UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA

DEPARTAMENTO DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

Este exemplar corresponde à redação final da t
defendida por Fernando Carvalho de Barros
e aprovada pela Comis
Julgadora em 20 12 / 91.

Orientador

UM SISTEMA AUTOMÁTICO DE LOCALIZAÇÃO DE OBJETOS NO ESPAÇO UTILIZANDO PROCESSAMENTO PARALELO

Fernando Carvalho de Barros

Orientador: Prof. Dr. Clésio Luiz Tozzi

Tese apresentada à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas, como parte dos requisitos exigidos para a obtenção do título de MESTRE EM ENGENHARIA ELÉTRICA.

- 1991 -

RESUMO

Esta dissertação apresenta o desenvolvimento e implementação de um sistema de Visão Computacional monocular para reconhecimento e localização de objetos no espaço tri-dimensional. São apresentadas também as técnicas de Processamento Paralelo MIMD e SIMD aplicadas à este sistema, visando uma redução de seu tempo de execução total. Este trabalho é multidisciplinar e pode ser inserido nos contextos de Visão Computacional, Automação Industrial, Robótica e Processamento Paralelo.

São apresentados resultados experimentais do sistema aplicado à imagens reais. Procura-se mostrar como os resultados de precisão e velocidade conseguidos no sistema possibilitam sua utilização em tarefas típicas de ambientes industriais.

Como um resultado prático, mostra-se que, utilizando-se 9 transputers, consegue-se que o sistema reconheça e localize, em 33ms, um paralelepípedo com complexidade de 706 pixels de borda. Mostra-se também que a precisão dos resultados obtidos situa-se dentro das especificações atualmente aceitas pela literatura especializada.

ABSTRACT

This dissertation describes the development and implementation of a Computer Vision System for the recognition and localization of objects in 3D space through monocular computer vision. This work is multidisciplinary and it can be inserted in the context of Computer Vision, Industrial Automation, Robotic and Parallel Processing.

It is shown the experimental results of this system applied to real images. And it is also shown how these results - mainly in precision and speed - allows its use in typical tasks in industrial environments.

As a practical result, with nine tranputers, real-time operation is achieved for an image of a parallelepiped with 706 edge pixels complexity. It is also presented that the precision achieved by this system falls into the specifications accepted by the specialyzed literature.

Dedico este trabalho à
minha Família:
José e Estela,
Cristina e Patricia

AGRADECIMENTOS

Aos meus pais, José e Estela, pelo amor, apoio e compreensão dados em todos os momentos e por sempre me darem exemplos de perseverança e humildade.

À minha Cí, pela paciência, companheirismo e incentivo constantes em nossa caminhada.

Ao professor Clésio L. Tozzi, pela amizade, confiança e valiosa orientação, indispensáveis à realização do presente trabalho.

Aos colegas Diego A. Pizzarro, Castanho, Luciano, Gorgônio, Miguel, Ricardo Loureiro e Ricardo Lüders, por sua ajuda sempre oportuna.

Aos professores Wagner C. do Amaral e Léo Pini Magalhães, pelo apoio, confiança e pelas sugestões dadas durante esses anos.

E a todos aqueles que colaboraram, direta ou indiretamente, para a realização deste trabalho.

Este trabalho contou com o apoio financeiro da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) através do contrato nº 89/1290-1.

Este trabalho foi conduzido utilizando-se os recursos computacionais dos seguintes órgãos:

- Laboratório de Engenharia de Computação e Automação Industrial (LCA) da faculdade de Engenharia Elétrica (FEE) da UNICAMP.

- Centro Tecnológico para a Informática (CTI), Campinas, SP.

- Northeast Parallel Architectures Center (NPAC) na Universidade de Syracuse, NY, EUA.

PREFÁCIO

A **Visão** é o nome dado ao processo através do qual relacionamos imagens com nosso conhecimento anterior, capacitando-nos a interagir inteligentemente com o mundo. Através da Visão podemos adquirir noções sobre as localizações ("onde") e identidades ("o que") dos objetos e as relações entre eles.

Segundo MARR (1982), a Visão é, em primeira instância, uma tarefa de processamento de imagens, responsável pela filtragem e extração de uma parte das informações disponíveis no ambiente. O órgão responsável por essa tarefa é o olho. Contudo, a Visão não pode ser pensada somente como uma tarefa de processamento de imagens. Se somos capazes de saber "onde" e "o que", é porque nosso córtex visual deve ser capaz de representar a informação fornecida pelos olhos, de modo a responder eficientemente a questões impostas pelo meio ambiente. O olho é o sensor mas o córtex visual do nosso cérebro é nosso verdadeiro órgão visual. Outros sentidos, que não a visão, podem auxiliar essa tarefa, fornecendo informação visual para o cérebro. Por exemplo, os sensores de ultra-som dos morcegos, etc.

A **Visão Computacional** é o campo de estudos que pretende compreender a visão humana e capacitar máquinas com algumas das habilidades do sistema visual. A primeira atividade está ligada à biologia, principalmente à neurofisiologia, conforme RUMELHART e McCLELLAND (1986) e ARBIB e HANSON (1987). E a segunda, que será vista com mais detalhes neste trabalho, está ligada à engenharia.

Um **Sistema de Visão Computacional** é um sistema capaz de adquirir uma ou mais imagens de um objeto, capaz de processar, analisar e medir atributos da imagem adquirida e interpretar os resultados dessas medidas. Essas medidas são transformadas em representações de tal modo que, em um nível superior de abstração, alguma decisão possa ser tomada a respeito do objeto. Considera-se o sistema de Visão Computacional como uma parte de uma entidade maior que interage com o ambiente. O sistema de Visão Computacional é, então, um "elemento de um ciclo de realimentação", segundo BRADY (1989), sendo responsável

apenas pelo sensoreamento e interpretação, enquanto outros elementos são dedicados a outras tarefas como tomadas de decisão, atuação, etc.

A Visão Computacional está intimamente relacionada a outras áreas, como: Processamento de Imagens, Reconhecimento de Padrões, Robótica e Inteligência Artificial. Segundo WALLACE (1988), aproximadamente 25% dos robôs industriais atuais são equipados com alguma forma de sensores de Visão.

Um objetivo para um sistema de Visão computacional, por exemplo, seria a análise em tempo-real de imagens com qualidade de TV. Imagens de TV são geradas a uma taxa de 30 quadros por segundo, portanto, ter-se-ia uma necessidade de processar uma imagem em aproximadamente 33ms.

Em face a essa demanda por altas taxas de computação, e à limitação das arquiteturas sequenciais, algumas abordagens têm sido tentadas através da utilização de **Computadores Paralelos**, conforme BACKUS (1977). Em geral, tem-se descoberto que, tanto os processadores têm de ser, por si só, mais potentes, quanto os sistemas precisam ser projetados de forma a que muitos processsadores possam ser montados em um sistema com uma estrutura simples, potente, modular e confiável.

A presente tese tem por finalidade estudar a aplicação de técnicas de **Processamento Paralelo** em um problema de **Visão Computacional**. Dentro da extensa classe de problemas em Visão Computacional, escolheu-se um com possibilidades de aplicação prática: o problema da "localização de objetos no espaço 3D em tempo-real". E dentro do conjunto de arquiteturas paralelas, dois tipos de máquinas foram utilizadas para "paralelização" do algoritmo: MIMD e SIMD.

SUMÁRIO

CAPÍTULO I : INTRODUÇÃO.....1

CAPÍTULO II : O ALGORITMO SERIAL.....7

2.1 - Reconhecimento e Localização de Objetos no Espaço 3D.....7

2.2 - As Fases do Algoritmo.....9

2.2.1 - Detecção de Bordas.....10

2.2.2 - Afinamento.....16

2.2.3 - Identificação das Bordas do Objeto.....19

2.2.4 - Classificação das Retas.....29

2.2.5 - Cálculo das Propriedades.....31

2.2.6 - Ajuste das Equações das Retas.....35

2.2.7 - Calibração.....37

CAPÍTULO III : CONSIDERAÇÕES SOBRE PROCESSAMENTO PARALELO.....43

3.1 - Arquiteturas Paralelas.....43

3.1.1 - Arquitetura MIMD.....43

3.1.2 - Arquitetura SIMD.....49

3.2 - Medidas de Desempenho do Paralelismo.....53

3.3 - A Implementação de Algoritmos Paralelos.....55

3.3.1 - Desenvolvimento de Software para Arquiteturas MIMD.....57

3.3.2 - Desenvolvimento de Software para Arquiteturas SIMD.....61

CAPÍTULO IV : O ALGORITMO PARALELO.....65

4.1 - Motivação.....65

4.2 - Detecção de Bordas.....66

4.2.1 - A Implementação Sistólica.....67

4.2.2 - A Implementação Massivamente Paralela.....72

4.3 - Afinamento.....73

4.4 - Identificação das Bordas do Objeto.....74
4.4.1 - Utilizando Arquiteturas MIMD.....75
4.4.2 - Utilizando Arquitetura SIMD.....86
4.5 - Classificação das Retas.....89

CAPÍTULO V : RESULTADOS EXPERIMENTAIS.....93
5.1 - As Imagens Utilizadas nos Experimentos.....93
5.2 - A Precisão do Algoritmo Serial.....94
5.3 - O Desempenho do Algoritmo Serial no Transputer.....97
5.4 - Observações sobre a "Paralelização" do Algoritmo.....100
5.5 - Utilizando Arquitetura MIMD Fracamente Acoplada.....101
5.5.1 - Detecção de Bordas.....102
5.5.2 - Identificação das Bordas do Objeto.....103
5.6 - Utilizando Arquitetura SIMD.....119
5.7 - Desempenho em Tempo-Real.....122

CAPÍTULO VI : CONCLUSÕES FINAIS.....124
6.1 - Sobre o Algoritmo Serial.....124
6.2 - Sobre a Implementação de Algoritmos Paralelos.....126
6.3 - Sobre as Arquiteturas Paralelas Utilizadas.....127
6.4 - Sobre as Soluções Paralelas Encontradas.....128
6.5 - Sobre os Resultados Experimentais Obtidos.....129
6.6 - Sobre as Aplicações e Extensões do Tarabalho.....130

APÊNDICE 1 : O Laplaciano de uma Gaussiana.....132
APÊNDICE 2 : A Minimização do Erro Quadrático Médio.....136
APÊNDICE 3 : O Algoritmo de Localização de Objetos no espaço 3D.....140
APÊNDICE 4 : Tabelas Completas dos Tempos de Execução.....153

REFERÊNCIAS BIBLIOGRÁFICAS.....157

CAPÍTULO I

INTRODUÇÃO

O estudo e a utilização da Visão Computacional tri-dimensional como ferramenta de auxílio às atividades ligadas à **Automação** - como montagem, inspeção e navegação automáticas - tem se caracterizado por uma atividade intensa, tanto teórica como experimental, conforme HORN (1986), MAHOWALD e MEAD (1991) e CHEN & TSAI (1991). Um dos processos básicos da Visão Computacional aplicada é o "reconhecimento e localização de objetos no espaço 3D". Este processo é responsável pelo fornecimento automático de informações úteis sobre uma cena do mundo real. Dentro do contexto de Visão Robótica, estas informações úteis podem ser, por exemplo, referências posicionais para que atuadores posssam interagir fisicamente com o ambiente.

O presente trabalho tem como objetivo desenvolver exatamente esse "sistema de reconhecimento" e "localização de objetos no espaço 3D" visando potenciais aplicações em ambientes industriais. Desse modo o conjunto de especificações a seguir foi definido para este sistema:

1) Reconhecer e Localizar Objetos no Espaço 3D.

Quando uma cena a ser processada contiver características bi-dimensionais (2D) - como ocorre no reconhecimento de caracteres escritos - ou quando as distâncias tri-dimensionais relativas entre os elementos da cena puder ser desprezada - como ocorre em sensoreamento remoto via satélites - as técnicas de Visão Computacional 2D são suficientes e devem ser utilizadas.

Já quando uma cena contiver objetos tri-dimensionais relativamente próximos - como ocorre quando se pega uma peça de uma esteira rolante - a natureza 3D dos objetos e da cena não pode ser ignorada. Desse modo, em adição às técnicas 2D, deve-se levar em conta no processamento, entre outras: as variações de iluminação (sombras, em particular) e as diversas possíveis posições do objeto em relação à câmera.

A idéia é reconhecer e localizar objetos poliédricos simples e possuidores de alguma simetria e depois estender o método para objetos mais complexos, como os encontrados em ambientes industriais ("workpieces").

Neste trabalho considera-se localizar um objeto (do tipo paralelpípedo) fornecer as coordenadas 3D dos vértices do objeto e os ângulos de orientação do mesmo em relação a um referencial situado no centro do plano imagem da câmera.

2) Utilizar Visão Monocular.

Neste trabalho, a Visão Monocular foi preferida à Visão Estéreo (MARR, 1982) por necessitar menos equipamentos e, portanto, ser mais barata e por demandar uma quantidade menor de processamento computacional. Como consequência, é necessário conhecer *a priori* algumas características do objeto que se quer reconhecer/localizar. Na abordagem monocular utilizam-se "marcas" (por exemplo, vértices, pontos de fuga, etc.) na cena para simplificar o problema e diminuir o tempo de computação. Essa simplificação é conseguida embutindo-se informações geométricas nas marcas, de modo que após a transformação perspectiva, estas auxiliem na resolução do problema 3D.

3) Execução Automática.

Há sistemas para localização de objetos que necessitam de pontos de calibração, conforme GONZALES e WINTZ (1982), TSAI e LENZ (1988) e BELLON (1990). A utilização de pontos de calibração, embora tenha comprovada precisão, necessita de um processamento muito demorado – muitas vezes exigindo até uma interação manual – desaconselhando sua utilização em ambientes industriais. Desse modo optou-se por desenvolver um método automático, baseado não em pontos de calibração, mas em *segmentos de retas*.

4) Desvio Médio de Posicionamento Relativo Menor que 5% e Desvio Médio de Orientação Menor que 2° .

O desvio de posicionamento é definido como o módulo da diferença entre as coordenadas 3D medidas manualmente (c_m) e as coordenadas 3D computadas (c_c). O desvio de posicionamento relativo (D_{pr}) é definido como a razão entre o desvio de posicionamento e as coordenadas 3D medidas manualmente, ou seja:

$$D_{pr} = \frac{| c_m - c_c |}{c_m} \cdot 100\ \%$$

É razoável supor que o desvio de posicionamento seja dependente de c_m, ou seja, quanto mais longe da câmera estiver o objeto, mais imprecisas serão as coordenadas calculadas. Desse modo, o desvio de posicionamento relativo procura mostrar a qualidade (precisão) das coordenadas computadas, independentemente de c_m.

O desvio de orientação é definido como o módulo da diferença entre os ângulos de orientação medidos manualmente e os ângulos de orientação computados. Este valor independe dos ângulos medidos manualmente e, portanto, costuma-se tomar apenas o desvio absoluto.

Para se validar o processo de localização de objetos no espaço, é necessário que este possua uma precisão maior que a dos atuadores. Segundo BRADY (1989), os valores acima citados são aceitáveis tanto para sua aplicação em veículos auto-guiados quanto em robôs manipuladores.

5) Tempo de resposta de 33ms.

Em um estudo detalhado do sistema de reconhecimento/localização de objetos no espaço observa-se que este possui estágios cujo tempo de processamento é alto e dependente da complexidade da cena. São os chamados "gargalos" do processo. Como os estágios mais lentos é que determinam o tempo de execução total do processo, a aceleração destes estágios torna-se prioritária.

Neste trabalho, o tempo de resposta do processo de localização de objetos no espaço é medido *após* a captura da imagem e vai até o fornecimento da localização do objeto no espaço 3D. Sistemas de

Tempo-Real são aqueles especificados para responder dentro de um tempo pré-especificado, definido pelo usuário ou por um sensor externo. Se o sistema de visão robótica tiver um tempo de resposta menor ou igual a (1/30)s ≅ 33ms, pode-se dizer que ele é executado em tempo-real.

Para se conseguir satisfazer esta especificação de desempenho é necessário fazer uso de Computadores Paralelos. Neste trabalho foram utilizados dois tipos de arquiteturas: Single-Instruction Multiple-Data (SIMD) e Multiple-Instruction Multiple-Data (MIMD).

Para que se pudesse realizar o desenvolvimento do sistema de reconhecimento e localização de objetos no espaço 3D que satisfizesse as 5 especificações acima citadas, o problema foi dividido em duas partes:

1) Desenvolvimento do algoritmo de reconhecimento e localização de objetos no espaço 3D em um ambiente sequencial visando, principalmente, validar o sistema segundo às especificações nº 1, 2, 3 e 4.

2) Desenvolvimento de algoritmos paralelos para os estágios mais demorados para diminuir seus tempos de execução com vistas a validar o sistema segundo a especificação nº 5.

O algoritmo de reconhecimento e localização de objetos no espaço 3D foi codificado em linguagem C e foi implementado em uma Workstation SUN. As imagens foram adquiridas através de uma placa digitalizadora TARGA-16, da Truevision Inc., compatível com o IBM-PC. A implementação paralela do processo foi desenvolvida em uma placa INMOS-B008 com 9 TRAMs (Transputer Modules) e na Connection Machine, da Thinking Machines Corp, utilizando as linguagens paralelas OCCAM, C/Paris, C-paralelo e C*.

Os objetivos deste trabalho podem ser resumidos da seguinte maneira:

OBJETIVO GERAL:

"Realizar o estudo do comportamento de um sistema paralelo de reconhecimento e localização automáticos de objetos no espaço 3D utilizando máquinas SIMD e MIMD fracamente acopladas e utilizando imagens reais".

OBJETIVOS ESPECÍFICOS:

a) Criar um sistema que permita localizar objetos no espaço 3D e que seja estensível para reconhecer objetos mais complexos.

b) Estudar aspectos de hardware e de software de máquinas paralelas MIMD e SIMD. Embora estas duas arquiteturas sejam multiprocessadoras, a implementação de algoritmos difere significativamente de uma para a outra.

c) Estudar e aplicar várias abordagens de "paralelização" de processos com vistas à implementação do sistema em diferentes arquiteturas multiprocessadoras. Estas abordagens incluem a definição de tarefas, seu escalonamento e balanceamento, configuração da arquitetura, etc.

d) Obter algumas soluções paralelas para a transformada de Hough pois esta é de fundamental importância para o sistema de reconhecimento e localização de objetos no espaço 3D. Estas soluções paralelas devem diminuir o tempo de execução do sistema, aumentando a possibilidade de sua aplicação em outros sistemas de Visão Computacional.

A apresentação deste trabalho está organizada da seguinte forma:

O capítulo 2 apresenta o desenvolvimento seguido para definir-se o processo de reconhecimento e localização de objetos no espaço 3D. São mostrados com detalhes os estágios que compõem o processo. Ao final são apresentados os resultados sobre a precisão da implementação.

O capítulo 3 apresenta considerações sobre a utilização de arquiteturas paralelas em Visão Computacional. São vistas em detalhes duas clases de arquiteturas: MIMD e SIMD. E são apresentadas aquelas em que o sistema paralelo de reconhecimento e localização de objetos no espaço 3D foi implementado. São apresentados ainda aspectos de software paralelo, delineando alguns métodos aplicados na paralelização do sistema descrito no capítulo 2.

O capítulo 4 apresenta o desenvolvimento do sistema paralelo proposto, bem como os passos seguidos para se chegar a ele. Mostra-se como o paralelismo pôde ser aplicado a cada fase do algoritmo. Destacam-se os algoritmos propostos para a implementação paralela da Transformada de Hough.

O capítulo 5 apresenta os resultados experimentais da paralelização do algoritmo de reconhecimento e localização de objetos no espaço 3D.

O capítulo 6 apresenta as conclusões e faz considerações finais sobre o trabalho e perspectivas para a continuação do mesmo.

O apêndice 1 apresenta a dedução matemática do operador de detecção de bordas de Marr (1ª fase do algoritmo). O apêndice 2 mostra o desenvolvimento matemático do algoritmo do ajuste de curvas (6ª fase do algoritmo). O apêndice 3 mostra a dedução matemática completa do algoritmo de calibração (7ª fase do algoritmo). E, finalmente, o apêndice 4 apresenta as tabelas de tempos de execução das várias soluções paralelas desenvolvidas e apresentadas nos capítulos 4 e 5.

CAPÍTULO II

O ALGORITMO SERIAL

Este capítulo apresenta o algoritmo de localização de objetos no espaço 3D. Procura-se fazer uma análise dos vários estágios do processo a fim de caracterizar sua sequência, estabelecer relações entre os estágios e verificar o fluxo de dados entre eles. Esta análise será utilizada no capítulo 4 como ponto de partida para a estratégia de abordagem do processo de reconhecimento e localização de objetos no espaço 3D através de processamento paralelo.

As seções estão ordenadas de forma a aumentar a abstração das técnicas de Visão Computacional utilizadas, começando com representações e algoritmos em nível baixo e caminhando por níveis intermediários até níveis mais altos.

2.1 - Reconhecimento e Localização de Objetos no Espaço 3D

O problema de reconhecimento e localização de objetos no espaço 3D pode ser ilustrado conforme mostra a figura 2.1. Uma cena 3D é capturada por uma câmera. Esta cena é processada por um computador para extrair alguns atributos, como bordas, vértices, pontos de fuga, etc. Este conjunto de atributos forma uma Representação 2D da cena. Por outro lado, objetos podem ser representados por modelos abstratos, como os utilizados em Computação Gráfica: um poliedro no espaço 3D, por exemplo, pode ser modelado por um conjunto das coordenadas de seus vértices ou pelo conjunto das equações paramétricas de suas bordas. Aplicando-se transformações de rotação, translação, escalamento e projeção perspectiva gera-se uma vista 2D do objeto. Esta vista 2D é utilizada para fazer o casamento ("matching") entre o modelo do objeto e a Representação 2D extraída da imagem real. O casamento divide-se em **topológico**, responsável pelo reconhecimento do objeto e o **geométrico**, responsável pela calibração da câmera e localização do objeto.

Desse modo, dada uma imagem real de um objeto e dados alguns modelos de objetos, procura-se o modelo que, após algumas transformações de coordenadas (cujos parâmetros necessitam ser calibrados), melhor se ajuste à imagem real. Para objetos poliédricos, utilizam-se os vértices dos objetos para se fazer este casamento.

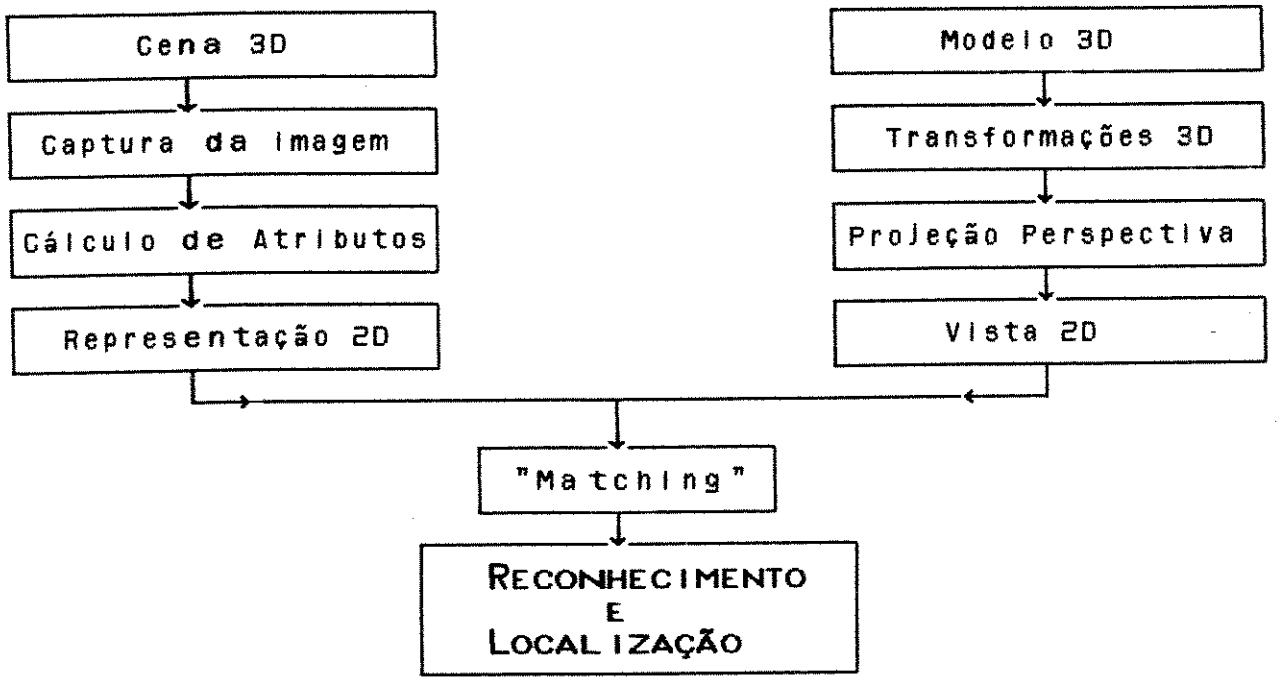


Figura 2.1. A idéia básica do Processo de Reconhecimento e Localização de Objetos no Espaço 3D.

A escolha do tipo de objeto 3D a ser detectado tem influência direta no tempo de execução do processo. Considera-se que um objeto 3D tem pouca complexidade quando possui poucas bordas, muitas simetrias geométricas e texturas simples. Nesta classe podem ser enquadrados objetos como esferas, pirâmides, cubos, paralelepípedos, tetraedros, etc. Objetos mais complexos possuem bordas de formatos irregulares e texturas variadas.

O algoritmo serial, que será descrito na próxima seção, possibilita a localização automática de objetos do tipo paralelepípedo no espaço 3D. Este algoritmo serial tem como entrada uma imagem em níveis de cinza de um objeto do tipo paralelepípedo e tem, como saída, as coordenadas dos vértices do objeto e os ângulos de orientação do mesmo em relação a um referencial (ver seção 2.2.7) colocado no centro do plano imagem na câmera.

O paralelepípedo pode estar em qualquer posição no campo visual da câmera. Portanto, há três casos possíveis para o posicionamento do paralelepípedo, conforme a figura 2.2: a) o caso em que apenas 4 arestas e 4 vértices são visíveis, implicando em uma observação frontal; b) o caso de 7 arestas e 6 vértices visíveis, implicando em uma observação "lateral"; e c) o caso de 9 arestas e 7 vértices visíveis.

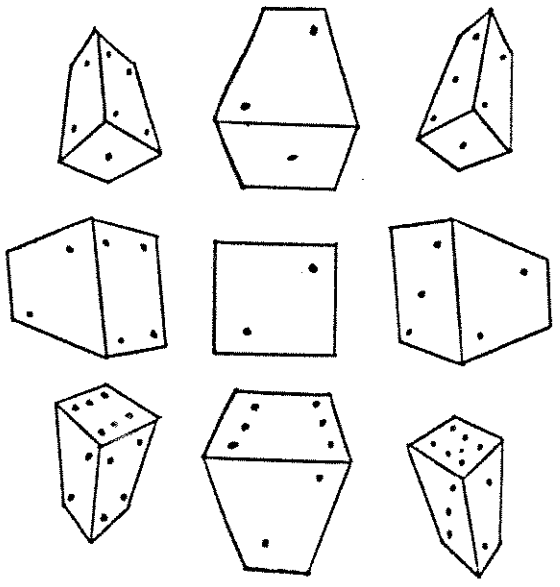


Figura 2.2. As orientações possíveis de um cubo no espaço 3D.

Para a implementação do algoritmo, os três casos citados têm complexidade crescente de (a) para (c).Os casos (a) e (b) podem ser considerados casos particulares do caso (c). O algoritmo descrito neste capítulo resolve o caso mais geral: o caso (c).

2.2 - As Fases do Algoritmo

O algoritmo implementado consiste de 7 fases e seu fluxograma é apresentado na figura 2.3. Serão descritas agora as 7 fases do algoritmo implementado, seguido de alguns resultados de sua aplicação e comentários.

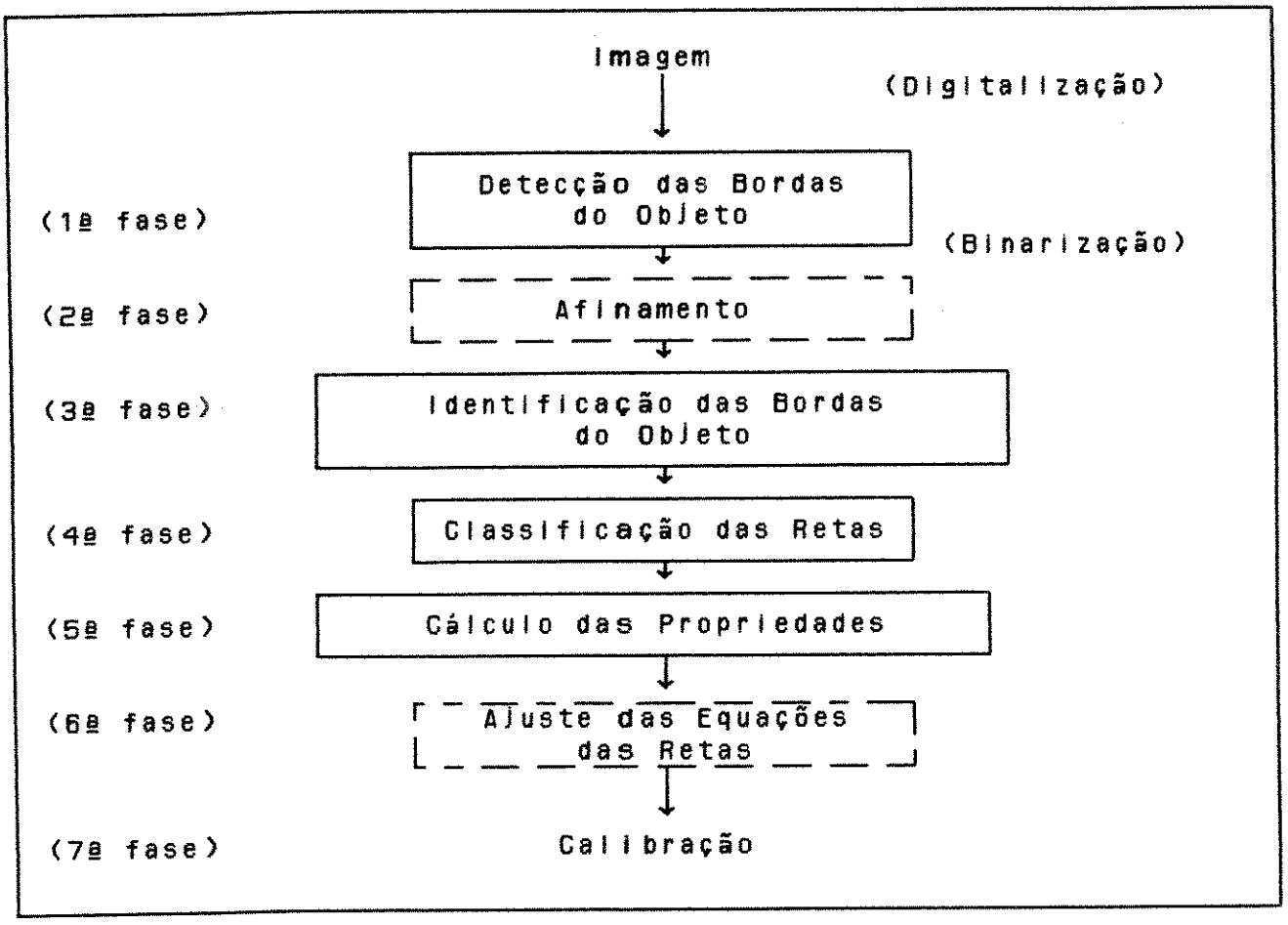


Figura 2.3. O fluxograma do algoritmo desenvolvido. As fases tracejadas são opcionais.

2.2.1 - (1ª fase) - Detecção das Bordas

A primeira fase do algoritmo é a fase de detecção das bordas do objeto. O sucesso dos próximos estágios depende muito desta fase. São citados na literatura - por exemplo, em BALLARD (1982) - diversos métodos para a detecção de bordas e o tipo mais comumente encontrado é o dos detectores "matemáticos". Detectores de bordas "matemáticos" são aqueles que tratam a **imagem** como uma **função 2D** e utilizam operadores matemáticos, que são **convoluídos** com a imagem, para revelarem as bordas presentes na imagem.

A operação de convolução discreta, denotada por "*", é assim definida: Dada uma função discreta - chamada de imagem - I(m,n) e uma função discreta - chamada de filtro ou máscara - M(m,n), a imagem convoluída (ou filtrada) G(m,n) é obtida através da operação:

$$G(m,n) = \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} I(p,q).M(m-p,n-q)$$

Onde p e q são variáveis auxiliares e m=0,1,2,...,M-1 e n=0,1,2,...,N-1. Este tipo de método para a detecção de bordas é de simples implementação e alguns deles já foram integrados em CIs "full-custom", conforme ARAMBEPOLA et al. (1988) e KANOPOULOS et al. (1988).

Há, basicamente, dois tipos de operadores matemáticos para a detecção de bordas:

(1) operadores **diferenciais direcionais** ou do tipo **gradiente**: Operadores deste tipo respondem com um pico de intensidade na localização da borda e fornecem uma estimativa da orientação da mesma. Matematicamente, o vetor gradiente calculado aponta na direção de máximo crescimento da função imagem (caracterizando, desse modo, uma borda na mesma) e seu comprimento é a taxa de crescimento nesta direção. São exemplos desse tipo de operador: o Gradiente propriamente dito e o operador de Sobel.

(2) operadores **diferenciais invariantes à rotação**: São aqueles que implementam, digitalmente, os operadores do tipo Laplaciano (linear) e o operador da 2ª derivada direcional na direção do gradiente (não-linear). Operadores deste tipo respondem com um cruzamento de zero (ou mudança de sinal) na localização da borda. Matematicamente, considera-se como sendo bordas, os pontos de inflexão da função imagem. São exemplos do primeiro o operador Laplaciano e o operador de Marr e do segundo, o operador de CANNY (1986).

Escolheu-se operadores dos dois tipos para se fazer uma comparação. Os operadores para detecção de bordas escolhidos foram o operador de Sobel, o operador Laplaciano e o operador de Marr e serão mostrados a seguir juntamente com o operador gradiente.

Operador Gradiente

O operador Gradiente, definido por:

$$\nabla I(x,y) = \frac{\partial I(x,y)}{\partial x}.\vec{i} + \frac{\partial I(x,y)}{\partial y}.\vec{j}$$

pode ser aproximado discretamente da maneira que se segue, onde as variáveis x e y foram alteradas para m e n para indicar que são variáveis discretas:

$$\nabla I(m,n) \cong [\, I(m+1,n)-I(m-1,n)\,].\vec{i} + [\, I(m,n+1)-I(m,n-1)\,].\vec{j}$$

Este operador pode ser implementado pelas 2 máscaras da figura 2.4a, onde M_y indica diferenciação sobre o eixo $\vec{j}$.

Operador de Sobel

O operador gradiente de Sobel, ou simplesmente operador de Sobel, faz uma extensão do operador gradiente para ser implementado por máscaras 3x3. Este operador pode ser descrito da seguinte maneira, onde m e n indicam que são variáveis discretas.

$$\begin{aligned}\nabla I(m,n) \cong\ & 1/4.\{\, [\, I(m+1,n+1)+2.I(m,n+1)+I(m-1,n+1)\,] - \\ & [\, I(m+1,n-1)+2.I(m,n-1)+I(m-1,n-1)\,]\,\}.\vec{i} + \\ & 1/4.\{\, [\, I(m+1,n+1)+2.I(m+1,n)+I(m+1,n-1)\,] - \\ & [\, I(m-1,n+1)+2.I(m-1,n)+I(m-1,n-1)\,]\,\}.\vec{j}\end{aligned}$$

que é implementado pelas máscaras da figura 2.4b.

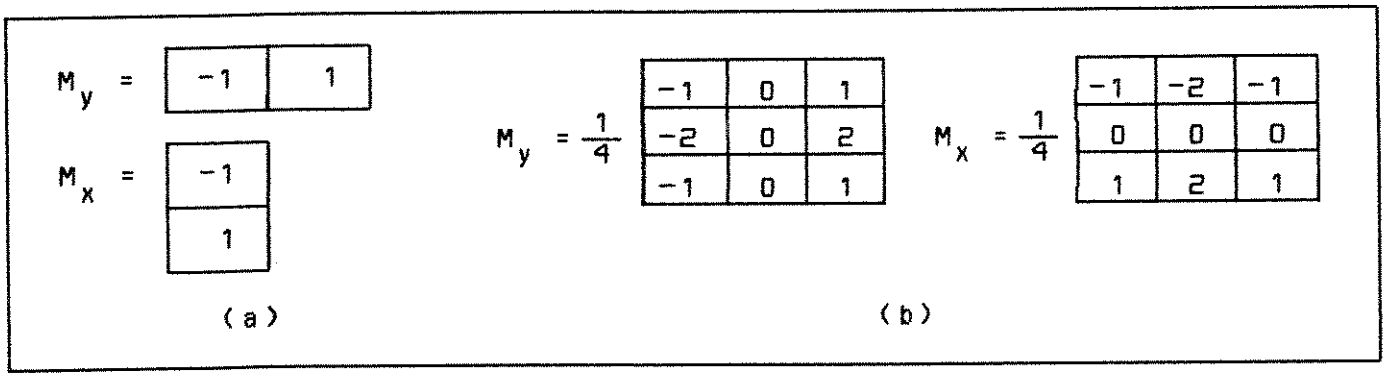


Figura 2.4. Máscaras do operador Gradiente e do operador de Sobel.

Assim como o Gradiente, o operador de Sobel possui módulo e direção. O módulo S e a direção Φ do operador de Sobel são definidos, respectivamente, por:

$$S = \sqrt{M_x^2 + M_y^2} \qquad \Phi = \text{atan}(M_y / M_x)$$

Operador Laplaciano

Dada uma imagem I(x,y), o Laplaciano da imagem é dado por:

$$\nabla^2 I(x,y) = \frac{\partial^2 I(x,y)}{\partial x^2} + \frac{\partial^2 I(x,y)}{\partial y^2}$$

Sua aproximação discreta é feita da seguinte maneira, onde as variáveis x e y foram alteradas para m e n para indicar que são variáveis discretas e onde ε é o espaçamento entre o centro de dois pixels.

$$\nabla^2 I(m,n) \cong \frac{4}{\varepsilon^2} \left[\frac{1}{4} .[\ I(m-1,n)+I(m,n-1)+I(m+1,n)+I(m,n+1)\ ]-I(m,n) \right]$$

que é implementado pela máscara da figura 2.5a.

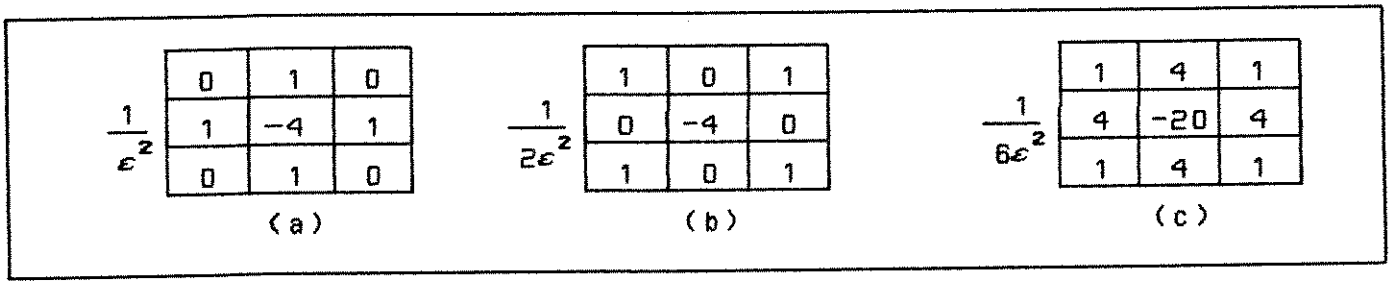


Figura 2.5. Máscaras do operador Laplaciano. HORN (1986).

Este operador apresenta resposta nula para regiões de nível de cinza (brilho) constante. O Laplaciano discreto pode ser implementado pela máscara da figura 2.5a e convoluído com a imagem I(x,y). Todavia, o operador não é simétrico em relação aos seus oito vizinhos. Se a

máscara da figura 2.5a for rotacionada de 45°, a máscara se torna a da figura 2.5b. Uma combinação linear destas duas máscaras (2/3 da 1ª somado a 1/3 da 2ª) fornece uma boa aproximação - ver HORN (1986) - para o Laplaciano e é mostrada na figura 2.5c.

Operador de Marr

De acordo com a teoria de detecção de bordas de MARR (1982), mudanças bruscas de intensidade (bordas) são detectadas em uma imagem I(x,y) encontrando-se o Laplaciano da convolução da imagem I(x,y) com uma Gaussiana G(x,y) e encontrando-se os pontos onde há troca de sinal na imagem resultante. Representa-se este fato, matematicamente, por:

$$\nabla^2 [G(x,y) * I(x,y)]$$

onde * é a operação de convolução. A Gaussiana é chamada de filtro regularizador e tem, basicamente, o objetivo de tornar a função I(x,y) suave e derivável. Como o Laplaciano é um operador linear, pode-se escrever:

$$\nabla^2 G(x,y) * I(x,y)$$

A função $\nabla^2 G(x,y)$, também chamada de LoG, é apresentada na figura 2.6 para o caso de coordenadas cilíndricas, ou seja, faz-se $r = \sqrt{x^2 + y^2}$. Esta função é deduzida inteiramente no apêndice 1 em coordenadas retangulares e também é apresentada graficamente.

$$\nabla^2 G(r) = \frac{-1}{2.\pi.\sigma^4} \cdot \left[2 - \frac{r^2}{\sigma^2} \right] . e^{\left[\frac{-r^2}{2.\sigma^2} \right]}$$

Figura 2.6. O operador $\nabla^2 G(x,y)$ em coordenadas cilíndricas. σ é o desvio padrão do filtro, conforme apêndice 1.

A sensibilidade deste filtro às bordas está associada à largura da região central da função LoG. Esta largura central é chamada w e é está relacionada ao desvio padrão σ, como mostra o apêndice 1. Quanto menor o valor do desvio padrão σ, mais sensível o filtro e, como consequência, mais detalhes são vistos e pior a tolerância ao ruído. MARR (1982) propõe que, para obter-se sucesso com esse operador, deve-se fazer um AND lógico entre as diferentes imagens resultantes de convoluções com várias LoG de w diferentes.

Um problema com essa proposta é a necessidade de muita memória para armazenamento de imagens intermediárias. Um outro problema é que geralmente são necessárias máscaras muito grandes, tipicamente de 10x10 a 30x30, tornando o cálculo muito lento.

A figura 2.7 apresenta a resposta limiarizada dos três operadores escolhidos a uma imagem típica da aplicação. A máscara do operador Laplaciano utilizada foi a da figura 2.5c, as máscaras do operador de Sobel utilizadas foram a da figura 2.4b e as duas máscaras do operador de Marr utilizadas possuíam σ = 1,17 e σ 2,35 e tamanhos 10x10 e 20x20.

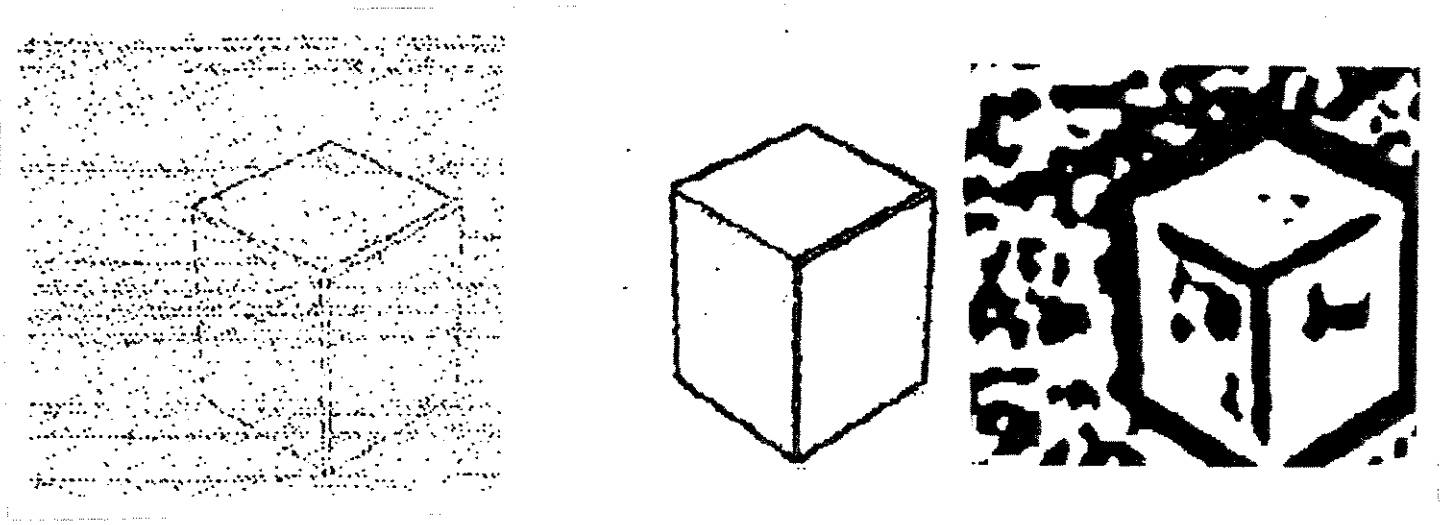


Figura 2.7. Comparação de três operadores de detecção de bordas. Os tempos de execução foram: Laplaciano → 0,34 s ; Sobel → 0,49 s ; Marr → 20,4 s.

Através de inspeção visual da figura 2.7, percebe-se que o operador de Sobel é mais imune ao ruído do que o operador Laplaciano. O operador de Marr é o mais demorado de todos os três. Desse modo, optou-se por utilizar o operador de Sobel na complementação do trabalho.

Após a aplicação do operador de Sobel, executa-se uma "limiarização com histerese" para tornar a imagem binária: esta limiarização com histerese é feita estipulando-se - empiricamente, de acordo com a iluminação do ambiente; ou computando-se matematicamente, como faz CANNY (1986) - dois valores, chamados de "limiar_alto" e "limiar_baixo". Os limiares são aplicados do seguinte modo:

1º) Eliminam-se pixels com magnitude abaixo do "limiar_baixo" e identifica-se os demais pixels acima do "limiar_alto" como sendo pixels de borda.

2º) Todos os pixels restantes, ou seja, com valores entre "limiar_baixo" e "limiar_alto", tornam-se bordas se puderem ser 8-conectados a um pixel já identificado.

O resultado da limiarização com histerese é uma imagem com a resolução 256x256x1, ou seja, é uma imagem binária, onde os valores `1' representam as bordas do objeto e os valores `0' representam o fundo ou as faces do paralelepípedo. Somente os valores `1' é que são de interesse. Após este estágio, passa-se para a etapa do afinamento da imagem.

2.2.2 - (2ª fase) - Afinamento

O afinamento tem a finalidade de extrair a informação mínima necessária de uma borda e obter o chamado esqueleto da borda. O afinamento retira todos os pontos redundantes preservando a estrutura básica e as características da imagem.

A imagem binária digital é definida por uma matriz I(x,y), onde cada pixel de I(i,j) vale '1' ou '0'. O objeto a ser trabalhado consiste dos pixels que possuem valor 1. Os vizinhos do ponto (i,j) são nomeados segundo a figura 2.8.

P_9 (i-1,j-1)	P_2 (i-1,j)	P_3 (i-1,j+1)
P_8 (i ,j-1)	P_1 (i ,j)	P_4 (i ,j+1)
P_7 (i+1,j-1)	P_6 (i+1,j)	P_5 (i+1,j+1)

Figura 2.8. O mapa de vizinhos.

O algoritmo implementado baseou-se nos trabalhos de ZHANG e SUEN (1984) e de LÜ e WANG (1985). Cada iteração é dividida em duas sub-iterações. Na primeira delas, o ponto de contorno P_1 é apagado da imagem, se esse satisfizer as seguintes condições:

(a) $3 \leq B(P_1) \leq 6$

(b) $A(P_1) = 1$

(c) $P_2 * P_4 * P_6 = 0$

(d) $P_4 * P_6 * P_8 = 0$

onde $A(P_1)$ é o número de sequências '01' no conjunto ordenado P_1, $P_2,\ldots,P_8,P_9$ que são os oito vizinhos de P_1 (ver figura 2.8). E $B(P_1)$ é o número de vizinhos não nulos de P_1, ou seja $B(P_1) = P_2+P_3+\ldots+P_8+P_9$.

Na segunda sub-iteração, apenas as condições (c) e (d) são alteradas para:

(c') $P_2 * P_4 * P_8 = 0$

(d') $P_2 * P_6 * P_8 = 0$

e as demais permanecem as mesmas.

O fluxograma desse algoritmo pode ser visto na figura 2.9. Neste fluxograma, M é o conjunto dos pontos que satisfizeram as condições de apagamento em alguma das 2 iterações, como já foi explicado. A condição C indica que houve um apagamento e, portanto, o algoritmo deve fazer mais uma iteração. E, finalmente, a operação "I=I-M" indica o apagamento do ponto não pertencente ao esqueleto.

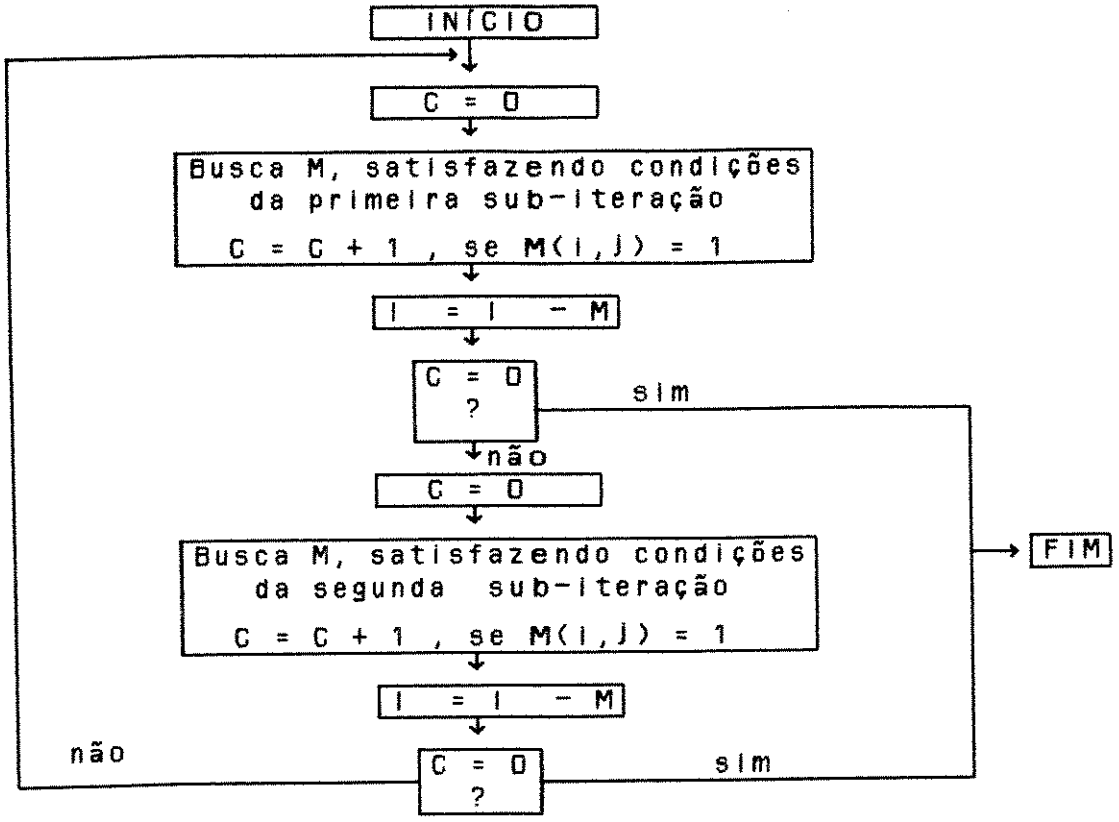


Figura 2.9. Fluxograma do algoritmo de afinamento

Este algoritmo de afinamento é iterativo e, portanto, tem um tempo de execução dependente dos dados de entrada. Um resultado típico da aplicação deste algoritmo é mostrado na figura 2.10.

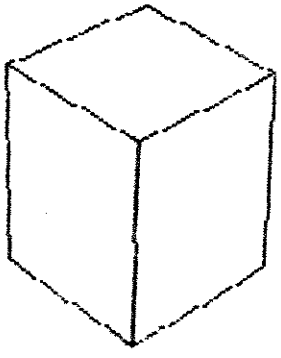


Figura 2.10. A resposta do algoritmo de afinamento.

Dependendo da aplicação este algoritmo de afinamento não funciona: por exemplo, para efeito de codificação de imagens, este algoritmo não gera um esqueleto apropriado porque cria alguns artefatos (ruídos) na imagem resultante. Todavia, para os propósitos deste trabalho, seu tempo de resposta é bastante satisfatório e as fases posteriores minimizam os efeitos dos problemas criados pelos artefatos indesejáveis sem perdas qualitativas na resposta final do algoritmo. Conforme o algoritmo utilizado na fase de identificação de bordas (3ª fase), esta fase de afinamento pode ser ou não necessária. A transformada de Hough clássica é suficientemente imune a segmentos de retas de pequenas dimensões e não exige um esqueleto como entrada, conforme será visto na seção 2.2.3.

2.2.3 - (3ª fase) - Identificação das Bordas do Objeto

Após o tratamento de baixo nível, é necessário identificar-se as retas suportes das arestas do objeto. Para se fazer isto, utiliza-se, basicamente, a Transformada de Hough. Esta foi introduzida por P.V.C. Hough em 1962 (conforme BALLARD, 1982) como um método para detecção de retas em imagens. A técnica evoluiu e hoje já existe uma variedade de transformadas de Hough. Uma das evoluções foi sua generalização para curvas paramétricas quaisquer, feita por BALLARD (1981).

Dois algoritmos da transformada de Hough foram estudados e implementados. O primeiro deles é a transformada clássica de Hough, o segundo é uma versão mais rápida, que utiliza informação direcional das bordas. Primeiramente será definida a transformada de Hough e serão mostrados aspectos de sua implementação discreta que influenciam em sua precisão. Em seguida será mostrada a utilização da informação direcional para diminuir o tempo de processamento da transformada.

Uma reta no plano imagem (2D) pode ser definida por dois parâmetros. Neste trabalho adota-se a parametrização θ-ρ, conforme a figura 2.11. O parâmetro ρ é convencionado como a distância da origem à reta. O parâmetro θ é convencionado como o ângulo entre a normal à reta que passa pela origem e o eixo X. Segundo estas convenções adotadas todos os pontos (x,y) que pertencem à reta satisfazem à equação 2.1.

$$x.\cos\theta + y.\text{sen}\theta = \rho \qquad \text{(equação 2.1)}$$

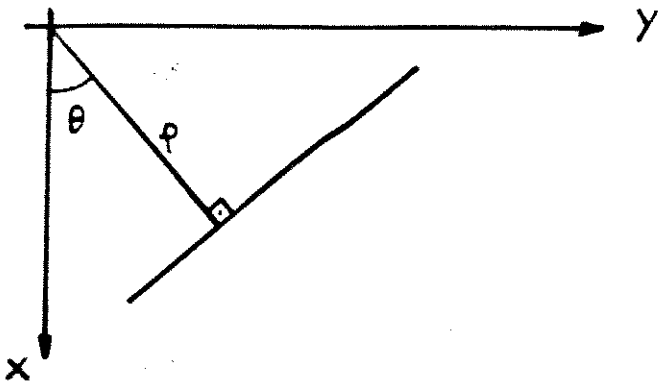


Figura 2.11. Representação Normal da reta, seguindo a orientação de eixos convencionada para o plano imagem.

A idéia da transformada de Hough é a seguinte: seja uma reta (ou um segmento de reta) discreta no plano imagem constituída por n pontos colineares (x_i, y_i). Existe um número infinito de retas que podem passar através de cada ponto (x_i, y_i) no plano imagem. Se cada ponto da reta for transformado pela equação 2.1 em uma curva senoidal no plano θ-ρ, então o conjunto de curvas do plano θ-ρ, que corresponde ao conjunto de pontos (colineares) pertencentes à reta, deverá possuir um ponto de intersecção no plano θ-ρ. Este ponto, representado pelos parâmetros ρ_p e θ_p, determina a equação da reta que é definida pela equação $\rho_p = x_i.\cos\theta_p + y_i.\text{sen}\theta_p$.

O plano θ-ρ é o conjunto de todas as células acumuladoras e também é chamado de matriz acumuladora ou matriz de votos ou espaço Hough. Este possui o nome de acumulador, pois cada ponto de uma dada reta r - parametrizado por ρ_r e θ_r - contribui com um voto no acumulador (θ_r, ρ_r) do espaço Hough. Inicialmente, toda a matriz de votos é inicializada com zeros. Ao final do cálculo da Transformada, o Espaço Hough possuirá picos máximos de votos nas coordenadas (ρ_r, θ_r) que correspondem às retas no plano imagem. O valor do voto é, geralmente, unitário.

Basicamente, para se fazer a transformada de Hough de um ponto de borda do plano imagem, deve-se fazer:

1º) Discretizar θ em k valores;

2º) Para cada ângulo θ_k calcular ρ (a partir da equação 2.1); e incrementar uma célula acumuladora (θ,ρ), correspondente ao ângulo θ_k utilizado e o ρ calculado;

3º) Fazer uma busca pelas células acumuladoras com maior número de votos, que correspondem aos parâmetros ρ_p e θ_p da reta procurada.

O espaço Hough (HS) tem dimensão $N_\theta \times N_\rho$, onde N_θ é o número (inteiro) de ângulos em que θ é discretizado e N_ρ é o número (inteiro) de distâncias em que ρ é discretizado. O valor mínimo para $\Delta\rho$ deve ser tal que o respectivo acumulador comporte o maior segmento de reta possível na imagem. O parâmetro ρ pode ser negativo e, portanto, é restrito a valores dentro do intervalo $[-L_{max}, L_{max}]$. E o parâmetro $\Delta\theta$ deve variar dentro do intervalo $[0,\pi]$ para não haver redundância de dados. Desse modo:

$$\Delta\rho \geq \frac{2 \cdot L_{max}}{N_\rho} \qquad \text{e} \qquad \Delta\theta = \frac{\pi}{N_\theta} \qquad \text{(equações 2.2)}$$

Onde

L_{max} = comprimento da maior reta possível na imagem.
N_ρ = número de células acumuladoras em ρ.
N_θ = número de células acumuladoras em θ.
$\Delta\theta$ = passo da discretização em θ.
$\Delta\rho$ = passo da discretização em ρ.

Para se fazer as acumulações na matriz de votos deve-se saber calcular os índices θ_i e ρ_j (inteiros) do acumulador $HS[\theta_i][\rho_j]$, que se pretende incrementar, a partir do ângulo θ_k utilizado e do valor de ρ calculado. Levando-se em conta o intervalo de valores possíveis para θ e ρ, obtém-se as equações 2.3.

$$\theta_i = \left\lfloor \frac{\theta_k}{\Delta\theta} \right\rfloor \qquad \rho_j = \left\lfloor \frac{\rho}{\Delta\rho} + \frac{N_\rho}{2} \right\rfloor \qquad \text{(equações 2.3)}$$

Onde o operador "$\lfloor\ \rfloor$" trunca o valor real para o maior inteiro menor que o valor real calculado. Estes valores inteiros são utilizados como índices da matriz de votos. Note-se que ρ pode ser negativo e, portanto, é necessário fazer uma translação de $N_\rho/2$ no cálculo de ρ_j. A resolução em θ afeta linearmente a velocidade do algoritmo. A literatura relata experimentos com as mais diversas resoluções em θ para o espaço Hough, dependendo do tipo de aplicação. Em experimentos com localização de objetos, onde é necessária uma resolução maior, a resolução utilizada varia em torno de um grau ou menos, conforme LI (1989).

Retas em imagens digitais são encontradas no espaço Hough através de uma busca das células com maior número de votos. Um segmento de reta ideal deveria ser mapeado em apenas uma célula do espaço Hough. Porém, devido a alguns fatores - como a quantização da imagem, a quantização do espaço Hough, a largura de retas que não foram afinadas, etc. - um segmento de reta real é mapeado em uma nuvem de pontos cujo pico tem a maior probabilidade de representar o segmento de reta original.

Uma propriedade da Transformada de Hough é que esta possui uma relação de reflexão nas bordas direita e esquerda do Espaço Hough. Esta propriedade advém da maneira pela qual θ e ρ trocam de sinal em $\pm 90^\circ$. Uma outra propriedade é que a Transformada de Hough é relativamente imune a falhas no traçado das retas e à presença de ruído. Esta característica faz com que a transformada de Hough seja largamente utilizada, conforme ILLINGWORTH e KITTLER (1988).

Pode-se também interpretar a transformada de Hough de um outro ponto de vista, conforme é mostrado na figura 2.12. Cada célula do espaço Hough (θ_k, ρ_l) corresponde no plano imagem a uma janela de direção θ_k com comprimento infinito e largura $\Delta\rho$. Desse modo, a transformada de Hough de um segmento de reta pode ser considerada como a **projeção** desse segmento sobre as janelas acima definidas. Aquela janela que contiver mais pontos projetados corresponde ao pico da nuvem de pontos no espaço Hough.

Uma vez fixado o tamanho da imagem, pode-se melhorar a precisão da transformada de Hough apenas variando-se a quantização do espaço Hough. Seguindo-se esta segunda interpretação da transformada de

Hough, considere-se um segmento de reta com largura suficientemente pequena. No pior caso, esse segmento não tem uma direção exatamente igual a um dos valores θ_k. Portanto, o segmento de reta cruza várias janelas (paralelas) ρ_L com uma mesma direção θ_k. O pico no espaço Hough está, desse modo, espalhado por uma nuvem de células na direção θ_k correspondendo a essa janela. Ou, de outra maneira, uma célula pertencendo ao pico corresponde a apenas uma parte do segmento de reta. Por exemplo, na figura 2.12, o segmento de reta de comprimento L está dividido em três células: (θ_k, ρ_{n-1}), (θ_k, ρ_n) e (θ_k, ρ_{n+1}), sendo que na segunda célula (θ_k, ρ_n), o número de pontos projetados é maior.

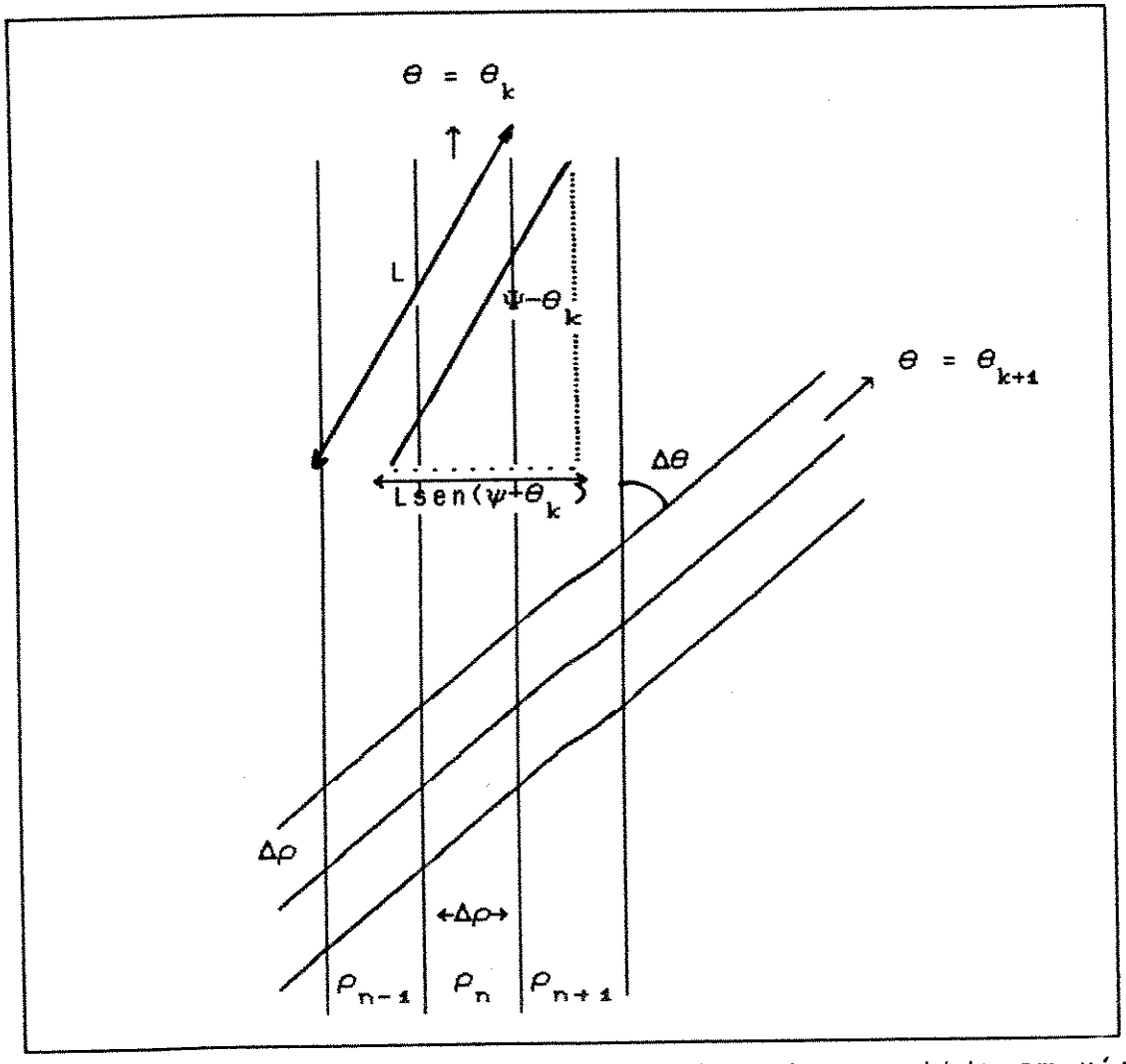


Figura 2.12. Um segmento de reta pode estar contido em várias "janelas", corespondendo a células (ρ,θ). VAN VEEN e GROEN (1981).

Prosseguindo nesta linha de interpretação, seja o segmento de reta com comprimento L e direção ψ, como mostra a figura 2.12. Para ψ pertencer à janela θ_k o segmento deve pertencer ao intervalo:

$$| \psi - \theta_k | \leq \frac{1}{2} \Delta\theta$$

Como pode ser observado na figura 2.12, o máximo número de células n_ρ na direção θ_k é:

Se $L.sen(\psi-\theta) < \Delta\rho$ → $n_\rho = 1$

Se $L.sen(\psi-\theta) = \Delta\rho$ → $n_\rho = 2$

Se $L.sen(\psi-\theta) > \Delta\rho$ → $n_\rho = \left\lfloor \frac{L.sen(\Delta\theta/2)}{\Delta\rho} \right\rfloor + 2$

(equação 2.4)

Para minimizar a extensão do pico em uma dada direção θ_k, pode-se exigir que, para um dado $\Delta\theta$ e L_{max}, $\Delta\rho$ seja escolhido tal que *o menor segmento de reta seja projetada dentro de apenas duas janelas (do pico) de* θ_k (vizinhas), no pior caso, e com uma direção mais próxima possível da do segmento de reta. Pode-se obter esta relação a partir da equação 2.4, fazendo-se :

$$\frac{L_{max}.sen(\Delta\theta/2)}{\Delta\rho} \leq 1$$

e, resolvendo para $\Delta\rho$:

$$\boxed{\Delta\rho \geq L_{max}.sen(\Delta\theta/2)} \qquad \text{(equação 2.5)}$$

Substituindo $\Delta\theta$ e $\Delta\rho$ das equações 2.2 na equação 2.5 pode-se obter uma equação que relaciona N_θ e N_ρ, segundo a minimização explicitada pela equação 2.5, que é muito útil na escolha da resolução do espaço Hough. Obtém-se desse modo a equação 2.6 que é apresentada graficamente na figura 2.13.

$$N_\rho = \left\lfloor \frac{2}{\operatorname{sen}\left(\frac{\pi}{2N_\theta}\right)} \right\rfloor \qquad \text{(equação 2.6)}$$

Segundo a equação 2.6, para uma resolução em θ de 256, deve-se ter uma resolução em ρ de 326 para que, teoricamente, o pico seja compartilhado com, no máximo, duas células, facilitando o procedimento de busca (seção 2.2.4).

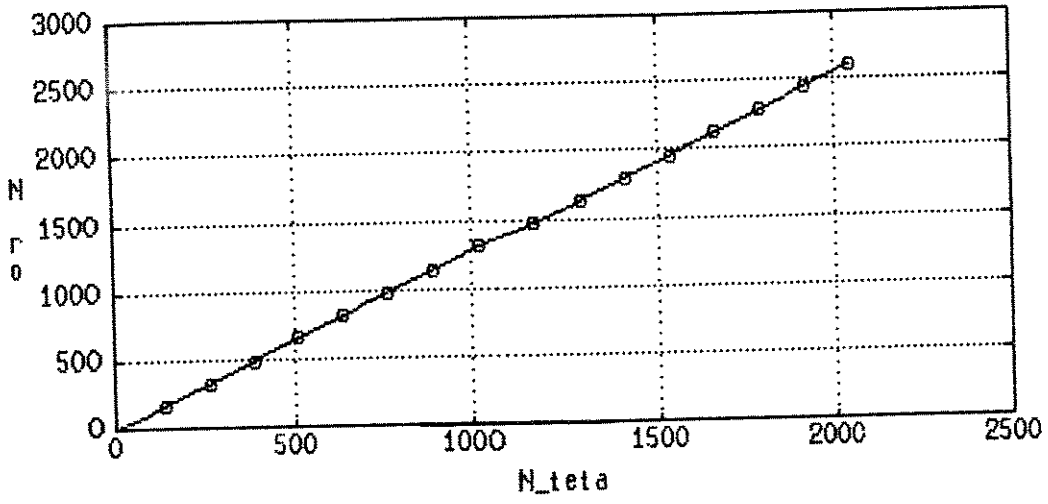


Figura 2.13. N_ρ em função de N_θ, a partir da eq. 2.6.

Utilização de Informação Direcional na Transformada de Hough

A utilização de informação direcional a respeito das bordas de uma curva pode ser de grande ajuda no cálculo da Transformada de Hough. O algoritmo original de Hough não utilizava esse tipo de informação. A informação direcional pode ser obtida da imagem original (em níveis de cinza) através de operadores diferenciais direcionais ou do tipo gradiente, conforme apresentado na seção 2.2.1.

Considerando a equação genérica de curvas analíticas da forma $f(\mathbf{z},\mathbf{a})=0$, onde $\mathbf{z}$ é um ponto (x,y) na imagem e $\mathbf{a}$ é um vetor de parâmetros da curva que se quer detectar é possível generalizar a

transformada de Hough para curvas analíticas quaisquer (BALLARD, 1981). No caso da transformada clássica de Hough para detectar retas o vetor **a** equivale a (θ,ρ). E, portanto,

$$f(\mathbf{z},\mathbf{a}) = f(x,y,\theta,\rho) = x.\cos\theta + y.\mathrm{sen}\theta - \rho = 0$$

Quando se utiliza a informação fornecida pelos operadores direcionais para detecção de bordas cria-se uma "imagem gradiente" onde cada pixel representa um vetor com direção e intensidade. A figura 2.7a mostra apenas a informação de intensidade. A "imagem gradiente" é modelada matematicamente por

$$\frac{df(\mathbf{z},\mathbf{a})}{dx} = 0$$

Formalmente esta equação introduz o termo dy/dx que, como se sabe, *está sempre em quadratura com o gradiente*, ou seja,

$$\frac{dy}{dx} = \mathrm{tg}[\Phi(x,y) \pm \pi/2\ ]$$

Onde $\Phi(x,y)$ é a direção do gradiente. Diferenciando a equação 2.1 em relação a x, obtém-se:

$$f(\mathbf{z},\mathbf{a}) = x.\cos\theta + y.\mathrm{sen}\theta - \rho = 0$$

$$\frac{df(x,y,\theta,\rho)}{dx} = \cos\theta + \frac{dy}{dx}.\mathrm{sen}\theta = 0$$

$$\frac{dy}{dx} = -\,\mathrm{cotg}(\theta) \qquad (\theta \neq k\pi,\ k = 1,2,\ldots)$$

mas,

$$\frac{dy}{dx} = \mathrm{tg}(\alpha)$$

e, portanto,

$$\mathrm{tg}(\alpha).\mathrm{tg}(\theta) = -1 \qquad (\theta \neq k\pi,\ k = 1,2,\ldots)$$

indicando que θ e α estão em quadratura. Este fato também pode ser observado da figura 2.14, que mostra uma função degrau ideal 2D (ou a borda de um objeto homogeneamente iluminado). A inclinação da reta que delimita a borda do objeto vale dy/dx = tg(α). Pode-se notar facilmente que $\alpha = \theta - 90^{\circ}$. Graficamente pode-se confirmar também que

a direção do gradiente Φ é a mesma do ângulo θ. E, portanto, a equação 2.1 se torna equivalente à equação 2.7:

$$x.\cos\Phi + y.\mathrm{sen}\Phi = \rho \qquad \text{(equação 2.7)}$$

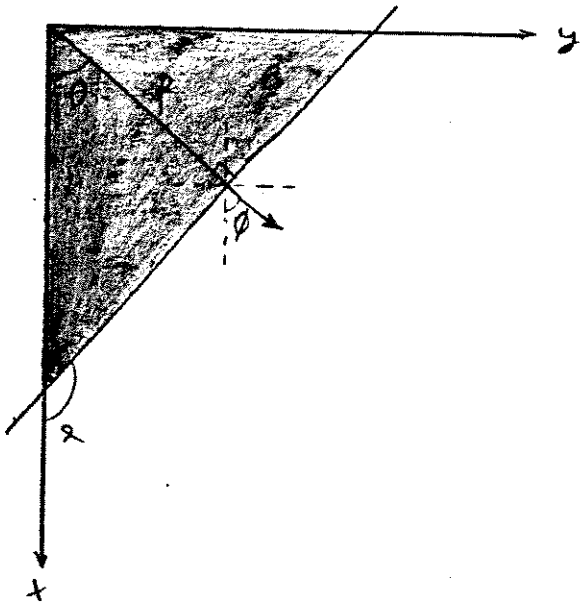


Figura 2.14. Uma imagem em níveis de cinza de uma borda e os ângulos de interesse: θ, α e Φ. Pode-se peceber que $\theta=\Phi$ e que $\alpha=\theta-90^{\circ}$

Se apenas a função $f(z,a)=0$ for utilizada, como vem sendo considerado até agora (forma "clássica"), o custo da computação é exponencial no número m de parâmetros, sendo proporcional a $O(n^{m-1})$ e, no caso de retas, proporcional a $O(n)$. Com o uso da informação direcional do gradiente, reduz-se o custo para $O(n^{m-2})$ e, no caso de retas, $O(1)$.

Em uma borda não-ideal, o vetor gradiente aponta sempre para a direção de maior variação da função. A direção do gradiente (Φ, na figura 2.14) será, teoricamente, perpendicular à reta que delimita a borda. E, portanto $\Phi \cong \theta$. Na prática, infelizmente, também não é possível obter grande precisão no cálculo do gradiente dos pixels da borda utilizando máscaras 3x3 como as implementadas. É, portanto, necessário utilizar um conjunto de valores de θ, centrado no valor Φ nominal do gradiente. O gradiente funciona, então, como um guia inicial da região onde é mais provável encontrar-se a nuvem de pontos.

Desse modo, para se utilizar a transformada de Hough com informação direcional, algumas modificações tiveram de ser feitas. O algoritmo modificado implementado é o seguinte: Para cada pixel da imagem $I(x,y)$, computa-se $|\nabla I(x,y)|$ e $\Phi(x,y)$ através do operador de Sobel. Se $|\nabla I(x,y)| \geq L$, onde L é um limiar pré-fixado, então o pixel pertence à borda do objeto. Para cada pixel de borda assim detectado, calcula-se ρ (pela equação 2.1 ou 2.7) para todos os ângulos situados no intervalo $[\Phi-(k/2).\Delta\theta, \Phi+(k/2).\Delta\theta]$, onde k é o número (inteiro) pré-fixado de ângulos em tôrno de Φ onde ρ também deve ser calculado e adiciona-se 1 à matriz de votos nas posições .

O valor de k é um valor que depende da aplicação. Aplicações que requerem maior precisão devem ter um k maior. Em compensação quanto maior k, maior o tempo de execução. No limite, para $k = N_\theta$, a transformada modificada de Hough que utiliza informação direcional equivale-se à transformada "clássica" de Hough. Nos experimentos foram utilizados os valores $10 \leq k \leq 20$. Ou seja, para $k = 20$, calcula-se ρ para os próximos 10 ângulos discretizados à esquerda de Φ e para os próximos 10 ângulos discretizados à direita de Φ.

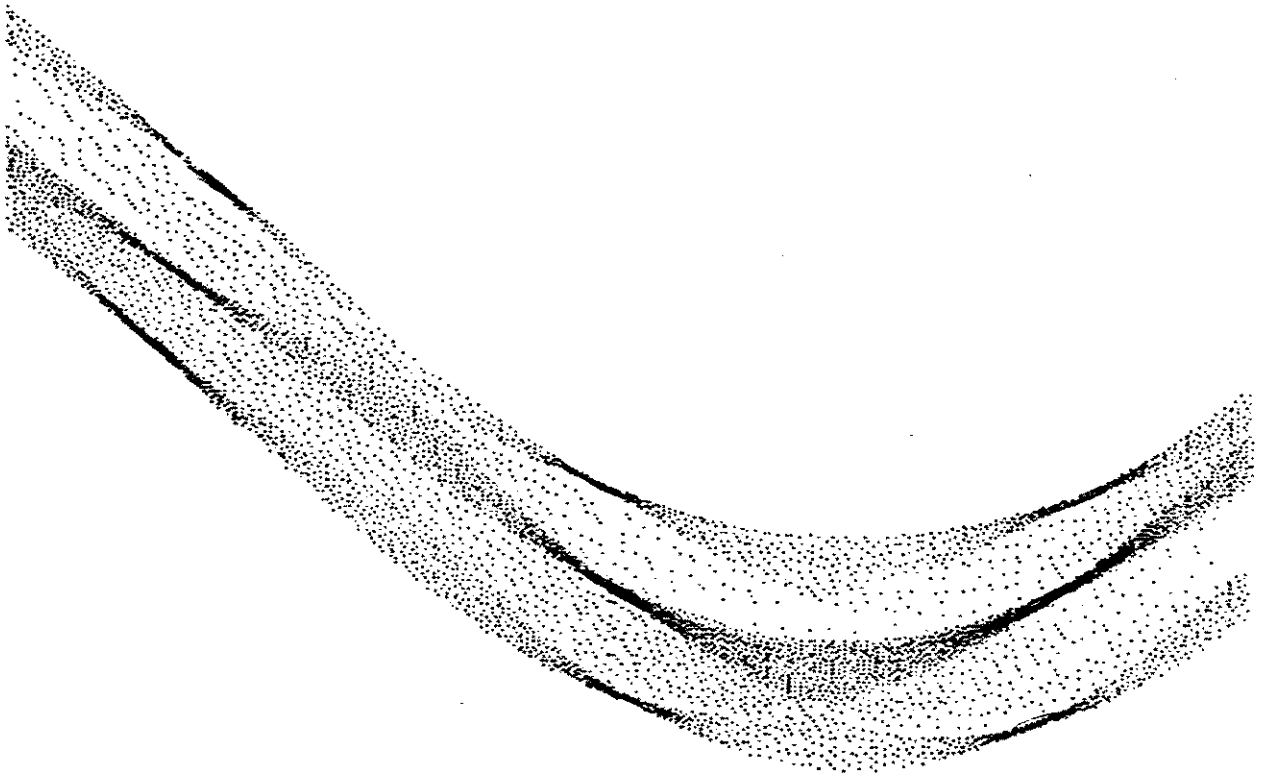


Figura 2.15. O espaço Hough para o algoritmo clássico.

A informação do gradiente também pode ser utilizada como uma regra heurística no procedimento de votação. Ao invés de se incrementar de um, a matriz de acumulação pode ser incrementada por uma função da magnitude do gradiente, conforme O'GORMAN & CLOWES (1976).

Embora a Transformada de Hough tenha comprovada capacidade de identificação de curvas parametrizadas, algumas críticas ainda podem ser feitas a ela: (1) **Seleção dos paramêtros**. A utilização de ρ e θ como parâmetros torna necessária a utilização de funções "seno" e "co-seno", que dispendem tempo de cálculo ou memória (se forem usadas LUTs). (2) **Sub-utilização da memória**. O espaço Hough possui uma forma irregular. Ou seja, nem todos os pares (ρ,θ) na matriz de votos representam retas possíveis na imagem. Portanto, existem acumuladores que jamais serão preenchidos. (3) **Segmentos de retas colineares são indistinguíveis**. A informação sobre a localização das retas detectadas é perdida. Da mesma forma, o comprimento real e o início de segmentos colineares não podem ser recuperados.

As nuvens de pontos no Espaço Hough representam as retas no plano imagem. Estas nuvens de pontos serão o objeto de análise da próxima etapa do algoritmo.

2.2.4 - (4ª fase) - Classificação das Retas

Após a construção do Espaço Hough, da maneira descrita na seção anterior, é necessário encontrar seus picos locais, que representam as retas desejadas. O problema que se coloca é: dado um espaço Hough, contendo várias nuvens de pontos, (1) detectá-las e isolá-las e (2) para cada nuvem isolada, determinar estatisticamente um ponto que melhor represente a nuvem. É importante ressaltar que supõe-se que a nuvem é suave e tem um único pico, representado por um, ou mais, pontos da mesma nuvem e **conectados**. Há algumas abordagens possíveis para a implementação do algoritmo de detecção e isolamento das nuvens, bem como para a determinação de seu "pico".

Uma primeira abordagem para a detecção das nuvens é limiarizar todo o Espaço Hough e, em seguida, fazer uma varredura para a detecção dos aglomerados de pontos. Contudo, sabe-se que, pela posição (3D) do

objeto na imagem, algumas de suas bordas podem possuir pequena dimensão. Como consequência, isto produz um máximo local correspondente - de importante significado - mas que pode estar abaixo do limiar escolhido. Uma segunda abordagem é convoluir o Espaço Hough com uma máscara de realce dos picos locais, como um filtro passa-altas. O problema com esse tipo de operador é que qualquer ruído no espaço Hough é amplificado.

O algoritmo implementado utiliza características das duas abordagens descritas para a detecção das nuvens. Primeiramente é feita uma limiarização, que tem por objetivo acabar com os "picos-ruído". Em seguida é feita a detecção das nuvens através da máscara de realce de picos. O resultado desta detecção é um conjunto de três vetores acumuladores v_i - um para acumular θ, outro para ρ e outro para $HS[\theta][\rho]$ - para cada pico detectado, isolando-o dos demais. Para dirigir esta busca são criadas também três variáveis para cada v_i: o limiar **t**, o centro da nuvem **c** e o raio de busca em torno da nuvem **d**. É importante notar que estes parâmetros podem ser pré-fixados (estáticos) ou obtidos como resultado da *última imagem analisada* (dinâmicos), conforme a necessidade da aplicação.

O algoritmo de classificação das retas pode ser descrito da seguinte maneira: cada valor (θ,ρ) da nuvem, pertence ao vetor acumulador v_i se (1) o valor $HS[\theta][\rho]$ for maior que um limiar **t** mínimo pré-fixado; e (2) se estiver a uma distância mínima **d** do centro atual **c** da nuvem. Desse modo, um número qualquer de retas pode ser obtido.

Alguns comentários devem ser feitos sobre as variáveis dinâmicas: t=limiar; d=tolerância; c=centro atual, já descritas. O limiar t deve existir por vários motivos: (1) existem partes do Espaço Hough que não correspondem a retas reais no plano imagem; (2) somente retas com um tamanho mínimo devem ser consideradas para haver uma certa imunidade ao ruído. A tolerância ou distância mínima d é um parâmetro que dá a dimensão permitida para a nuvem. Este parâmetro não deve ser muito grande - para não embaralhar retas distintas, mas com parâmetros próximos - nem ser muito pequeno, comprometendo a precisão da análise estatística dos dados. O centro atual da nuvem é o parâmetro sobre o qual é medida a tolerância, já citada. Na implementação relizada, o parâmetro c é definido pelas coordenadas do ponto que contém o maior número de votos da nuvem.

$\theta = 23,73^{\circ}$	$\rho = 112,94$
$\theta = 27,07^{\circ}$	$\rho = 170,04$
$\theta = 30,58^{\circ}$	$\rho = 248,72$
$\theta = 88,94^{\circ}$	$\rho = 233,49$
$\theta = 91,58^{\circ}$	$\rho = 168,77$
$\theta = 93,51^{\circ}$	$\rho = 106,59$
$\theta = 149,58^{\circ}$	$\rho = -72,96$
$\theta = 154,68^{\circ}$	$\rho = -16,49$
$\theta = 156,79^{\circ}$	$\rho = 25,37$

Figura 2.16. Os parâmetros das equações das retas da figura 2.15.

Uma vez detectada e isolada a nuvem de pontos, resta analisar estatisticamente os acumuladores v_i e obter, portanto, uma melhor aproximação para o valor real do pico local do aglomerado de pontos. Para cada vetor acumulador, avalia-se qual o par ρ_m e θ_m que melhor representa a reta correspondente à nuvem. Para se fazer a análise estatística das nuvens é utilizada uma das seguintes operações: média geométrica, média ponderada, mediana, etc. Um exemplo do resultado final desta fase é apresentado na figura 2.16, mostrando os parâmetros das 9 retas presentes na figura 2.15.

Em caso de estar-se utilizando o algoritmo de classificação de retas em uma sequência temporal de imagens, pode não ser preciso executar-se a operação de detecção e isolamento dos picos. Nesse caso assume-se que o movimento da câmera foi suficientemente pequeno para que a vizinhança de um pico isolado na imagem anterior ainda contenha o verdadeiro pico. Desse modo não é necessário varrer todo o espaço Hough para isolar a nuvem: basta fazer uma busca na vizinhança do pico detectado na imagem anterior, guiando-se a busca pelas variáveis c, t e d, calculadas na imagem anterior. Consegue-se, assim, ganhos no tempo de busca e de classificação das nuvens.

2.2.5 - (5ª fase) - Cálculo das Propriedades

Após as retas terem sido detectadas, deve-se casar cada uma delas com o modelo do paralelepípedo que se está querendo identificar. Portanto, no pior caso, existem 9 retas visíveis, correspondendo às

bordas visíveis do paralelepípedo, e 7 vértices, correspondendo aos vértices visíveis do mesmo.

Conforme apresentado na seção 2.2.4, o espaço Hough não fornece diretamente os vértices dos objetos, necessários para se fazer o reconhecimento do objeto. É necessário fazer-se uma interpretação das equações das retas, detectadas na fase anterior (4ª fase), e obter as coordenadas dos vértices através do ponto de intersecção das retas que se interceptam no vértice. Esta operação é necessária para qualquer tipo de objeto poliédrico.

Observando-se a estrutura dos picos no espaço Hough - entre elas, a figura 2.15 - pode-se observar cinco propriedades importantes:

(1) O número de bordas visíveis de um objeto na imagem é igual ao número de nuvens mapeadas no espaço Hough. Segmentos de retas (ou bordas) distintos, porém colineares, são mapeados na mesma nuvem.

(2) Retas equidistantes da origem na imagem, ou seja, que possuem o mesmo $|\rho|$, são mapeadas em nuvens verticalmente alinhadas (paralelas ao eixo θ) no espaço Hough.

(3) Retas com mesma inclinação na imagem, ou seja retas paralelas, possuem mesmo θ, e são mapeadas em nuvens horizontalmente alinhadas (paralelas ao eixo ρ) no espaço Hough.

(4) N Retas concorrentes em um único ponto na imagem (por exemplo, um vértice), são mapeadas em um arranjo senoidal de N nuvens no espaço Hough.

(5) Dois ou mais arranjos senoidais - definido na propriedade 4 - de nuvens no espaço Hough interceptando-se em uma outra nuvem correspondem a vértices compartilhando uma borda comum na imagem.

Através dessas 5 propriedades pode-se definir um algoritmo para se fazer o casamento entre os picos das nuvens detectadas no espaço Hough e as arestas do modelo de, teoricamente, qualquer objeto poliédrico. Será visto como as propriedades 1, 2 e 3 foram utilizadas para se detectar um paralelepípedo. As propriedades 4 e 5 podem ser utilizadas para se ampliar a quantidade de objetos que se pode reconhecer, incluindo, por exemplo, pirâmides, prismas, cunhas, tetraedros e as várias combinações entre estas.

No caso de objetos do tipo paralelepípedo os seguintes passos são seguidos: A propriedade 1 é utilizada para se definir em qual dos 3 casos da figura 2.2 o objeto se encontra. As propriedades 2 e 3 são

utilizadas para se fazer o isolamento das nuvens da maneira que será descrita a seguir.

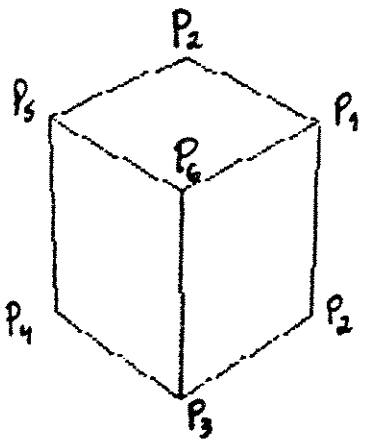


Figura 2.17. Paralelepípedo com os vértices identificados.

Qualsquer que sejam as posições possíveis do paralelepípedo no plano imagem (4, 7 ou 9 arestas, conforme figura 2.2) pode-se relacionar os picos do espaço Hough com as retas do modelo. Analisando-se o conjunto de retas fornecidas pela 4ª fase do algoritmo, vê-se que estas podem ser separadas em 3 grupos diferentes, utilizando-se para isso o valor de θ (ver figura 2.16). O motivo disto é que o paralelepípedo (3D) é formado por 3 conjuntos distintos de retas paralelas. Portanto, no plano imagem estas retas possuem aproximadamente a mesma inclinação (pela propriedade 3). Assim, separam-se, pelo ângulo θ, as 9 retas iniciais em 3 conjuntos com 3 retas cada um. Os limites desta classificação inicial dependem do modelo de objeto escolhido.

Após esta primeira classificação, cada conjunto de três retas é finalmente separado segundo ρ para isolar cada nuvem (através da propriedade 2). Isto pode ser observado na figura 2.17: dentro de cada conjunto de três nuvens, aquela que tem a menor distância ρ está mais próxima à origem e, portanto, refere-se a uma das três arestas mais próximas da origem, de acordo com o conjunto a que pertencer. As demais arestas são identificadas do mesmo modo.

```
ponto 0:  intersec(pra_la[2],pra_ca[0],ponto[0]);

ponto 1:  intersec(de_pe[0],pra_ca[0],tmp[0]);
          intersec(pra_ca[0],pra_la[1],tmp[1]);
          intersec(pra_la[1],de_pe[0],tmp[2]);
          ponto[1][0] = (tmp[0][0] + tmp[1][0] + tmp[2][0])/3;
          ponto[1][1] = (tmp[0][1] + tmp[1][1] + tmp[2][1])/3;

ponto 2:  intersec(de_pe[0],pra_la[0],ponto[2]);

ponto 3:  intersec(pra_ca[2],pra_la[0],tmp[0]);
          intersec(de_pe[1],pra_la[0],tmp[1]);
          intersec(de_pe[1],pra_ca[2],tmp[2]);
          ponto[3][0] = (tmp[0][0] + tmp[1][0] + tmp[2][0])/3;
          ponto[3][1] = (tmp[0][1] + tmp[1][1] + tmp[2][1])/3;

ponto 4:  intersec(de_pe[2],pra_ca[2],ponto[4]);

ponto 5:  intersec(de_pe[2],pra_la[2],tmp[0]);
          intersec(pra_la[2],pra_ca[1],tmp[1]);
          intersec(de_pe[2],pra_ca[1],tmp[2]);
          ponto[5][0] = (tmp[0][0] + tmp[1][0] + tmp[2][0])/3;
          ponto[5][1] = (tmp[0][1] + tmp[1][1] + tmp[2][1])/3;

ponto 6:  intersec(de_pe[1],pra_la[1],tmp[0]);
          intersec(de_pe[1],pra_ca[1],tmp[1]);
          intersec(pra_la[1],pra_ca[1],tmp[2]);
          ponto[6][0] = (tmp[0][0] + tmp[1][0] + tmp[2][0])/3;
          ponto[6][1] = (tmp[0][1] + tmp[1][1] + tmp[2][1])/3;
```

```
P_F_1 :   intersec(de_pe[0],de_pe[1],tmp[0]);
          intersec(de_pe[1],de_pe[2],tmp[1]);
          intersec(de_pe[2],de_pe[0],tmp[2]);
          p_fuga[0][0] = (tmp[0][0] + tmp[1][0] + tmp[2][0])/3;
          p_fuga[0][1] = (tmp[0][1] + tmp[1][1] + tmp[2][1])/3;

P_F_2 :   intersec(pra_ca[0],pra_ca[1],tmp[0]);
          intersec(pra_ca[1],pra_ca[2],tmp[1]);
          intersec(pra_ca[2],pra_ca[0],tmp[2]);
          p_fuga[0][0] = (tmp[0][0] + tmp[1][0] + tmp[2][0])/3;
          p_fuga[0][1] = (tmp[0][1] + tmp[1][1] + tmp[2][1])/3;

P_F_3 :   intersec(pra_la[0],pra_la[1],tmp[0]);
          intersec(pra_la[1],pra_la[2],tmp[1]);
          intersec(pra_la[2],pra_la[0],tmp[2]);
          p_fuga[0][0] = (tmp[0][0] + tmp[1][0] + tmp[2][0])/3;
          p_fuga[0][1] = (tmp[0][1] + tmp[1][1] + tmp[2][1])/3;
```

Figura 2.18. As fórmulas para o cálculo dos vértices e dos pontos de fuga do paralelepípedo da figura 2.17.

Assim, identifica-se qual reta do modelo do paralelepípedo corresponde à reta detectada. Com esta classificação é possível determinar-se os vértices do paralelepípedo e os seus 3 pontos de fuga, fazendo-se a intersecção correspondente dos vértices

identificados, conforme figura 2.18. Deve-se notar que estes 3 pontos de fuga são chamados pontos de fuga principais porque o sistema de coordenadas fixado - conforme será visto na seção 2.2.7 - está *alinhado com as arestas do paralelepípedo.*

A projeção perspectiva de qualquer conjunto de retas paralelas, que NÃO sejam paralelas ao plano de projeção (nesse caso, o plano imagem), convergirá para um ponto de fuga. No espaço 3D os pontos de fuga podem ser encarados como a projeção de um ponto no infinito. Existem, é claro, infinitos pontos de fuga. Se o conjunto de retas é paralelo a um dos três eixos de um sistema de coordenadas, o ponto é chamado de PONTO DE FUGA PRINCIPAL. Existem, no máximo, três pontos de fuga principais, correspondendo ao número de eixos principais cortados pelo plano de projeção. Os pontos de fuga são utilizados na 7ª fase para se fazer a calibração da câmera.

Nesta fase, portanto, os dados de saída são: os 3 pontos de fuga, as equações das 9 retas que suportam o paralelepípedo e as coordenadas 2D dos 7 vértices visíveis da mesma.

2.2.6 - (6ª fase) - Ajuste das Equações das Retas

Os pontos de fuga e os vértices do paralelepípedo encontrados através da transformada de Hough estão sujeitos a alguns erros. Um modo de se melhorar o método é utilizar uma equação de reta fornecida pela 5ª fase do algoritmo e compará-la com a imagem (binária) de entrada. Utiliza-se essa equação de reta como uma estimativa inicial a ser comparada com os pixels que compõem a reta exata na imagem de entrada.

Uma realimentação desse tipo utiliza a transformada de Hough apenas como ferramenta para se encontrar a região onde o segmento de reta se situa. Ao determinar-se a região onde se encontra a reta (ou segmento de reta) desejada, faz-se uma varredura nesta região para coletar os pixels que realmente constituem a reta e, através de um ajuste de curvas, obter uma equação mais precisa da reta desejada.

A região de busca dos pixels é retangular, centrada na estimativa inicial da equação da reta e tem como extremidades os vértices calculados. Esta região possui uma largura w que depende da largura da reta na imagem. Se o algoritmo de afinamento tiver sido aplicado, a

região pode ter w ≅ 2. Em caso contrário, a largura w pode chegar a até 10 pixels, dependendo da largura da reta na imagem. Uma versão simplificada do algoritmo pode ser vista na figura 2.19.

Nos vértices do paralelepípedo, onde há retas concorrentes, a região de busca pode conter pixels que não pertencem à reta. Um modo de se tratar esse problema, no caso da detecção do paralelepípedo, é não se considerar os pixels próximos às extremidades da região de busca.

Geralmente, esta tarefa de ajuste das equações de reta não toma muito tempo de processamento, ver tabela 5.1. O tempo de processamento depende da largura w e do comprimento do segmento de reta a ser encontrado.

Existem vários métodos de ajustes de curvas propostos na literatura: ver, por exemplo FISCHLER & BOLLES (1981). O método mais tradicional é o de minimização do erro quadrático médio - que é desenvolvido no apêndice 2 - e que foi escolhido para ser implementado.

```
PROC ajusta.reta( [2]INT v, [N][2]S )
  ...  declarações e inicializações
  SEQ
    ...  pixel v será a semente
    ...  apaga pixel semente
    ...  inclui pixel semente na lista S
    ...  inclui pixel semente na lista L
    WHILE  ( lista L não vazia )
      SEQ
        ...  retira o primeiro elemento da lista L
        SEQ i = -1 FOR 3
          SEQ  j = -1 FOR 3
            IF
              ( imagem[(i+x)][(j+y)] := PRETO )
                SEQ
                  imagem[(i+x)][(j+y)] := BRANCO
                  ...  inclui (i+x) e (j+y) na lista S
                  ...  inclui (i+x) e (j+y) na lista L
              TRUE
                SKIP
```

Figura 2.19. Algoritmo de segmentação da reta.

A idéia deste algoritmo pode ser resumida da seguinte maneira: (1) Para cada uma das 9 equações define-se uma região de busca com largura w e comprimento igual à distância entre os vértices da respectiva aresta. (2) Para cada uma das 9 regiões definidas, faz-se uma busca na imagem de entrada e formam-se 9 vetores com pixels de borda pertencentes àquela região. (3) Para cada um dos 9 vetores de pixels, ajusta-se uma reta segundo as equações abaixo, cuja dedução é apresentada no apêndice 2. O método de ajuste pode ser qualquer um dos 3 descritos no apêndice 2. As equações abaixo apresentadas correspondem ao 2º método. (4) Pode-se, então, recalcular as coordenadas dos 7 vértices e dos 3 pontos de fuga com precisão maior.

$$\rho = \frac{(\cos\theta.\Sigma x + \operatorname{sen}\theta.\Sigma y)}{n} \qquad \text{e} \qquad \theta = \tan^{-1}[(A \pm \sqrt{A^2 + 4B^2})/2B]$$

onde:

$$A = \frac{(\Sigma x)^2}{n} - \frac{(\Sigma y)^2}{n} + \Sigma y^2 - \Sigma x^2 \qquad \text{e} \qquad B = \Sigma xy - \frac{\Sigma x.\Sigma y}{n}$$

Uma das vantagens do ajuste das equações das retas é a de permitir que o algoritmo da transformada de Hough não necessite ter grande precisão. Uma desvantagem é que o custo computacional desta fase é grande e tende a crescer com o tamanho e o número de retas detectadas. Abordagens diferentes para esta fase são apresentadas por DUDANI e LUK (1978) e O'GORMAN e CLOWES (1976) utilizando também dados da transformada de Hough em conjunto com informações direcionais.

2.2.7 - (7ª fase) - Calibração

Calibração é o processo de determinação das características geométricas e ópticas internas da câmera (chamados **parâmetros intrínsecos** da câmera) e a posição e orientação 3D do referencial da câmera em relação ao referencial do espaço objeto (chamados **parâmetros extrínsecos** da câmera). O objetivo da calibração é estabelecer o relacionamento entre as coordenadas 3D do mundo real e as coordenadas 2D da imagem a ser processada. Os parâmetros intrínsecos da câmera são:

- A distância focal da câmera (f).
- O centro do plano imagem (o).
- Os coeficientes de distorção óptica (k).
- Os fatores de escala do plano imagem (s_x e s_y).

Os parâmetros extrínsecos da câmera são aqueles desenvolvidos no modelo de colinearidade (ver apêndice 3) :

- Posição de {C} . (x_c , y_c , z_c)
- Orientação de {C} . (θ, ϕ, ψ)

onde

- θ = rotação pelo eixo z
- ϕ = rotação pelo eixo x
- ψ = rotação pelo eixo y
- x_c , y_c , z_c são a posição da câmera no espaço

Os parâmetros intrínsecos dependem do tipo de câmera utilizada, segundo TSAI e LENZ (1988). Como para a aplicação em pauta são utilizadas câmeras CCD, somente este caso será analisado. A distância focal da câmera é o principal parâmetro da transformação perspectiva. O centro do plano imagem serve como parâmetro de alinhamento do sistema de coordenadas da lente e o sistema de coordenadas do plano imagem. Idealmente, esses devem ser coincidentes. Todavia, já foram citados na literatura, erros de até 20 pixels nesse parâmetro, conforme TSAI e LENZ (1988). O desvio encontrado deve ser corrigido em todos os valores do plano imagem.

Os coeficientes de distorção óptica retratam o desvio, em relação à trajetória ideal, do raio incidente ao atravessar a lente. As distorções nas lentes surgem devido à impossibilidade prática de se produzir lentes com a forma de um paraboloide de revolução, recorrendo-se então a uma aproximação esférica. Vários tipos de erros são gerados: astigmatismos, curvatura de campo, coma, distorção, etc. A distorção é caracterizada como radial e simétrica em relação ao centro do plano imagem e pode ser modelada por um polinômio. Os coeficientes desse polinômio corrigem a localização dos pontos do seguinte modo:

$$x = x_d \cdot (1+kr^2) \qquad y = y_d \cdot (1+kr^2)$$

onde (x_d, y_d) são as coordenadas distorcidas e $r^2 = (x_d + y_d)$.

Os fatores de escala s_x e s_y são consequência do processo de digitalização da imagem. Devido ao método de varredura, a distância vertical entre 2 elementos do sensor (CCD) é conhecida com grande precisão. Ou seja, s_y pode ser obtido do fabricante. A incerteza no valor do fator de escala horizontal s_x acontece devido à coincidência imperfeita entre o hardware do sistema de aquisição da imagem e o hardware da câmera. Durante a varredura, os sinais discretos captados por cada linha da matriz de sensores, são convertidos para um sinal analógico e posteriormente amostrados pelo sistema de aquisição de imagens em amostras discretas e colocados no "frame buffer". O fator de escala é a relação entre o número de elementos sensores em uma linha e o número de pixels em uma linha do "frame-buffer". Ou seja:

$$s_x \cdot \frac{f_e}{f_p} = s_x \cdot \frac{N_{cx}}{N_{fx}}$$

onde,

f_e = frequência do registrador de deslocamento do sensor
f_p = frequência do "frame grabber"
N_{cx} = número de elementos do sensor na direção x
N_{fx} = número de pixels amostrados por uma linha

A utilização de Visão monocular para a localização de objetos e pontos no espaço 3D tem sido muito pesquisada: DHOME et al. (1989) descrevem um método para se localizar a câmera no espaço 3D através da interpretação de 3 retas de um objeto e utilizando algumas regras heurísticas. LEE et al. (1990) descrevem um método para se localizar a câmera no espaço 3D utilizando-se Visão monocular de formas retangulares; LIU, HUANG e FAUGERAS (1990) descrevem um método para a calibração de câmeras utilizando retas no espaço 3D. WANG e TSAI (1990) e ECHIGO (1990) descrevem um método analítico para localização da câmera no espaço 3D, utilizando-se os pontos de fuga de um paralelepípedo. CHEN e TSAI (1990) descrevem um método para se calibrar uma câmera utilizando-se retas no espaço 3D cujas posições

relativas são conhecidas *à priori*. Este algoritmo é melhorado e generalizado pelos mesmos autores, em CHEN e TSAI (1991), onde descrevem um método para a determinação da localização da câmera através de "pincéis" e utilizando vários tipos de objetos simples. CHEN e JIANG (1991) descrevem um método iterativo para localização da câmera no espaço 3D, utilizando-se os pontos de fuga de uma grade de retas.

O método de calibração aqui desenvolvido baseia-se no trabalho de HARALICK (1980), CHEN et al. (1989) e WANG e TSAI (1990), e está descrito detalhadamente no apêndice 3. O algoritmo consta de 4 passos sequenciais: (a) Cálculo do centro do plano imagem; (b) Cálculo dos ângulos de orientação da câmera; (c) Cálculo da distância focal ; e (d) Cálculo da localização da câmera em relação ao objeto. Este algoritmo utiliza os 3 pontos de fuga calculados e as coordenadas 3D de dois vértices quaisquer do objeto.

A idéia deste algoritmo de calibração é que, colocando-se a origem do referencial 3D no próprio objeto que se quer localizar, a calibração dos parâmetros extrínsecos da câmera *já fornece a localização do objeto* em relação a este mesmo referencial. Como é necessário o conhecimento de dois vétices do objeto, toma-se um deles como sendo a origem (0,0,0) e o outro pode ser qualquer outro vértice, por exemplo, (0,0,d) onde d é uma dimensão do objeto, em metros.

Segundo a figura 2.17, considera-se o vértice P_3 do paralelepípedo como sendo a origem do referencial. O vértice P_2 está sobre o eixo X e o vértice P_4 está sobre o eixo Y e o vértice P_6 está sobre o eixo Z.

Os outros dois parâmetros intrínsecos restantes (coeficiente de distorção óptica e os fatores de escala) não necessitaram ser calculados: os coeficientes de distorção óptica (k) foram desprezados, pois têm valor insignificante para a câmera utilizada e os fatores de escala do plano imagem (s_x e s_y) são obtidos do fabricante. O fluxograma da fase de calibração é mostrado na figura 2.20.

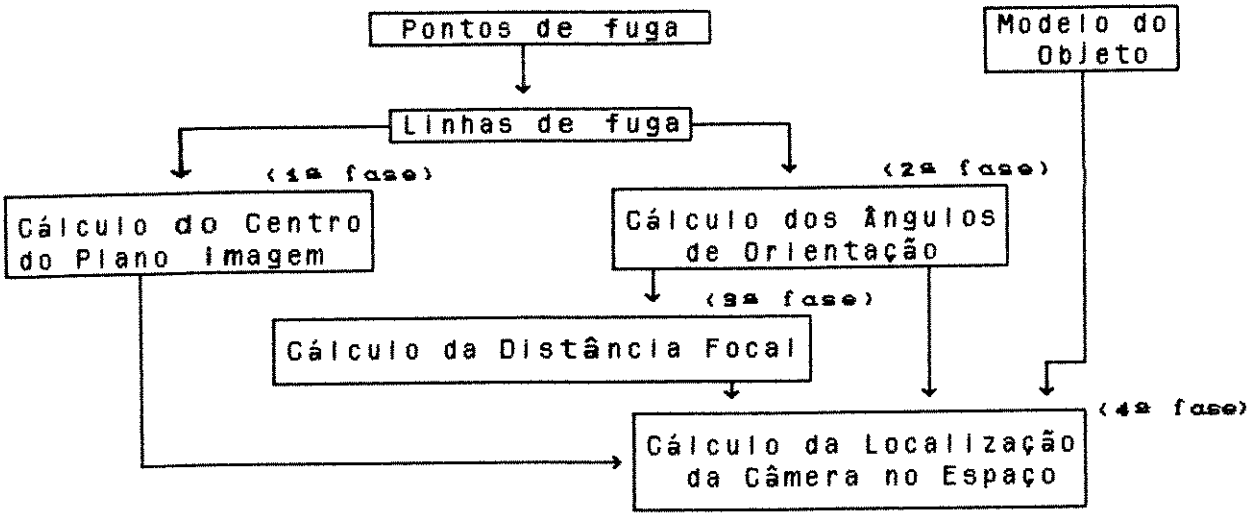


Figura 2.20. Diagrama simplificado de dependências das 4 fases do algoritmo de calibração.

Para verificar-se o funcionamento do algoritmo de calibração, foi feito um simulador para o processo de imageação (câmera). Primeiramente entra-se com os parâmetros intrínsecos e extrínsecos da câmera. Com estes parâmetros, obtêm-se as coordenadas no plano imagem (2D), referentes aos vértices do paralelepípedo a ser utilizado na calibração. A unidade métrica de saída deste simulador é o pixel. As coordenadas do objeto no plano imagem são então corrompidas por um ruído branco (amplitude 0,01 e média nula) para verificar-se a precisão e estabilidade do algoritmo.

Em seguida entra-se com as coordenadas do objeto no algoritmo de calibração. Este executa as 7 fases do processo de reconhecimento e localização de objetos do tipo paralelepípedo (descritos nas seções 2.2.1 a 2.2.7) resultando nos parâmetros de calibração procurados.

A figura 2.21 apresenta uma tabela comparativa dos resultados obtidos desse simulador e os resultados obtidos do programa de calibração.

	Parâmetros exatos	Parâmetros calculados
X	-2,00 m	-2,043 m
Y	-2,00 m	-2,043 m
Z	2,00 m	2,025 m
θ	-45,0°	-45,00°
ϕ	-45,0°	-43,72°
ψ	0,00°	0,00°
f	28,0 mm	28,52 mm

	Parâmetros exatos	Parâmetros calculados
X	-3,00 m	-3,053 m
Y	-4,00 m	-4,041 m
Z	2,00 m	2,085 m
θ	-30,0°	-29,11°
ϕ	-60,0°	-59,52°
ψ	50,0°	48,73°
f	28,0 mm	28,52 mm

Figura 2.21. Resultados do processo de localização de um paralelepípedo para duas imaens sintetizadas.

Os resultados da figura 2.21 validam o processo de reconhecimento e localização de objetos do tipo paralelepípedo quanto à precisão. Os próximos capítulos mostram uma tentativa para se acelerar o algortimo utilizando Processamento Paralelo.

CAPÍTULO III

CONSIDERAÇÕES SOBRE PROCESSAMENTO PARALELO

Neste capítulo são feitas algumas considerações sobre o desenvolvimento e implementação de algoritmos paralelos em máquinas multiprocessadoras. Procura-se fazer esta análise para dois tipos de máquinas: Single Instruction Multiple Data (SIMD) e Multiple Instruction Multiple Data (MIMD). São apresentados os aspectos de "hardware" e "software" das máquinas SIMD e MIMD ligados às implementações apresentadas nos capítulos 4 e 5.

São também apresentados aspectos de desempenho de algoritmos paralelos os quais serão utilizadas no capítulo 5 para se avaliar os resultados da paralelização do processo de visão computacional descrito no capítulo 2.

3.1 - Arquiteturas Paralelas

Existem inúmeras possibilidades de classificação de arquiteturas paralelas. Um método de classificação muito difundido foi proposto por FLYNN (1966). Seu método baseia-se nas possibilidades de combinação entre um ou mais fluxos de instruções, atuando sobre um ou mais fluxos de dados.

3.1.1 - Arquitetura MIMD

Multiple Instruction Multiple Data. São os computadores paralelos que possuem múltiplos fluxos de instrução que são executados sobre vários conjuntos de dados simultaneamente. Estes são formados através da conexão de dois ou mais processadores independentes, e interconectados por uma ligação física (uma memória ou um canal), conforme é apresentado na figura 3.1.

As arquiteturas MIMD podem ser divididas em sistemas fortemente acoplados (ou com memória compartilhada) e sistemas fracamente acoplados (ou com memória distribuída). Em sistemas fortemente acoplados, a comunicação inter-processadores é feita através da memória global. Em sistemas fracamente acoplados, a comunicação inter-processadores é feita através de uma rede de conexão.

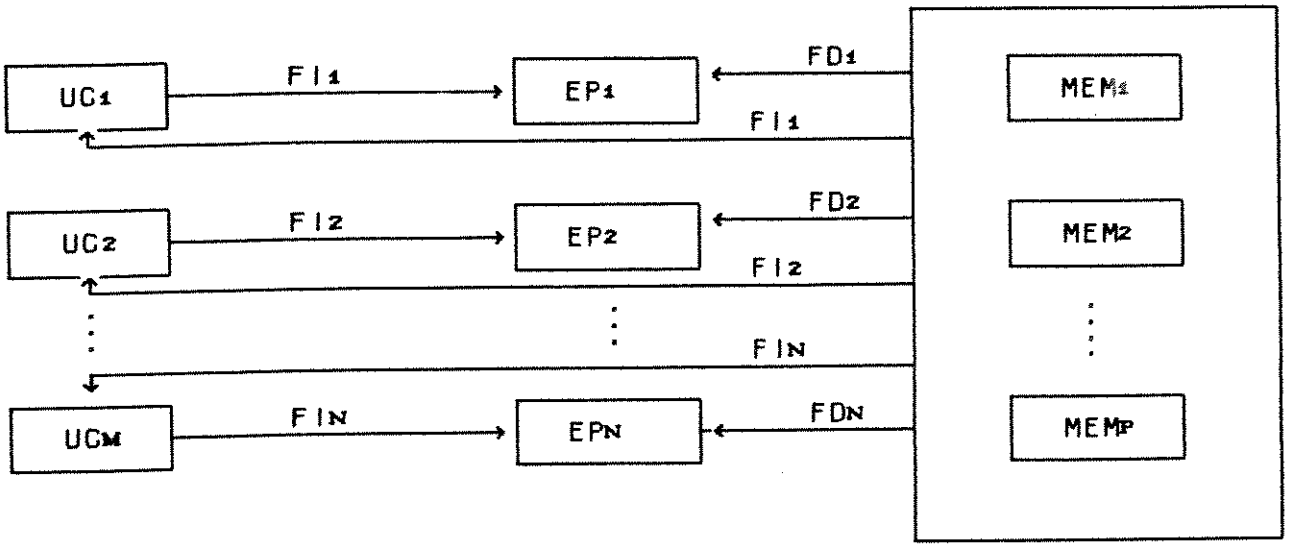


Figura 3.1. Máquina MIMD genérica. UC = Unidade de Controle; EP = Elemento Processador; MEM = Memória; FI/D = Fluxo de Instr./Dados

Arquiteturas MSIMD (Multiple SIMD) formam uma subclasse especial de computadores MIMD, conforme MARESCA et AL (1988). Neste caso, existem vários fluxos de instrução que manipulam múltiplos conjuntos de dados (como o fazem os SIMD). O escalonamento de recursos em arquiteturas MSIMD pode ser modelado por uma "rede de filas". Existem m unidades de controle idênticas no sistema, cada uma manipulando um fluxo de instruções destinadas a um conjunto compartilhado de n elementos processadores. Cada subconjunto de elementos processadores pode ser alocado a uma unidade de controle, através de uma rede de interconexão.

Um exemplo de máquina MIMD fracamente acoplada é uma rede de transputers. O transputer é um microprocessador projetado de forma a oferecer características para o uso em sistemas com multi-processamento, tais como: comunicação entre processadores por "links", mecanismo de "time-sharing" e primitivas de programação em tempo-real.

Um exemplar da família transputer é o IMST800, ver INMOS (1989). Este processador RISC de propósitos gerais, combina processamento, memória e interconexão em um único chip VLSI, (ver figura 3.2). Ele opera com 32 bits - e possui instruções para 64 bits - com velocidade de pico de 2,5 MFLOPS. O chip possui integrado uma CPU escalar e uma unidade de ponto flutuante.

O IMST800 possui uma RAM interna - não é uma memória cache - de 4K Bytes com tempo de acesso de 50 ns, uma interface para memória externa configurável e 4 canais de comunicação seriais bi-direcionais. Seu conjunto de instruções permite uma implementação eficiente de linguagens de alto nível e proporciona um suporte direto para a linguagem de programação concorrente OCCAM; conforme HOARE (1987) e INMOS (1989). Os canais de comunicação permitem a construção de diferentes topologias de redes de transputers por meio de ligações diretas entre chips, sem necessidade de lógica adicional. A velocidade média do link é de 10Mbit/s.

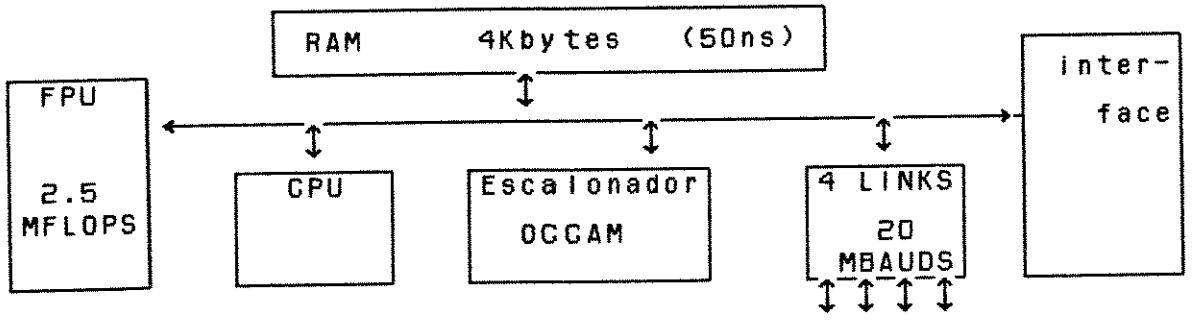


Figura 3.2. Diagrama de blocos do T800.

No IMST800 a FPU opera concorrentemente com a CPU. Portanto, é possível realizar um cálculo de endereço na CPU enquanto a FPU realiza um cálculo de ponto-flutuante. Esta concorrência leva a um aumento significativo no desempenho quando utilizado em aplicações vetoriais intensivas.

Um transputer pode implementar, através de software, todas as funções e algoritmos necessários ao processamento de imagens e visão computacional. Para aumentar seu desempenho, podem ser utilizados vários transputers em uma configuração adequada - também programável por software ou configuradas em hardware - para explorar a

concorrência entre os processos. Uma coleção de transputers pode ser configurada em uma malha com diversas topologias. Algumas topologias genéricas são apresentadas na sequência. A utilização destas topologias será descrita no capítulo 5.

Um conceito muito importante quando se trata de topologias é o conceito de **diâmetro** de uma topologia. O diâmetro de uma topolgia é definido como a maior distância entre dois nós quaisquer de uma dada topologia. E, portanto, retrata o número máximo de vezes que uma mensagem pode ser (re)transmitida entre nós quando viaja de um nó para outro nó qualquer.

Anel

Esta é a topologia mais simples. A topologia básica do tipo anel é aquela onde cada nó está ligado a apenas 2 outros nós, formando assim uma cadeia unidimensional fechada, como mostra a figura 3.3.

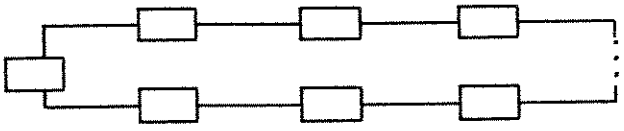


Figura 3.3. Topologia em anel.

O diâmetro do anel cresce linearmente com o número de nós processadores. Um anel pode também ter uma **orientação**. Ou seja, uma topologia do tipo anel onde as mensagens caminham em apenas uma direção pode ser diferenciada de outra topologia anel conectada por canais de comunicação bi-direcionais. A aplicação (software) é que define como utilizar as (potenciais) capacidades de orientação da topologia anel.

Array 2D

Uma topologia do tipo array d-dimensional é o conjunto ordenado e limitado de nós do hiper-espaço de d dimensões. Um anel pode ser

considerado como um array de uma dimensão (d=1).

As topologias bidimensionais (d=2), conforme figura 3.9b, têm sido muito utilizadas. Embora esta topologia tenha um diâmetro relativamente grande, $O(n\sqrt{2})$, esta se casa a muitos problemas; em particular, problemas relacionados com a geometria do espaço físico, conforme SOUCEK E SOUCEK (1988). Exemplos de tais problemas incluem simulações em hidrodinâmica, eletrodinâmica, processamento de imagens, computação gráfica, roteamentos e layouts de circuitos, etc. Em cada um destes exemplos, os padrões de comunicação são locais na rede.

O número P de nós processadores em uma topologia matriz d-dimensional é dado por P^d, onde d é a dimensão da matriz.

Árvore

As topologias da família das árvores são aquelas que possuem um nó raiz e vários ramos que se distanciam da raiz recursivamente (ou não). Um tipo muito comum de árvore é a **árvore m-ária**, onde m é comumente igual a 2. Um outro tipo são as **árvores m-lineares**. Uma árvore m-linear com apenas um ramo é equivalente a uma topologia em anel. A figura 3.4 mostra estes dois tipos de árvores.

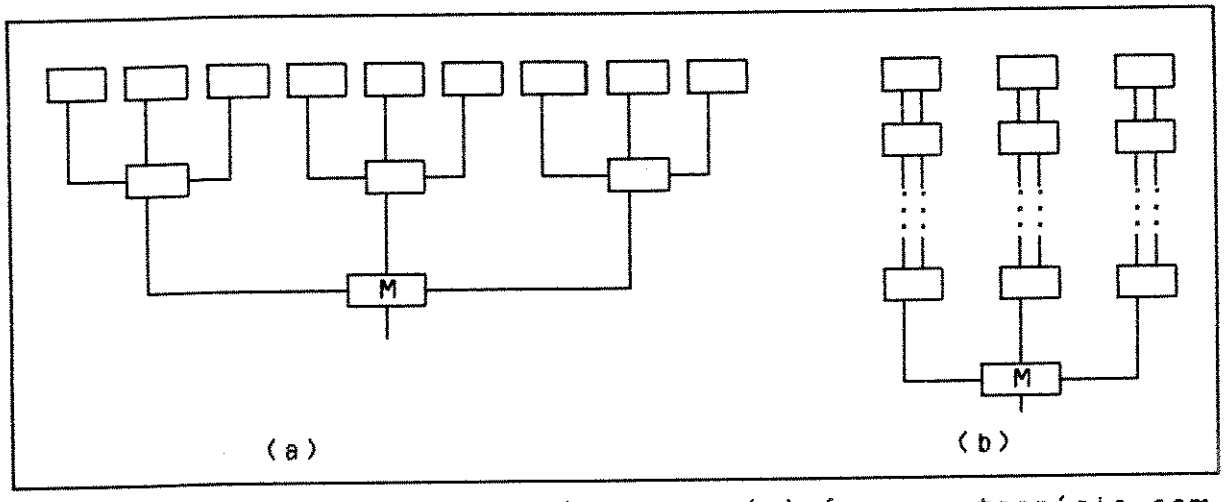


Figura 3.4 . Topologias em árvore. (a) Árvore ternária com 3 níveis. (b) Árvore 3-linear genérica com apenas uma raiz.

As vantagens das árvores incluem o pequeno diâmetro - da ordem de log(n) - e layouts eficientes em duas dimensões. Uma desvantagem é o

gargalo de comunicação na raiz da árvore. Para resolver este problema tem-se tentado aumentar o número de links perto da raiz: são as chamadas "fat-trees". As árvores m-árias possuem um diâmetro menor que as árvores m-lineares. Porém estas podem utilizar dois links ao invés de um para conectar-se com seus vizinhos.

O número P de nós processadores para uma topologia do tipo árvore binária é dado por $P = 2^{l}-1$, onde l é o nível de ramificação da árvore. Para uma árvore ternária, como a da figura 3.5, tem-se que $P = 3^{l-1}$, onde l é o nível de ramificação da árvore.

As topologias do tipo árvore são frequentemente utilizadas para o modelo de "processor farm", conforme será visto na seção 4.4.1.

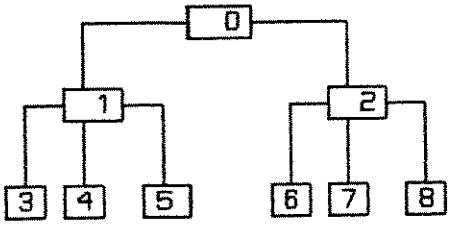


Figura 3.5. Árvore híbrida utilizada para implementar o modelo "processor farm", da seção 4.4.1.

Hipercubo

Esta topologia é adotada na Connection Machine e também pode ser programada em uma rede de transputers. Esta topologia também é citada na literatura com o nome de "Cosmic Cube", "n-cube", "binary n-cube", "boolean n-cube", etc. Nesta topolgia, cada unidade de processamento, chamada nó, pode comunicar-se diretamente com os seus n vizinhos mais próximos no espaço n-dimensional, conforme MUDGE e RAHMAN (1987) e FOX et al (1988). Atualmente tem sido comprovado que técnicas consagradas utilizadas para processadores fortemente acoplados podem ser mapeadas em um hipercubo, segundo LINDSAY (1988).

A topologia hipercúbica consiste de $N = 2^{n}$ elementos processadores (nós) e $n.2^{n}$ conexões entre eles. Os nós contêm, cada um deles, sua própria memória, processador e interfaces de conexões com seus n vizinhos. A distância média entre os nós é dada por $(n.2^{n-1})/(2^{n}-1)$ e tende a n/2 quando n tende ao infinito: n é chamada

a dimensão do hipercubo.

Um hipercubo é construído rotulando-se $N = 2^n$ nós com os números binários de 0 a (2^n-1) e conectando-se os nós de tal modo que os nós adjacentes na i-ésima dimensão sejam diferentes apenas no i-ésimo bit, conforme mostra a figura 3.6. Esta numeração é chamada de numeração **física** dos nós de um hipercubo.

Uma propriedade do hipercubo é a de que este pode ser construído recursivamente a partir de hipercubos de dimensões menores. Deste modo um hipercubo com dimensão n contém todos os outros hipercubos de dimensão menores que n. Por exemplo, o hipercubo de dimensão 3 da figura 3.6 contém simultaneamente 2 hipercubos de dimensão 2.

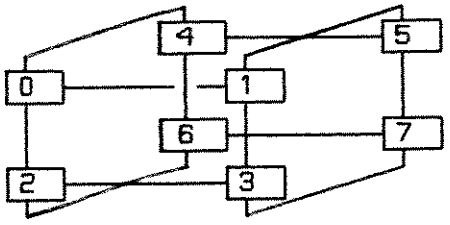


Figura 3.6. Um hipercubo com dimensão 3.

Um dos fatores que têm contribuído para o aparecimento de um grande número de máquinas com topologia hipercúbica é a possibilidade de se mapear outras topologias na topologia hipercúbica. Desse modo, algoritmos paralelos, que foram desenvolvidos para uma outra arquitetura, podem ser implementados em um hipercubo com um mínimo de adaptação no software.

3.1.2 - Arquitetura SIMD

Single Instruction Multiple Data. Corresponde aos processadores matriciais (array processors) e aos processadores associativos, nos quais vários elementos processadores são conectados a uma única unidade de controle (ver figura 3.7). A unidade de controle decodifica sequencialmente instruções e as transmite para todos os elementos processadores, que calculam seus próprios dados, em paralelo.

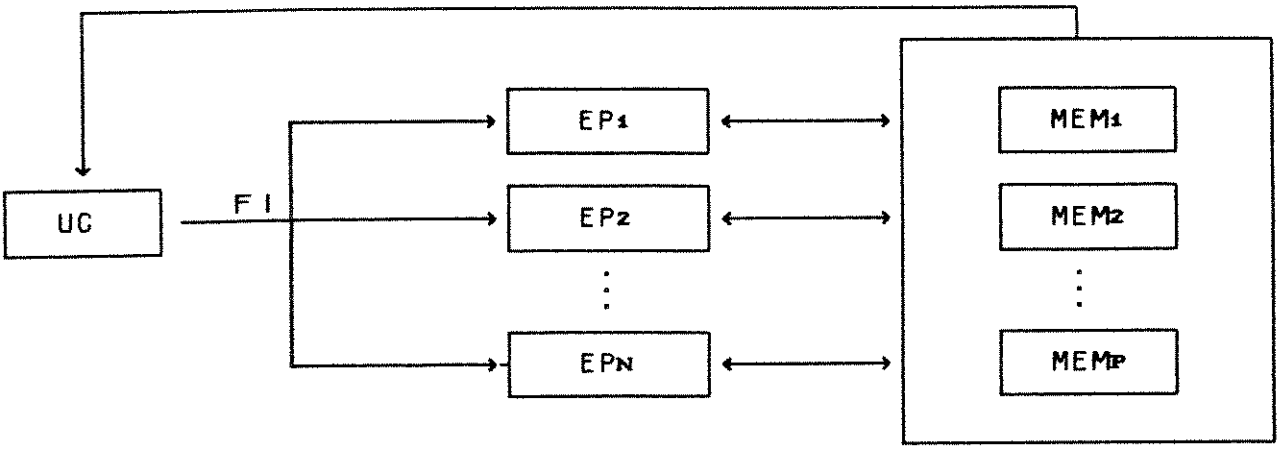


Figura 3.7. Máquina SIMD genérica. UC = Unidade de Controle; EP = Elemento Processador; MEM = Memória; FI/D = Fluxo de Instr./Dados

Um exemplar de máquina SIMD é a Connection Machine (CM), que foi introduzida por HILLIS (1985) e é um arquitetura SIMD de alta granularidade (com um grande número de células de processamento). Sua topologia é do tipo hipercúbica. A operação da CM baseia-se na interação de várias células simples e idênticas. Devido ao fato destas interações serem concorrentes, esta máquina pode ter um desempenho significativo em relação a outros computadores paralelos.

A CM possui um único relógio, um único fluxo de instruções e cada célula pode decidir - através do "bit de contexto" - se executa ou não a instrução. Suas células foram construídas com a abordagem bit-serial e com tecnologia RISC. A nanoinstrução executada na maioria dos programas é do tipo "execute a função X de sua Look-up table".

A CM é um computador acoplado ("attached computer"). Os programas para a CM residem e rodam no hospedeiro ("front-end"). A parte serial de um programa é executada no hospedeiro da maneira usual ("Von Neumann"). Quando há uma função que deve ser executada em paralelo, as instruções e os dados são enviados via um buffer de comunicação para a CM. A arquitetura básica da CM, apresentada na figura 3.8, consiste de seguintes blocos principais:

O Hospedeiro comunica-se com a CM via um barramento de memória de alta velocidade. Este computador hospedeiro armazena os programas e especifica o funcionamento das células.

O Microcontrolador é um intermediário através do qual o hospedeiro conversa com as células. Ele interpreta a macroinstrução recebida do hospedeiro e transforma-as nas chamadas microinstruções. Cada microinstrução é, posteriormente, quebrada em nanoinstruções. Estas últimas são executadas diretamente pelas células.

A Célula é a menor parte da CM. O elemento chave da máquina é um chip VLSI. Este chip contém três seções principais: a unidade de controle, a matriz de 16 células e uma unidade roteadora que faz parte da rede de transmissão de pacotes.

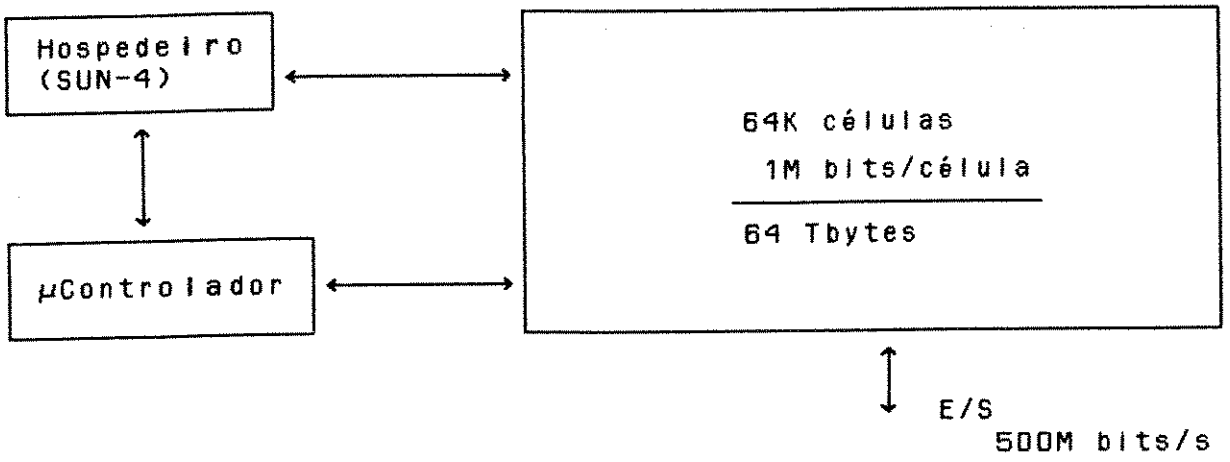


Figura 3.8 Diagrama de blocos da CM-2.

A unidade de controle recebe e decodifica as nanoinstruções enviadas pelo microcontrolador. Ela também gera os sinais necessários para controlar o processador e o roteador, sincronizado com um relógio externo. A matriz de processadores contém 16 células.

Cada célula possui uma memória interna (32K Bits a 1M Bits), uma ULA e 8 bits de status (flags). Os dados são processados um bit de cada vez. Sob a direção da unidade de controle, a célula lê dados da memória, executa operações lógicas/aritméticas nos dados, e armazena o resultado na memória, acionando um flag apropriado.

O roteador é o responsável pela transmissão e recepção de mensagens entre chips e pela entrega destas ao destino especificado por um endereço. O roteador também está diretamente conectado às memórias exteriores à célula, que servem como buffer de mensagens externas. Todas as operações do roteador são controladas pela unidade de controle.

A rede de interconexão dentro e entre os chips segue a estrutura de hipercubo de 12 dimensões, sendo que em cada vértice existe um cubo de dimensão 4, partilhando o mesmo roteador de mensagens. Cada vértice sendo integrado em um chip, acima denominado célula.

Existem vários mecanismos de comunicação na CM. Estes podem ser divididos em duas categorias: *comunicação entre hospedeiro e a CM* e *comunicação entre células*. A comunicação entre o hospedeiro e a CM é feita por três tipos de vias: **o barramento de distribuição**, que é uma árvore binária que carrega as nanoinstruções e os dados para cada uma das células; **o barramento de combinação global**, que é uma árvore binária que combina resultados de cada célula em um dado e devolve para o hospedeiro; e o **barramento escalar de memória**, por onde é possível acessar uma única célula da CM. A comunicação entre células da CM é feita por dois tipos de vias: **a rede hipercúbica**, que permite a comunicação de dados localmente, i.é, entre células vizinhas; e **os roteadores**, que possibilitam que cada processador envie uma mensagem para qualquer outro processador, simultaneamente, através do algoritmo de comunicação resumido na figura 3.10.

As operações de ponto flutuante são executadas por células especiais, também instaladas no hipercubo. Cada grupo de 32 processadores compartilha um co-processador de ponto flutuante. A interface entre as células e o co-processador é feita através de "transposers".

Esta arquitetura suporta também processadores virtuais. Um conjunto de processadores virtuais é uma estrutura lógica que permite que uma única célula possa ser subdividida em um conjunto de processadores virtuais. Um conjunto de processadores virtuais não pode ser maior que o tamanho físico de uma célula. Modelos diferentes de CM têm um número diferente de células. Uma CM com 64K células, por exemplo, pode operar como se possuísse 128k, 256k, 1M processadores virtuais. O tempo necessário para efetuar-se cada operação aumenta com o número de processadores virtuais.

Há outros tipos de arquiteturas paralelas utilizadas em Visão Computacional como, por exemplo, as Pirâmides, Árvores e outras; ver MIKLOSKO (1984), HWANG e BRIGGS (1984), AMORIM et AL (1988), RUIZ (1988) e BARROS (1990). Contudo estas arquiteturas ainda pertencem ao ambiente acadêmico e estão, portanto, ainda em fase de desenvolvimento.

3.2 - Medidas de Desempenho do Paralelismo

O tempo para um algoritmo rodar em uma arquitetura paralela, com n processadores, é dado por:

$$T(n) = T_i(n) + T_n(n) + (1-\alpha).T_c(n) + T_o(n)$$

onde n indica o número de processadores na arquitetura, α indica um "coeficiente de eficiência das comunicações", T_i o tempo de entrada dos dados na arquitetura, T_n o tempo de processamento em um nó, T_c o tempo de comunicação entre nós, e T_o o tempo para os dados saírem da arquitetura. Todos estes parâmetros são funções de n. O tempo de comunicação é uma consequência de se ter mais de um processador. T_i e T_o são geralmente dependentes da arquitetura ou, mais precisamente, dependem do modo pelo qual os dados a serem processados entram ou saem da arquitetura. T_c depende da estrutura do software de roteamento e do hardware da topologia (nº de links, configuração, velocidade dos links, etc.). T_n depende do software de aplicação. Se α puder ser mantido próximo a 1.0, os efeitos de T_c podem ser minimizados, aumentando o desempenho do algoritmo paralelo.

O sucesso de uma implementação de um algoritmo paralelo pode ser medido por um conjunto de **parâmetros de desempenho**:

ACELERAÇÃO. É o parâmetro que descreve a vantagem temporal do algoritmo paralelo comparado ao melhor algoritmo serial existente. Se T(n) é o tempo de execução do algoritmo em uma arquitetura paralela com n processadores, o speed-up é dado pela razão

$$Sp(n) = T(1) / T(n)$$

EFICIÊNCIA. É o parâmetro que mede a fração de tempo em que um processador é utilizado, ou seja, a contribuição média dos processadores para a aceleração. É calculado pela razão:

$$Ep(n) = \frac{Sp(n)}{n} = \frac{T(1)}{n.T(n)}$$

Um programa que roda em uma máquina paralela é composto de uma porção serial — geralmente operações de entrada/saída - e uma porção paralela - que é o algoritmo paralelo propriamente dito. Seja n o número de processadores, s o tempo de duração da parte serial do programa e p o tempo de duração da parte do programa que pode ser executada em paralelo. Assumindo-se que p é independente de n, pode-se tomar um problema de um tamanho fixo e rodá-lo com um número variável de processadores. Nesse caso, a aceleração do programa é dada por:

$$Sp(n) = T(1) \;/\; T(n) = (s + p)/(s + p/n) = 1/(s + p/n) = n/[1+(n-1)s]$$

Dispondo-se de infinitos processadores, percebe-se que, no limite, quando $n \to \infty$, a aceleração tende a 1/s (onde fez-se s + p = 1 por simplicidade algébrica). Este fato é conhecido como a "Lei de Amdahl". Segundo esta lei, mesmo quando a porção serial do problema for pequena, a aceleração máxima obtida é apenas 1/s.

Porém, conforme GUSTAFSON (1988), na prática, apenas a parte *paralela* do programa escala-se com n. Quando há mais processadores, o problema geralmente expande-se para utilizar os novos recursos. Os usuários têm controle sobre variáveis como resolução da imagem e tamanho do pacote de mensagens que podem ser ajustados para que o algoritmo paralelo seja executado dentro de um limite de tempo pré-fixado. Sob este ponto de vista é mais útil assumir que o **tempo de execução** é constante, e não o tamanho do problema.

Considerando a dimensão da parte paralela do problema como uma função do número de processadores, pode-se modelar este fato fazendo-se, por exemplo, p(n)=K.n, onde a constante vale K = p(1). Para poder-se comparar as acelerações, considera-se o tempo de execução da tarefa em n processadores como sendo T(n) = s + p(n) = 1. Ou seja, fixa-se o tempo de execução do problema em 1. Esta mesma parte paralela do problema deveria rodar em um processador serial n vezes mais lenta, ou seja, T(1) = s + n.p(n). Deste modo, a aceleração é medida pela razão:

$$Sp(n) = T(1)/T(n) = \frac{s + p(n).n}{s + p(n)} = s+(1-s)n = s+n-sn = n+(1-n)s$$

Com esta segunda abordagem, dispondo-se de infinitos processadores, pode-se aumentar a dimensão do problema também infinitamente. Estas duas abordagens de cálculo teórico da aceleração sugerem dois tipos de medidas de aceleração de algoritmos. De fato, os estudos sobre a paralelização do algoritmo apresentado no capítulo 2 seguem as duas possibilidades acima citadas: (1) Fixa-se a dimensão do problema de aplicação e varia-se o número de processadores; e (2) Fixa-se um tempo de execução e escala-se o algoritmo de modo que este rode dentro do tempo fixado.

Para o caso ideal, $Sp(n)=n$ e $Ep(n)=1$. Para isto ocorrer, o algoritmo paralelo deve ser tal que nenhum processador fique inativo ou faça computações desnecessárias. Esta situação teórica é inatingível na prática. Geralmente as curvas de aceleração possuem um comportamento de **saturação**. Uma implementação de um algoritmo é dita saturada quando um aumento no número de processadores não resulta no correspondente decréscimo no tempo de processamento. Os principais motivos que contribuem para a saturação são:

(a) **Balanceamento de Carga**. O speedup geralmente é limitado pelo nó mais lento, exigindo um balanceamento global da carga de processamento em todos os processadores.

(b) **Overhead de Comunicação**. Qualquer tempo perdido na comunicação diminui o desempenho global, em comparação com o caso sequencial.

(c) **Contenção**. Muitas vezes é necessário esperar por recursos não disponíveis de imediato, retardando o processamento.

(d) **Latência**. A comunicação dos dispositivos de entrada e saída não é instantânea e leva a um retardo adicional no processamento.

3.3 - A Implementação de Algoritmos Paralelos

No modelo de processamento sequencial, um programa especifica uma lista de comandos, que são executados sequencialmente. Em processamento paralelo MIMD, um programa paralelo especifica duas ou mais listas de comandos que são executados concorrentemente como processos paralelos. Em processamento paralelo SIMD, um programa

paralelo especifica uma lista de comandos, que é executada concorrentemente por todos os processadores.

O termo **concorrente** é utilizado para descrever processos que têm potencial para execução em paralelo. **Programação concorrente** é o nome dado às técnicas de programação que visam expressar o paralelismo potencial e resolver os problemas de sincronização e comunicação resultantes, sem se preocupar fundamentalmente com sua implementação.

O modelo de computação sequencial é chamado modelo RAM, de "Random Access Machine" (ELGOT e ROBINSON, 1964). O modelo RAM consiste de um único processador com acesso a uma capacidade ilimitada de memória. Esta memória armazena os dados e o programa. O programa é executado pelo processador da maneira usual, tranformando os dados iniciais, e gerando outros dados até uma parada, se ocorrer.

Os **modelos computacionais paralelos** são aqueles desenvolvidos para simular o comportamento dos algoritmos concebidos para serem executados em arquiteturas paralelas. Existe uma variedade de modelos computacionais paralelos na literatura, cada um deles assumindo diferenças no poder de computação dos processadores individuais e nos mecanismos de troca de informações entre os mesmos. Os modelos computacionais são definidos por um conjunto de **operações primitivas**, que podem ser mapeadas em instruções das máquinas paralelas. São exemplos de modelos computacionais o modelo **PRAM**, de "Parallel Random Access Machine" (TERADA, 1990), **CSP**, de "Concurrent Sequential Processes" (HOARE, 1987), **VRAM**, de "Vector Random Access Machine" (BLELLOCH, 1990) e **Sistólico** (KUNG, 1982).

Atualmente as pesquisas em modelos computacionais visam estabelecer um conjunto de primitivas que possam ser mapeadas eficientemente em vários tipos de arquiteturas paralelas (SIMD, MIMD, etc.). Todavia, esta generalidade almejada por um modelo muitas vezes não permite aos algoritmos explorar propriedades específicas de uma arquitetura, topologia de interconexão ou estrutura da máquina. A seguir serão apresentadas algumas técnicas de desenvolvimento de software paralelo que sustentam as implementações dos algoritmos dos capítulos seguintes.

3.3.1 - Desenvolvimento de Software para Arquiteturas MIMD

Na implementação de algoritmos paralelos em máquinas com arquiteturas do tipo MIMD - e, em particular, em máquinas fracamente acopladas baseadas no transputer (seção 3.1.1) - três aspectos devem ser considerados prioritariamente: o particionamento das tarefas, o gerenciamento das tarefas e a distribuição de dados entre os módulos processadores.

A meta do **particionamento** de tarefas é aumentar a utilização dos elementos processadores, maximizando o paralelismo e minimizando o custo de comunicação. As sub-tarefas resultantes são organizadas de modo que de seu grafo de dependência seja abstraída uma sub-tarefa de **gerenciamento** do fluxo de instruções. A **distribuição dos dados** entre os processadores está relacionada ao custo de comunicação. Desse modo, atividades como balanceamento de carga, balanceamento da troca de mensagens, término de tarefas devem ser cuidadosamente analisados, sob pena de criar gargalos e "deadlocks" no algoritmo paralelo.

Por exemplo, no caso de uma rede de transputers, há um conjunto de técnicas para se melhorar o desempenho de programas que rodam sobre ela. Essas técnicas baseiam-se no hardware do transputer e na linguagem OCCAM e são descritas a seguir.

(1) **Uso da memória interna**. O acesso à memória local necessita de um ciclo de memória (50 ns) enquanto a acesso à RAM externa leva 5 ciclos. Desse modo, alocando-se as variáveis mais utilizadas, por exemplo, tabelas de seno e cosseno, na memória interna pode-se aumentar a velocidade do algoritmo. (2) **Abertura de loops**. A abertura de loops pode fornecer algum ganho em tempo de processamento, pagando-se por isso em tamanho de código. Se a discretização de θ for baixa, representando um número pequeno de variáveis correspondentes a θ, esta alternativa é viável. (3) **Comunicação em pacotes**. Os links de comunicação estão conectados a unidades de DMA. Estas unidades possuem maior eficiência quanto maior for o tamanho do bloco transferido. Nos experimentos realizados, o tamanho do pacote ótimo se encontra na faixa entre 600 e 1500 bytes. (4) **Priorizar as comunicações**. Se uma mensagem que é transmitida para um transputer necessita ser retransmitida para outro, é essencial que o transputer receba os dados e os transmita o mais rápido possível. Isto é conseguido

priorizando-se as comunicações e evitando que nós fiquem presos esperando por mensagens. (5) **Desligar "range checker"**. O range checker é uma opção (default) do compilador que checa a validade dos índices dos arrays e conversões de tipos. Esta opção é muito útil durante o desenvolvimento do programa. Quando desligada, há ganhos em velocidade e diminuição do tamanho do código. (6) **Utilização de buffers**. A utilização de buffers para receber mensagens através dos links seriais desacopla a comunicação da computação. Desse modo, o processador continua em execução simultaneamente à recepção de uma mensagem.

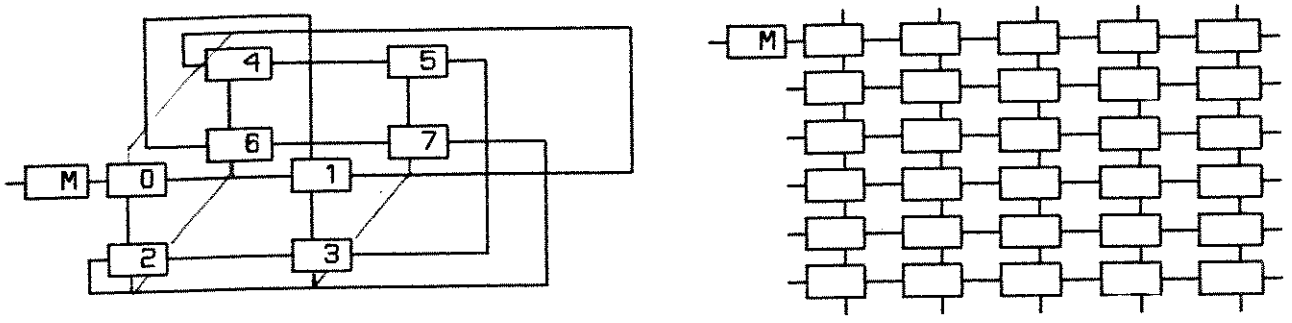


Figura 3.9 Duas topologias possíveis para os transputers.

Uma vez definido o algoritmo paralelo a ser implementado em uma arquitetura MIMD fracamente acoplada, como uma rede de transputers, deve-se escolher a topologia mais adequada ao mesmo. Há uma vasta quantidade de tipos de topologias para o processamento paralelo em arquiteturas MIMD fracamente acopladas. Na escolha da topologia, os objetivos podem ser divididos em duas categorias: desempenho e custo. Do ponto de vista do desempenho, as topologias devem possuir as seguintes características:

(1) **Pequeno diâmetro**. O diâmetro é o número máximo de vezes que uma mensagem precisa trafegar entre dois processadores quaisquer da topologia, conforme foi visto na seção 3.1.1. Se esta distância for pequena, então os processadores provavelmente irão se comunicar mais rapidamente.

(2) **Uniformidade**. É desejável que todos os pares de processadores se comuniquem com igual facilidade, ou, pelo menos, que o tráfego seja balanceado como um todo. Isto diminui a probabilidade de gargalos de comunicação.

(3) **Caminhos redundantes**. Se houver múltiplos caminhos entre cada par de processadores, uma malha parcialmente defeituosa pode continuar a funcionar. Do mesmo modo, se uma passagem é bloqueada pelo tráfego, a mensagem pode ser enviada ao longo de outra rota.

Do ponto de vista do custo, deve-se buscar as seguintes características:

(1) **Número mínimo de conexões.** Cada conexão física é cara e, otimizando-se a utilização destas, o custo pode diminuir.
(2) **Processo simples de roteamento**. O processo de roteamento (ou algoritmo de roteamento) é a camada de software responsável pela comunicação de mensagens. Como os roteadores são controlados localmente, isto mantém o custo dos roteadores em um nível baixo.

Como a rede de transputers implementa um mecanismo de comunicação por comutação de pacotes, pode ser necessário que uma mensagem caminhe através de nós intermediários até o seu destino. O algoritmo de comunicação (ou roteamento) é o responsável pela comunicação dessas mensagens. Cada transputer deve conter um algoritmo de comunicação. O algoritmo de comunicação deve, portanto, intermediar mensagens não destinadas ao transputer onde reside, receber mensagens a ele destinadas e enviar mensagens geradas pelo próprio transputer.

Talvez o fato mais importante a respeito do algoritmo de comunicação seja o de que apenas este toma conhecimento da conexão física (topologia) dos transputers. Uma vez fixada a topologia e o algoritmo de comunicação, pode-se programar o algoritmo da tarefa sem se preocupar com a arquitetura da máquina.

O algoritmo de comunicação pode ser implementado algebricamente ou a partir de tabelas. Topologias simples, podem ter algoritmos de comunicação simples. Por exemplo, a figura 3.9b é uma topologia em matriz e o algoritmo de comunicação é resumido na figura 3.10. Cada nó da matriz é numerado segundo sua coordenada (linha, coluna) = (a,b). Já para a topologia da figura 3.9a é mais difícil encontrar-se uma fórmula genérica para comunicação das mensagens. A topologia da figura 3.9a assemelha-se a um hipercubo onde foram adicionadas mais três conexões para que a distância máxima entre dois nós fosse 2.

```
BYTE a, b :              -- coordenadas particulares de cada nó
[4]CHAN OF (x,y) out, in :  -- canais mapeados nos links

  PROC comunica(CHAN OF (x,y) entra )
    :
    PROC envia(BYTE x , y )
      IF
        ( x < 0 )
          out[0] ! (a-x) ; (b-y)          -- sul
        ( x > 0 )
          out[1] ! (a-x) ; (b-y)          -- norte
        ( y < 0 )
          out[2] ! (a-x) ; (b-y)          -- leste
        ( y > 0 )
          out[3] ! (a-x) ; (b-y)          -- oeste
    :
    SEQ
      ALT i = 0 FOR 4
        in[i] ? x ; y
          SEQ
            CASE i
              0
                x := x - 1          --  sul
              1
                x := x + 1          --  norte
              2
                y := y - 1          --  leste
              3
                y := y + 1          -- oeste
            IF
              ( (x=0) AND (y=0) )     -- chegou ao destino
                entra ! x ; y
              TRUE
                envia(x,y)
    :
```

Figura 3.10. Algoritmo de comunicação para a matriz de processadores da figura 3.9b.

O algoritmo genérico de comunicação utilizando tabelas pode ser visto na figura 3.11. Assume-se que os canais in[i] e out[i], com i no intervalo $0 \leq i < 4$, estão mapeados nos links do transputer e os links in[4] e out[4] são canais internos. Assume-se ainda que "aqui" é o número do processador onde este algoritmo de comunicação está alocado e "D" é o endereço final da mensagem. Deste modo, mensagens que entram por qualquer um dos 5 canais são testadas para saber o seu destino. Se tiverem de ser repassadas, faz-se uma consulta a uma tabela para saber-se por qual link a mensagem deve ser enviada. Exemplos dessas tabelas podem ser vistos nas figuras 5.4 e 5.5.

```
ALT i = 0 FOR 5
  in[i] ? D[i] ; θ[i] ; ρ[i]
    IF
      D[i] = aqui
        entra ! D[i] ; θ[i] ; ρ[i]
      TRUE
        out[tabela[D[i]][aqui] ! D[i] ; θ[i] ; ρ[i]
```

Figura 3.11. Algoritmo Simplificado de comunicação utilizando tabelas de links.

3.3.2 - Desenvolvimento de Software para Arquiteturas SIMD

Na implementação de algoritmos paralelos em máquinas com arquiteturas do tipo SIMD - e, em particular, na Connection Machine (seção 3.1.1) - nota-se que, por possuir dezenas de milhares (e até centenas de milhares) de processadores, torna-se complexo o gerenciamento e a sincronização das tarefas. Para lidar com este problema utilizam-se *modelos computacionais massivamente paralelos*. Um deles é o modelo computacional VRAM, que também é uma extensão do modelo RAM, de ELGOT e ROBINSON (1964).

O modelo computacional Vector-RAM (VRAM), introduzido por BLELLOCH (1990), é um modelo padrão serial RAM onde é adicionado um conjunto de registradores vetoriais e um processador vetorial. Este modelo pode ser implementado em máquinas SIMD massivamente paralelas, como a Connection Machine, e em máquinas MIMD com memória distribuída.

Cada registrador vetorial pode conter um vetor arbitrariamente longo. Cada instrução do processador vetorial opera em um número determinado de vetores, da memória vetorial, e possivelmente escalares, da memória escalar. Uma instrução vetorial pode, por exemplo, somar os elementos de um vetor arbitrariamente longo, rearranjar a ordem de seus elementos ou concatenar os elementos de dois vetores em um tempo proporcional a O(1) pois os elementos são manipulados em paralelo.

Um programa para o modelo VRAM não é diferente de um programa para o modelo serial RAM, exceto por ele poder incluir primitivas vetoriais adicionais. A diferença marcante deste modelo para o modelo PRAM (TERADA, 1990) é que o modelo VRAM opera em coleções de valores (ou vetores) através de primitivas paralelas; e o modelo PRAM executa uma série de primitivas seriais em paralelo, uma em cada processador.

São introduzidas agora algumas operações primitivas do modelo VRAM que demonstram como o paralelismo é abordado neste modelo. São descritas as primitivas SCAN, PERMUTE E SEG.

a) **Scan** . Esta primitiva também é chamada de "prefix". Uma instrução Scan para um operador associativo binário $\oplus$ toma um vetor A de valores e retorna para cada posição de um novo vetor B (do mesmo tamanho de A) a soma de todas as posições anteriores em A. Por exemplo:

$$A = [\ 1\ 3\ 5\ 7\ 9\ ]$$
$$B = +\_SCAN(A) = [\ 0\ 1\ 4\ 9\ 16]$$

Se cada processador contiver um elemento do vetor, a operação é executada em um tempo que é proporcional a O(logn). Isto pode ser feito organizando-se os dados em uma árvore binária e executando-se as somas em cada um dos (logn) níveis da árvore em paralelo.

Pode parecer que a computação destas somas parciais seja um processo inerentemente serial, porque os primeiros k elementos devem ser somados antes de se somar o elemento (k+1). Realmente, com um único processador, a solução é esta. Porém, com vários processadores, pode-se fazer mais, essencialmente porque em O(logn) unidades de tempo com n processadores, podem ser feitas O(nlogn) operações individuais. Esta primitiva pode ser assim descrita, em OCCAM:

```
SEQ j = 1 FOR logn
  PAR k = 1 FOR n
    IF
       k ≥ 2^j
         x[k] := x[k-2^(j-1)] ⊕ x[k]
```

Como se percebe, o algoritmo executa de uma forma SIMD, com n processadores. Este algoritmo tem uma boa eficiência: durante o passo j, $(n-2^{j-1})$ processadores estão ativos.

Esta primitiva pode ser generalizada a partir do exemplo dado com somas para qualquer combinação de operações associativas. Algumas dessas operações são: soma, produto, máximo, mínimo, AND, OR ou XOR.

Na Connection Machine, a operação de +_scan, com elementos de 32 bits, leva 300 μs, segundo LITTLE et AL. (1989).

b) **Permutação** . A operação de permutação reordena os elementos de um vetor. Esta operação permuta os elementos do vetor de dados A para as posições especificadas pelos índices de um vetor de índices I.

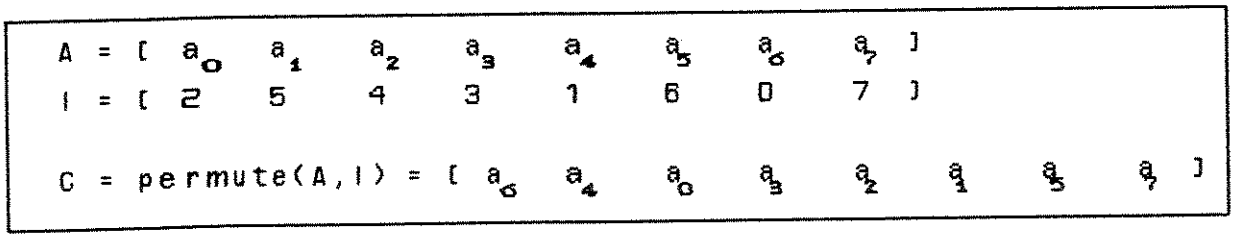


Figura 3.12. Exemplo de operação PERMUTE.

Esta operação é implementada na Connection Machine pelos roteadores de dados. Para ser implementada em uma máquina MIMD, cada processador lê seu valor e índice respectivo e envia o valor para o processador especificado pelo seu índice.

Por definição, todos os elementos do vetor I devem ser distintos, para não haver colisões e o processamento ser mais eficiente. A figura 3.12 apresenta um exemplo desta primitiva.

Na Connection Machine a operação "permute", com elementos de 32 bits, leva 500 μs em média, segundo LITTLE et AL. (1989).

c) **Primitivas Segmentadas** . Um vetor segmentado é aquele particionado em um conjunto de segmentos contíguos. O vetor segmentado pode ser dividido em dois vetores, um contendo os valores e um outro o comprimento de cada segmento. Por exemplo:

A = [P A R A L E L O]
L = [3 1 4]

representa o valor

A' = [PAR] [A] [LELO]

As primitivas segmentadas são versões das instruções vetoriais, que são executadas independentemente sobre cada segmento de um vetor segmentado. Por exemplo:

```
A = [6]   [1 3 7 9]
B = [2 4 7]   [4 8]

seg_+_scan(A)      [0]      [ 0 1 4 11]
seg_merge(A,B)     [2 4 6 7]      [1 3 4 7 8 9]
```

Há várias linguagens para a Connection Machine que seguem o modelo VRAM. Neste trabalho utilizou-se a linguagem C* (TMC, 1990), que é um super-conjunto da linguagem C, agregando os comandos paralelos do modelo VRAM à linguagem C (ANSI).

A Connection Machine possui uma topologia fixa e, desse modo, não é necessário configurá-la a exemplo da malha de transputers (seção 3.3.1). Devido ao fato da topologia da CM ser do tipo hipercubo, outras topologias podem ser mapeadas logicamente sobre esta, ver FOX et AL (1988). Por exemplo, pode-se mapear uma malha 2D (figura 3.9b), topologias em anel e topologias em árvore em um hipercubo apenas alterando logicamente os endereços dos nós processadores.

CAPÍTULO IV

O ALGORITMO PARALELO

Este capítulo apresenta as técnicas implementadas para acelerar o algoritmo de localização e reconhecimento de objetos no espaço 3D - descrito no capítulo 2 - utilizando processamento paralelo.

A partir da tabela de tempos de execução de cada fase do algoritmo de localização e reconhecimento de objetos no espaço 3D identifica-se a 3ª fase (Transformada de Hough) como sendo a mais demorada. São apresentadas as soluções encontradas para se diminuir seu tempo de execução.

4.1 - Motivação

Um ponto de partida para a "paralelização" de um algoritmo são as tabelas de tempo da implementação serial do mesmo. A tabela 4.1 apresenta a carga de tempo de CPU destinada a cada uma das fases do algoritmo serial desenvolvido no capítulo 2.

Fase	Nome	Tempo de CPU	%
1ª	Detecção de bordas	0,491 s	10,31%
2ª	Afinamento	0,664 s	13,94%
3ª	Transformada de Hough	2,031 s	42,65%
4ª	Classificação das retas	0,932 s	19,58%
5ª	Cálculo das propriedades	0,006 s	0,13%
6ª	Ajuste das retas	0,633 s	13,27%
7ª	Calibração	0,002 s	0,04%
	TOTAL	4,759 s	

Tabela 4.1. Tabela de tempos do algoritmo serial, apresentado no capítulo 2, em uma Sparc-Station, da Sun MicroSystems.

O algoritmo implementado para obter-se os dados da tabela 4.1 utiliza o algoritmo de detecção de Sobel e a transformada clássica de Hough na 3ª fase, como descrita na seção 2.2.3, com resolução do

espaço de Hough de 256x326. Os resultados da tabela 4.1 foram obtidos tirando-se a média ponderada do processamento de 10 imagens com resolução de 200x256x8, com complexidades variando entre 2000 e 4000 pixels de borda.

Analisando-se o fluxograma da versão serial (figura 2.3), nota-se que há um sequenciamento natural das diversas fases do algoritmo, mostrando também uma relação de dependência entre as fases. Este fluxograma - também considerado uma instância do Paradigma de Visão Computacional (BALLARD e BROWN, 1982) - deve ser obedecido tanto pelo algoritmo serial como pelo algoritmo paralelo.

Da tabela 4.1 nota-se que a transformada de Hough é a fase que toma mais tempo de CPU (≅42%) e essa influência da transformada de Hough torna-se mais notável se as 2ª e 6ª fases não forem utilizadas - por serem opcionais segundo a seção 2.2 - elevando o percentual para um valor em torno de 57%. Das observações feitas acima, duas conclusões podem ser tiradas:

1) A exploração do paralelismo inter-fases é complexa, mas o paralelismo intra-fases é possível.
2) A transformada de Hough é a fase que consome mais tempo e, portanto, deve ser prioritariamente paralelizada.

As seções que se seguem apresentam os aspectos de paralelização das etapas do algoritmo em duas arquiteturas paralelas diferentes: MIMD fracamente acoplada e SIMD. A 5ª e a 7ª fases do algoritmo não foram consideradas para a paralelização por contribuirem muito pouco no tempo total do algoritmo.

4.2 - (1ª fase) - Detecção de Bordas

Conforme visto na seção 2.2.1, os operadores "matemáticos" para detecção de bordas baseiam-se na operação de convolução bi-dimensional da função (discreta) imagem com uma função (discreta) chamada filtro ou máscara. Nesta seção serão apresentados dois tipos de implementação da operação de convolução. A seção 4.2.1 apresenta o desenvolvimento de um algoritmo **sistólico** para a convolução 2D. A partir da convolução

unidimensional, faz-se a extensão "bottom-up" para a convolução bi-dimensional genérica e, na sequência, mostra-se como a implementação dos operadores Laplaciano e Sobel são executadas com esta técnica. A seção 4.2.2 mostra, através da implementação **masssivamente paralela** na Connection Machine (introduzida na seção 3.1.2), a facilidade de se implementar a operação de convolução 2D dispondo-se de um processador para cada pixel. É mostrada a implementação do operador Laplaciano com esta técnica.

4.2.1 - Implementação Sistólica

Esta paralelização do algoritmo de convolução em duas dimensões baseia-se no modelo sistólico, intoduzido por KUNG (1982). O algoritmo é sistólico pois o paralelismo temporal é aproveitado através do "bombeamento rítmico", dos pixels de entrada para dentro dos blocos simples que compõem o algoritmo. O algoritmo também é modular, podendo ser expandido para utilizar máscaras de tamanho arbitrário ou concatenando-se módulos, conforme será apresentado na sequência.

Os pixels entram no módulo de convolução ordenados por linhas, ou seja, na ordem da varredura da imagem. E saem do módulo ordenados da mesma maneira. Portanto esse algoritmo simula um processador de imagens colocado entre a saída do conversor A/D e a entrada do "frame buffer" em uma placa de aquisição de imagens.

A figura 4.1 mostra o princípio da convolução unidimensional sistólica. Há dois fluxos de dados em direções opostas, conforme a figura 4.1: O **fluxo de entrada** conduz os pixels que vêm diretamente da câmera. O **fluxo de saída** conduz os pixels já convoluídos. Os blocos representam elementos da máscara de convolução.

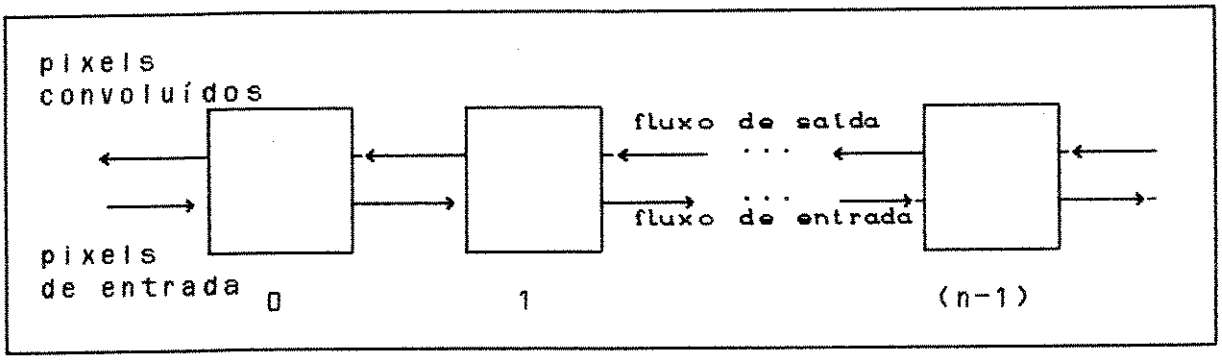
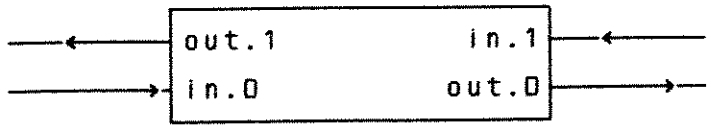


Figura 4.1. Convolução unidimensional sistólica com máscara de tamanho n, KUNG (1982).

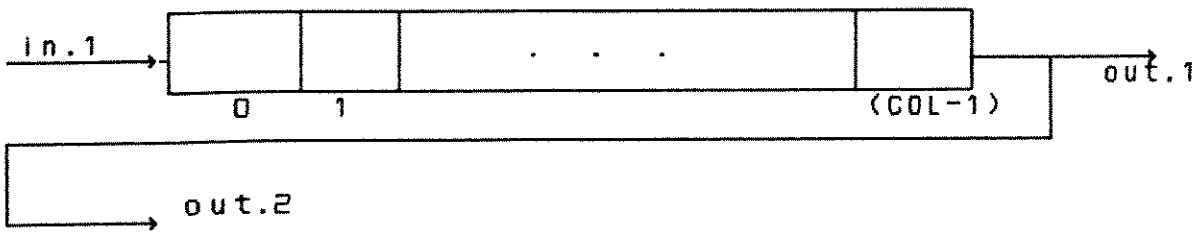
O algoritmo bi-dimensional é uma extensão do algoritmo unidimensional apresentado na figura 4.1. Ele possui 2 módulos básicos: O módulo de "multiplicação" (detalhado na figura 4.2) e o módulo de "linha de atraso" (detalhado na figura 4.3). Cada bloco i ($0 \geq i \geq n-1$) da figura 4.1 é um bloco de multiplicação. Para cada pixel do fluxo de entrada e a cada pixel do fluxo de saída que entram simultaneamente no bloco de multiplicação, executa-se uma função de multiplicação do pixel, que entrou pelo fluxo de entrada, por um peso armazenado (w_i) no bloco e uma soma com o pixel que entra no sentido inverso, colocando-se o resultado no fluxo de saída. A máscara unidimensional tem dimensão k e, portanto, são necessários k blocos de multiplicação para se fazer a convolução.




```
PROC mult(CHAN OF INT in.0, in.1, out.0, out.1, INT w)
  INT  a,b,c :
  SEQ
    out.1 ! 0
    WHILE ( a <> fim.de.dados.i )
      SEQ
        PAR
          in.0 ? a
          in.1 ? b
        IF
          a <> fim.de.dados.i
            SEQ
              c := ((a * w) + b)
              PAR
                out.0 ! a
                out.1 ! c
          TRUE
            SKIP
:
```

Figura 4.2. O bloco de multiplicação da convolução sistólica, e sua implementação em OCCAM.

Deve-se assegurar que os pixels estejam nos k blocos de multiplicação no momento certo. Há várias maneiras de se fazer isto. Uma delas é utilizar linhas de atraso. A linha de atraso é um registrador de deslocamento do comprimento de uma linha da imagem. A figura 4.3 apresenta o bloco de linha de atraso.




```
PROC  linha.de.atraso(CHAN OF BYTE in.1,out.1,out.2)
  INT i :
  [COL]BYTE linha :
  SEQ
    SEQ i = 0 FOR COL
      linha[i] := 0 (BYTE)
    WHILE (linha[0] <> fim.de.dados)
      SEQ
        PAR
          out.1 ! linha[0]
          out.2 ! linha[0]
        SEQ i = 0 FOR (COL-1)
          linha[i] := linha[i+1]
        in.1 ? linha[COL-1]
    PAR
      out.1 ! fim.de.dados
      out.2 ! fim.de.dados
    SEQ i = 0 FOR (COL-1)
      out.2 ! 0 (BYTE)
:
```

Figura 4.3. O bloco de linha de atraso da convolução sistólica, e sua implementação em OCCAM.

O bloco de linha de atraso assegura, através de deslocamento síncrono de seus registros, que os blocos de multiplicação recebam os k pixels vizinhos em um instante especificado. A cada pixel que entra por um lado da linha de atraso, o último pixel da fila é enviado pelos dois canais de saída.

Utilizando-se os dois blocos descritos (figuras 4.2 e 4.3) constrói-se o algoritmo de convolução em duas dimensões. A figura 4.4 apresenta o diagrama de blocos desse algoritmo.

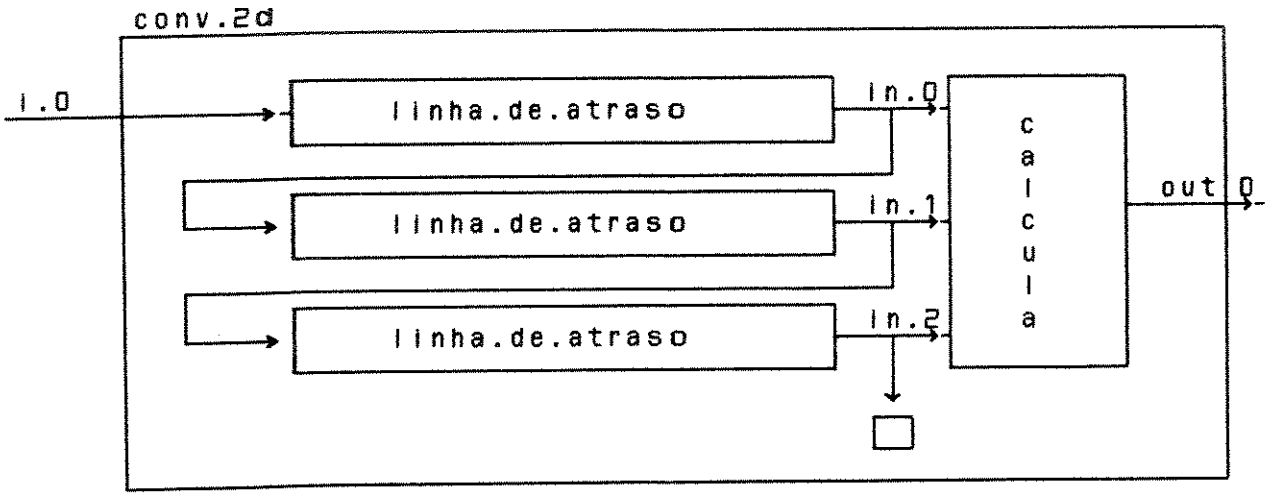


Figura 4.4. O diagrama do processo de convolução 2D.

O bloco da figura 4.4, denominado "calcula", é composto de três blocos de convolução unidimensional e um bloco de soma. Este último é responsável pela acumulação simples dos resultados que são "bombeados" pelos outros três. A figura 4.5 detalha este feito. Após a soma pode-se eventualmente efetuar uma binarização da imagem ajustando-se um valor de limiarização.

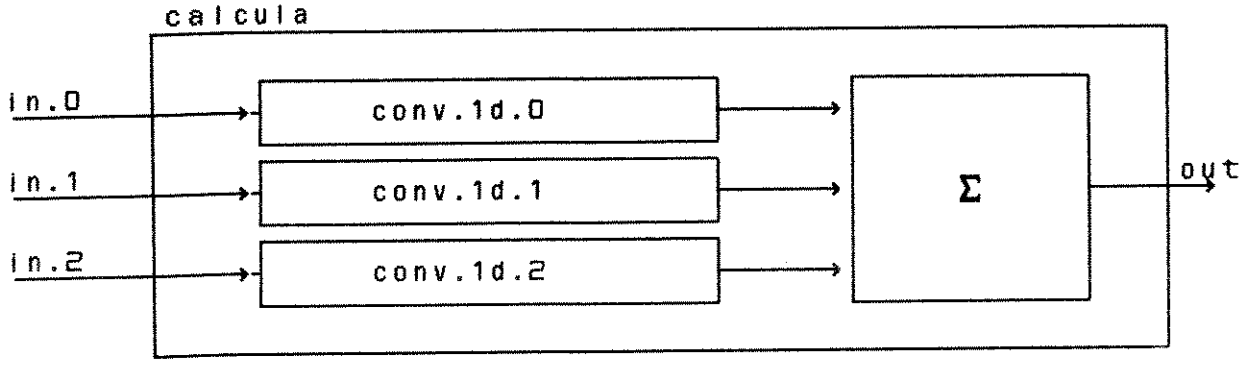


Figura 4.5. O bloco calcula. As entradas in.0, in.1 e in.2 são as mesmas da figura 4.4.

Até este momento mostrou-se apenas uma implementação sistólica do algoritmo de convolução bi-dimensional. Na sequência será mostrado como fazer para implementar os operadores "matemáticos" de detecção de bordas sobre esta implementação descrita até agora nesta seção. Para implementar-se operadores que utilizam máscaras de dimensão 3x3 -

como, por exemplo, o Laplaciano - são necessários k=9 blocos de multiplicação e três blocos de linha de atraso. Cada um dos três blocos de convolução unidimensional sistólica deve possuir três blocos de multiplicação e um bloco para receber os pixels do fluxo de entrada e enviar zeros pelo fluxo de saída, conforme figura 4.6.

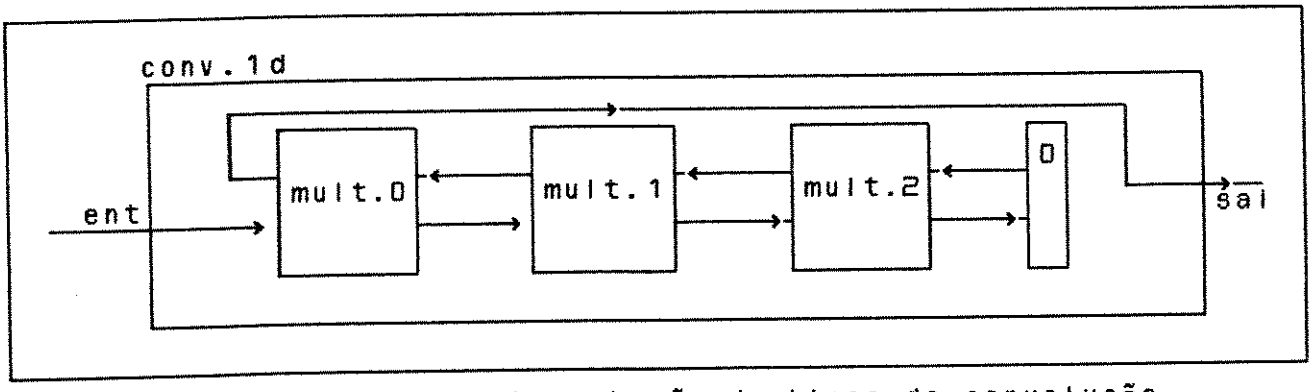


Figura 4.6. Implementação do bloco de convolução unidimensional 1x3, que faz parte da 1ª linha de uma máscara 3x3.

De fato a figura 4.6 é uma particularização da figura 4.1, para máscaras 3x3. Para utilizar este algoritmo de convolução bidimensional sistólico descrito com máscaras de **qualquer** tamanho, basta dimensionar os parâmetros de projeto para implementar o algoritmo desejado. Os parâmetros a serem dimensionados são: o tamanho da máscara, os coeficientes da máscara (parâmetro w da figura 4.2), o tamanho da imagem de entrada e o valor do limiar.

As figuras 4.4, 4.5 e 4.6 sugerem um método direto para a implementação de detectores de borda que utilizam uma máscara 3x3, como o Laplaciano, visto na seção 2.2.1. Para isto, basta atribuir os 9 coeficientes do operador Laplaciano nos 9 blocos de multiplicação (parâmetro w da figura 4.2).

Na sequência será mostrado como foi implemetado o algoritmo de detcção de bordas de Sobel a partir dos blocos do algoritmo de convolução 2D genérico acima descrito. O operador de Sobel utiliza duas máscaras, conforme visto na seção 2.2.1. Desse modo, é necessário um bloco para duplicar a entrada e outro para combinar os resultados da saída. O diagrama do algoritmo de Sobel implementado está na figura 4.7. O bloco "conv.2d.h" executa a convolução com a máscara da direção horizontal.

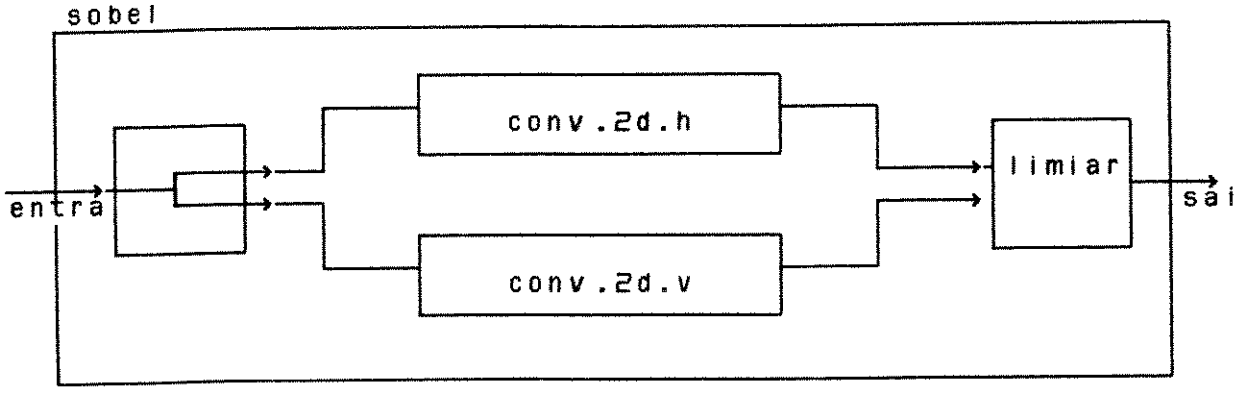


Figura 4.7. Diagrama da implementação paralela do operador de Sobel

A implementação dessa operação de detecção de bordas utilizando a o modelo sistólico possui algumas características importantes:

- Os blocos do algoritmo podem ser mapeados em hardware. Os blocos funcionais satisfazem algumas exigências de projeto em VLSI: (a) Possui um fluxo de dados simples e regular, (b) possibilita o "pipelining" de dados, (c) possuem uma estrutura hierárquica e (d) podem ser implementados em dispositivos de baixo custo. De fato a literatura mostra algumas implementações, em hardware, semelhantes ao algoritmo descrito: ver RUETZ e BRODERSEN (1988) e CASTANHO (1990).
- Na programação de algoritmos do tipo sistólico, uma linguagem paralela permite que o paralelismo seja explicitado dentro do programa, facilitando sua correção e sincronização. O algoritmo implementado - em OCCAM - tem por volta de 40 blocos (PROCs) rodando concorrentemente. A programação deste algoritmo em uma linguagem serial é muito difícil.

4.2.2 - A Implementação Massivamente Paralela

Esta paralelização do algoritmo da convolução em duas dimensões baseia-se no modelo paralelo de dados, intoduzido por HILLIS e STEELE (1986). Este algoritmo de convolução, que foi implementado na Connection Machine, utiliza os roteadores de dados para comunicar os níveis de cinza entre os 4 vizinhos.

Como a convolução é uma operação local, a eficiência deste algoritmo é muito alta, ou seja, todos os processadores trabalham todo o tempo. A figura 4.8 mostra a implementação simplificada do algoritmo.

```
shape [LIN][COL]frame ;
unsigned char:frame image_i, image_o;
    ⋮
image_o =  [j-1][i]image_i +          /* oeste */
           [j][i-1]image_i +          /* norte */
           [j][i+1]image_i +          /* sul */
           [j+1][i]image_i            /* leste */
         -4*[j][i]image_i ;
```

Figura 4.8. Implementação simplificada da convolução com a máscara da fig 2.5a, em linguagem C* (TMC, 1990).

Nota-se a ausência dos comandos "for" (ou "PAR"), pois todas as comunicações e cálculos são feitos em paralelo. Neste caso, o algoritmo tem complexidade O(k), onde k é o tamanho da máscara.

O algoritmo de Sobel foi implementado de maneira semelhante à do Laplaciano. Neste caso duas imagens intermediárias - imagem_Sx e imagem_Sy - foram criadas para armazenar temporariamente os resultados das convoluções com as duas máscaras do detector de Sobel. Após as convoluções com as duas máscaras, faz-se a soma dos módulos dos resultados, também em paralelo, de todos os pixels das duas imagens intermediárias e armazena-se o resultado em uma imagem de saída.

No caso do operador de Marr, que possui máscaras bem maiores, há outras técnicas utilizando as operações de **scan** (seção 3.3.2) que são mais eficientes, ver LITTLE, BLELLOCH e CASS (1989).

4.3 - (2ª fase) - Afinamento

O algoritmo de afinamento descrito na seção 2.2.2 pode ser paralelizado, pois o novo valor dado a um pixel na iteração (n) depende apenas de seu próprio valor e do valor de seus oito vizinhos na iteração (n-1).

O algoritmo de afinamento pode ser implementado aplicando-se o paralelismo massivo em uma arquitetura SIMD, conforme é apresentado na figura 4.9a, ou utilizando-se uma arquitetura MIMD, conforme é

apresentado na figura 4.9b. De acordo com a figura 4.9a, a cada iteração, uma série de pixels é apagada, em paralelo. O comando "where(imagem)" ativa apenas os pixels pretos, que correspondem à borda. O comando "where(C)" seleciona apenas os pixels que ainda não foram apagados pelo algoritmo "afinamento()", descrito na seção 2.2.1. E, finalmente, o teste de parada "while(|=C)" faz um OR global da condição C, indicando, a cada iteração, quantos processadores ainda estão ativos.

```
{  unsigned char:frame imagem ;
   bool:frame C;
    C = TRUE ;
    do {
     where(imagem)
     {
       where(C)
        {
          afinamento();
          C = (I==M);
        }
     }
   }while (|= C);
}
```

(a)

```
PAR I = 0 FOR LIN
  PAR J = 0 FOR COL
    WHILE (cont[I][J])
      IF
        (imagem[I][J])
          SEQ
            ...  afinamento
            IF
              (não.modI)
                cont[I][J]=FALSE
              TRUE
                SKIP
        TRUE
          SKIP
```

(b)

Figura 4.9. Implementação do algoritmo de Afinamento (a) na Connection Machine e (b) em uma malha de transputers.

Como se observa do algoritmo descrito na figura 4.9b, a cada iteração, pixels que não pertencem ao "esqueleto" são apagados em paralelo. O comando "IF (imagem [I][J])" seleciona apenas os pixels pretos que correspondem à borda. O comando "afinamento()" executa o algoritmo descrito na seção 2.2.2. E, finalmente, o teste de parada "IF (não.modI)" e a variável "cont" indicam, a cada iteração, quantos processadores ainda estão ativos.

4.4 - (3ª fase) - Identificação das Bordas do Objeto

O tempo total necessário para o algoritmo sequencial é a soma do tempo de computação dos dois processos seguintes: (1) Varrer a imagem N x N para obter as coordenadas dos pixels de borda. Este processo

requer $O(N^2)$ unidades de tempo. (2) Para cada pixel da borda, computar o valor ρ para cada um dos N_θ valores de θ. A complexidade de tempo total desta fase é portanto

$$O(N^2+N_\theta)$$

Na sequência, serão descritas as implementações paralelas da transformada de Hough em dois tipos de arquiteturas diferentes: MIMD e SIMD.

4.4.1 - Utilizando Arquiteturas MIMD

As soluções de problemas através de algoritmos para processadores MIMD fracamente acoplados podem ser classificadas de acordo com o modelo do paralelismo explorado: **Paralelismo Espacial** e **Paralelismo Temporal**. As soluçãoes que exploram o paralelismo temporal são aqueles que fazem uso de técnicas de pipelining e técnicas sistólicas, utilizando chips VLSI full-custom, como o algoritmo sistólico descrito em LI, PAO e JAYAKUMAR (1989).

Alguns algoritmos de Visão Computacional de baixo nível possuem características de paralelismo espacial. Um exemplo é o algoritmo de detecção de bordas de Marr, descrito na seção 2.2.1. Pelo modelo do paralelismo espacial a imagem de entrada pode ser sub-dividida em sub-imagens, que são distribuídas a vários processadores, de modo que o processamento possa ser feito nas várias sub-imagens de maneira simultânea.

O paralelismo espacial da transformada de Hough pode ser aproveitado de diversas maneiras. Pode-se particionar tanto a imagem quanto o espaço Hough, ou ambos, de modo a se conseguir um melhor desempenho. Foram estudadas algumas implementações paralelas da transformada de Hough clássica e da transformada de Hough utilizando informação direcional, que já foram descritas na seção 2.2.3 e foram desenvolvidas seis opções de implementação de algoritmos paralelos para se fazer um estudo prático comparativo a respeito de seu desempenho, speedup e eficiência.

As implementações paralelas desenvolvidas para a malha de transputers têm algumas características em comum. A primeira delas é que há um processo **mestre**, responsável pelo início do envio dos dados e que concentra os resultados finais. Este processo é sempre alocado

no transputer base, também chamado de módulo-raiz, e é aquele que se comunica com o "front-end". Há também os processos **escravos**. Estes são compostos de dois processos: o **algoritmo de comunicação** e o **algoritmo da tarefa**. O primeiro depende da solução paralela utilizada e da topologia adotada. O segundo depende apenas da aplicação, executando os cálculos, armazenando dados, etc. Na apresentação das soluções, que se segue, apenas o algoritmo da tarefa será mostrado. Os algoritmos de comunicação já foram introduzidos na seção 3.3.1.

Serão descritas na sequência as diferentes implementações da transformada de Hough na rede de transputers, conforme mostra a tabela 4.2. Segundo esta tabela, as três primeiras soluções paralelas - (a), (b) e (c) - referem-se ao algoritmo clássico da Transformada de Hough e os duas últimas - (d) e (e) - referem-se ao algoritmo da Transformada de Hough que utiliza a informação direcional. A solução (f) pode ser utilizada com qualquer um dos dois algoritmos.

Tipo da Transformada de Hough	Solução Proposta	
"Clássica"	(a)	Partição do Espaço Hough
	(b)	Partição da Imagem
	(c)	Partição da Imagem e do Espaço Hough
"Direcional"	(d)	Partição da Imagem e do Espaço Hough
	(e)	"Processor Farm"
"Clássica ou "Direcional"	(f)	Partição da Imagem

Tabela 4.2. Resumo das soluções propostos para a paralelização da Transformada de Hough em arquiteturas MIMD.

Transformada de Hough "clássica" (T.H.C.)

Para a tarefa de se encontrar linhas retas, a transformada "clássica" de Hough é utilizada na impossibilidade de se obter informação sobre a derivada direcional. A transformada "clássica" de Hough, assim como a Transformada Generalizada de Hough (introduzida

por BALLARD, 1981), têm sido utilizadas para as mais diversas aplicações onde a informação direcional não pode ser obtida, conforme ILLINGWORTH e KITTLER (1988): Determinação de parâmetros de movimento de corpos rígidos, reconhecimento de padrões genéricos, reconhecimento de padrões em "range images", compressão de dados, etc.

As técnicas desenvolvidas nesta sessão procuram explicitar o paralelismo inerente à transformada "clássica" de Hough.

T.H.C. - SOLUÇÃO (a) Partição do Espaço Hough

A idéia básica desta solução é cada nó processador escravo conter uma partição - em θ - do espaço Hough. Desse modo, cada processador trabalha em um intervalo menor de ângulos θ, diminuindo o tempo de processamento.

De acordo com a descrição simplificada da figura 4.10, os pixels de borda são detectados no mestre e transmitidos para os escravos e estes recebem **todos** os pixels de borda, através da malha de interconexão, e executam o cálculo da transformada de Hough apenas para um intervalo de valores de θ referentes ao processador. Ao final do processamento, cada escravo faz uma varredura de sua partição do espaço Hough e transmite ao processador mestre, as nuvens de pontos acima de um limiar pré-fixado.

```
PAR
  SEQ                -- mestre
    PAR
      ...  transmissão de pacote com todos os pixels de borda
      ...  recepção de pacote de picos e atualiz. do E. Hough
    ...  detecção de picos
    ...  continuação do algoritmo
  PAR i = 0 FOR n.de.escravos          --  escravos
    SEQ
      SEQ
        ...  recepção do pacote
        ...  cálculo T. de Hough no sub-espaço Hough
      ...  varredura do sub-espaço Hough e limiarização
      ...  transmissão de pacote de picos para o mestre
```

Figura 4.10. A solução (a), para a Transformada Clássica de Hough.

O volume de dados envolvidos na comunicação pode ser quantificado da seguinte maneira: O mestre envia (B*n) pares (x,y) referentes aos B pixels de borda detectados. Esta operação equivale a um "broadcast" dos B pixels de borda detectados para os n escravos. Cada escravo recebe B pares (x,y) e calcula $B*(N_\theta/n)$ pares (θ,ρ) correspondentes à sua partição do espaço Hough.

Cada escravo deve fazer $B*(N_\theta/n)$ atualizações na sua partição do espaço Hough. A quantidade mínima de memória de dados necessária em cada escravo depende do tamanho do sub-espaço Hough, ou seja, $(N_\theta/n).N_\rho$, onde n é o número de escravos, N_θ vale $\pi/\Delta\theta$ e N_ρ é a discretização em ρ.

T.H.C. – SOLUÇÃO (b) Partição da Imagem

A idéia básica desta solução é cada nó escravo conter uma partição do conjunto total de pixels de borda e uma cópia do espaço Hough completo. Cada escravo atualiza seu espaço Hough e envia os acumuladores com votos para o escravo vizinho, o qual atualiza seu próprio espaço Hough, até que o mestre receba o espaço Hough completo.

```
PAR
  SEQ              -- mestre
    PAR
      ...   transmissão de pacotes de pixels de borda
      ...   recepção de pacote de picos
      ...   continuação do algoritmo
  PAR i = 0 FOR n.de.escravos          -- escravos
      SEQ
        ...   recepção de pacote com pixels
        ...   cálculo T. de Hough
        SEQ j = 0 FOR ( n.de.pacotes - 1 )
          SEQ
            PAR
              ...   recepção de pares (θ,ρ) de outros escravos
              ...   transmissão de pares (θ,ρ) para escravos
            ...   atualização do espaço Hough
        ...   devolve picos para o mestre
```

Figura 4.11. A solução (b), para a Transformada Clássica de Hough.

Os pixels de borda são detectados no mestre e transmitidos, em pacotes, para os escravos. Diferentes pacotes de pixels de borda chegam, através da malha de interconexão, até os escravos. Estes calculam um conjunto de pares (θ,ρ) para cada pixel de borda, atualizando seu próprio espaço Hough. Em seguida são trocados pacotes de (θ,ρ) entre os escravos e uma nova atualização do espaço Hough é feita. Ao final do processamento, o mestre recebe uma cópia do espaço Hough completo. Um resumo simplificado do algoritmo é apresentado na figura 4.11.

A quantidade de memória de dados necessária em cada escravo é resultante da soma do número de pixels de borda do pacote (B/p) e da memória destinada ao espaço Hough, que vale $N_\theta x N_\rho$, onde B é o número de pixels de borda e p é o número de pacotes. Como em geral o tamanho do espaço Hough é superior ao tamanho da imagem, esta solução é o que mais consome memória em relação aos outros.

T.H.C. – SOLUÇÃO (c) Partição do Espaço Hough e da Imagem

A idéia básica desta solução é unir as características das soluções (a) e (b). Procura-se fazer os escravos trocarem pacotes de dados entre si – como na solução (b) – e fazer cada escravo conter apenas uma partição do espaço Hough (em θ), como na solução (a). Esta solução assemelha-se ao de GUERRA e HAMBRUSH (1989), que foi implementado em uma arquitetura SIMD.

Os pixels de borda são detectados no mestre e transmitidos, em pacotes, para os escravos. Diferentes pacotes de pixels de borda chegam, através da malha de interconexão, para os escravos. Cada nó escravo atualiza, a partir de seu pacote de pixels de borda, sua partição do espaço Hough. Em seguida, retransmite o pacote de pares (x,y) para os demais escravos e uma nova atualização do sub-espaço Hough é feita. Ao final do processamento, cada escravo faz uma varredura de sua partição do espaço Hough e transmite ao processador mestre, as nuvens de pontos acima de um limiar pré-fixado. Esta solução resumida é apresentado na figura 4.12.

```
PAR
  SEQ                -- mestre
    PAR
      ...  transmissão de pacotes de pixels de borda
      ...  recepção de pacotes de picos
    ...  detecção de picos
    ...  continuação do algoritmo
  PAR i = 0 FOR n.de.escravos          -- escravos
    SEQ
      SEQ j = 0 FOR (n.de.pacotes-1)
        SEQ
          ...  recepção de pacote de pixels de borda
          ...  cálculo da T. de Hough
          ...  transmissão de pacotes (x,y) para
                                o escravo (i+1)\n.de.escravos
      ...  varredura de sub-espaço Hough e limiarização
      ...  transmissão de pacote com picos para o mestre
```

Figura 4.12. A solução (c), para a Transformada Clássica de Hough.

A quantidade de dados envolvida na comunicação, para esta solução, pode ser analisada como se segue: O mestre envia B pares (x,y) para n escravos. Portanto cada escravo recebe (B/n) pares (x,y) referentes aos pixels de borda detectados no mestre. Cada escravo, por sua vez, calcula (BN_θ/N^2) pares (θ,ρ) e acumula-os em suas respectivas partições do espaço Hough. Em seguida repete-se (n-1) vezes a seguinte sequência: 1) envia pacote com (B/n) pares (x,y) para outro escravo; 2) recebe outro pacote com (B/n) pares (x,y) de outro escravo; 3) calcula (BN_θ/N^2) pares (θ,ρ) e faz as respectivas acumulações.

A quantidade de memória de dados necessária em cada escravo é resultante da soma do número de pixels de borda do pacote (B/p) com a memória destinada à partição do espaço Hough, que vale $(N_\theta/n)xN_\rho$, onde B é o número de pixels de borda, n é o número de escravos e p o número de pacotes.

A tabela 4.3 apresenta uma comparação entre computação e comunicação nas soluções (a), (b) e (c). Computação refere-se à quantidade de cálculos de pares (θ,ρ), através da equação 2.1. Comunicação refere-se à quantidade de pares (x,y) ou (θ,ρ) que atravessam a rede de processadores.

Para se comparar as soluções (a) e (c), dois aspectos devem ser analisados: A topologia adotada e a possibilidade de superposição das comunicações de mensagens. A topologia adotada é um parâmetro

importante, pois o processo mestre centraliza o fornecimento inicial dos dados e, consequentemente, há uma sobrecarga de mensagens nos links próximos a ele. Topologias que têm pequeno diâmetro podem diminuir o efeito desta sobrecarga. A possibilidade de sobreposição de comunicação de mensagens consiste na comunicação simultânea e ordenada de pacotes de dados entre os processadores escravos. A ordenação, utilizada na solução (c), permite que se possa analisar e organizar o padrão das comunicações de mensagens pela topologia adotada.

"Transformada Clássica de Hough"	Solução (a)	Solução (b)	Solução (c)
Comunicação MESTRE→ESCRAVO	B*n	B	B
Comunicação ESCRAVO→ESCRAVO	0	——	(n-1)*B/n
Computação/ESCRAVO	$B*N_\theta/n$	$B*N_\theta/n^2$	$B*N_\theta/n$

Tabela 4.3. Computação x Comunicação para as soluções paralelas (a), (b) e (c). B é o número de pixels de borda detectados, N_θ é a discretização em θ e n é o número de processadores escravos.

Como pode ser visto na tabela 4.3, o número de computações nas soluções (a) e (c) é constante em ambas. Isto acontece pois, sob o ponto de vista da partição do espaço Hough, as soluções (a) e (c) são idênticos. Percebe-se também que a solução (a) apresenta maior congestionamento de mensagens nas proximidades do mestre, pois necessita comunicar n vezes mais dados que a solução (a). A solução (c) utiliza a comunicação entre escravos para descongestionar os links próximos ao mestre. Nota-se que o custo de comunicação da solução (c) é menor que o da solução (a). Esta vantagem é dada pela razão

$$\frac{Bn}{2*B-B/n} = \frac{n^2}{2n-1}$$

que mostra que, quanto mais escravos, mais distibui-se as comunicações e, portanto, menor o congestionamento próximo ao mestre.

Transformada de Hough "Direcional" (T.H.D)

Como foi visto na seção 2.2.3, a transformada de Hough que utiliza informação direcional, usa um intervalo de k ângulos θ (centrado no valor Φ do gradiente) para determinar quais os acumuladores (θ,ρ) que serão atualizados. Como os valores de θ_k são determinados dinamicamente - ou seja, para cada pixel de borda define-se um intervalo de ângulos θ - não é possível planejar-se uma partição justa do trabalho entre os processadores como foi feito na transformada "clássica" de Hough e mostrado na tabela 4.3. Este fato geralmente é fonte de ineficiências das implementações paralelas da transformada de Hough que utilizam informação direcional.

T.H.D. - SOLUÇÃO (d) Partição do Espaço Hough e da Imagem

A idéia básica deste algoritmo é a mesma do algoritmo (c). Porém, como somente é necessário calcular k pares (θ,ρ) para cada ponto de borda, o algoritmo é alterado ligeiramente. Nesta solução, o par (θ_k,ρ) caminha pela malha até encontrar o processador correspondente ao sub-espaço Hough contendo o ângulo θ_k.

Primeiramente, os pixels de borda e o gradiente são detectados no mestre e transmitidos, em pacotes, para os escravos. Cada um dos escravos recebe um pacote, de tamanho (B/p), contendo a tripla (x,y,θ); onde p é o número de pacotes, x e y são as coordenadas dos pixels de borda e θ é a orientação do gradiente no ponto (x,y). Em seguida os escravos calculam os k.(B/p) pares (θ,ρ) onde k é o número de ângulos que devem ser calculados, conforme explicado na seção 2.2.3. Os ângulos θ calculados servem como endereço para o processador que contiver o acumulador (θ,ρ). A mensagem contendo o par (θ,ρ) é enviada para seu destino. Ao final do processamento, após uma limiarização, os escravos transmitem de volta para o mestre os picos das nuvens de pontos detectados.

```
PAR
  SEQ                -- mestre
    PAR
      ...   transmissão de pacotes de (x,y,θ)
      ...   recepção de pacotes de picos e atualiz. do E. Hough
    ...   detecção de picos
    ...   continuação do algoritmo
  PAR i = 0 FOR n.de.escravos          -- escravos
    SEQ
      ...   recepção de pacote de (x,y,θ)
      ...   cálculo da T. de Hough
        PAR i = 0 FOR n.de.pacotes
          ...  transmissão de pacotes de (θ,ρ)
          ...  recepção de pacotes de (θ,ρ) e atualização
                                (ou não) do sub-espaço Hough
      ...   varredura de sub-espaço Hough e limiarização
      ...   transmissão de pacote com picos detectados
                                          para o mestre
```

Figura 4.13. A solução (d), para a Transformada Direcional de Hough.

A quantidade de memória de dados necessária em cada escravo é resultante da soma do número de pares (θ,ρ) calculados, que vale k(B/p) com a memória destinada à partição do espaço Hough, que vale $(N_\theta/n)xN_\rho$, onde B é o número de pixels de borda, n é o número de escravos e p o número de pacotes.

T.H.D. - SOLUÇÃO (e) "Processor Farm"

Para muitas aplicações em computação paralela, é util criar programas aplicativos que se autoconfiguram para rodar em qualquer número de processadores. Tal aplicação irá, **teoricamente**, rodar mais rápido, quanto mais processadores forem adicionados à rede, sem recompilação nem reconfiguração. Esta aceleração não é conseguida na prática devido ao fenômeno da saturação.

Na técnica de **"processor farm"**, uma aplicação é codificada como um processo mestre que quebra o problema em partes pequenas e independentes, chamadas pacotes de trabalho ("work packets"), que são processados por um número qualquer de processos trabalhadores anônimos, denominados processos escravos. Os pacotes de trabalho são automaticamente distribuídos através de uma rede arbitrária de

processadores pelo software de roteamento de mensagens, denominado algoritmo de comunicação. Todas as tarefas trabalhadoras devem rodar o mesmo software. Cada processo escravo simplesmente aceita pacotes de trabalho, processa-os e envia de volta pacotes de resultados ("result packets") através do mesmo roteador. Um processo escravo é simplesmente um loop sequencial: lê o pacote de trabalho, processa-o e envia o pacote de resultados. Para facilitar a descrição da solução assume-se que o processo mestre e cada processo escravo estão alocados em nós processadores diferentes.

```
PAR
  SEQ              -- mestre
    PAR
      ...  transmissão de pacotes de (x,y,θ)
      ...  recepção de pacotes de picos e atualiz. do E. Hough
    ...  detecção de picos
    ...  continuação do algoritmo
  PAR i = 0 FOR n.de.escravos          -- escravos
    SEQ
      ...  recepção de pacote de (x,y,θ)
      ...  cálculo da T. de Hough
      ...  envia pacote de (θ,ρ) para o mestre
```

Figura 4.14. A solução (e), para a Transformada Direcional de Hough.

A idéia desta solução é a seguinte: o conjunto de pixels de borda é subdividido em p pequenos pacotes (x,y,θ), de tamanho (B/p), onde B é o número de pixels de borda e p é o número de pacotes. Geralmente, faz-se b $\gg$ n, onde n é o número de processos escravos. O mestre é o único a conter o espaço Hough. Os escravos apenas calculam seus resultados e os enviam de volta para o mestre. Um resumo desta solução pode ser vista na figura 4.14.

Primeiramente, os pixels de borda e o gradiente são detectados no mestre e transmitidos, em pacotes, para os escravos. Cada um dos escravos recebe um pacote, de tamanho (B/p), contendo a tripla (x,y,θ); onde p é o número de pacotes, x e y são as coordenadas dos pixels de borda e θ é a orientação do gradiente no ponto (x,y). Em seguida os escravos calculam os k.(B/p) pares (θ,ρ) onde k é o valor de ângulos que devem ser calculados, conforme exposto na seção 2.2.3. Pacotes de k pares de (θ,ρ) são enviados de volta para o mestre, que atualiza seu espaço Hough. Quando o mestre recebe os dados, libera um pacote de triplas (x,y,θ) da fila de pacotes de de saída.

A principal característica da soluçã denominada "Processor Farm" é que poucos processadores ficam por pouco tempo inativos. Esta característica é mais visível quando se percebe que, nos algoritmos de (a) a (d), a comunicação entre o mestre e a malha de escravos é feita em tempos separados: Primeiramente o mestre envia os dados e somente após o processamento é que os resultados são devolvidos ao mestre. Na soluçã "Processor Farm" esta comunicação é sobreposta, possibilitando a simultaneidade de comunicação "mestre → escravos" e "escravos → mestre". Diz-se, no caso da solução "Processor Farm", que há um balanceamento dinâmico de comunicação entre o mestre e os escravos.

Transformada de Hough "Clássica" ou "Direcional" (T.H.C. ou T.H.D.)

Arquiteturas MIMD fortemente acopladas podem simular algoritmos desenvolvidos para rodarem em qualquer outra arquitetura paralela. Desse modo, apresenta-se na sequência um algoritmo genérico da transformada de Hough (T.H.C. ou T.H.D.) programado de modo a ser implementado em uma arquitetura MIMD fortemente acoplada.

T.H.C. ou T.H.D. - SOLUÇÃO (f) Partição da Imagem

A idéia básica desta solução é cada nó escravo receber uma partição do conjunto total de pixels de borda e apenas o mestre conter o espaço Hough. Os escravos calculam pares de (θ,ρ), e os enviam para o mestre, que atualiza o espaço Hough.

Ambos os algoritmos da transformada de Hough podem ser utilizados, diferindo apenas na mensagem lida pelos escravos: (x,y) no algoritmo clássico e (x,y,θ) no outro. Pacotes diferentes de pixels de borda chegam, através da malha de interconexão, até os escravos. Estes calculam um conjunto de pares (θ,ρ) para cada pixel de borda e os transmitem de volta para o mestre.

```
                    shared hs[512][512] ;
                    shared imagem[256][256];

slave(n)                                        master(procs)
  ⋮                                               ⋮
for(i=0;i<256;i++)                              for(i=0;i<procs;i++)
 for(j=0;j<256;j++)                              thread_id[i] =
  if(imagem[i][j])                                cthread_fork(slave,n_ang*i);
   for(θ=n;θ<(n+180/p);θ++)
    {                                           for(j=0,j<procs;j++)
    ρ = i.cos[θ] + j.sen[θ] ;                     total = total +
    atualiza_hs(θ,ρ)                               cthread_join(thread_id[j]);
    }
  ⋮                                               ⋮

                    atualiza_hs(θ,ρ)
                      ⋮
                    mutex_lock(lock);
                    hs[θ][ρ] = hs[θ][ρ] + 1 ;
                    mutex_unlock(lock);
                      ⋮
```

Figura 4.15. Solução (f), para a Transformada de Hough Clássica em uma máquina MIMD fortemente acoplada.

Uma característica importante do mestre é que este deve fazer a utilização do espaço Hough em exclusão mútua. Portanto, é necessário que o acesso ao espaço Hough seja controlado por **semáforos**. A figura 4.15 mostra esta tarefa implementada pela função atualiza_hs().

4.4.2 - Utilizando Arquitetura SIMD

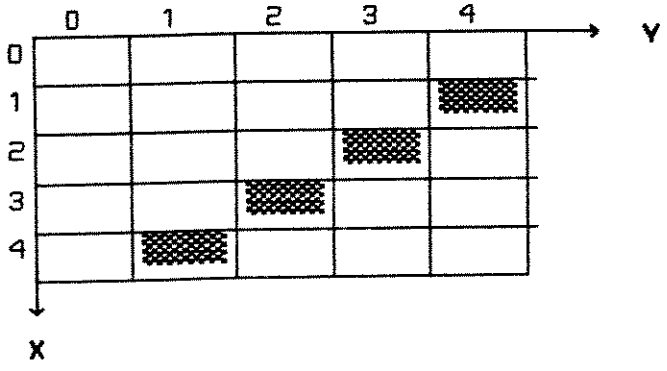
A implementação da transformada de Hough em uma arquitetura SIMD massivamente paralela, como a Connection Machine, aproveita a possibilidade de se manipular todos os elementos de um vetor simultaneamente. Este algoritmo transforma os dados de entrada em vetores, que são facilmente manipulados por operações primitivas do modelo V-RAM, seção 3.3. A idéia básica desta implementação é a de que o espaço Hough é, na verdade, um **Histograma**, em duas dimensões, dos pares (ρ_i, θ_i). Primeiramente será descrita a maneira como os dados da imagem de entrada são armazenados. Em seguida será mostrado como os dados são combinados de forma a produzir um vetor com os pares (ρ_i, θ_i). Posteriormente, será descrita a operação de cálculo do histograma.

Inicialmente, as coordenadas dos pixels das bordas detectadas na imagem de entrada são alocados em dois vetores "b_x" e "b_y" de dimensões 1xNb, onde Nb é o número de pixels de borda. Os pixels são atribuídos aos processadores utilizando o barramento escalar de acesso. De maneira semelhante são criadas as LUTs com os valores das funções trigonométricas. Cada vetor da look-up table tem dimensão [k][1], conforme é apresentado na figura 4.16.

$$ang = \begin{bmatrix} \theta \\ \theta_1^{o} \\ . \\ . \\ \theta_{k-1} \end{bmatrix} ; \quad l\_c = \begin{bmatrix} \cos\theta \\ \cos\theta_1^{o} \\ . \\ . \\ \cos\theta_{k-1} \end{bmatrix} ; \quad l\_s = \begin{bmatrix} sen\theta \\ sen\theta_1^{o} \\ . \\ . \\ sen\theta_{k-1} \end{bmatrix}$$

Figura 4.16. LUT's do algoritmo da transformada de Hough na Connection Machine.

As figuras 4.17 e 4.18 mostram um exemplo ilustrativo do funcionamento deste algoritmo. Supõe-se uma imagem com dimensões 5x5 e um espaço Hough com $\Delta\theta = \pi/4$.



$$b\_x = [\ 1 \quad 2 \quad 3 \quad 4\ ]$$

$$b\_y = [\ 4 \quad 3 \quad 2 \quad 1\ ]$$

$$ang = [\ 0 \quad \frac{\pi}{4} \quad \frac{\pi}{2} \quad \frac{3\pi}{4}\ ]$$

$$l\_c = [\ 1 \quad \frac{\sqrt{2}}{2} \quad 0 \quad -\frac{\sqrt{2}}{2}\ ]$$

$$l\_s = [\ 0 \quad \frac{\sqrt{2}}{2} \quad 1 \quad \frac{\sqrt{2}}{2}\ ]$$

Figura 4.17. Uma reta do tipo "y=5-x" no plano imagem e os vetores criados pelo algoritmo.

O produto cartesiano Tmp1 entre "l_c" e "b_x" e o produto cartesiano Tmp2 entre "l_s" e "b_y" associam cada ponto do plano imagem a cada ângulo de rotação. Em seguida, computa-se um novo vetor

P, elemento por elemento, executando uma operação de soma P = Tmp1 + Tmp2. P é um vetor de dimensões Nb×k e contém Nb conjuntos de dimensão k de valores de ρ calculados.

$$Tmp1 = I\_c \times b\_x =$$

$$Tmp1 = \left[\ 1\ 2\ 3\ 4\ \frac{\sqrt{2}}{2}\ \sqrt{2}\ \frac{3\sqrt{2}}{2}\ 2\sqrt{2}\ 0\ 0\ 0\ 0\ -\frac{\sqrt{2}}{2}\ -\sqrt{2}\ -\frac{3\sqrt{2}}{2}\ -2\sqrt{2}\ \right]$$

$$Tmp2 = I\_s \times b\_y =$$

$$Tmp2 = \left[\ 0\ 0\ 0\ 0\ 2\sqrt{2}\ \frac{3\sqrt{2}}{2}\ \sqrt{2}\ \frac{\sqrt{2}}{2}\ 4\ 3\ 2\ 1\ 2\sqrt{2}\ \frac{3\sqrt{2}}{2}\ \sqrt{2}\ \frac{\sqrt{2}}{2}\ \right]$$

$$P = Tmp1 + Tmp2 =$$

$$P = \left[\ 1\ 2\ 3\ 4\ \frac{5\sqrt{2}}{2}\ \frac{5\sqrt{2}}{2}\ \frac{5\sqrt{2}}{2}\ \frac{5\sqrt{2}}{2}\ 4\ 3\ 2\ 1\ \frac{3\sqrt{2}}{2}\ \frac{\sqrt{2}}{2}\ -\frac{\sqrt{2}}{2}\ -\frac{3\sqrt{2}}{2}\ \right]$$

Conforme as equações 2.3 da seção 2.2.3, para se calcular o índice da célula a ser incrementada basta fazer-se:

$$\rho = \left\lfloor -\frac{3 \cdot P}{5\sqrt{2}} + 3 \right\rfloor$$

$$\rho = [\ 2\ 3\ 3\ 4\ \ 5\ 5\ 5\ 5\ \ 4\ 3\ 3\ 2\ \ 4\ 3\ 3\ 3\ ]$$

O vetor ρ corresponde a um espaço Hough da forma mostrada na figura 4.18. A Transformada de Hough é o histograma segmentado de cada um dos Nb conjuntos de elementos de P. Cada segmento corresponde a um ângulo θ. Este algoritmo tem muitas vantagens. Uma delas é que o espaço utilizado fica restrito àquelas células que realmente têm votos. Outra vantagem é que o tempo de processamento é diminuído devido à simultaneidade das comunicações dos dados.

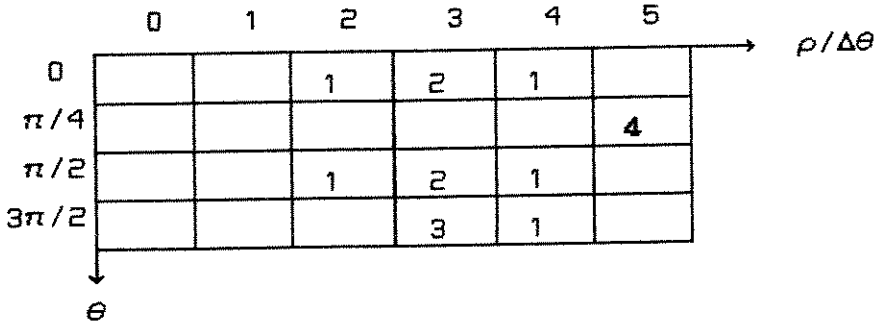


Figura 4.18. Espaço Hough resultante da Transformada de Hough indicada na figura 4.17.

Há várias outras tentativas de paralelização da transformada de Hough utilizando arquiteturas paralelas. IBRAHIM, KENDER e SHAW (1985) apresentam a implementação paralela da transformada de Hough utilizando uma arquitetura MSIMD, na máquina paralela NON-VON. CYPHER, SANZ e SNYDER (1987) utilizam uma matriz NxN de processadores para implementar a transformada de Hough. O método tem complexidade $O(N_\theta \times N)$, onde N_θ é a discretização da variável θ. Esse método baseia-se na rotação da imagem de tal modo que todos os pixels de borda com um ângulo θ se alinham com alguma linha (ou coluna) da matriz de processadores. O cálculo da transformada de Hough fica, então, trivial. CHANDRAN e DAVIS (1987) apresentam a implementação paralela da transformada de Hough em uma arquitetura fortemente acoplada: a BBN Butterfly.

HANAHARA, MARUYAMA e UCHIYAMA (1988) implementaram a transformada de Hough em circuitos integrados. Sua abordagem sobrepõe o cálculo dos picos e os incrementos dos acumuladores.

GUERRA e HAMBRUSH (1989) apresentam 2 algoritmos paralelos, também utilizando uma matriz NxN: o MPP. Um deles particiona o espaço Hough em N_ρ sub-espaços **quadrados** de tamanho igual a $\lceil\sqrt{N_\theta}\rceil \times \lceil\sqrt{N_\theta}\rceil$. O outro é equivalente ao de Cypher, Sanz e Snyder. A complexidade do algoritmo é $O(N_\rho\lceil\sqrt{N_\theta}\rceil + N_\theta)$.

FISCHER e HINGHMAN (1989) descrevem um método de se paralelizar a transformada de Hough utilizando um array de processadores SIMD. Um EP é atribuído a cada coluna da imagem e o array se move sobre a imagem, processando todos os pixels de uma linha, concorrentemente. LI, PAO e JAYAKUMAR (1989) apresentam uma implementação sistólica da transformada de Hough. TZVI e SANDLER (1990) apresentam um CI analógico para computar a transformada de Hough de uma imagem. Para este caso, a detecção dos picos não é feita por hardware.

4.5 - (4ª fase) - Classificação das Retas

Nesta seção serão apresentados dois métodos paralelos para se implementar a fase de classificação das retas. Primeiramente será mostrado como a fase de isolamento das nuvens do espaço Hough - introduzida na seção 2.2.4 - pode ser implementada na Connection Machine. Em seguida apresenta-se um método baseado na solução de

"Processor Farm" - solução (e), da seção 4.4.1 - para se executar a classificação das retas **simultaneamente** à Transformada de Hough.

A disposição de uma arquitetura massivamente paralela, ou seja, onde cada processador possui uma célula acumuladora do Espaço de Hough, permite simplificações notáveis no algoritmo de isolamento das nuvens. A figura 4.19 mostra como isto é possível: A variável ("read-only") paralela "hs" representa todo o Espaço Hough. O comando "where" seleciona apenas as células acima de uma limiar pré-fixado, da mesma maneira que foi feito na seção 2.2.4. O teste "if-else" é implementado na CM do seguinte modo: Cada célula que passou pelo teste de limiarização lê de seus 8 vizinhos o conteúdo da variável hs dele e compara com sua própria variável hs. Se a variável hs de algum dos vizinhos for maior que sua própria, zera-se a célula. Ao final restam apenas as células com os valores máximos locais das nuvens.

```
where(hs >= limiar)
{
  if(hs >= 8_vizinhos) hs = PRETO ;
  else hs = BRANCO ;
}
```

Figura 4.19. Segmentação das retas em paralelo.

Utilizando-se máquinas MIMD fracamente acopladas pode-se fazer o isolamento das nuvens - sequencialmente - e o modelamento dos dados em paralelo, conforme figura 4.20.

```
SEQ
  SEQ i = 0 FOR LIN
    SEQ j = 0 FOR COL
    ...  separação nos vetores acumuladores
  PAR i = 0 FOR número_de_retas_detectadas
    ...  escolhe o parâmetro que melhor represente a reta
```

Figura 4.20. O algoritmo de análise.

Porém, este estágio tem complexidade $O(N_\theta N_\rho + B/n)$, onde B é o número de pixels de borda e n é o número de processadores utilizados para escolher o parâmetro que melhor representa os dados de um vetor acumulador. Isto pode ser melhorado, como será visto na sequência.

A fase de classificação das retas - 4ª fase, conforme apresentado na seção 2.2.4 - pode ser implementada de duas maneiras. A primeira delas faz uma varredura completa no espaço Hough para isolar as nuvens de pontos, conforme apresentado na figura 4.20. E a segunda utiliza informação da imagem anterior e faz apenas uma busca em uma janela no espaço Hough.

Experimentos com diversas imagens mostraram que, para um espaço Hough com resolução de 256x326 e utilizando-se 9 janelas de busca (correspondentes às 9 retas do paralelepípedo) com dimensões de 10x10 células, consegue-se uma redução de cerca de 60% no tempo de execução desta fase utilizando-se o algoritmo simplificado.

Como já foi apontado nas seções 2.2.4 e 4.5, esta implementação da 4ª fase utilizando informação da imagem anterior somente pode ser feita em regime, ou seja, supondo-se que o movimento da câmera seja sufucientemente pequeno para que os picos das nuvens do espaço Hough permaneçam nos arredores do pico encontrado na imagem anterior. Será apresentado um algoritmo paralelo para ser utilizado em regime.

```
PAR
  SEQ
    PAR          -- mestre
      ...  transmissão de pacotes (x,y,θ)
      ...  recepção e classificação de pacotes de picos
      ...  transmissão de pacotes (θa,ρa)
      ...  recepção dos picos
    ...  cálculo das propriedades
    ...  calibração
  PAR i = 0 FOR n.de.escravos          -- escravos
    SEQ
      ...  recebe o pacote
      IF
        pacote = (x,y,θ)
          SEQ
            ...  calcula a transformada de Hough
            ...  envia pares (θ,ρ) para o mestre
        pacote = (θi,ρi)
          ...  acumulação das propriedades da nuvem i
        pacote = fim.de.dados
          ...  transmite (θfinal,ρfinal)
```

Figura 4.21. Algoritmo paralelo mostrando a sobreposição potencial da Transformada de Hough e o Cálculo das Propriedades em diferentes processadores.

A idéia do algoritmo em regime é, basicamente, a de fazer-se a sobreposição da 3ª e 4ª fases seguindo a idéia de "processor farm", do modelo (e) da Transformada de Hough Direcional, vista na seção 4.4.1. O processo mestre é responsável pela transmissão de pacotes (x,y,θ), pelo isolamento das nuvens e pelo término do processo. Uma descrição simplificada pode ser vista na figura 4.21.

Como pode ser observado na figura 4.21, há uma potencial sobreposição das fases da Transformada de Hough Direcional e do Cálculo das Propriedades: Por exemplo, o processador 4 pode estar calculando a Transformada de Hough enquanto o processador 6 pode estar acumulando as propriedades da nuvem 6 e o processador 2 já pode ter terminado seu trabalho.

Pode-se notar que há um congestionamento de mensagens nos links próximos ao mestre, pois este é que centraliza as comunicações de mensagens. Todavia, este fato pode ser atenuado através da troca de mensagens **em pacotes** e de sua utilização **em regime**. A transmissão de mensagens em pacotes é mais eficiente quanto maior o pacote, ver INMOS (1988).E embora o algoritmo da figura 4.21 possa ser utilizado na primeira imagem captada, seu desempenho melhora muito a partir da segunda imagem, pois a quantidade de trabalho do mestre é atenuada.

Este capítulo apresentou um conjunto de soluções - baseadas em técnicas de processamento paralelo - para as diversas fases do algoritmo de reconhecimento e localização de objetos no espaço 3D. Os resultados das aplicações destas soluções são apresentados no capítulo 5.

CAPÍTULO V

RESULTADOS EXPERIMENTAIS

Neste capítulo pretende-se quantificar, através de medidas de precisão e desempenho, a aplicação de técnicas de processamento paralelo ao algoritmo de localização de objetos do tipo paralelepípedo no espaço 3D, apresentado no capítulo 2 em imagens reais (não simuladas). Primeiramente são analisados alguns aspectos sobre a precisão global do algoritmo serial. Estes resultados servirão para se avaliar os resultados de aceleração do algoritmo "paralelizado". Em seguida são apresentados os resultados experimentais da aplicação das técnicas de processamento paralelo.

5.1 - As Imagens Utilizadas nos Experimentos

Para a aquisição das imagens, foi utilizada uma placa digitalizadora **TARGA 16**, da Truevision, Inc. Esta placa é compatível com o IBM-PC. Possui resolução espacial máxima de 512x512 e 32768 cores, utilizando 5 bits para cada banda (RGB). Utilizando-se imagens em preto e branco, dispõe-se de 5 bits/pixel ou 32 níveis de cinza. Utilizou-se a câmera TK-204, da Kentec, que é uma câmera CCD com elementos de imagem dispostos em uma matriz de 485(V)x570(H). O sinal de vídeo é codificado em NTSC e possui uma frequência nominal de varredura de 15,734 KHz/59,94 Hz (entrelaçado).

As imagens foram obtidas com este equipamento de digitalização e transportadas para os sistemas de processamento. As imagens típicas utilizadas para os testes consistiram de **um fundo homogêneo, ou seja, sem mudanças bruscas de cor e o objeto do tipo paralelepípedo em primeiro plano.**

Foi produzido um pacote para apresentação de imagens, coloridas ou em tons de cinza, para workstations do tipo SUN. Este foi implementado em linguagem C e inclui bibliotecas do utilitário SunView. Através deste pacote pode ser feito um acompanhamento visual de todas as fases do algoritmo do capítulo 2, bem como quaisquer imagens que utilizam o mesmo formato de arquivo.

5.2 - A Precisão do Algoritmo Serial

Nesta seção apresentam-se os resultados dos testes de precisão global do algoritmo serial aplicado à imagens **reais**. Na seção 2.2.7 foram apresentados os resultados da aplicação do algoritmo de calibração em imagens **artificiais**. A especificação nº 4 do algoritmo de reconhecimento e localização de objetos no espaço, vista no capítulo 1, mostra um tipo de medida de precisão muito encontrado na literatura (LEE et AL, 1991): O desvio médio relativo de posição e o desvio médio de orientação.

Dependendo do tipo da aplicação dada ao algoritmo apresentado no capítulo 2 - por exemplo: auto-navegação, inspeção automática, "pick-and-place", etc. - especifica-se um tipo de medida de precisão para o algoritmo de localização no espaço. Uma dificuldade frequentemente encontrada é a de se encontrar "manualmente" valores acurados das medidas de precisão para se poder comparar com os resultados calculados pelo algoritmo. Por exemplo: no caso do algoritmo descrito no capítulo 2, as medidas das localizações são obtidas em relação ao centro do plano imagem. O centro do plano imagem localiza-se dentro da câmera, dificultando o acesso a ele para se fazer uma medida manual que comprove a precisão do algoritmo.

Este mesmo problema ocorre na medida da precisão da orientação do paralelpípedo em relação ao mesmo referencial. Contudo, em manipulação robótica, as garras possuem uma tolerância maior para o desvio de orientação pois possuem soluções mecânicas para lidar com tais diferenças. Desse modo, considera-se os desvios apresentados na simulação da seção 2.2.7 como suficientes para validar o algoritmo neste aspecto.

Como medida da precisão do algoritmo quanto às coordenadas de posicionamento do paralelpípedo, adotou-se o parâmetro **D**, definido como a distância, em metros, do vértice P_3 do paralelpípedo até o centro do plano imagem. Esta é uma medida simples de ser feita manualmente pois basta esticar uma linha do ponto P_3 até a matriz de fotosensores - situada logo atrás das lentes na câmera "CCD" - e medir o comprimento, em metros, desta linha. Matematicamente D é dado pela distância:

$$D = \sqrt{X_C^2 + Y_C^2 + Z_C^2}$$

Onde X_c , Y_c e Z_c são as coordenadas do centro do plano imagem em relação ao vértice P_3 que é adotado como origem do sistema de coordenadas no espaço, conforme a seção 2.2.7. Desse modo pode-se comparar o valor de D medido manualmente com o valor de D calculado pelo algoritmo. Estima-se empiricamente que o erro de medida manual de posicionamento situe-se em torno de 1,00cm.

A imagem de um paralelepípedo com tamanho 10cm x 10cm x 20 cm foi capturada de diversas posições no espaço. Cada pixel da matriz de sensores, segundo o manual do fabricante, possui um tamanho retangular $T_x = 1{,}123.10^{-5}$ m/pixel, $T_y = 9{,}897.10^{-6}$ m/pixel, onde T_x é a dimensão horizontal do pixel. A tabela 5.1 mostra os resultados experimentais das medidas do valor de D, para imagens do paralelpípedo com diferentes ângulos de rotação e mantendo-se a origem e a câmera fixas.

θ	ϕ	ψ	D_m	D_c	E	E_{pr}
-45,0°	-45,0°	0,0°	126,0 ±1,0cm	128,4cm	-2,4cm	1,90%
-60,0°	-45,0°	0,0°	126,0 ±1,0cm	129,2cm	-3,2cm	2,53%
-45,0°	-60,0°	0,0°	126,0 ±1,0cm	131,9cm	-5,9cm	4,68%
-45,0°	-45,0°	30,0°	126,0 ±1,0cm	130,8cm	-4,8cm	3,80%

Tabela 5.1. Resultados do teste de precisão do algoritmo descrito no capítulo 2. As resoluções da imagem e do Espaço Hough são 200x256 e 512x512, respectivamente.

Na tabela 5.1, θ, ϕ e ψ são os ângulos aproximados de rotação sobre os eixos z, x e y respectivamente; D_m é a distância, medida em centímetros, manualmente entre o vértice P_3 do paralelpípedo e o centro do plano imagem; D_c é calculado como a raiz quadrada da soma dos quadrados das coordenadas X_c, Y_c e Z_c computadas pelo algoritmo; E indica a diferença entre D_m e D_c ; E_{pr} especifica o desvio relativo no cálculo de D_c, ou seja, é a razão entre o módulo da diferença entre D_m e D_c e o valor de D_m.

De acordo com a tabela 5.1, os valores de E_{pr} situam-se próximos, ou mesmo dentro, da especificação Nº4 do capítulo 1. Para se melhorar a precisão dos resultados deve-se *aumentar* a discretização em θ (N_θ), conforme será explicado na figura 5.1.

O desvio foi menor quando $(\theta,\phi,\psi) \cong (-45^\circ,-45^\circ,0^\circ)$. Isto pode ser explicado pelo fato de que é nesta posição que a imagem do paralelpípedo apresenta o maior comprimento médio de suas arestas. De uma maneira geral, E_{pr} é sensível à alterações de ϕ, de ψ e de θ pois as rotações diminuem o tamanho das arestas na imagem, prejudicando sua identificação durante a fase de classificação de retas e as fases seguintes.

Outro fato observável na tabela 5.1 é que todos os valores de E foram negativos. Isto sugere que há uma polarização dos valores finais das coordenadas calculadas. Todavia, para valores maiores de f (distância focal da câmera), as simulações mostraram que este fator de polarização decresce, diminuindo consequentemente os desvios. Câmeras com lentes de diferentes distâncias focais, ou mesmo uma objetiva com auto-foco, permitem a aquisição de imagens a diferentes distâncias do objeto, permitindo também analisar sua influência nos resulatados finais. Este fato não pode ser comprovado na prática pois dispunha-se apenas de uma lente de 16mm.

Outro parâmetro - além da distância focal - que tem influência na precisão final do cálculo das coordenadas do vértice P_3 refere-se à transformada de Hough: a discretização em θ (N_θ). A figura 5.1 mostra uma curva experimental representando o desvio (E_{pr}) na localização do vértice P_3 do paralelepípedo em função da resolução em θ para 10 imagens com orientações e posicionamento diferentes. As resoluções da imagem e do Espaço Hough são 200x256 e 256x326, respectivamente.

Na execução dos testes com as soluções paralelas para a transformada de Hough, descritos na seção 5.5, adotou-se a resolução N_θ=256, ou seja, conforme a figura 2.13, N_θxN_ρ= 256x326. Este valor apresenta experimentalmente um bom compromisso entre precisão e tempo de processamento, conforme pode ser visto na figura 5.1. Como já foi apontado no capítulo 1, é necessário que o algoritmo de localização de objetos no espaço 3D possua uma precisão superior à dos atuadores. Portanto, dependendo da aplicação, a precisão conseguida com este algoritmo é bastante satisfatória. E, para melhorar sua precisão, basta aumentar N_θ segundo a curva da figura 5.1.

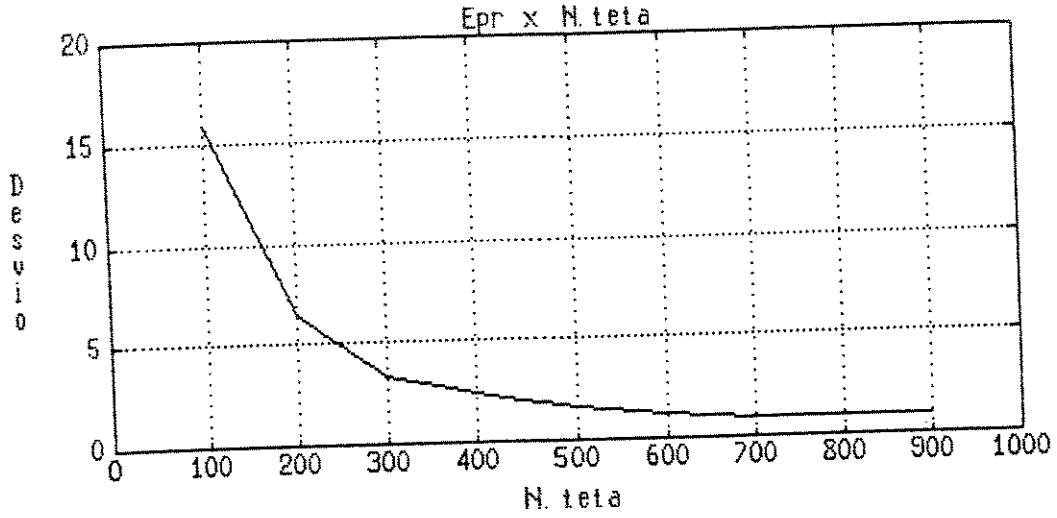


Figura 5.1. Desvio percentual na localização de P_3 em função de N_θ.

5.3 - O Desempenho do Algoritmo Serial no Transputer

O tempo de execução do algoritmo depende da complexidade da imagem, ou seja, do número de pixels de borda da imagem. Uma imagem típica para a aplicação em pauta tem resolução de 200x256x8 e possui cerca de 2500 pixels de borda. Geralmente, a razão do número de pixels de borda e o de pixels total é da ordem de 5%, ou seja,

$$C = \frac{\text{número de pixels '1'}}{\text{número de pixels total}} \cong 5\%$$

Conforme apresentado na seção 3.2, para se medir os parâmetros de desempenho **aceleração** e **eficiência** é necessário conhecer-se T(1), que é o tempo de execução do algoritmo quando se utiliza apenas um processador. Portanto, apresenta-se aqui os valores de T(1) para a transformada clássica de Hough e para a transformada direcional de Hough.

A tabela 5.2 apresenta os tempos de execução das diversas fases do algoritmo serial de reconhecimento e localização de objetos no espaço 3D utilizando-se apenas um transputer. Na tabela 5.2a utilizou-se a transformada clássica de Hough e na tabela 5.2b utilizou-se a transformada direcional de Hough com k=20.

O algoritmo implementado para obter-se os dados da tabela 5.2 utiliza o algoritmo de detecção de Sobel e a transformada clássica de Hough na 3ª fase, como descrita na seção 2.2.3, com resolução do espaço de Hough de 256x326. Os resultados da tabela 5.2 foram obtidos processando-se 10 imagens com resolução de 200x256x5 com complexidades em torno de 5% e tirando-se a média ponderada pela respectivas suas complexidades.

De acordo com o apresentado na seção 2.2.3, o detector de bordas de Sobel é executado internamente à fase da Transformada Direcional de Hough. E, portanto, a tabela 5.2b não apresenta medidas de tempo paraa 1ª e 2ª fases.

Fase	T.H.C.	
	Tempo de CPU	%
1ª	0,573 s	10,01%
2ª	0,881 s	15,40%
3ª	2,355 s	41,17%
4ª	1,127 s	19,70%
5ª	0,008 s	0,14%
6ª	0,774 s	13,53%
7ª	0,002 s	0,03%
	5,720 s	

(a)

T.H.D. (k=20)	
Tempo de CPU	%
----	---
----	---
2,179 s	55,08%
1,006 s	25,42%
0,006 s	0,15%
0,763 s	19,28%
0,002 s	0,50%
3,956 s	

(b)

Tabela 5.2. Tabela de tempos do algoritmo serial, T(1), apresentado no capítulo 2, implementado na placa IMSB004 ("monoputer").

A metodologia para a determinação dos tempos de execução apresentados na tabela 5.2 teve como base a utilização do relógio embutido no transputer IMST800. Este relógio é diretamente acessado através de comandos da linguagem OCCAM. Um "tick" do relógio equivale a 1μs. Conforme já foi apontado no capítulo 1, os tempos medidos neste trabalho não levam em conta o tempo de carga da imagem, ou seja, o tempo de recuperação da imagem, proveniente do disco, para a RAM. Os resultados obtidos supõem, deste modo, a presença de um hardware específico para a aquisição de imagens em tempo-real - geralmente com múltiplos "buffers" de imagens - existentes em estações de trabalho para visão computacional; ver, por exemplo, CASTANHO (1990).

Os erros de medida dos tempos de execução apresentados na tabela 5.2, utilizando-se o transputer, são reduzidos. As medidas de tempo de execução obtidas na "workstation" Sun, e apresentadas na tabela 4.1, são menos precisas que as obtidas no transputer devido ao sistema operacional ser multiusuário. Este fato é mais notável na seção 5.6, onde os tempos de execução são bem menores e o erro relativo da medida de tempo se torna significativo.

A figura 5.2 mostra uma curva obtida experimentalmente e que representa o tempo de execução da transformada direcional de Hough em função do número (k) de ângulos calculados pelo algoritmo da transformada direcional de Hough em um único transputer. Foram utilizadas 10 imagens com orientações e posicionamento diferentes. As resoluções da imagem e do Espaço Hough foram 200x256 e 256x326, respectivamente.

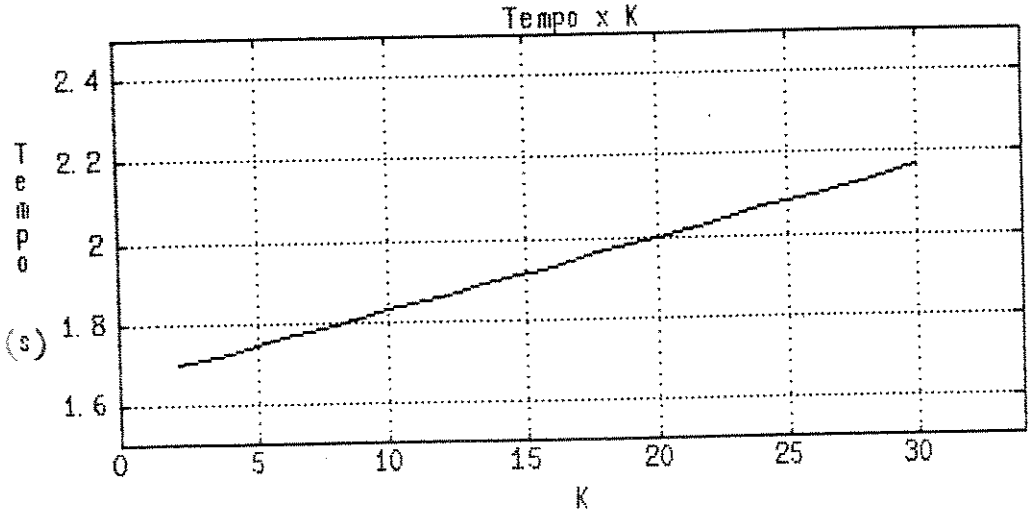


Figura 5.2. Tempo de Execução (monoprocessador) x k .
Resolução da imagem: 200x256x8. Complexidade da imagem: 5%.

Os tempos de execução - T(1) - apresentados na tabela 5.2 e os tempos de execução para k = 1 e k = 20, mostrados na figura 5.2, serão utilizados para se medir os parâmetros **aceleração e eficiência** nas seções que se seguem. Uma tabela de tempos de execução mais completa, correspondente à figura 5.2, é apresentada no apêndice 4.

5.4 - Observações sobre a "Paralelização" do Algoritmo

Um dos objetivos da paralelização do algoritmo apresentado no capítulo 2 é conseguir uma redução no tempo de processamento e, se possível, sua execução em tempo-real. Como foi observado no capítulo 1, um objetivo simples para um sistema de Visão aplicado a robótica, por exemplo, é determinar a calibração da câmera em ciclos de 1/30s $\cong$ 33,3 ms.

Em Visão Computacional - bem como em Processamento de Imagens - há dois conjuntos de variáveis para se analisar e controlar, visando-se acelerar o algoritmo:

(a) **Variáveis relacionadas à Imagem**: resolução da imagem, complexidade da imagem, tamanho da máscara de detecção de bordas, limiar de binarização, resolução do espaço Hough, tamanho (k) do intervalo de ângulos da transformada de Hough que utiliza informação direcional, etc.

(b) **Variáveis relacionadas ao Processamento**: utilização (ou não) de CIs para processamento de nível baixo, tipo de placa digitalizadora, número de processadores, tipo de topologia, classe da arquitetura paralela, tempo de carga da imagem, tamanho do pacote de mensagens, velocidade dos links de comunicação, etc.

A utilização inteligente destas variáveis é a chave para obter-se um bom desempenho na paralelização do algoritmo. Tendo este fato em vista, 6 variáveis de controle foram escolhidas. **variáveis de imagem**: resolução da imagem, resolução do espaço Hough, número de ângulos da transformada direcional de Hough (k); **variáveis de processamento**: tamanho do pacote de mensagens, topologia, classe da arquitetura.

De acordo com a tabela 5.2, as fases do algoritmo de localização de objetos do tipo paralelepípedo têm características bem definidas quanto ao tempo de execução:

(1) O tempo de carga da imagem não precisa ser considerado. Assume-se portanto que dispõe-se de hardware de aquisição de imagens em tempo real.

(2) A 1ª fase pode ser implementada utilizando-se o algoritmo sistólico da seção 4.2.1, utilizando-se circuitos VLSI e,

portanto, pode-se considerar que esta fase não acarreta atraso no tempo de entrada da imagem na memória.

(3) A 2ª e a 6ª fases, conforme exposto no capítulo 2, são opcionais e sua paralelização não foi considerada nesta seção.

(4) A 3ª e a 4ª fases são as fases mais demoradas, exigindo a implementação prioritária das técnicas abordadas nas seções 4.4.1 e 4.4.2.

(5) A 5ª e a 7ª fases são as fases mais rápidas e não é necessário despender esforços na sua aceleração.

O paralelismo foi considerado na fase de detecção de bordas, na Transformada de Hough e na Classificação das retas. As seções 5.5 e 5.6 apresentam os resultados da paralelização do algoritmo em dois tipos de arquitetura paralela: MIMD fracamente acoplada e SIMD, respectivamente. Conforme apresentado na seção 3.2, há duas abordagens para se medir a aceleração do algoritmo paralelo. A seção 5.5 emprega a 1ª abordagem, ou seja, fixa-se algumas variáveis e altera-se o número de processadores. A seção 5.7 emprega a segunda abordagem, ou seja, fixa-se um tempo de execução e escala-se algumas variáveis de controle modo que ele seja executado dentro do tempo determinado ("tempo-real").

5.5 - Utilizando Arquitetura MIMD Fracamente Acoplada

Foi utilizado o sistema de processamento baseado nos processadores transputer. O algoritmo foi implementado em duas placas de transputers: (1) Placa INMOS-B004 com 1 transputer T800 e 2Mbytes de RAM, 20 MHz (LCA/DCA/FEE/UNICAMP). Software utilizado: OCCAM2 e TDS2, conforme INMOS (1988).(2) Placa INMOS-B008 com 9 TRAMs, 20MHz, um cross-bar IMS C004, para configuração das topologias, e um total de 2,25 Mbytes de RAM (CTI). Software utilizado: TDS2 e MMS2, conforme INMOS (1988).

5.5.1 - (1ª fase) Detecção de Bordas

O algoritmo sistólico apresentado na seção 4.2.1 foi implementado em OCCAM2 e roda em uma rede de transputers. O operador de Sobel nesta versão sistólica é muito ineficiente, se implementado em poucos transputers, devido ao grande número de processos que devem ser executados em paralelo para cada pixel da imagem.

Para cada pixel que "entra" no algoritmo, são efetuadas, basicamente, (6*COL) deslocamentos **paralelos** de dados nas 6 linhas de atraso (3 para cada máscara), onde COL é a variável que quantifica o número de colunas da imagem. Se COL for igual a 256, mais de 1500 deslocamentos são necessários para que todos os dados se sincronizem nos respectivos processos. Embora as instruções desses processos sejam extremamente simples - de fato são praticamente movimentações de conteúdos de variáveis - estes deslocamentos são feitos sequencialmente e envolvem um "overhead" significativo no escalonamento paralelo destes processos por parte do transputer.

Tamanho da Imagem	Nº aprox. de Processos Paralelos	Tempo de Execução	$S_p(6)$
50x50	300	2,35s	0,21
100x100	600	4,88s	0,10
150x150	900	6,07s	0,08
200x200	1200	7,46s	0,07
256x256	1500	9,76s	0,05
512x512	3000	17,49s	0,02

Tabela 5.3. Influência do número de processos paralelos ("PAR PROC's") no algoritmo de convolução sistólico.

A tabela 5.3 apresenta os resultados da implementação sistólica do operador de Sobel, apresentado na figura 4.6, para diferentes tamanhos de imagens de entrada. Os dados desta tabela foram obtidos configurando-se o algoritmo em 6 transputers (2 para as linhas de atraso, 2 para os blocos calcula, 1 para o bloco de entrada (base) e 1 para receber os dados e fazer a limiarização). As comunicações foram feitas com protocolo **simples**, que é o nome que se dá em OCCAM à mensagem contendo apenas um pixel.

Esta implementação sistólica trabalha tanto melhor quanto maior for o número de processadores configurados. Na verdade, seria necessário dispor-se de cerca de 1500 processadores para estes deslocamentos pudessem ser feitos realmente em paralelo.

Conclui-se que o "overhead" do gerenciamento dessa quantidade de processos em poucos transputers e o pequeno "bandwidth" de comunicação inter-processadores tem significativa influência no tempo de execução do algoritmo. Por essa ineficiência e pela possibilidade de se ter o algoritmo integrado (ver CASTANHO, 1989) optou-se por considerar a imagem binária com as bordas já detectadas como a imagem de entrada do algoritmo. Na seção 5.6 serão apresentados os resultados da implementação desta operação de detecção de bordas na Connection Machine.

5.5.2 - (3ª fase) Identificação das Bordas do Objeto

Nesta seção serão analisadas as soluções paralelas propostas na seção 4.4.1 para a implementação paralela da Transformada de Hough. Dentro de cada solução analisada será apresentada também a influência de diferentes topologias no desempenho da solução paralela. Em cada experimento um conjunto de variáveis, de imagem e de processamento, foi fixado para se analisar a influência da topologia na solução escolhida. Estas variáveis fixadas, bem como outros dados de interesse, são apresentados em tabelas. Um resumo das soluções paralelas - propostas na seção 4.4.1 - para a transformada de Hough está na tabela 5.4.

Para se fazer o mapeamento da transformada de Hough na rede de transputers, os recursos consistiram, basicamente, de uma placa com 9 tranputers e de um "cross-bar". Desse modo, do ponto de vista do

custo, dispunha-se de 4 links por nó processador e um circuito programável de interconexão entre os links. Do ponto de vista do desempenho, procurou-se topologias com pequeno diâmetro.

Tipo da Transformada de Hough	Solução Proposta	
"Clássica"	(a) (b) (c)	Partição do espaço Hough Partição da Imagem Partição da Imagem e do Espaço Hough
"Direcional"	(d) (e)	Partição da Imagem e do Espaço Hough "Processor Farm"
"Clássica ou "Direcional"	(f)	Partição da Imagem

Tabela 5.4. Resumo das soluções propostas para a paralelização da Transformada de Hough em arquiteturas MIMD.

Nos experimentos foram utilizadas 4 topologias: Árvore, Anel, Hipercubo e Malha 2D com "wrap-around". Os links dos transputers são nomeados segundo a figura 5.3.

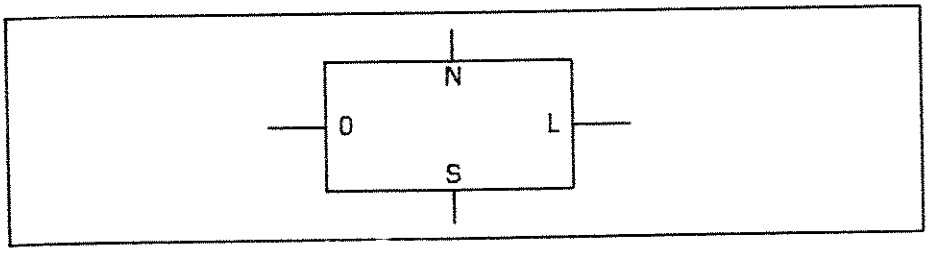
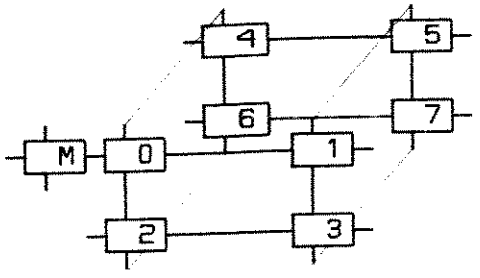


Figura 5.3. Nomeação dos links do transputer.
N=0, L=1, S=2 e O=3.

A topologia anel foi implementada como mostra a figura 3.3. Neste caso o algoritmo de roteamento (comunicação) é extremamente simples pois há somente um caminho - embora bi-direcional - para a mensagem percorrer.

A topologia árvore foi implementada como mostram as figuras 3.4. e 3.5, dependendo do número de processadores. O algoritmo de roteamento (comunicação) - cuja versão generalizada foi apresentada na figura 3.11 - também foi implementado através da técnica de tabelas de roteamento.

O algoritmo de roteamento (comunicação) adotado para a topologia hipercúbica também adota a técnica de tabelas de roteamento e pode ser visto na figura 5.4b. O nó "M" é aquele onde é alocado o processo mestre. Uma tabela como esta deve estar presente em cada nó processador. Esta tabela fornece o caminho a ser seguido por uma mensagem enquanto ainda não tiver chegado ao seu destino, conforme algoritmo generalizado mostrado na figura 3.11.



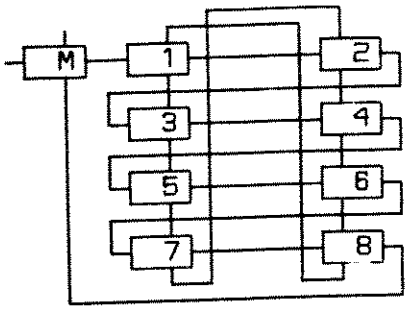
(a)

para \ de	M	0	1	2	3	4	5	6	7
M	-	0	0	N	N	N	N	N	N
0	L	-	0	N	N	N	N	N	N
1	L	L	-	N	N	L	N	L	S
2	L	S	S	-	0	S	0	S	0
3	L	S	S	L	-	N	S	S	S
4	L	N	N	S	S	-	0	N	0
5	L	N	N	L	S	L	-	L	N
6	L	S	0	S	0	S	0	-	0
7	L	L	S	L	S	L	S	L	-

(b)

Figura 5.4. Hipercubo de dimensão 3 e seu algoritmo de comunicação. Os links são nomeados segundo a figura 5.3.

Como já foi visto na seção 3.11, um hipercubo de dimensão 3 também possui diâmetro igual a 3. Contudo, conforme pode ser visto na figura 5.4a, alguns links não são utilizados. Através da utilização destes links extras pode-se criar um "hipercubo degenerado" - conforme visto na figura 3.9a - e diminuir o diâmetro da topologia resultante. Uma outra solução topológica que apresenta diâmetro menor que o hipercubo é a topologia malha com "wrap-around", que é apresentada na figura 5.5a. Esta malha difere daquela mapeada no hipercubo pois une as diagonais opostas e não os extremos opostos. A distância média entre o nó mestre e qualquer um dos nós escravos é 2,0. A distância média entre quaisquer dois nós escravos é 1,4643. E a distância média entre quaisquer dois nós é 1,5833.



(a)

de

para	M	1	2	3	4	5	6	7	8
M	-	O	O	N	N	S	S	L	L
1	L	-	O	N	N	N	S	S	S
2	L	L	-	O	N	S	N	S	O
3	L	S	L	-	O	N	O	N	S
4	L	L	S	L	-	O	N	O	N
5	S	S	N	S	L	-	O	N	O
6	S	N	S	L	S	L	-	O	N
7	S	L	N	S	L	S	L	-	O
8	S	N	O	N	S	L	S	L	-

(b)

Figura 5.5. Malha com "wrap-around" (diâmetro igual 2) e seu algoritmo de comunicação.

Transformada Paralela Clássica de Hough - (T.P.C.H.)

Será feita uma análise sobre a influência de diversas topologias e das soluções descritas na seção 4.4.1 na implementação paralela da transformada "clássica" de Hough. A imagem utilizada é a apresentada na figura 2.7b. A tabela 5.5 apresenta o conjunto das variáveis fixadas para a execução dos testes na rede de transputers. Conforme apresentado na seção 4.4.1, cada processador escravo pode conter uma partição do espaço Hough ou da imagem, dependendo da solução escolhida.

CONDIÇÕES DOS TESTES	
Resolução da Imagem	200x256x8
Resolução do Espaço Hough	256x326
Complexidade da Imagem	4%
Limiar de Binarização	60
Tempo de carga da imagem	2,7561 s
Velocidade dos links	20 MHz
Tamanho do Pacote de Mensagens	600
Classe de Arquitetura	MIMD
T(1)	2,355 s

Tabela 5.5. Variáveis fixadas para a execução dos testes com as soluções (a) e (c) na rede de transputers.

Devido ao tamanho da memória disponível para cada transputer da malha ser pequeno (32Kbytes), as soluções (b) e (f) não puderam ser implementadas e as soluções (a) e (c) somente podem ser implementadas com, no mínimo, 5 processadores, consistindo de um mestre e mais 4 escravos.

As soluções (a) e (c), da seção 4.4.1, para a transformada clássica de Hough foram implementadas em três topologias: anel com 4 a 8 escravos, malha com "wrap-around" com 4, 6 e 8 escravos e hipercubo com 4 e 8 escravos. A topologia árvore não foi implementada para estas soluções pois possui um diâmetro muito grande, obstáculo para um bom desempenho. A topologia árvore se casa bem com a solução (e). As figuras 5.6 e 5.7 apresentam as curvas de aceleração e eficiência da solução (a) em função do número de processadores para as três topologias: anel, hipercubo e malha com "wrap-around".

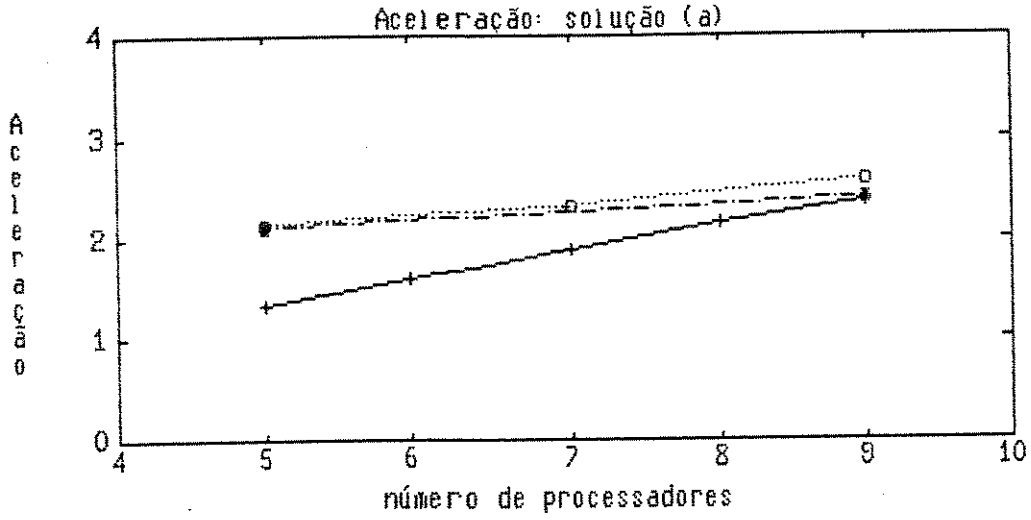


Figura 5.6. Aceleração da solução (a) para 3 topologias: anel ("---"), hipercubo ("-.-") e malha ("...").

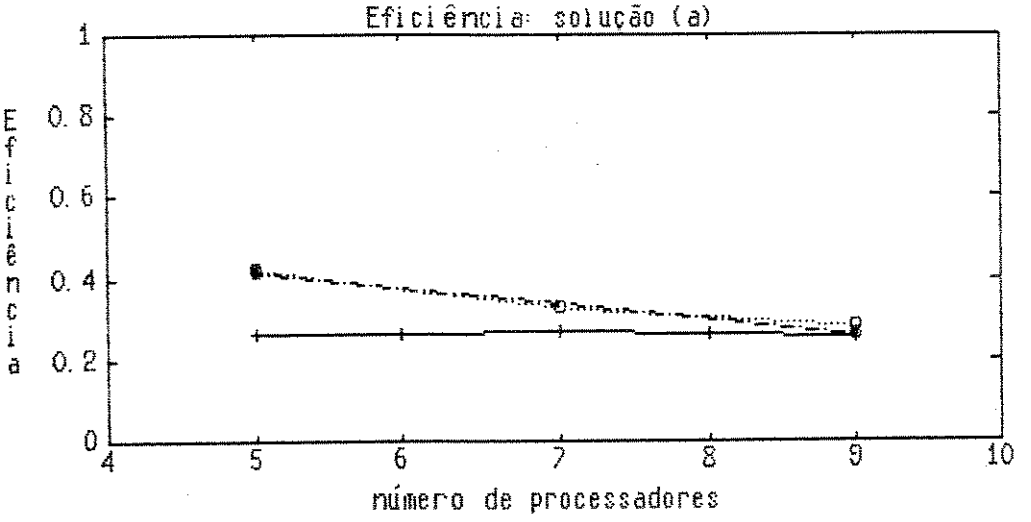


Figura 5.7. Eficiência da solução (a) para 3 topologias: anel ("---"), hipercubo ("-.-") e malha ("...").

Como se observa das figuras 5.6 e 5.7, para a solução (a), os valores de aceleração (e de eficiência) têm valores bem aquém do ideal. A topologia anel não apresenta comportamento de saturação da aceleração, até 9 processadores. A eficiência se mantém aproximadamente constante, também dentro deste intervalo de n (número de processadores). A topologia hipercúbica apresenta um comportamento semelhante ao da malha com "wrap-around". Esta não apresenta comportamento de saturação da aceleração. Todavia, possui baixa eficiência. A topologia malha com "wrap-around" possui os menores tempos de execução e, portanto, apresenta maior aceleração dentre as três topologias implementadas para esta solução.

As figuras 5.8 e 5.9 apresentam as curvas de aceleração e eficiência da solução (c) em função do número de processadores para as três topologias: anel, hipercubo e malha com "wrap-around".

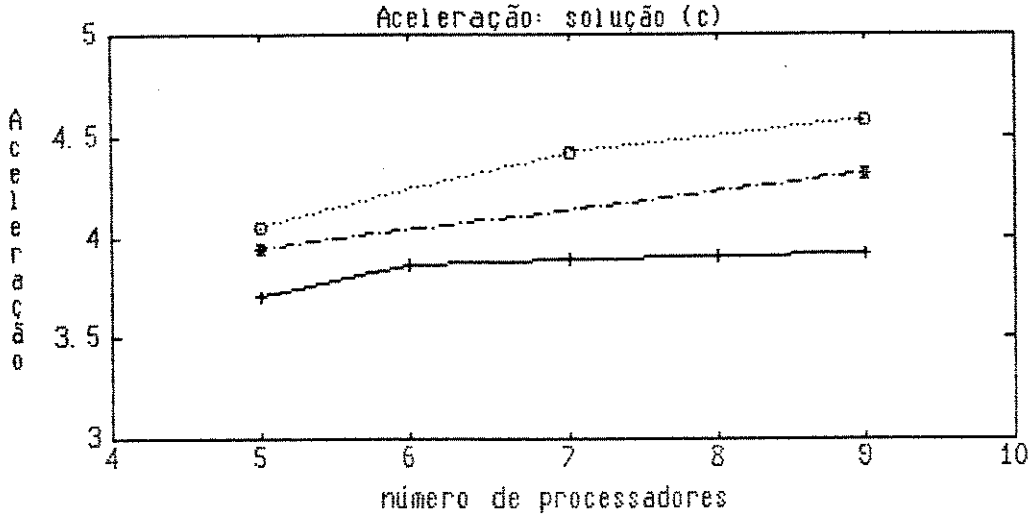


Figura 5.8. Aceleração da solução (c) para as topologias: anel ("---"), hipercubo ("-.-") e malha ("...").

Como se observa das figuras 5.8 e 5.9, referentes à solução (c), os valores de aceleração são maiores que os obtidos com a solução (a). Isto pode ser confirmado pela alta eficiência conseguida (maior que 50% neste intervalo de n). A topologia anel apresenta uma curva de aceleração com inclinação mais acentuada até n = 6 processadores. Para n maiores que 6, seu crescimento é mais lento. A topologia hipercúbica possui uma curva de aceleração com taxa aproximadamente constante de crescimento. Seus valores de aceleração são

intermediários entre os obtidos com as topologias anel e malha com "wrap-around". A topologia malha com "wrap-around" possui os melhores valores de aceleração entre as 3 topologias estudadas, para esta solução. A eficiência para 9 processadores é aproximadamente 50%.

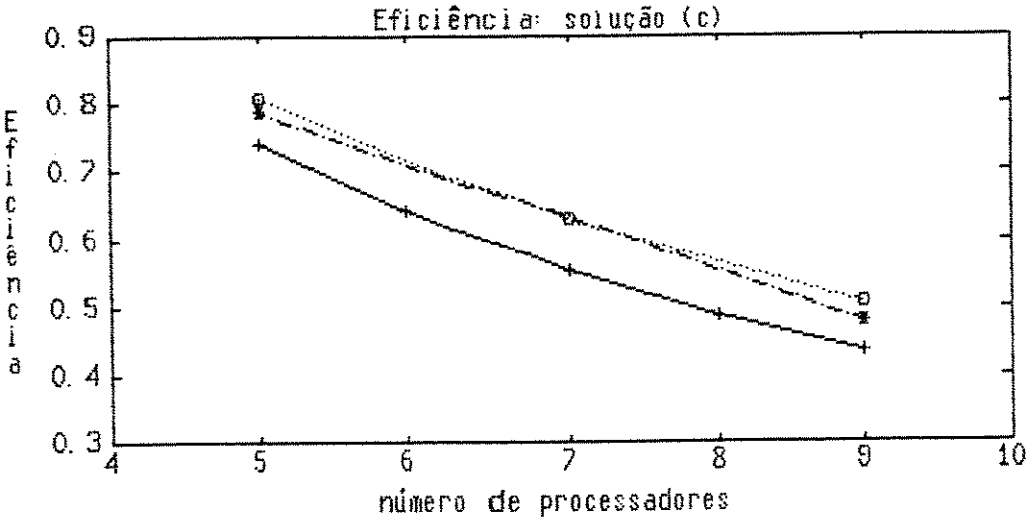


Figura 5.9. Eficiência da solução (c) para 3 topologias: anel ("---"), hipercubo ("-.-") e malha ("...").

Os dados sobre o desempenho máximo das duas soluções paralelas (a) e (c) da seção 4.4.1 são apresentados nas tabelas 5.6 e 5.7. As tabelas completas de tempos de execução encontram-se no apêndice 4.

	Solução (a)		
	anel n=9	malha n=9	hipercubo n=9
tempo de execução	1,003 s	0,912 s	0,983 s
Aceleração	2,348	2,582	2,396
Eficiência	0,261	0,287	0,266

Tabela 5.6. O desempenho máximo da Transformada de Hough - solução (a) - da seção 4.4.1.

	Solução (c)		
	anel n=9	malha n=9	hipercubo n=9
tempo de execução	0,602 s	0,582 s	0,597 s
Aceleração	3,912	4,573	4,305
Eficiência	0,435	0,508	0,478

Tabela 5.7. O desempenho máximo da transformada de Hough solução (c), da seção 4.4.1.

Como se observa das tabelas 5.6 e 5.7, a Transformada Clássica de Hough (T.C.H.) é executada com maior velocidade utilizando-se a solução (c) sobre uma topologia em malha com "wrap-around". Conforme a tabela 5.7, a melhor aceleração obtida é 4,573, também com essa topologia.

Este comportamento acontece devido ao fato do diâmetro das topologias malha com "wrap-around" e do hipercubo serem menores que o da topologia anel e, consequentemente, o tempo médio de troca de mensagens é menor, diminuindo o tempo total de execução.

A solução (a) para a T.C.H., apresentada na seção 4.4.1, exige que todos os processadores escravos recebam todos os pixels de borda. Esta grande quantidade de dados que chega (em pacotes) aos escravos, leva tanto mais tempo para chegar a eles quanto maior for o diâmetro da topologia. No caso da topologia anel, com 8 escravos, por exemplo, o conjunto de todos os pixels de borda pode ter que ser roteado por até 4 processadores intermediários, elevando excessivamente o custo de comunicação "mestre→escravo" desta solução.

A solução (c) para a T.C.H., apresentada na seção 4.4.1, procura diminuir o custo excessivo de comunicação "mestre→escravo" da solução (a). Conforme previsto na seção 4.4.1, esta solução possui um custo menor de comunicação pois faz maior uso da comunicação "escravo→escravo", diminuindo os valores do tempo de execução e, consequentemente, elevando os valores de aceleração.

Na sequência é apresentada uma análise sobre a influência da topologia e das soluções da seção 4.4.1 na implementação paralela da Transformada Direcional de Hough. A tabela 5.8 apresenta o conjunto das variáveis fixadas para a execução dos testes na rede de transputers. A imagem utilizada é a mesma utilizada nos casos anteriores (figura 2.7b). Conforme descrito na seção 4.4.1, a variável k representa o número de ângulos que deve ser calculado ao redor da direção Φ do gradiente obtido através das máscaras de Sobel.

As figuras 5.10 e 5.11 apresentam as curvas de aceleração e eficiência, respectivamente, da solução (d) em função do número de processadores para as três topologias: anel, hipercubo e malha com "wrap-around" e para k = 1.

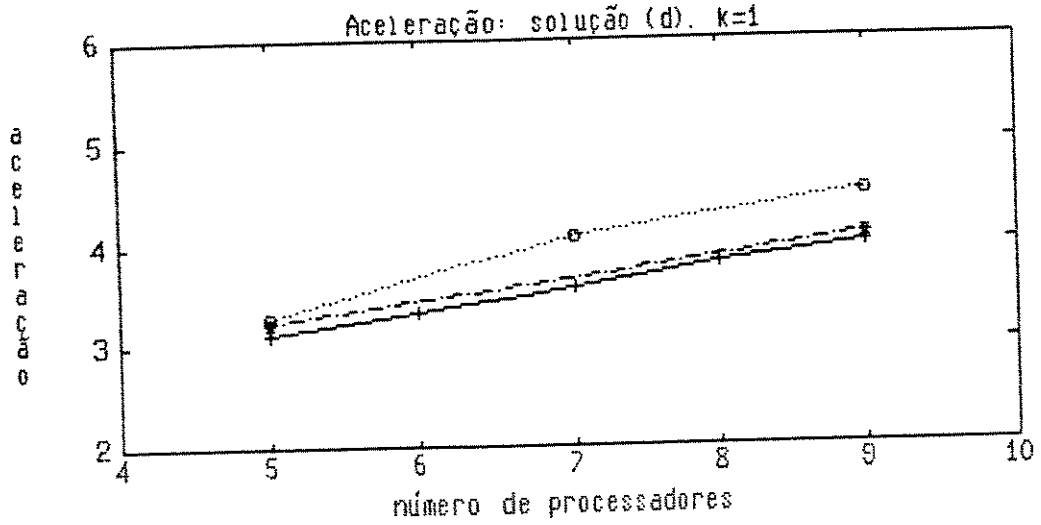


Figura 5.10. Aceleração da solução (d) para as topologias: anel ("---"), hipercubo ("-.-") e malha ("...") e k = 1.

CONDIÇÕES DO TESTE	
Resolução da Imagem	200x256x8
Resolução do Espaço Hough	256x326
Limiar de Binarização	60
Tempo de carga da imagem	2,7561 s
Velocidade dos links	20 MHz
Classe de Arquitetura	MIMD
Tamanho do Pacote de Mensagens	70
T(1); k = 1	1,677 s
T(1); k = 20	2,179 s

Tabela 5.8. Variáveis fixadas para a execução dos testes com as soluções (d) e (e) na rede de transputers.

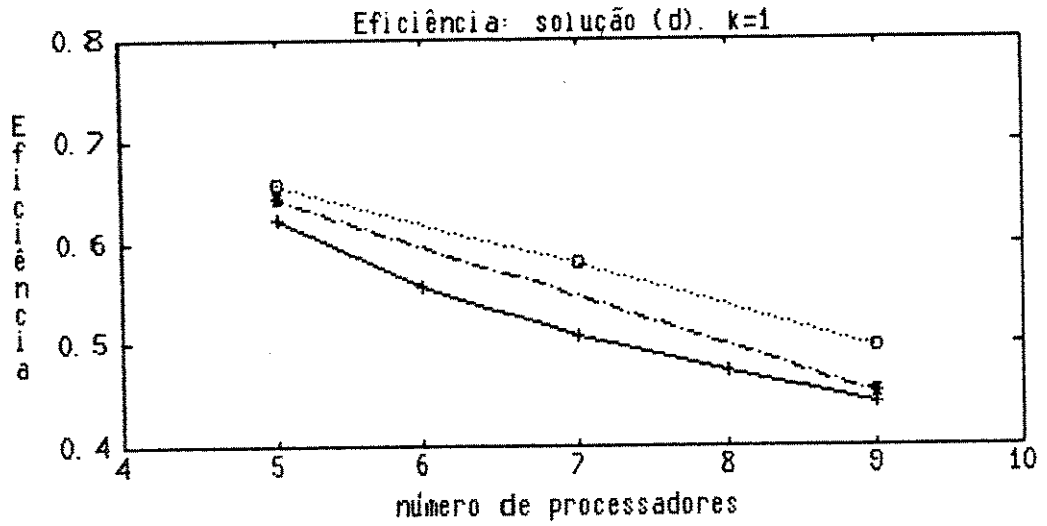


Figura 5.11. Eficiência da solução (d) para 3 topologias: anel ("---"), hipercubo ("-.-") e malha ("...") e k = 1.

Conforme se observa nas figuras 5.10 e 5.11, na solução (d), para k = 1, as curvas de aceleração possuem taxa de crescimento aproximadamente constante e não apresentam comportamento de saturação, para este intervalo de n.

A topologia anel possui os valores de aceleração mais baixos dentre as três topologias consideradas.

Os valores de aceleração conseguidos com a topologia hipercúbica são piores que os valores obtidos para a topologia malha com "wrap-around".

A topologia malha com "wrap-around" apresenta bons resultados de eficiência ($\cong$ 50%) e possui os melhores resultados dentre as três topologias.

As figuras 5.12 e 5.13 apresentam as curvas de aceleração e eficiência, respectivamente, da solução (d) em função do número de processadores para as três topologias: anel, hipercubo e malha com "wrap-around" e para k = 20.

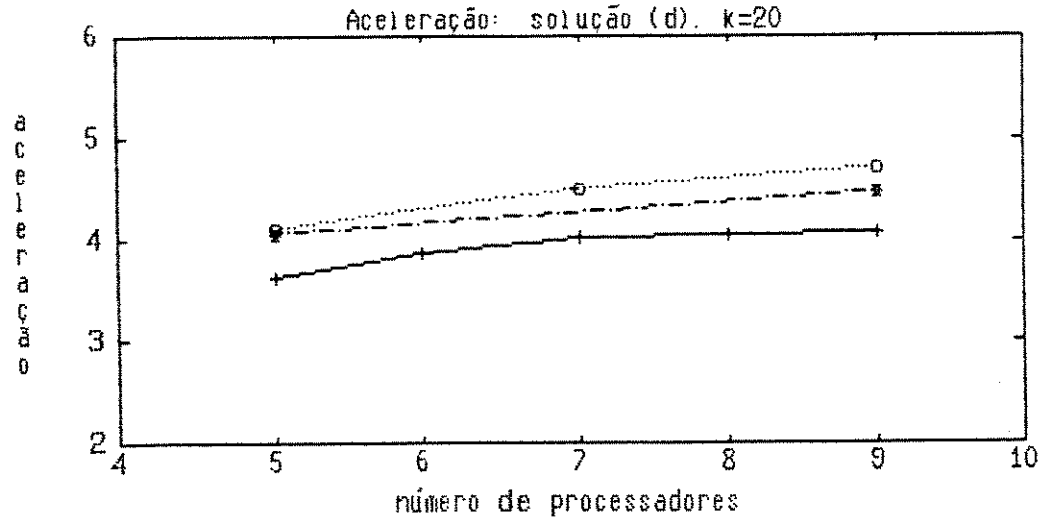


Figura 5.12. Aceleração da solução (d) para as topologias: anel ("---"), hipercubo ("-.-") e malha ("...") e k = 20.

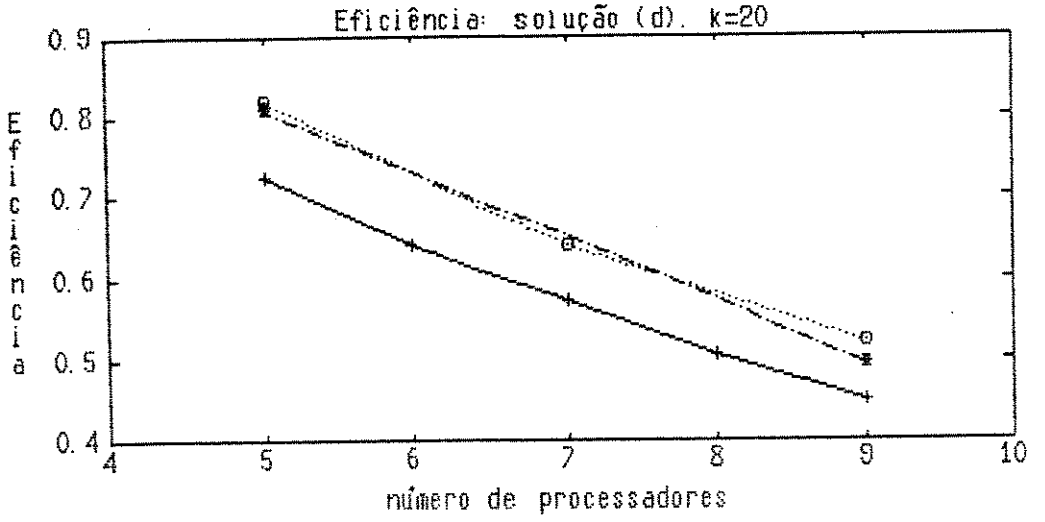


Figura 5.13. Eficiência da solução (d) para 3 topologias: anel ("---"), hipercubo ("-.-") e malha ("...") e k = 20.

Conforme se observa nas figuras 5.12 e 5.13, a solução (d), para k = 20, e para a topologia anel, as curvas de aceleração tendem a saturar para n superior a 8, indicando que o congestionamento de mensagens torna-se mais significativo no tempo total de execução. Na topologia anel apresenta a pior curva de aceleração para esta solução.

As figuras 5.14 e 5.15 apresentam as curvas de aceleração e eficiência da solução (e) em função do número de processadores para três topologias: árvore, malha com "wrap-around" e hipercubo para valores de k = 1. A utilização da topologia árvore em substituição à topologia anel se justifica por dois motivos: (1) A topologia anel apresentou até aqui os piores resultados de aceleração; (2) A topologia árvore é reconhecidamente - INMOS (1989) - uma boa opção para se implementar a solução do tipo "processor farm".

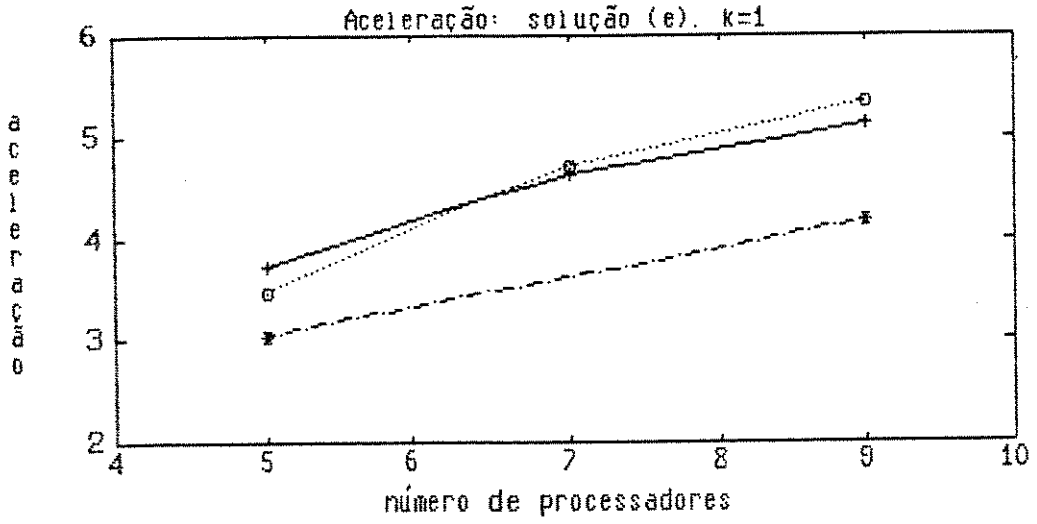


Figura 5.14. Aceleração da solução (e) para as topologias: árvore("---"), hipercubo ("-.-") e malha ("..."), para k = 1.

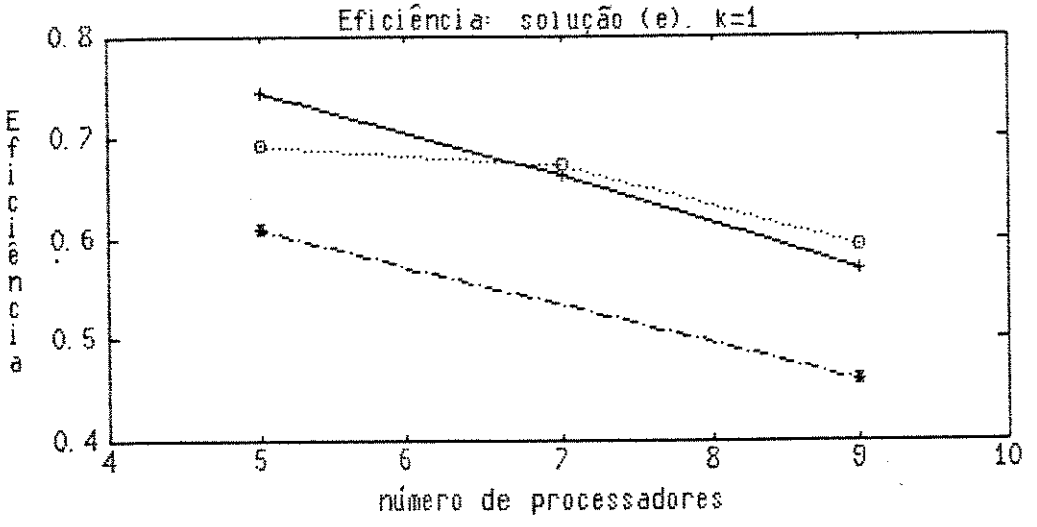


Figura 5.15. Eficiência da solução (e) para as topologias: árvore("---"), hipercubo ("-.-") e malha ("..."), para k = 1.

Conforme se observa nas figuras 5.14 e 5.15, a solução (e), para k = 1, a topologia hipercúbica possui os menores valores de aceleração entre as três topologias. Para n menor que 6 processadores, a topologia árvore apresenta valores maiores de aceleração em comparação à topologia malha com "wrap-around". Para n maior que 6 processadores, por fazer uso de todos os links disponíveis, a topologia malha com "wrap-around" consegue "desafogar" o trânsito de mensagens pela rede de transputers, diminuindo o tempo de execução da solução (e). Todas as topologias do tipo árvore têm inerentemente um congestionamento de mensagens nos links próximos à base (raiz). Quanto maior a árvore e quanto maior o número de mensagens trafegando por ela, maior o congestionamento e pior o desempenho final.

Pode-se notar também que, de um modo geral, o desempenho desta solução, para as topologias malha com "wrap-around" e hipercúbica, é superior ao desempenho da solução (d) para k=1.

As figuras 5.16 e 5.17 apresentam as curvas de aceleração e eficiência da solução (e) em função do número de processadores para três topologias: árvore, hipercubo e malha com "wrap-around" para valores de k = 20.

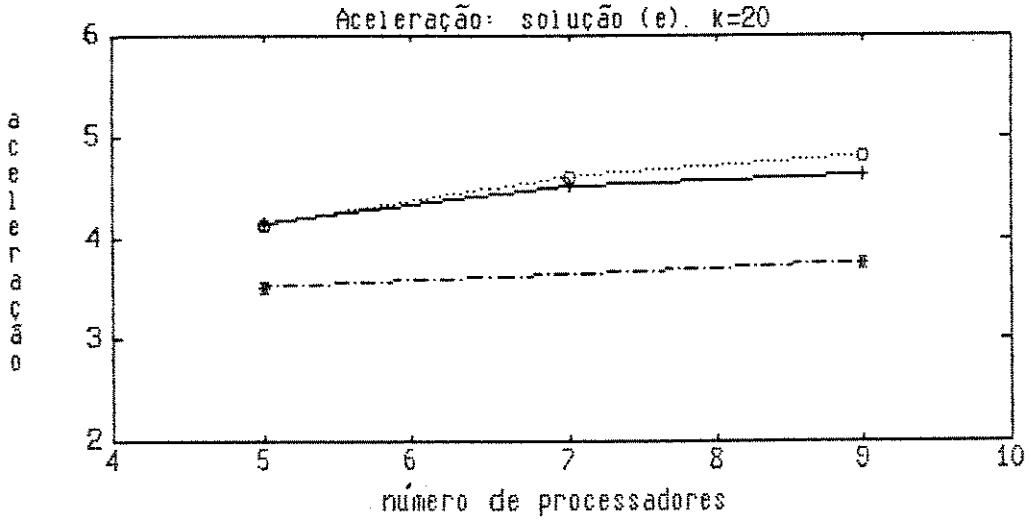


Figura 5.16. Aceleração da solução (e) para as topologias: árvore("---"), hipercubo ("-.-") e malha ("..."), para k = 20.

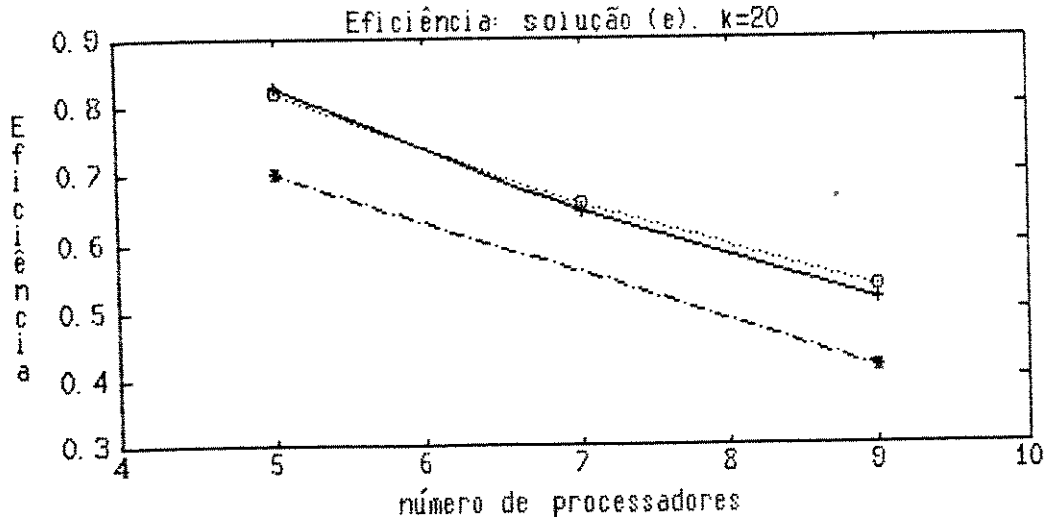


Figura 5.17. Eficiência da solução (e) para as topologias: árvore("---"), hipercubo ("-.-") e malha ("..."), para k = 20.

Como pode ser observado nas figuras 5.16 e 5.17, a solução (e), para k = 20, os valores de aceleração são menores que aqueles obtidos para a mesma solução e k=1. Isto se deve ao fato do tamanho do pacote ser proporcional a k e, como se sabe, quanto maior a mensagem, maior a probabilidade de haver congestionamento de mensagens na topologia. Em relação à solução (d) e k=20, os valores de aceleração e eficiência são superiores. Isto acontece pela diferença estrutural das duas soluções, apontada na seção 4.4.1: a solução (e) faz uma sobreposição de mensagens "escravo→mestre" e "mestre→escravo" enquanto na solução (d) isto não ocorre.

Nota-se também o comportamento semelhante das duas topologias: malha com "wrap-around" e árvore. Isto indica que, com o crescimento de k, aumenta-se o custo de comunicação de mensagens pela topologia. E com isso, as topologias que possuem maior capacidade de roteamento de dados - ou seja, maior número de links - têm maior possibilidade de "desafogarem o congestionamento". Extrapolando-se os dados das figuras 5.14 e 5.16, espera-se que, para k maiores que 20, a topologia malha com "wrap-around" tenha uma curva de aceleração com comportamento superior ao da topologia árvore.

Os dados sobre o desempenho máximo das duas soluções paralelas (d) e (e) da seção 4.4.1 são apresentados nas tabelas 5.9 a 5.12, para dois valores de k (1 e 20). As tabelas completas dos tempos de execução estão no apêndice 4.

k = 1	Solução (d)		
	anel n=9	malha n=9	hipercubo n=9
tempo de execução	0,421 s	0,374 s	0,412 s
Aceleração	3,983	4,484	4,070
Eficiência	0,443	0,499	0,452

Tabela 5.9. O desempenho máximo da transformada direcional de Hough, solução (d), seção 4.4.1.

k = 20	Solução (d)		
	anel n=9	malha n=9	hipercubo n=9
tempo de execução	0,536 s	0,463 s	0,487 s
Aceleração	4,065	4,706	4,474
Eficiência	0,452	0,523	0,497

Tabela 5.10. O desempenho máximo da transformada direcional de Hough, solução (d), seção 4.4.1.

Como se observa das tabelas 5.9 e 5.11, a transformada de Hough que utiliza a informação direcional - com k = 1 - é executada com maior velocidade utilizando-se a solução (e) sobre uma topologia em malha com "wrap-around". Conforme a tabela 5.9, para k = 1, a melhor aceleração obtida é 4,484, também com essa malha. A melhor eficiência conseguida foi com a topologia árvore com cinco processadores (± 75%), conforme a figura 5.11.

k = 1	Solução (e)		
	anel n=9	malha n=9	hipercubo n=9
tempo de execução	0,327 s	0,314 s	0,403 s
Aceleração	5,128	5,341	4,161
Eficiência	0,570	0,593	0,462

Tabela 5.11. O desempenho máximo da transformada Direcional de Hough, solução (e), seção 4.4.1.

k = 20	Solução (e)		
	anel n=9	malha n=9	hipercubo n=9
tempo de execução	0,471 s	0,453 s	0,583 s
Aceleração	4,626	4,810	3,738
Eficiência	0,514	0,534	0,415

Tabela 5.12. O desempenho máximo da transformada Direcional de Hough, solução (e), seção 4.4.1.

Conforme apresentado nas tabelas 5.10 e 5.12, para k = 20, a solução (e) apresenta o melhor valor de aceleração (4,810) para a topologia malha com "wrap-around". Todavia, a solução (d), com 9 processadores e topologia malha com "wrap-around" apresenta valores maiores de aceleração que a solução (e) com 9 processadores e topologia hipercúbica ou árvore. Isto indica que a solução (e) deve possivelmente apresentar um comportamento, para n maior que 9, pior que a solução (d).

O tempo de execução, para as soluções (d) e (e), é menor que o obtido através da transformada clássica de Hough principalmenete porque há menos pares (θ,ρ) sendo calculados e transferidos na rede. Na solução "farm" há uma sobreposição entre entrada e saída de mensagens, enquanto que nas soluções (c) e (d) nenhuma resposta chega da rede de escravos antes do último pacote de dados ter sido enviado pelo mestre. O baixo número de mensagens faz com que a contenção diminua e, consequentemente, aumente a velocidade de execução.

Através dos resultados desta seção, conclui-se que, para 9 processadores, a solução (e) é superior às outras quando k é pequeno e, consequentemente, o número de mensagens trafegando pela rede é menor. Para k = 20, as soluções (d) e (e) têm comportamentos muito semelhantes, representados pelos valores de aceleração: 4,706 e 4,810, respectivamente. E também, para 9 processadores, a topologia malha com "wrap-around" é a que apresenta melhor resultado médio.

5.6 - Utilizando Arquitetura SIMD

Esta seção apresenta os resultados da implementação de alguns algoritmos paralelos, de Visão Computacional de nível baixo na máquina SIMD Connection Machine, modelo CM-2, da Thinking Machines Corp. com 32K processadores. Os softwares utilizados para programação foram: C* e Paris/C, conforme TMC (1990).

Foram executados alguns testes utilizando-se a Connection Machine para executar as fases de nível baixo do algoritmo desenvolvido no capítulo 2. Os algoritmos implementados foram: (a) a convolução com as máscaras do detector de bordas Laplaciano e de Sobel, (b) o afinamento, descrito na seção 4.3 e (c) a versão massivamente paralela da transformada de Hough, descritas na seção 4.4.2. Os tempos de execução apresentados nas tabelas de resultados desta seção não incluem o tempo de carga da imagem, proveniente do disco, para a RAM.

Quando se utiliza a Connection Machine são necessários outros critérios de desempenho dos algoritmos paralelos implementados. Não é possível conseguir T(1) - que, conforme a seção 3.2, denomina o tempo de execução do algoritmo em um único processador - na CM. As medidas mais comuns de desempenho são a Utilização dos processadores e o Tempo de execução versus o VP_Ratio. A primeira delas mostra quantos

processadores se tornam inativos com o correr das iteraçções de um algoritmo. A segunda medida - que pode ser vista como uma espécie de Aceleração - mede a vantagem temporal de uma versão do algoritmo rodando em diversos processadores (sempre potências de 2). Na sequência desta seção serão vistos a utilização destes parâmetros de desempenho em algumas fases do algoritmo do capítulo 2.

O algoritmo do detector de bordas Laplaciano, mostrado na figura 4.7, utiliza a rede hipercúbica de comunicação para obter o valor dos pixels dos quatro vizinhos e executa as operações de multiplicação internamente a cada elemento processador simultaneamente. A convolução com as máscaras do operador de Sobel também é implementada de maneira semelhante. A tabela 5.13 apresenta os resultados de aceleração deste algoritmo para N = 8K, 16K e 32K processadores. O parâmetro VP_Ratio está definido na seção 3.1.2.

VP_Ratio	N	Tempo de Execução (Laplaciano)	Tempo de Execução (Sobel)
8	8K	0,0439 s	0,497 s
4	16K	0,0260 s	0,293 s
2	32K	0,0113 s	0,167 s

Tabela 5.13. Tempos de Execução dos operadores de Laplace e de Sobel sobre uma imagem 256x256x8 e com complexidade $\cong$ 20%.

Conforme pode ser visto na tabela 5.13, para uma imagem com 64K pixels, consegue-se com 16K (VP_Ratio = 4) e 32K (VP_Ratio = 2) processadores aplicar o operador de Laplace em tempo-real, ou seja, com tempo de execução menor que 33ms. O mesmo acontece com o operador de Sobel.

O algoritmo de afinamento, mostrado na figura 4.8, é apropriado para sua implementação na CM, pois o resultado de uma iteração é função unicamente dos vizinhos 8-conectados na iteração anterior. A figura 5.15 mostra a utilização dos processadores durante a execução do algoritmo.

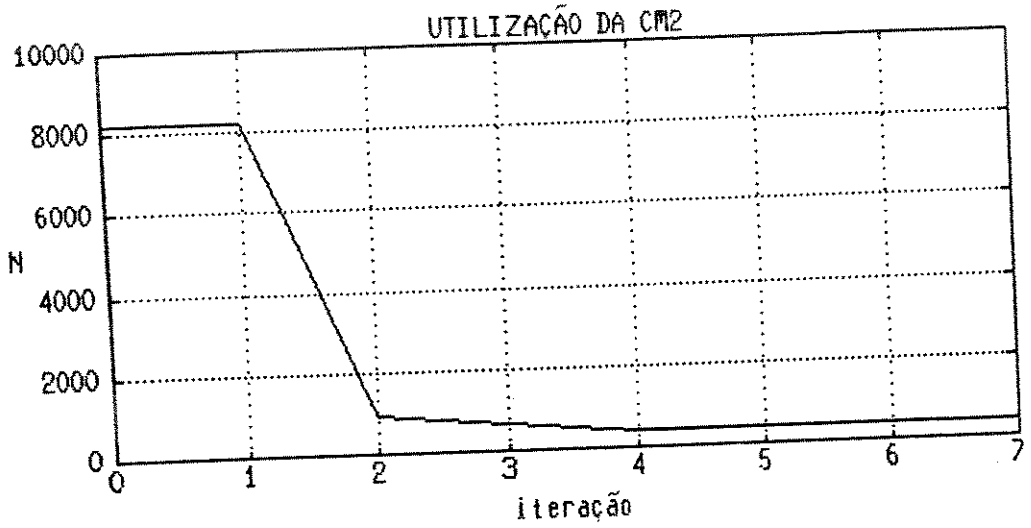


Figura 5.15. O valor da variável C, da figura 4.8a, mostrando a utilização dos processadores durante as iterações.

A tabela 5.14 apresenta os tempos de execução deste algoritmo de afinamento massivamente paralelo em função do número de processadores. Conforme mostra a figura 5.15, este algoritmo convergiu em 5 iterações, para uma imagem 256x256x8 e com complexidade ≅ 20%.

VP_Ratio	N	Tempo de Execução
8	8K	0,0545 s
4	16K	0,0397 s
2	32K	0,0172 s

Tabela 5.14. Tempos de Execução do algoritmo de Afinamento sobre uma imagem 256x256x8 e com complexidade ≅ 20%.

Como se observa na tabela 5.14, quando se aumenta o número de processadores e, consequentemente, diminui-se o VP_Ratio, os tempos de execução se tornam menores. Em particular, para 32K processadores, este algoritmo pode rodar em tempo-real.

A transformada de Hough utilizando o modelo de dados paralelos foi implementada conforme apresentada na seção 4.4.2. Esta implementação apresenta um modo de se abordar a transformada de Hough utilizando-se um processamento massivamente paralelo. A utilização dos processadores é alta e o tempo de execução é muito pequeno.

A tabela 5.15 mostra os tempos de execução dos três algoritmos para uma imagem de entrada de 256x256x8 com 1024 pixels de borda e resolução do espaço Hough de 128x128

VP_Ratio	N	Tempo de Execução
8	8K	0,0885 s
4	16K	0,0697 s
2	32K	0,0331 s

Tabela 5.15. Tempos de Execução da Implementação Massivamente Paralela da transformada de Hough, apresentada nas figuras 4.15 a 4.17, e para uma resolução de Espaço Hough de 128x128

Os valores apresentados nas tabelas 5.13 a 5.15 são influenciados pelo "time-sharing" do sistema operacional. Note-se que na tabela 5.15 a resolução do espaço Hough é de 128x128. Como pode ser observado na tabela 5.15, para uma imagem de entrada com 1024 pixels de borda, o menor tempo de excução foi conseguido com 32K processadores onde consegue-se que o algoritmo rode - para essa imagem - em 33,1ms.

5.7 - Desempenho em Tempo-Real

Nesta seção será aplicada a segunda abordagem de cálculo da aceleração, conforme foi visto na seção 3.2, em máquinas MIMD. A solução escolhida para se fazer estes testes é aquela apresentada na figura 4.21. Conforme visto na seção 4.5, esta solução executa uma sobreposição entre a Transformada Direcional de Hough e a fase de Classificação das Retas.

Para os testes com esta solução, fixou-se o tempo de execução em 25,0 ms pois, de acordo com os dados da tabela 4.1, este tempo somado ao da 5ª e 7ª fases e supondo-se um circuito VLSI detector de bordas, determina um tempo total de execução de, aproximadamente, 33ms.

Para se obter a curva de aceleração através desta abordagem, entra-se com um número crescente de pares (θ, x, y) - representando os pixels de borda - e mede-se o tempo de execução da solução, para as várias topologias configuradas. A curva apresentada na figura 5.16 mostra, para diferentes números de processadores e topologias, o número máximo de pares (θ, x, y) que podem ser tratados pela solução em menos de 25,0 ms.

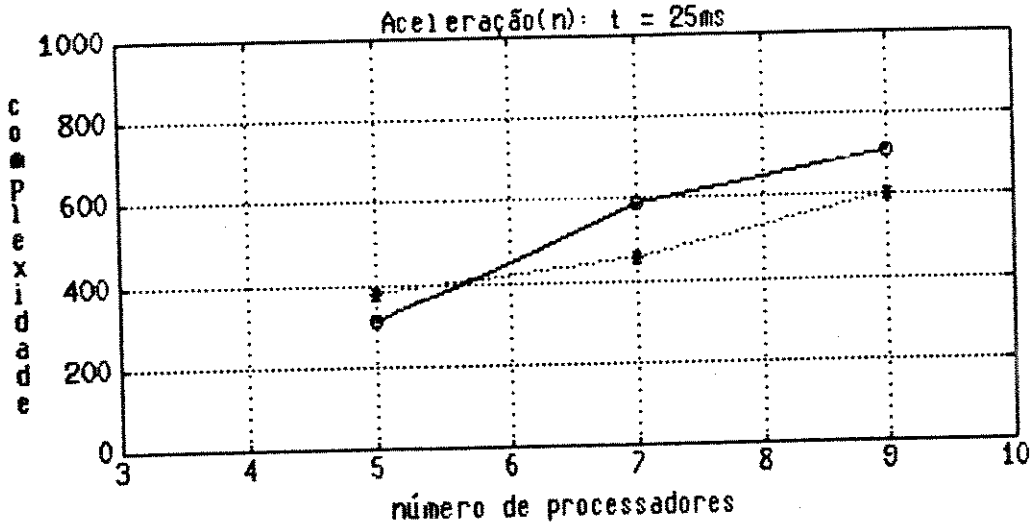


Figura 5.16. Gráfico da aceleração da solução apresentada na seção 4.5, para t = 25ms. As condições dos testes são as da figura 5.8. As topologias adotadas foram: Malha (---) e Árvore (...).

A figura 5.16 mostra o resultado da implementação da solução "Processor-Farm", apresentada na seção 4.5, em duas topologias. A topologia malha com "wrap-around", com 9 transputers permite a execução deste algoritmo em tempo-real para imagens com 706 pixels de borda. A topolgia árvore possui maior aceleração para n menor que 6.

Ao contrário das curvas de aceleração obtidas na seção 5.5.2, as estimativas teóricas da seção 3.2 apontam para um formato de curva sem saturação, ou seja, sempre será possível encontrar um número n de processadores que resolva o problema em um tempo pré-fixado.

De uma maneira mais prática, com 9 processadores, basta alterar qualquer variável (de processamento ou de imagem) para que imagens mais complexas também possam ser executadas em tempo-real.

CAPÍTULO VI

Conclusões Finais

Esta dissertação procurou mostrar a aplicação de técnicas de processamento paralelo ao problema de Visão Computacional denominado "Reconhecimento e Localização de Objetos no Espaço 3D".

Foi desenvolvido e implementado um algoritmo para resolver este problema no caso de objetos do tipo paralelpípedo. Técnicas de Processamento Paralelo foram desenvolvidas e implementadas ao algoritmo permitindo sua execução dentro das especificações definidas no capítulo 1.

Na sequência serão apresentados alguns comentários e conclusões relevantes sobre o trabalho desenvolvido bem como perspectivas de continuação, otimização e extensão do mesmo.

6.1 - Sobre o Algoritmo Serial

Na fase de detecção de bordas, três operadores matemáticos foram estudados. A detecção de bordas de imagens reais através do operador de Sobel apresentou o melhor compromisso entre a imunidade ao ruído e o tempo de processamento, conforme apresentado na seção 2.2.1. O operador Laplaciano, embora seja mais rápido, realça o conteúdo de alta frequência da imagem, amplificando ruídos indesejáveis. O operador de Marr apresenta a melhor resposta - dispensando a fase de afinamento - mas é o mais lento devido à necessidade de operaçãoes em ponto flutuante e utilização de várias máscaras.

Para as imagens utilizadas nos testes, o algoritmo de afinamento geralmente converge em 5 iterações. Todavia, conforme apontado na seção 2.2.2, dependendo do operador de detecção de bordas utilizado e o histograma da imagem de entrada, esta fase pode não ser necessária. Uma imagem capturada em um ambiente com iluminação controlada possui um histograma praticamente bimodal, garantindo uma boa detecção de bordas. Os testes com o algoritmo paralelo dispensaram esta fase do

algoritmo serial.

A transformada de Hough "clássica", conforme comentado na seção 2.2.3, possui algumas deficiências como: sub-utilização da memória e indistinguibilidade de segmentos de retas colineares. A transformada de Hough que utiliza a informação direcional do operador de Sobel não é muito precisa, embora rápida. Um compromisso entre as duas transformadas foi implementado e consiste no cálculo da transformada de Hough de k ângulos nas vizinhanças do gradiente nominal.

A fase de classificação das retas - descrita na seção 2.2.4 - é dividida em duas sub-fases: (a) segmentação das nuvens de pontos do espaço Hough e (b) modelamento dos dados. A sub-fase de segmentação é, inicialmente, implementada através de uma busca no espaço Hough. A partir da segunda imagem da sequência, pode-se executar esta busca apenas na vizinhança do pico detectado na imagem anterior, diminuindo o tempo de execução.

A fase de cálculo das propriedades, apresentada na seção 2.2.5, faz uso das simetrias do paralelepípedo para identificar unicamente, as equações suporte de cada uma das arestas do objeto. É devido a estas simetrias do modelo que esta fase não é demorada. Para objetos mais complexos o tempo de execução desta fase aumenta.

O paralelepípedo foi escolhido por permitir o cálculo dos pontos de fuga da imagem e por apresentar simetrias que facilitam sua identificação pela transformada de Hough.

As cinco propriedades das estruturas dos picos das nuvens no espaço Hough - seção 2.2.5 - fornecem um ponto de partida para o reconhecimento de objetos mais complexos. Em particular, a propriedade 4 mostra como o algoritmo pode ser modificado para detectar pirâmides ou cantos de objetos. O reconhecimento de esferas também pode ser feito utilizando-se uma extensão da Transformada de Hough para detectar círculos (BALLARD, 1981).

A fase de ajuste das equações das retas foi a solução encontrada para melhorar a precisão das equações das retas suporte das arestas do paralelepípedo. Utilizando-se este artifício pode-se diminuir a quantização do espaço Hough e ainda conseguir-se uma boa precisão do resultado final.

A fase de calibração apresenta uma precisão de 2,0-5,0 cm em 1,2-1,5m de profundidade na localização do objeto no espaço e de 2° na sua orientação. A precisão é maior quanto maior for a relação "pixels/aresta" detectados, ou seja, quanto maior forem as arestas no plano-imagem. Estes dados são resultantes da aplicação do algoritmo em **imagens artificiais** e **imagens reais**.

O algoritmo serial desenvolvido possui grande portabilidade, já tendo sido implementado em várias máquinas uniprocessadoras como base de trabalho para outras pesquisas.

6.2 - Sobre a Implementação de Algoritmos Paralelos

A interconexão de um grande número de elementos processadores apresenta vários problemas teóricos e de engenharia. Mecanismos de comunicação e comportamento de sistemas paralelos têm sido extensamente estudados, mas metodologias para a implementação e avaliação de redes de interconexão ainda não existem. A definição da topologia é, portanto, uma tarefa que necessita ser guiada pela aplicação e pelo programador.

Os modelos paralelos de linguagens têm sido a base teórica de um grande número de liguagens paralelas. O modelo VRAM (de Vector Random Access machine, seção 3.3.2) apresenta um ambiente simples e produtivo para a programação de máquinas massivamente paralelas. Já o modelo PRAM (de Parallel Random Access Machine, seção 3.3.2) e o modelo CSP (de Concurrent Sequential Processes, HOARE, 1987) se ajustam bem a um conjunto menor de processadores. Estes modelos procuram, dentre outros objetivos, um consenso em linguagens paralelas. Todavia, este consenso só poderá ser alcançado quando se tiver uma padronização maior nas arquiteturas paralelas.

OCCAM e C* estão dentre as linguagens paralelas mais utilizadas na atualidade. A linguagem OCCAM possui muitas qualidades, pois é praticamente a implementação direta da teoria dos Processos Sequenciais Comunicantes ("CSP", de Hoare, 1987). Por exemplo, evita-se muitos problemas tradicionais impostos pelo paralelismo, como interferência, exclusão mútua, semáforos, etc. Todavia, a linguagem

OCCAM ainda está em sua "infância" e necessita evoluir muito para se disseminar e se tornar mais prática. A linguagem C*, por outro lado, procurou estender-se a partir de uma linguagem bem disseminada para facilitar sua aceitação e, portanto, já pode ser considerada uma linguagem "adulta".

Várias ferramentas de software têm sido desenvolvidas para melhorar a interação entre o programador e o computador paralelo. Contudo ainda não há uma padronização de linguagens paralelas nem de sistemas operacionais paralelos. E para explorar-se cada máquina paralela é, geralmente, necessário aprender uma linguagem específica e um sistema operacional específico da máquina.

6.3 - Sobre as Arquiteturas Paralelas Utilizadas

Vários tipos de arquiteturas paralelas foram estudados e, devido à disponibilidade de uma arquitetura MIMD fracamente acoplada e uma arquitetura SIMD, optou-se pela avaliação do algoritmo de Visão Computacional nessas duas máquinas multiprocessadoras. As etapas do algoritmo de reconhecimento e localização de objetos no espaço escolhidas para serem paralelizadas foram implementadas em uma malha de Transputers e na Connection Machine.

A malha de transputers fornece a possibilidade de sua configuração em diversas topologias fornecendo mais um grau de liberdade para o pesquisador/programador testar seus programas. Contudo, com o aumento do número de processadores, a necessidade de configurar uma malha grande de processadores se torna proibitiva. Desse modo, máquinas paralelas com um grande número de processadores têm a tendência de ter uma topologia fixa; há excessões.

A Connection Machine emprega a maior parte de seu custo na rede de comunicação entre seus muitos nós processadores e deixa por conta do software balancear o *processamento numérico* e as *comunicações*, para se conseguir um melhor desempenho. Um dos problemas com a Connection Machine é que o tamanho de sua memória por nó ainda é pequeno. Máquinas MIMD, por outro lado, executam esta fase de processamento numérico muito mais rápido do que transferem dados entre seus nós. Portanto, nesse caso, a tarefa mais importante a se assegurar é que exista uma grande quantidade de processamento numérico em cada nó e

que esta seja balanceada entre os nós, minimizando a comunicação.

6.4 – Sobre as Soluções Paralelas Encontradas

A tarefa de paralelização de um algoritmo deve iniciar-se pela análise do grafo de dependência das tarefas envolvidas. Em seguida deve-se verificar quais as variáveis que podem ser controladas para se modificar o desempenho do algoritmo. Faz-se, então, a opção entre a adaptação do algoritmo ou sua total reprogramação. Esta escolha deve-se basear também na arquitetura-alvo onde o algoritmo deve ser executado.

O paradigma de Visão Computacional é sequencial. Portanto, no caso deste trabalho, a paralelização entre estágios é muito mais complexa. Optou-se por acelerar o algoritmo através da exploração do paralelismo interno aos estágios.

Algumas técnicas de programação de algoritmos paralelos foram discutidas nas seções 3.3.1 e 3.3.2. Entre elas destacam-se a técnica *sistólica*, a técnica *massivamente paralela* e a técnica que aproveita o *paralelismo espacial* do algoritmo. Nesta última - aplicada na malha de transputers - algumas variáveis de controle como tamanho do pacote de mensagens, velocidade dos links de comunicação, etc têm importância muito grande no desempenho final do algoritmo.

A linguagem OCCAM fornece a possibilidade de se desenvolver o algoritmo em apenas um transputer e, posteriormente reconfigurá-lo em uma malha de transputers mapeando-se logicamente os canais utilizados. Contudo, esta tarefa somente se torna simples se alguns dos módulos particionados ("SC's") forem inteiramente independentes uns dos outros.

Do estudo realizado no capítulo 4, nota-se que a transformada de Hough é a etapa mais demorada do algoritmo. Por esta razão as pesquisas em estratégias de particionamento e distribuição ("paralelização") desta etapa foram de vital importância na execução do algoritmo de reconhecimento e localização de objetos no espaço 3D.

6.5 - Sobre os Resultados Experimentais Obtidos

Alterando-se a distância entre a câmera e a origem do sistema de referência (vértice P_3), encontrou-se que à medida que esta distância aumenta, aumenta também o desvio médio relativo de posição. Quando se fixa esta distância, a precisão dos parâmetros extrínsecos calculados aumenta com o aumento da distância focal da câmera.

A precisão dos parâmetros extrínsecos de orientação são maiores quanto maiores forem os comprimentos das bordas detectadas. Ou seja, quanto mais os ângulos se aproximam de $(\theta,\phi,\psi) = (-45^\circ,-45^\circ,0^\circ)$.

A avaliação da paralelização de algoritmos paralelos foi feita de duas maneiras: (1) Fixando-se o tamanho dos dados do algoritmo e variando-se o número de processadores (seção 5.5); e (2) Fixando-se o tempo de execução e escalando-se o tamanho dos dados do algoritmo para satisfazer a especificação de tempo (seção 5.7). A utilização do primeiro método exige que se tenha uma máquina com muitos processadores ou que se faça uma simulação - conforme SOARES (1990) ou PIZARRO (1991) - para se tirar a "curva de saturação" do algoritmo. Na paralelização de algoritmos visando sua execução em tempo-real (definido no capítulo 1) opta-se geralmente pelo segundo método, embora neste trabalho ambos os métodos tenham sido aplicados. Este segundo método é, portanto, mais "otimista" por mostrar a possibilidade de execução em tempo-real com exatamente os recursos computacionais disponíveis.

Para a paralelização da Transformada Clássica de Hough a solução (c) "Partição da Imagem e Partição do Espaço Hough" apresentou os melhores resultados.

A técnica de "processor farm" - chamda de solução (e) - mostrou-se a mais eficiente para a paralelização do algoritmo da Transformada Direcional de Hough, utilizando-se 9 transputers. O principal motivo deste fato é que há um balanço dinâmico entre a entrada e saída dos dados.

A topologia malha, com 9 transputers permite a execução do algoritmo de Visão Computacional em tempo-real de imagens com até 706 pixels de borda, conforme apontado na seção 5.7.

Utilizando-se o modelo VRAM, o algoritmo da transformada de Hough foi reprogramado, mostrando uma outra abordagem para a programação de algoritmos paralelos em arquiteturas massivamente paralelas.

6.6 - Sobre as Aplicações e Extensões do Trabalho

As técnicas desenvolvidas neste trabalho podem ter aplicações em outras áreas que trabalham com imagens digitais. Grande parte dos procedimentos de Visão Computacional de nível baixo - como filtragem, detecção de bordas e segmentação - podem ser acelerados aplicando-se técnicas de processamento paralelo que foram utilizadas neste trabalho.

Alguns problemas relacionados ao algoritmo de reconhecimento e localização de objetos no espaço 3D ainda permenecem em aberto. Por exemplo: (1) Neste trabalho, os testes com o algoritmo foram "estáticos", ou seja, não havia movimento relativo entre o objeto e a câmera. Conforme apontado na seção 5.2, seria necessário utilizar uma câmera com foco variável (ou auto-foco) para se aplicar o algoritmo "dinamicamente" e avaliar a influência de outros fatores, como as possíveis distorções ópticas, nos resultados numéricos obtidos com imagens reais. (2) Se existir movimento relativo entre o objeto e a câmera, as arestas do objeto apresentam-se "tortas" na imagem, prejudicando a extração das respectivas equações de reta. É portanto necessário obter informações (precisas) sobre o movimento da câmera para corrigir tais distorções.

Uma maneira de se abordar estes problemas em aberto é através da utilização de informações obtidas de sensores de diversos tipos - por exemplo, sensores de estado interno, sensores de contato, sensores de não-contato, sensores de visão computacional ou sensores dependentes da aplicação - para poder **fazer inferências** sobre o ambiente e **tomar decisões** levando em consideração esta maior gama de informações. Esta técnica é chamada de **Integração de Sensores**, conforme BRADY (1989).

Este trabalho foi uma contribuição na direção da utilização de Visão Computacional em Robôs, dotando-os de capacidade de Visão 3D e possibilitando assim sua aplicação potencial em ambientes industriais

como, por exemplo, em **inspeção** e **navegação automáticas**.

Extensões deste trabalho podem seguir por dois caminhos principais: (a) Processamento paralelo e (b) Aplicação do algoritmo.

(a) **Processamento Paralelo**. Pode-se aumentar o número de processadores e fazer as mesmas análises da seção 5.5 para outras topologias. Ou utilizar o algoritmo paralelo em imagens com complexidades maiores, conforme análises feitas na seção 5.7.

(b) **Aplicação do Algoritmo**. Pode-se estender o algoritmo para outros tipos de objetos, outros métodos de calibração e utilizar outros métodos de reconhecimento de objetos e segmentação. Por exemplo, ver DHOME e KASWAND (1989).

APÊNDICE 1

O Laplaciano de uma Gaussiana. (LoG)

O operador Laplaciano é definido por

$$\nabla^2 I(x,y) = \frac{\partial^2 I(x,y)}{\partial x^2} + \frac{\partial^2 I(x,y)}{\partial y^2}$$

E seja uma Gaussiana genérica em duas dimensões:

$$G(x,y) = \frac{1}{2.\pi.\sigma_1.\sigma_2} . e^{\left[\frac{-x^2}{2.\sigma_1^2} - \frac{-y^2}{2.\sigma_2^2}\right]}$$

Calculando-se a primeira derivada, tem-se

$$\frac{\partial G(x,y)}{\partial x} = \frac{-1}{2.\pi.\sigma_1.\sigma_2} . e^{\left[\frac{-x^2}{2.\sigma_1^2} - \frac{-y^2}{2.\sigma_2^2}\right]} . \frac{2.x}{2.\sigma_1^2}$$

e

$$\frac{\partial G(x,y)}{\partial y} = \frac{-1}{2.\pi.\sigma_1.\sigma_2} . e^{\left[\frac{-x^2}{2.\sigma_1^2} - \frac{-y^2}{2.\sigma_2^2}\right]} . \frac{2.y}{2.\sigma_2^2}$$

Calculando a segunda derivada, tem-se:

$$\frac{\partial^2 G(x,y)}{\partial x^2} = \frac{-1}{2.\pi.\sigma_1.\sigma_2} . e^{\left[\frac{-x^2}{2.\sigma_1^2} - \frac{-y^2}{2.\sigma_2^2}\right]} . \left[1 - \frac{2.x^2}{2.\sigma_1^2}\right]$$

e

$$\frac{\partial^2 G(x,y)}{\partial y^2} = \frac{-1}{2.\pi.\sigma_1.\sigma_2} \cdot e^{\left[\frac{-x^2}{2.\sigma_1^2} - \frac{-y^2}{2.\sigma_2^2}\right]} \cdot \left[1 - \frac{2.y^2}{2.\sigma_2^2}\right]$$

Somando-se os termos, tem-se:

$$\nabla^2 G(x,y) = \frac{-1}{2.\pi.\sigma_1.\sigma_2} \cdot e^{\left[\frac{-x^2}{2.\sigma_1^2} - \frac{-y^2}{2.\sigma_2^2}\right]} \cdot \left[2 - \frac{x^2}{\sigma_1^2} - \frac{y^2}{\sigma_2^2}\right]$$

$$\nabla^2 G(x,y) = \left[\frac{-1}{2.\pi.\sigma_1^3.\sigma_2} + \frac{x^2}{2.\pi.\sigma_1^5.\sigma_2} - \frac{1}{2.\pi.\sigma_1.\sigma_2^3} + \frac{x^2}{2.\pi.\sigma_1.\sigma_2^5}\right] \cdot e^{\left[\frac{-x^2}{2.\sigma_1^2} - \frac{-y^2}{2.\sigma_2^2}\right]}$$

$$\nabla^2 G(x,y) = \frac{-1}{2.\pi.\sigma_1.\sigma_2} \cdot \left[\frac{1}{\sigma_1^2} - \frac{x^2}{\sigma_1^4} + \frac{1}{\sigma_2^2} - \frac{y^2}{\sigma_2^4}\right] \cdot e^{\left[\frac{-x^2}{2.\sigma_1^2} - \frac{-y^2}{2.\sigma_2^2}\right]}$$

Se $\sigma_1 = \sigma_2$, tem-se que:

$$\nabla^2 G(x,y) = \left[\frac{-1}{2.\pi.\sigma_1^4} + \frac{x^2}{2.\pi.\sigma_1^6} - \frac{1}{2.\pi.\sigma_1^4} + \frac{x^2}{2.\pi.\sigma_1^6} \right] . e^{\left[\frac{-x^2}{2.\sigma_1^2} - \frac{-y^2}{2.\sigma_2^2} \right]}$$

$$\nabla^2 G(x,y) = \frac{-1}{2.\pi.\sigma^4} . \left[2 - \frac{(x^2+y^2)}{\sigma^2} \right] . e^{\left[\frac{-(x^2+y^2)}{\sigma^2} \right]}$$

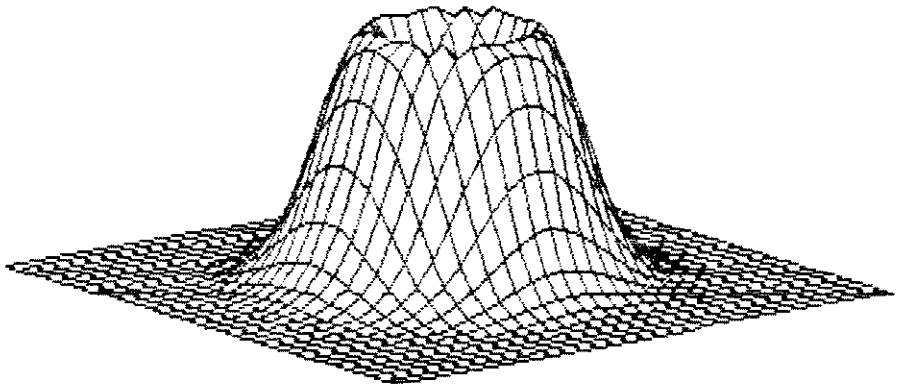
ou, em coordenadas cilíndricas:

$$\nabla^2 G(r) = \frac{-1}{2.\pi.\sigma^4} . \left[2 - \frac{r^2}{\sigma^2} \right] . e^{\left[\frac{-r^2}{2.\sigma^2} \right]}$$

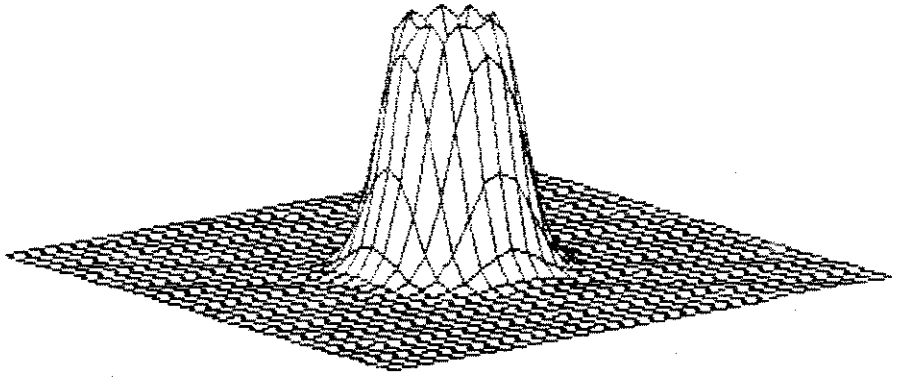
Um parâmetro importante na LoG é sua largura da região central w. Fazendo-se $\nabla^2 G = 0$, encontra-se $r = \sigma.\sqrt{2}$ e, portanto:

$$w = 2.\sigma.\sqrt{2}$$

O tamanho das máscaras é da ordem de 3w (ou 8,5.σ), para 99,7% (correspondente a três desvios padrão) da área sob LoG estar contida na máscara. A seguir são mostradas duas máscaras LoG: a primeira com desvio σ = 1,9 e a segunda com σ = 1,17.



LoG : σ = 1,9



LoG : σ = 1,17

APÊNDICE 2

A minimização do erro quadrático médio

Dado um conjunto de dados observados, pode-se querer condensá-los ajustando-os a um modelo que depende de alguns parâmetros. A abordagem geral é encontrar uma medida do "mérito", que quantifica o casamento entre os dados observados e o modelo ecolhido. Uma das medidas de mérito mais simples é o **erro quadrático médio**. Seguem-se três métodos para se minimizar o erro quadrático médio no cálculo dos parâmetros de uma reta. O primeiro deles trabalha com coordenadas cartesianas e o segundo utiliza os parâmetros da transformada de Hough. O terceiro é equivalente ao primeiro mas inclui-se a restrição de que a reta deve passar por um determinado ponto.

A)

Seja E^2 o erro quadrático médio dos n pontos observados em relação a um modelo de uma reta ($Y = a.X + b$). E^2 pode ser escrito como

$$E^2 = \sum_{i=1}^{n} (y_i - ax_i - b)^2$$

Abrindo a equação e substituindo $\sum_{i=1}^{n}$ por Σ, x_i por x e y_i por y, tem-se:

$$E^2 = \Sigma [(y^2+(ax+b)^2-2y(ax+b)] = \Sigma [y^2+a^2.x^2+2abx+b^2-2axy-2by]$$

$$E^2 = \Sigma y^2 + a^2.\Sigma x^2 + 2ab\Sigma x + \Sigma b^2 - 2a\Sigma xy - 2b\Sigma y$$

O ajuste da curva pelo erro quadrático médio requer que E^2 seja minimizado para todos os a e b. Diferenciando E^2 em relação a **a** e **b** e igualando a zero, tem-se :

$$\frac{\partial E^2}{\partial a} = 2(a\Sigma x^2 + b\Sigma x - \Sigma xy) = 0 \quad \Rightarrow a\Sigma x^2 + b\Sigma x = \Sigma xy$$

$$\frac{\partial E^2}{\partial b} = 2(a\Sigma x + nb - \Sigma y) = 0 \quad \Rightarrow a\Sigma x + nb = \Sigma y$$

Resolvendo o sistema por Kramer, tem-se:

$$a = \frac{\Sigma x^2 . \Sigma y - \Sigma x . \Sigma xy}{n\Sigma x^2 - (\Sigma x)^2} \qquad b = \frac{n\Sigma xy - \Sigma x . \Sigma y}{n\Sigma x^2 - (\Sigma x)^2}$$

B)

Seja E^2 o erro quadrático médio da transformada de Hough. E^2 pode ser escrito como

$$E^2 = \sum_{i=1}^{n} (\rho - x_i . \cos\theta - y_i . \operatorname{sen}\theta)^2$$

Abrindo a equação e substituindo $\sum_{i=1}^{n}$ por Σ, x_i por x e y_i por y, tem-se:

$$E^2 = \Sigma(\rho - x.\cos\theta - y.\operatorname{sen}\theta)^2$$
$$E^2 = \Sigma[\rho^2 - 2\rho.(x.\cos\theta + y.\operatorname{sen}\theta) + (x.\cos\theta + y.\operatorname{sen}\theta)^2]$$
$$E^2 = \Sigma[\rho^2 - 2\rho.(x.\cos\theta + y.\operatorname{sen}\theta) + x^2.\cos^2\theta + 2x.y.\cos\theta.\operatorname{sen}\theta + y^2.\operatorname{sen}^2\theta]$$

$$E^2 = n.\rho^2 - 2\rho.\cos\theta.\Sigma x - 2\rho.\operatorname{sen}\theta.\Sigma y + \cos^2\theta.\Sigma x^2 + \operatorname{sen}^2\theta.\Sigma y^2 + 2\cos\theta.\operatorname{sen}\theta.\Sigma xy$$

O ajuste da curva pelo erro quadrático médio requer que E^2 seja minimizado para todos os θ e ρ. Diferenciando E^2 em relação a ρ e igualando a zero, tem-se :

$$\frac{\partial E^2}{\partial \rho} = 2\rho.n - 2(\cos\theta.\Sigma x + \operatorname{sen}\theta.\Sigma y) = 0$$

$$\rho = (\cos\theta.\Sigma x + \operatorname{sen}\theta.\Sigma y) / n$$

Diferenciando E^2 em relação a θ e igualando a 0, tem-se :

$$\frac{\partial E^2}{\partial \rho} = 2\rho.\Sigma x.\operatorname{sen}\theta - 2\rho.\Sigma y.\cos\theta - 2\Sigma x^2.\operatorname{sen}\theta.\cos\theta + 2\Sigma y^2.\operatorname{sen}\theta.\cos\theta + 2\Sigma xy.\cos^2\theta - 2\Sigma xy.\operatorname{sen}^2\theta = 0$$

$$\frac{\partial E^2}{\partial \rho} = \frac{(\cos\theta.\Sigma x + \operatorname{sen}\theta.\Sigma y)\Sigma x.\operatorname{sen}\theta}{n} - \frac{(\cos\theta.\Sigma x + \operatorname{sen}\theta.\Sigma y)\Sigma y.\cos\theta}{n} - (\Sigma y^2 - \Sigma x^2)\operatorname{sen}\theta.\cos\theta + \Sigma xy.\cos^2\theta - \Sigma xy.\operatorname{sen}^2\theta = 0$$

$$\frac{\partial E^2}{\partial \rho} = (\Sigma x)^2 \operatorname{sen}\theta.\cos\theta/n + (\Sigma x.\Sigma y)\operatorname{sen}^2\theta/n - (\Sigma x.\Sigma y)\cos^2\theta/n - (\Sigma x)^2 \operatorname{sen}\theta.\cos\theta/n + (\Sigma y^2 - \Sigma x^2)\operatorname{sen}\theta.\cos\theta + \Sigma xy.\cos^2\theta - \Sigma xy.\operatorname{sen}^2\theta = 0$$

$$\frac{\partial E^2}{\partial \rho} = \operatorname{sen}\theta.\cos\theta.\left[\frac{(\Sigma x)^2}{n} - \frac{(\Sigma y)^2}{n} + \Sigma y^2 - \Sigma x^2\right] + \operatorname{sen}^2\theta.\left[\frac{\Sigma x.\Sigma y}{n} - \Sigma xy\right] + \cos^2\theta.\left[\Sigma xy - \frac{\Sigma x.\Sigma y}{n}\right] = 0$$

$$\frac{\partial E^2}{\partial \rho} = \operatorname{sen}\theta.\cos\theta.A - \operatorname{sen}^2\theta.B + \cos^2\theta.B = 0$$

$$\frac{\partial E^2}{\partial \rho} = \tan\theta.A - \tan^2\theta.B + B = 0$$

$$\tan\theta = \frac{A \pm \sqrt{A^2 + 4B^2}}{2B} \qquad \text{ou}$$

$$\theta = \tan^{-1}\left[\left(A \pm \sqrt{A^2 + 4B^2}\right)/2B\right]$$

onde,

$$A = \frac{(\Sigma x)^2}{n} - \frac{(\Sigma y)^2}{n} + \Sigma y^2 - \Sigma x^2 \qquad \text{e}$$

$$B = \Sigma xy - \frac{\Sigma x.\Sigma y}{n}$$

C)

Pode-se querer que o método do ítem B) tenha uma restrição do tipo: " a reta ajustada deve passar por um dado ponto C ". Desse modo, dado o conjunto de pontos (x_i, y_i) e o ponto C, com coordenadas (x_c, y_c), deve-se encontrar a reta que minimiza E^2 e que satisfaça a restrição $\rho - x_c.\cos\theta - y_c.\operatorname{sen}\theta = 0$.

Este problema pode ser resolvido da seguinte maneira: primeiramente translada-se a origem do sistema de coordeanadas dos pontos (x_i, y_i) para o ponto (x_c, y_c) criando, assim, novas coordenadas da forma $(x_i', y_i') = (x_i - x_c, y_i - y_c)$. Para este novo sistema de coordenadas, a restrição do ponto C fica $\rho = 0$. Isto faz com que $E^2 = \sum (-x_i'.\cos\theta - y_i'.\mathrm{sen}\theta)$. Através dos mesmos passos seguidos no ítem B), encontra-se:

$$\theta = \tan^{-1}[(A \pm \sqrt{A^2 + 4B^2})/2B]$$

$$A = \sum (x')^2 - (y')^2$$

$$B = \sum x'y'$$

E ρ é calculado a partir da restrição : $\rho - x_c.\cos\theta - y_c.\mathrm{sen}\theta = 0$.

APÊNDICE 3

O algoritmo de localização de objetos no espaço 3D

Se não houvesse erros de quantização e ruído, poder-se-ia afirmar que um ponto no espaço imagem, seu correspondente no espaço objeto e o centro perspectivo formam uma reta. Este fato é a base para o modelo de colinearidade.

O sistema de coordenadas da câmara (C) é definido do seguinte modo: O eixo **V** é o eixo óptico da câmara e o plano UW é paralelo ao plano de imagem localizado em V=q, onde q é a distância focal. O eixo U' e o eixo W', que são paralelos ao eixo U e ao eixo W, respectivamente, definem as coordenadas de qualquer ponto no plano imagem.

Para obter-se as coordenadas de um ponto P no espaço (3D) no plano imagem, adota-se a seguinte ordem de transformações homogêneas: primeiramente translada-se o ponto para um sistema de coordenadas centrado na lente da câmara. Em seguida é feita uma rotação por θ, seguida de uma rotação por ϕ e, finalmente, uma rotação por ψ. A matriz M abaixo representa esta sequência de tranformações.

$$M = R_{\psi} \cdot R_{\phi} \cdot R_{\theta} = \begin{bmatrix} (\cos\theta.\cos\psi + \operatorname{sen}\theta.\operatorname{sen}\phi.\operatorname{sen}\psi) & (\operatorname{sen}\theta.\operatorname{sen}\phi.\cos\psi - \cos\theta.\operatorname{sen}\psi) & (-\operatorname{sen}\theta.\cos\phi) & 0 \\ (\cos\theta.\operatorname{sen}\psi - \cos\theta.\operatorname{sen}\phi.\operatorname{sen}\psi) & (-\cos\theta.\operatorname{sen}\phi.\cos\psi - \operatorname{sen}\theta.\operatorname{sen}\psi) & (\cos\theta.\cos\phi) & 0 \\ (\cos\phi.\operatorname{sen}\psi) & (\cos\phi.\cos\psi) & \operatorname{sen}\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ou, simplificadamente :

$$M = \begin{bmatrix} A & B & C & 0 \\ D & E & F & 0 \\ G & H & I & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A transformação global, incluindo-se uma translação e a projeção perspectiva, está explicitada na eq. 1, abaixo. Esta equação também é chamada equação de colinearidade. A equação de colinearidade permite transformar as coordenadas de espaço objeto (3D) para espaço imagem (2D).

$$(u',\ w') = \left[f.\frac{A.(x-x_c) + B.(y-y_c) + C.(z-z_c)}{D.(x-x_c) + E.(y-y_c) + F.(z-z_c)} \ , \ f.\frac{G.(x-x_c) + H.(y-y_c) + I.(z-z_c)}{D.(x-x_c) + E.(y-y_c) + F.(z-z_c)} \right]$$

(equação 1)

Serão apresentados agora os 4 passos do algoritmo de calibração.

1º Passo – Calibração Do Centro Do Plano Imagem

Os três pontos de fuga principais V_x, V_y, V_z são utilizados para construir 3 linhas de fuga, cada uma delas passando por dois dos 3 pontos de fuga. As três linhas de fuga constituem um triângulo $\Delta V_x V_y V_z$.

A partir da equação de colineridade (eq. 1),podem ser derivadas as coordenadas dos três pontos de fuga principais.

Para calcular-se $V_x = (u_x', w_x')$ faz-se x tender ao infinito; ou seja:

$$(u',\ w') = \left[\lim_{x\to\infty} f.\frac{A.(x-x_c) + B.(y-y_c) + C.(z-z_c)}{D.(x-x_c) + E.(y-y_c) + F.(z-z_c)} \ , \ \lim_{x\to\infty} f.\frac{G.(x-x_c) + H.(y-y_c) + I.(z-z_c)}{D.(x-x_c) + E.(y-y_c) + F.(z-z_c)} \right]$$

$$= (\ f.A/D \ , \ f.G/D \ )$$

Que pode ser reduzido a:

$$(u_x', w_x') = \left(f.\ \frac{-\cos\theta.\cos\psi - \operatorname{sen}\theta.\operatorname{sen}\phi.\operatorname{sen}\psi}{\operatorname{sen}\theta.\cos\phi},\right.$$

$$\left. f.\ \frac{\cos\theta.\operatorname{sen}\psi - \operatorname{sen}\theta.\operatorname{sen}\phi.\cos\psi}{\operatorname{sen}\theta.\cos\phi} \right) \qquad \text{(Eq. 1.1)}$$

Da mesma maneira, $V_y = (u_y', w_y')$ pode ser encontrado fazendo-se y tender ao infinito; ou seja:

$$(u', w') = \left[\lim_{y\to\infty} f.\ \frac{A.(x-x_c) + B.(y-y_c) + C.(z-z_c)}{D.(x-x_c) + E.(y-y_c) + F.(z-z_c)}\right.,$$

$$\left. \lim_{y\to\infty} f.\ \frac{G.(x-x_c) + H.(y-y_c) + I.(z-z_c)}{D.(x-x_c) + E.(y-y_c) + F.(z-z_c)} \right]$$

$$= (f.B/E\ ,\ f.H/E)$$

Que reduz-se a:

$$(u_y', w_y') = \left(f.\ \frac{\operatorname{sen}\theta.\cos\psi - \cos\theta.\operatorname{sen}\phi.\operatorname{sen}\psi}{\cos\theta.\cos\phi},\right.$$

$$\left. f.\ \frac{-\operatorname{sen}\theta.\operatorname{sen}\psi - \cos\theta.\operatorname{sen}\phi.\cos\psi}{\cos\theta.\cos\phi} \right) \qquad \text{(Eq. 1.2)}$$

E as coordenadas de V_z são computadas similarmente a V_x e V_y:

$$(u', w') = \left[\lim_{z\to\infty} f.\ \frac{A.(x-x_c) + B.(y-y_c) + C.(z-z_c)}{D.(x-x_c) + E.(y-y_c) + F.(z-z_c)}\right.,$$

$$\left. \lim_{z\to\infty} f.\ \frac{G.(x-x_c) + H.(y-y_c) + I.(z-z_c)}{D.(x-x_c) + E.(y-y_c) + F.(z-z_c)} \right]$$

$$= (f.C/F\ ,\ f.I/F)$$

Que é reduzido a:

$$(u_z', w_z') = \left(f.\frac{\cos\phi.\operatorname{sen}\psi}{\operatorname{sen}\phi}\ ,\quad f.\frac{\cos\phi.\cos\psi}{\operatorname{sen}\phi}\right) \qquad \text{(Eq. 1.3)}$$

Ligando-se 2 pontos de fuga principais V_x e V_y obtém-se a Linha de fuga L_{xy}. A direção de L_{xy} - denotada por $\overrightarrow{L_{xy}}$ - pode ser obtida por:

$$\overrightarrow{L_{xy}} = (u_y' - u_x', w_y' - w_x')$$

$$= \left(f.\frac{\cos\psi}{\operatorname{sen}\theta.\cos\theta.\cos\phi}\ ,\ f.\frac{-\operatorname{sen}\psi}{\operatorname{sen}\theta.\cos\theta.\cos\phi}\right) \qquad \text{(Eq. 1.4)}$$

Do mesmo modo obtém-se $\overrightarrow{L_{yz}}$ e $\overrightarrow{L_{zx}}$:

$$\overrightarrow{L_{yz}} = (u_z' - u_y', w_z' - w_y')$$

$$= \left(f.\frac{\cos\theta.\operatorname{sen}\psi - \operatorname{sen}\theta.\operatorname{sen}\phi.\cos\psi}{\cos\theta.\operatorname{sen}\phi.\cos\phi}\ ,\right. \qquad \text{(Eq. 1.5)}$$

$$\left. f.\frac{\cos\theta.\cos\psi + \operatorname{sen}\theta.\operatorname{sen}\phi.\operatorname{sen}\psi}{\cos\theta.\operatorname{sen}\phi.\cos\phi}\right)$$

$$\overrightarrow{L_{zx}} = (u_x' - u_z', w_x' - w_z')$$

$$= \left(f.\frac{-\operatorname{sen}\theta.\operatorname{sen}\psi - \cos\theta.\operatorname{sen}\phi.\cos\psi}{\operatorname{sen}\theta.\operatorname{sen}\phi.\cos\phi}\ ,\right. \qquad \text{(Eq. 1.6)}$$

$$\left. f.\frac{-\operatorname{sen}\theta.\cos\psi + \cos\theta.\operatorname{sen}\phi.\operatorname{sen}\psi}{\operatorname{sen}\theta.\operatorname{sen}\phi.\cos\phi}\right)$$

A partir das equações (1.1) e (1.5) pode-se provar que o vetor que vai do centro do plano de imagem até o ponto de fuga principal V_x é perpendicular ao vetor $\overrightarrow{L_{yz}}$, pois o produto escalar $\overrightarrow{OV_x}.\overrightarrow{L_{yz}} = 0$.

$$\overrightarrow{OV}_x \times \overrightarrow{L_{yz}} =$$

$$f^2[(\frac{-\cos\theta.\cos\psi - \operatorname{sen}\theta.\operatorname{sen}\phi.\operatorname{sen}\psi}{\operatorname{sen}\theta.\cos\phi}).(\frac{\cos\theta.\operatorname{sen}\psi - \operatorname{sen}\theta.\operatorname{sen}\phi.\cos\psi}{\cos\theta.\operatorname{sen}\phi.\cos\phi})] +$$

$$f^2[(\frac{\cos\theta.\operatorname{sen}\psi - \operatorname{sen}\theta.\operatorname{sen}\phi.\cos\psi}{\operatorname{sen}\theta.\cos\phi}).(\frac{\cos\theta.\cos\psi + \operatorname{sen}\theta.\operatorname{sen}\phi.\operatorname{sen}\psi}{\cos\theta.\operatorname{sen}\phi.\cos\phi})] =$$

$$f^2[(\frac{-\cos^2\theta.\cos\psi.\operatorname{sen}\psi + \cos\theta.\operatorname{sen}\theta.\operatorname{sen}\phi.\cos^2\psi -}{\operatorname{sen}\theta.\cos\theta.\operatorname{sen}\phi.\cos\phi.\cos\phi}$$

$$\frac{\operatorname{sen}\theta.\operatorname{sen}\phi.\operatorname{sen}^2\psi.\cos\theta + \operatorname{sen}^2\theta.\operatorname{sen}^2\phi.\operatorname{sen}\psi.\cos\psi) +}{\operatorname{sen}\theta.\cos\theta.\operatorname{sen}\phi.\cos\phi.\cos\phi}$$

$$(\frac{\cos^2\theta.\cos\psi.\operatorname{sen}\psi + \cos\theta.\operatorname{sen}\theta.\operatorname{sen}\phi.\operatorname{sen}^2\psi -}{\operatorname{sen}\theta.\cos\theta.\operatorname{sen}\phi.\cos\phi.\cos\phi}$$

$$\frac{\operatorname{sen}\theta.\operatorname{sen}\phi.\cos^2\psi.\cos\theta - \operatorname{sen}^2\theta.\operatorname{sen}^2\phi.\operatorname{sen}\psi.\cos\psi)] = 0}{\operatorname{sen}\theta.\cos\theta.\operatorname{sen}\phi.\cos\phi.\cos\phi}$$

Do mesmo modo, a partir das equações 1.2 e 1.6, prova-se que $\overrightarrow{OV}_y$ e $\overrightarrow{L_{zx}}$ são perpendiculares, pois $\overrightarrow{OV}_y . \overrightarrow{L_{zx}} = 0$.

$$\overrightarrow{OV}_y \times \overrightarrow{L_{zx}} =$$

$$f^2[(\frac{\operatorname{sen}\theta.\cos\psi - \cos\theta.\operatorname{sen}\phi.\operatorname{sen}\psi}{\cos\theta.\cos\phi}).(\frac{-\operatorname{sen}\theta.\operatorname{sen}\psi - \cos\theta.\operatorname{sen}\phi.\cos\psi)]}{\operatorname{sen}\theta.\operatorname{sen}\phi.\cos\phi}$$

$$f^2[(\frac{-\operatorname{sen}\theta.\operatorname{sen}\psi - \cos\theta.\operatorname{sen}\phi.\cos\psi}{\cos\theta.\cos\phi}).(\frac{-\operatorname{sen}\theta.\cos\psi + \cos\theta.\operatorname{sen}\phi.\operatorname{sen}\psi)] =}{\operatorname{sen}\theta.\operatorname{sen}\phi.\cos\phi}$$

$$f^2[(\frac{-\operatorname{sen}^2\theta.\cos\psi.\operatorname{sen}\psi - \operatorname{sen}\theta.\cos\theta.\operatorname{sen}\phi.\cos^2\psi +}{\operatorname{sen}\theta.\cos\theta.\operatorname{sen}\phi.\cos\phi.\cos\phi}$$

$$\frac{\operatorname{sen}\theta.\operatorname{sen}\phi.\operatorname{sen}^2\psi.\cos\theta + \cos^2\theta.\operatorname{sen}^2\phi.\operatorname{sen}\psi.\cos\psi) +}{\operatorname{sen}\theta.\cos\theta.\operatorname{sen}\phi.\cos\phi.\cos\phi}$$

$$(\frac{\operatorname{sen}^2\theta.\cos\psi.\operatorname{sen}\psi - \cos\theta.\operatorname{sen}\theta.\operatorname{sen}\phi.\operatorname{sen}^2\psi +}{\operatorname{sen}\theta.\cos\theta.\operatorname{sen}\phi.\cos\phi.\cos\phi}$$

$$\frac{\operatorname{sen}\theta.\operatorname{sen}\phi.\cos^2\psi.\cos\theta - \cos^2\theta.\operatorname{sen}^2\phi.\operatorname{sen}\psi.\cos\psi)] = 0}{\operatorname{sen}\theta.\cos\theta.\operatorname{sen}\phi.\cos\phi.\cos\phi}$$

E, finalmente, a partir das equações 1.3 e 1.4 , prova-se que $\overrightarrow{OV}_z$ e $\overrightarrow{L_{xy}}$ são perpendiculares pois $\overrightarrow{OV}_z . \overrightarrow{L_{xy}} = 0$.

$\overrightarrow{OV_z} \cdot \overrightarrow{L_{xy}} =$

$$f^2 \left[\left(\frac{\cos\phi . \operatorname{sen}\psi}{\operatorname{sen}\phi} \cdot \frac{\cos\psi}{\operatorname{sen}\theta . \cos\theta . \cos\phi} \right) + \left(\frac{\cos\phi . \cos\psi}{\operatorname{sen}\phi} \cdot \frac{-\operatorname{sen}\psi}{\operatorname{sen}\theta . \cos\theta . \cos\phi} \right) \right] =$$

$$f^2 \left[\frac{\cos\phi . \operatorname{sen}\psi . \cos\psi}{\operatorname{sen}\theta . \cos\theta . \operatorname{sen}\phi . \cos\phi} - \frac{\cos\phi . \cos\psi . \operatorname{sen}\psi}{\operatorname{sen}\theta . \cos\theta . \cos\phi . \operatorname{sen}\phi} \right] = 0$$

Portanto, pode-se concluir que o ortocentro do triângulo $\Delta V_x V_y V_z$ é exatamente o centro do plano de imagem.

As coordenadas do centro do plano de imagem (u_o' , w_o') podem ser obtidas através de quaisquer duas relações, por exemplo: $\overrightarrow{OV_x} . \overrightarrow{L_{yz}} = 0$ e $\overrightarrow{OV_y} . \overrightarrow{L_{zx}} = 0$. Ou seja, expandindo e resolvendo este sistema de equações, pode-se obter a coordenada do centro do plano imagem.

$\overrightarrow{OV_x} . \overrightarrow{L_{yz}} = 0$

$$(u_x - u_o , w_x - w_o) . (u_z - u_y , w_z - w_y) = 0$$
$$(u_x - u_o) . (u_z - u_y) + (w_x - w_o) . (w_z - w_y) = 0$$
$$(u_x . u_z - u_x . u_y) - u_o . (u_z - u_y) + (w_x . w_z - w_x . w_y) - w_o . (w_z - w_y) = 0$$

$$u_o . (u_z - u_y) + w_o . (w_z - w_y) = (u_x . u_z - u_x . u_y) + (w_x . w_z - w_x . w_y)$$

ou

$$u_o . a + w_o . b = c$$

e

$\overrightarrow{OV_y} . \overrightarrow{L_{zx}} = 0$

$$(u_y - u_o , w_y - w_o) . (u_x - u_z , w_x - w_z) = 0$$
$$(u_y - u_o) . (u_x - u_z) + (w_y - w_o) . (w_x - w_z) = 0$$
$$(u_y . u_x - u_y . u_z) - u_o . (u_x - u_z) + (w_y . w_x - w_y . w_z) - w_o . (w_x - w_z) = 0$$

$$u_o . (u_x - u_z) + w_o . (w_x - w_z) = (u_y . u_x - u_y . u_x) + (w_y . w_x - w_y . w_z)$$

ou

$$u_o . d + w_o . e = f$$

Resolvendo o sistema por Kramer:

$$u_o = \frac{\begin{vmatrix} c & b \\ f & e \end{vmatrix}}{\begin{vmatrix} a & b \\ c & d \end{vmatrix}} \qquad w_o = \frac{\begin{vmatrix} a & c \\ d & f \end{vmatrix}}{\begin{vmatrix} a & b \\ c & d \end{vmatrix}}$$

2º Passo – Calibração dos Ângulos de Orientação da Câmara

Nesta seção derivam-se os parâmetros de orientação da câmara através das inclinações das 3 linhas de fuga L_{xy} , L_{yz} e L_{zx} . Supõe-se que as inclinações de L_{xy} , L_{yz} e L_{zx} são m_1 , m_2 , m_3, respectivamente.

A partir da equação (1.4) computa-se m_1 :

$$m_1 = \frac{f.\dfrac{-\text{sen}\psi}{\text{sen}\theta.\cos\theta.\cos\phi}}{f.\dfrac{\cos\psi}{\text{sen}\theta.\cos\theta.\cos\phi}} \qquad \text{(Eq. 1.7)}$$

$$= -\text{tg}\psi$$

Portanto, o ângulo ψ pode ser computado por: $\psi = \text{atan}(-m_1)$

Similarmente, obtem-se:

$$m_2 = \frac{\cos\theta.\cos\psi + \text{sen}\theta.\text{sen}\phi.\text{sen}\psi}{\cos\theta.\text{sen}\psi - \text{sen}\theta.\text{sen}\phi.\cos\psi} \qquad \text{(Eq. 1.8)}$$

e

$$m_3 = \frac{-\text{sen}\theta.\cos\psi + \cos\theta.\text{sen}\phi.\text{sen}\psi}{-\text{sen}\theta.\text{sen}\psi - \cos\theta.\text{sen}\phi.\text{com}\psi} \qquad \text{(Eq. 1.9)}$$

As duas equações anteriores podem ser simplificadas para

$$(m_2.\cos\psi + \text{sen}\psi).\text{sen}\theta.\text{sen}\phi = (m_2.\text{sen}\psi - \cos\psi).\cos\theta \qquad \text{(Eq. 1.10)}$$

e

$$(m_3.\cos\psi + \text{sen}\psi).\cos\theta.\text{sen}\phi = (-m_3.\text{sen}\psi + \cos\psi).\text{sen}\theta \qquad \text{(Eq. 1.11)}$$

multiplicando-se as equações (1.10) e (1.11) tem-se que:

$$\text{sen}^2\phi = \frac{-(m_1.m_2+1).(m_1.m_3+1)}{(m_1-m_2).(m_1-m_3)} \qquad \text{(Eq. 1.12)}$$

que pode ser utilizado para calcular o ângulo ϕ. Dividindo-se as equações (1.10) e (1.11) tem-se que

$$\text{tg}^2\theta = \frac{(m_1.m_2+1).(m_1-m_3)}{(m_1.m_3+1).(m_1-m_2)} \qquad \text{(Eq.1.13)}$$

Esta última pode ser usada para computar o ângulo θ. Os sinais de θ e ϕ podem ser determinados após a computação da distância focal f.

3º Passo - Calibração da Distância Focal da Câmara

Agora será mostrado que a área do triângulo $\Delta V_x V_y V_z$ pode ser usada para calcular a distância focal da câmara.

A área do triângulo $\Delta V_x V_y V_z$ é dada por:

Área do triângulo = A

$$A = 1/2 \cdot \begin{vmatrix} u_x' & w_x' & 1 \\ u_y' & w_y' & 1 \\ u_z' & w_z' & 1 \end{vmatrix}$$

$$A = 1/2.(u_x'.w_y' + u_z'.w_x' + u_y'.w_z' - u_z'.w_y' - u_x' w_z' - u_y'.w_x')$$

onde os termos u_i' e w_i' são dados pelas equações 1.1, 1.2 e 1.3. Calculando-se o determinante, tem-se:

$$(u_x'.w_y' / f^2) = \left(\frac{-\cos\theta.\cos\psi - \text{sen}\theta.\text{sen}\phi.\text{sen}\psi}{\text{sen}\theta.\cos\phi}\right).$$

$$\left(\frac{-\text{sen}\theta.\text{sen}\psi - \cos\theta.\text{sen}\phi.\cos\psi}{\cos\theta.\cos\phi}\right)$$

$$= \frac{\cos\theta.\cos\psi.\text{sen}\theta.\text{sen}\psi}{\text{sen}\theta.\cos\phi.\cos\theta.\cos\phi} + \frac{\cos\theta.\cos\psi.\cos\theta.\text{sen}\phi.\cos\psi}{\text{sen}\theta.\cos\phi.\cos\theta.\cos\phi} +$$

$$+ \frac{\text{sen}\theta.\text{sen}\phi.\text{sen}\psi.\text{sen}\theta.\text{sen}\psi}{\text{sen}\theta.\cos\phi.\cos\theta.\cos\phi} + \frac{\text{sen}\theta.\text{sen}\phi.\text{sen}\psi.\cos\theta.\text{sen}\phi.\cos\psi}{\text{sen}\theta.\cos\phi.\cos\theta.\cos\phi}$$

$$= \frac{\cos\psi.\operatorname{sen}\psi}{\cos\phi.\cos\phi} + \frac{\cos\theta.\cos\psi.\operatorname{sen}\phi.\cos\psi}{\cos\phi.\operatorname{sen}\theta.\cos\phi} +$$

$$\frac{\operatorname{sen}\phi.\operatorname{sen}\psi.\operatorname{sen}\theta.\operatorname{sen}\psi}{\cos\phi.\cos\theta.\cos\psi} + \frac{\cos\psi.\operatorname{sen}\phi.\operatorname{sen}\psi.\operatorname{sen}\phi}{\cos\phi.\cos\phi}$$

$$(u_z'.w_x' \;/\; f^2) = \frac{(\cos\phi.\operatorname{sen}\psi)}{\operatorname{sen}\phi} \cdot \frac{(\cos\theta.\operatorname{sen}\psi - \operatorname{sen}\theta.\operatorname{sen}\phi.\cos\psi)}{\operatorname{sen}\theta.\cos\phi}$$

$$= \frac{\cos\theta.\operatorname{sen}\psi.\cos\phi.\operatorname{sen}\psi}{\operatorname{sen}\theta.\cos\phi.\operatorname{sen}\phi} - \frac{\cos\phi.\operatorname{sen}\psi.\operatorname{sen}\theta.\operatorname{sen}\phi.\cos\psi}{\operatorname{sen}\phi.\operatorname{sen}\theta.\cos\phi}$$

$$= \frac{\cos\theta.\operatorname{sen}\psi.\operatorname{sen}\psi}{\operatorname{sen}\theta.\operatorname{sen}\phi} - \operatorname{sen}\psi.\cos\psi$$

$$(u_y'.w_z' \;/\; f^2) = \frac{(\operatorname{sen}\theta.\cos\psi - \cos\theta.\operatorname{sen}\phi.\operatorname{sen}\psi)}{\cos\theta.\cos\phi} \cdot \frac{(\cos\phi.\cos\psi)}{\operatorname{sen}\phi}$$

$$= \frac{\operatorname{sen}\theta.\cos\psi.\cos\phi.\cos\psi}{\cos\theta.\cos\phi.\operatorname{sen}\phi} - \frac{\cos\theta.\operatorname{sen}\phi.\operatorname{sen}\psi.\cos\phi.\cos\psi}{\cos\theta.\cos\phi.\operatorname{sen}\phi}$$

$$= \frac{\operatorname{sen}\theta.\cos\psi.\cos\psi}{\cos\theta.\operatorname{sen}\phi} - \operatorname{sen}\psi.\cos\psi$$

$$(u_z'.w_y' \;/\; f^2) = \frac{(\cos\phi.\operatorname{sen}\psi)}{\operatorname{sen}\phi} \cdot \frac{(-\operatorname{sen}\theta.\operatorname{sen}\psi - \cos\theta.\operatorname{sen}\phi.\cos\psi)}{\cos\theta.\cos\phi}$$

$$= \frac{-\cos\phi.\operatorname{sen}\psi.\operatorname{sen}\theta.\operatorname{sen}\psi}{\operatorname{sen}\phi.\cos\theta.\cos\phi} - \frac{\cos\phi.\operatorname{sen}\psi.\cos\theta.\operatorname{sen}\phi.\cos\psi}{\operatorname{sen}\phi.\cos\theta.\cos\phi}$$

$$= \frac{-\operatorname{sen}\psi.\operatorname{sen}\theta.\operatorname{sen}\psi}{\operatorname{sen}\phi.\cos\theta} - \operatorname{sen}\psi.\cos\psi$$

$$(u_x'.w_z' \;/\; f^2) = \frac{(-\cos\theta.\cos\psi - \operatorname{sen}\theta.\operatorname{sen}\phi.\operatorname{sen}\psi)}{\operatorname{sen}\theta.\cos\phi} \cdot \frac{(\cos\phi.\cos\psi)}{\operatorname{sen}\phi}$$

$$= \frac{-\cos\theta.\cos\psi.\cos\phi.\cos\psi}{\operatorname{sen}\theta.\cos\phi.\operatorname{sen}\phi} - \frac{\operatorname{sen}\theta.\operatorname{sen}\phi.\operatorname{sen}\psi.\cos\phi.\cos\psi}{\operatorname{sen}\theta.\cos\phi.\operatorname{sen}\phi}$$

$$= \frac{-\cos\theta.\cos\psi.\cos\psi}{\operatorname{sen}\theta.\operatorname{sen}\phi} - \operatorname{sen}\psi.\cos\psi$$

$$(u_y'.w_x' \;/\; f^2) = \frac{(\operatorname{sen}\theta.\cos\psi - \cos\theta.\operatorname{sen}\phi.\operatorname{sen}\psi)}{\cos\theta.\cos\phi} \cdot \frac{(\cos\theta.\operatorname{sen}\psi - \operatorname{sen}\theta.\operatorname{sen}\phi.\cos\psi)}{\operatorname{sen}\theta.\cos\phi}$$

$$= \frac{sen\theta.cos\psi.cos\theta.sen\psi}{cos\theta.cos\phi.sen\theta.cos\phi} - \frac{sen\theta.cos\psi.sen\theta.sen\phi.cos\psi}{cos\theta.cos\phi.sen\theta.cos\phi} -$$

$$\frac{cos\theta.sen\phi.sen\psi.cos\theta.sen\psi}{cos\theta.cos\phi.sen\theta.cos\phi} + \frac{cos\theta.sen\phi.sen\psi.sen\theta.sen\phi.cos\psi}{cos\theta.cos\phi.sen\theta.cos\phi}$$

$$= \frac{cos\psi.sen\psi}{cos\phi.cos\phi} - \frac{sen\theta.cos\psi.sen\phi.cos\psi}{cos\theta.cos\phi.cos\phi} -$$

$$\frac{sen\phi.sen\psi.cos\theta.sen\psi}{sen\theta.cos\phi.cos\phi} + \frac{sen\phi.sen\psi.sen\phi.cos\psi}{cos\phi.cos\phi}$$

Somando-se os termos dois a dois, tem-se:

$$(u'_x.w'_y - u'_y.w'_x) / f^2 = \frac{cos\theta.cos\psi.sen\phi.cos\psi}{cos\phi.sen\theta.cos\phi} + \frac{sen\phi.sen\psi.sen\theta.sen\psi}{cos\phi.cos\theta.cos\phi} +$$

$$\frac{sen\theta.cos\psi.sen\phi.cos\psi}{cos\theta.cos\phi.cos\phi} + \frac{sen\phi.sen\psi.cos\theta.sen\psi}{sen\theta.cos\phi.cos\phi}$$

$$= \frac{cos\theta.sen\phi\ (cos\psi.cos\psi + sen\psi.sen\psi)}{sen\theta.cos\phi.cos\phi} +$$

$$\frac{sen\phi.sen\theta.(cos\psi.cos\psi + sen\psi.sen\psi)}{cos\theta.cos\phi.cos\phi}$$

$$= \frac{cos\theta.sen\phi}{sen\theta.cos\phi.cos\phi} + \frac{sen\phi.sen\theta}{cos\theta.cos\phi.cos\phi}$$

$$(u'_z.w'_x - u'_x.w'_z) / f^2 = \frac{cos\theta.sen\psi.sen\psi}{sen\theta.sen\phi} + \frac{cos\theta.cos\psi.cos\psi}{sen\theta.sen\phi}$$

$$= \frac{cos\theta.(sen\psi.sen\psi + cos\psi.cos\psi)}{sen\theta.sen\phi}$$

$$= \frac{cos\theta}{sen\theta.sen\phi}$$

$$(u'_y.w'_z - u'_z.w'_y) / f^2 = \frac{sen\theta.cos\psi.cos\psi}{cos\theta.sen\phi} + \frac{sen\psi.sen\theta.sen\psi}{cos\theta.sen}$$

$$= \frac{sen\theta.(cos\psi.cos\psi + sen\psi.sen\psi)}{cos\theta.sen\phi}$$

$$= \frac{sen\theta}{cos\theta.sen\phi}$$

Juntando-se todos os termos, tem-se:

$$2A/f^2 = \frac{\cos\theta.\operatorname{sen}\phi}{\operatorname{sen}\theta.\cos\phi.\cos\phi} + \frac{\operatorname{sen}\phi.\operatorname{sen}\theta}{\cos\theta.\cos\phi.\cos\phi} + \frac{\cos\theta}{\operatorname{sen}\theta.\operatorname{sen}\phi} + \frac{\operatorname{sen}\theta}{\cos\theta.\operatorname{sen}\phi}$$

$$= \frac{1}{\operatorname{sen}\phi} \cdot \left(\frac{\operatorname{sen}\theta}{\cos\theta} + \frac{\cos\theta}{\operatorname{sen}\theta}\right) + \frac{\operatorname{sen}\phi}{\cos\phi.\cos\phi} \cdot \left(\frac{\operatorname{sen}\theta}{\cos\theta} + \frac{\cos\theta}{\operatorname{sen}\theta}\right)$$

$$= \frac{1}{\operatorname{sen}\phi} \cdot \left(\frac{1}{\cos\theta.\operatorname{sen}\theta}\right) + \frac{\operatorname{sen}\phi}{\cos\phi.\cos\phi} \cdot \left(\frac{1}{\cos\theta.\operatorname{sen}\theta}\right)$$

$$= \left(\frac{1}{\cos\theta.\operatorname{sen}\theta}\right).\left(\frac{1}{\operatorname{sen}\phi} + \frac{\operatorname{sen}\phi}{\cos\phi.\cos\phi}\right)$$

$$= \left(\frac{1}{\cos\theta.\operatorname{sen}\theta}\right).\left(\frac{\cos\phi.\cos\phi + \operatorname{sen}\phi.\operatorname{sen}\phi}{\operatorname{sen}\phi.\cos\phi.\cos\phi}\right)$$

$$= \frac{1}{\cos\theta.\operatorname{sen}\theta.\operatorname{sen}\phi.\cos\phi.\cos\phi}$$

Portanto,

$$f = \sqrt{2.A.\left|\cos\theta.\operatorname{sen}\theta.\operatorname{sen}\phi.\cos^2\phi\right|} \qquad \text{(Eq. 1.14)}$$

Uma vez computado f, usando-se a equação (1.3) para se checar o sinal de ϕ e posteriormente determinar o sinal do ângulo θ, a partir de (1.1) ou (1.2).

4º Passo - Localização da Câmara no Espaço

Sejam $P_1=(x_1, y_1, z_1)$ e $P_2=(x_2, y_2, z_2)$ dois pontos com posições conhecidas no espaço e sejam $P_1' = (u_1', w_1')$ e $P_2' = (u_2', w_2')$ suas projeções no plano imagem, respectivamente. Pela equação de colinearidade (eq. 1), tem-se:

(Eq. 1.15)

$$(u_1', w_1') = \left[f.\frac{A.(x_1-x_c) + B.(y_1-y_c) + C.(z_1-z_c)}{D.(x_1-x_c) + E.(y_1-y_c) + F.(z_1-z_c)} \;,\; f.\frac{G.(x_1-x_c) + H.(y_1-y_c) + I.(z_1-z_c)}{D.(x_1-x_c) + E.(y_1-y_c) + F.(z_1-z_c)} \right]$$

(Eq. 1.16)

e

(Eq. 1.17)

$$(u_2', w_2') = \left[f.\frac{A.(x_2-x_c) + B.(y_2-y_c) + C.(z_2-z_c)}{D.(x_2-x_c) + E.(y_2-y_c) + F.(z_2-z_c)} \; , \; f.\frac{G.(x_2-x_c) + H.(y_2-y_c) + I.(z_2-z_c)}{D.(x_2-x_c) + E.(y_2-y_c) + F.(z_2-z_c)} \right]$$

(Eq. 1.18)

Quaisquer três dessas 4 equações podem ser escolhidas para derivar-se soluções analíticas para os parâmetros de posicionamento da câmara x_c , y_c , z_c. Aqui, escolheu-se as equações 1.15, 1.16 e 1.17. As três equações podem ser transformadas em

$$a_1.x_c + b_1.y_c + c_1.z_c = d_1$$
$$a_2.x_c + b_2.y_c + c_2.z_c = d_2$$
$$a_3.x_c + b_3.y_c + c_3.z_c = d_3$$

Onde

$$a_1 = u_1'.D - f.A$$
$$b_1 = u_1'.D - f.B$$
$$c_1 = u_1'.D - f.C$$
$$d = a.x + b.y + c.z$$

$$a_2 = w_1'.D - f.G$$
$$b_2 = w_1'.D - f.H$$
$$c_2 = w_1'.D - f.I$$
$$d = a.x + b.y + c.z$$

$$a_3 = u'_2.D - f.A$$
$$b_3 = u_2'.D - f.B$$
$$c_3 = u_2'.D - f.C$$
$$d_3 = a_2.x_c + b_2.y_c + c_2.z_c$$

Posteriormente, resolvendo-se este sistema de equações lineares, consegue-se soluções únicas para x_c , y_c , z_c como se segue (método de Kramer):

$$x_c = \frac{\begin{vmatrix} d_1 & b_1 & c_1 \\ d_2 & b_2 & c_2 \\ d_3 & b_3 & c_3 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}} \qquad y_c = \frac{\begin{vmatrix} d_1 & d_1 & c_1 \\ d_2 & d_2 & c_2 \\ d_3 & d_3 & c_3 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}}$$

$$z_c = \frac{\begin{vmatrix} d_1 & b_1 & d_1 \\ d_2 & b_2 & d_2 \\ d_3 & b_3 & d_3 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}}$$

APÊNDICE 4

Tabelas Completas dos Tempos de Execução

Apresenta-se o conjunto de tabelas de tempos de execução das soluções apresentadas nos capítulos 4 e 5 e que foram utilizadas na confecção dos gráficos do capítulo 5. É importante ressaltar que os tempos de execução **não** consideram o tempo de carga da imagem de entrada no processador mestre.

Note-se também que nem todas as topologias podem ser construídas com qualquer nº de processadores. Por exemplo, a topologia hipercúbica possui 2^n+1 (2^n escravos mais um mestre) processadores , onde n é a dimensão do hipercubo, conforme visto na seção 3.1.1. Não se pode construir um hipercubo (não-degenerado) com, por exemplo, 7 processadores. Fato semelhante acontece com a malha com wrap-around.

K	Tempo (s)
1	1,677
2	1,703
4	1,756
8	1,858
10	1,911
16	2,064
20	2,179
26	2,327
30	2,433

Tabela A4.1. Tempo de execução – T(1) – da Transformada Direcional de Hough em função do número (k) de ângulos calculados.

N	ANEL	MALHA	HIPERCUBO
4	---	---	---
5	1,789 s	1,114 s	1,121 s
6	1,470 s	---	---
7	1,249 s	1,019 s	---
8	1,102 s	---	---
9	1,003 s	0,912 s	0,983 s

Tabela A4.2. Tempo de execução da solução (a) para a transformada clássica de Hough paralela.

N	ANEL	MALHA	HIPERCUBO
4	---	---	---
5	0,636 s	0,582 s	0,597 s
6	0,610 s	---	---
7	0,606 s	0,533 s	---
8	0,604 s	---	---
9	0,602 s	0,515 s	0,547 s

Tabela A4.3. Tempo de execução da solução (c) para a transformada clássica de Hough paralela.

N	ANEL	MALHA	HIPERCUBO
4	---	---	---
5	0,538 s	0,511 s	0,520 s
6	0,503 s	---	---
7	0,472 s	0,413 s	---
8	0,442 s	---	---
9	0,421 s	0,374 s	0,412 s

Tabela A4.4. Tempo de execução da solução (d) para a transformada paralela direcional de Hough, com k = 1.

N	ANEL	MALHA	HIPERCUBO
4	---	---	---
5	0,602 s	0,531 s	0,538 s
6	0,567 s	---	---
7	0,544 s	0,486 s	---
8	0,538 s	---	---
9	0,536 s	0,463 s	0,487 s

Tabela A4.5. Tempo de execução da solução (d) para a transformada paralela direcional de Hough, com k =20.

N	ÁRVORE	MALHA	HIPERCUBO
4	---	---	---
5	0,448 s	0,486 s	0,551 s
6	---	---	---
7	0,362 s	0,356 s	---
8	---	---	---
9	0,327 s	0,314 s	0,403 s

Tabela A4.6. Tempo de execução da solução (e) para a transformada paralela direcional de Hough com k = 1.

N	ÁRVORE	MALHA	HIPERCUBO
4	---	---	---
5	0,525 s	0,530 s	0,622 s
6	---	---	---
7	0,482 s	0,473 s	---
8	---	---	---
9	0,471 s	0,453 s	0,583 s

Tabela A4.7. Tempo de execução da solução (e) para a transformada paralela direcional de Hough com k = 20.

Referências Bibliográficas

01 - AMORIM, C.L, BARBOSA, V.C., FERNANDES, E.S.T. **Uma Introdução à Computação Paralela e Distribuída**. (Versão preliminar preparada para a VI Escola de Computação, Campinas, Jul. 1988), Campinas, UNICAMP, IMECC, 1988.

02 - ARAMBEPOLA, B., PATEL, V.B., CHEUNG, G. Casacadable One/Two-Dimensional Digital Convolver. **IEEE Journal of Solid-State Circuits**, v. 23, n. 2, p. 351-357, 1988.

03 - ARBIB, M.A., HANSON, A.R. **Vision, Brain, and Cooperative Computation**. Cambridge (MA): MIT Press, 1987.

04 - BACKUS, J. Can Programming be Liberated from the von Neumann Style ? (ACM Turing Award-1977). **Communications of the ACM**, v. 8, p. 613-641, 1978.

05 - BALLARD, D.H. Generalizing the Hough Transform to Detect Arbitrary Shapes. **Pattern Recognition**, v. 13, n. 2, p. 111-122, 1981.

06 - BALLARD, D.H., BROWN, C.M. **Computer Vision**. Englewood Cliffs (NJ): Prentice Hall, 1982.

07 - BARROS, F.C. Multiprocessamento Aplicado ao Tratamento de Imagens. **Trabalho de fim de curso**, Faculdade de Engenharia Elétrica/UNICAMP, 1989.

08 - BARROS, F.C. Arquiteturas Paralelas para Visão Computacional. **Relatório técnico Nº 13/1990**, Depto. de Computação e Automação Industrial/Faculdade de Engenharia Elétrica/UNICAMP, 1990.

09 - BELLON, O.R. Visão Computacional: Um Sistema para Localização de Objetos Poliédricos no Espaço 3D. **Tese de Mestrado**. Faculdade de Engenharia Elétrica/UNICAMP, 1990.

10 - BLELLOCH, G.E. **Vector Models for Data-Parallel Computing**. Cambridge (MA): MIT Press, 1990.

11 - BRADY, M. (Coord.). **Robotics Science**. Cambridge: MIT Press, 1989.

12 - CASTANHO, J.E.C. Desenvolvimento de uma Estação de Trabalho para Visão Computacional. **Tese de Mestrado**. Faculdade de Engenharia Elétrica/UNICAMP, 1990.

13 - CANNY, J. A Computational Approach to Edge Detection. **IEEE Trans. on PAMI**, v. 8, n. 6, p. 679-697, 1986.

14 - CHANDRAN, S., DAVIS, L.S. Parallel Vision Algorithms: an Approach. **Proceedings 3rd SIAM Conference on Parallel Proc. for Scientific Computing**. p. 235-249, 1987.

15 - CHEN, S.Y., TSAI, W.H. A Systematic Approach to Analytic Determination of Camera Parameters by Line Features. **Pattern Recognition**, v. 23, n. 8, p. 859-877, 1990.

16 - CHEN, S.Y, TSAI, W.H. Determination of Robot Location by Common Object Shapes. **IEEE Trans. on RA**, v. 7, n. 1, p. 149-156, 1991.

17 - CHEN, W., JIANG, B.C. 3D Camera Calibration Using Vanishing Point Concept. **Pattern Recognition**, v. 24, p. 57-67, 1991.

18 - CHEN, Z., TSENG, D.C., LIN, J.Y. A Simple Vision Algorithm for 3D Position Determination Using a Single Calibration Object. **Pattern Recognition**, v. 22, n. 2, p. 173-187, 1989.

19 - CYPHER, R.E, SANZ, J.L.C., SNYDER, L. The Hough Transform has O(n) complexity on SIMD NxN Mesh Array Architectures. **Proceedings of the IEEE Workshop on PAMI**, p. 115-121, 1987.

20 - DHOME, M., KASVAND, T. Polyedra Recognition by Hypothesis Accumulation. **IEEE Trans. on PAMI**, v. 9, n. 3, p. 429-439, 1987.

21 - DHOME, M., RICHETIN, M., LAPRESTÉ. J.T., et al. Determinatioan of the Attitude of 3D Objects from a Single Perspective View. **IEEE Trans. on PAMI**, v. 11, n. 12, p. 1265-1278, 1989.

22 - DUDANI, S.A., LUK, A.L. Locating Straight-Line Edge Segments on Outdoor Scenes. **Pattern Recognition**, v. 10, p. 145-157, 1978.

23 - ECHIGO, T. A Camera Calibration Technique using Three sets of Parallel Lines. **Machine Vision and Aplications**, v. 3, n. 3, p. 159-167, 1990.

24 - ELGOT, C.C., ROBINSON, A. Random Access Stored Program Access Machines. **Journal of the ACM**, v. 11, n. 4, p. 365-399, 1964.

25 - FISHER, A.L., HINGHNAM, P.T. Computing the Hough Transform on a Scan-Line Array Processor. **IEEE Trans. on PAMI**, v. 11, n. 3, p. 262-265, 1989.

26 - FISCHLER, M.A., BOLLES, R.C. Random Sample Consensus: A Paradigm for Model Fitting with Application to Image Analysis and Automated Cartography. **Coomunications of the ACM**, v. 24, n. 6, p. 381-395, 1981.

27 - FLYNN, M.J. Very High Speed Computing Systems. **Proceedings of the IEEE**, v. 54, p. 1901-1909, 1966.

28 - FONSECA, L.E.N. Desenvolvimento de um Sistema para Colorização de Imagens Digitais. **Tese de Mestrado**. Faculdade de Engenharia Elétrica/UNICAMP, 1990.

29 - FOX, G.C. et al. **Solving Problems on Concurrent Computers**. Englewood Cliffs (NJ): Prentice Hall, 1988.

30 - GONZALES, R.C., WINTZ, P. **Digital Image Processing**. Reading (MA): Addison-Wesley, 1982.

31 - GUSTAFSON, J.L. Reevaluating Amdahl's Law. **Communications of the ACM**, v. 31, n. 5, p. 532-533, 1988.

32 - GUERRA, C., HAMBRUSCH, S. Parallel Algorithms for Line Detection on a Mesh. **Journal of Parallel and Distrib. Comput.** v. 6, p. 1-19, 1989.

33 - HANAHARA, K., MARUYAMA, T., UCHIYAMA, T. A Real-Time Processor for the Hough Transform. **IEEE Trans. on PAMI**, v. 10, n. 1, p. 121-125, 1988.

34 - HARALICK, R.M. Using Perspective Transformations in Scene Analysis. **Computer Graphics and Image Processing**, v. 13, p. 191-221, 1980.

35 - HILLIS, W.D. **The Connection Machine**. Cambridge (MA): MIT Press, 1985.

36 - HILLIS, W.D., Steele, G.L. Data Parallel Algorithms. **Communicaions of the ACM**, v. 29, n. 12, p. 1170-1183, 1986.

37 - HOARE, C.A.R. **Processus Sequentiels Communicants**. Tradução de Alain Kermarrec. Paris: Masson, 1987.

38 - HORN, B.K.P. **Robot Vision**. Cambridge (MA): MIT Press, 1986.

39 - HWANG, K., BRIGGS, F. **Computer Architecture and Parallel Procesing**. New York (NY): McGraw Hill, 1984.

40 - IBRAHIM, H.A.H., KENDER, J.R., SHAW, D.S. The Analysis and Performance of two Middle-Level Vision Tasks on a Fine-Grained SIMD Tree Machine. **Proceedings of the IEEE Conf. on PAMI**, p. 248-256, 1985.

41 - ILLINGWORTH, J., KITTLER, J. A Survey of the Hough Transform. **Computer Vision. Graph. and Image Proc.** v. 44, p. 87-116, 1988.

42 - INMOS Limited (Inglaterra). **IMS B008 User Guide and Reference Manual**, 1988.

43 - INMOS Limited (Inglaterra). **The Transputer Development Databook**, 1989.

44 - KANOPOULOS, N., VASANTHAVADA, N, BAKER, R. Design of an Image Edge Detection Filter Using the Sobel Operator. **IEEE Journal of Solid-State Circuits**, v. 23, n. 2, p. 358-367, 1988.

45 - KUNG, H.T. Why Systolic Architectures ? **IEEE COMPUTER**, v. 13, n. 1, p. 37-46, 1982.

46 - LEE, R., LU, P.C., TSAI, W.H. Robot Location Using Single Views of Rectangular Shapes. **Photogrammetric Engineering and Remote Sensing** v. 56, n. 2, p. 231-238, 1990.

47 - LI, H.F., PAO, D., JAYAKUMAR, R. Improvements and Systolic Implementation of The Hough Transformation for Line Detection. **Pattern Recognition**, v. 22, n. 6, p. 697-706, 1989.

48 - LINDSAY, D.C. Towards a Shared Memory Hypercube. **Internal Report** Carnegie Mellon University. CMU-CS-88-190. nov./1988.

49 - LITTLE, J.J., BLELLOCH, G.E., CASS, T.A. Algorithmic Techniques for Computer Vision on a Fine-Grained Parallel Machine. **IEEE Trans. on PAMI**, v. 11, n. 3, p. 244-257, 1989.

50 - LIU, Y., HUANG, T.S., FAUGERAS, O.D. Determination of Camera Location from 2D to 3D Line and Point Correspondences. **IEEE Trans. on PAMI**, v. 12, n. 1, p. 28-37, 1990.

51 - LOWE, D.G. Three-Dimensional Object Recognition from Single Two-Dimensional Images. **Artificial Intelligence**, v. 31, p. 355-395, 1987.

52 - LÜ, H.E., WANG, P.S.P. An Improved Fast Parallel Thinning Algorithm for Digital Patterns. **Proceedings of the IEEE Conf. on PAMI**, p. 364-367, 1985.

53 - MAHOWALD, M.A., MEAD, C. The Silicon Retina. **Scientific American**, v. 264, n. 5, p.40-46, mai./1991.

54 - MARESCA, M., LAVIN, M.A, LI, H. Parallel Architectures for Vision. **Proceedings IEEE**, v. 76, n. 8, p. 970-981.0-981, 1988.

55 - MARR, D. **Vision**. New York (NY): Freeman, 1982.

56 - MIKLOSKO, J. **Algorithms, Software and Hardware of Parallel Computers**. Berlim: Springer, 1984.

57 - MUDGE, T.N., RAHMAN T.S.A. Vision Algorithms for Hypercube Machines. **Journal of Parallel and Distrib. Comput.** v. 4, p. 79-94, 1987.

58 - O'GORMAN, F., CLOWES, M.B. Finding Picture Edges through Collinearity of Feature Points. **IEEE Trans. on Computers**, v. 25, p. 449-456, 1976.

59 - PIZARRO, D.A. Implementação Paralela do Algoritmo Linhas e Superfícies Escondidas em Máquinas MIMD Fracamente Acopladas. **Tese de Mestrado**. FEE/UNICAMP, 1991.

60 - ROSENFELD, A., ORNELLAS, J.J., HUNG, Y. Hough Transform Algorithms for Mesh-Connected SIMD Parallel Processors. **Comput. Vision Graphics and Image Processing** v. 41, p. 293-305, 1988.

61 - RUETZ, P.A., BRODERSEN, R.W. An Image-Recognition System using Algorithmically Dedicated Integrated Circuits. **Machine Vision and Applications**, v. 1, p. 13-22, 1988.

62 - RUIZ, E.E.S. Comparação de Técnicas e Métodos para Visão Computacional em Ambientes Industriais. **Tese de Mestrado**. Faculdade de Engenharia Elétrica/UNICAMP, 1989.

63 - RUMELHART, D., McCLELLAND, J.L. **Parallel and Distributed Processing**. Cambridge (MA): MIT Press, 1986.

64 - SOARES, L.F.G. **Modelagem e Simulação Discreta de Sistemas.** (Versão preliminar preparada para a VII Escola de Computação, São Paulo, USP).1990.

65 - SOUSEK, B., SOUSEK, M. **Neural and Massively Parallel Computers**. New York (NY): John Wiley, 1988.

66 - TERADA, R. **Introdução à Complexidade de Algoritmos Paralelos.** (Versão preliminar preparada para a VII Escola de Computação, São Paulo, USP). 1990.

67 - TMC. (Thinking Machines Corp.). **C* Programming Guide.** Version 6.0 Beta. ago./1990.

68 - TSAI, R.Y., LENZ, R.K. Real Time Versatile Robotics Hand/Eye Calibraion Using 3D Machine Vision. **Proceedings of the IEEE Conference on PAMI**. p. 554-561, 1988.

69 - TZVI, D.B., SANDLER, M. Analog Implementation of the Hough Transform Suitable for Single Chip VLSI Implementation. **Proceedings of the IEEE Symp. on Circuits and Systems**. p. 53-56, 1990.

70 - VAN VEEN, T.M., GROEN, F.C.A. Discretization Errors in the Hough Transform. **Pattern Recognition**, v. 14, p. 137-145, 1981.

71 - WALLACE, A.M.. Industrial Applications of Computer Vision since 1982. **IEE Proceedings**. v. 135, n. 3, p. 117-131, 1988.

72 - WANG, L.L., TSAI, W.H. Computing Camera Parameters Using Vanishing-Line Information of a Rectangular Parallelepiped. **Machine Vision and Applications**, v. 3, n. 3, p. 129-141, 1990.

73 - WANG, L.L., TSAI, W.H. Camera Calibration by Vanishing Lines for 3-D Computer Vision. **IEEE Transactions on PAMI.** v.13, n.4, p. 370-376. abr./1991.

74 - ZHANG, T.Y., SUEN, C.Y. A Fast Parallel Algorithm for Thinning Digital Patterns. **Communications of the ACM**, v. 27, n. 3, p. 236-239, 1984.