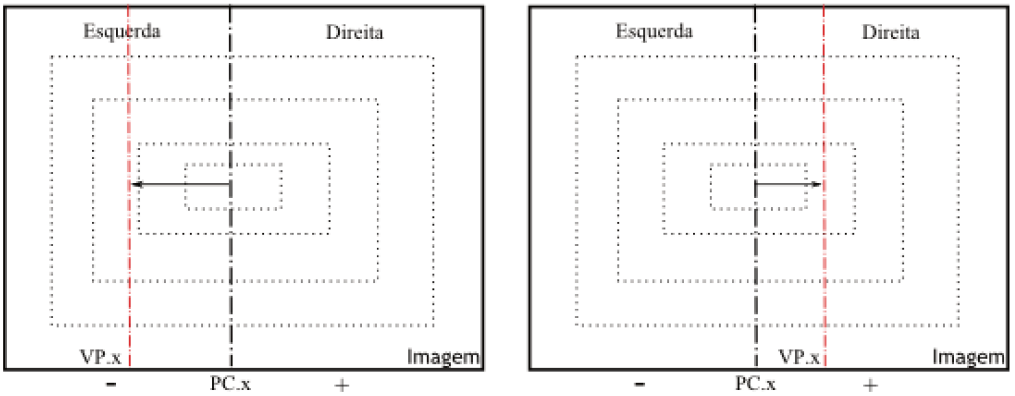


Figura 6.1: Comandos para deslocamentos horizontais

No entanto, como o valor de *PR* possui uma variação ocasionada pela variação da luminosidade ao longo do tempo, abordado na seção 5.4, isto pode gerar movimentação indesejada ao robô. Caso o valor *VP.x* varie próximo ao valor *PC.x*, este pode gerar movimentos contraditórios como "Para Esquerda", "Para Direita", "Para Esquerda" ao longo das imagens adquiridas. Para contornar esta situação, criaram-se áreas de pertinência na dimensão horizontal da imagem com a finalidade de evitar estas movimentações. Três áreas foram criadas com os respectivos nomes de "Esquerda", "Parado" e "Direita", onde o movimento do robô é dado em relação a estas áreas como mostra a Figura 6.2.

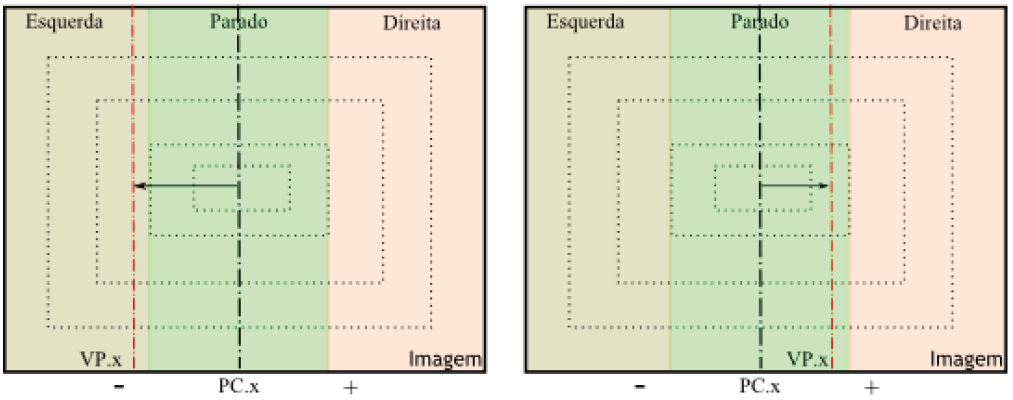


Figura 6.2: Áreas de pertinência para a variável VP.x

As regras de movimentações referentes às orientações no eixo horizontal foram desenvolvidas conforme explicação anterior. Neste momento, será exposta qual a estratégia para movimentações de profundidade do ambiente, observando que, como este projeto trabalha com

mono-visão, a informação de profundidade da cena é perdida na constituição da imagem. Para perceber quando o alvo está se deslocando no eixo da profundidade, dois aspectos podem ser extraídos como informação, a variação de *PR* no eixo *Y* dado por *VP.y*, e a área do alvo *A*.

A obtenção do deslocamento do alvo por parte da variação dado por *VP.y* é caracterizada pelo eixo óptico *EO* da câmera não ser paralelo com o eixo *Z* do ambiente. Deste modo, o deslocamento do alvo no eixo *Z* do ambiente irá refletir no eixo *Y* da câmera conforme Figura 6.3.

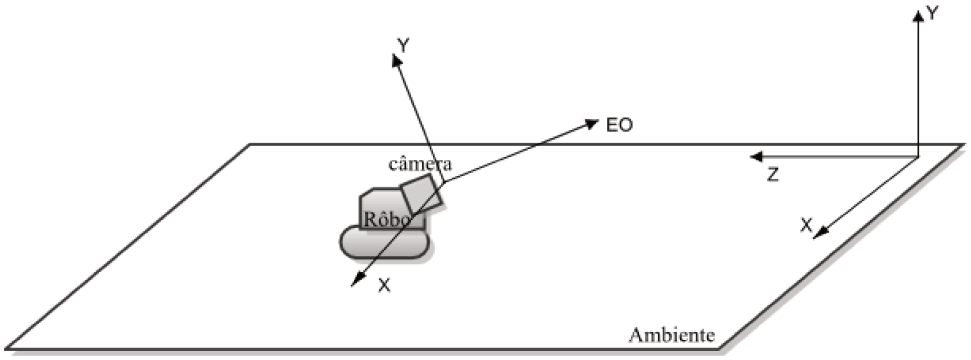


Figura 6.3: referencial de câmera, eixo óptico não é paralelo ao eixo Z do ambiente

Portanto, a primeira informação da profundidade para a regra de movimentação é obtida través desta relação dos eixos apresentada. Para isso, criaram-se na imagem as áreas de pertinência para a variável *VP.y* denominadas de "Perto", "Normal" e "Longe", conforme a Figura 6.4. Para cada área de pertinência foi gerado o respectivo movimento de "Para Trás", "Permanecer Parado" e "Para Frente".

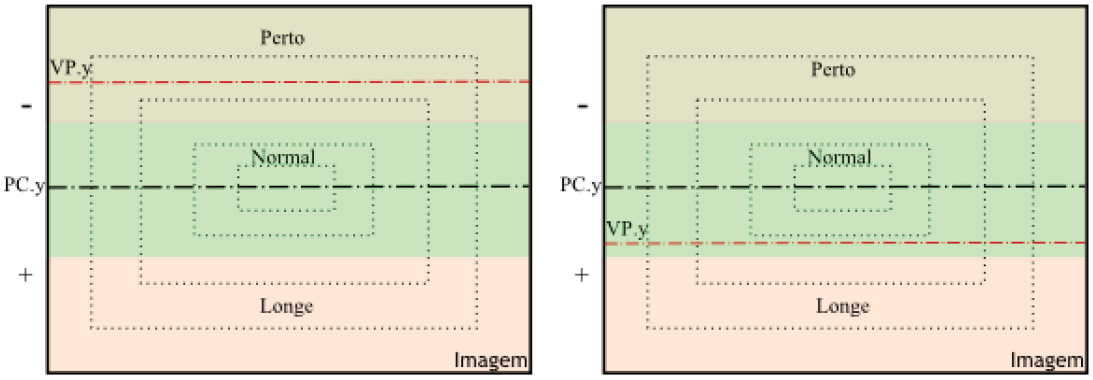


Figura 6.4: Área de pertinência para a variável VP.y

Observando que toda a base de regras até o momento foi criada sobre o mesmo domínio de imagem, desta maneira, existe uma relação entre estas regras (vide Figura 6.5(a)) que irá proporcionar mais orientação de movimentação como “Para Trás/Esquerda”, “Para Trás/Direita”, “Para Frente/Esquerda” e “Para Frente/Direita”, conforme Figura 6.5(b).

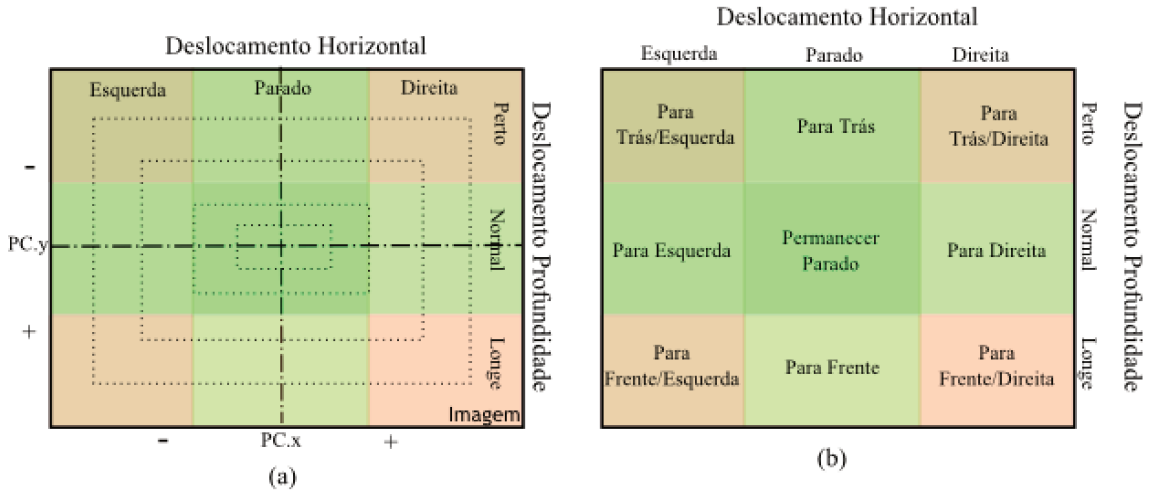
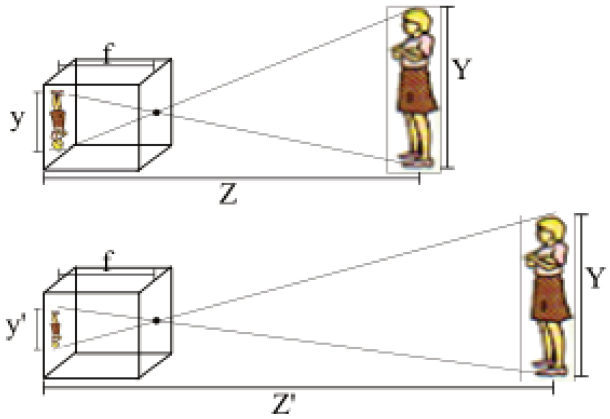


Figura 6.5: Base de regras para movimentação. (a) Relação das áreas de pertinência. (b) Tabela de movimentos associadas às áreas.

O segundo aspecto para extrair a informação auxiliar da profundidade, é obtida por parte da área *A*. Esta é caracterizada pela projeção perspectiva na formação da imagem, abordados em (GONZALEZ e WOODS, 2002) e (BALLARD e BROWN, 1982). De forma geral, a relação geométrica existente pode ser apresentada pela conhecida câmera de pinhole dando origem ao modelo perspectiva, conforme Figura 6.6.



$$-\frac{y}{f} = \frac{Y}{Z - f}$$

(a) (b)

Figura 6.6: Extração da informação de Profundidade (a) Projeção perspectiva para diferentes distâncias em Z (b) Modelo perspectiva

Neste sentido podemos concluir que, dado uma área central denominada de *AC*, caracterizando $y = AC$ do alvo na imagem referente a uma distância *Z*, a variação da área *A* denotado por *VA* em relação à área central *AC*, nos instantes de tempo *t*, indicará o deslocamento no eixo *Z*. Como apresentado na seção 5.4, a variável *A* também oscila com a iluminação, gerando diferentes valores ao longo do tempo referenciado por A_t. Para amenizar este problema, duas estratégias foram desenvolvidas, sendo a primeira na modelagem do valor da área central *AC*, e a segunda baseada na questão das áreas de pertinência vistas anteriormente.

A primeira estratégia para determinação do valor *AC* foi modelada com o intuito de encontrar um valor médio ao longo do tempo, que representasse o valor fixo da área quando o sistema estivesse em repouso. Para que esta estratégia seja disparada, é necessário antes colocar o sistema em repouso através da informação do PR, isto é, para a região alvo selecionada em qualquer lugar da imagem, será utilizada a informação do PR para centralizar este alvo, e então iniciar a determinação do *AC*. O alvo é dito como centralizado caso o valor de *PR* esteja na área de pertinência do movimento "Permanecer Parado". Esta ação implica que provavelmente o robô e o alvo estejam parados, caracterizando que qualquer variação da área A_t ao longo deste tempo "Permanecer Parado" represente a mesma área *A*. Nesta condição é disparada a modelagem de obtenção de *AC* dado pela equação 6.1.3, onde *td* é definido como tempo de disparo $td = t$, *n* como sendo uma constante dada por 100 e *t* é o número de aquisição da imagem ao longo do tempo.

$$AC = \frac{1}{n}\sum_{t=td}^{td+n} A_t \tag{6.1.3}$$

Definido a área central AC, a variação de A_t referente à AC dada por VA é obtido por:

$$VA = \left(\frac{AC - A_t}{AC}\right) \tag{6.1.4}$$

Encontrado o valor da variação da área VA, a segunda estratégia é modelar a área de pertinência para esta variável. Como este valor também caracteriza o deslocamento no eixo Z, a nomenclatura das áreas de pertinência foram mantidas em "Perto", "Normal" e "Longe", bem como seus respectivos movimentos "Para Trás", "Permanecer Parado" e "Para Frente", visto na Figura 6.7.

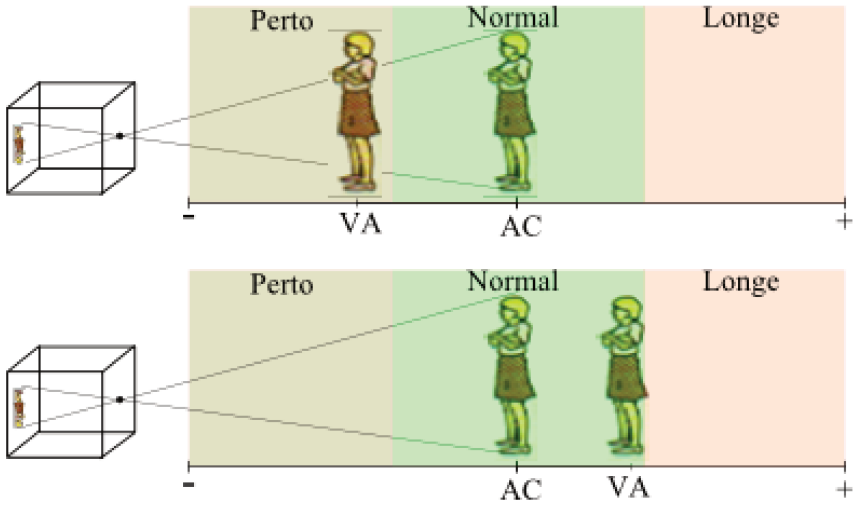


Figura 6.7: Área de pertinência para a variável VA, abordado sobre o eixo Z

Neste ponto, as duas informações de profundidade representadas pelas variáveis *(VA e VP.y)* teoricamente teriam que resultar em uma mesma informação do ambiente, porém na prática isto pode não ocorrer, devido aos ruídos gerados no sistema como um todo (fatores internos e externos). Um exemplo deste fenômeno contraditório seria o valor de *VP.y* estivesse na área de

pertinência “Perto” da imagem e a variável *VA* com valor positivo caindo na área de pertinência “Longe”. Deste modo a camada de Controle e Navegação entraria em conflito. Para contornar esta questão, foi implementado um processo de ponderação que enfatiza a informação advinda da variável *VP.y*, pelo fato desta sofrer mesma influência de fatores externos.

6.2 Plataforma de Desenvolvimento

Nesta seção será apresentada a arquitetura funcional do sistema, compreendendo a estruturação das camadas físicas concebidas por Servidor, Comunicação e Eletrônica Embarcada. O layout da Figura 6.8 demonstra a relação destas três camadas.

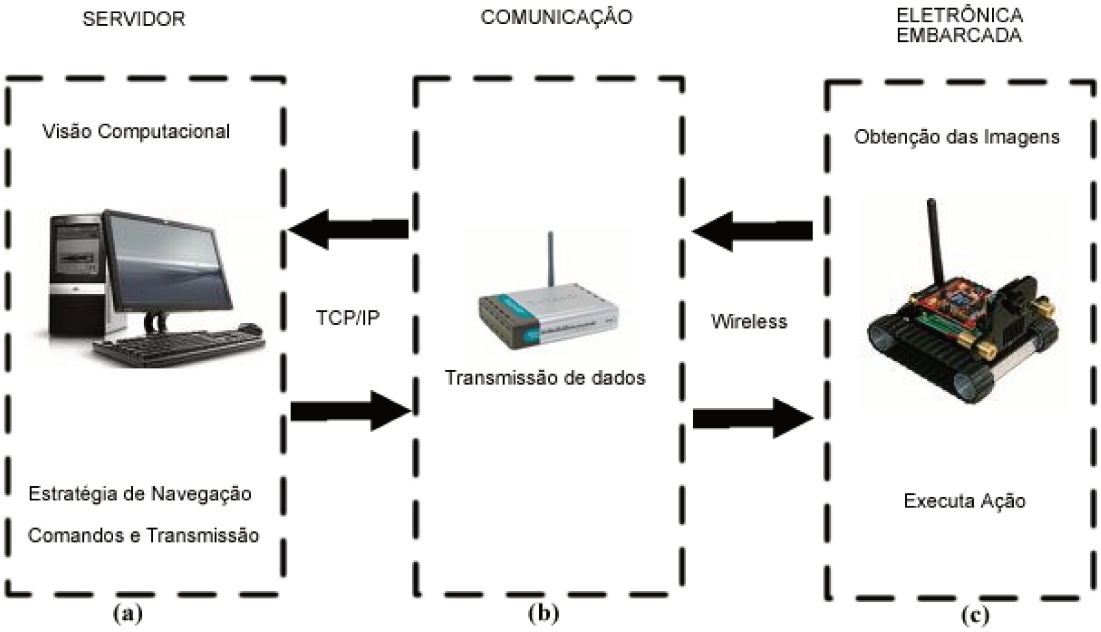


Figura 6.8: Arquitetura Funcional do Sistema

6.2.1 Servidor

Considerando as características de grande parte dos sistemas robóticos e também devido a exigências de recursos de processamento para os algoritmos de visão computacional, a arquitetura Cliente-Servidor foi selecionada. Conseqüentemente gera a necessidade de um sistema de comunicação entre o Servidor e o Robô.

Neste projeto, o servidor (vide Figura 6.8(a)) é caracterizado por um microcomputador multiprocessado AMD Phenom II X3 (2.6Ghz 7.5Mb Cache) com 4 GB RAM e uma GeForce GTX295 com 1792Mb, que ficou responsável pelo software de rastreamento e navegação. Este é baseado no conceito de sistema distribuído, sendo que todas as camadas independentes apresentam se como um único e consistente sistema.

6.2.2 Comunicação

O sistema de comunicação se faz necessário para permitir a comunicação do servidor com o robô visto na Figura 6.8(b). Como a plataforma embarcada dá suporte à utilização de uma rede wireless baseada no protocolo TCP/IP, a camada foi constituída tendo um roteador wireless como ponte de comunicação entre as 2 partes.

Neste sentido se faz necessário a configuração do robô para acessar a rede local. Tal configuração seguiu a seguinte especificação para acesso:

SSID: ROBOLAB
IP: 192.168.10.10
Mask: 255.255.255.0
GT: 192.168.10.1
DNS: 143.106.9.2
Channel: 1

Desta forma, a comunicação de recebimento das imagens e envio dos comandos de execução serão endereçados com o IP apresentado.

6.2.3 Eletrônica embarcada

Conforme apresentado na Figura 6.8 e também na seção 2.4, o robô utilizado para os experimentos deste projeto chama SRV-1 cujas características principais emprega o SRV-1 Blackfin Camera Board com processador 1000MIPS 500MHz analógico BF537, uma câmera digital com resolução de 160x126 até 1280x1024 pixels, ponteiro lazer e WLAN 802.11b/g rede sobre uma base móvel.

6.3 Comandos e Transmissão

Com base na abordagem da arquitetura funcional apresentada anteriormente na Figura 6.8, a camada de comandos e transmissão está representada no caminho lógico entre o Servidor (Figura 6.8(a)) e a eletrônica embarcada (Figura 6.8(c)).

O objetivo da camada de Comandos e Transmissão é codificar a informação de movimento recebido da camada de Estratégia de Controle e Navegação e transmitir o comando para o robô executar a dada movimentação

A definição do protocolo de comandos a serem enviados para o robô é apresentada na Tabela 6.1.

Tabela 6.1: Relação dos movimentos com os comandos do robô

Movimento do Robô	Comando enviado para o Robô
Para Trás/Esquerda	4
Para Trás	2
Para Trás/Direita	6
Esquerda	4
Parado	0
Direita	6
Para Frente/Esquerda	3
Para Frente	8
Para Frente/Direita	1

6.4 Software Desenvolvido

O software desenvolvido é responsável por integrar todas as camadas apresentadas anteriormente e responsável também, pelo gerenciamento das informações distribuídas entre os processos. A princípio é demonstrada a estruturação a qual o software foi modelado, observado na Figura 6.9.

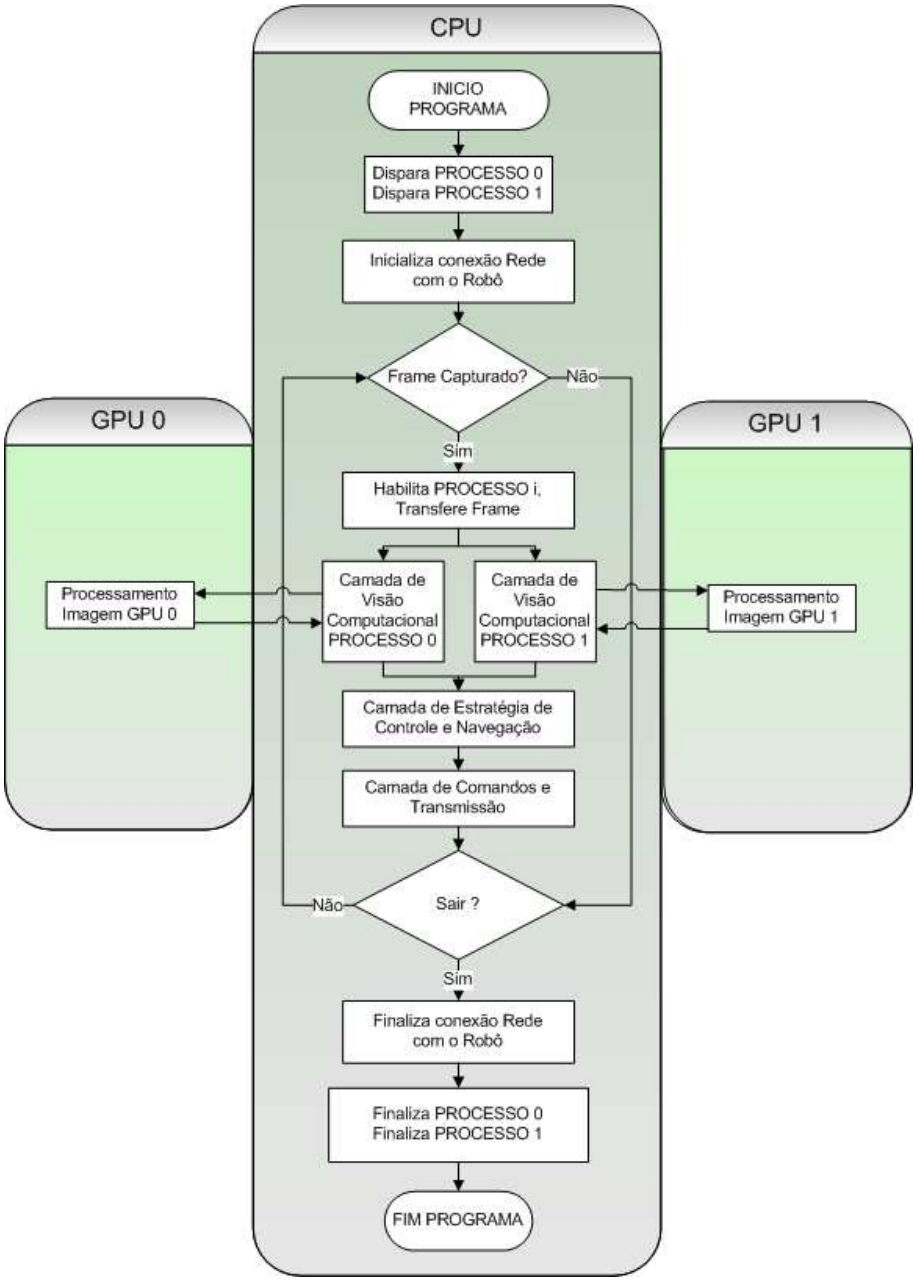


Figura 6.9: Fluxograma do Software desenvolvido

Como observado no fluxograma da Figura 6.9, o software inicialmente cria dois processos que serão responsáveis por controlar cada qual uma GPU (Vide Figura 6.10), sendo conhecido por processos concorrente, pois compartilha os mesmos recursos de um computador como exemplo espaço de memória e tempo de CPU. Esta abstração se fez necessária para executar trechos do código em processadores reais distintos, que é o caso deste trabalho, que envolve as duas GPU´s em uma mesma placa gráfica. É importante ressaltar que apesar das GPU´s estarem na mesma placa gráfica, não compartilham nenhuma informação entre elas, ficando por conta do CPU a troca de informações entres estes dispositivos caso houver alguma necessidade.

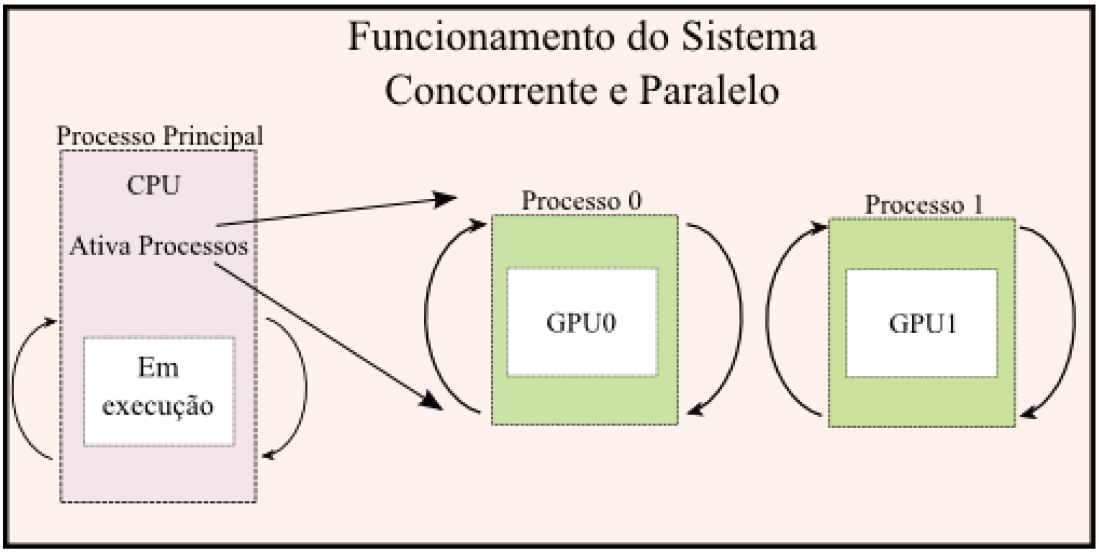


Figura 6.10: Funcionamento do sistema Concorrente e Paralelo

Criado os processos concorrentes, há a necessidade da distribuição de carga entre estes de modo a não sobrecarregar algum processo. Deste modo, como a carga a ser compartilhada são os dados da imagem, este poderia ser dividido em partes iguais e alocado para cada processo executar sua fatia na GPU responsável. No entanto, o fato do algoritmo desenvolvido precisar das informações dos dados de toda a imagem, e visto que, as GPU´s não se comunicam, esta estratégia foi inviabilizada, pois a única alternativa encontrada para contornar este problema era fazer uma copia da imagem para cada processo e este processar sua parte. Esta solução traria uma queda na performance do software.

A estratégia aplicada neste caso foi trabalhar os dados da imagem individualmente em cada processo, onde a cada instante de tempo que a imagem é obtida, esta é alocada para o

processo que não esteja em atividade, isto é, aquele que já tiver concluído sua tarefa, estando ocioso. Como tarefa de cada processo vinculada a esta estratégia, foi encapsular a camada de visão computacional vista no capítulo 5 em cada processo, possuindo então duas camadas de visão computacional, conforme o fluxograma da Figura 6.9.

Nota-se que a forma como foi modelado o processo, em função das imagens adquiridas, a execução dos processos acontece de forma assíncrona, no entanto, deve-se garantir o sincronismo entre a seqüência de imagens adquiridas, ou seja, o processo que está executando a imagem I_t necessariamente deve terminar antes que o processo que está executando a imagem I_{t+1}. Para controle deste sincronismo são utilizados dois semáforos binários, sendo um para cada processo. Este método indica se o processo finalizou a tarefa, sendo assim controlado e garantindo o sincronismo. Para exemplificar a forma como acontece este sincronismo, uma analogia ao mecanismo pipeline pode ser feito, onde os estágios do pipeline seriam os processos (no caso 2), e as instruções como sendo o processamento da imagem dentro da camada de visão computacional. Desta forma a Figura 6.11 apresenta a execução ao longo do tempo t.

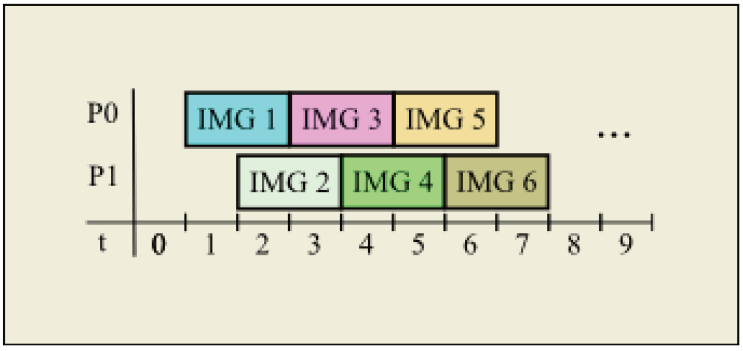


Figura 6.11: Execução do sincronismo entre imagens

De modo geral, salientamos que este software utiliza os princípios do processamento concorrente, advindos da criação e gerenciamento dos processos, do processamento paralelo, definido pelas 2 GPU´s e também com o processamento SIMD, utilizado dentro das GPU´s.

O software foi desenvolvido no LINUX 9.10 utilizando a linguagem C ANSI, a biblioteca SDL (Simple Directmedia Layer) para acesso a rede (comunicação com o robô), a biblioteca OpenGL para interface com o usuário e GPU, e CUDA para aceleração do processamento de imagem.

Capítulo 7

EXPERIMENTOS E RESULTADOS

Neste capítulo são descritos os experimentos realizados, sendo efetuados pela utilização do software apresentado no capítulo 6. O objetivo é avaliar o desempenho do sistema de rastreamento e navegação em tempo real aplicado no controle de robôs autônomos, bem como avaliar suas potencialidades e limitações.

Visando observar de maneira mais criteriosa o sistema de rastreamento, em específico a camada de visão computacional, a seção 7.1 realiza algumas medidas do tempo de processamento gasto pela GPU no cálculo dos filtros abordados. Em seguida, na seção 7.2, alguns casos foram observados para avaliar o rastreamento e a influência de seus principais parâmetros. Após esta análise, os casos apresentados são resultados de simulações realizadas em ambiente real, em áreas internas e externas, visto na seção 7.3. Certamente, a simulação virtual de sistemas de rastreamento e navegação autônomos traz facilidades, no entanto, MIRANDA NETO (2007) diz que é praticamente impossível reproduzir todas as características disponíveis nos ambientes reais, principalmente quando tratamos de um sistema de visão computacional como único sensor de entrada de dados, onde a influência da iluminação é um fator determinante.

7.1 Tempo de processamento na GPU

Esta seção visa fornecer uma avaliação do tempo de processamento gasto pela GPU na execução do processamento de imagens, baseados nas implementações paralelas descrita no capítulo 4.

As tabelas a seguir apresentam os resultados de cada um dos códigos paralelos, seguindo o fluxograma apresentado no capitulo 5, pela figura 5.1. Os experimentos foram feitos com imagens de resolução 320x240 e 640x480, executados sobre quatro diferentes tipos de imagens, visto na figura 7.1. O motivo de avaliar os algoritmos com imagens de diferentes características

deve se ao fato do filtro HMIN e a transformada Watershed possuírem laços de estabilização que são diretamente influenciados pelos dados de entrada.

Figura 7.1: Imagens de entrada para calculo do tempo de processamento da GPU. a) Imagem cameraman b) Imagem lena c) imagem baboon d) imagem peppers

Como primeiro passo, a Tabela 7.1 apresenta a medida do tempo realizada na transferência dos dados da imagem contida na memória da CPU para a memória global da GPU.

Tabela 7.1: Tempo de transferência dos dados na CPU para a GPU.

Dados de entrada	Resolução 320x240	Resolução 640x480
Cameraman	0.6444 (ms)	1.0816 (ms)
Lena	0.6380 (ms)	0.9850 (ms)
Baboon	0.6604 (ms)	1.1954 (ms)
Peppers	0.5887 (ms)	1.1094 (ms)
Tempo médio:	0.6303 (ms)	1.0928 (ms)

Uma questão importante a mencionar é que em várias aplicações, as sucessivas transferências dos dados entre *host* e *device* tornam-se inviável a utilização da GPU, não sendo o caso deste projeto.

Logo adiante, pela

Tabela **7.2** pode-se verificar o tempo gasto para processar o filtro KNN.

Tabela 7.2: Tempo de processamento do filtro KNN em GPU

Dados de entrada	Resolução 320x240	Resolução 640x480
Cameraman	0.5604 (ms)	1.4883 (ms)
Lena	0.6298 (ms)	1.3802 (ms)
Baboon	0.6778 (ms)	1.4533 (ms)
Peppers	0.5621 (ms)	1.2788 (ms)
Tempo médio:	0.6075 (ms)	1.4001 (ms)

Em seguida, a Tabela 7.3 apresenta o tempo de processamento para converter uma imagem colorida para escala de cinza.

Tabela 7.3: Tempo de processamento para converter RGB para Escala de Cinza

Dados de entrada	Resolução 320x240	Resolução 640x480
Cameraman	0.6932 (ms)	0,9038 (ms)
Lena	0.9764 (ms)	0.8865 (ms)
Baboon	0.9151 (ms)	1.3258 (ms)
Peppers	0.7947 (ms)	1.5310 (ms)
Tempo médio:	0.8448 (ms)	1.1617 (ms)

A Tabela 7.4 mostra o tempo gasto no processamento do filtro Sobel.

Tabela 7.4: Tempo de processamento para o filtro Sobel

Dados de entrada	Resolução 320x240	Resolução 640x480
Cameraman	0.8260 (ms)	0.8859 (ms)
Lena	0.7310 (ms)	0.8023 (ms)
Baboon	0.8798 (ms)	1.2028 (ms)
Peppers	0.8768 (ms)	1.2434 (ms)
Tempo médio:	0.8284 (ms)	1.0336 (ms)

De forma semelhante, a Tabela 7.5 apresenta o tempo gasto para processar o filtro reconstrutivo HMIN, observando que este já possui influência dos dados de entrada, bem como o valor do parâmetro *H* que pode também alterar seu tempo de processamento. Neste caso, o valor determinado para o parâmetro *H* foi 15.

Tabela 7.5: Tempo de processamento para o filtro reconstrutivo HMIN

Dados de entrada	Resolução 320x240	Resolução 640x480
Cameraman	32.9193 (ms)	114.8369 (ms)
Lena	39.6275 (ms)	106.5124 (ms)
Baboon	31.4233 (ms)	61.9335 (ms)
Peppers	35.2347 (ms)	64.4120 (ms)
Tempo médio:	34.8012 (ms)	86.9237 (ms)

O último algoritmo medido foi a transformada Watershed, onde na Tabela 7.6 apresenta o tempo de processamento utilizando vizinhança 8, e relembrando que este algoritmo sofre do fator de estabilização, onde influência o tempo de processamento.

Tabela 7.6: Tempo de processamento para a transformada Watershed

Dados de entrada	Resolução 320x240	Resolução 640x480
Cameraman	5.2960 (ms)	37.1740 (ms)
Lena	7.1216 (ms)	38.9620 (ms)
Baboon	6.3160 (ms)	33.8402 (ms)
Peppers	5.4825 (ms)	38.3104 (ms)
Tempo médio:	6.0540 (ms)	37.0716 (ms)

Como resultado final do processamento, realizando a união dos tempos de processamento de cada algoritmo medido, com relação a uma GPU, o tempo total gasto segue conforme a Tabela 7.7.

Tabela 7.7: Tempo total de processamento na GPU

Algoritmos	Resolução 320x240	Resolução 640x480
Transf. Memórias	0.6303 (ms)	1.0928 (ms)
Filtro KNN	0.6075 (ms)	1.4001 (ms)
RGBtoGRAY	0.8448 (ms)	1.1617 (ms)
Filtro Sobel	0.8284 (ms)	1.0336 (ms)
Filtro HMIN	34.8012 (ms)	86.9237 (ms)
T. Watershed	6.0540 (ms)	37.0716 (ms)
Tempo Total:	43.1359 (ms)	127.5907 (ms)

A respeito do tempo de processamento aqui apresentado, deve-se ressaltar que as medições foram realizadas com todo o sistema em funcionamento. Deste modo, existe implícito nos tempos o gerenciamento dos processos concorrentes no CPU, pois cada filtro foi medido dentro de um processo, tendo este processo um tempo de execução escalonado pelo CPU. De fato, como os filtros KNN, RGBtoGRAY e Sobel possuem um kernel para serem processados, eles não sofrem o efeito de controle do CPU, diferente do HMIN e Watershed que possuem mais de um kernel e sua iteração depende do CPU. Este evento pode elevar o tempo de processamento gasto pela GPU. Entretanto, visto pela tabela 7.7, este evento não é significativo o suficiente para inviabilizar a utilização da GPU como coprocessador da CPU, no sentido de processar a imagem em tempo real. Nota-se que o resultado apresentado é bastante expressivo para área de visão robótica, visto que o sistema pode atingir uma taxa aproximada de 47fps para imagens com resolução 320x240 e 16fps para resolução 640x480, considerando o funcionamento das duas GPU´s.

7.2 Rastreamento do Sistema de Visão Computacional

Como parte fundamental do sistema, o rastreamento deve ser avaliado de forma a verificar quais os limitadores do sistema proposto. Nota-se que a solução proposta possui três parâmetros de ajuste que afetam o rastreamento do alvo frente a um dado ambiente, proporcionando um correto rastreamento conforme os valores definidos para os três parâmetros. Tais parâmetros, que foram abordados no capítulo 5, constituem a variável *H* do filtro HMIN, *LimiarSimilaridadeCor* e *LimiarDistancia* do algoritmo de correspondência. Utilizando uma seqüência de imagens obtidas do ambiente com o robô estático, dentre as diversas possibilidades de análise, os seguintes aspectos serão considerados:

A influência isolada da variável *H* no sistema;
A influência isolada da variável *LimiarSimilaridadeCor* no sistema;
A influência isolada da variável *LimiarDistancia* no sistema;
A relação existente entre estas variáveis na execução do rastreamento.

Influência H

O primeiro passo para avaliação da variável *H* foi fixar o valor *LimiarSimilaridadeCor* em *0.01f* e fixar o valor de *LimiarDistancia* em *40px* de forma heurística. A região alvo foi selecionada na mesma posição em todas as variações de *H* que foram *H = {6, 15, 30, 50}*.

Para análise qualitativa das variações em *H* foi utilizado um banco de imagens, conforme mencionado anteriormente, contendo 130 frames. Avalia-se então o parâmetro *H* através da comparação das respostas geradas pelo sistema de rastreamento e o conjunto ideal de respostas, obtendo dois índices de avaliação conhecidos como precisão e revocação.

O índice de precisão mede a relação de regiões corretamente selecionadas frente ao total de regiões selecionadas, e o índice revocação mede a proporção de regiões corretamente selecionadas com relação ao número de regiões que poderiam ser selecionadas corretamente. Para melhor visualização, a figura 7.2 apresenta as regiões que compõe tais índices. As regiões com seus respectivos centróides em azul representam o universo de regiões delimitadas pelo

LimiarDistancia. As regiões com centróides em vermelho representam as regiões selecionadas. O número de regiões corretamente selecionadas é obtido ao observar a quantidade de regiões com centróide vermelho que estão contidos no alvo, que é a camisa. De forma análoga, para obter o número de regiões que poderiam ser selecionadas corretamente, basta observar as regiões com centróide azul contidas na camisa.

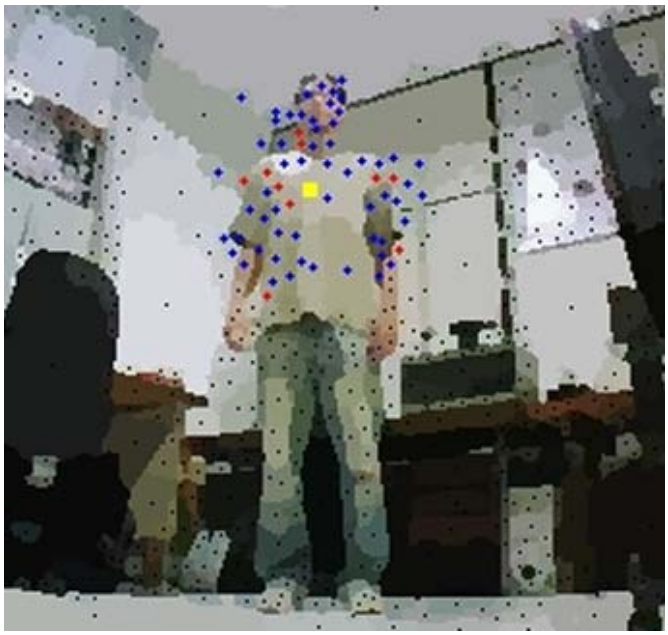


Figura 7.2: Representação das regiões para análise.

Desta forma, pode-se apresentar o índice de precisão pela equação (7.2.1):

$$precisão = \frac{regiões\ corretamente\ selecionadas}{regiões\ selecionadas} \tag{7.2.1}$$

E revocação pela equação (7.2.2):

$$revocação = \frac{regiões\ corretamente\ selecionadas}{regiões\ que\ poderiam\ ser\ selecionadas\ corretamente} \tag{7.2.2}$$

Com base nesta avaliação, a proposta é verificar qual a faixa de valores para o parâmetro *H* que obtenha os melhores índices. Observa-se que existe uma relação inversa entre a precisão e revocação, pois quanto maior for a revocação, menor será a precisão, e vice-versa. Neste sentido, a busca torna-se também encontrar valores que mantenham estes índices em equilíbrio. A tabela 7.8 apresenta a análise para cada variação, tendo a primeira coluna representando a média do número de regiões selecionadas, a segunda coluna indicando a média do número de regiões selecionadas corretamente, a terceira coluna representa a média das regiões que poderiam ser selecionadas corretamente, e por fim as duas últimas com os índices. Ressalta-se que estas médias foram extraídas do banco de imagens.

Tabela 7.8: Análise qualitativa do parâmetro H.

Variação H	Regiões selec.	Regiões selec. corretamente	Regiões pod. ser selec.corretamente	Precisão	Revocação
6	8,34	7,5	44,68	0,9	0,17
15	3,57	3,31	21,25	0,93	0,16
30	2,66	0,76	3,45	0,28	0,22
50	0,55	0,4	2,95	0,72	0,13

Realizando agora uma análise em relação ao processamento de imagem e a influência do ambiente, observa-se que, o aumento no valor de *H* proporciona a união de regiões vizinhas, podendo esta vizinhança unir o fundo com o alvo. Outra questão é o fato da união gerar regiões maiores as quais afetará o valor do atributo RGB médio da região, conseqüentemente a variância dentro desta será maior. Como o sistema de rastreamento leva em conta este atributo para a correspondência de regiões, a influência da luminosidade do ambiente no aumento desta variância pode gerar erros de classificação em relação ao Atributo RGB médio. A Figura 7.3 apresenta o resultado da influência da variável *H*. Sendo o *PR* em amarelo quando o alvo foi associado a alguma região da imagem, e *PR* em vermelho demonstrando a não correspondência.

Figura 7.3: Resultado da influência do parâmetro H no rastreamento.

Influência *LimiarSimilaridadeCor*

Como abordado anteriormente, a avaliação da variável *LimiarSimilaridadeCor* foi fixado o valor de *H* em 15 e o valor de *LimiarDistancia* em *40px*. A região alvo foi selecionada na mesma posição em todas as variações que foram *LimiarSimilaridadeCor = {0.004, 0.008, 0.03, 0.10}*.

Utilizando os mesmos princípios abordados para o parâmetro *H*, a tabela 7.9 apresenta os dados analisados para o parâmetro *LimiarSimilaridadeCor*. Observa-se através dos índices, que uma melhor opção para o valor deste parâmetro varia em torno de 0.008f.

Tabela 7.9: Análise qualitativa do parâmetro LimiarSimilaridadeCor.

Variação H	Regiões selec.	Regiões selec. corretamente	Regiões pod. ser selec.corretamente	Precisão	Revocação
0.004f	1,58	1,56	22,32	0,99	0,07
0.008f	4,86	4,49	17,77	0,92	0,25
0.030f	9,77	0,38	3,41	0,04	0,11
0.100f	10	0,8	9,34	0,08	0,09

Observando a variação escolhida, é evidente que a alteração desta irá causar efeito direto no rastreamento. A escolha de valores abaixo de 0.004f torna o rastreamento muito fino ao ponto de não encontrar nenhuma região similar as informações do alvo, devido ao impacto direto da variação da luminosidade do ambiente. Nota-se também que valores muito altos para o rastreamento proposto, como *0.10f* permite a classificação de regiões não similares, ocorrendo o não rastreamento do alvo. É importante ressaltar que uma classificação errada irá influenciar na classificação seguinte, pois como visto, as informações do alvo são formadas pela média das 10 melhores regiões encontradas, sendo assim, caso uma região classificada errada estiver contida entre estes melhores, isto irá impactar as decisões futuras. A Figura 7.4 apresenta esta influência.

Figura 7.4: Resultado da influência do parâmetro LimiarSimilaridadeCor no rastreamento.

Influência *LimiarDistancia*

Para a avaliação da variável *LimiarDistancia* fixou-se o valor *H* em 15 e o valor de *LimiarSimilaridadeCor* em *0.010f*. A região alvo foi selecionada na mesma posição em todas as variações que foram *LimiarDistancia = {6, 20, 50, 80}*.

A determinação desta variável também irá impactar diretamente no resultado do rastreamento. Aplicando um raio de distância baixo, os centróides das regiões contidas dentro desta circunferência pode não satisfazer a exigência da similaridade, deste modo o alvo não é encontrado. No caso do raio ser muito grande, surge regiões não vinculadas ao alvo, de forma que, se alguma região de fundo for similar ao alvo, isto irá afetar o rastreamento.

A outra questão associada a esta variável diz respeito à taxa de captura da imagem e a velocidade do deslocamento tanto do alvo quanto do robô, pois como a taxa de captura é de 10fps

para este protótipo, se o deslocamento do alvo for superior ao limiar distância, estão o alvo será perdido. De forma análoga, se o deslocamento do alvo estiver dentro do previsto, e neste mesmo instante o robô se locomover de modo que o deslocamento resultante seja maior que a área da circunferência, então mais uma vez o alvo será perdido. Este fato é nitidamente percebido devido aos comandos de movimentação gerados para o robô se locomover não serem condizentes com a movimentação ideal desejada, como mostra a Figura 7.5.

Figura 7.5: Resultado da influência do parâmetro LimiarDistancia no rastreamento.

A relação existente entre as variáveis

Avaliado a influencia individual das variáveis no sistema, é possível estabelecer algumas relações entre estas variáveis. Pode-se concluir que existe uma relação inversa entre as variáveis *LimiarSimilaridadeCor* e *LimiarDistancia*, pois a medida em que o valor de *LimiarDistancia* aumenta, mais fino deve ser a classificação, deste modo o valor do *LimiarSimilaridadeCor* deve

ser reduzido. A outra relação com respeito a variável H é que a medida em que aumenta, o valor do *LimiarDistancia* deve aumentar para atingir os centróide das regiões.

De maneira geral, entendido quais as relações, as influências individuais bem como a faixa de valores que melhor se ajusta ao ambiente, o rastreamento se torna robusto, frente à simplicidade da correspondência entre as regiões. Para apresentar o resultado final, tendo os parâmetros devidamente ajustados, a Figura 7.6 apresenta o rastreamento da mão com diversas deformações bem como a variação da luminosidade. Complementarmente, o vídeo deste experimento pode ser visto em http://br.video.yahoo.com/watch/7642526/20300493.

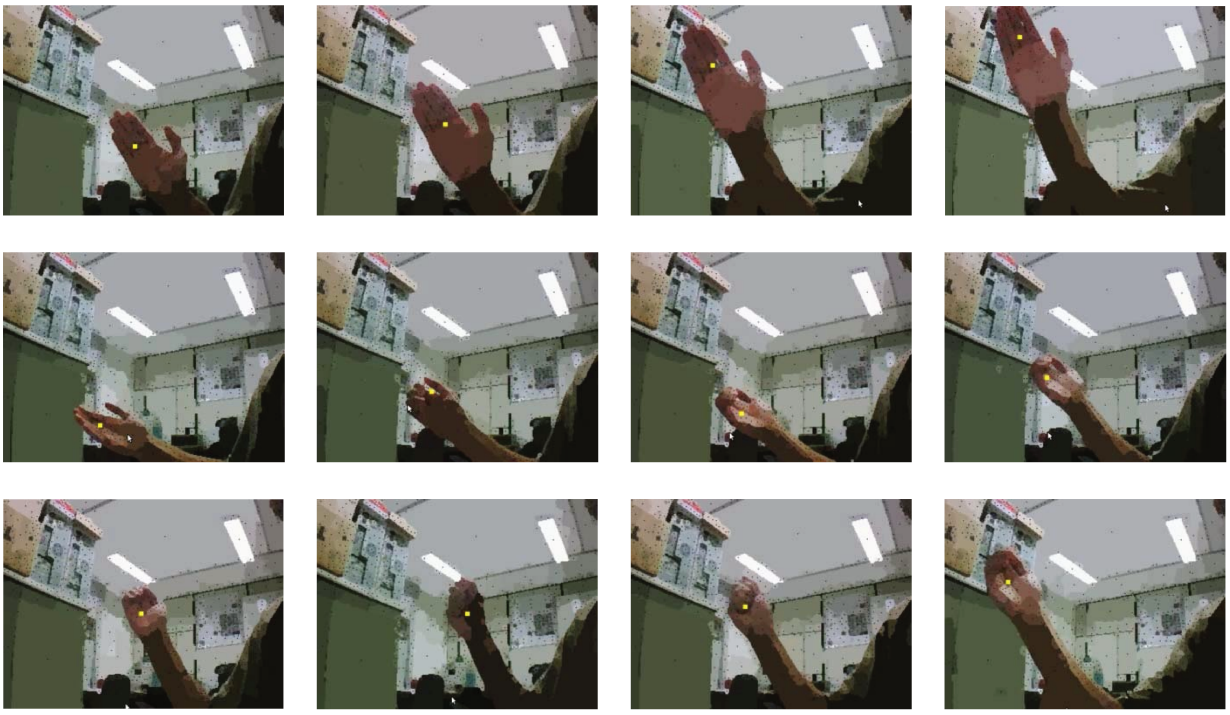


Figura 7.6: Rastreamento da mão com parâmetros devidamente ajustados.

7.3 Simulações em Ambiente real

Os experimentos envolvendo todo o sistema de rastreamento e navegação é apresentada a seguir, sendo divido em 3 casos. O primeiro caso aborda um ambiente interno tendo o rastreamento e a navegação vista pela "visão" do robô, como mostra a Figura 7.7. A princípio o robô está a certa distância do alvo, o qual é selecionado. A seqüência de imagens demonstra o

robô seguindo o alvo, sendo que este está dentro de uma sala e logo depois saindo para o corredor estreito. Apesar do alvo apresenta bastante contraste com o fundo, observa-se em algumas imagens que a influência da luminosidade da lâmpada não afeta o rastreamento do alvo, com relação às tonalidades da cor alvo. O vídeo gerado deste experimento apresenta nitidamente o comportamento do robô diante da relação de deslocamento do alvo frente ao movimento executado pelo robô, conforme discutido na seção 7.2.

Figura 7.7: Simulação em ambiente interno, "Visão" do robô.

O segundo caso aborda um corredor mais largo, onde o sol entrando pelas janelas irá influenciar o sistema de visão do robô apresentado na Figura 7.8. Observa-se que apesar desta influência, o robô é capaz de seguir a pessoa até o limite onde o alvo se perde no ambiente escuro. Na figura 7.8, as imagens das linhas 1 e 3 apresentam o robô seguindo o alvo, sendo que

nas linhas 2 e 4 apresentam respectivamente a visão que o robô esta capturando do ambiente naquele instante.

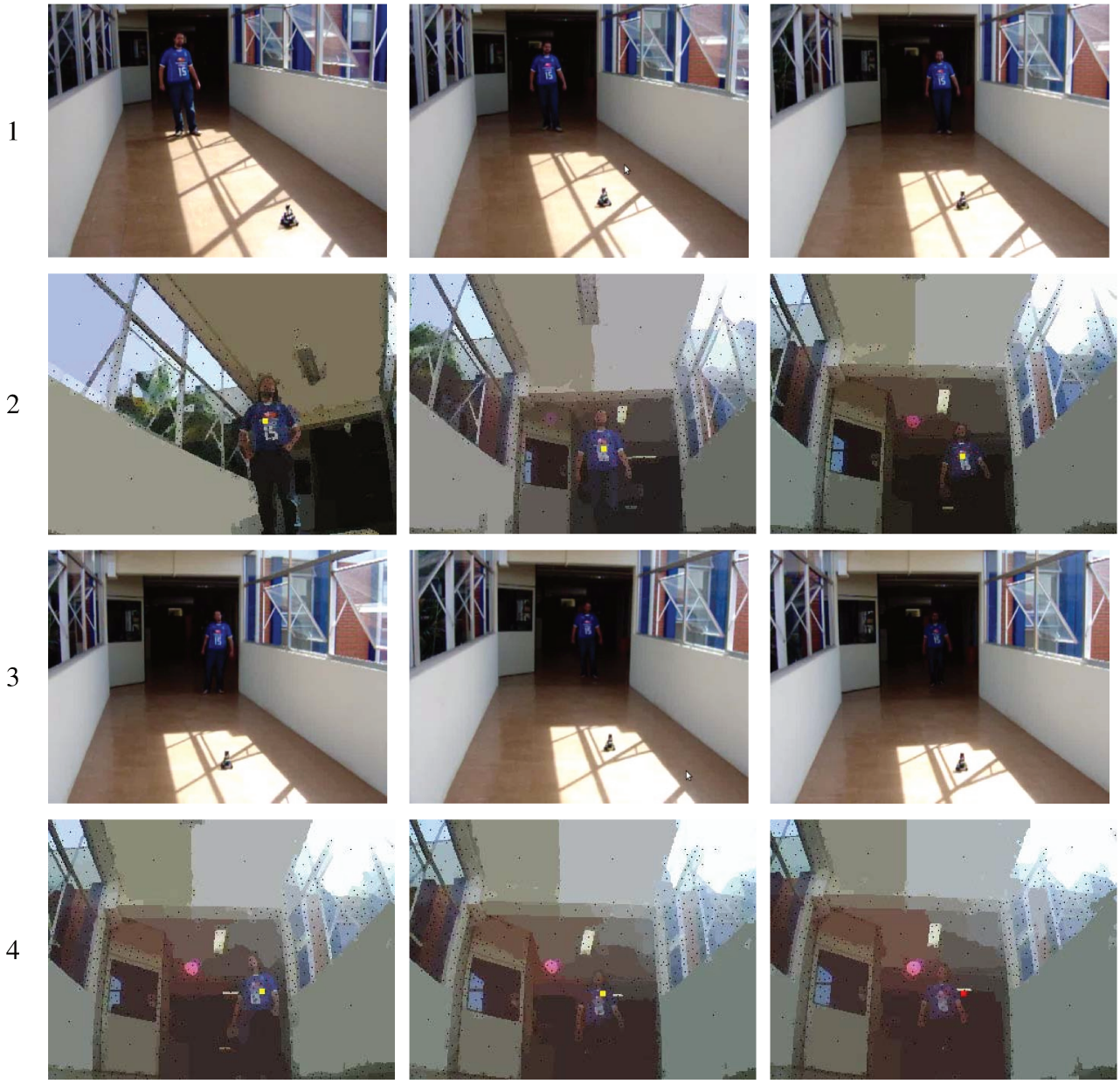


Figura 7.8: Simulação em ambiente interno, corredor influenciado pelo sol.

Por fim, o terceiro caso feito em ambiente externo, demonstrada pela seqüência de imagens da Figura 7.9, o robô perseguindo o alvo. Nota-se neste experimento que além do ambiente sofrer forte influência do sol, a cor do alvo é bastante próxima do fundo dificultando

seu rastreamento. É importante ressaltar que em alguns frames o alvo se perde mais logo é identificado. De forma análoga a vista anteriormente, as linhas 1 e 3 apresentam as imagens do robô perseguindo o alvo, e as linhas 2 e 4 apresentam respectivamente a visão que o robô está tendo naquele momento, com a captura das imagens.

Figura 7.9: Simulação em ambiente externo, com fundo semelhante ao alvo.

Complementarmente, o vídeo do experimento apresentado pela figura 7.7 pode ser visto em http://br.video.yahoo.com/watch/7573142/20080308, o experimento da figura 7.8 em

http://br.video.yahoo.com/watch/7573133/20080266 e o experimento da figura 7.9 em http://br.video.yahoo.com/watch/7573081/20080171.

Capítulo 8

CONCLUSÃO E TRABALHOS FUTUROS

Neste capítulo são feitas as considerações e conclusões baseadas nos estudos, experimentos e análises realizadas. A seção 8.1 apresenta os comentários e considerações específicas a cerca do projeto. Na seção 8.2 realiza-se a conclusão geral, trazendo a contribuições do projeto. Por fim a seção 8.3 aponta as perspectivas futuras, vislumbrando possíveis aprimoramentos, implementações e idéias.

8.1 Comentários

A partir dos resultados obtidos e das análises dos experimentos realizados, a aplicação da abordagem em camadas proposto por MIRANDA NETO e RITTER (2006) proporcionou melhor estruturação do software, conseguindo modularizar o sistema de forma a encapsular cada parte do código condizente com sua funcionalidade nas camadas, tornando possível o sistema distribuído. Conseqüentemente, esta escolha impactou diretamente na complexidade de gerenciamento do sistema, facilitando seu entendimento e controle.

Com relação ao algoritmo aqui desenvolvido, utilizando o poder de aceleração das GPU´s, pode-se concluir que a metodologia proposta apresenta-se como boa candidata para o processo de rastreamento em tempo real. No entanto, observando a influência que os parâmetros *H*, *LimiarDistancia* e *LimiarSimilaridadeCor* exercem sobre a confiabilidade do rastreamento, adicionais informações de características do alvo devem ser incorporadas ao modelo de correspondência para tornar o atual rastreamento robusto e confiável mediante as possíveis variações de luminosidade, rotação, translação, escala, deformação, dentre outras que possam vir a afetar este procedimento.

Avaliando o desempenho do processamento de imagem na GPU, observou-se que o tempo demandado no processamento de uma imagem com resolução de 320x240 para cada camada da

visão computacional foi de aproximadamente 43ms (≅23 fps), e para imagem com resolução de 640x480 foi de aproximadamente 127ms (≅8 fps) (a configuração do computador utilizado pode ser vista na seção 6.2.1). Tais resultados garantem a aplicação em tempo real, bem como proporcionam a incorporação de diversas melhorias no sistema a ponto de satisfazer as necessidades apresentadas anteriormente. Outro fator relevante é que apesar do tempo de execução dos algoritmos na GPU apresentarem bons resultados para sistemas de tempo real, os algoritmos não foram implementados de forma totalmente otimizada, podendo reduzir ainda mais o tempo de processamento caso esta otimização seja feita.

Com relação ao sistema de controle e navegação juntamente com a plataforma robótica, observou a necessidade de um controle mais preciso nos comandos enviados ao robô.

8.2 Conclusão

Na concepção geral, o desenvolvimento de uma solução para rastreamento e navegação de robôs móveis em ambientes desconhecidos, baseado no controle visual utilizando a mono-visão como único sensor de entrada de dados foi concluído.

Esta solução de rastreamento e navegação para robôs móveis que interagem em ambientes reais, inerentemente dinâmicos, envolve a integração de diversos aspectos multidisciplinares necessários para a completa realização das tarefas, tornando complexa sua implementação. Além das técnicas de visão computacional estudadas, o projeto também incluiu o conhecimento de vários requisitos com o intuito de solucionar a compatibilidade com sistemas de tempo real, a compatibilidade com a nova geração de computadores multiprocessados, arquitetura cliente-servidor e de sistemas distribuídos.

Este trabalho apresentou um sistema cuja solução pode ser aplicada em diversas atividades. Na robótica terrestre podemos citar o seguimento automático entre dois veículos como exemplo os comboios do exército, e, mira automática para perseguição de alvos nos sistemas de armamento. Na robótica aérea podemos citar as inspeções de fronteira seguindo alvos suspeitos, ou até mesmo a perseguição entre caças.

As principais contribuições de nosso trabalho estão associadas aos métodos aplicados à visão computacional, destacando as implementações do processamento na GPU vistas no capítulo 4, e também no gerenciamento de sistema distribuído escalável, podendo adicionar várias placas gráficas para aumentar o poder de processamento da camada de visão computacional.

Muitos avanços na área de sistemas robóticos utilizando a visão computacional tem sido expressivos, porém, ainda existe uma grande lacuna entre o que o ser humano é capaz de inferir a partir de uma imagem, e o que os mais avançados sistemas computacionais são capazes. Contudo, a avaliação deste projeto aponta simplesmente para o início de uma nova concepção de programação paralela que proporcionará o desenvolvimento de métodos computacionais robustos aplicados aos sistemas robóticos em tempo real que aproximará cada vez mais esta relação.

8.3 Perspectivas Futuras

Como perspectiva futura, pretende-se tornar o método de correspondência mais robusto, utilizando para isto as métricas discutidas por GALO (2003), que aborda a união de diversas características de uma primitiva para estabelecimento da correspondência.

Estudar formas alternativas às implementadas, de maneira a alcançar melhor otimização para os algoritmos paralelos desenvolvidos, bem como paralelizar as técnicas de extração de características e correspondência que se encontram serializadas.

Com base também na utilização da GPU, proporcionar uma semântica topológica envolvendo as regiões com características distintas extraídas de um mesmo alvo, bem como desenvolver um algoritmo para inferir o posicionamento do alvo na imagem seguinte, como exemplo o filtro de Kalman ou filtro de partículas. Desta forma, os diversos objetos semelhantes ao alvo que possam surgir na cena, não seriam confundidos com o alvo em questão.

Para a camada de estratégia e navegação pretende-se desenvolver um algoritmo que possa proporcionar o controle robusto do robô, podendo até implementar alguma técnica para desviar de algum objeto que surgir entre o robô e o alvo.

A última perspectiva futura seria a utilização de uma plataforma robótica mais robusta para aquisição de imagens e execução da ação de movimentos precisos, sem que este interfira na obtenção das imagens.

REFERÊNCIAS BIBLIOGRÁFICAS

ABDOU, I.; PRATT, W. K. **Quantitative design and evaluation of enhancement / thresholding edge detectors**. Proceedings of the IEEE. [S.l.]: [s.n.]. 1979. p. 753–763.

AMORIM, R. M. **Solução das Equações do Bidomínio em PRocessadores Gráficos**. Universidade Federal de Juiz de Fora - Dissertação (Mestrado em Modelagem Computacional). Juiz de Fora, p. 151. 2009.

AREFI, H.; HAHN, M. A Morphological Reconstruction Algorithm for separating off-terrain points from terrain points in laser scanning data. **ISPRS WG III/3, III/4, V/3 Workshop "Laser scanning 2005"**, Enschede, the Nertherlands, p. 12-14, September 2005.

AWCOCK, G. W.; THOMAS, R. **Applide Image Processing**. New York: McGraw Hill International Editions, 1996.

BAGGIO, D. L. **GPGPU Based Image Segmentation Livewire Algorithm Implementation**. Technological Institute of Aeronautics - Thesis of Master in Science. São José dos Campos, p. 108. 2007.

BALKENIUS, C.; KOPP, L. **Visual tracking and target selection for mobile robots**. Proceedings of the First Euromicro Workshop on Advanced Mobile Robots (EUROBOT '96). Los Alamitos: IEEE Computer Society Press. 1996. p. pp. 166-171.

BALLARD, D. H.; BROWN, C. M. **Computer Vision**. 1. ed. Englewood Cliffs, New Jersey: Prentice-Hall. INC., 1982.

BARRERA, J.; BANON, J.; LOTUFO, R. A. **Image algebra and morphological image processing**. International Symposium on Optics, Imaging and Instrumentation, SPIE´s Annual Meeting. San Diego, USA: [s.n.]. 1994.

BERTOZZI, M.; BROGGI, A.; FASCIOLI, A. **Vision-based intelligent vehicles:** state of the art and perspectives. Robotics and Autonomous systems. [S.l.]: [s.n.]. 2000. p. 1-16.

BEUCHER, S.; MEYER, F. **The morphological approach to segmentation:** The watershed transformation. In Proc. of the International Symposium on Mathematical Morphology. [S.l.]: [s.n.]. 1993. p. 433-481.

BIENIEK, A.; MOGA, A. **A connected component approach to the watershed segmentation**. ISMM '98: Proceedings of the fourth international symposium on Mathematical morphology and its applications to image and signal processing. Norwell, MA, USA: Kluwer Academic Publishers. 1998. p. 215–222.

BROIDA, T.; CHELLAPA, R. Estimation of object motion parameters from noisy images. **IEE Trans. Patt. Analy. Mach. Intell.**, v. 8, n. 1, p. 90-99, 1986.

BUADES, A.; COLL, B.; MOREL, J. **Neighborhood Filters and PDE´s**. CMLA - Technical Report. [S.l.]: [s.n.]. 2005.

BUCK, I. et al. Brook for gpus: stream computing on graphics hardware. **ACM Trans. Graph.**, New York, NY, USA, v. 23, n. 3, p. 777–786, 2004. ISSN ISSN 0730-0301.

CAVALLARO, A.; STEIGER, O.; EBRAHIMI, T. **Multiple video object tracking in complex scenes**. Proceedings of ACM International Conference on Multimedia. Juan Les Pins: [s.n.]. 2002. p. 523-532.

CAZANGI, R. R. **Uma Proposta Evolutiva para Controle Inteligente em Navegação Autônoma de Robôs**. Universidade Estadual de Campinas - UNICAMP - Dissertação de Mestrado(Faculdade de Engenharia Elétrica e de Computação). [S.l.]. 2004.

CHEN, T. M.; LUO, R. C.; HSIAO, T. H. **Visual tracking using adaptive color histogram model**. Industrial Electronics Society 3. [S.l.]: [s.n.]. 1999. p. 1336-1341.

COMANICIU, D.; RAMESH, V.; MEER, P. Kernel-based object tracking. **IEEE Trans. Patt. Analy. Mach. Intell.**, v. 25, p. 564-575, 2003.

COUSTY, J. et al. Watershed cuts: Minimum spanning forests and the drop of water principle. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 31, n. 8, p. 1362–1374, 2009.

CUDA. NVIDIA CUDA SDK - Graphics Interop. **NVIDIA DEVELOPER ZONE**, 2007. Disponivel em: <http://developer.download.nvidia.com/compute/cuda/1_1/Website/Graphics_Interop.html>. Acesso em: 15 abr. 2010.

DOLCE, O.; POMPEO, J. N. **Fundamentos de Matemática Elementar - Geometria Plana**. 7. ed. [S.l.]: [s.n.], v. 9, 2005. 456p. p.

DOUGHERTY, E. R.; LOTUFO, R. A. **Morphological Processing of Gray-Scale Images**. Bellingham: SPIE Press, 2003. DOI: 10.1117/3.501104.Ch6.

FALCÃO, A. X.; STOLFI, J.; LOTUFO, R. A. The image foresting transform: theory, algorithms, and applications. **Pattern Analysis and Machine Intelligence, IEEE Transactions**, v. 26, n. 1, p. 19–29, 2004.

FARRUGIA, J. P. et al. **Gpucv:** A framework for image processing acceleration with graphics processors. ICME. IEEE. [S.l.]: [s.n.]. 2006. p. 585–588.

GALO, M. **Automação dos processos de correspondência e orientação relativa em visão estéreo**. Universidade Estadual de Campinas - UNICAMP - Tese de Doutorado(Faculdade de Engenharia Elétrica e de Computação). Campinas, p. 262. 2003.

GONZALEZ, R. C.; WOODS, R. E. **Processamento de Imagens Digitais**. São Paulo: Edgard Blucher LTDA., 2000.

GONZALEZ, R. C.; WOODS, R. E. **Digital Image Processing**. 2. ed. New Jersey: Prentice Hall, v. 1, 2002.

GRIESSER, A. et al. **GPU-based foreground-background segmentation using na extended colinearity criterion**. In Proceedings of Vision, Modeling, and Visualization. [S.l.]: [s.n.]. 2005. p. 319-326.

HARALICK, R.; SHAPIRO, L. G. **Computer and Robot Vision**. [S.l.]: Addison-Wesley Publishing Company, v. II, 1993. 630 p.

HAWICK, K. A.; LEIST, A.; PLAYNE, D. P. **Parallel Graph Component Labelling with GPUs and CUDA**. Technical report, Institute of Information and Mathematical Sciences, Massey University. Auckland, New Zealand: [s.n.]. 2009. http://www.massey.ac.nz/~kahawick/cstn/089/cstn-089.html.

HORN, B. K. P. **Robot Vision**. [S.l.]: MIT press, 1986.

IRANI, M.; ANANDAN, P. About Direct Method. In: TRIGGS, B. A. Z. A. A. S. R. **Vision Algorithms:** Theory and Practice. [S.l.]: Springer Berlin / Heidelberg, v. 1883, 2000. p. 267-277. http://dx.doi.org/10.1007/3-540-44480-7_18.

JAIN, R.; KASTURI, R.; SCHUNCK, B. G. **Machine Vision**. 1. ed. New York: McGraw-Hill International Editions, 1995.

KANG, J.; COHEN, I.; MEDIONI, G. **Object reacquisition using geometric invariant appearance model**. International Conference on Pattern Recongnition (ICPR). [S.l.]: [s.n.]. 2004. p. 759-762.

KHARLAMOV, A.; PODLOZHNYUK, V. **Imagem Denoising**. NVIDIA Corporation. [S.l.]. 2007. Acessado em 10-06-10: http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/imageDenoising/doc/imageDenoising.pdf.

KHIYAL, M. S. H.; KHAN, A.; BIBI, A. Modified watershed algorithm for segmentation of 2D images. **The Journal of Issues in informing Science and information Technology**, v. 6, 2009. http://iisit.org/Vol6/IISITv6p877-886Khiyal593.pdf.

KHRONOS. **OpenCL - The open standard for parallel programming of heterogeneous systems**. Khronos Group. [S.l.]. 2008. Acessado em: 21/05/2010 - http://www.khronos.org/opencl/.

KIRK, D. B.; HWU, W. **Programming Massively Parallel Processors:** A Hands-on Approach, chapter Parallel Programming and Computational Thinking. 1. ed. Burlington, MA, USA: Morgan Kaufmann, 2010.

LEFOHN, A.; KNISS, J. M.; OWENS, J. D. Implementing efficient parallel data structures on gpus. In: PHARR, M.; FERNANDO, R. **GPU Gems 2**. [S.l.]: Addison-Wesley, 2005. p. 521–545. ISBN ISBN 0321335597.

LOPES, R. R. **Um modelo perceptivo de limiarização de imagens Digitais**. Universidade Federal do Paraná (UFPR) - Dissertação de Mestrado. [S.l.]. 2003.

LOTUFO, R.; FALCÃO, A. The ordered queue and the optimality of the watershed approaches. **In Proceedings of the 5th International Symposium on Mathematical Morphology and its Applications to Image and Signal Processing**, v. 18, p. 341–350, 2000.

LUO, R. C.; CHEN, T. M. **Autonomous mobile target tracking system based on grey-fuzzy control algorithm**. IEEE Transaction on Industrial Eletronics. [S.l.]: [s.n.]. 2000. p. 920-931.

LUO, Y.; DURAISWAMI, R. **Canny Edge Detection on Nvidia CUDA**. CVGPU08. [S.l.]: [s.n.]. 2008. http://www.visionbib.com/bibliography/edge226.html#TT19952.

MENDES, V.; FADEL, E. **Polícia Federal testa VANT israelense e Despreza congêneres Nacionais**. São Paulo. 2009. Acessado em 11/05/2010 - http://defesabr.com/blog/index.php/16/07/2009/policia-federal-testa-vant-israelense-e-despreza-congeneres-nacionais/.

MEYER, F. Topographic distance and watershed lines. **Signal Processing**, v. 38, n. 1, p. 113–125, 1994.

MIRANDA NETO, A. D. **Navegação de robôs autônomos baseada em monovisão**. Universidade Estadual de Campinas - UNICAMP - Dissertação de Mestrado(Faculdade de Engenharia Mecânica). Campinas - São Paulo. 2007.

MIRANDA NETO, A.; RITTER, L. . **A Simple and Efficient Road Detection Algorithm for Real Time Autonomous Navigation based on Monocular Vision**. IEE 3rd Latin American Robotics Symposium (LARS 2006). [S.l.]: [s.n.]. 2006.

MOESLUND, T. B.; GRANUM, E. **A Survey of Computer Vision-Based Human Motion Capture.** Computer Vision and Image Understanding 81. [S.l.]: [s.n.]. 2001. p. 231–268. doi:10.1006/cviu.2000.0897.

MORTENSEN, E. N.; BARRETT, W. A. **Intelligent scissors for image composition**. SIGGRAPH ’95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques. New York, NY, USA, p. 191–198. 1995. (ISBN 0-89791-701-4).

NVIDIA. **Nvidia cuda Programing guide 3.0**. NVIDIA Corporation. Santa Clara, CA. 2010. http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_CUDA_ProgrammingGuide.pdf.

OWENS, J. D. et al. A Survey of General-Purpose Computation on Graphics Hardware. **Eurographics 2005, State of the Art Reports**, p. 21-51, Agosto 2005.

OWENS, J. D. et al. A Survey of General-Purpose Computation on Graphics Hardware. **Computer Graphics Forum**, v. 26, n. 1, p. 80–113, Março 2007.

OWENS, J. D. et al. Gpu computing. **Proceedings of the IEEE**, v. 96, n. 5, p. 879-899, 2008.

OZYILDIZ, E.; KRAHNSTOVER, N.; SHARMA, R. **Adaptive texture and color segmentation for tracking moving objects**. Pattern Recognition. [S.l.]: [s.n.]. 2002. p. 2013-2029.

PEREIRA, F. L. **Sistemas e Veículos Autônomos – Aplicações na Defesa**. Instituto de Defesa Nacional. [S.l.], p. 82. 2005. http://paginas.fe.up.pt/~flp/papers/SVA-AD_flp_cdn05.pdf.

PHARR, M.; FERNANDO, R. **GPU Gems 2:** programming techniques for high-performance graphics and general-purpose computation. [S.l.]: Addison Wesley, 2005. ISBN ISBN 0-321-33559-7.

ROJAS, R. **Car Steered with Driver's Eyes**. Institute of Computer Science, Freie Universität Berlin. berlin. 2010. Acessado em 23/04/2010 - http://www.fu-berlin.de/en/presse/fup/2010/fup_10_106/index.html.

RONDINA, J. **Segmentação interativa do ventrículo esquerdo em seqüência de imagens de ressonância magnética (Cine MR)**. Universidade Estadual de Campinas - UNICAMP - Dissertação de Mestrado(Faculdade de Engenharia Elétrica e de Computação). Campinas. 2001.

SEBE, N. et al. **Machine Learning in Computer Vision**. 1. ed. [S.l.]: Springer, 2005. 240 p.

SHAPIRO, L.; STOCKMAN, G. **Computer Vision**. 1. ed. [S.l.]: Prentice Hall, 2001. 609 p.

SHIROMA, P. M. **Controle por Visão de veículos Robóticos**. Univesidade Estadual de Campinas - UNICAMP - Dissertação Mestrado(Ciência da Computação). Campinas. 2004.

SILVEIRA, G.; MALIS, E. Unified direct visual tracking of rigid and deformable surfaces under generic illumination changes in grayscale and color images. **International Journal of Computer Vision**, v. 89, p. 84-105, 2010. ISSN 1.

SILVEIRA, V. **Cresce investimento em projetos de inovação na área da defesa**. IPEA. São Paulo. 2010. Acessado em 11/05/10 - http://www.ipea.gov.br/003/00301009.jsp?ttCD_CHAVE=13712.

SMITH, S. M.; BRADY, J. M. Susan – a new approach to low level image processing. **International Journal of Computer Vision**, v. 23, n. 1, p. 45-78, 1997.

TARJAN, R. E. **Data structures and network algorithms**. Society for Industrial and Applied Mathematics. Philadelphia, PA, USA: [s.n.]. 1983.

TOMASI, C.; MANDUCHI, R. **Bilateral filtering for gray and color images**. Sixth International Conference on Computer Vision. [S.l.]: [s.n.]. 1998. p. 839-846.

TSAI, L. W. **Robot analysis:** the mechanics of serial and parallel manipulators. New York, NY - USA: John Wiley & Sons, 1999.

VASCONCELOS, C. N. **Algoritmos para Processamento de Imagens e Visão Computacional para Arquiteturas Paralelas em Placas Gráficas**. Pontifícia Universidade Católica do Rio de Janeiro, (PUC-Rio) - Tese de Doutorado(Departamento de Informática). Rio de janeiro, p. 155. 2009.

VIDAL, F. B. **Rastreamento visual de objetos utilizando métodos de similaridade de regiões e filtragem estocástica**. Universidade de Brasília - Tese de Doutorado(Departamento de Engenharia Elétrica). Brasília, p. 99. 2009.

VINCENT, L. Morphological Gray scale Reconstruction in Image Analysis: Applications and Efficient Algorithms. **IEEE TRANSACTIONS ON IMAGE PROCESSING**, v. 2, n. 2, Abril 1993.

VINCENT, L.; SOILLE, P. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 13, n. 6, p. 583–598, 1991.

VITOR, G. B.; FERREIRA, J. V.; KÖRBES, A. **Fast image segmentation by watershed transform on graphical hardware**. In Proceedings of the 30° CILAMCE. Armação dos Búzios: [s.n.]. 2009.

YANG, R.; WELCH, G. Fast image segmentation and smoothing using commodity graphics hardware. **Journal of graphics tools**, v. 7, n. 4, p. 91-100, 2003.

YILMAZ, A. **Object Tracking by Asymmetric Kernel Mean Shift with Automatic Scale and Orientation Selection**. Ohio State University. IEE 1-4244-1180-7/07. [S.l.]: [s.n.]. 2007.

YILMAZ, A.; JAVED, O.; SHAH, M. **Object Tracking:** A survey. ACM Comput. Surv. 38. [S.l.]: [s.n.]. 2006. p. 45. Article 13.

APÊNDICE A - Informações da GPU

```
CUDA Device Query (Runtime API) version (CUDART static linking)
There are 2 devices supporting CUDA
```

Device 0: "GeForce GTX 295"

```
  CUDA Driver Version:                           2.30
  CUDA Runtime Version:                          2.30
  CUDA Capability Major revision number:         1
  CUDA Capability Minor revision number:         3
  Total amount of global memory:                 938803200 bytes
  Number of multiprocessors:                     30
  Number of cores:                               240
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       16384 bytes
  Total number of registers available per block: 16384
  Warp size:                                     32
  Maximum number of threads per block:           512
  Maximum sizes of each dimension of a block:    512 x 512 x 64
  Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
  Maximum memory pitch:                          262144 bytes
  Texture alignment:                             256 bytes
  Clock rate:                                    1.24 GHz
  Concurrent copy and execution:                 Yes
  Run time limit on kernels:                     Yes
  Integrated:                                    No
  Support host page-locked memory mapping:       Yes
  Compute mode:                                  Default (multiple host
threads can use this device simultaneously)
```

Device 1: "GeForce GTX 295"

```
  CUDA Driver Version:                           2.30
  CUDA Runtime Version:                          2.30
  CUDA Capability Major revision number:         1
  CUDA Capability Minor revision number:         3
  Total amount of global memory:                 939261952 bytes
  Number of multiprocessors:                     30
  Number of cores:                               240
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       16384 bytes
  Total number of registers available per block: 16384
  Warp size:                                     32
  Maximum number of threads per block:           512
  Maximum sizes of each dimension of a block:    512 x 512 x 64
  Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
  Maximum memory pitch:                          262144 bytes
  Texture alignment:                             256 bytes
  Clock rate:                                    1.24 GHz
  Concurrent copy and execution:                 Yes
  Run time limit on kernels:                     No
  Integrated:                                    No
  Support host page-locked memory mapping:       Yes
  Compute mode:                                  Default (multiple host
threads can use this device simultaneously)
```