SISTEMA DE RECONHECIMENTO DE IMAGENS BASEADO NO MODELO GSN DE REDE NEURAL

MÁRIO LUIZ TRONCO

Tese apresentada à Escola de Engenharia de São Carlos, da Universidade de São Paulo, como parte dos requisitos para obtenção do Título de Doutor em Engenharia Mecânica.

ORIENTADOR: Prof. Dr. Arthur José Vieira Porto

São Carlos
1999

Ficha catalográfica preparada pela Seção de Tratamento da Informação do Serviço de Biblioteca – EESC/USP

T853s

Tronco, Mário Luiz
Sistema de reconhecimento de imagens, baseado no modelo GSN de rede neural / Mário Luiz Tronco. -- São Carlos, 1999.

Tese (Doutorado) -- Escola de Engenharia de São Carlos-Universidade de São Paulo, 1999.
Área: Engenharia Mecânica.
Orientador: Prof. Dr. Arthur José Vieira Porto.

1. Reconhecimento de imagens. 2. Redes neurais. 3. Modelo GSN. 4. Processamento paralelo. 5. Visão artificial. 6. Montagem automatizada. I. Título.

FOLHA DE APROVAÇÃO

Candidato: Engenheiro **MARIO LUIZ TRONCO**

Tese defendida e aprovada em 12.03.1999
pela Comissão Julgadora:

Prof. Associado **ARTHUR JOSÉ VIEIRA PORTO (Orientador)**
(Escola de Engenharia de São Carlos - Universidade de São Paulo)

Prof. Doutor **EDUARDO VILA GONÇALVES FILHO**
(Escola de Engenharia de São Carlos - Universidade de São Paulo)

Doutor **RICARDO YASSUSHI INAMASU**
(Pesquisador - EMBRAPA)

Prof. Doutor **JOSÉ FRANCISCO FERREIRA RIBEIRO**
(Instituto de Ciências Matemáticas e Computação - Universidade de São Paulo)

Prof. Doutor **JORGE LUIZ E SILVA**
(Universidade Federal de São Carlos - UFSCar)

Prof. Associado **ARTHUR JOSÉ VIEIRA PORTO**
Coordenador da Área de Engenharia Mecânica

JOSE CARLOS A. CINTRA
Presidente da Comissão de Pós-Graduação

À minha esposa, Veridiana Agda
e aos meus pais, Antonio e Vilma.

AGRADECIMENTOS:

Ao Prof. Dr. Arthur José Vieira Porto, pela orientação, incentivo e amizade;

À FAPESP, pelo financiamento do Projeto;

Ao Prof. André C. P. L. F. Carvalho, pelo auxílio no início do trabalho;

Ao Prof. Weber Martins, pelo auxilio na fase de testes;

À amiga Jandira, pela amizade e grande contribuição no desenvolvimento das rotinas de software, sem a qual este trabalho não teria sido viabilizado;

Aos amigos Pós-Graduandos Edilson, Lobão, Orides, Osvaldo e Sayuri, pela amizade e colaboração.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE DEFINIÇÕES E TERMOS TÉCNICOS

AGV - (*Automated Guide Vehicle*) Veículo Guiado Automaticamente

ASRS - (*Automated Storage & Retrieval System*) Sistema de Carga e Descarga Automatizada

CAQC - (*Computer Aided Quality Control*) Sistemas Computacionais de Auxílio ao Controle de Qualidade

CAM - (*Computer Aided Manufacturing*) Sistemas Computacionais de Auxílio à Manufatura

CEP - Controle Estatístico de Processos

CIM - (*Computer Integrated Manufacturing*) Manufatura Integrada por Computador

CNC - Comando Numérico Computadorizado

Coarse Coading - Técnica de Codificação de Imagens, com redução de dimensionalidade

CPAD - Controlador Paralelo de Alto Desempenho

DFA - (*Design for Assembly*) Projeto voltado à Montagem

DFM - (*Design for Manufacturing*) Projeto voltado à Manufatura

FMS - (*Flexible Manufacturing System*) Sistema Flexível de Manufatura

GRAM - (*Generalizing Random Access Memory*) modelo booleano de Rede Neural o qual possui uma fase de propagação

GSN - (*Goal Seeking Neuron*) Modelo de Rede Neural (FILHO, 1990)

Aprendizado *Lazy* – Técnica de Aprendizado onde o nerônio aprende somente quando estritamente necessário

HONN - (*Higher Order Neural Networks*) Redes Neurais de Alta Ordem

MPLN - (*Multi PLN*) PLN de N estados

PCA - (*Principal Components Analysis*) Análise de Componentes Principais

PLN - (*Probabilistic Logic Neuron*) modelo booleano de Rede Neural o qual trabalha com um terceiro valor, indefinido

RNA - Redes Neurais Artificiais

SDM - (*Sparse Distributed Memory*) modelo de Rede Neural, baseado em RAM, proposto por KANERVA

SLAM - (*Stored Logic Adaptive Memory*) modelo booleano baseado em RAM

T9000 - Última geração de processadores voltada para o processamento paralelo, desenvolvida pela INMOS

TMS320C40 - Processador DSP voltado para o processamento paralelo desenvolvido pela Texas Instruments

RESUMO

Este trabalho trata da implementação de um sistema de reconhecimento de imagens, baseado em Redes Neurais, para ser utilizado em operações de montagem automatizada.

O sistema é baseado na utilização do modelo GSN de Rede Neural, incorporando modificações no modelo básico, as quais são baseadas em conceitos de Algoritmos Genéticos. Tais modificações proporcionam o controle dinâmico da geração de redes iniciais, da ordem de apresentação das imagens ao sistema e da geração de códigos de saída.

A principal característica apresentada pelo sistema é a de invariância geométrica, possibilitando ao mesmo trabalhar com imagens transladadas, escalonadas e rotacionadas em relação às imagens previamente aprendidas. Outra característica importante do sistema é sua capacidade de trabalhar com imagens ruidosas. Todos estes aspectos levam em conta também o requisito de tempo real de operação, necessário para os requisitos impostos pela aplicação em questão.

Em sua estrutura final, o sistema foi projetado de forma a poder ser implementado em uma arquitetura paralela, em especial a Arquitetura CPAD.

Palavras Chave: Reconhecimento de Padrões, Redes Neurais Artificiais, Modelo GSN, Montagem Automatizada, Processamento Paralelo.

ABSTRACT

This work concerns the implementation of na image recognition systems, based on neural networks, to be used in automated assembly operations.

The system is based on the use of the GSN model of neural network, incorporating modifications to the basic model, which are based on concepts of Genetic Algorithms. Such modifications provide a dynamic control of the generation of initial nets, presentation order of the images to the system and generation of output codes.

The main characteristic presented by the system is the geometric invariance, allowing it to work with translated, scaled and rotated images in relation to the learned images. Another important characteristic of the system is its capacity to work with noise. All these aspects also take into account the requirement of real time operation, necessary for the requirements rates of the application under study.

In its final structure, the system was designed to allow parallel architecture implementation, specially the CPAD architecture.

Keywords: Pattern Recognition, Artificial Neural Networks, GSN Model, Automated Assembly, Parallel Processing.

1. INTRODUÇÃO

1.1 CONSIDERAÇÕES INICIAIS

As operações de montagem, uma das principais aplicações de robôs, situam-se numa área onde a automação total é rara, devido aos altos custos envolvidos, além de existirem poucos sistemas comerciais implementados com êxito.

A evolução de sistemas mecânicos de montagem, os quais baseiam-se em tarefas simples e repetitivas de manipulação, com alto grau de tolerância, num meio cuidadosamente controlado, leva aos sistemas de montagem automatizada. Tais sistemas, de montagem automatizada, empregam visão artificial para trabalhar com problemas relacionados à tolerância das peças, peças defeituosas, além de erros de posicionamento das mesmas (DAVIES & GILL, 1993).

A utilização de Sistemas de Visão Artificial para o reconhecimento de objetos vem despertando bastante interesse em diversas áreas de aplicação. Avanços tecnológicos vêm possibilitando a implementação de algoritmos mais complexos de análise de imagens. Processadores mais rápidos e paralelos, grandes quantidades de memória disponíveis a baixo custo e desenvolvimento de técnicas de processamento e reconhecimento de imagens são os principais agentes deste processo (COSTA, 1996).

Existem vários métodos para o reconhecimento de padrões, a maioria dos quais utilizando técnicas de processamento baseadas em algoritmos convencionais de processamento de imagem. Recentemente, com o crescente interesse em torno de Redes Neurais Artificiais (RNA's), sistemas de reconhecimento de padrões baseados em modelos neurais de processamento vem sendo investigados, implementados e testados. Tais sistemas, incorporados aos robôs manipuladores, resultam em sistemas inteligentes os quais podem operar em meios estruturados e não estruturados, através do uso de mecanismos avançados de realimentação sensorial, e tomando decisões usando algoritmos de aprendizado e raciocínio.

O modelo GSN (Goal Seeking Neuron) de Rede Neural, é um modelo booleano desenvolvido por FILHO (1990) e que tem como principal inovação em relação aos modelos booleanos anteriores o fato de que as redes GSN podem armazenar e também fornecer valores indefinidos, u, em suas saídas. Este modelo, portanto, trabalha com incertezas internas.

Existem diversas arquiteturas possíveis para o modelo GSN. Embora utilize um algoritmo rápido de treinamento e seja bastante apropriado às aplicações que envolvam tarefas de reconhecimento de imagens, conforme exaustivamente explorado em CARVALHO (1994), algumas modificações foram propostas no modelo original, com o objetivo de otimizar o funcionamento do mesmo.

MARTINS (1994) implementou modificações no modelo GSN original, utilizando conceitos de Algoritmos Genéticos, as quais relacionam-se à otimização das pirâmides geradas inicialmente, da ordem de apresentação dos exemplos à Rede Neural e da geração dos códigos de saída, resultando em um sistema hierárquico.

1.2 OBJETIVOS DO TRABALHO

Os objetivos do presente trabalho estão diretamente relacionados àqueles definidos no **Projeto FAPESP** "Aplicação de Arquiteturas Paralelas em Controle de Tempo Real", o qual foi desenvolvido no Laboratório de Máquinas Ferramentas - LAMAFE, da EESC - USP. Tal projeto envolveu a pesquisa e desenvolvimento de arquiteturas paralelas voltadas para o controle em tempo real de sistemas que demandam elevado desempenho computacional.

A Arquitetura Paralela Desenvolvida no projeto, denominada CPAD - Computador Paralelo com Alto Desempenho (OSHIRO, 1998) foi utilizada, como definida nos objetivos do Projeto FAPESP, para o desenvolvimento de um sistema de visão artificial de robô, para ser utilizado em tarefas de montagem automatizada.

Assim, o objetivo principal deste trabalho foi o desenvolvimento e implementação de um sistema de reconhecimento de imagens, baseado no modelo GSN de Rede Neural, com propriedades de invariância a transformações geométricas, que tolere imagens ruidosas e que possa operar em tempo real em operações de montagem automatizada, trabalhando com imagens de grandes dimensões como 1024x1024, 2048x2048 pixels, etc. Tal sistema deve ser apropriado para trabalhar sob a Arquitetura CPAD de processamento paralelo (OSHIRO, 1998).

O sistema deve receber como entrada uma imagem real, obtida do objeto a ser reconhecido. Após as operações de aquisição da imagem, incluindo filtragem de ruído, o padrão é Pré-Processado por um estágio que garante a invariância geométrica. O padrão resultante é então processado por um bloco gerador de campos *coarse*, no qual é implementado um esquema de compressão de dados. Os campos *coarse* são classificados por uma Rede Neural baseada em GSN, a qual fornece como saída a classificação do padrão.

Através da utilização de processamento paralelo, cada campo *coarse* gerado é classificado por um processador, o que resulta em baixo tempo de processamento, apropriado aos requisitos de tempo real que a aplicação em questão exige.

Neste sentido, imagens reais devem ser testadas para avaliar o sistema. Os principais parâmetros envolvidos devem ser avaliados: geração de dados para o módulo de controle (ângulo do eixo principal, posição, escala e dimensões do objeto), desempenho do sistema de classificação, tolerância a transformações geométricas, além dos tempos de processamento envolvidos.

O sistema final, em função de sua flexibilidade, deverá ser adequado a diversas outras aplicações industriais, além daquela originalmente desenvolvida (montagem automatizada) envolvendo classificação e análise de imagens, além de aplicações em outras áreas, como a área médica (diagnósticos por imagens) e de segurança (identificação baseada em padrões da íris, impressões digitais, entre outros).

Para que o sistema apresentasse as características citadas anteriormente, diversos algoritmos foram desenvolvidos, os quais implementaram os diversos módulos componentes do sistema. Todos os algoritmos foram desenvolvidos levando em conta as características específicas do sistema proposto, apresentando portanto diversas modificações e adaptações em relação às implementações tradicionalmente apresentadas na literatura, além de levar em conta a necessidade de integração em *software* e *hardware* e de adequação à arquitetura CPAD.

1.3 ORGANIZAÇÃO DO TRABALHO

O presente trabalho, além da introdução, possui outros cinco capítulos. No segundo capítulo é feita uma introdução à Manufatura Integrada por Computador, através de seus conceitos básicos. Os principais aspectos relativos a CIM são apresentados, assim como seus componentes, evidenciando a necessidade de integração dos mesmos. Os níveis de gerenciamento são também apresentados. A seguir, apresenta-se uma introdução à Montagem Automatizada, no ambiente CIM. Os principais conceitos são apresentados e aspectos relativos a sua implementação avaliados. Finalmente, o estágio atual da Montagem Automatizada é apresentado, em suas principais abordagens, através dos principais trabalhos publicados na área.

O terceiro capítulo traz uma introdução às Redes Neurais Artificiais, através da descrição dos conceitos básicos relativos ao tema, além do histórico de seu desenvolvimento. Apresenta-se então um estudo de Redes Booleanas, através de seus conceitos básicos e principais modelos. Tais modelos são apresentados e avaliados. O modelo GSN de Rede Neural Artificial é então apresentado, através da descrição de seu funcionamento e conceitos básicos. As principais arquiteturas possíveis de serem implementadas utilizando o modelo GSN são apresentadas e avaliadas. Os principais conceitos envolvendo a operação deste modelo em tarefas de reconhecimento de padrões são mostrados, além das abordagens existentes. Conceitos de Algoritmos Genéticos, aplicados ao modelo GSN são então apresentados, além dos principais aspectos envolvendo o funcionamento do modelo GSN modificado.

No quarto capítulo é apresentado um estudo da Visão Computacional, através da descrição dos principais conceitos relacionados ao tema. O problema do Reconhecimento de Padrões é então discutido e avaliado, através da descrição das principais abordagens possíveis. Dá-se ênfase especial ao problema do Reconhecimento de Padrões utilizando Redes Neurais Artificiais, o qual é descrito em detalhes, além da apresentação de trabalhos recentes realizados na área. O capítulo é finalizado com a apresentação de trabalhos relativos à implementação de Visão Computacional para Robôs.

O quinto capítulo apresenta a implementação do Sistema de Visão Artificial para Robô de Montagem proposto. Os blocos componentes do sistema são então apresentados, através da descrição da estratégia de implementação e características de funcionamento. Exemplos de resultados experimentais obtidos utilizando cada bloco componente do sistema proposto são mostrados e avaliados. A integração dos blocos é proposta, utilizando a arquitetura CPAD, através da descrição da estrutura e forma de funcionamento do sistema integrado. O capítulo termina com a apresentação dos principais resultados obtidos.

No sexto capítulo são apresentadas as principais conclusões do presente trabalho, além de uma discussão a respeito de trabalhos futuros e perspectivas de implementação de um sistema para utilização em tarefas reais de manipulação de peças.

O Apêndice-I traz o Teorema Fundamental dos Algoritmos Genéticos, com conceitos que são utilizados no terceiro capítulo e que também estão inseridos no modelo de classificador proposto e implementado no presente trabalho.

No Apêndice II, as principais características da Arquitetura CPAD, implementada por OSHIRO (1998) são apresentadas, através de conceitos de funcionamento e de implementação. Diagramas da implementação do arranjo de processadores também são mostrados.

O Apêndice III traz as principais rotinas de *Software* implementadas (em C - plataforma Borland 4.0).

2. MANUFATURA FLEXÍVEL E SISTEMAS DE MONTAGEM AUTOMATIZADA

2.1 INTRODUÇÃO

Analisando-se os sistemas produtivos nos últimos anos, observa-se que estes sofreram grandes alterações, passando-se de uma situação em que a principal característica da produção era a baixa diversificação de produto, com um mercado consumidor pouco exigente, para a situação atual, onde a característica principal é a grande variedade de produtos fabricados, com a predominância de produtos individualizados. Na situação atual, as empresas tem que reagir rapidamente às necessidades de mercado, mantendo baixos os níveis de custos de produção, para poder competir num mercado cada vez mais dinâmico.

Para alcançar níveis de produção competitivos, trabalhando com mudanças rápidas de mercado, as empresas têm buscado dois objetivos principais: Automação e Integração. Através destes objetivos, a filosofia de produção de variedades de produtos pode ser implementada, buscando na flexibilidade de adaptação do sistema produtivo às mudanças de produto, a solução para acompanhar as alterações de mercado e consequentemente poder atingir níveis competitivos.

A implementação de novas tecnologias na manufatura é uma estratégia multidimensional (BOHN & JAIKUMAR, 1988; CAMARINHA-MATOS et al., 1996; JAIKUMAR & SHIRLEY, 1987), já que tem com conseqüências: aumento na uniformidade de produção, redução de custos e variabilidade de produtos, eliminação de atividades de risco, além da utilização de recursos humanos em trabalhos que envolvam maior uso de capacidade intelectual e menor esforço físico.

A Automação da Manufatura pode ser dividida em três categorias básicas: Automação do "Chão-de-Fábrica", Automação de Engenharia e Automação no Planejamento e Controle.

A nível de "Chão-de-Fábrica", utilizam-se Sistemas de Computadores no Auxílio à Manufatura (CAM - Computer Aided Manufacturing), Controle de Qualidade (CAQC - Computer Aided Quality Control) e ferramentas de Controle Estatístico de Processos (CEP).

O CAM origina-se do desenvolvimento do processamento de informações, especialmente para o controle de máquinas ferramentas, representando a automação de uma indústria a nível de "Chão-de-Fábrica", através do uso de Células e Sistemas Flexíveis de Manufatura.

O conceito de Manufatura Integrada por Computador (CIM - Computer Aided Manufacturing) refere-se basicamente ao uso da tecnologia de computadores ligando todas as funções relacionadas à manufatura de um produto, caracterizando-se como um sistema de informação e controle de manufatura.

Os benefícios que resultam dos investimentos da implementação de CIM são:

- ✓ Mudanças na Estrutura de Custos, relacionadas basicamente à substituição do trabalho humano pelas máquinas e redução de custos variáveis;
- ✓ Aumento da Repetibilidade dos Processos, o qual tem impacto competitivo, devido à redução do trabalho de correção e melhoria de desempenho dos produtos;
- ✓ Redução de Inventários, através de redução de tempo de montagem, minimizando a necessidade de estoques;
- ✓ Aumento da Flexibilidade, através de rápidas trocas de ferramentas e equipamentos, possibilitando mudanças rápidas de produtos, em resposta às variações de demanda de mercado;
- ✓ Redução do Tempo de Trânsito entre as estações de processamento, através de redução de distâncias de movimentação de materiais e otimização das rotas a serem seguidas pelos mesmos.

O objetivo da utilização do conceito de CIM é incrementar a habilidade de uma empresa em produzir e vender produtos, através da utilização de pessoas, instalações físicas, equipamentos e depósitos de materiais. Três objetivos chave são estabelecidos para este propósito (MATHER & GARNER, 1989; GARRET, 1989):

- ✓ Manufatura consistente de produtos de qualidade, os quais devem satisfazer as necessidades do mercado consumidor;
- ✓ Incrementar a produtividade da instalação física, através de ferramentas e informações adequadas ao pessoal de operação e gerenciamento;
- ✓ Estratégia de produção de bens de maneira responsiva à atuação da empresa no mercado.

A figura 2.1, a seguir, ilustra o conceito de CIM.

As operações com clientes são mostradas no topo da figura. A disposição física da fábrica é mostrada na parte inferior. Ligando as duas partes está o conceito de CIM. A planta física deve ser efetivamente integrada com operações de gerenciamento da empresa. Estas devem ser ligadas aos sistemas de operações com clientes através de um fluxo de produtos e informações necessário para tomada de decisões. A meta final é um fluxo de informações e materiais demanda/fornecedor ligando as várias organizações e operações de forma efetiva e integrada.

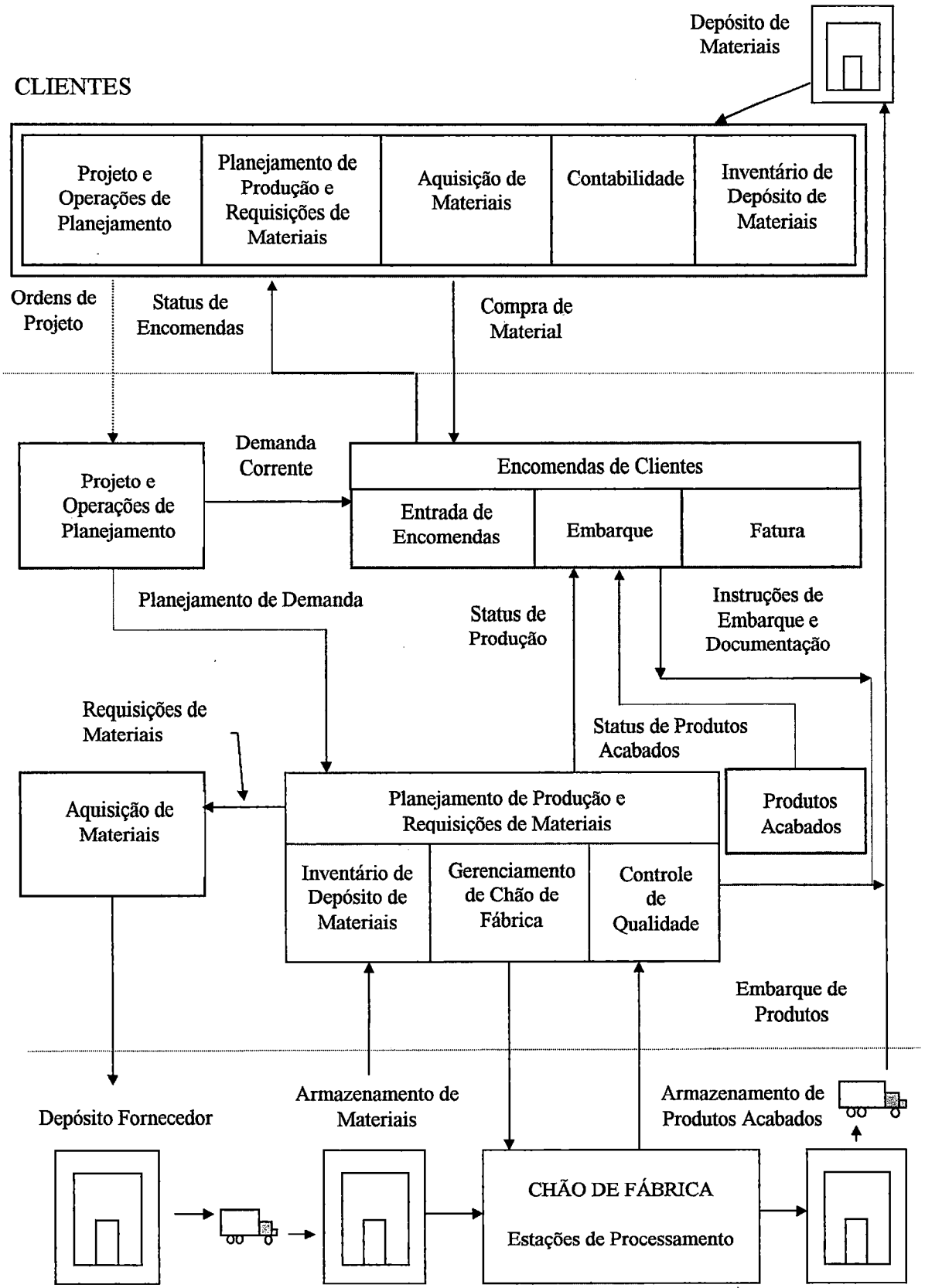


Figura 2.1. Implementação de CIM (MATHER, 1989).

O conceito de CIM incrementa a eficiência das operações, facilitando a tomada de decisões, proporcionando informações de forma rápida, precisa e efetiva. Isto proporciona planos diários que são parte de operações diárias, os quais podem ser adaptados ou trocados de acordo com as necessidades da empresa ou em função de variações nos produtos. Estes diversos processos de tomada de decisão são organizados em níveis hierárquicos, sendo o de maior prioridade o Escritório Central, passando pelos níveis de Gerenciamento de Operações e Controle de Chão de Fábrica, e terminando na Planta Física, que tem a menor prioridade (MATHER, 1989).

O conceito de CIM também é organizado em níveis **em relação à arquitetura**, como pode ser visto na figura 2.2, a seguir.

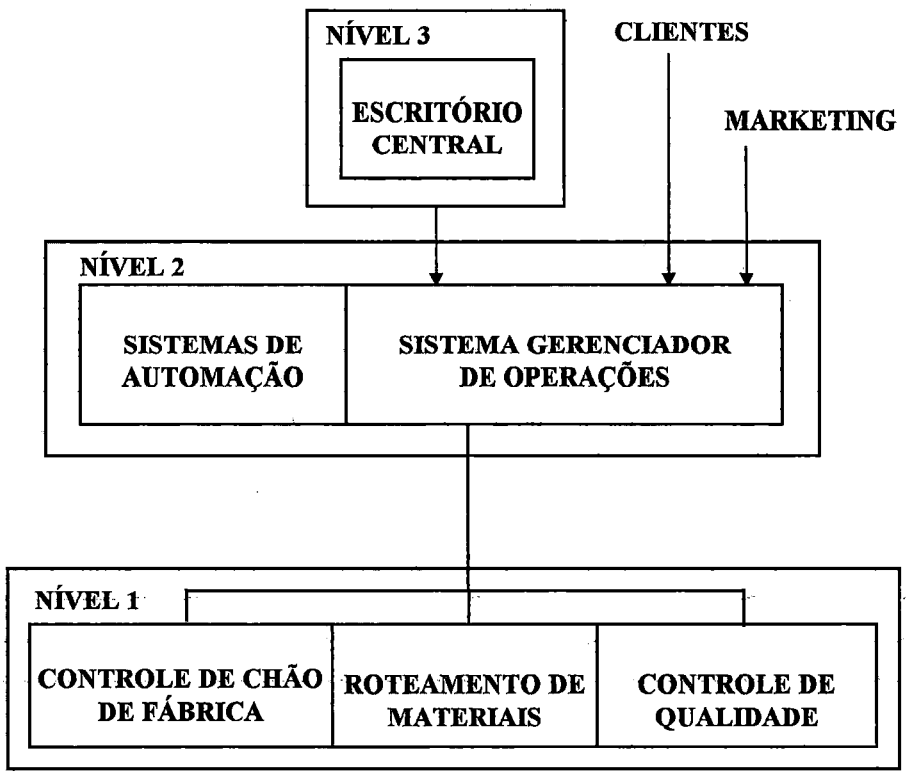


Figura 2.2. Arquitetura CIM (MATHER, 1989).

No nível 1, tem-se o controle do processo físico; o nível 2 mostra os sistemas de gerenciamento de operações e, no nível 3, o escritório central. A planta física (nível 0) não é mostrada.

Os sistemas de controle de chão de fábrica incluem: Controle de Processo Físico, Roteamento de Materiais, Produtos Acabados e Produtos em Processamento, além de Sistemas de Controle de Qualidade. Os níveis 1 e 2 são interligados via rede local de comunicação de dados, a qual também interliga os vários elementos do sistema de controle de chão de fábrica. Através de Links de Comunicação (linhas privativas de comunicação de dados, acessos comutados, etc.) a fábrica é interligada ao escritório central, clientes e serviços de vendas.

A estratégia das instalações industriais, a nível de implementação de CIM, incluí **níveis de integração**, como mostrado a seguir (MEREDITH & HILL, 1987):

- **NÍVEL 1**: Definido como standard, representando o hardware padrão, normalmente controlado por computadores existentes nas máquinas ou por controladores programáveis. Este nível incluí equipamentos com AGV`s, máquinas ferramentas NC, robôs, além de outros equipamentos que fazem uso limitado de informações locais.

- **NÍVEL 2**: Consiste de grupos celulares de equipamentos e materiais para a produção de famílias de peças, através de um elevado grau de integração e comunicação. Normalmente, este nível é obtido através do agrupamento celular de diversos equipamentos individuais do nível 1, realizando diversas funções, utilizando-se das potencialidades do sistema integrado de informações.

- **NÍVEL 3**: Este nível é obtido através da conexão de diversas Células do nível 2, formando ilhas, através da utilização de redes informatizadas. A principal característica das ilhas é a flexibilidade.

- **NÍVEL 4**: Este nível representa a integração total. Grandes redes de informações interligam todas as funções de manufatura. Este nível inclui sistemas de nível 3, além de equipamentos de transporte e níveis de gerenciamento. Este nível de integração representa o conceito de CIM.

Em relação às estruturas de produção, as principais categorias podem ser definidas como segue (AGOSTINHO, 1989):

- ✓ Linhas de Transferência Flexíveis;
- ✓ Células Flexíveis de Manufatura;
- ✓ Sistemas Flexíveis de Manufatura.

Linhas de Transferência Flexíveis são normalmente utilizadas para fabricação em alta escala, havendo a possibilidade de produção de peças diferenciadas.

Células Flexíveis de Manufatura são utilizadas na produção de peças individuais ou pequenos lotes de peças. Realizam todas as funções necessárias para completar o processo de produção da peça programada. São muito versáteis quanto a variações no tipo de peças fabricadas, dependendo de programação de seus elementos componentes para alterações no processo produtivo.

O *Sistema Flexível de Manufatura* é composto por estações de processamento interligadas por sistemas automatizados de manipulação e de carga/descarga de materiais, permitindo a produção de volumes variáveis de peças diferentes.

2.2 SISTEMAS FLEXÍVEIS DE MANUFATURA (FMS)

Um Sistema Flexível de Manufatura (FMS) consiste tipicamente de um conjunto de Células (unidades autônomas que realizam diversas funções de manufatura), interligadas por um Sistema Automático de Manipulação de Materiais. O Sistema Automático de Manipulação de Materiais tem por função proporcionar a entrada apropriada para as células, de tal maneira que estas possam realizar suas tarefas de manufatura, além de retirar o produto destas operações, ou seja, peças, produtos acabados, ferramentas, etc. (ODED & ODED, 1986; CHETTY et al., 1996).

Em contraste com linhas de transferência (alta produção) onde todas as peças seguem a mesma seqüência de operações, no FMS o Sistema de Manipulação de Materiais permite às peças seguirem rotas diferentes. O FMS pode processar um volume grande de pequenos ou médios lotes de peças, de razoável complexidade geométrica.

As características de produção permitidas num FMS são mostradas a seguir (GROOVER, 1987):

- ✓ Produção de famílias de peças;
- ✓ Possibilidade de introdução instantânea de novas peças no sistema;
- ✓ Redução do tempo de comando de produção;
- ✓ Redução de inventários em processo;
- ✓ Redução de trabalho direto e indireto;
- ✓ Melhor controle de manutenção.

A característica principal do FMS é a grande versatilidade de seus componentes, de maneira que, quando uma peça é introduzida no sistema, ela é automaticamente roteada para as diversas estações necessárias para o seu processamento.

Todas as atividades de coordenação e controle no FMS são realizadas por computadores. Estas atividades podem ser agrupadas como segue (GROOVER, 1987):

- ✓ Controle Numérico individual das Máquinas;
- ✓ Controle Computacional do Sistema de Manipulação de Materiais;
- ✓ Monitoração de dados de produção, relacionados ao controle das peças, trocas de ferramentas, utilização das máquinas, etc.;
- ✓ Controle Supervisor, através de funções relacionadas ao controle de produção, controle de tráfego no sistema de manipulação de materiais e controle de ferramentas.

2.2.1. Componentes do FMS

Os componentes do FMS podem ser classificados em três categorias principais:

- ✓ Estações de Processamento;
- ✓ Sistema Automático de Manipulação de Materiais;
- ✓ Sistema de Controle Computadorizado.

Estações de Processamento

As Estações de Processamento de um FMS dependem, geralmente, dos requisitos de processamento do sistema, incluindo máquinas ferramentas a controle numérico (CN) e máquinas a Controle Numérico Computadorizado (CNC). Uma máquina CN pode executar programas que controlam seus movimentos, necessitando, no entanto, de um operador para carregar as peças, ferramentas, etc. Uma máquina CNC tipicamente possui um computador para executar e armazenar programas de operações. Possuem também dispositivos automáticos de carga/descarga de peças e troca de ferramentas., além da capacidade de desenvolvimento on-line de programas, permitindo ao operador introduzir novos dados e alterar os programas de operações.

De maneira geral, as estações de processamento devem possuir uma certa parcela de flexibilidade, já que realizam uma grande variedade de operações em diversas geometrias de peças (MEREDITH, 1987).

Sistema Automático de Manipulação de Materiais

O Sistema Automático de Manipulação de Materiais movimenta peças entre estações de processamento, locais de armazenamento e pontos de descarga. Estes sistemas incluem veículos guiados automaticamente (AGV), robôs e armazéns automatizados. Através da conexão de pontos distintos do sistema de produção, são implementadas funções de integração que reduzem o tempo de trânsito entre diferentes pontos do processo de produção.

O Sistema de Manipulação de Materiais deve proporcionar movimento independente das peças, além de buffers de armazenamento para os locais de carga/descarga e estações de processamento. Outra característica importante do sistema relaciona-se à compatibilidade com o Sistema de Controle, de tal maneira que possa suportar alterações e mudanças no layout (alterações da capacidade de manipulação) (PIERSON, 1984). Aspectos relacionados ao acesso às máquinas ferramentas e operação em ambiente de chão de fábrica devem ser levados em consideração quando da implementação do sistema, de tal maneira que possibilite acesso fácil e desobstruído às estações de processamento, além de operar corretamente na presença de cavacos de metal, fluidos, poeira, etc.

Sistema de Controle Computadorizado

O Sistema de Controle de um FMS trabalha com grandes quantidades de dados (configuração de máquinas, status do sistema, planejamento de processos) através de complexos algoritmos de controle. Para tal, o sistema deve ter como requisitos básicos (NOF et al., 1980):

- ✓ Armazenamento de programas de produção de peças;
- ✓ Distribuição dos programas de produção de peças para as estações de processamento;
- ✓ Controle da Produção;
- ✓ Controle de tráfego, no sistema de transporte de peças;
- ✓ Controle da carga/descarga de peças nas estações de processamento;
- ✓ Monitoração do Sistema de Manipulação de Materiais;
- ✓ Controle das Ferramentas;
- ✓ Monitoração do desempenho do sistema de produção.

Devido à complexidade do FMS, à potencial diversidade de roteamento de peças e à variabilidade em tempos de operação, o controle do sistema é segmentado em níveis de **Planejamento de Pré-liberação**, **Controle de Entrada** e **Controle Operacional** (BUZACOTT & SHANTHIKUMAR, 1980). Na fase de Pré-liberação, decide-se quais peças serão manufaturadas pelo sistema, identifica-se a seqüência de operações e estima-se a duração das operações. No Controle de Entrada determina-se a seqüência e temporização das operações liberadas pelo sistema. No nível de Controle Operacional, determina-se as rotas de movimentação das peças entre as estações de processamento.

Em função da inerente complexidade do controle operacional, pode-se transferir certas decisões do nível operacional para o nível de pré-liberação. O Controle de Entrada e o Controle Operacional também podem ser simplificados, reduzindo a quantidade de informações coletadas no sistema. Assim, as decisões referentes aos trabalhos a serem realizados podem ser tomadas com base em informações coletadas diretamente das estações de processamento.

Basicamente, o sucesso do controle automatizado de sistemas de manufatura depende do equacionamento do processo de tomada de decisão humana e a captura da essência desta formulação em estruturas apropriadas de controle (CAMARINHA-MATOS et al., 1996; SACKETT & FAN, 1989).

2.2.2. Sistema Automático de Manipulação de Materiais

O uso de robôs e máquinas controladas por computador tem resultado em instalações capazes de produzir uma larga faixa de produtos os quais, em geral, requerem a realização de diferentes operações. Para que a potencial capacidade destas instalações possa ser explorada, é necessário prover um Sistema de Manipulação de Materiais, o qual permita que as estações de processamento possam ser visitadas em qualquer seqüência.

Para tal, mecanismos de manipulação de materiais e algoritmos de roteamento devem ser utilizados, de acordo com as características da instalação.

A função do sistema de manipulação de materiais é movimentar peças, normalmente fixadas em pallets, de um ponto a outro. Para tal, diversos mecanismos são utilizados, como segue (ZIRSK, 1983, EGBELU & TANCHOCO, 1984, RAVINDRAN et al., 1988):

- ✓ Veículos Comandados;
- ✓ Transportadores;
- ✓ Robôs;
- ✓ Dispositivos de Armazenamento Automatizado.

Veículos Comandados

Nos sistemas comandados, as peças são transportadas de um ponto a outro através de um carro comandado individualmente, incluindo AGV`s (veículos guiados automaticamente), Carros Automotores e Veículos Guiados por Trilhos.

Um **AGV - Automated Guided Vehicle** é um avançado sistema transportador, composto por um veículo sem condutor, o qual segue uma trilha e é controlado por um computador ou microprocessador de bordo. O AGV é capaz de selecionar sua própria rota. Tem basicamente a mesma liberdade de um empilhadeira manual, sem necessitar de um operador, possuindo computador de bordo, o que permite o recebimento de comandos (do sistema de controle), identificação da carga a ser transportada, destino a ser atingido, além de outras funções.

AGV's podem ser facilmente interfaceados com robôs, AS/RS's, máquinas CNC e, de maneira geral, com todo tipo de equipamento automatizado controlado por computador. Como sistema de transporte, o AGV proporciona flexibilidade e adaptação, podendo adaptar-se a alterações no produto fabricado ou no processo de produção, num tempo bastante reduzido.

Basicamente, os AGV's podem ser classificados em dois tipos:

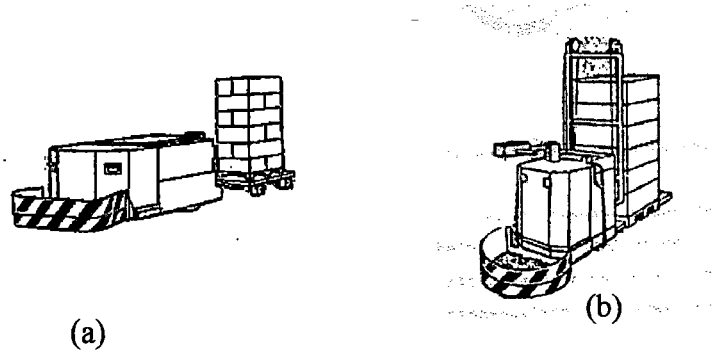
- Carregadores de Material;
- Transportadores de Carga Específica.

AGV's carregadores de material compreendem Veículos Rebocadores e Carros-Pallet.

Veículos Rebocadores são carros guiados, aos quais são acoplados vagões (traillers), sendo aplicados principalmente no transporte de materiais envolvendo longas distâncias e poucos pontos de destino.

Carros Pallets são utilizados para transporte de material nos casos em que há baixo fluxo de material através de distâncias moderadas, com muitos pontos de destino.

A figura 2.3, a seguir, ilustra os dois tipos de AGV`s.



(a) (b)

Figura 2.3. AGV's do Tipo Veículo Rebocador (a) e Carro-Pallet (b).

Transportadores de Carga Específica são construídos para transportar cargas simples ou múltiplas, através da utilização de uma plataforma denominada convés. Esta plataforma, na qual a (s) peça (s) está fixada, pode ser provida de mecanismos de transferência os quais podem movimentar acessórios, ferramentas, peças, etc. de uma célula para outra, possibilitando ao AGV funções de montagem dentro do FMS.

A figura 2.4, a seguir, mostra dois tipos de Transportadores de Carga Específica

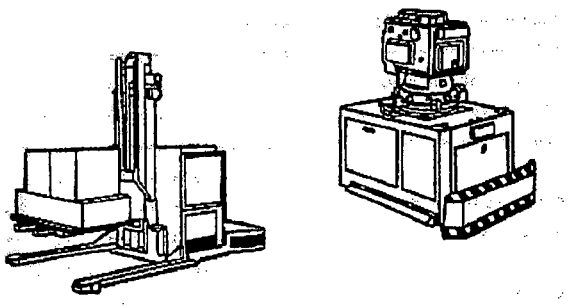


Figura 2.4.Transportadores de Carga Específica

Os AGV's vem sendo integrados a outros tipos de dispositivos automatizados, tais como robôs, máquinas CNC, estações de inspeção automática, sistema de armazenamento automatizado, além de sistemas de montagem automatizada. Características como flexibilidade, adaptabilidade e manuseabilidade fazem do AGV (principalmente o de carga específica) um dispositivo indispensável em sistemas avançados de manufatura.

Alimentadores são mecanismos que utilizam correntes tracionadas, às quais prendem-se braços suporte. As peças a serem transportadas são presas aos braços, através de mecanismos apropriados. O conjunto é montado em trilhos suspensos e o layout é determinado de tal maneira que todos os pontos de coleta/entrega tenham acesso ao alimentador.

Através da implementação de dispositivos de desconexão, as peças podem ser retiradas do alimentador, quando requisitadas num determinado ponto; através de desvios, normalmente próximos ao ponto requisitado, as peças podem ser colocadas no alimentador, fazendo com que os braços suporte passem por dispositivos de embarque.

Carros Automotores são pequenos transportadores com um motor de bordo (elétrico). Através do trilho de suporte, são fornecidos a alimentação do motor e sinais de controle do mesmo. A conexão do carro de um trilho para outro é feita através de dispositivos eletrônicos os quais, através da leitura de sensores, identificam o carro e seu destino e o conectam ao trilho determinado.

Transportadores

Transportadores são equipamentos utilizados para movimentação de peças e materiais, utilizando dispositivos carregadores os quais se deslocam através de diversas técnicas de propulsão. Os principais tipos de transportadores incluem: Transportadores por roletes, por camada de ar e transportadores modulares.

Robôs

Um robô pode ser definido como um manipulador programável, multifuncional, projetado para manipular materiais, peças, ferramentas ou dispositivos especiais através de movimentos programáveis, para executar tarefas variáveis. Os tipos principais de robôs incluem: Manipuladores Mecânicos (Pick and Place) e Robôs Programáveis, figura 2.5, a seguir. Os robôs programáveis podem ser agrupados em gerações, de acordo com a capacidade de programação, realimentação e sensoreamento.

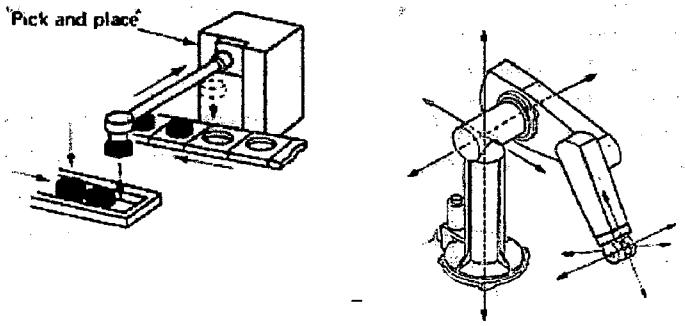


Figura 2.5. Manipuladores Mecânicos e Robôs Industriais.

Quando utilizados em Sistemas Flexíveis de Manufatura, os robôs podem ser classificados como:

Manipuladores Mecânicos: são utilizados em funções específicas nas quais realizam movimentos determinados, para a realização de funções tais como carga e descarga de máquinas, transporte de peças, embalagem e distribuição.

Robôs Programáveis: Permitem um controle contínuo de trajetória, ponto a ponto, gerando posicionamento preciso, com repetibilidade. São utilizados em substituição às operações manuais, tais como furação, pintura e montagem automatizada.

Dispositivos de Armazenamento Automatizado

O termo depósito de materiais envolve um conjunto de sistemas que são desenvolvidos para armazenamento de peças brutas, produtos acabados e componentes de produtos que serão utilizados em sua montagem. Estes sistemas variam muito em sua forma e capacidade de armazenamento, sendo que alguns conjugam armazenamento e manipulação em um só sistema. A tendência é a automação total do sistema, possibilitando sua implementação de maneira integrada ao sistema de manufatura flexível.

Existem diversos tipos de sistemas automáticos de armazenamento. Para cargas específicas AS/RS (Automated Storage/Retrieval Systems), Sistemas *Car-in-Lane,* empilhadeiras e algumas máquinas operadas manualmente. Para pequenas peças existem sistemas baseados em AS/RS Mini Carga, pequenas Máquinas de Seleção operadas manualmente e Carrosséis Verticais e Horizontais. Em cada instância existem projetos padrões e especiais (KARPUSHENKO & LIVINTSEV, 1985; KUPRIANOV et al., 1985).

2.3. MONTAGEM AUTOMATIZADA

2.3.1. Introdução

As tendências de mercado atuais apontam para a competição internacional acirrada, através de grande diversidade de produtos com pequeno ciclo de vida, aliados à redução de tempo de entrega e incremento de requisitos de qualidade. Além destes desenvolvimentos relacionados ao mercado, os desenvolvimentos tecnológicos também desempenham um papel determinado, oferecendo novas oportunidades para otimizar custos, qualidade e tempo de entrega. Tais desenvolvimentos tecnológicos englobam ainda conceitos de tecnologia de informação, novas estratégias de desenvolvimento, novas técnicas de processamento, além da viabilidade de sistemas flexíveis de manufatura.

As empresas tem que se ajustar (adequar) a este mercado e aos desenvolvimentos tecnológicos, de uma maneira específica que é determinada pelos seus próprios objetivos e pela sua estratégia. Sob a influência dos desenvolvimentos externos, os objetivos das empresas podem, em geral, ser divididos em: alta flexibilidade, alta produtividade, qualidade de produto alta e constante, pequenos tempos de entrega e baixos custos de produção. A otimização destes fatores competitivos normalmente resulta em maiores lucros. Para alcançar este objetivo, a maior parte das empresas busca a seguinte estratégia: aplicação de tecnologias avançadas de produção, controle de qualidade e melhoria das condições de trabalho. Em relação ao desenvolvimento de produto e produção, pode-se fazer a seguinte subdivisão de estratégias, a qual envolve:

- *Produto*: Design for Manufacturing/Assembly DFM/DFA), pequeno tempo de desenvolvimento, desenvolvimento mais freqüente de novos produtos, integração de funções para minimização do número de peças, miniaturização e padronização;
- *Processo*: Melhor Controle, ciclos com menores tempos e estoque mínimos;
- *Sistema de Produção*: Uso de componentes de sistemas modulares, universais e confiáveis, alta flexibilidade de sistema (em relação à diminuição do tamanho dos lotes e crescente variação de produtos) e integração total do produto no sistema de produção.

Dentro deste contexto, a utilização de sistemas robotizados oferece boas perspectivas para a racionalização das atividades de montagem.

Peças manufaturadas e montadas juntas formam sub-processos inseridos no processo de produção. Na manufatura, a matéria prima é processada e transformada em peças do produto mudando a forma, tamanhos e/ou propriedades do material. Na montagem, as peças do produto são colocadas juntas (montadas) em sub-montagens ou produtos finais. A figura 2.6, a seguir, mostra as relações entre estes processos funcionais e os processos de controle mais importantes numa indústria (RAMPERSAD, 1995).

Nesta figura, pode-se ver que a montagem é ligada à produção de peças através do fluxo de materiais ou produto e é também integrada ao marketing, planejamento e desenvolvimento do produto, planejamento de processo e controle de produção através do fluxo de informações.

A montagem forma um link extremamente importante no processo total de manufatura, já que sua atividade operacional é responsável por um parte importante dos custos totais de produção e do tempo de entrega. É um dos setores de trabalho mais intensivo no qual a porção de custos pode representar de 25% a 75% dos custos totais de produção (RAMPERSAD, 1994a).

A racionalização e automação da montagem oferece boa oportunidade para minimização de tempos de produção e custos. No entanto, o sucesso depende de vários fatores, como a percepção integral da montagem em conjunto com marketing, planejamento e desenvolvimento de produto, planejamento de processo, controle de produção e manufatura de peças.

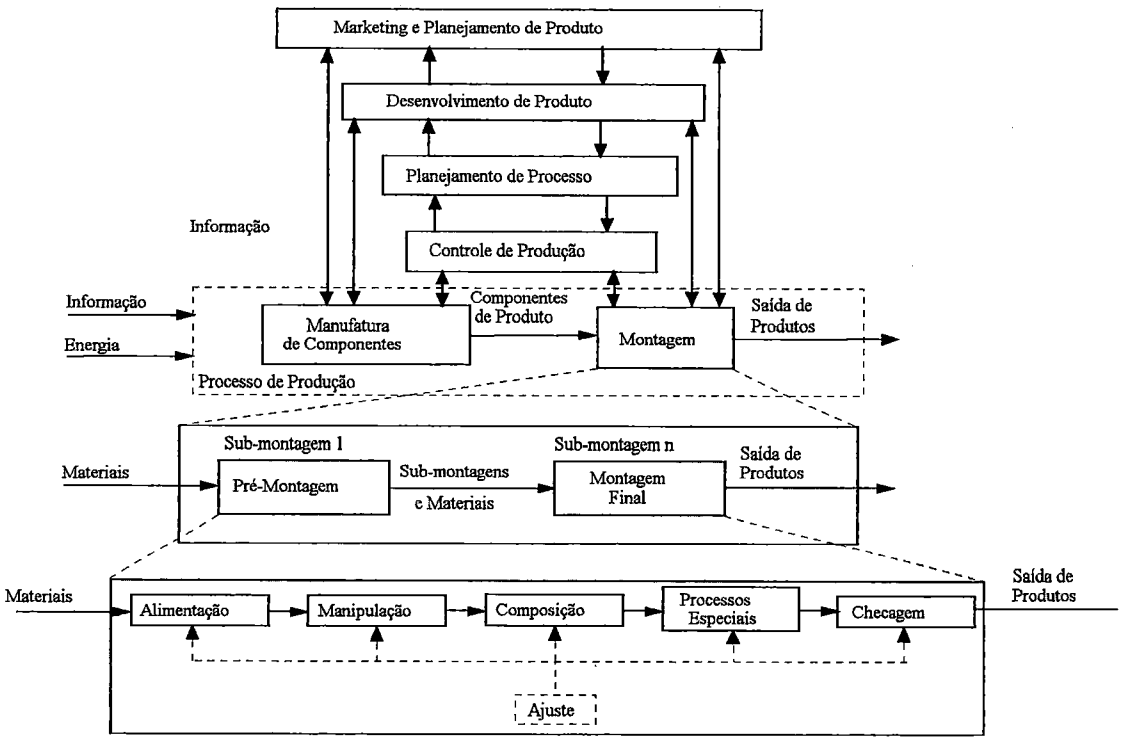


Figura 2.6. Montagem como parte do processo de produção (RAMPERSAD, 1995)

Exceto pela complexidade do projeto do produto e do processo, o desempenho de sistemas de montagem robotizada é também determinado pelo grau de sincronização entre o sistema de montagem e a manufatura de peças, a flexibilidade da garra e do equipamento periférico, assim como pela configuração do sistema (RAMPERSAD, 1994b; RAMPERSAD, 1994c).

2.3.2. AUTOMAÇÃO DA MONTAGEM

A montagem, uma das principais aplicações de robôs, situa-se numa área onde a automação total é rara, devido aos altos custos envolvidos, além de existirem poucos sistemas comerciais implementados com êxito.

A evolução de sistemas mecânicos de montagem, os quais baseiam-se em tarefas simples e repetitivas de manipulação, com alto grau de tolerância, num meio cuidadosamente controlado, leva aos sistemas de montagem automatizada.

Tais sistemas, de montagem automatizada, empregam visão artificial para trabalhar com problemas relacionados à tolerância das peças, peças defeituosas, além de erros de posicionamento das mesmas (DAVIES & GILL, 1993).

Para que um determinado sistema de montagem automatizada, baseado em robô, seja adequado às necessidades da indústria, o requisito básico é que ele deve desempenhar as habilidades de um operador humano, tais como a manipulação precisa, identificação rápida de defeitos, etc., além de desempenhá-las a baixo custo.

Os robôs produzem movimentos que resultam em manipulação ou locomoção, manipulando peças e ferramentas para realizarem tarefas de manufatura, tais como manipulação de materiais, soldagem, pintura e montagem. Veículos guiados automaticamente (AGV's) transportam materiais em fábricas e depósitos. Mecanismos baseados em robôs permitem a operação no espaço ou debaixo d'água.

As aplicações de robôs industriais tem sido, por muitos anos, dominadas por tarefas simples de manufatura (MEYER, 1988). Os robôs empregados para estes propósitos variam desde pequenas até grandes máquinas, com grande capacidade de carga.

A abordagem utilizada até poucos anos atrás era a de que o robô era uma máquina ferramenta numericamente controlada ou um determinado dispositivo periférico atuando de forma isolada e eram programados para realizar uma determinada tarefa ou uma seqüência específica de tarefas com realimentação apenas de baixo nível de sensores externos. Esta abordagem não é mais adequada devido à interdependência de requisitos mecânicos, eletrônicos e computacionais de máquinas controladas por computador, as quais utilizam sensoreamento interativo.

Numa nova abordagem, sistemas inteligentes vêm sendo pesquisados e desenvolvidos, os quais podem operar em meios estruturados e não estruturados, através do uso de mecanismos avançados de realimentação sensorial, e tomando decisões usando algoritmos de aprendizado e raciocínio.

Experiências práticas têm demonstrado que alguns gargalos colocam-se ainda como obstáculos nas aplicações de sistemas de montagem robotizada. Eles incluem: alta complexidade de produto e processo, baixa qualidade das peças do produto, assim como dependência do produto em relação aos equipamentos periféricos. Esta dependência envolve robôs e os gargalos relacionados são:

- *Aceleração e desaceleração limitadas;*
- *Meios insuficientes de Integração de Sensores Complexos;*

- *Flexibilidade Limitada das Garras e outras ferramentas de montagem;*
- *Flexibilidade Limitada dos Equipamentos Periféricos;*
- *Confiabilidade Limitada dos Equipamentos Periféricos;*
- *Pouco Acesso aos componentes individuais do sistema.*

Estes gargalos normalmente resultam em um alto consumo de capital e maior tempo de ciclo do sistema de montagem. Coerência e sincronização insuficientes entre o projeto do produto, do processo e do sistema são a base disto (RAMPERSAD, 1993).

Há alguns anos, diversos métodos DFA (Design for Assembly) vem sendo desenvolvidos para otimizar o projeto do produto, reduzindo a complexidade do processo de montagem e , consequentemente, reduzindo custos. Eles são baseados em dois princípios: eliminar e/ou simplificar operações de montagem.

A eliminação das operações de montagem pode ser realizada através do projeto modular do produto e eliminação de peças, resultado de integração. Já a simplificação das operações de montagem pode ser obtida levando-se em conta algumas regras de projeto, tais como somente uma direção de montagem, alimentação, manipulação e composição simplificada de peças, assim como bom acesso ao local de montagem.

Na área de processos de montagem, existem também novos desenvolvimentos ocorrendo, especialmente na composição de peças, onde novos processos de junção estão sendo aplicados, tais como:

- adesivos;
- encaixe rápido;
- técnicas de inserção e extração.

Com exceção dos desenvolvimentos na área de projeto de produto e processo, novos desenvolvimentos na área de sistemas de montagem robotizada tem emergido sob pressão dos gargalos mencionados e sob a influencia de desenvolvimentos externos.

Estes desenvolvimentos podem ser classificados em dois grupos: aqueles que envolvem o robô e aqueles na área de equipamentos periféricos (RAMPERSAD, 1995).

Os desenvolvimentos relativos ao robô são:

- ***Cinemática e Acionamento***: novas configurações, com novos sistemas de acionamento os quais garantem maiores velocidades e maior acuracidade;
- ***Controle***: o surgimento de melhores facilidades de controle e programação, assim como o desenvolvimento de interfaces padronizadas para a interação com o meio e para comunicação com sistemas de controle em níveis mais altos hierarquicamente. CAD e sistemas de simulação vem sendo também aplicados para programação off-line de sistemas robotizados de montagem;

- ***Sensores***: novos desenvolvimentos na área de sensores ópticos e táteis oferecem boas oportunidades para aumentar o grau de controle do processo de montagem;
- ***Atuadores***: novos desenvolvimentos na área de ferramentas de montagem e garras, especialmente a integração de sensores ópticos e táteis, assim como desenvolvimentos na área de interfaces mecânicas oferecem, em conjunto com equipamentos periféricos flexíveis, oportunidades de montagem de várias famílias de produto em um único sistema.

Novos desenvolvimentos na área de equipamentos periféricos podem ser descritos como segue.

- Desenvolvimento de sistemas de alimentação e magazines programáveis, os quais podem ser utilizados para mais de um tipo de peça;
- Integração de sensores nos equipamentos periféricos para organização de peças e controle de qualidade;
- Aumento do nível de miniaturização, universalidade e modularidade dos componentes do sistema;
- Utilização de AGV's (Automated Guided Vehicles) como sistema de transporte.

Estes desenvolvimentos são particularmente iniciados por fabricantes de robôs e instituições de pesquisa tecnológica, enquanto que do ponto de vista de engenharia industrial, existe um interesse principalmente em novas estratégias para o desenvolvimento de layouts eficientes de sistema, permitindo que vários produtos sejam eficientemente montados em pequenos lotes e em pequenos volumes.

Inspeção Visual

Inspeção é o processo pelo qual determina-se se um produto (peça, item, etc.) desvia-se de um conjunto de especificações. A inspeção normalmente envolve medidas de características específicas das peças, tais como integridade de montagem, acabamento superficial e dimensões geométricas. É uma tarefa de controle de qualidade, diferenciando-se de tarefas de testes as quais envolvem examinação ativa de funções operacionais específicas do produto. Identificação de peças, ou Reconhecimento, em contraste, consiste na identificação positiva de um objeto. A identificação do objeto normalmente não é necessária para a tarefa de inspeção (HWANG et al., 1996; ALEXIEF et al., 1996; PLATERO et al., 1996; NEWMAN & JAIN, 1995, PHAM & BAYRO-CORROCHANO, 1994).

2.3.3. DESENVOLVIMENTOS RECENTES NA AUTOMAÇÃO DA MONTAGEM

Desenvolvimentos recentes incluem a utilização de técnicas de Redes Neurais Artificiais e Visão Artificial, em conjunto com sistemas de montagem automatizada.

A aplicação de técnicas de redes neurais para o posicionamento e controle de robôs de montagem também é uma área de crescente interesse. FUKUDA et al. (1991) apresentaram um servo controlador, baseado em redes neurais, para o controle de posição, força e pegada de um robô manipulador, o qual pode se ajustar a diferentes tipos de objetos manipulados.

RABELO & AVULA (1991) apresenta um esquema de controle inteligente da garra de um robô, utilizando redes neurais hierárquicas, o qual apresenta como característica principal a capacidade de adaptação às mudanças do meio, contornando situações de falhas e imperfeições.

WU et al. (1991) e POURBOGHRAT (1993) também utilizaram redes neurais para implementar o controle de robôs, através de sistemas adaptativos.

SHIN & CUI (1991) descreveram um método, utilizando controle inteligente e técnicas de redes neurais, para evitar colisões em sistemas que utilizam multi-robôs. Este sistema utiliza uma estrutura hierárquica para coordenar os movimentos dos robôs no espaço de trabalho comum.

GREWAL et al. (1995) desenvolveram um Software para planejamento e gerenciamento da tarefa de montagem, o qual prioriza aspectos como: atributos das peças para a montagem; tarefas de montagem envolvidas, equipamentos necessários, tempos de montagem além do balanceamento das linhas de montagem. Demonstraram, através de aplicações práticas, a potencialidade da nova ferramenta de software apresentada.

WONG & LEU (1993) descrevem a implementação de algoritmo genético adaptativo para o planejamento da montagem de placas de circuito impresso. O problema de planejamento do deslocamento/sequenciamento de inserção, além do setup de máquina é equacionado através da abordagem utilizando algoritmos genéticos. O algoritmo utiliza operadores genéticos para gerar, interativamente, soluções potencialmente melhores na otimização do processo de montagem, de forma similar ao processo de evolução biológico. Eles concluíram que o método é vantajoso, já que o algoritmo pode ser facilmente modificado para resolver problemas em muitos tipos de máquinas/sistemas de montagem.

PLESCHBERGER & HITOMI (1993) apresentaram um método para implementação de montagem flexível em ambiente de Just-in-Time, o qual melhora a autonomia dos alimentadores de peças e a flexibilidade da linha de montagem. O método é formulado matematicamente, sendo facilmente implementado em sistemas que operam na filosofia de Just-in-Time. Exemplos práticos são apresentados, validando a melhoria da flexibilidade em sistemas existentes.

LING & CHANG (1993) apresentam uma metodologia para a geração automática de planos de montagem para produtos mecânicos tri-dimensionais. Eles desenvolveram uma abordagem efetiva para descrever e planejar a montagem de produtos mecânicos. Eles demonstraram soluções que transformam um projeto tri-dimensional de produto em instruções de manufatura para a montagem do produto projetado, levando em conta informações geométricas e não geométricas de montagem. Eles concluíram que a abordagem desenvolvida era facilmente aplicável a produtos mecânicos reais e adaptável a vários meios de manufatura.

GRUVER (1994), faz uma revisão da utilização de Robôs Inteligentes nas tarefas de manufatura. Aspectos como desenvolvimentos em atuadores, controladores, mecanismos, programação e sensoreamento são mostrados num ambiente de integração, envolvendo sistemas especialistas, redes neurais e fuzzy logic. Vários aspectos relativos à implementação de montagem automatizada utilizando robôs são apresentados. O autor enfatiza a necessidade de pesquisas nas áreas de coordenação de ambiente multi-robôs, robôs com estruturas paralelas e graus de liberdade redundantes, além do controle baseado em sensoreamento.

ARAI et al. (1995) avaliaram um sistema utilizando dois manipuladores, em um sistema de montagem cooperativo. O esquema de controle cooperativo é mostrado, avaliado e validado através de exemplos práticos.

Sistemas de Montagem Automatizada, utilizando robôs, em aplicações de montagem e soldagens de componentes eletrônicos também vem sendo apresentados e avaliados (ROTH & SCHNEIDER, 1993, FELDMANN & GERHARD, 1995).

MAKINO (1995) descreve uma técnica de avaliação de taxas de produção em sistemas de Montagem Flexível, mostrando que, como sistemas flexíveis de montagem consistem de vários sub-sistemas (transporte, alimentação, montagem, etc.), o tempo de ciclo total depende do número de ciclos e do número de repetições de cada operação. Cada fator afetando os resultados é apresentado e discutido.

A coordenação garra-visão também tem sido objeto de estudo para aplicações de montagem automatizada. SMAGT et al. (1993) apresentam um sistema, de coordenação garra-visão, baseado em redes neurais, o qual elimina a necessidade de calibrações, adaptando-se às mudanças no meio de trabalho. A monitoração de tarefas de montagem também vem sendo objeto de estudo, com abordagens utilizando redes neurais (EL-MARAGHY et al., 1993).

Outra área de grande interesse tem sido o controle de trajetória, baseado em redes neurais (SONG & CHANG, 1996; TAI et al., 1992).

NGUYEN & MILLS (1996) e NOF & RAJAN (1993) descrevem métodos de planejamento da tarefa de montagem, em ambiente Multi-Robôs, através da técnica de Cooperação. Aspectos como Flexibilidade e Acuracidade, Interações entre Planejamento e Projeto, Programação em ambiente de Multiprocessador e Planejamento da Cooperação Distribuída são apresentados e avaliados. Eles concluem evidenciando a necessidade de abordagens utilizando a técnica de Cooperação para implementação de tarefas de montagem automatizada.

A integração da montagem automatizada, levando em conta os requisitos necessários para atuação em ambiente de Manufatura Flexível, é uma área de bastante interesse e diversas abordagens vem sendo investigadas e implementadas (COLLET & SPICER, 1995; AMIRAT et al., 1995).

CAMARINHA-MATOS et al., 1996 apresenta e avalia uma arquitetura genérica de supervisão de tarefas robotizadas de manufatura. Através do modelamento apresentado, o desempenho dos sistemas robotizados pode ser melhorado, levando em conta parâmetros obtidos nas tarefas realizadas.

A abordagem utilizando Redes de Petri é apresentada por CHETTY et al (1996) para modelar e avaliar sistemas de montagem flexível.

SHARMA & VISWANADHAM (1996) investigaram, modelaram e avaliaram o problema de carga de máquina em sistemas de montagem automatizada, através de uma algoritmo de *simulated annealing*.

KOUVELIS & CHIANG (1996) avaliaram o problema da disposição física da matéria prima a ser utilizada em células de montagem flexível. O problema de otimização dos manipuladores também é avaliado e modelado na operação global da célula. Outro problema de manipulação, envolvendo ferramentas utilizadas em máquinas de células flexíveis de montagem automatizada, foi também avaliado e modelado por AGNETIS et al. (1996).

O problema do balanceamento de cargas e sequenciamento da montagem em linhas de montagem flexíveis foi investigado e avaliado por KORCYL et al. (1995).

TZAFESTAS & STAMOU (1997) analisaram os sistemas de montagem automatizada e propuseram um sistema especialista hierárquico, de montagem automatizada, baseado em *fuzzy logic*, o qual pode trabalhar com incertezas em termos de posicionamento e variabilidade das peças. Os resultados experimentais obtidos são apresentados e avaliados, mostrando a viabilidade de implementação do sistema.

KOH & CHOI (1997) modelaram um sistema de montagem automatizada utilizando Redes de Petri, identificando os parâmetros necessários para a otimização de desempenho das operações e minimização dos tempos inativos das máquinas. A técnica permitiu, também, a identificação do número mínimo de recursos necessários para realizar uma determinada operação, dentro dos parâmetros definidos inicialmente. O método mostrou-se bastante eficiente, o que pode ser comprovado pelos resultados experimentais obtidos através de simulações.

KIM et al. (1997) desenvolveram um sistema de sensoreamento, com múltiplas vistas, para a montagem de peças flexíveis, o qual leva em conta as deformações sofridas pelas mesmas durante a montagem. Tal sistema mede as deformações da peça em várias direções e pode ser utilizado para medir as deformações e erros de alinhamento das peças, aumentando sensivelmente o desempenho das operações de montagem deste tipo de peça.

HOVLAND & McCARRAGHER (1997) utilizaram um método que combina medidas de força e posição nas operações de montagem. Uma Rede Neural Multilayer Perceptron foi utilizada como classificador, sendo que as saídas desta rede correspondem às variáveis de contato durante o processo de montagem. A implementação da Rede Neural foi feita utilizando uma placa baseada no 68040 da Motorola e os resultados experimentais obtidos proporcionaram altas taxas de reconhecimento, acima de 94%.

WANG & CHANG (1997) apresentaram um sistema, baseado em Redes Neurais, capaz de auto-ajustar a garra de um robô manipulador utilizado em operações de montagem automatizada. O sistema opera em duas etapas: a primeira etapa envolve o treinamento da Rede Neural baseada em diversas posições de pega da garra em relação a um determinado universo de peças; na segunda etapa, a garra se auto ajusta, em função da posição da peça, para que a melhor posição de pega possa ser utilizada. A tecnologia apresentada, segundo os autores, não somente simplifica as estratégias de controle mas também proporciona maior flexibilidade na operação de auto-ajuste da garra.

TLALE et al. (1998), analisaram o uso de uma garra de uso geral, em substituição às diferentes garras normalmente utilizadas, em operações de montagem de placas de circuito impresso. Esta garra, de baixo custo, resultou em um processo mais efetivo, aliado ao seu baixo custo, quando comparada com a utilização de diversas garras. O projeto conceitual da garra foi baseado em Tecnologia de Grupo. A integração desta garra no processo de montagem resultou em flexibilidade e eficiência das estações de montagem. NEATHWAY & CANTUA (1998) também estudaram o processo de montagem de circuitos impressos, utilizando manipuladores *pick-and-place*.

AKELLA & MASON (1998) avaliaram o problema de orientação das peças, utilizando orientação sensorial parcial, em operações de montagem automatizada. Eles caracterizaram a relação entre o tamanho da peça e as condições de orientação e reconhecimento, implementando planos de operação, os quais podem trabalhar com peças de múltiplas formas e tamanhos. Tais planos são baseados no sequenciamento das operações de montagem.

FELDMANN & COLOMBO (1998) também utilizaram conceitos de Redes de Petri para avaliar o fluxo de materiais e seqüência de operações em sistemas flexíveis de manufatura, com abordagem voltada às operações de montagem. O sistema final envolve coordenação das atividades de montagem, a nível de recursos e estrutura lógica, para o controle de sequenciamento. Este sistema minimiza recursos e diminui os períodos de tempo inativos.

LIN et al. (1998) utilizaram conceitos de algoritmos genéticos para implementação de controle de inspeção em sistemas de montagem flexível. O sistema desenvolvido, por utilizar conceitos de mutação, opera de maneira eficiente na presença de falhas e proporciona maior confiabilidade às operações de inspeção.

FUNABIKI et al. (1998) também utilizaram conceitos de algoritmos genéticos em operações de montagem automatizada, para trabalhar com problemas de orientação das peças.

SHARMA & SRINIVASA (1998) utilizaram conceitos de redes neurais artificiais para calibração de câmera acoplada ao robô de montagem. Neste sistema, a calibração da câmera é feita de forma independente do esquema de controle do robô, proporcionando o conceito de visão ativa, aliada à maior flexibilidade de operação.

Nos capítulos 3 e 4, a seguir, serão discutidos com maior detalhamento os aspectos relativos à implementação de sistemas de visão artificial de robôs, para atividades de montagem, utilizando diversas técnicas, com especial interesse em Redes Neurais Artificiais.

2.4 CONSIDERAÇÕES FINAIS

Apresentou-se, neste capítulo, uma introdução ao conceito de Manufatura Integrada por Computador - CIM (Computer Integrated Manufacturing), através da descrição de seus objetivos básicos, seus componentes e possiblidades de integração. Foi feita uma descrição detalhada dos componentes de CIM, com o objetivo de destacar a importância de tais componentes na integração da arquitetura. Evidenciou-se também a importância da estratégia de integração adotada, a qual determina, quase sempre, o bom desempenho do sistema.

A montagem automatizada foi então descrita, dentro do contexto de CIM, através de seus conceitos básicos. Uma descrição dos desenvolvimentos recentes envolvendo operações de montagem automatizada foi também apresentada, através da descrição das principais abordagens apresentadas para a implementação da mesma.

O capítulo 3, a seguir, traz uma introdução aos conceitos de Redes Neurais e é finalizado com a apresentação do Modelo GSN de Rede, o qual será utilizado neste trabalho.

3. REDES NEURAIS ARTIFICIAIS - O MODELO GSN

3.1 INTRODUÇÃO

Nos últimos anos, vem sendo crescente o interesse em torno de Redes Neurais Artificiais (RNA), envolvendo pesquisadores de diversas áreas, como físicos, matemáticos, engenheiros, psicólogos, biólogos, etc. Este novo paradigma baseia-se no reconhecimento, por esta comunidade diversa, de que o cérebro processa informações de forma diferente de um sistema de computação convencional (ROMERO, 1993).

Redes Neurais Artificiais apresentam-se como um conceito computacional alternativo à abordagem convencional, baseada em uma seqüência programada de instruções.

A estrutura de uma rede neural, inicialmente, baseou-se na estrutura do cérebro e sistema nervoso. Ela consiste de uma certa quantidade de elementos de processamento (neurônios) cada qual possuindo diversas entradas, mas somente uma saída. Cada saída, no entanto, ramifica-se como entrada de outros elementos de processamento, de tal maneira que cada neurônio recebe sinais de muitos outros neurônios. Em uma rede neural típica, muitas das entradas dos neurônios são outros neurônios; o restante são do mundo exterior. Se a soma das entradas de um dado neurônio excede um "valor de entrada" (determinado por uma equação diferencial de primeira ordem) o neurônio *dispara* e envia o sinal a outros neurônios.

A interligação entre neurônios não é uma simples conexão. Cada entrada de um neurônio tem um valor ponderado que determina a intensidade da interconexão e, portanto, a contribuição de cada interconexão em cada decisão dispara/não dispara do neurônio seguinte. Em algumas redes, este valor é fixo, em outras ele pode ser modificado externamente à rede, ou pelos próprios neurônios em resposta às entradas. Essa é a chave para a habilidade da rede em exibir aprendizagem e memória.

A estrutura de redes neurais resulta então em dispositivos de computação massivamente paralela e não algorítmica. No entanto, diferentemente dos computadores paralelos, baseados em múltiplos processadores de von Neumann, elas não requerem a programação trabalhosa necessária para particionar um determinado problema numa forma aceitável de processamento paralelo.

Realmente, uma rede neural não pode ser programada. Em vez disso, ela é treinada expondo-a a um conjunto específico de dados do meio de informação, o qual habilita-a a capacidades de auto-organização e aprendizado para atingir uma meta desejada.

Uma rede neural tem a topologia de um grafo direto. Os nós nas redes neurais são denominados elementos de processamento, e os links diretos (canais de informação) são chamados interconexões. A figura 3.1, a seguir, ilustra uma rede neural típica.

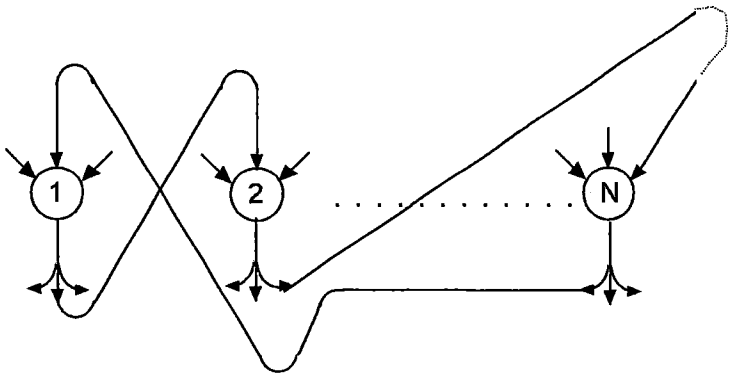


Figura 3.1 Rede Neural

Os nós 1,2,....., N representam elementos de processamento. Cada elemento de processamento aceita múltiplas entradas e gera um único sinal de saída, o qual ramifica-se em múltiplas cópias que são distribuídas a outros elementos de processamento como sinais de entrada. Cada operação no elemento de processamento é determinada pela equação diferencial que descreve como o sinal de saída irá comportar-se no tempo, em função dos sinais de entrada. Tais equações são chamadas de equações da função de transferência do elemento de processamento. Estes elementos de processamento possuem também equações diferenciais para modificar os coeficientes variáveis em sua equações da função de transferência principal. Elas são chamadas de leis de aprendizado do elemento de processamento. A rede neural completa pode então ser visualizada com um grande sistema de equações diferenciais ordinárias acopladas.

O campo de Redes Neurais, para computação, tem emergido como uma ciência interdisciplinar. Redes Neurais situam-se em uma grande classe de redes, as quais são denominadas redes de aprendizagem. Em geral, redes de aprendizagem podem ser modeladas como sistemas dinâmicos usando ferramentas matemáticas da teoria de sistemas não lineares. Na maior parte, métodos determinísticos tem sido utilizados. Redes de aprendizagem podem ser também representadas por modelos probabilísticos baseados em conceitos da teoria da informação.

Pesquisas recentes na área de redes neurais artificiais podem ser divididas em dois grupos principais. O primeiro grupo relaciona-se ao estudo dos princípios e mecanismos envolvidos no processamento de informações realizado pelo cérebro (VON DER MALSBURG, 1973, GROSSBERG, 1976a, GROSSBERG, 1976b, LINSKER, 1986a, LINSKER, 1986b, LINSKER, 1986c). As atividades de pesquisa desenvolvidas por este grupo estão direcionadas ao modelamento e compreensão da estrutura e comportamento do cérebro.

O segundo grupo está diretamente relacionado ao uso de redes neurais artificiais como um método computacional para resolver problemas práticos. O objetivo principal deste grupo não é a medida de quanto o sistema parece-se com o comportamento cerebral, mas sim quão eficientemente tais sistemas podem tratar de problemas práticos os quais não tem sido satisfatoriamente manipulados por métodos computacionais tradicionais (ARMSTRONG & GECSEI, 1979, KOHONEN, 1988, FAIRHURST et al., 1991).

Histórico

A principal motivação para o estudo de Redes Neurais é a compreensão dos mecanismos de funcionamento do cérebro. Tarefas como memorização, aprendizado, percepção, criatividade, entre outras, são executadas pelo cérebro. Embora, a princípio, encaradas como triviais, tais tarefas são de difícil simulação e implementação em máquinas tradicionais.

Redes Neurais Artificiais, de maneira geral, simulam a estrutura básica e o comportamento do cérebro, o qual é composto por aproximadamente 10^{11} células especializadas, denominadas neurônios, as quais tem um altíssimo grau de interconectividade: cada neurônio está ligado, em média, a outros 10^{4} neurônios (CARVALHO, 1994). A figura 3.2, a seguir, ilustra um neurônio típico. Basicamente, o neurônio é composto por 3 partes: soma, axônio e dendritos.

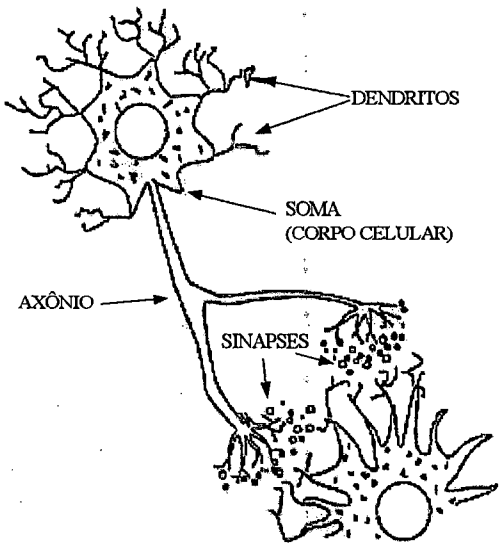


Figura 3.2. Diagrama esquemático de um neurônio.

O processamento de informações ou a comunicação entre os neurônios ocorre da seguinte forma: os sinais chegam ao corpo do neurônio pelos dendritos e lá são processados de maneira que se o somatório de todos os sinais for maior que um certo limiar intrínseco do neurônio, ele emite um pulso elétrico que se propaga através do axônio para outros neurônios; caso contrário, ele permanece inativo. A conexão entre neurônios é feita através das sinapses, as quais podem ser excitatórias ou inibitórias. Os sinais recebidos por um neurônio através de suas sinapses aumentarão ou diminuirão o potencial elétrico no interior do corpo celular. Se o potencial alcança um determinado valor de limiar, um pulso de tamanho e duração determinados é enviado pelo axônio e o neurônio dispara. Depois que disparou, o neurônio tem que esperar por um determinado período antes de ser disparado novamente.

A interação massiva de um grande número de neurônios, os quais desempenham funções relativamente simples, resulta em estruturas capazes de realizar funções altamente complexas, originando o comportamento inteligente exibido pelo cérebro.

O conceito de Redes Neurais é bastante antigo e teve como ponto de partida um artigo publicado em 1943 por Warren McCulloch e Walter Pitts (McCULLOCH & PITTS, 1943), no qual eles afirmavam que redes compostas por unidades de soma ponderada (elementos de threshold) poderiam modelar o comportamento do cérebro, implementando qualquer função matemática ou lógica. Tais unidades simulam o comportamento de um neurônio real e podem ser vistas esquematicamente na figura 3.3, a seguir.

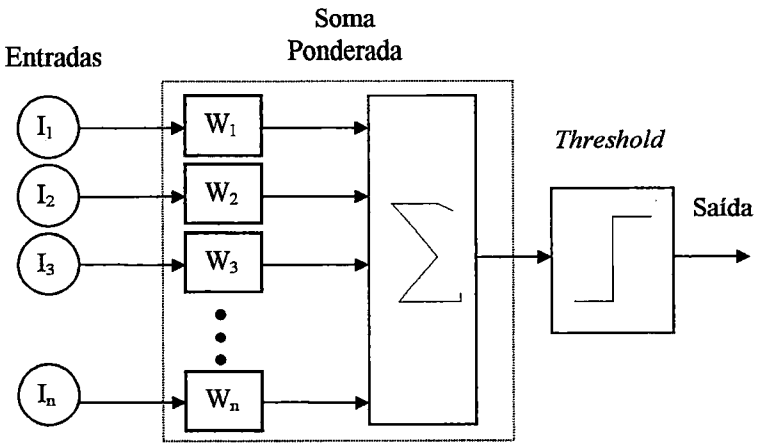


Figura 3.3. Neurônio de McCulloch - Pitts

O neurônio somente dispara quando a soma ponderada dos valores presentes nas entradas exceder um determinado valor de limiar. Os pesos, conectados às entradas, representam a intensidade das sinapses em um sistema neural real. Este modelo foi o precursor de todos os estudos que se desenvolveram na área de Redes Neurais desde então. McCulloch e Pitts, não demonstraram aplicações práticas para o modelo assim como não propuseram uma maneira de alteração dinâmica dos pesos. Tal ponto de vista, no entanto, inspirou outros pesquisadores nas áreas de Inteligência Artificial e Psicologia Cognitiva, os quais pesquisaram maneiras de incorporar uma função adaptativa e como permitir que um processo de treinamento/aprendizagem pudesse ser realizado com tais redes.

Donald O. Hebb (HEBB, 1949) contribuiu significativamente para as pesquisas na área, apresentando uma resposta ao problema do aprendizado. Sua idéia principal era que o aprendizado ocorria pela troca da intensidade das sinapses. Propôs então uma regra para o aprendizado para redes neurais artificiais, baseada neste princípio. A regra de Hebb, como ficou conhecida, definia que sempre que dois neurônios conectados fossem excitados simultaneamente, a intensidade da conexão deveria ser incrementada.

ROSENBLATT (1962) propôs um sistema prático baseado nestas idéias básicas de redes neurais. Ele desenvolveu o *perceptron*, um dispositivo de reconhecimento de padrões baseado nas conexões neurais presente nos olhos, composto de dois estágios: uma **fase de treinamento**, baseada em ajustes nos valores dos pesos para que certos padrões pudessem ser reconhecidos e **uma fase de reconhecimento**, permitindo o reconhecimento de padrões de classes presentes no conjunto de treinamento.

Esta regra de aprendizagem para o perceptron pode ser generalizada pela *regra delta* (RUMELHART et al., 1986), a qual determina que os pesos devem ser modificados por uma porção da diferença entre a ativação atual e a ativação desejada fornecida pelo professor. O perceptron também foi capaz de reconhecer padrões levemente diferentes daqueles do conjunto de treinamento.

O perceptron, mesmo sendo uma rede camada-simples, sem qualquer realimentação, foi capaz de resolver problemas simples de reconhecimento. O trabalho de Rosenblatt, provando a convergência de um algoritmo de atualização interativa dos pesos para que o perceptron realizasse uma determinada tarefa, gerou um grande interesse na área.

Surgiram então outros modelos de redes neurais, mas uma análise das limitações de estruturas baseadas em perceptrons publicada por MINSKY & PAPERT (1969), reduziu consideravelmente o interesse em redes neurais. Nesta análise mostrava-se que estruturas com camadas simples não resolviam problemas não lineares, tais como a Função XOR. Rosenblatt, de fato, estudou estruturas multicamadas mas, por não haver naquela época um algoritmo efetivo para o treinamento destas estruturas, seu trabalho ficou praticamente estagnado.

Apesar da redução do interesse na área, vários pesquisadores como ALEKSANDER & MAMDANI (1968), KOHONEN (1989), FUKUSHIMA (1975), GROSSBERG (1976a, 1976b) continuaram suas pesquisas, gerando resultados importantes que contribuíram em grande parte para o ressurgimento das redes neurais artificiais no início de 1980.

O renascimento do interesse das pesquisas em redes neurais ocorreu com a publicação de um trabalho de HOPFIELD (1982), no qual mostrava-se que redes neurais poderiam ser tratadas como sistemas dinâmicos. No entanto, somente depois de proposto um algoritmo efetivo para o treinamento de perceptrons multi-camadas, denominado *back-propagation* (RUMELHART et al., 1986) é que houve um renovado interesse na área de redes neurais artificiais. Muito do trabalho que vem sendo desenvolvido em redes neurais está direta ou indiretamente envolvido com o uso do algoritmo *back-propagation* e suas variações.

A figura 3.4, a seguir, ilustra um perceptron multi-camadas com duas camadas de neurônios de McCulloch - Pitts e uma camada de unidades de entrada.

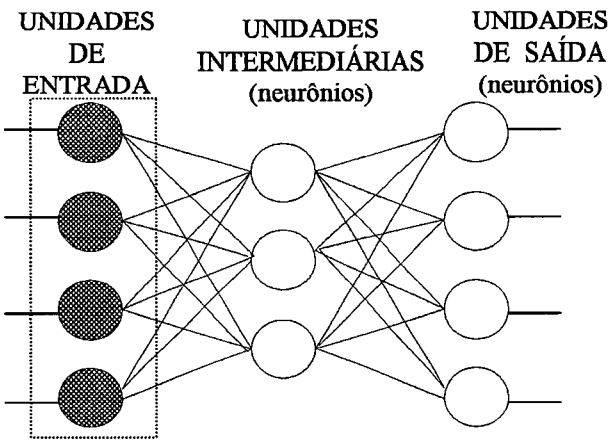


Figura 3.4. Perceptron Multi-Camadas

Nesta estrutura, os neurônios utilizam funções semilineares para gerar suas saídas, ao invés da função original de limiar. Funções de ativação semilineares são funções não-decrescentes e diferenciáveis da soma ponderada dos pesos dos estímulos de entrada. Na figura, pode-se ver as conexões entre as unidades e os neurônios na primeira camada e entre os neurônios na primeira e segunda camadas. Cada conexão tem um valor de peso associado, o qual é ajustado através de várias interações usando o back-propagation.

3.2 REDES NEURAIS BOOLEANAS

3.2.1 Introdução

Abordagens utilizando redes neurais artificiais para a realização de tarefas de classificação de imagens vem sendo investigadas nos últimos anos com bastante interesse por parte dos pesquisadores da área.. No entanto, tais abordagens tem sido insatisfatórias em muitas aplicações práticas devido às dificuldades impostas nas velocidades de processamento alcançadas, particularmente na implementação de algoritmos apropriados de treinamento e na implementação em hardware. Redes Neurais Booleanas oferecem uma abordagem alternativa ao projeto de redes neurais pela minimização de tais problemas (CARVALHO, 1994).

Redes Booleanas vem sendo estudadas por vários pesquisadores, incluindo KAUFFMAN (1969), WALKER (1971) e MARTLAND (1987a, 1987b). Sistemas práticos de reconhecimento baseados em unidades funcionais Booleanas foram desenvolvidas por ALEKSANDER et al., (1984).

Uma rede Booleana é um sistema composto por unidades interconectadas, cada uma tendo um certo número de entradas, e uma única saída. A saída pode alimentar várias outras unidades, como mostrado no exemplo da Figura 3.5, a seguir.

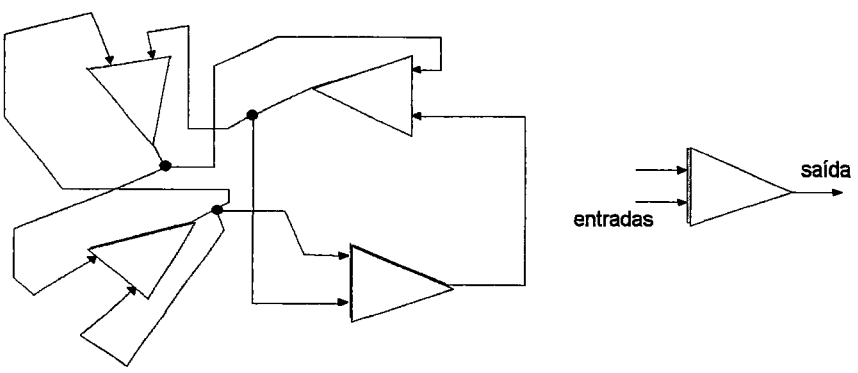


Figura 3.5. Rede Booleana.

Cada unidade na rede avalia suas entradas, as quais são tomadas no domínio B = {0,1}, e produz uma saída também no domínio {0,1}. Então cada unidade *i* na rede avalia uma função $f_i : B \times B \times B \times \ldots\ldots \times B \rightarrow B$ ou simplesmente $f_i : B^n \rightarrow B$ onde *n* é o número de entradas de cada unidade. Não há restrições nas funções usadas na rede, e elas podem ser convenientemente representadas por uma tabela verdade em cada unidade, como mostrado na figura 3.6, a seguir. Cada unidade pode realizar uma função lógica Booleana diferente na rede.

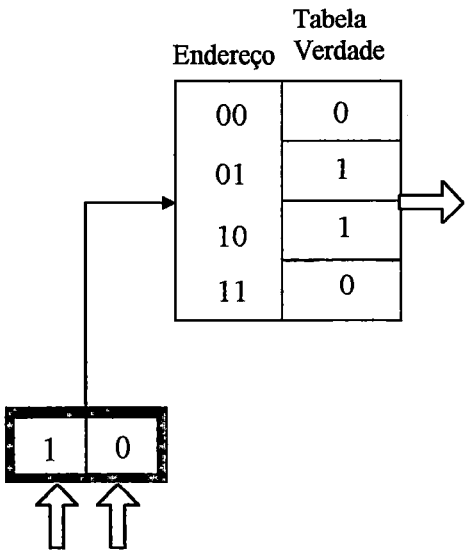


Figura 3.6. Tabela Verdade

Redes Neurais Booleanas não foram desenvolvidas com o objetivo de modelar o sistema nervoso, mas sim como ferramentas de engenharia para aplicações de reconhecimento de padrões. Em contraste com redes neurais analógicas (cujos neurônios geralmente são baseados no modelo de McCulloch and Pitts), oferecem uma abordagem alternativa ao projeto de redes neurais.

Modelos de Redes Neurais Booleanas empregando somente valores binários, utilizam algoritmos rápidos de aprendizado, sendo sua implementação em hardware potencialmente muito simples e eficiente.

Vários algoritmos diferentes de treinamento de Redes Booleanas vem sendo propostos. A maioria destes algoritmos trabalha atualizando diretamente a função lógica dos neurônios. Estas atualizações ocorrem através de aprendizado *one-shot*, por exemplo GSN feedforward (FAIRHURST et al., 1991), ou através de gradiente descendente, como ALN (ARMSTRONG & GECSEI, 1979), ou usando *simulated annealing* (WUENCHE, 1992). Outra técnica que vem sendo investigada altera as conexões ao invés dos valores armazenados nos conteúdos de memória usando *simulated annealing* (PATARNELLO & CARNEVALI, 1987), (DOYLE, 1990). Ambas as abordagens, aprendizado supervisionado e não supervisionado tem sido estudadas com modelos Booleanos.

Existem diversos modelos de Redes Neurais Booleanas, cada uma com suas características próprias. De maneira geral, podemos dividir os modelos em dois grupos:

- ✓ arquiteturas cujos neurônios são baseados em unidade de memória ou portas lógicas universais e podem ser facilmente implementados por dispositivos RAM (Random Acess Memory) - denominados modelos baseados em RAM;
- ✓ arquiteturas que utilizam pesos binários (WILLSHAW et al., 1969), (LIPPMANN, 1987) ou que restringem as funções representadas por suas unidades a um número reduzido de funções Booleanas (ARMSTRONG & GECSEI, 1979), (GARZOTTO, 1992).

Tais arquiteturas são discutidas a seguir.

3.2.2 Arquiteturas Booleanas Baseadas em RAM

Nesta seção são descritas as arquiteturas baseadas em RAM mais conhecidas, além de uma avaliação da generalização apresentada pelos modelos.

Um **neurônio-RAM** aceita e produz somente valores binários, podendo armazenar qualquer função binária, desde que uma tabela verdade esteja disponível. Esta funcionalidade permite que redes de neurônios-RAM generalizem e aprendam mais rápido que os tradicionais neurônios de McCulloch e Pitts.

O número de posições internas necessário em um neurônio-RAM é 2^n, onde n é o número de entradas. Um neurônio-RAM, como mostrado na figura 3.7, a seguir, pode ser definido por:

- um conjunto de n terminais de entrada;
- um terminal de saída;
- um conjunto de 2^n posições internas de armazenamento;
- um terminal para a saída desejada (usado durante o treinamento);
- um terminal para status de comportamento (treinamento/lembrança).

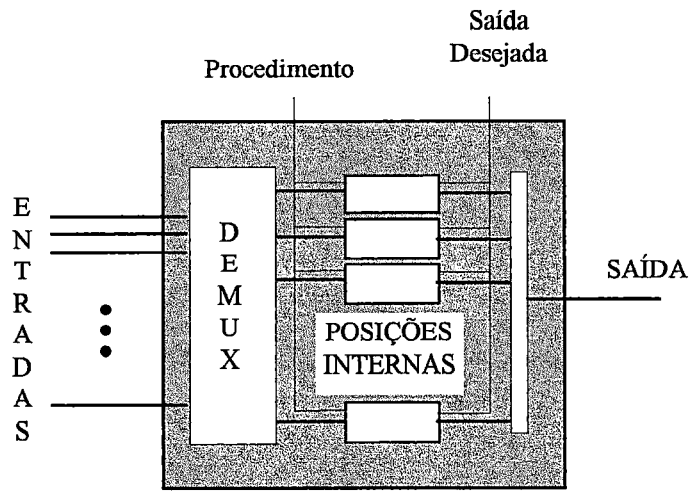


Figura 3.7. Neurônio-RAM.

Tais neurônios-RAM são utilizados em tarefas de Reconhecimento de Padrões, conforme descrito a seguir. Primeiro, o valor "0" é armazenado em todas as posições internas. Durante o treinamento, as posições endereçadas pelas entradas recebem o valor "1". Portanto, quando um novo padrão é apresentado e é similar a outros que foram "ensinados", ele usará muitos endereços comuns a estes e muitos neurônios gerarão a mesma saída. Um segundo estágio, como mostrado na figura 3.8, a seguir, é utilizado para combinar as saídas dos neurônios para fornecer uma resposta de maneira restritiva.

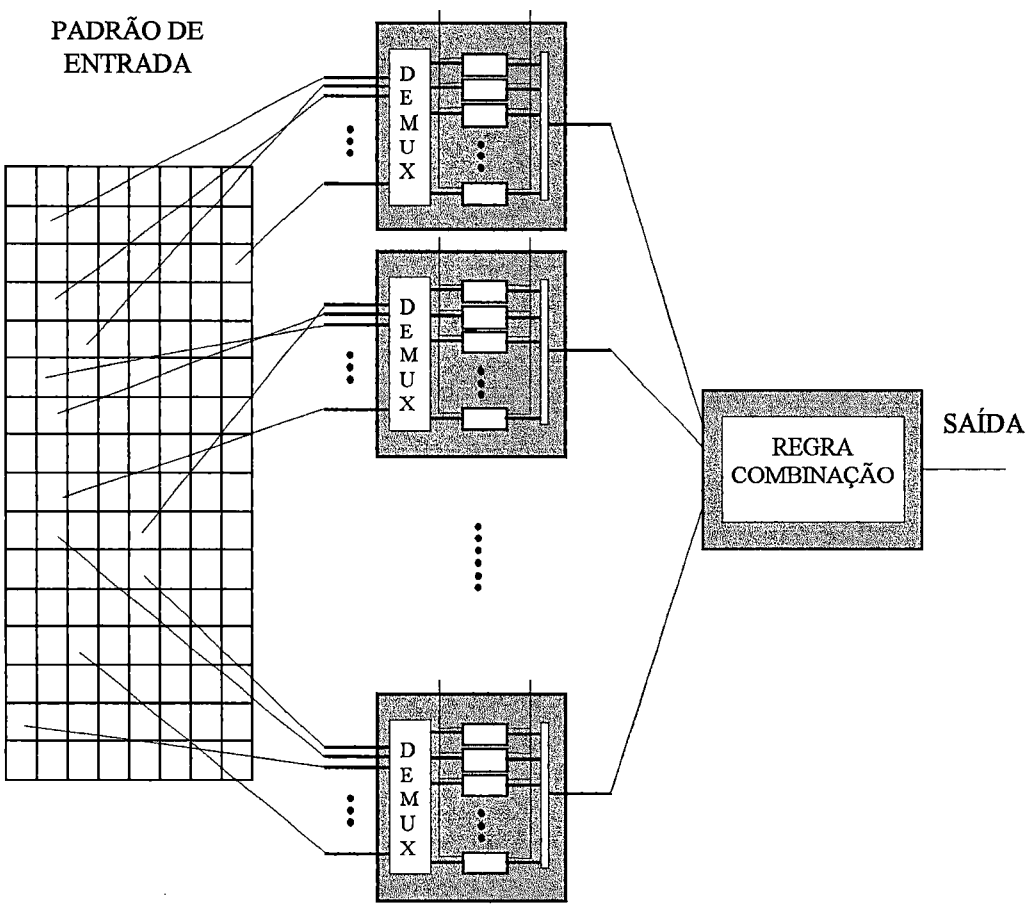


Figura 3.8. Neurônios-RAM aplicados à tarefa de Reconhecimento de Padrões.

ALEKSANDER (1989a), implementou em hardware o modelo de neurônio baseado em RAM, denominando esta implementação de **SLAM** (Stored Logic Adaptive Memory).

Através da utilização de componentes como decodificadores e flip-flops, ele obteve comportamento inteligente de dispositivos eletrônicos. Aleksander usou SLAM's como elementos básicos para construir redes neurais e justificou seu uso em função da dificuldade de implementação de redes neurais analógicas com a tecnologia de hardware disponível na época. A figura 3.9 ilustra a estrutura de uma SLAM, onde:

t → conectividade (número de terminais de entrada)

Xi → terminal de entrada (recebe padrão de entrada)

C[i] → conteúdo de memória

o → terminal de saída (transmite a saída do neurônio)

d → terminal de entrada (recebe a saída desejada)

s → estratégia de treinamento

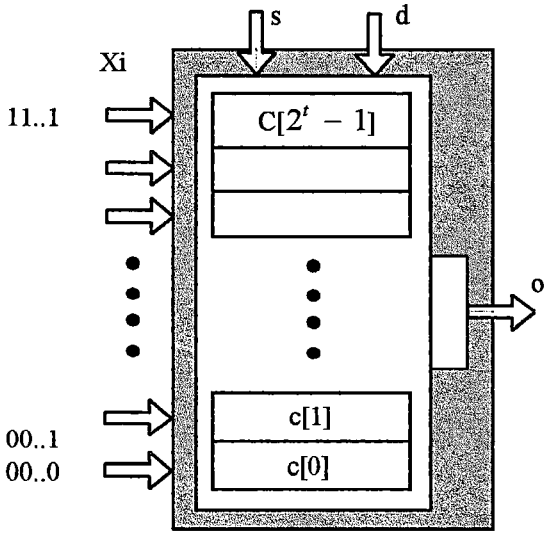


Figura 3.9. SLAM.

Como pode ser visto na figura 3.9, um SLAM tem um conjunto de terminais de entrada; X = {x1, x2,.....xt}, os quais são utilizados para receber a entrada do neurônio, um terminal de saída, *o*, que representa a saída do neurônio, um terminal de aprendizagem, s, o qual indica se o neurônio está na fase de aprendizagem ou de lembrança (recall), e um terminal de saída desejada, d, o qual é setado para o valor de saída desejada quando o neurônio está na fase de aprendizagem. Os terminais de saída de um neurônio Booleano podem endereçar um conjunto de 2^t células diferentes, onde cada célula pode armazenar um valor do conjunto {0,1}. Durante a fase de treinamento, o terminal de aprendizagem recebe o valor 1 e o valor armazenado na célula endereçada pelos terminais de entrada é alterado para o valor da saída desejada. Na fase de lembrança (recall), o terminal de aprendizagem recebe o valor 0, e a saída do neurônio é computada como o valor armazenado na célula endereçada pelos terminais de entrada.

Antes do treinamento, todas as memórias contém o valor 0 armazenado. Um valor 1 é armazenado em cada conteúdo de memória endereçado durante o treinamento. Um conjunto de SLAM's, denominado discriminador, é associado a cada classe de padrão de entrada e a classificação de um padrão desconhecido é obtida selecionando o discriminador o qual provém a maior porção de 1's em seus terminais de saída. A figura 3.10, a seguir, ilustra uma estrutura discriminador.

Aleksander desenvolveu também um dispositivo reconhecedor de padrões, em maior escala, denominado WISARD (ALEKSANDER, 1984).

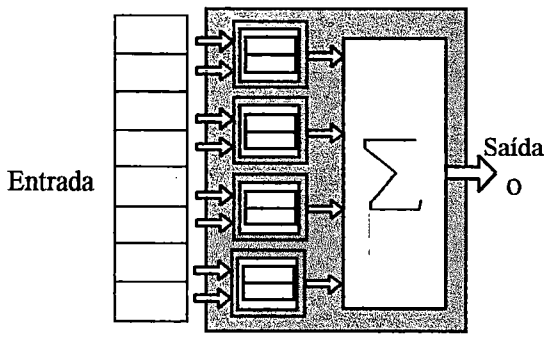


Figura 3.10. Discriminador

Os discriminadores usam um limiar (da mesma maneira que neurônios analógicos), gerando um valor 1 se a soma das saídas dos neurônios Booleanos for maior que este valor (valor de limiar). Estas estruturas proporcionam maior generalização quando comparadas a um simples neurônio cobrindo o mesmo conjunto entrada.

O modelo **PLN** (Probabilistic Logic Neuron) foi desenvolvido através de pesquisa e aperfeiçoamento em arquiteturas neurais, baseadas em modelos Booleanos. Através da utilização de um terceiro valor lógico, u (indefinido) foi possível o uso de um estado *desconhecido* na operação de arquiteturas Booleanas.

O armazenamento do valor desconhecido em todos os conteúdos de memória da rede, antes da fase de treinamento, possibilitou a manipulação do estado de *ignorância* da rede antes de ser treinada. O uso deste valor, de acordo com KAN & ALEKSANDER (1987), melhora o desempenho e a generalização da rede.

De maneira inovativa em relação aos modelos neurais baseados em RAM anteriores, ALEKSANDER (1989) propôs o uso de nós PLN em arquiteturas multi-camadas, chamadas pirâmides. Uma estrutura deste tipo pode ser vista na figura 3.11 a seguir.

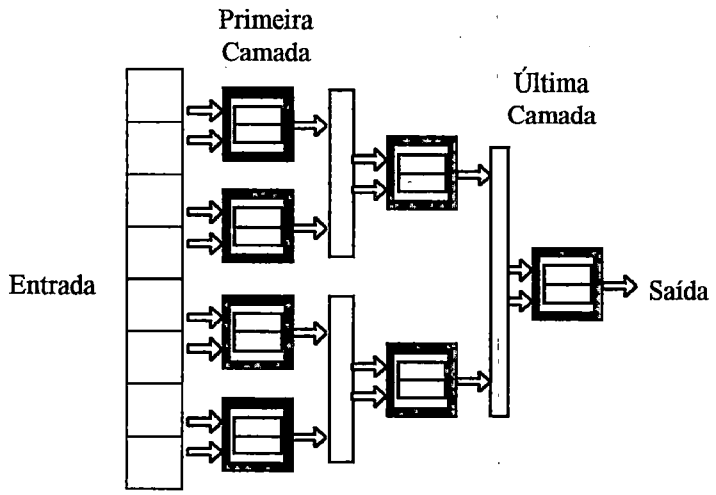


Figura 3.11. Estrutura Pirâmide

Uma pirâmide é uma estrutura multi-camadas, onde cada camada tem um número fixo de neurônios com um fan-out de 1 e um baixo fan-in. Pirâmides tem uma menor funcionalidade em relação a um neurônio simples cobrindo o mesmo campo de entrada. No entanto, seu uso reduz a quantidade de memória necessária e aumenta a generalização (KAN & ALEKSANDER, 1987)

Redes PLN tem dois estados distintos: um estado de aprendizado, onde a rede está sendo treinada, e um estado de lembrança (recall), onde a rede é usada para reconhecer padrões. MYERS & ALEKSANDER, (1988), propuseram um algoritmo de aprendizado o qual utiliza várias apresentações do conjunto de treinamento para ensinar pirâmides PLN's, empregando uma fase de recompensa (*reward*) e uma fase de punição (*punish*). Cada vez que um padrão é apresentado à pirâmide, valores definidos são propagados através dos nós. Quando um conteúdo de memória armazenando um valor indefinido é endereçado, um valor definido aleatoriamente é produzido. Se o valor gerado pelo nó no ápice da pirâmide for o correto, então todos os conteúdos de memória endereçados contendo valores indefinidos tem o valor definido produzido aleatoriamente pelo valor armazenado ali. O padrão é então reapresentado e novos valores definidos são produzidos. Se o valor correto não for gerado depois de um certo número de tentativas, um valor indefinido é armazenado em todos os conteúdos de memória e o aprendizado é reinicializado.

Na fase de lembrança, um padrão de entrada é apresentado às pirâmides. Cada vez que um conteúdo de memória contendo um valor indefinido é acessado, um valor definido é produzido (aleatoriamente) de tal maneira que valores definidos são propagados através de cada camada da pirâmide, gerando o valor de saída da pirâmide. O valor de saída produzido por cada pirâmide forma um vetor saída, o qual é comparado a um vetor saída desejada, representando cada uma das classes existentes, para que uma decisão seja tomada.

MYERS (1988) propôs a PNL N-estados ou MPLN. Esta estrutura permite o armazenamento de uma faixa maior de valores em cada conteúdo de memória. Qualquer valor real no intervalo [0.0,1.0] pode ser armazenado em um conteúdo de memória MPLN, sendo que o valor armazenado trabalha como um peso. Esta modificação permite a introdução de pequenos ajustes nos valores armazenados nos conteúdos de memória durante a fase de aprendizado. Pequenos incrementos/decrementos nestes valores levam a uma gradual aproximação aos valores desejados. Durante a fase de lembrança (recall), quanto maior o valor armazenado no conteúdo de memória endereçado, maior a probabilidade de o neurônio gerar o valor 1 como saída.

ALEKSANDER (1989b), propôs um novo modelo de neurônio booleano, denominado GRAM (Generalizing Random Acess Memory), no qual foi inserida uma fase de propagação. Nesta fase, os conteúdos de memória - para os endereços de memória modificados durante a fase de aprendizado - com endereços próximos (em termos de distância Hamming) que não foram aprendidos são também atualizados.

GORSE & TAYLOR (1988), buscando um modelo neural o qual incluísse propriedades de não-linearidade e generalização, desenvolveram um modelo Booleano baseado em PLN, o qual apresentou propriedades probabilísticas e não lineares. Este modelo, denominado **pRAM**, foi biologicamente inspirado, permitindo também, como no MPLN, o armazenamento de uma faixa de valores reais no intervalo [0.0,1.0].

O treinamento de redes pRAM é feito através de aprendizado reforçado. A fase de lembrança é similar à usada pelo PLN sendo que a saída de um neurônio pRAM apresenta uma característica não presente em outros modelos Booleanos, que é sua capacidade de aceitar valores reais em sua entrada.

KANERVA (1988) propôs uma arquitetura baseada em RAM denominada **SDM** (*Sparse Distributed Memory)*, onde cada conteúdo de memória armazena uma palavra de w elementos, sendo que cada elemento $w_i \in \{+1,-1\}$. Através do conceito de memória distribuída, somente uma fração destes conteúdos de memória é utilizada (outros modelos baseados em RAM anteriores necessitavam de 2^N conteúdos de memória quando trabalhando com N terminais de entrada). Neste caso, M endereços são selecionados aleatoriamente de 2^N endereços possíveis. Durante as fases de aprendizado e lembrança um vetor entrada I endereça todos os conteúdos de memória Mj cuja distância Hamming entre I e Aj (endereço de Mj) é menor que uma constante K.

Na fase de aprendizado, a palavra (vetor) com todos os conteúdos de memória endereçados é atualizada adicionando cada um deles à palavra desejada de saída. Cada elemento de uma palavra pode também ser visto como um contador. Na fase de lembrança, as palavras armazenadas nos conteúdos de memória endereçados são somados e cada elemento da palavra resultante, Ri, é binarizado, tornando-se -1 se for menor que zero ou +1 caso contrário.

A **generalização** é um fator de grande importância em tarefas de reconhecimento de padrões, nas quais é necessário trabalhar com padrões não presentes no conjunto de treinamento. Uma boa generalização implica em classificar corretamente estes padrões que não foram apresentados no conjunto de treinamento.

A definição do grau exato de generalização desejado não é uma tarefa trivial, já que a generalização excessiva reduz a habilidade da rede em distinguir padrões pertencentes a classes diferentes; quanto maior a generalização, menor a diferenciação.

Várias abordagens vem sendo apresentadas para controlar a generalização apresentada por modelos de redes neurais, conforme mostrado a seguir.

FERNANDES (1985) propôs controlar a proporção de cada valor definido armazenado nos conteúdos de memória, de tal maneira que quanto maior a diferença entre o número de ocorrências dos valores definidos, maior é a generalização.

ALEKSANDER (1989) e BOWMAKER & COGHILL (1992) utilizam uma outra abordagem, a qual armazena o mesmo valor em conteúdos de memória cujos endereços são similares. Desta maneira, entradas similares naturalmente produzirão as mesmas saídas.

KANERVA (1988), empregou um método o qual utiliza a técnica de *sparse coding*, para representar os vetores entrada, sendo que vetores similares endereçam conjuntos similares de conteúdos de memória. Esta abordagem tem a vantagem de permitir grandes vetores entrada sem que a quantidade necessária de memória cresça exponencialmente. Uma outra abordagem possível é a redução da conectividaade, aumentando o número de neurônios usados, substituindo um neurônio de conectividade N por uma pirâmide ou estrutura de discriminador. Esta abordagem pode levar a consideráveis economias no número total de conteúdos de memória usados. Devido à redução no número de conteúdos de memória necessários, as duas últimas abordagens levam a uma redução no número de funções as quais podem ser representadas por redes que as utilizam.

Outras Arquiteturas Booleanas

Redes Neurais Booleanas não se resumem a modelos baseados em RAM. Existem diversas arquiteturas Booleanas que usam abordagens diferentes para implementar suas funções de processamento neural Booleano.

WILLSHAW (1969) descreveu um modelo booleano o qual não utiliza neurônios baseados em RAM em seu modelo de rede, *Willshaw Net*. Tal rede é utilizada como uma memória associativa e pode ser representada por uma matriz N x M de elementos binários, onde cada elemento pode ser visto como uma interligação entre uma linha de elementos de entrada e uma coluna de elementos de saída e utiliza a técnica de aprendizado *one-shot* para setar os elementos da matriz. Esta arquitetura, no entanto, apresenta problemas de saturação rápida e baixa tolerância a ruído, os quais restringiram sua utilização em aplicações práticas.

AUSTIN & STONHAM (1987) desenvolveram uma arquitetura associativa a qual combina um processador n-tuple (front end) com dois estágios de *Willshaw Net*, denominada arquitetura ADAM - Advanced Distributed Associative Memory. Esta arquitetura apresenta alta capacidade de armazenamento e tolerância a erros.

LIPPMANN (1987) apresenta uma arquitetura Booleana radicalmente diferente dos demais modelos, a qual utiliza pesos (de dois valores) e valores de limiar, trabalhando com valores binários de entrada, sendo composta por duas camadas. Basicamente, os neurônios da primeira camada calculam a distância Hamming entre o padrão de entrada e um exemplo da classe associada ao mesmo, de tal maneira que quanto menor a distância, maior a saída produzida pelo neurônio. Os neurônios da segunda camada são associados um a um aos da primeira camada. Estes neurônios tem suas entradas conectadas às suas próprias saídas através de ligações "positivas" e a outros neurônios, através de links negativos. Haverá interação entre os neurônios até que apenas a saída de um dos neurônios seja positiva.

A arquitetura ALN - Adaptive Logic Network (ARMSTRONG & GECSEI, 1979) pode ser entendida como uma versão simplificada da abordagem baseada no back-propagation, utilizando um determinado conjunto de funções booleanas (AND, OR, LEFT, RIGHT), apresentando altas velocidades de processamento.

3.3 O MODELO GSN DE REDE NEURAL

3.3.1 Introdução

GSN (Goal Seeking Neuron) é um neurônio Booleano desenvolvido com o objetivo de superar os problemas potencialmente criados em alguns modelos Booleanos, como pouca generalização, armazenamento de memória ineficiente e indeterminismo. Redes GSN diferem de outras Redes Booleanas em diversos aspectos, sendo que a diferença fundamental está no uso do valor "indefinido", u, além dos valores binários. A diferença entre o valor indefinido usado pela PLN e o usado pelo GSN é que o GSN não é representado por um valor probabilístico, mas sim por um valor *fuzzy* (CARVALHO, 1994).

Um neurônio GSN pode receber, armazenar e gerar valores iguais a 0, 1 ou u. Se há ao menos um valor u em seus terminais de entrada, então um conjunto de conteúdos de memória será endereçado, sendo o endereço destas posições de memória derivado do endereço original pela troca de valores u com 1 ou 0. A rede, antes de ser treinada, tem todos os seus conteúdos de memória preenchidos com valores indefinidos, demonstrando que a rede inicialmente não armazena qualquer conhecimento.

O neurônio GSN tem um conjunto a mais de terminais em relação aos outros modelos Booleanos, representando os terminais de entrada desejada, os quais são usados para determinar a saída desejada de outros neurônios GSN, quando se utiliza mais do que uma camada de neurônios. A figura 3.12, a seguir mostra, esquematicamente o neurônio GSN.

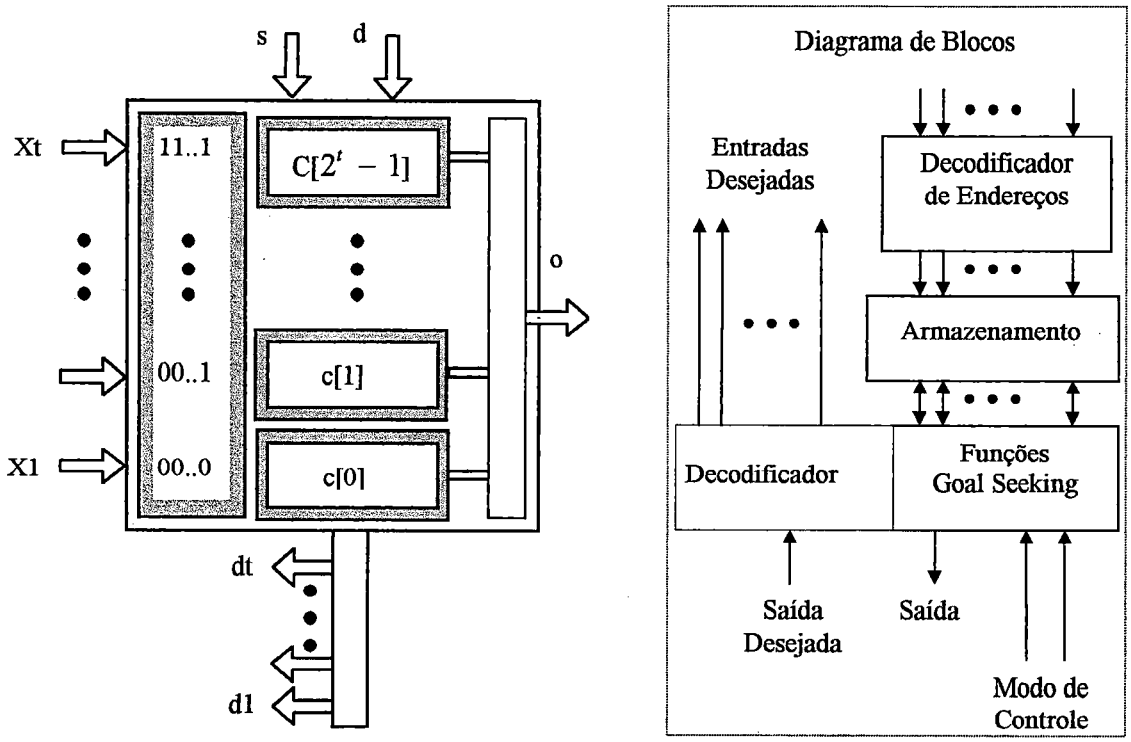


Figura 3.12. Neurônio GSN

onde:

t → conectividade (número de terminais de entrada)
Xi → terminal de entrada (recebe padrão de entrada)
C[i] → conteúdo de memória
o → terminal de saída (transmite a saída do neurônio)
d → terminal de entrada (recebe a saída desejada)
s → estratégia de treinamento
di → terminal de saída (transmite a saída desejada a um neurônio na camada anterior)

O modelo GSN utiliza um algoritmo de treinamento rápido e eficiente, já que maximiza a eficiência do armazenamento de valores em suas posições de memória, evitando o armazenamento de uma nova informação sempre que possível e armazenando informação nova sem perda da informação armazenada anteriormente. Ele é rápido porque emprega algoritmo de aprendizado *one-shot*, o qual permite que o conjunto de treinamento seja apresentado somente uma vez (ou seja, o treinamento acontece em um simples período).

Uma arquitetura baseada em GSN utiliza uma quantidade fixa de estruturas denominadas pirâmides, onde cada pirâmide cobre um subconjunto da entrada. O processamento de redes GSN pode ser dividido em três fases, cada uma associada a uma meta específica: uma fase de validação, uma fase de aprendizado e uma fase de lembrança. O objetivo da fase de validação é produzir um valor "validado" para cada pirâmide. Um valor "validado" é a saída da pirâmide para um dado padrão de entrada, definindo qual(is) valor(es) definido(s) ela pode aprender na próxima fase de aprendizado.

Uma pirâmide não pode ser ensinada quando seu valor "validado" é o oposto de seu valor de saída desejada. A fase de aprendizado ensina a pirâmide trocando os valores armazenados nas posições de memória de seus neurônios, e a fase de lembrança busca fornecer o valor definido com a maior ocorrência nos conteúdos de memória endereçáveis (CARVALHO, 1994).

A disposição dos Neurônios GSN tem sido usados em três arquiteturas diferentes, cada uma adequada a uma determinada gama de aplicações. Estas arquiteturas são: GSN feedforward (GSN^{f}), GSN feedback (GSN^{fb}) e GSN self-organizing (GSN^{s}) (FAIRHURST et al., 1991a).

A seguir, é feito o modelamento matemático do neurônio GSN.

Estado de validação

O objetivo, no estado de validação, é determinar os valores que o neurônio pode aprender na próxima fase (de aprendizagem), sem deteriorar a informação armazenada anteriormente. Para tal, a entrada a ser aprendida é alimentada nos terminais de entrada e uma função é aplicada aos valores armazenados nos conteúdos de memória GSN endereçados para determinar seu estado "validado".

Este estado é propagado através da pirâmide até seu ápice. Os valores "validados" produzidos pelo neurônio no ápice da pirâmide serão os valores "validados" da pirâmide. Se a saída é igual a u (indefinido) então ela pode aprender qualquer saída desejada. Se, por outro lado, a saída é um valor definido (0 ou 1), somente este valor pode ser ensinado. A saída de um neurônio ni, no estado de validação, é dada pela equação 3.1, a seguir:

$$o_i = \begin{cases} 0 \text{ se } \forall a_{im} \in A_i, C_i[a_{im}] = 0; \\ 1 \text{ se } \forall a_{im} \in A_i, C_i[a_{im}] = 1; \\ u \text{ se } \exists a_{im} \in A_i \,|\, C_i[a_{im}] = u \text{ ou } \exists a_{im}, a_{il} \in A_i \,|\, C_i[a_{im}] \neq C_i[a_{il}] \end{cases} \quad (3.1)$$

Nesta equação, $C_i[a_{im}]$ e $C_i[a_{il}]$ são os conteúdos de n_i que são endereçados por a_{im} e a_{il} respectivamente.

De acordo com a equação 3.1, a saída gerada no estado de validação pode ser igual a 0, 1 ou u. Quando todos os valores armazenados nos conteúdos endereçáveis são iguais a 0, a saída será igual a zero. Se, por outro lado, todos os valores armazenados nos conteúdos endereçáveis são iguais a 1, a saída será igual a 1. A saída será igual a u em qualquer outro caso.

Estado de Aprendizagem

O objetivo na fase de aprendizado é armazenar a saída desejada na posição endereçada pelos terminais de entrada. Para usar suas células eficientemente, GSN tenta armazenar sua saída desejada em um conteúdo endereçado de memória o qual já armazene este valor, de tal maneira que, quando há mais do que uma opção, uma posição de memória contendo um valor indefinido é selecionada. A equação 3.2 abaixo mostra a escolha do conteúdo.

$$a_{im} = \begin{cases} Ran(A_{i/d_i}) \text{ se } \|A_{i/d_i}\| > 0 \\ Ran(A_{i/u}) \text{ se } \|A_{i/d_i}\| = 0 \end{cases} \quad (3.2)$$

onde:

$A_{i/s} = \{a_{ik} \in A_i \,|\, C_i[a_{ik}] = s\}$ representa o conjunto endereçável do neurônio ni que armazena o valor s;

$Ran(A_{i/s})$ é um elemento escolhido aleatoriamente de $A_{i/s}$;

$\|A_{i/s}\|$ é o número de elementos que pertencem a $A_{i/s}$.

Através da equação 3.2, GSN busca por uma célula em seu conjunto endereçável a qual armazene um valor igual à sua saída desejada. Se não há tal célula, ele escolhe uma célula que armazene um valor indefinido. Se, para um dos dois casos anteriores, existir mais do que uma escolha possível, uma célula é escolhida aleatoriamente. Após escolhida a célula, o endereço da mesma deve ser enviado de volta, através dos terminais de entrada desejada, para serem usados como saída desejada pelos neurônios na camada anterior. Assim, cada neurônio nj recebe sua saída desejada, aprende e proporciona saídas desejadas para camadas anteriores. A equação 3.3, a seguir, mostra as entradas d_{ij} que devem ser enviadas à camada anterior.

$$d_{ij} = x_j \Big| \sum_{j=1}^{j=C_i} x_j z_{ij} = a_{im} \tag{3.3}$$

Estado de Lembrança (recall)

Quando o neurônio alcança o estado de lembrança, ele tem como meta produzir o valor com a maior ocorrência no conjunto endereçável. Este valor é propagado através das camadas da pirâmide da mesma maneira que o valor validado produz o valor de lembrança da pirâmide. A equação 3.4, abaixo, mostra o método usado para definir o valor de lembrança.

$$o_i = \begin{cases} 0 \text{ se } \|A_{i/0}\| > \|A_{i/1}\|; \\ 1 \text{ se } \|A_{i/1}\| > \|A_{i/0}\|; \\ u \text{ se } \|A_{i/0}\| = \|A_{i/1}\|; \end{cases} \tag{3.4}$$

Nesta equação, o neurônio fornecerá um valor 1 como saída somente se o número de tais valores (1's) nas posições endereçáveis for maior que o número de 0's. Se o inverso ocorrer, ou seja, se o número de 0's for maior que o número de 1's, a saída será igual a 0. Não importa quantos valores indefinidos existam nas posições endereçáveis. Esta regra tem o efeito de minimizar a propagação de valores indefinidos.

3.3.2 Arquiteturas GSN

Arquitetura Auto-Organizativa

Redes Neurais auto-organizativas, baseadas neste paradigma de aprendizado não supervisionado, têm uma vasta área de aplicações potenciais, principalmente em problemas de reconhecimento de padrões onde o número de classes não é conhecido à priori. Um dos princípios envolvidos na aplicação de tais arquiteturas auto-organizativas em problemas de reconhecimento de padrões é que os padrões que compartilham aspectos comuns são agrupados, com cada agrupamento representando uma e somente uma classe.

A arquitetura GSN (GSN^s) proposta por FAIRHURST et al., (1991a) segue tais princípios, agrupando padrões similares em uma estrutura de pirâmide. GSN^s cria novas pirâmides para armazenar padrões não-similares e agrupa padrões similares na mesma pirâmide, exibindo comportamento típico de uma arquitetura auto-organizativa (FAIRHURST et al., 1990; FAIRHURST et al., 1990a).

A computação realizada pela arquitetura GSN^s, como em outras estruturas GSN, é dividida em três fases: a fase de validação, a fase de aprendizado e a fase de lembrança.

Nesta arquitetura, durante o treinamento padrões pertencentes aos dados de treinamento são armazenados nas pirâmides, de tal forma que quando cessa o processo de aprendizado, cada pirâmide está associada a uma e somente uma classe de padrões, embora a mesma classe possa ser representada por mais do que uma pirâmide. Isto acontece porque padrões referentes à mesma classe podem ser compostos de aspectos diferentes, e esta arquitetura trabalha agrupando somente padrões que compartilham aspectos similares na mesma pirâmide.

Uma função de lembrança (recall) levemente diferente é usada por GSN^s, a qual é ilustrada pela equação 3.5.

$$o_i = \begin{cases} 0 \text{ se } \|A_{i/u}\| < \|A_{i/0}\| > \|A_{i/1}\|; \\ 1 \text{ se } \|A_{i/u}\| < \|A_{i/1}\| > \|A_{i/0}\|; \\ u \text{ se } \|A_{i/0}\| \leq \|A_{i/u}\| \geq \|A_{i/1}\|; \end{cases} \qquad (3.5)$$

Como pode ser visto nesta equação, a função lembrança GSN^s é mais rigorosa na generalização de uma saída definida, e um neurônio somente gerará um valor definido se, no conjunto endereçável, o número de ocorrências deste valor é maior do que o número de valores indefinidos e o número de valores opostos.

Embora relacionados à mesma estrutura geral que outras arquiteturas, GSN^s incorpora aspectos específicos próprios de sua natureza auto-organizativa. Em particular, GSN^s emprega uma fase de lembrança mais rigorosa e usa uma operação de comparação com um classificador do tipo *winner takes all* durante as fases de aprendizado e lembrança. O número de pirâmides é dinamicamente definido durante a fase de aprendizado. Inicialmente a rede consiste de somente uma pirâmide ou *cluster*, a qual é usada para armazenar o primeiro padrão do conjunto de treinamento. Quando, durante a fase de treinamento, um dado padrão P é apresentado à rede, a fase de validação é aplicada a todas as pirâmides.

A figura 3.13, a seguir, mostra esquematicamente a arquitetura GSN^s

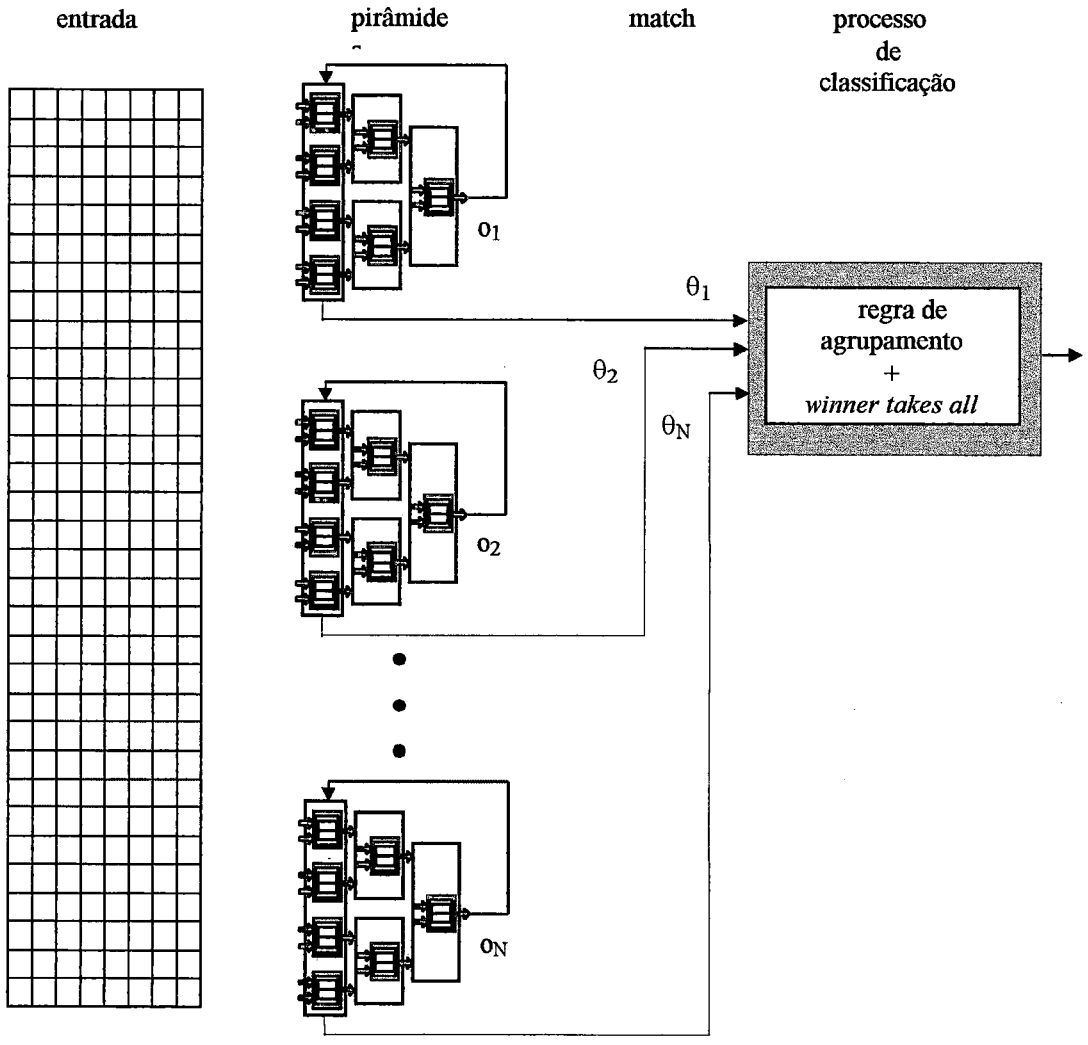


Figura 3.13. Arquitetura GSN^s.

Usando somente as pirâmides que produzem o valor 1 na fase de validação, uma comparação é feita entre *P* e os conteúdos dos neurônios na primeira camada de cada uma destas pirâmides, medindo a similaridade entre o padrão de entrada e os endereços dos conteúdos de memória os quais armazenam valores definidos. Se ao menos um dos valores estiver acima de um *valor de aprendizado*, um classificador do tipo *winner takes all* seleciona a pirâmide com o maior valor. A fase de aprendizado é então aplicada à pirâmide que produziu o maior valor, com o valor 1 sendo usado como a resposta desejada. O valor "validado" é usado para assegurar que um padrão não similar aos padrões armazenados em qualquer uma das pirâmides não seja armazenado nela.

Quando nenhuma pirâmide produz 1 na fase de validação ou nenhuma pirâmide produz um valor maior que o valor de aprendizado, uma nova pirâmide é criada e ensinada a reconhecer *P*. Quando o processo de aprendizado termina, cada pirâmide está associada à classe que é mais freqüentemente contida nela. Na fase de reconhecimento , a mesma operação de comparação é utilizada pelo classificador *winner takes all* para decidir a qual pirâmide, e portanto a qual classe, um padrão desconhecido pertence. Na fase de reconhecimento um valor de lembrança é usado para garantir que padrões que não produzem um valor de comparação alto o bastante para todas as pirâmides, não seja reconhecido como pertencente a qualquer das classes (CARVALHO, 1994).

Operação *Matching*

A **operação *matching***, por ser usada nas fases de aprendizado e lembrança, tem importância vital para a arquitetura GSN^s. Na fase de aprendizado, é usada para agrupar padrões similares na mesma pirâmide, enquanto que na fase de reconhecimento ela é usada para definir a qual classe um padrão desconhecido pertence.

Durante o processo *matching*, uma medida (match) θ_i é definida para cada pirâmide Ci. Seu valor descreverá quanto um padrão é similar aos padrões armazenados na pirâmide. Um valor determinado é utilizado para definir o grau mínimo de comparação aceitável, para o padrão ser aceito como pertencente a uma pirâmide.

Para definir θ_i para uma pirâmide Pi, o valor de comparação (match) θ_{ij} para cada neurônio n_{ij} inserido na base desta pirâmide deve ser calculado. A equação 3.6, a seguir mostra como θ_{ij} é computado.

$$\theta_{ij} = 1 - \frac{\sum \frac{Ham(i, a_{im})}{C_i}}{\left\| A^*_{i/d} \right\|}, \forall a_{im} \in A^*_{i/d} \qquad (3.6)$$

Nesta equação, a_{im} é o padrão apresentado aos terminais de entrada do neurônio nij, Ham(x,y) dá a distância Hamming entre os vetores x e y e $\left\| A^*_{i/d} \right\|$ representa o número de posições que armazenam um valor definido. θ_{ij} representa a média das distâncias Hamming entre o padrão de entrada e o endereço de cada conteúdo de memória armazenando um valor definido (FILHO, 1990).

Depois de definido o valor de θ_{ij} para cada neurônio nij na base da pirâmide Pi, o valor de comparação (match) θ_i é calculado de acordo com a equação 3.7, abaixo:

$$\theta i = \frac{\sum \theta_{ij}}{\left\| C_i \right\|}, \forall n_j \in P_i \qquad (3.7)$$

Após a avaliação do valor *match* para cada pirâmide, as pirâmides cujos valores excedem o valor de vigilância são selecionadas.

Fases de Validação e de Aprendizado

As pirâmides GSN^s são ensinadas usando somente o valor 1 para a saída desejada, de tal maneira que o neurônio no topo da pirâmide somente armazenará os valores 1 ou u. Assim, a fase de validação fará cada pirâmide fornecer o valor 1 ou o valor u. Logo após uma pirâmide aprender seu primeiro padrão, ela somente produzirá um valor de saída "validado" igual a 1 se o padrão de entrada for exatamente igual ao padrão aprendido anteriormente.

Como somente as pirâmides que produzem um valor "validado" igual a 1 são selecionadas para aprender, uma pirâmide armazenando um padrão não pode aprender qualquer padrão novo. Assim, uma pirâmide nunca armazenará mais que um padrão, ou seja, um novo padrão não pode ser aprendido por qualquer pirâmide existente. O processo de aprendizado deve então ser mais flexível, permitindo às pirâmides, cujos valores "validados" são indefinidos, passarem pela operação *matching*.

Supondo que as pirâmides sejam expostas à operação *matching* independentemente do valor produzido na fase de validação e levando-se em conta que uma pirâmide fornece u ou 1, pode-se dizer que a fase de validação deve ser usada para definir os conteúdos endereçados a serem aprendidos durante a fase de aprendizado, mas ela deve ser feita com um algoritmo mais simples (CARVALHO, 1994).

Se as saídas validadas são ignoradas e somente o resultado da operação de comparação (match) é usado para selecionar a pirâmide que irá aprender, o uso de 1 como única saída desejada resultará em somente os neurônios na primeira camada armazenando mais do que um valor definido. Os conteúdos de memória nos neurônios de outras camadas que já armazenam valores definidos serão selecionados pelo algoritmo de aprendizagem para criar o mapeamento entre o padrão de entrada e a saída desejada 1. Isto se deve ao fato de que, quando os neurônios de outras camadas estão sendo ensinados, toda vez que há uma opção entre um conteúdo endereçado armazenando um valor definido e um conteúdo endereçado armazenando um valor indefinido, o primeiro será selecionado, fazendo com que o mesmo caminho seja selecionado.

Quando termina o processo de aprendizado, cada pirâmide está associada a uma classe. Para definir qual classe será relacionada com uma dada pirâmide Pi, o número de padrões de cada classe armazenada em Pi é calculado, sendo a classe mais representativa associada a Pi.

Fase de lembrança

Na fase de lembrança são definidos os valores de saída para todas as pirâmides, sempre que um novo padrão for apresentado para reconhecimento.

Utilizando somente as pirâmides que produziram 1 como valor de saída, um valor *match* é calculado para cada uma das pirâmides, sendo estes valores definidos da mesma maneira que na fase de aprendizado. As pirâmides cujos valores *match* forem maiores que um determinado valor (vigilância de lembrança) competem entre si, para determinar qual tem o maior valor. Esta competição poderá produzir:

- uma decisão correta de reconhecimento;
- um reconhecimento errado;
- uma rejeição (nenhuma decisão tomada).

Como cada classe pode ser representada por mais de uma pirâmide, a pirâmide com o valor *match* máximo de cada classe contribui para a decisão do reconhecimento. Se uma pirâmide Pi produz um valor *match* maior que um determinado valor (de vigilância) e maior que todos os outros valores *match* produzidos por pirâmides associadas a outras classes, então o resultado será um reconhecimento correto, já que o padrão realmente pertence à classe associada a Pi.

Caso contrário, ele representará um erro. Se o padrão não puder ser designado a qualquer pirâmide, ele será rejeitado.

CARVALHO (1994) mostra diversas funções lembrança que podem ser usadas por GSN^s, além de modificações feitas no modelo básico GSN^s, as quais são descritas a seguir.

A primeira mudança é usar somente uma camada de neurônios, descartando a fase de validação e simplificando a fase de aprendizado. Com isso, somente dois valores necessitam ser armazenados nos conteúdos de memória dos neurônios. Um dos valores é usado para indicar que um conteúdo de memória foi endereçado e o outro é usado para mostrar que a localidade não foi acessada durante a fase de aprendizado.

Se a fase de validação não é utilizada e a fase de reconhecimento é executada usando somente a operação *match*, então o uso de estruturas de pirâmides nesta arquitetura torna-se desnecessário. Uma arquitetura com uma simples camada será então suficiente para desenvolver uma arquitetura Booleana auto-organizativa para reconhecimento de padrões.

A computação desta nova arquitetura GSN, denominada GSN^{sf}, pode então ser dividida em duas fases: a fase de aprendizagem e a fase de reconhecimento. Estas duas fases usam a operação *match*, sendo esta operação extremamente importante na determinação do desempenho alcançado pela rede. Somente uma camada de neurônios é utilizada, sendo os mesmos agrupados em *clusters*, ao invés de utilizar pirâmides. Um *cluster* Ci pode ser entendido como a camada de base de uma pirâmide Pi. A figura 3.14, a seguir, ilustra a arquitetura modificada.

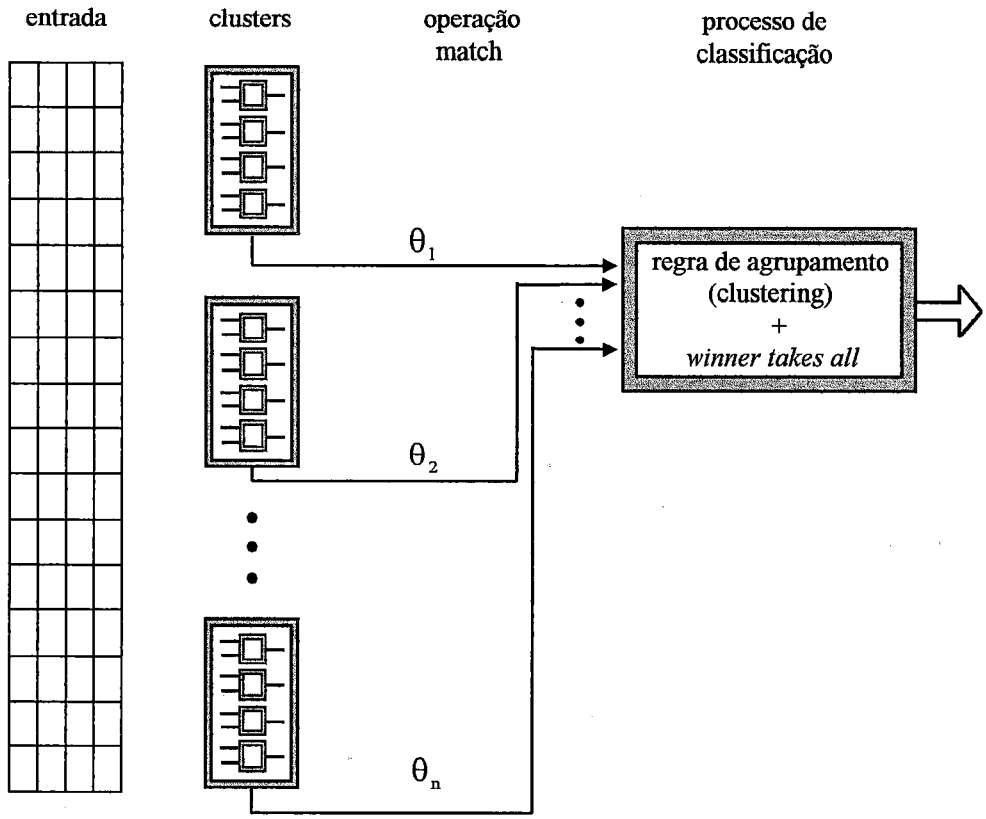


Figura 3.14. Arquitetura GSN^{sf} (CARVALHO, 1994).

CARVALHO (1994) realizou experimentos de reconhecimento de padrões, com objetivo de determinar o quanto estas modificações afetam o desempenho do GSN^s. Em tal experimento, a rede foi treinada para reconhecer números escritos à máquina (0 a 9) extraídos de códigos postais de envelopes do correio. Os resultados obtidos mostraram que o GSN^{sf} proporciona uma melhor performance com uma estrutura muito mais simples. Isto reforça o dado de que a arquitetura anterior tem problemas estruturais e de processamento.

Outras modificações também foram avaliadas por CARVALHO (1994) de tal forma que alguns refinamentos adicionais foram incorporados à estrutura básica, para aumentar a performance alcançada em tarefas práticas de classificação de padrões. O ganho obtido com cada modificação foi medido repetindo o experimento de reconhecimento com o mesmo dado. Uma arquitetura GSN^s otimizada é então apresentada. A estrutura final troca neurônios Booleanos baseados em GSN por pesos binários, os quais reduzem a memória necessária e simplificam a operação *match* utilizada.

Arquiteturra GSN Feedforward

GSN^f (FAIRHURST, 1991a) é a arquitetura baseada em GSN mais comumemte utilizada, já que supera várias limitações de outros modelos Booleanos, incorporando características como facilidade de implementação em hardware e simulação em ambiente paralelo. A arquitetura é composta por um número fixo de pirâmides, sendo que cada uma delas trabalha com um subconjunto dos dados de entrada. A escolha destes subconjuntos desempenha um papel importante na distribuição de tarefas entre as pirâmides já que uma má alocação pode resultar numa distribuição inapropriada, sobracarregando certas pirâmides e subutilizando outras (CARVALHO, 1994).

O processamento de uma rede GSN^f pode ser dividido em três fases diferentes, cada uma associada a uma meta específica:

- fase de validação;
- fase de aprendizado;
- fase de lembrança.

Na fase de validação, o objetivo é obter um valor "validado" para cada pirâmide, sendo este valor a saída natural da pirâmide para um dado padrão de entrada, definindo quais valores específicos a pirâmide pode aprender para o padrão de entrada em questão. Uma pirâmide não pode ser ensinada quando seu valor "validado" é o oposto de seu valor saída desejada. Tal situação é conhecida como "conflito de aprendizagem". Assim, para cada passo de aprendizagem, somente o subconjunto de pirâmides que não tiveram conflitos de aprendizado é treinado.

Na fase de aprendizado, as pirâmides são ensinadas através da troca dos valores armazenados nas posições de memória de seus neurônios.

O reconhecimento de padrões é realizado na fase de lembrança, a qual busca fornecer para cada neurônio um valor definido o qual tenha a maior freqüência de ocorrência nos conteúdos de memória. A propagação destes valores através das camadas das pirâmides produz seu valor de lembrança.

A figura a seguir ilustra tal arquitetura.

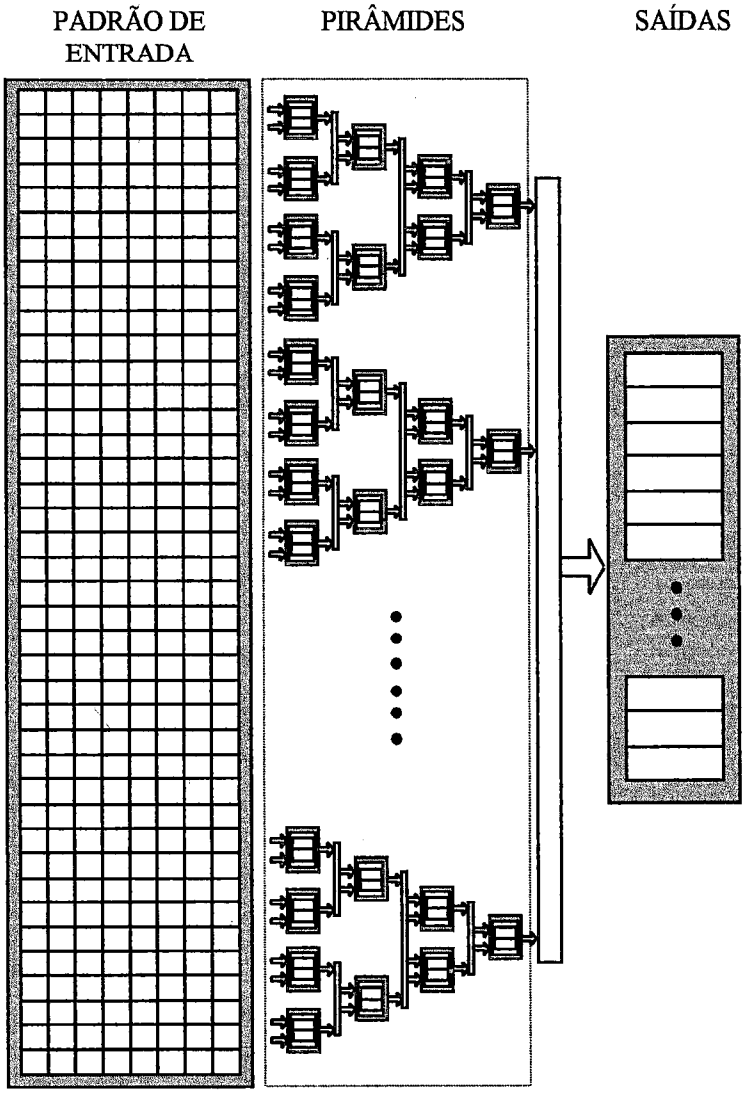


Figura 3.15. Arquitetura GSN^f

A aplicação de tal arquitetura para o reconhecimento de padrões faz uso de vetores binários. Estes vetores (de saída) são associados a cada classe do padrão de entrada, sendo definidos antes da fase de aprendizado.

A figura 3.16, a seguir, mostra a utilização da arquitetura em questão para a classificação de padrões, através do uso dos vetores.

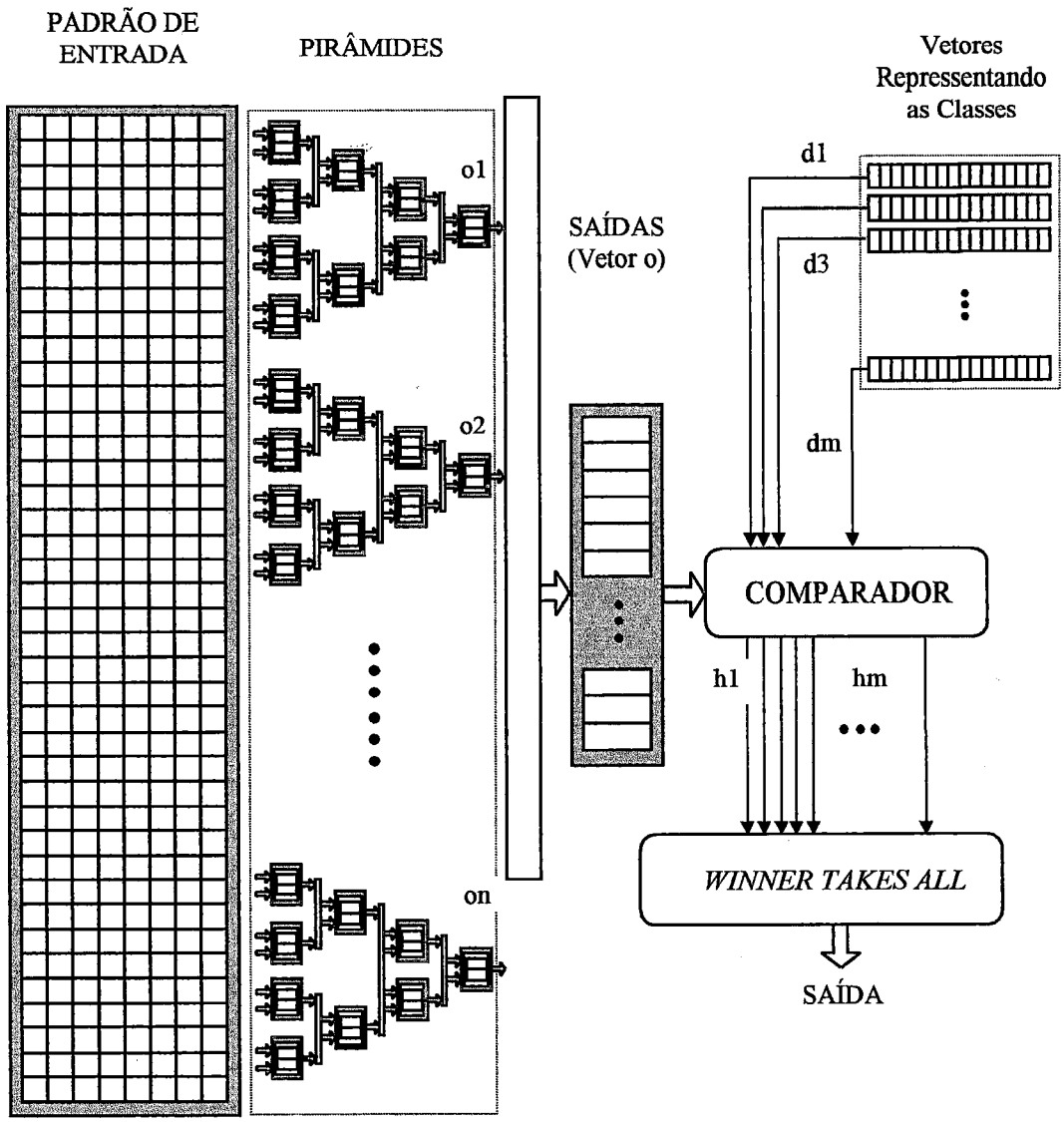


Figura 3.16. Classificação usando GSN^f.

As pirâmides GSN^f geram o vetor saída o. A saída desejada associada a uma determinada classe i é representada pelo vetor di. A medida de similaridade entre a saída desejada, di, e a saída das pirâmides (vetor o) é feita através de hi, baseado em distância Hamming. Através da utilização de uma operação do tipo *winner takes all*, a classe cujo vetor saída desejada está mais próximo do vetor de saída gerado pelas pirâmides GSN^f é selecionada.

Definição dos vetores saída desejada

A escolha dos vetores de saída desejada pode ser feita escolhendo-os **aleatoriamente**. Para tal, deve-se observar, no entanto, a distância Hamming entre quaisquer dois vetores. Esta não deve ser menor que um determinado valor, normalmente igual à metade da dimensão dos vetores (CARVALHO, 1994).

Os vetores saída desejada são utilizados para definir a quais pirâmides um determinado padrão de treinamento pode ser ensinado, conforme mostrada na equação 3.8, abaixo.

$$\text{pirâmide ensinada} \begin{cases} \text{não, se } v_{ij} = -d_{ij} \\ \text{sim, caso contrário} \end{cases} \qquad (3.8)$$

onde:

dij → representa a saída desejada para a classe i e para a pirâmide j;

vij → representa o valor "validado" para a classe i e para a pirâmide j.

Os vetores saída desejada também são utilizados para definir a qual classe pertencerá um padrão de teste apresentado, conforme mostrado na equação 3.9, a seguir.

$$\text{classe} = \begin{cases} c_j \text{ se } \forall c_i \in \{c_0, \ldots\ldots c_9\}, \sum_{k=1}^{N_p} \left| d_{jk} - v_{jk} \right| > (\sum_{k=1}^{N_p} \left| d_{ik} - v_{ik} \right| + \theta); \\ \text{nenhuma, caso contrário} \end{cases} \qquad (3.9)$$

onde:

ci → classe i

Np → representa o número de pirâmides na arquitetura

djk → resposta desejada para classe j e pirâmide k

ojk → saída gerada pela pirâmide k para classe j

θ → valor de threshold

Nesta equação, é selecionada a classe cujo vetor saída tem mais elementos em comum com a saída gerada. Ela utiliza um threshold para garantir que há uma distância mínima para o vetor saída desejada da segunda classe mais similar. Saídas indefinidas são tomadas como 0.5.

FAIRHURST et al. (1992) demonstraram a saturação em pirâmides de arquitetura GSN[f] usando neurônios GSN com dois terminais de entrada, principalmente na primeira e última camadas após a apresentação de alguns padrões do conjunto de treinamento. Demonstrou também que os primeiros padrões apresentados em cada classe tem uma maior chance de ser aprendido em relação aos últimos, evidenciando a influência da ordem de apresentação dos padrões de treinamento no resultado final obtido.

CARVALHO (1994) desenvolveu experimentos para testar o desempenho desta arquitetura no reconhecimento de padrões e os resultados que obteve mostraram um desempenho muito baixo, grande dependência da ordem de apresentação dos padrões. Observou também que o baixo desempenho de reconhecimento estava fortemente relacionado à escolha dos vetores saída desejada. Observou que quando tais vetores são definidos usando métodos mencionados anteriormente, a variabilidade do conjunto de treinamento pode determinar que a mesma pirâmide seja treinada com dois vetores diferentes para o mesmo sub-padrão. Portanto, uma má escolha destes vetores pode favorecer a ocorrência de conflitos durante a fase de aprendizado, os quais ocorrem quando o valor gerado pela fase de validação, para uma dada pirâmide, é o complemento do valor saída desejada. Como conseqüência, aquela pirâmide particular não pode aprender o mapeamento entre seu padrão de entrada e a saída desejada.

Tal problema pode ser contornado utilizando a **regra SDO** (Self-desired output), descrita por FILHO (1990). Esta técnica redefine dinamicamente os vetores saída desejada durante a fase de aprendizado baseado na estatística dos valores "validados" produzidos por cada pirâmide para os padrões pertencentes a cada classe. Assim, logo que o processo de treinamento for completado, compara-se o número de saídas desejadas 0's e 1's gerados durante as fases de validação para cada pirâmide e para cada classe, de tal maneira que os resultados desta comparação permitem que novos vetores saída desejada sejam designados para cada classe. A equação 3.10, a seguir, mostra como os novos vetores são definidos.

$$d_{ij} = \begin{cases} 0 & \text{se } N_{ij}^{0} > N_{ij}^{1} \\ 1 & \text{se } N_{ij}^{0} > N_{ij}^{1} \\ d_{ij}^{anterior} & \text{se } N_{ij}^{0} = N_{ij}^{1} \end{cases} \qquad (3.10)$$

onde :

$d_{ij} \rightarrow$ nova saída desejada para classe i e pirâmide j

$d_{ij} \rightarrow$ saída desejada anterior para classe i e pirâmide j

$N_{ij}^{x} \rightarrow$ número de vezes que a pirâmide j foi ensinada com saída desejada x ou não pôde ser ensinada com o valor -x quando os padrões pertencentes à classe i são apresentados.

Após concluída a fase de aprendizado, se, para pelo menos a metade dos padrões de entrada um valor saída desejada não foi ensinado à sua pirâmide correspondente, ele é trocado por seu complementar.

O uso de da regra SDO reduz drasticamente a influência da ordem de apresentação no desempenho de reconhecimento, já que se mais da metade dos padrões de uma classe não podem ser ensinados a uma pirâmide, a pirâmide troca sua saída desejada relacionada àquela classe. CARVALHO (1994) avaliou este desempenho através de experimentos e concluiu que o uso da regra SDO não somente reduz as variações no desempenho diferentes ordens de apresentação são usadas, mas também incrementa a performance média alcançada.

Outra técnica, **SDO ponderada**, leva em conta uma medida mais precisa de cada saída desejada "preferida" da pirâmide para cada classe. Isto é feito tomando a proporção de saídas 1 desejadas para cada pirâmide e para cada classe, ao invés de somente o valor saída desejada mais freqüente. A equação 3.11, a seguir, ilustra esta técnica.

$$d_{ij}^{w} = \frac{N_{ij}^{1}}{N_{ij}^{1} + N_{ij}^{0}} \qquad (3.11)$$

onde:

$d_{ij}^{w} \rightarrow$ saída desejada ponderada para classe i e pirâmide j

$N_{ij}^{x} \rightarrow$ n.o de vezes que a pirâmide foi ensinada com saída desejada x ou não pôde ser ensinada com o valor -x quando padrões pertencentes à classe i são apresentados.

As saídas desejadas são elementos binários pequenos, possibilitando que operações mais complexas sejam usadas para classificar padrões. Esta medida mais precisa das estatísticas das saídas desejadas é utilizada para melhorar as taxas de reconhecimento correto.

O efeito do uso desta técnica no reconhecimento de padrões também foi avaliado por CARVALHO (1994). Ele concluiu que uma visível melhoria nas taxas de reconhecimento correto são alcançadas usando a técnica SDO ponderada para definir vetores saída desejada. Os resultados também mostraram um pequeno incremento na diferença das taxas de reconhecimento quando diferentes ordens de apresentação foram utilizadas. Ele também propôs uma técnica a qual realiza uma atualização contínua dos vetores saída desejada durante a fase de treinamento. Como conseqüência, cada vez que um padrão novo é aprendido, o vetor saída desejada relacionado a esta classe é redefinido. Os resultados de seus experimentos mostraram que melhorias adicionais no desempenho podem alcançados trocando dinamicamente os vetores saída desejada durante a fase de aprendizado.

Outro aspecto importante na construção de uma arquitetura baseada em GSN é definição de como os terminais de seus neurônios serão conectados ao padrão de entrada. A escolha deste mapeamento definirá quais funções poderão ser aprendidas por cada pirâmide. Este mapeamento é particularmente importante quando se utiliza arquiteturas GSN feedforward, já que cada pirâmide está relacionada somente a uma sub-área da imagem de entrada.

Vários experimentos vem sendo realizados para avaliar a influência de diferentes mapeamentos entre os neurônio na primeira camada e o padrão de entrada na performance alcançada pelas arquiteturas GSN feedforward, os quais podem ser divididos em dois grupos:

- Mapeamento Contínuo;
- Mapeamento Aleatório.

O Mapeamento Contínuo divide a área de entrada em N blocos de igual tamanho, um bloco para cada pirâmide, onde um bloco representa um conjunto de pixels cobrindo uma sub-área contínua da entrada. No Mapeamento Aleatório, os terminais de entrada de cada pirâmide são conectados aleatoriamente a qualquer pixel da imagem de entrada (CARVALHO, 1994). Estes mapeamentos são ilustrados nas figuras 3.17 e 3.18, a seguir.

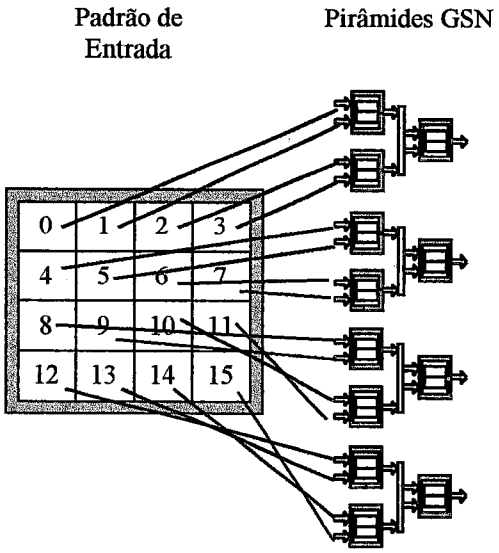


Figura 3.17. Mapeamento contínuo.

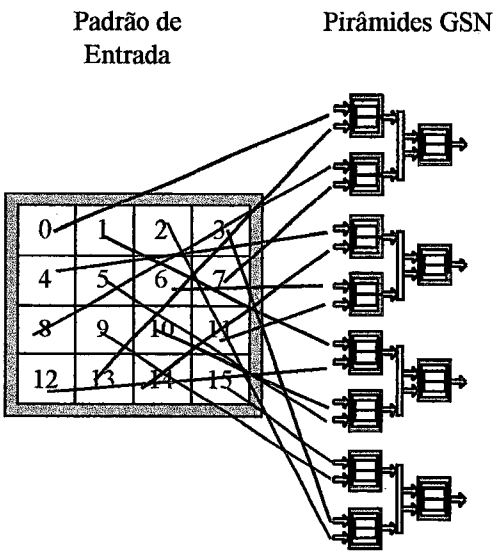


Figura 3.18. Mapeamento aleatório.

Estudos experimentais mostram que o uso do mapeamento aleatório melhora o desempenho de reconhecimento.

3.3.3 Estratégias de aprendizado para GSN^f

O desempenho alcançado por Redes Neurais está diretamente relacionado ao algoritmo de aprendizado utilizado, ou seja, a performance de reconhecimento de uma rede neural é uma conseqüência direta do desenvolvimento correto de uma função que implemente eficientemente o mapeamento entre o conjunto de padrões de treinamento e seus vetores saída associados. Uma má estratégia torna isto muito difícil, senão impossível. A escolha da estratégia apropriada pode depender dos requisitos de aprendizado do classificador. Estes requisitos podem ser, por exemplo, velocidade, capacidade de aprendizado, saturação e/ou performance de reconhecimento.

Algoritmos de aprendizado para redes Booleanas trabalham basicamente trocando as conexões da rede (PATARNELLO & CARNEVALI, 1987, DOYLE, 1990), a função Booleana realizada pelos nós (KANERVA, 1988) ou os valores armazenados nos conteúdos de memória (BLEDSOE & BROWNING, 1966, ALEKSANDER, 1967, ARMSTRONG & GECSEI, 1979, KAN & ALEKSANDER, 1987, MYERS & ALEKSANDER, 1988, BRADY et al., 1989).

Os algoritmos de aprendizado utilizados por GSN^f estão relacionados com a questão de quando e onde a saída desejada do neurônio (0 ou 1) deve ser armazenada na(s) localidade(s) endereçada(s) por seus terminais. Antes da fase de aprendizado, um valor "validado" é gerado para cada neurônio através de uma operação de validação. O valor "validado" do neurônio no ápice de uma pirâmide define qual valor a pirâmide pode aprender na fase de aprendizado subsequente. Para usar estas células mais eficientemente, o neurônio GSN^f busca armazenar a saída desejada em uma posição de memória a qual já armazene este valor. Quando isto não é possível, uma localidade de memória a qual armazena um valor indefinido é selecionada.

Duas estratégias principais foram desenvolvidas para ensinar Redes Neurais GSN^f, as quais utilizam diferentes métodos para selecionar o conteúdo de memória que irá armazenar o valor saída desejada quando existir mais do que uma opção. O primeiro método escolhe a posição de memória aleatoriamente e o segundo escolhe uma posição de memória cujo endereço tem o maior número de bits em comum com o valor definido a ser aprendido.

Usando estas estratégias de aprendizado, GSN^f tem alcançado um bom desempenho em tarefas de reconhecimento. Sua característica *one-shot* de aprendizado a faz particularmente apropriada para aplicações de reconhecimento de padrões que necessitam de aprendizado rápido.

A escolha do conteúdo de memória na primeira estratégia é mostrado através da equação 3.12, abaixo.

$$a_{im} = \begin{cases} Ran(A_{i/di}) \text{ se } \|A_{i/di}\| > 0; \\ Ran(A_{i/u}) \text{ se } \|A_{i/di}\| = 0. \end{cases} \quad (3.12)$$

onde:

$a_{im} \rightarrow$ endereco do conteú dode memória selecionado

$A_{i/v} = \{a_{ik} \in Ai | C_i[a_{ik}] = v\} \rightarrow$ representa o conjunto enderecavel do neurônio n_i que armazena o valor v.

$Ran(A_{i/v}) \rightarrow$ elemento aleatoriamente escolhido de $A_{i/v}$

$\|A_{i/v}\| \rightarrow$ nú mero de elementos que pertencem a $A_{i/v}$

Através deste algoritmo, o neurônio busca por uma célula que já armazene este valor de saída desejado. Se não há tal célula, o neurônio escolhe uma célula com um valor indefinido. Se, em qualquer dos dois casos mencionados, há mais do que uma escolha possível, a célula é escolhida aleatoriamente. Depois de escolhida a célula, o endereço desta célula deve ser enviado de volta, através dos terminais de estrada desejada, para ser usado como saída desejada pelos neurônios na camada anterior.

Na segunda abordagem, busca-se manter o mesmo valor em localidades de memória com endereços similares e valores opostos em localidades de memória cujos endereços são muito diferentes. Dois endereços são similares quando têm uma pequena distância Hamming (menor que a metade do número de terminais de entrada) e vice versa. A equação 3.13, a seguir, mostra a seleção do endereço.

$$a_{im} = \begin{cases} a_{ij} \in A_{i/di} \left| H_{aij}^{Di} = min(H_{aik}^{Di}), \forall a_{ik} \in A_{i/di}, se \|A_{i/di}\| > 0 \right. \\ a_{ij} \in A_{i/u} \left| H_{aij}^{Di} = min(H_{aik}^{Di}), \forall a_{ik} \in A_{i/di}, se \|A_{i/u}\| = 0 \right. \end{cases} \quad (3.13)$$

$D_i \rightarrow$representa uma matriz da mesma dimensão de aij onde todos seus componentes são iguais a di;

$H_{aij}^{Di} \rightarrow$ distância Hamming entre Di e aij

O endereço aij é aquele com a menor distância Hamming a Di

Esta equação difere da equação 3.12 somente no caso onde há mais do que um conteúdo de memória endereçado, já que o conteúdo de memória selecionado é aquele cujo endereço tem mais bits em comum com a saída desejada.

O objetivo deste algoritmo é tornar as representações internas de padrões da mesma classe tão similares quanto possível e vice-versa, aumentando a discriminação entre padrões de classes diferentes e reduzindo a discriminação entre membros da mesma classe (BOWMAKER & COGHILL, 1992).

Uma análise destes algoritmos de aprendizado, empregados por GSN, tem mostrado que eles ensinam à rede mais do que é necessário. Este ensino excessivo acontece por dois motivos: primeiro, os conteúdos de memória podem ser ensinados com valores definidos que já estão armazenados lá, ou seja, armazenam um valor definido em um conteúdo de memória de cada neurônio na pirâmide que estão ensinando, mesmo que estes conteúdos de memória já armazenem o mesmo valor definido e, segundo, conteúdos de memória armazenando valores indefinidos são desnecessariamente atualizados quando poderiam ser mantidos inalterados para necessidades futuras.

CARVALHO (1994) propôs algoritmos alternativos de aprendizado para GSN^f, os quais são apresentados a seguir.

O algoritmo **Lazy** emprega uma técnica onde um neurônio somente aprende quando estritamente necessário, reduzindo ao mínimo o armazenamento de valores definidos nos conteúdos de memória, diminuindo a saturação, tornando menos provável os conflitos de aprendizado e acelerando o processo de aprendizado. Estas melhorias são alcançadas evitando ensinar um neurônio cada vez que sua saída desejada for igual ao seu valor "validado" ou o neurônio para o qual sua saída é conectada não foi ensinado. Assim, se um neurônio no ápice de uma pirâmide não é ensinado, a pirâmide total não é ensinada.

De fato, esta classe de aprendizado é desnecessária já que, se o valor "validado" vij gerado por um dado neurônio nij em uma pirâmide Pi é um valor definido, então todos os neurônios nik na sub-pirâmide Pij (com o neurônio nij no ápice) cujo valor validado é indefinido, podem ser ensinados com qualquer valor definido. O valor definido escolhido não causa qualquer alteração para o valor que deve ser gerado para o mesmo padrão de entrada na fase de lembrança, já que, se um valor definido é gerado para um padrão na fase de validação, devido à natureza da função de lembrança, ele necessariamente será gerado pelo mesmo padrão na fase de lembrança. Assim, o aprendizado **Lazy** ignora estes neurônios e permite que sejam atualizados por outros padrões que realmente necessitam atualizá-los.

O **aprendizado progressivo** treina as pirâmides numa direção contrária aquela utilizada pelos algoritmos anteriores. É um algoritmo que proporciona maior velocidade ao processo de aprendizado para arquiteturas GSN^f, eliminando uma das operações utilizadas pelos algoritmos citados anteriormente, já que somente um e não um conjunto de conteúdos de memória é endereçado em cada nó durante a fase de aprendizado. Este algoritmo inicia o treinamento com os neurônios na base e, progressivamente, ensina todo os neurônios em cada camada, até o ápice.

Durante o treinamento da primeira camada de uma dada pirâmide, todas as posições de memória desta camada as quais são endereçadas pelo padrão de entrada e armazenam valores indefinidos tem seu valor trocado para o valor saída desejada associado à pirâmide. Para camadas sucessivas, as saídas das camadas anteriores produzem as entradas para as próximas.

Este algoritmo permite que valores definidos armazenados em conteúdos de memória sejam alterados sem que haja uma grande perda de informação.

CARVALHO (1994) realizou estudos experimentais comparando as diversas abordagens para o aprendizado em GSN^f, levando em conta diversos parâmetros como saturação de memória, conflitos de aprendizagem taxa de atualização de memória, tempo e desempenho de treinamento. Após diversos testes, concluiu que a melhor performance global pode ser obtida utilizando a estratégia de aprendizado progressivo: maiores taxas de saturação e menores taxas de conflito de aprendizado indicam que mais padrões são aprendidos por GSN^f utilizando esta estratégia.

MARTINS (1994) apresentou duas melhorias em classificadores baseados em redes feed-forward baseadas em GSN e ALN, explorando novas estratégias para o treinamento e avaliação de tais classificadores. Ele utilizou conceitos de Algoritmos Genéticos em um sistema hierárquico para gerar topologias iniciais melhoradas com uma nova família de algoritmos para o treinamento de GSN, nos quais um controle dinâmico da ordem de apresentação dos exemplos para a rede GSN é proposto. Resultados experimentais mostraram a viabilidade dos novos algoritmos. Os principais conceitos utilizados neste trabalho são mostrados no item a seguir.

CARVALHO et al. (1997) apresentaram estratégias distintas de aprendizado para as arquiteturas baseadas em GSN^f. Resultados experimentais foram apresentados, através da avaliação de tempo de treinamento, taxas de saturação, além de desempenho de reconhecimento.

BOTELHO et al. (1996) aplicaram o modelo GSN de Rede Neural para o controle de navegação de Robô, através da implementação de funções lógicas e aritméticas (NOT, AND, OR, Adição, etc.) Os resultados obtidos foram comparados com outros modelos de Rede Neural.

3.4 ALGORITMOS GENÉTICOS E GSN

3.4.1 Introdução

Em biologia, adaptação designa qualquer processo através do qual uma estrutura é progressivamente modificada para obter um melhor desempenho no seu ambiente. Neste contexto, os processos adaptativos são processos de otimização. Os Algoritmos Genéticos são definidos como algoritmos de busca que se sustentam em mecanismos da genética natural e da seleção natural. São também considerados como métodos robustos, porque o espaço de soluções é irrestrito e porque apresentam bom desempenho mesmo em uma maior diversidade de problemas.

Os cromossomos (indivíduos), em termos de Algoritmos Genéticos, constituem um dos elementos da evolução natural, sendo que seus processos de codificação não são totalmente conhecidos. No entanto, algumas das características da Teoria da evolução são completamente aceitas, tais como:

- evolução é um processo que ocorre nos cromossomos;
- seleção natural é a ligação entre os cromossomos e o desempenho de suas estruturas decodificadas; este processo propicia a reprodução das melhores estruturas. A evolução ocorre no processo de reprodução, o que significa que processos de mutações e de recombinação podem causar diferenças entre os cromossomos dos descendentes e dos pais;
- evolução biológica não tem memória, ou seja toda evolução parte dos elementos codificados nos cromossomos.

Toda tarefa de busca e otimização possui vários componentes, entre eles o espaço de busca, onde são consideradas todas as possibilidades de solução de um determinado problema e a função de avaliação (ou função de custo), uma maneira de avaliar os membros do espaço de busca. Existem diversos métodos de busca e funções de avaliação (DAVIS, 1991).

As técnicas de busca e otimização tradicionais iniciam-se com um único candidato que, iterativamente, é manipulado utilizando algumas heurísticas (estáticas) diretamente associadas ao problema a ser solucionado. Geralmente estes processos heurísticos não são algorítmicos e sua simulação em computadores pode ser muito complexa. Apesar destes métodos não serem suficientemente robustos, são utilizados amplamente em inúmeras aplicações.

Por outro lado, as técnicas de computação evolucionária operam sobre uma população de candidatos, podendo realizar a busca em diferentes áreas do espaço de solução, alocando um número de membros apropriado ara a busca em várias regiões.

Os Algoritmos Genéticos diferem dos métodos tradicionais de busca e otimização, principalmente, em quatro aspectos:

1. Trabalham com uma codificação do conjunto de parâmetros e não com os próprios parâmetros;
2. Trabalham com uma população e não com um único ponto;
3. Utilizam informações de custo ou recompensa e não derivadas ou outro tipo de conhecimento auxiliar;
4. Utilizam regras de transição probabilísticas e não determinísticas.

Algoritmos Genéticos são muito eficientes para busca de soluções ótimas, ou aproximadamente ótimas em uma grande variedade de problemas, pois não impõem muitas das limitações encontradas nos métodos de busca tradicionais. Por serem baseados na evolução biológica, são capazes de identificar e explorar fatores ambientais e convergir para soluções ótimas, ou aproximadamente ótimas, em níveis globais.

"Quanto melhor um indivíduo se adaptar ao seu meio ambiente, maior será sua chance de sobreviver e gerar descendentes" - este é o conceito básico da evolução genética biológica. A área biológica mais proximamente ligada aos Algoritmos Genéticos é a Genética Populacional.

3.4.2 Características Gerais

Algoritmos Genéticos são algoritmos de otimização global, baseados nos mecanismos de seleção natural e da genética. Eles empregam uma estratégia de busca paralela e estruturada, porém aleatória, a qual é voltada em direção ao reforço da busca de pontos de *alta aptidão*, ou seja, pontos nos quais a função a ser minimizada (ou maximizada) tem valores relativamente baixos (ou altos). Apesar de aleatórios eles não são *caminhadas aleatórias não direcionadas*, pois exploram informações históricas para encontrar novos pontos de busca. Isto é feito através de processos iterativos, onde cada iteração é denominada geração.

Durante cada iteração, os princípios de seleção e reprodução são aplicados a uma população de candidatos que pode variar, dependendo da complexidade do problema e dos recursos computacionais disponíveis. Através da seleção, determina-se quais indivíduos conseguirão se reproduzir, gerando um número determinado de descendentes para a próxima geração, com uma probabilidade determinada pelo seu índice de aptidão, ou seja, os indivíduos com maior adaptação relativa tem maiores chances de se reproduzir.

O ponto de partida para a utilização de Algoritmos Genéticos como ferramenta para solução de problemas é a representação destes problemas de maneira tal que os Algoritmos Genéticos possam trabalhar adequadamente sobre eles. A maioria das representações são genotípicas, utilizando vetores de tamanho finito em um alfabeto finito.

Tradicionalmente, os indivíduos são representados genotipicamente por vetores binários, onde cada elemento de um vetor denota a presença (1) ou ausência (0) de uma determinada característica: o seu genótipo. Os elementos podem ser combinados formando as características reais do indivíduo, ou seja, seu fenótipo. Teoricamente, esta representação é independente do problema, pois uma vez encontrada a representação em vetores binários, as operações padrão podem ser utilizadas, facilitando o seu emprego em diferentes classes de problemas.

A utilização de representações em níveis de abstração mais altos tem sido investigada. Como tais representações são mais fenotípicas, facilitariam sua utilização em determinados ambientes, onde essa transformação *fenótipo-genótipo* é muito mais complexa. Neste caso, precisam ser criados os operadores específicos para utilizar estas representações.

O princípio básico do funcionamento dos Algoritmos Genéticos é que um critério de seleção vai fazer com que, depois de muitas gerações, o conjunto inicial de indivíduos gere indivíduos mais aptos. A maioria dos métodos de seleção são projetados para escolher preferencialmente indivíduos com maiores notas de aptidão, embora não exclusivamente, a fim de manter a diversidade da população.

Um conjunto de operações é necessário para que, dada uma população, se consiga gerar populações sucessivas que (espera-se) melhorem sua aptidão com o tempo. Tais operadores são: cruzamento (crossover) e mutação. Eles são utilizados para assegurar que a nova geração seja totalmente nova, mas possua, de alguma forma, características de seus pais, ou seja, a população se diversifica e mantém características de adaptação adquiridas pelas gerações anteriores.

Para garantir que os melhores indivíduos não desapareçam da população pela manipulação dos operadores genéticos, eles podem ser automaticamente colocados na próxima geração, através da reprodução elitista.

Este ciclo é repetido um determinado número de vezes. Abaixo é mostrado um exemplo de algoritmo genético. Durante este processo, os melhores indivíduos, assim como alguns dados estatísticos, podem ser coletados e armazenados para avaliação.

```
Procedimento AG
        {t=0;
        inicia_população (P,t)
        avaliação (P,t);
        repita até (t=d)
                {t=t+1;
                seleção_dos_pais (P,t);
                recombinação (P,t);
                mutação (P,t);
                avaliação (P,t);
                sobrevivem (P,t);
                }

        }
```

sendo:

t - tempo atual;
d - tempo determinado para finalizar o algoritmo;
P - população.

Tais algoritmos, embora computacionalmente muito simples, são extremamente poderosos.

3.4.3 Operadores Básicos dos Algoritmos Genéticos

O processo de reprodução permite definir quais cromossomos irão permanecer na geração de novos cromossomos. A seleção dos pais, por outro lado, tem por objetivo proporcionar maiores chances de reprodução aos indivíduos ou cromossomos que têm uma função aptidão com uma melhor adaptação. Uma técnica comumente utilizada é a seleção dos pais através da "roleta", na qual cada indivíduo da população tem um número de posições proporcional à sua adaptação.

Um Algoritmo Genético simples, capaz de produzir bons resultados um muitos problemas práticos, é composto por três operadores básicos: reprodução, cruzamento e mutação (MICHALEWICZ, 1992).

Operador de Reprodução

A reprodução é a substituição dos cromossomos de uma população velha por uma população nova. Entre as técnicas de reprodução podem ser citadas:

- reprodução tradicional;
- reprodução com elitismos;
- reprodução em estado estacionário;
- reprodução em estado estacionário sem duplicação.

A reprodução tradicional é um processo de substituição de todos os indivíduos da população velha pelos seus descendentes. A seleção dos pais é feita para proporcionar maiores chances de reprodução aos indivíduos ou cromossomos que têm uma função aptidão com uma melhor adaptação.

Operador de Cruzamento

O operador cruzamento trabalha em combinação com a reprodução, ou seja, inicialmente são selecionados os pais, passando então o par de cromossomos por um processo de cruzamento, gerando, dessa forma, dois novos cromossomos. Existem vários tipos de cruzamento: cruzamento simples, cruzamento duplo, cruzamento uniforme, entre outros. O cruzamento é o operador responsável pela recombinação de características dos pais durante a reprodução, permitindo que as próximas gerações herdem essas características. É considerado o operador genético predominante. Abaixo é mostrado, esquematicamente, o processo de cruzamento.

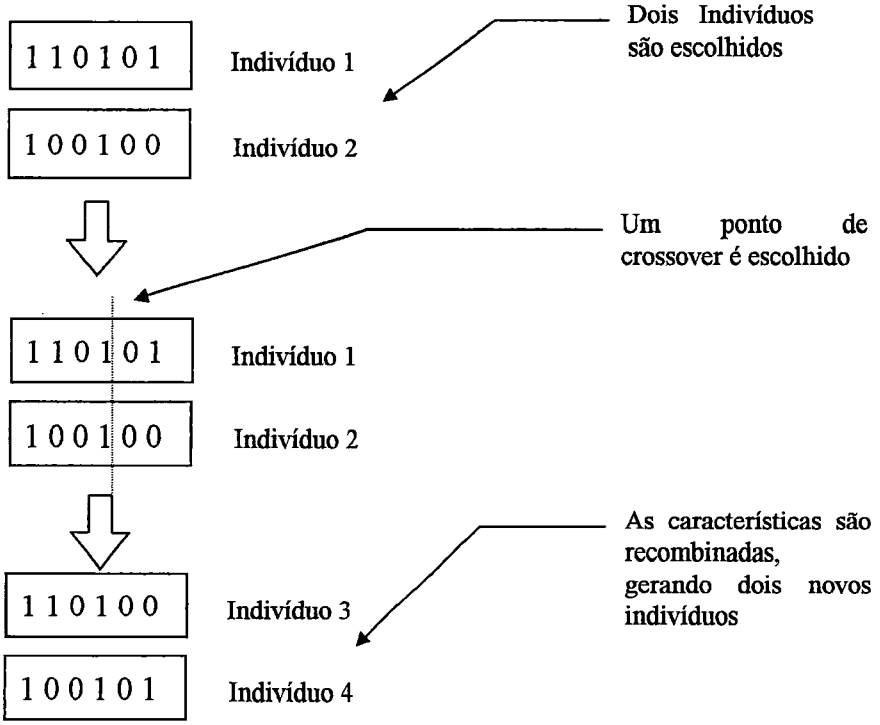


Figura 3.19 - Cruzamento (Crossover).

Operador Mutação

O operador mutação é utilizado na fase reprodução-cruzamento e é um processo que altera informações codificadas, através da mudança aleatória ou ocasional de caracteres do cromossomo. Ele é necessário para a introdução e manutençao da diversidade genética da população, alterando arbitrariamente um ou mais componentes de uma estrutura escolhida, fornecendo meios para a introdução de novos elementos na população. Assim, a mutação assegura que a probabilidade de se chegar a qualquer ponto do espaço de busca nunca será zero, além de contornar o problema de mínimos locais.

O operador de mutação é aplicado aos indivíduos com uma probabilidade dada pela taxa de mutação. Abaixo é mostrado um procedimento de mutação.

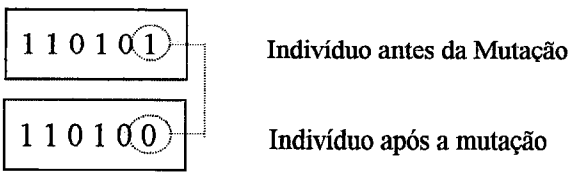


Figura 3.20. Mutação.

Os conceitos e definições apresentadas anteriormente são melhor explorados através do Teorema Geral dos Algoritmos Genéticos, mostrado no APÊNDICE I.

3.4.4 Algoritmos Genéticos e Redes Neurais

Dois pontos são básicos em Algoritmos Genéticos: a representação de possíveis soluções em uma *string* de dígitos binários - o *aspecto de codificação* - e um algoritmo para avaliar quão boa é uma possível solução - *o aspecto de avaliação*. Tais aspectos são representados na figura 3.21, a seguir.

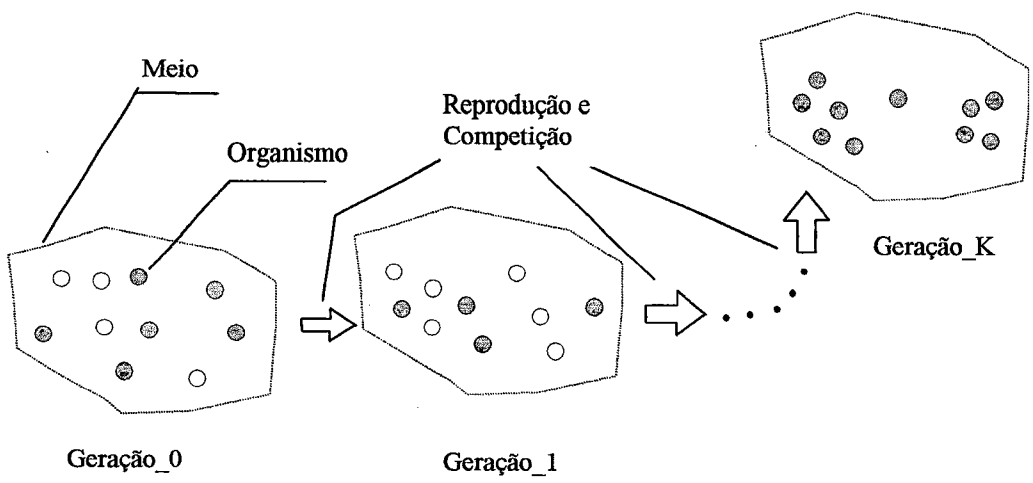


Figura 3.21. O mecanismo de evolução modelado pelos Algoritmos Genéticos (MARTINS, 1994)

Usando analogia biológica, strings são cromossomos e cada dígito binário é um gene. Organismos cujos cromossomos são responsíveis por seu alto grau de aptidão tem mais chance de se reproduzir e passam suas qualidades às suas próximas gerações. Quando a reprodução acontece, material genético é combinado de tal forma que os novos organismos refletem características de seus pais. Algum tipo de controle deve ser realizado para garantir que a probabilidade de reprodução seja diretamente relacionada à aptidão.

Nos problemas de otimização, a codificação ajuda a criar a geração inicial randomicamente ou utilizando alguma heurística. Esta geração é avaliada e permite a reprodução levando em conta as aptidões individuais. A implementação tradicional, abordagem por *geração*, usa a nova geração para substituir a antiga. Na abordagem por estado estacionário, novas gerações competem (por espaço) desde que o número de organismos seja mantido dentro de algum intervalo predefinido. A evolução da população no tempo levará a organismos altamente adaptados ao meio, ou seja, ao problema que deve ser resolvido.

No caso de busca de boas topologias para redes GSN, cada organismo representa uma escolha específica. A medida natural de aptidão para uma rede é sua performance no conjunto de treinamento. Esta medida pode ser acentuada incluindo aspectos como grau de saturação, ou seja, quantas posições internas foram ocupadas, e construir redes que correspondam a algum conhecimento de alto nível. Para tarefas de reconhecimento de imagens, por exemplo, é possível controlar as características do campo receptivo.

Embora GSN possa ser treinado muito rapidamente devido ao algoritmo *one-shot learning*, a aplicação de Algoritmos Genéticos em sua forma usual (como descrito acima) consumirá muito tempo computacional. Além disso, a medida de aptidão deve equalizar o efeito de diferentes topologias, e o treinamento (validação e aprendizado) deve ser realizado quando um novo organismo nascer. Estes problemas são resolvidos quando um sistema hierárquico é utilizado. Um sistema deste tipo foi proposto, implementado e avaliado por MARTINS (1994) e será apresentado a seguir.

Algoritmos para otimização de redes iniciais

MARTINS (1994) implementou uma estratégia para superar o problema da escolha das redes iniciais para classificadores baseados em GSN. Como pirâmides são utilizadas (pretendidas), um sistema hierárquico foi criado, conforme mostrado na figura 3.22, a seguir.

Cada nível é um meio diferente onde uma simples busca genética ocorre. A estrutura deste sistema tem entradas externas no nível mais baixo (nível_0) e usa nós de um nível para construir nós do próximo nível. Como os nós do nível n identificam pirâmides de 2^n entradas, isto implica que pirâmides de 2^n entradas são combinadas para construir pirâmides de 2^{n+1} entradas. Mais precisamente, nós são criados combinando nós do nível inferior ou permutando sub-árvores (irmãos) entre nós do mesmo nível. O primeiro mecanismo, *resampling*, implementa uma busca *bottonn-up* e o segundo, *recombinação*, atua na direção oposta (*top-down*).

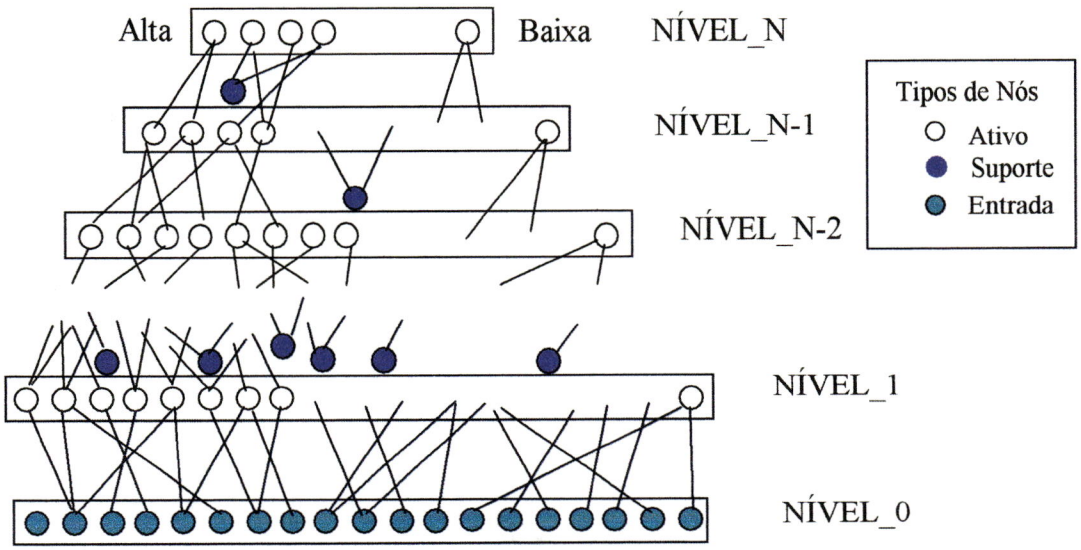


Figura 3.22. Arquitetura do Sistema Hierárquico de busca genética (MARTINS, 1994)

Em cada nível, há pirâmides com o mesmo número de entradas. Mais precisamente, no nível n há pirâmides de 2^n entradas. Depois da recombinação e *resampling*, a competição ocorre dentro de todos os níveis (exceto o nível 0). O sistema irá, portanto, conduzir buscas concorrentes independentes dentro de cada nível. Em termos de classificador global (para problemas multi-classes), como cada bit posicional dos códigos de saída impõem um mapeamento diferente, um sistema hierárquico separado é fornecido para cada bit posicional.

Existem três tipos de nós no sistema: nós de entrada, nós de suporte e nós ativos. O nível mais baixo, nível_0, é composto de nós de entrada, ou seja, nós que transmitem as entradas externas. Nós ativos são aqueles que pertencem a qualquer outro nível. Seus valores de aptidão são altos o suficiente para garantir sua sobrevivência. Nós são de suporte se sua sobrevivência é derivada da sobrevivência de nós de níveis superiores.

Para inicializar o sistema, uma primeira geração de nós é criada de baixo para cima, já que somente o conjunto de treinamento está disponível. As entradas externas são avaliadas para cada bit posicional do código de saída para gerar seus valores de aptidão. Depois que o nível mais baixo é calculado, ele torna-se permanente. Organismos do nível 1 podem, então, ser criados por amostragem. Esta estratégia recursiva repete-se até que todos os organismos sejam criados.

O conjunto de treinamento é analisado para cada variável de entrada e bit posicional dos códigos de saída. Basicamente, esta análise pune (dá menores valores de aptidão) entradas que permanecem constantes mais do que uma vez e gratifica (recompensa) aquelas cujos valores estão fortemente relacionados com a saída desejada.

A expressão matemática usada para calcular a aptidão de entrada para cada bit posicional do código de saída é dada a seguir.

$$f_{ib} = v_i Max[Sim(b_i, d_b), Sim(b_{\bar{i}}, d_b)] \qquad (3.14)$$

onde:

f_{ib} é a aptidão da i-ésima entrada para o bit posicional b do código de saída;

v_i é a medida da variabilidade da i-ésima entrada definida por (3.15);

b_i é o comportamento da i-ésima entrada para o conjunto de treinamento;

d_b é a saída desejada para os exemplos no conjunto de treinamento;

Sim(x,y) é a similaridade entre os vetores x e y;

$\bar{i}$ é a i-ésima entrada na forma complementar.

$$v_i = \begin{cases} \frac{2}{NTrain} \cdot \sum_{j=1}^{NTrain/8} Ones[b_{ij}] \text{ se } \sum_{j=1}^{NTrain/8} Ones[b_{ij}] \leq \frac{NTrain}{2} \\ \frac{2}{NTrain} \cdot \sum_{j=1}^{NTrain/8} Ones[b_{ij}] - 2 \text{ se } \sum_{j=1}^{NTrain/8} Ones[b_{ij}] \geq \frac{NTrain}{2} \end{cases} \quad (3.15)$$

onde:

Ones é uma tabela que retorna quantos bits "1" existem em um dado byte;
NTrain é o número de exemplos no conjunto de treinamento;
b_{ij} é o j-ésimo byte da i-ésima entrada comportamento para oito exemplos.

Estas expressões podem ser implementadas em procedimentos rápidos, já que os cálculos podem ser reduzidos a simples comparações de bits.

Uma vez completado o nível mais baixo, o próximo pode ser criado por amostragem. Um caminho prático para seguir a distribuição dinâmica de aptidão de um nível inferior é gerar uniformemente números randômicos, *rand*, de zero a $\sum_{i=1}^{Numero\_de\_Entradas} f_{ib}$. Então, uma busca seqüencial escolhe a mais baixa j-ésima entrada que satisfaça $rand < \sum_{i=1}^{j} f_{ib}$. A mesma estratégia é aplicada para construir sucessivamente os níveis mais altos. Como esta é a população inicial, não há competição. Além disso, nenhuma medida de variabilidade é usada no cálculo da aptidão de um nó. A aptidão de um nó é definida pela equação 3.16, a seguir.

$$f_{kb} = Sim(b_k, d_b) \quad (3.16)$$

onde:

f_{kb} é a aptidão do i-ésimo nó para o bit posicional b do código de saída;
b_k é o comportamento do nó para todos os exemplos no conjunto de treinamento;
d_b são as respostas desejadas para o bit posicional b do código de saída;
Sim é o produto escalar entre os vetores b_k e d_b.

Para evitar a necessidade de re-treinamento de todos os nós da pirâmide quando um novo é criado, uma correlação direta entre a resposta de um nó e a saída desejada é imposta do nível_1 para cima. O vetor comportamento de um nó, b_k, no qual cada bit é uma resposta para um exemplo no conjunto de treinamento, é determinado combinando os mesmos vetores de suas sub-árvores.

Este procedimento é rápido, já que é local e pode ser implementado por cálculo de bit. Além disso, somente no nível 1, o nível que combina entradas externas, o comportamento a priori é totalmente desconhecido. Nos outros níveis, é certo que a saída segue as entradas quando elas são idênticas. Para nós do nível_1, todas as quatro possibilidades "00", "01", "10" e "11" tem que ser analisadas. O procedimento adotado aqui designa dois contadores para cada situação, C0 e C1, para tratar com situações onde a saída desejada é "0" ou "1". Se a diferença entre C0 e C1 é menor que um determinado threshold, *MinDif*, o valor indefinido é armazenado. Se a diferença está acima de *MinDif*, o nó assume o comportamento ditado pelo maior valor.

O controle de situações no comportamento indefinido pela introdução de threshold *MinDif* pode ser entendido como a ferramenta de projeto para dizer o quanto é desejável que o sistema classificador siga o conjunto de treinamento. Se *MinDif* é zero, o comportamento do nó pode ser determinado pela existência de um simples exemplo. Se *MinDif* é muito grande, a porcentagem de posições internas indefinidas será muito alta.

Na implementação do sistema, o algoritmo *one-shot* de treinamento de GSN foi modificado para um de incremento local por lotes, já que este possibilita o uso de buscas genéticas em níveis independentes que não são expensivas em termos computacionais. Esse algoritmo remove a escolha aleatória quando uma posição interna indefinida deve ser escolhida. Sua estratégia é tomar a posição interna mais próxima de outra que armazene a saída final.

Depois que a primeira geração é criada, a evolução começa por aplicações alternadas de recombinação (busca *top-down*) e *resampling* (busca *botton-up*). Apesar da competição manter o número de organismos sob controle, o processo de *resampling* é idêntico à formação da população inicial, já que segue a mesma estratégia na criação de novos organismos. A competição é implementada testando a aptidão de novos organismos depois de sua criação. Se sua aptidão é maior ou igual à do nó de aptidão mínima de seu nível, este é morto e o formador é incluído. Esta estratégia é particularmente conveniente onde faz-se necessário economia de memória.

Para recombinar organismos de um nível e gerar outros novos que competirão por sobrevivência no mesmo nível (figura 3.23) a distribuição de valores de aptidão do nível atual é utilizada para escolher parentes. Uma vez que um parente é escolhido, uma das sub-árvores deve ser selecionada. Diversos procedimentos são possíveis, mas o presente algoritmo simplesmente faz esta escolha de forma randômica.

Neste sistema, nenhum mecanismo de mutação é implementado, fazendo com que o processo possa convergir mais rapidamente.

As condições para terminar o processo podem ser escolhidas de várias maneiras possíveis. As duas mais simples são diretamente relacionadas ao tempo e performance.

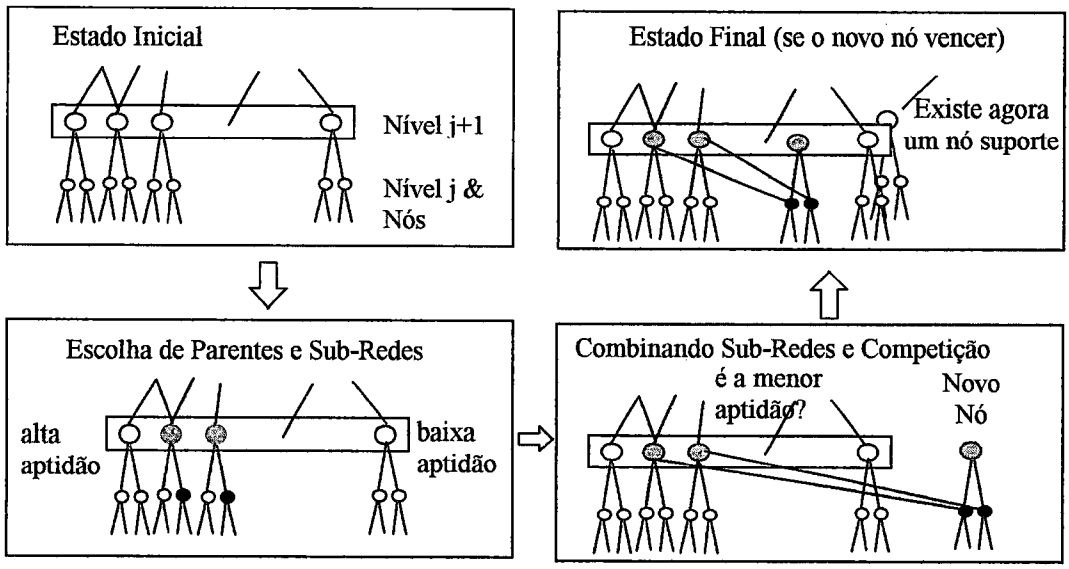


Figura 3.23 Visualização do Processo de Recombinação (MARTINS, 1994).

MARTINS (1994) realizou diversos experimentos com esta estrutura de sistema, testando diversas topologias de rede com o modelo GSN padrão. Os resultados obtidos mostraram que o sistema hierárquico produziu melhorias significativas no procedimento padrão GSN.

Além da implementação do sistema hierárquico, MARTINS (1994) também implementou modificações para otimização dos códigos de saída. Estas modificações levam em conta ganhos relacionados a alterações nos bits de código de saída (ganho posicional). O algoritmo implementado, para buscar melhores códigos é mostrado abaixo.

```
Inicie com uma codificação Hadamard
        construa matriz distância inter_classes
        modifique as distâncias inter-classes
        stop_now = falso;
        repeat
                Calcule as medidas inter_classes e variância
                Para cada código de saída faça
                        Para cada bit posicional faça
                                calcule o ganho individual das trocas
                possível_alteração = mais proveitoso (útil)
                new = variância (possível_alteração)
                se new < old então faça a alteração
                        senão stop_now = true
        until stop_now
```

Utilizando estas modificações, MARTINS (1994) realizou experimentos que comprovaram as melhorias nas taxas de reconhecimento do conjunto de teste.

Outro aspecto considerado foi o controle dinâmico da ordem de apresentação dos exemplos, ou seja, a possibilidade de impor algum comportamento cooperativo entre redes que formam o classificador.

A ordem de apresentação para cada rede foi considerado através da escolha de um vetor *ranking* global. Inicialmente, os exemplos estão numa ordem randômica e cada um tem um "0" como seu *ranking* associado. O vetor *ranking* é atualizado durante o treinamento da rede. Uma família de algoritmos, utilizada para este fim é mostrada a seguir.

```
Ranking ← 0
for B de 1 até Bmax faça                (bit posicional para o código de saída)
    for T de 1 até Tmax faça            (número de redes por bit posicional)
        for E de 1 até Emax faça        (exemplos de treinamento)
        ensine_e_atualize(ranking [E], B, T);
        sort_ranking;
end.
```

Uma regra simples aumenta o *ranking* dos exemplos aprendidos. Apesar de simples, esta regra pode controlar, dependendo de quanto os códigos de saída são pré-arranjados, quando o conjunto de treinamento inteiro foi ensinado.

Os experimentos conduzidos por MARTINS (1994), utilizando este controle dinâmico demonstraram melhorias significativas no desempenho do classificador.

3.5 CONSIDERAÇÕES FINAIS

Apresentou-se, neste capítulo, uma introdução aos conceitos de Redes Neurais Artificiais, além de um histórico de seu desenvolvimento. Fez-se então um estudo de Redes Booleanas, através de seus conceitos básicos e principais modelos. Tais modelos são apresentados e avaliados.

O modelo GSN foi então apresentado, através da descrição de seu funcionamento e conceitos básicos. As principais arquiteturas possíveis de serem implementadas utilizando o modelo GSN foram apresentadas e avaliadas. Os principais conceitos envolvendo a operação deste modelo em tarefas de reconhecimento de padrões foram apresentados, além das abordagens existentes.

O capítulo 4, a seguir, tratará do problema de Reconhecimento de Padrões, envolvendo principalmente abordagens utilizando Redes Neurais Artificiais.

4. VISÃO COMPUTACIONAL

4.1 VISÃO COMPUTACIONAL E ROBÓTICA

Pesquisas e aplicações práticas na área de visão computacional tiveram um grande progresso nos últimos anos. Áreas como robótica, automação industrial, sensoreamento remoto, reconhecimento de objetos, reconhecimento de texto, entre outras, utilizam os conceitos de visão computacional.

No contexto relacionado à automação de processos industriais, pode-se dividir algumas áreas relevantes da visão computacional em duas sub-áreas (claramente relacionadas): ***Processamento de imagens***, a qual trata principalmente com operações nas imagens e ajudam a melhorar sua "qualidade", ou enfatizar aspectos de importância particular e o ***Reconhecimento de Padrões***, o qual está relacionado com a identificação ou interpretação das imagens. Ela ajuda a extrair informação (em alto nível) daquilo que a imagem tenta comunicar.

Sistemas de visão computacional são compostos tipicamente de quatro tarefas (SPIRKOVSKA & REID, 1994):

- *Pré-Processamento*: para remoção de ruído e acentuação de bordas;
- *Segmentação:* para delimitar contornos de objetos ou padrões contidos em uma cena;
- *Reconhecimento de Padrões*: para determinar os padrões presentes na imagem;
- *Compreensão da Imagem*: para interpretar os relacionamentos entre os padrões contidos na cena.

A integração destas quatro tarefas resulta em sistemas capazes de identificar objetos e analisar cenas, para aplicações que demandem tal nível de realimentação sensorial.

A robótica é uma das principais áreas de utilização de tais sistemas, em aplicações envolvendo tarefas que incluem a manipulação de pequenos itens em operações repetitivas, montagem de produtos formados por diversas peças, além de tarefas de inspeção visual. Outras abordagens utilizam sistemas capazes de analisar o ambiente de operação, levando em conta aspectos característicos do meio, para tomada de decisão, utilizando realimentação sensorial sofisticada, baseada principalmente em dados obtidos do sistema de visão acoplado ao robô. Sistemas de rastreamento de alvo também utilizam a informação visual para implementar tarefas como soldagem e pintura (SONG & CHANG, 1996; NAJJARI & STEINER, 1995).

De maneira geral, podemos considerar um robô como sendo constituído de três elementos principais, como mostrado na figura 4.1, a seguir.

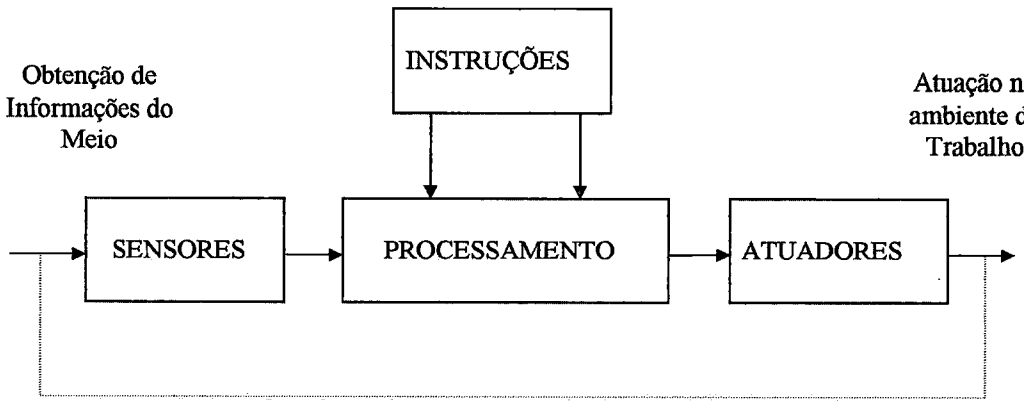


Figura 4.1. Estrutura básica de um robô.

O elemento de processamento é o bloco principal, controlando precisamente quais ações o sistema executará. Nos sistemas mais simples isto permite o movimento retilíneo em uma seqüência programada de instruções definidas por um conjunto específico de instruções, sendo o ambiente de operação "constante", ou seja, todos os componentes são individualmente localizados e todas as relações espaciais são completamente determinadas.

Em um ambiente mais sofisticado, ou seja, não controlado, a unidade de sensoreamento desempenha um papel extremamente importante na operação global do sistema, já que diversas variáveis do meio devem ser monitoradas e controladas, de acordo com requisitos definidos pelo processo em execução.

Para que um robô possa ter as habilidades de interação com o meio de trabalho de maneira precisa e inteligente, ele deve trabalhar com um poderoso sistema de realimentação de sinais. A visão situa-se como um importante componente deste sistema. Ela proporciona uma variedade de informações que podem habilitar um determinado mecanismo a trabalhar com incertezas inerentes à tarefa, reagir a variações do meio e, de forma delicada, recuperar-se de falhas (NAYAR et al., 1996).

Visão computacional e suas aplicações são de capital importância na obtenção de meios tecnológicos para a implementação de robôs inteligentes, processando e analisando imagens e fornecendo sua descrição. Tal descrição é normalmente na forma simbólica e depende da aplicação em questão.

Os sistemas de visão computacional estão relacionados a três campos específicos (KARATHANASSI et al., 1996):

- Processamento de Imagens digitais;
- Reconhecimento de Padrões;
- Análise de Cena;

O *processamento* de imagens digitais tem como pré requisitos diversos passos de pré-processamento, em cada aplicação específica. Durante o processo de aquisição das imagens, diversos erros internos e externos ao processo degradam a qualidade das mesmas, afetando consequentemente a qualidade e consequentemente a fidelidade dos dados obtidos através da análise subsequente. Através do processamento digital, novas imagens são obtidas, utilizando diversas técnicas as quais minimizam tais erros e melhoram a qualidade das imagens originalmente obtidas.

O *reconhecimento de padrões* trabalha com a classificação de objetos em categorias definidas. Cada categoria é única, baseada em características específicas. Cada objeto da imagem é classificado em uma determinada categoria depois de comparação das características individuais.

A *análise da cena* trata da transformação de uma representação simbólica da imagem para outra que seja menos complicada e mais fácil de ser interpretada. Normalmente, representação de conhecimento e inferência são utilizados para este propósito.

Os avanços na área de visão artificial estão intimamente relacionados ao desenvolvimento de hardware e muitos deste desenvolvimentos são impulsionados pela necessidade de processamento mais rápido de imagens. Algumas aplicações de visão em robótica são:

- Detecção de presença ou tipo de objeto;
- Determinação de localização e orientação de objeto antes do acionamento da garra;
- Realimentação, durante a manipulação com a garra;
- Realimentação para controle de trajetória em processos de soldagens e outros processos contínuos;
- Realimentação em processos de ajustes, durante operações de montagem;
- Leitura de códigos de identificação;
- Contagem de objetos;
- Inspeção de peças e objetos em processos de controle de qualidade.

A configuração básica de um sistema de visão é mostrada na figura 4.2, a seguir.

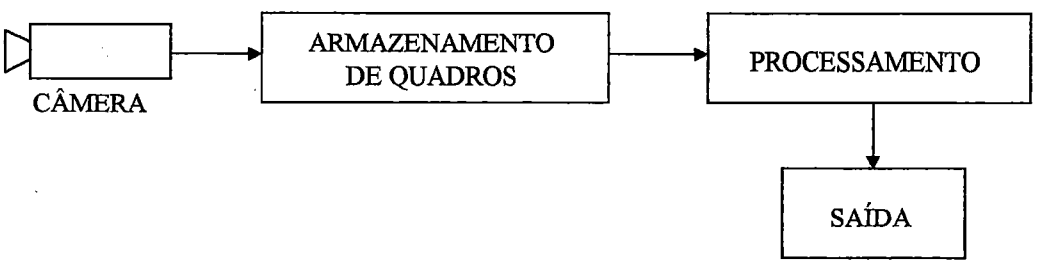


Figura 4.2. Sistema básico de Visão.

A câmera é utilizada para capturar uma imagem que será armazenada e acessada pelo sistema de processamento e análise utilizado.

A informação obtida da câmera normalmente gera uma imagem em tons de cinza na qual o nível de intensidade de cada pixel da imagem é representado por um valor binário. Normalmente 6 ou 8 bits são utilizados para codificar a escala de cinza. Se o requisito de cor for especificado, então pode ser utilizado um esquema de filtragem apropriado, às expensas de um aumento nos requisitos de armazenamento dos dados e tempo de processamento. A resolução é determinada pelo número de pixels utilizado. Para diversas aplicações industriais, imagens de 256 x 256 ou 512 x 512 pixels, trabalhando com escala de tons de cinza de 8bits, necessitando de 64k ou 256K bytes de armazenamento são suficiente. Para outras aplicações como na área médica e de satélites, resoluções bem maiores são necessárias, trabalhando com imagens da ordem de $10^5 x 10^5$ pixels.

Depois que a imagem foi capturada pela câmera e armazenada, ela deve ser processada para extração da informação nela contida, envolvendo um estágio *de pré-processamento*, para correção de distorções em um estágio de *extração de características*, no qual características determinantes da informação de imagem são extraídas e utilizadas para propósitos de classificação.

Em relação aos tipos de visão computadorizada aplicados à robótica pode-se citar:

- **Bi-Dimensional com objetos isolados e imagens binárias**, cujo objetivo é determinar a posição e orientação de um objeto, de tal maneira que um robô possa manipulá-lo. Os objetos normalmente são do mesmo tipo, sendo desnecessária a tarefa de reconhecimento;
- **Bi-Dimensional, com objetos isolados e imagens em tons de cinza**, onde a informação dos tons de cinza é utilizada para delimitação de fronteira e, consequentemente, para diferenciação de objetos;
- **Bi-Dimensional, com objetos sobrepostos**, onde comparações são feitas, levando em conta dados como ângulos e distâncias, formando modelos que são utilizados para determinar quais os tipos de objeto presentes na cena;
- **Inspeção Bi-Dimensional**, onde a presença de defeitos e falhas é checada, numa imagem do objeto sob análise, utilizando métodos de comparação e definição de regiões de interesse;
- **Rastreamento Bi-Dimensional**, onde o manipulador (ou robô) deve seguir (rastrear) uma trilha definida, utilizando-a como base para a operação que está realizando (soldagem, normalmente);
- **Extração de informação Tri-Dimensional de um objeto**, onde o sistema deve ser capaz de medir algumas dimensões específicas de uma classe determinada de objetos. Normalmente é utilizada a técnica de visão estéreo para que a informação de profundidade possa ser obtida;
- **Análise Tri-Dimensional de cena**, onde as informações obtidas dos itens anteriores são utilizadas para determinar situações específicas presentes numa determinada cena, utilizando diversas técnicas e procedimentos.

Diversos trabalhos vem sendo desenvolvidos na área de visão artificial, levando em conta os requisitos de integração de informação sensorial no esquema de acionamento e posicionamento utilizados.

4.1.1 Processamento Digital de Imagens

Na manipulação de imagens, um requisito fundamental é gerar um método apropriado de representação dos dados obtidos desta imagem. Tal método deve encapsular a informação que define as características importantes da imagem e deve proporcionar uma base para o processamento eficiente e conveniente através de computador.

Se considerarmos o conceito de imagem de forma simplificada podemos considerá-la como uma função bi-dimensional, onde o valor desta função f(x,y), para as coordenadas (x,y) define uma medida da intensidade de luz ou luminosidade para aquele ponto.

Para propósitos de visão computacional, será utilizado o conceito de imagem digital, o qual pode ser obtido de duas maneiras (FAIRHURST, 1988);

- **Digitalização Espacial**, a qual envolve a representação da função contínua original como um arranjo de exemplos para pontos discretos dentro da estrutura bi-dimensional de referência. Tal imagem, digitalizada espacialmente, consiste de N x N exemplos igualmente distribuídos, onde cada ponto discreto do arranjo é identificado como um *pixel*;
- **Digitalização em Amplitude**, onde um nível de intensidade de imagem é designado como um *nível de cinza* e a faixa total disponível de tais níveis é referenciada como *escala de tons de cinza*. O número de níveis de cinza é escolhido, por conveniência computacional, como potência de 2, e então qualquer pixel no arranjo N x N tem um nível de amplitude, g, onde $0 \leq g \leq 2^l - 1$ e l é um inteiro.

O número de bits necessário para o armazenamento de uma imagem digitalizada é dado por:

$$Número\ de\ Bits = P \times l$$

onde P é o número de pixels no arranjo digitalizado e 2^l representa o número de tons disponíveis na escala de cinza.

Um caso particular de digitalização de amplitude ocorre quando a escala de cinza consiste de somente dois níveis possíveis (ou seja, l=1), tal que a imagem resultante consiste de um arranjo de pontos, cada um dos quais completamente branco ou completamente preto. Tal imagem é conhecida como **imagem binarizada**. A geração de uma imagem binarizada a partir de uma imagem geral em escala de tons de cinza é bastante simples, bastando para tal a definição de um valor apropriado de *thresholding* para particionar a imagem em pixels com somente um de dois valores possíveis.

Assim, se f(x,y) é o valor da intensidade para as coordenadas (x,y) na imagem e T é o valor de *thresholding*, então a imagem representada por f(x,y) é binarizada aplicando a regra:

Se $f(x,y) \geq T$ então $f'(x,y) = 1$
Se $f(x,y) < T$ então $f'(x,y) = 0$

onde $f'(x,y)$ denota a versão binarizada de $f(x,y)$.

Como imagens binarizadas trabalham com um único bit de informação, elas representam vantagens no processo de manipulação de imagens.

O processamento digital de imagens pode ser entendido como sendo a análise e manipulação de imagens pelo computador, e tem como finalidade:

- Extrair informação da imagem;
- Transformar a imagem (por exemplo, aumentar o contraste, realçar bordas) de tal modo que a informação seja mais facilmente discernível.

Operações básicas

São aquelas relacionadas com a melhoria da qualidade visual da imagem, alterando os valores dos níveis de cinza dos pontos da imagem, de tal maneira que a imagem resultante destas operações contenha propriedades desejáveis que a imagem original não contém. Envolvem as seguintes operações:

- **realce da imagem**, que procura enfatizar alguma característica interessante da imagem original.
- **restauração**, a qual procura corrigir alguma distorção sofrida pela imagem original.

Neste dois grupos encontram-se operações de Transformação de Escala de Cinza, Modificação de Histograma e Filtragem (por convolução, não-linear, etc.), Modelos de Degradação, Determinação de Função Espalhamento Pontual, Técnicas de Reconstrução, além de outras.

Operações Estruturais

São aquelas onde a geometria da imagem é alterada, mantendo-se o máximo possível os valores dos níveis de cinza. Elas incluem:

- Normalização de Posição;
- Escalonamento;
- Rotação;

- Detecção de Borda;
- Segmentação;
- Codificação.

Tais operações geralmente são utilizadas, numa fase de pré-processamento, gerando uma imagem com melhor qualidade em relação àquela originalmente obtida pela câmera de video.

Dentre as operações citadas, as operações de filtragem e detecção de borda são de especial interesse nos sistemas de visão artificial. A Filtragem é responsável principalmente pela eliminação de ruído da imagem original e Detecção de Borda segmenta objetos presentes numa determinada cena pela definição de regiões (grupos de pontos conectados) as quais descrevem os objetos.

Uma visão teórica aprofundada de tais operações, assim como exemplos de aplicação podem ser obtidos em GONZALEZ & WINTZ (1987).

4.2 RECONHECIMENTO DE PADRÕES

Nos sistemas de Visão Computacional, o Reconhecimento de Padrões tem como função a extração de informações de uma determinada imagem ou a classificação de seu conteúdo, podendo ser dividido em duas abordagens:

- Classificação de Padrões, cujo objetivo é relacionar um objeto desconhecido a uma determinada classe;
- Reconhecimento Sintático, no qual a descrição de um objeto é feita através de relações entre suas partes.

A classificação de padrões normalmente utiliza abordagens estatísticas para implementação do classificador, o qual é fundamentado numa rigorosa base probabilística, assumindo que a estrutura e distribuição dos dados no espaço do problema são bem conhecidos a priori. A construção dos classificadores se baseia na conversão da probabilidade a priori das classes dos padrões de treinamento em medidas de probabilidade condicionada a posteriori de um dado padrão pertencer a uma dada classe de treinamento.

O Reconhecimento Sintático baseia-se no fato de que inter-relacionamentos ou interconexões de características fornecem importantes informações estruturais que podem ser utilizadas na descrição de padrões e em sua classificação (VASCONCELOS, 1995).

Os sistemas de reconhecimento de padrões apresentam normalmente três fases:

- *Treinamento*, onde o sistema aprende a discriminar as classes através da apresentação de exemplos representativos das classes de padrões;

- *Teste*, onde um conjunto de padrões não vistos na etapa de treinamento é apresentado, de tal maneira que a capacidade de generalização (derivação correta de classes) do sistema é avaliada;
- *Execução*, onde tarefas práticas de reconhecimento são realizadas pelo sistema.

A estrutura típica de um sistema de reconhecimento de Padrões pode ser vista na figura 4.3, abaixo.

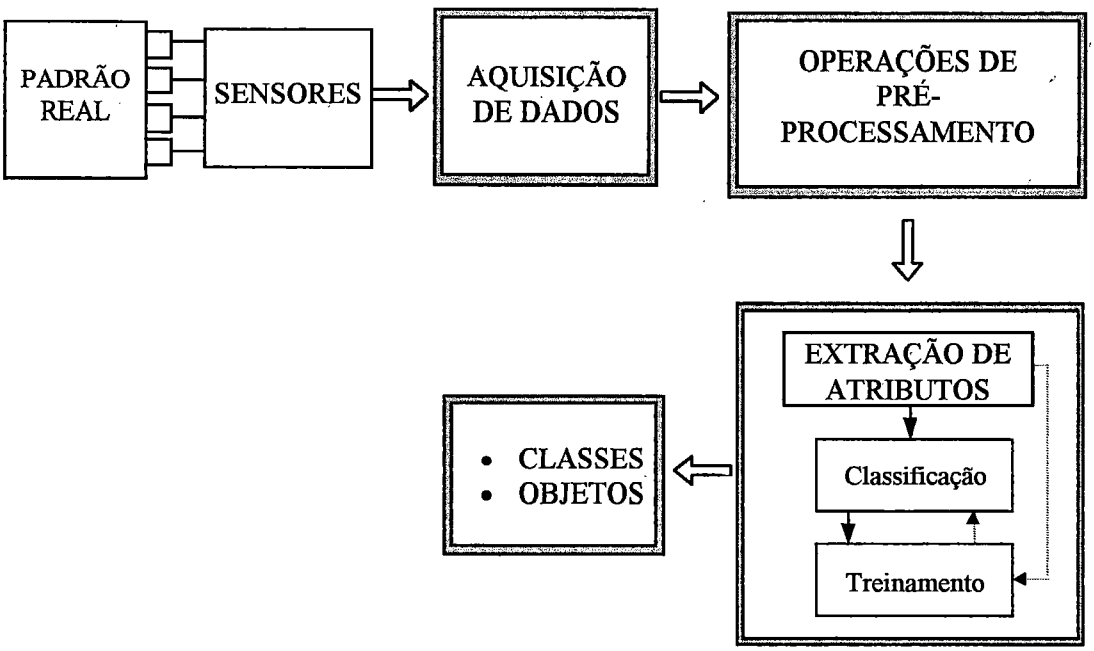


Figura 4.3. Sistema de Reconhecimento de Padrões (VASCONCELOS, 1995).

A tarefa de Reconhecimento pode ser dividida, conforme a figura, em quatro etapas:

- Aquisição de Dados;
- Pré-Processamento;
- Extração de Atributos;
- Classificação.

Na etapa de **aquisição de dados**, os objetos a serem classificados devem ser captados por sensores de imagens, os quais irão produzir modelos de imagens com algumas simplificações em relação às imagens reais: são discretos, bidimensionais, limitados em extensão e em número de cores ou níveis de cinza possíveis. Estes modelos, os quais são uma representação numérica da imagem, tornam possíveis o armazenamento e processamento das imagens através de computadores digitais. Tamanhos típicos de imagens são 128x128, 512x512, sendo 256 um número máximo de tonalidades de cinza, o qual normalmente é utilizado em função de requisitos de armazenamento.

A etapa de **pré-processamento** tem como objetivo filtrar ou minimizar os ruídos e distorções que possam resultar do processo de aquisição dos dados de entrada. Também podem ser realizadas, em alguns casos, operações de *segmentação* e *normalização* dos dados de entrada. A segmentação faz a separação de padrões que estejam de alguma forma ligados a outros, enquanto que a normalização modifica os dados de entrada reduzindo-o à escala normal das classes dos padrões, limitando o espaço de entrada.

A etapa de **extração de atributos** consiste na obtenção de medidas relevantes (atributos) que possam ser usadas na caracterização de padrões. Tais características podem ser numéricas (área, volume), simbólicas (cor), ou uma combinação das duas. A extração de atributos é também uma forma de compressão de dados, já que reduz a dimensionalidade da informação, elimina a redundância e aumenta a discriminação entre classes.

A escolha de um conjunto de atributos deve levar em conta algumas propriedades dos mesmos (PARKER, 1994):

- Velocidade de Processamento;
- Grande Discriminação de Classes;
- Pouca Variância em cada Classe;
- Completude da Descrição.

A utilização de atributos invariantes a transformações é um método bastante utilizado no reconhecimento de padrões, sendo que a escolha destes na maioria das vezes influencia as taxas de reconhecimento alcançadas. Fatores como tempo de cálculo dos atributos (na fase de pré-processamento) e redução de dados (compressão) devem também ser levados em conta (YUCEER & OFLAZER, 1993).

A etapa de **classificação** realiza um mapeamento entre os atributos e as classes dos padrões, ou seja, através dos atributos calculados, o classificador define uma determinada classe baseado numa função de decisão.

Os n atributos podem ser visualizados como coordenadas de um espaço n-dimensional e os objetos a serem identificados podem ser representados por pontos neste espaço. As diversas classes de objetos particionam o espaço de atributos, formando subconjuntos (durante a fase de treinamento do sistema). Tais subconjuntos são regiões de decisão. Assim, um objeto será atribuído a uma determinada classe se seus pontos de atributos representativos estiverem em apenas uma região de decisão. Caso contrário, o objeto deve ser rejeitado (não classificado).

Abordagens utilizando Redes Neurais Artificiais vem sendo investigadas e utilizadas nas etapas do Reconhecimento de Padrões descritas anteriormente. A seguir, é feita uma descrição de alguns trabalhos relevantes.

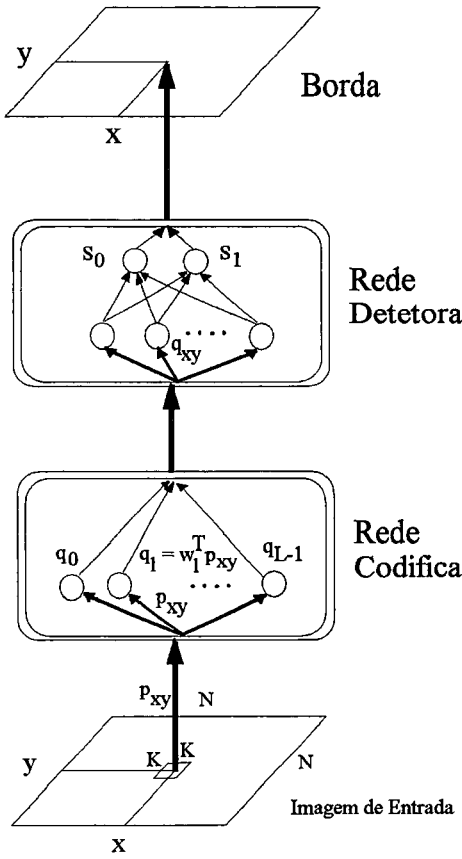
MOH & SHIH (1995), apresentam um modelo geral, para operações em imagens, baseado em Perceptrons Multicamadas. Os autores fazem uma comparação com o sistema visual humano e definem as operações de baixo nível do processamento de imagens, tais como suavização, realce, detecção de borda, remoção de ruído, além de operações morfológicas como aquelas de reação automática, não necessitando de inteligência. Tal reação, de adaptação, a qual produz uma imagem apropriada para propósitos de reconhecimento, é uma tarefa altamente paralela em sistemas biológicos.

Neste sentido, apresentam a utilização de modelos neurais, altamente paralelos, para realizar operações de processamento de imagens. Concluíram que a arquitetura apresentada, baseada em Redes Neurais, é flexível, podendo ser treinada com a maioria das operações de processamento de imagem (baixo nível).

A detecção de borda, tarefa que reduz drasticamente a quantidade de dados a serem processados também vem sendo implementada utilizando Redes Neurais Artificiais. SRINIVASAN et al. (1994) descrevem uma rede neural que opera em grande campo receptivo, numa abordagem inspirada na observação de que as primeiras camadas neurais do sistema visual de mamíferos extrai as bordas da imagem na retina. O detector de borda descrito consiste de dois estágios: o primeiro é um codificador de camada simples com 16 entradas o qual realiza, após treinamento, a compressão parcial da imagem em um campo receptivo 15x15. Diferentemente da compressão geral de imagem realizada através de uma rede backpropagation, o objetivo aqui é codificar seletivamente a informação de borda na imagem. Isto é conseguido treinando a rede com campos receptivos extraídos de imagens sintéticas consistindo de bordas de diferentes intensidades e orientações. O segundo estágio é um detector cujas 16 entradas são fornecidas pelo primeiro estágio. Ele gera duas saídas representando as componentes de um vetor borda. A compressão de dados alcançada pelo primeiro estágio simplifica o treinamento do segundo estágio, através da redução do número de entradas para o mesmo.

Tal implementação pode ser vista na figura 4.4, a seguir.

A rede opera em uma campo perceptivo KxK centrado na coordenada de pixel (x,y) de uma imagem NxN. O codificador, com L neurônios, gera um vetor saída de imagem comprimida $q_{xy} = [q_0 q_1 \ldots q_{L-1}]^T$ de um vetor entrada $p_{xy} = [p_0 p_1 \ldots p_{K^2-1}]^T$ cujos elementos são obtidos da s intensidades de pixel no campo KxK. O próximo estágio é um detector que mapeia q_{xy} ao vetor borda bidimensional $\mathrm{s}_{xy} = s_0 \mathrm{i} + s_1 \mathrm{j}$. Este vetor dá a intensidade ao longo das duas direções perpendiculares a (x,y) do qual a borda e direção resultantes podem ser calculadas.
O codificador é treinado de tal maneira que, para um dado campo de entrada, com um vetor p_{xy} associado, ele produz um vetor saída comprimida $\mathrm{q}_{xy} = \mathrm{W}^T \mathrm{p}_{xy}$.
A matriz peso $\mathrm{W} = [\mathrm{w}_0 \mathrm{w}_1 \ldots \mathrm{w}_{L-1}]$ tem vetores $\mathrm{w}_0 \mathrm{w}_1 \ldots \mathrm{w}_{L-1}$ que representam vetores pesos ligando cada um dos L neurônios ao campo de entrada. O detector é uma rede de duas camadas *feed-forward*. A camada hidden contem M neurônios e aceita o vetor codificado q_{xy} como entrada.



A camada de saída tem dois neurônios e fornece um vetor borda S_{xy} de saída . O vetor saída $S_{xy} = s_0 i + s_1 j$ é o vetor borda para a posição (x,y) na imagem de entrada, onde s_0 e s_1 são as intensidades de borda ao longo das duas direções perpendiculares i e j. A intensidade e orientação de borda resultantes podem ser calculados por $S_{xy} = \sqrt{s_0^2 + s_1^2}$ e $\theta_{xy} = \tan^{-1} \frac{S_1}{S_0}$.

Para cada pixel (x,y) da imagem de entrada, a rede mapeia o vetor codificado q_{xy} obtido como resultado da codificação da janela em torno de (x,y) no campo de entrada para um vetor borda bidimensional S_{xy}.

Figura 4.4. Rede Neural para a Deteção de Borda.

A utilização deste tipo de detector de borda, traz resultados bastante satisfatórios quando comparados à outros tipos de implementação.

LU & SZETO (1993) apresentaram um modelo hierárquico, composto por três módulos de redes neurais, o qual realiza o realce de bordas de imagens em operações de processamento de imagens. Cada módulo está associado a uma orientação particular e contém processos funcionais selecionados. A operação do sistema mostra a eficiência da arquitetura, através de resultados práticos em tarefas de processamento de imagens.

Operações de segmentação de imagens vem sendo também implementadas através do uso de Redes Neurais Artificiais. KOH et al. (1995) apresentaram uma Rede Neural do tipo *Multilayer Self-Organizing Feature Map* para a segmentação de imagens, consistindo de múltiplas camadas competitivas. Mostraram também as vantagens da utilização deste tipo de abordagem em relação aos métodos tradicionais. PAL et al. (1993) também utilizaram Redes Neurais (Perceptron Multilayer) para extração de atributos (representação da forma) de imagens em aplicações de processamento de imagens. O modelo proposto é aplicável a detecção de formas côncavas e convexas e, através de dados experimentais, a efetividade do modelo é provada. Outros trabalhos, usando Redes Neurais, para operações de segmentação, localização de objetos e separação de background em imagens são mostrados em VAILLANT et al. (1994), SHIRVAIKAR & TRIVEDI, (1995).

Uma abordagem utilizando Redes Neurais para compressão de dados de imagens é apresentada por FANG et al. (1992), o qual utiliza Redes Self Organization em uma arquitetura de Processador Neural VLSI. O artigo descreve a implementação de um chip VLSI, o qual proporciona a implementação de sistemas baseados na abordagem proposta.

4.2.1 Métodos para o Reconhecimento de Padrões

Diversos métodos tem sido utilizados para a realização da tarefa de Reconhecimento de Padrões. Os principais envolvem: métodos estatísticos, métodos utilizando Redes Neurais, método Vizinho Mais Próximo, método utilizando lógica *fuzzy*, além de métodos utilizando algoritmos genéticos.

O **Reconhecimento Estatístico de Padrões** utiliza teoremas como o de Bayes para o cálculo da probabilidade das classes de objetos, assumindo conhecimento prévio da distribuição das variáveis utilizadas na classificação (MASTERS, 1995). Num problema prático envolvendo um certo número de classes e, sendo conhecidas as funções densidade de probabilidade associadas a cada classe, $P(C_i)$, utiliza-se a regra de probabilidade condicional de Bayes para estimar a classe de cada padrão.

Como o conjunto de atributos ($A= a_1, a_2, \ldots\ldots a_n$) é também um conjunto das medidas da instância cuja classe se deseja determinar, a probabilidade condicional $P(C_i/A)$ fornece uma visão do quanto as medidas estão relacionadas aos valores padrão para a classe determinada.

A decisão por uma determinada classe será feita através de comparações destas probabilidades para cada classe, sendo o maior valor tomado como resultante.

A probabilidade condicional é dada por:

$$P(C_i / A) = \frac{P(A / C_i).P(C_i)}{\sum_j P(A / C_j).P(C_j)} \qquad 4.1$$

onde C_i representa as classes, A representa o conjunto de atributos e $P(C_i/A)$ a probabilidade condicional.

Redes Neurais Artificiais vem sendo utilizadas no Reconhecimento de Padrões através de diversas abordagens e arquiteturas. Serão vistas com mais detalhes no próximo item.

O método **Vizinho Mais Próximo** utiliza o conceito de distância entre a instância de classe conhecida e os padrões previamente aprendidos. Diversas abordagens podem ser utilizadas para obter a distância entre os atributos do objeto e dos padrões armazenados, entre elas a Distância de Hamming, Distância Euclidiana, Distância Quadrada, entre outras (COSTA, 1996).

O conceito de **Fuzzy Logic** é bastante atrativo pelo fato de fornecer uma estrutura potente e flexível para representar conceitos vagos e pouco definidos, representando e numericamente manipulando regras linguísticas de maneira natural. Os métodos utilizando tais conceitos são apropriados para as situações em que os atributos são vagos.

O reconhecimento de padrões é realizado como segue. Fontes de informação são identificadas (atributos, informação contextual, etc.) e uma medida fuzzy é gerada para cada classe de padrão, usando uma medida subjetivamente gerada ou estimada dos dados de treinamento ou de valores de todos os subconjuntos das fontes de informação. Esta geração de medidas é a fase de treinamento para a abordagem fuzzy. Dado um padrão a ser classificado, uma função evidência $h_i(x_j)$ é avaliada para cada fonte de informação x_j e cada classe w_i. As funções h_i são então integradas em relação a suas correspondentes medidas fuzzy de classe, resultando em um valor confidência para cada classe. Estes valores confidência são usados para uma tomada de decisão final de classificação, ou seja, designar o padrão à classe com a maior confidência. Um exemplo de um classificador usando fuzzy logic pode ser visto em KELLER et al. (1994).

Algoritmos Genéticos também vem sendo investigados e classificadores baseados nestes conceitos vem sendo implementados, principalmente em conjunto com Redes Neurais Artificiais. O conceito de Algoritmo Genético busca, como em Redes Neurais, simular um processo biológico natural, utilizando-se de dois conceitos básicos: representação de soluções possíveis em strings de dígitos binários - *o conceito de codificação* e utilização de um algoritmo para avaliar a solução - ***o conceito de avaliação***. Numa analogia biológica, os strings são cromossomos e cada dígito binário é um gene. Estes conceitos são utilizados em conjunto com outros do tipo *crossover* e *mutação*. MARTINS (1994) apresenta uma descrição de tais conceitos, além da utilização dos mesmos em conjunto com Redes Neurais Artificiais para tarefas de reconhecimento de padrões.

4.3 RECONHECIMENTO DE PADRÕES USANDO REDES NEURAIS

O Reconhecimento de Padrões vem sendo implementado, através da abordagem de Redes Neurais Artificiais, utilizando diversos modelos. O objetivo principal é reconhecer um objeto, apesar de mudanças no mesmo em relação ao campo de entrada. Tal conceito, o de Invariância Geométrica, possibilita o Reconhecimento de Padrões invariante a transformações na escala, rotação e translação do padrão de entrada apresentado à Rede.

Para tal, existem diversas abordagens, as quais podem ser classificadas em quatro categorias principais:

- Normalização dos Padrões;
- Invariância pela Estrutura da Rede;
- Invariância Pelo Treinamento;
- Utilização de Atributos Invariantes.

A **Invariância pelo Treinamento** consiste na apresentação seqüencial à Rede Neural, durante seu treinamento, de diversas vistas transformadas do padrão a ser reconhecido. O objetivo, ao incluir vistas transformadas no conjunto de treinamento, é que as camadas *hidden* extraiam as características invariantes necessárias e a rede generalize. A figura 4.5, abaixo, ilustra esta abordagem.

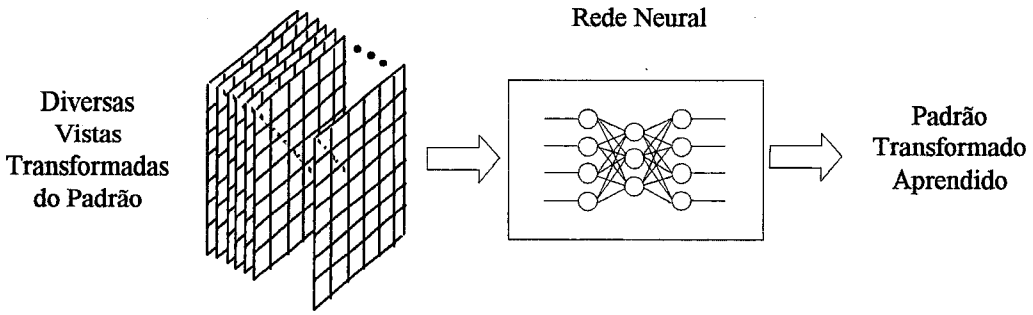


Figura 4.5. Abordagem utilizando Invariância pelo Treinamento.

Esta abordagem implica em um grande conjunto de treinamento e uma boa capacidade de generalização do modelo de Rede Neural utilizado. Outro requisito é que o número de classes não seja muito grande. Tal abordagem vem sendo implementada através do uso de redes baseadas em backpropagation. O tempo de treinamento, portanto, será proporcional ao tamanho do conjunto de treinamento, o qual deve envolver diversas características de cada padrão a ser reconhecido (TRUXEL et al., 1988; RUMELHART et al., 1986; ZHANG et al., 1991). Tais características, intrínsecas à abordagem, não são apropriadas às aplicações descritas neste trabalho e, portanto, não serão objeto de detalhamento maior.

As demais abordagens serão apresentadas a seguir.

4.3.1 NORMALIZAÇÃO DOS PADRÕES

A normalização dos padrões envolve a apresentação dos mesmos através de uma maneira padrão ao classificador. Isto implica em transformações no padrão de entrada, de tal maneira que sua posição, orientação e tamanho sejam representados sempre da mesma forma ao classificador, de maneira standard.

A implementação desta abordagem se dá através de um estágio de pré-processamento, como mostrado na figura 4.6, a seguir.

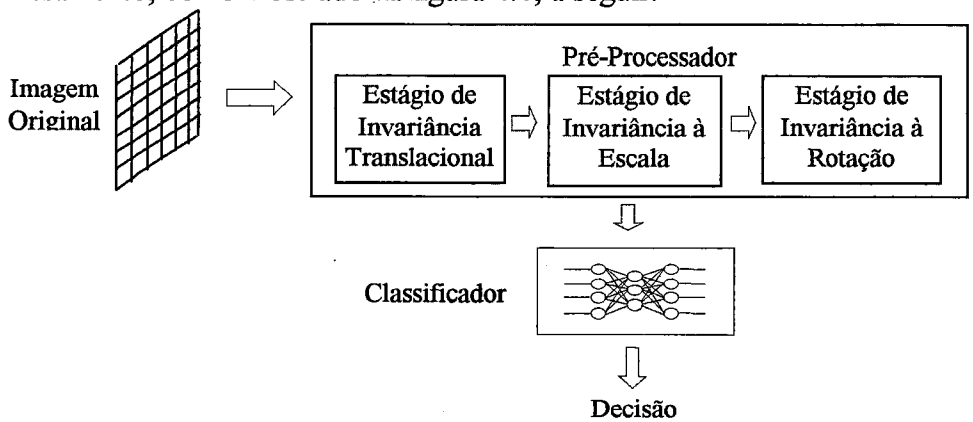


Figura 4.6. Abordagem utilizando Normalização de Padrões.

Uma das técnicas de obtenção de invariância através desta abordagem é a utilização da transformada de Fourier, a qual normaliza o padrão em relação à posição.

4.3.2 INVARIÂNCIA PELA ESTRUTURA DA REDE

O objetivo, em relação à invariância em posição, escala e rotação (PSRI - Position, Scale and Rotation Invariance), no domínio do reconhecimento de objetos, é reconhecer um objeto, apesar de mudanças em sua posição no campo de entrada, ou rotação no plano. Várias técnicas vem sendo utilizadas para obtenção deste objetivo, utilizando a abordagem de inserção, no modelo da rede, de características que forçam o modelo de rede neural a classificar vistas transformadas do objeto como sendo da mesma classe. Tal abordagem inclue Redes Neurais de Alta Ordem - Higher-Order Neural Networks - HONN (GILES & MAXWELL, 1987; GILES et al., 1988, REID et al., 1989, SPIRKOVSKA & REID, 1990, KOWALCZYK & FERRA, 1994), Neocognitron (FUKUSHIMA, 1983; FUKUSHIMA, 1985; FUKUSHIMA, 1989), entre outras (WIDROW et al., 1988; VON DER MALSBURG, 1988).

Nestas abordagens, relações conhecidas são exploradas e as invariâncias desejadas são construídas diretamente na arquitetura da rede. Construindo tais conhecimentos de domínio específico na arquitetura da rede, obtêm-se uma rede que é "pré-treinada" e não necessita "aprender" invariância a transformações.

Para cada novo conjunto de objetos treinados, a rede necessita somente aprender a distinguir entre os objetos treinados; ela não necessita generalizar o conceito embutido nas invariâncias. Portanto, o tempo de treinamento é significativamente reduzido e tais redes necessitam ser treinadas em somente uma vista de cada objeto, não em numerosas vistas transformadas. Além do mais, acuracidade de 100% no reconhecimento é garantida para imagens livres de ruído caracterizadas pelas transformações embutidas (SPIRKOVSKA & REID, 1993).

A seguir, tais abordagens são apresentadas e discutidas.

REDES NEURAIS DE ALTA-ORDEM (Higher Order Neural Networks - HONN)

Uma Rede Neural de Alta Ordem (Higher Order Neural Network), aqui denominada HONN, pode ser projetada para ser invariante à escala, translação e rotação no plano. Invariâncias são construídas diretamente na arquitetura da HONN e não precisam ser aprendidas. Consequentemente, poucos passos e um pequeno conjunto de treinamento são necessários para aprender a distinguir os objetos (WANG & SUN, 1996; SPIRKOVSKA & REID, 1993, TAKANO et al. 1994).

Redes Neurais HONN são uma forma de processamento para as Redes Neurais backpropagation padrão que usam combinações não lineares de pixels de uma cena, motivadas geometricamente, para obter características invariantes para o reconhecimento de padrões (VILLALOBOS & MERAT, 1995).

A saída de um nó, denotado por y_i para o nó i, em uma Rede Neural HONN (Higher-Order Neural Networks) é dada por:

$$y_i = \theta (\Sigma_j w_{ij} x_j + \Sigma_j \Sigma_k w_{ijk} x_j x_k + \Sigma_j \Sigma_k \Sigma_l w_{ijkl} x_j x_k x_l + \ .....) \qquad 4.2$$

onde θ é uma função limiar não linear, os x_j's são os valores de excitação dos nós de entrada e os elementos da matriz de interconexão, w_{ij}, determinam o peso que cada entrada terá na soma. Estes termos higher-order podem ser usados para construir invariância à transformação diretamente na arquitetura da rede, usando informação sobre relações esperadas entre os nós de entrada. Como exemplo, uma rede estritamente de terceira ordem pode ser usada para construir invariância simultânea à translação, escala e rotação no plano. A saída para uma rede de terceira ordem é dada pela função:.

$$y_i = \theta (\Sigma_j \Sigma_k \Sigma_l w_{ijkl} x_j x_k x_l) \qquad 4.3$$

Como mostrado na figura 4.7, a seguir, em uma rede de terceira ordem os pixels de entrada são primeiro combinados em *triplets* e então a saída é determinada através de uma soma ponderada destes produtos.

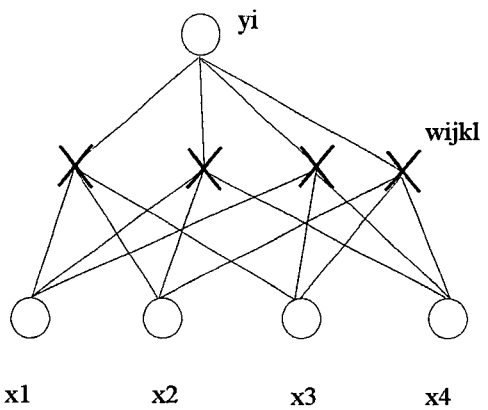


Figura 4.7. Rede de 3.a ordem com 4 entradas e 1 saída.

As entradas são primeiro combinadas e então multiplicadas por um peso antes de serem somadas (SPIRKOVSKA & REID, 1993).

Estas *triplets* de pixels de entrada representam triângulos com ângulos (α,β,γ). Para construir invariância para as três transformações na arquitetura da rede, os pesos são forçados de tal maneira que todas as combinações de três pixels, as quais definem triângulos similares, são conectados à saída com o mesmo peso. Isto é:

$$w_{ijkl} = w_{imno} \qquad 4.4$$

se os ângulos ordenados do triângulo formado pelos pixels conectados (j,k,l) forem iguais aos ângulos ordenados do triângulo formado pelos pixels conectados (m,n,o).

Os ângulos são ordenados de tal maneira que o menor ângulo é listado primeiro e os próximos dois ângulos são listados na ordem que seriam encontrados se visitados na direção horária.

Então, os três triângulos representados pelos ângulos (30 60 90), (60 90 30) e (90 30 60) são todos conectados ao nó de saída com o peso associado a (30 90 60), ao passo que os três triângulos com ângulos (30 90 60), (90 60 30) e (60 30 90) são todos conectados ao nó de saída com o peso associado a (30 90 60). Isto extrai todos os triângulos os quais são similares geometricamente. Estas características similares são invariantes a todas as transformações geométricas as quais não variam os ângulos internos, incluindo translação e rotação no plano, conforme mostrado na figura 4.8, a seguir.

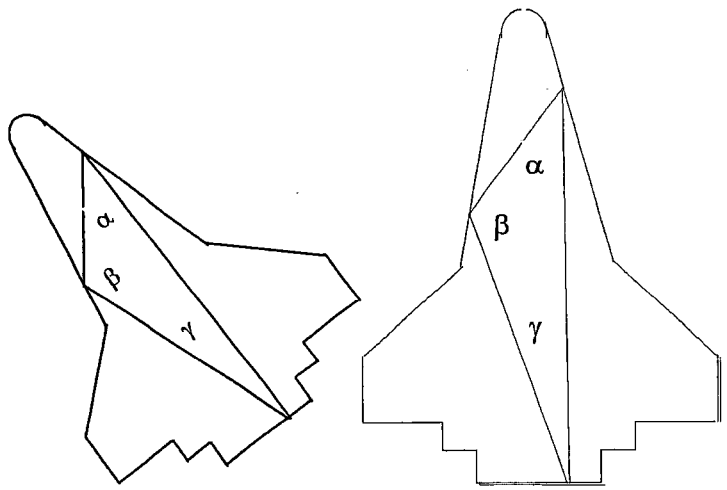


Figura 4.8. Triângulos similares formados por *triplets* de pixels do padrão de entrada (SPIRKOVSKA & REID, 1993).

Triângulos similares são também invariantes a pequenas variações da escala.

Ao contrário das modificações de translação e rotação no plano, onde um objeto e uma vista transformada do mesmo contém exatamente o mesmo número de cada possível triângulo dispensando o sistema de distinguir entre as duas vistas, o escalonamento de um objeto introduz novos triângulos. Este problema é minimizado construindo invariância parcial à escala de duas maneira. Primeiro, limita-se a resolução para a qual os ângulos internos são calculados, decrementando o possível número de triângulos únicos. Segundo, usa-se imagens representadas somente pela borda, reduzindo o número de novos triângulos introduzidos. Para minimizar o problema, a rede pode também ser treinada em vistas escaladas de cada objeto para que possa ser hábil a determinar o conteúdo relativo dos triângulos em vistas com escala alterada de um objeto versus vistas (com escalas alteradas) de outros objetos

As conexões para triângulos similares e equivalência de pesos são estabelecidas antes que qualquer imagem esteja presente no campo de entrada. Como tais invariâncias estão contidas na arquitetura da rede antes que qualquer vetor de entrada seja representado, a rede precisa aprender a distinguir entre somente uma vista de cada objeto, ao invés de numerosas vistas transformadas.

Diversos trabalhos envolvendo o uso de HONN para tarefas de reconhecimento de padrões vem sendo desenvolvidos. Uma breve descrição dos mesmos é mostrada a seguir.

GILES & MAXWELL (1987) apresentam um estudo do comportamento de HONN's em relação a aspectos como invariância e generalização. Neste trabalho, concluem que as mesmas possuem capacidades de armazenamento e aprendizado tais que podem ser projetadas para trabalhar satisfatoriamente de forma invariante a transformações geométricas.

SPIRKOVSKA & REID (1992a, 1992b, 1994) usaram a abordagem de HONN para tarefas de reconhecimento de padrões em duas e três dimensões, avaliando também as limitações de tal abordagem. A principal limitação, o tamanho do campo visual, demanda uma grande quantidade de memória para o armazenamento das interconexões. Técnicas de *coarse coading* são então apresentadas e avaliadas. Esta técnica, a qual envolve conectividade parcial, contorna o problema de limitação de memória, permitindo o uso da abordagem em problemas práticos (imagens reais) de reconhecimento de objetos).

O problema de armazenamento em memória foi estudado por STRAVOS & LISBOA (1992), os quais apresentaram uma abordagem que reduz o número de pesos da rede neural, produzindo altas taxas de reconhecimento, com menores requisitos de armazenamento e processamento.

SCHIMIDT (1993) utilizou a abordagem de HONN para detectar objetos em cenas ruidosas. Concluiu que uma grande redução de tempo de processamento pode ser obtida, utilizando tal abordagem, proporcionando características apropriadas a utilização em aplicações de tempo real.

TIRAKIS et al. (1994) propuseram uma arquitetura baseada na relação entre correlações triplas e redes neurais, resultando em um sistema que é invariante às transformações geométricas. Simulações mostraram a efetividade do modelo apresentado, através da utilização de imagens reais de teste.

SKONECZNY et al. (1995) utilizaram HONN's no reconhecimento de faces. O sistema apresentado utiliza uma rede Sigma-Pi como extrator de *features*.

WANG & SUNG (1996) apresentaram uma estrutura híbrida, utilizando HONN e mapeamento bidirecional log-polar, para o reconhecimento de alvo. A estrutura contorna o problema de armazenamento de pesos e apresenta propriedades de invariância geométrica (PSRI - Position, Scale and Rotation Invariance).

NEOCOGNITRON

Uma abordagem diferente para o reconhecimento de objetos, usando Redes Neurais, é o Neocognitron. Desenvolvido por K. Fukushima (FUKUSIMA, 1980), baseado no sistema visual humano, o Neocognitron é construído em uma estrutura de hierarquia de camadas (FUKUSHIMA, 1989). O primeiro estágio é a camada de entrada - um arranjo bi-dimensional de células receptoras. Cada estágio sucessivo consiste de duas camadas compostas por dois tipos de células diferentes. Um tipo de célula extrai características locais enquanto o outro extrai características globais.

O Neocognitron aprende com uma regra não-supervisionada reforçando os pesos, através da utilização de aprendizado competitivo, extraindo os atributos dos padrões e classificando-os. Através de treinamento, o Neocognitron pode reconhecer um padrão independentemente de sua posição no campo de entrada, de uma pequena variação em seu tamanho ou de uma pequena deformação.

Embora as deformações possam incluir pequenas rotações (de poucos graus) o neucognitron não foi projetado para tratar da invariância a qualquer tipo de rotação. Tal abordagem tem ao menos duas limitações. Primeiro, o número de células no modelo aumenta quase que linearmente com o número de objetos que necessita aprender a distinguir. Isto torna o processo de treinamento muito lento. Também, como em redes de primeira ordem, os pesos são específicos aos padrões e, portanto, a rede deve ser completamente re-treinada para cada novo conjunto de padrões.

Um sistema de reconhecimento de caracteres alfanuméricos foi desenvolvido baseado no modelo Neocognitron (FUKUSHIMA, 1992). Os princípios da atenção seletiva podem ser estendidos para diversas aplicações, por exemplo, o reconhecimento e segmentação de caracteres conectados em manuscritos de palavras na língua inglesa ou o reconhecimento de caracteres chineses.

Diversas aplicações envolvendo a abordagem do Neocognitron vem sendo apresentadas e investigadas.

CHAO (1992) introduziu numa rede neural óptica baseada no paradigma do Neocognitron. Um novo aspecto do projeto da arquitetura é a invariância a translações usando correlações ópticas multicanais de Fourier em cada camada de processamento. Processamento multi-camadas foi obtido interativamente retroagindo a saída do correlator de atributos para a entrada do modulador espacial de luz e atualizando os filtros de Fourier.

CHAO & STONER (1993) aplicaram o modelo do Neocognitron para o reconhecimento de alvos, em aplicações militares.

XING & FELTHAM (1994) desenvolveram uma arquitetura piramidal de Rede Neural para ser utilizada no reconhecimento de padrões, particularmente na identificação de tumores cancerosos, a qual é baseada no Neocognitron.

As principais desvantagens da abordagem baseada no Neocognitron são a necessidade de muitos elementos processadores e de estruturas complexas de processamento (MAREN et al., 1990; COSTA, 1996).

OUTRAS TÉCNICAS

Um esquema utilizando duas redes neurais, para reconhecimento invariante a translação, foi proposto por BUHMANN & SCHULTEN (1988) o qual utiliza as redes para, respectivamente extrair e codificar atributos do padrão e os memoriza.

WIDROW et al. (1988) utilizaram uma rede baseada em estruturas ADALINE para a obtenção de invariância geométrica. Tal estrutura, no entanto, apresenta necessidades de altos requisitos de hardware, não sendo apropriada para aplicações práticas.

Algumas modificações foram introduzidas no modelo original do Neocognitron para tentar superar as deficiências apresentadas no mesmo (HOSOKAWA et al., 1989; CRUZ et al., 1989, FUKUMI et al., 1992). Tais modificações resultaram em melhor desempenho que o modelo original, sendo aplicadas em alguns tipos de problemas práticos de reconhecimento de objetos.

Outra abordagem envolve o uso de Arquitetura de Ligações Dinâmicas (LADES et al., 1993, VON DER MALSBURG, 1988), na qual as imagens são representadas por grafos topológicos rotulados. Tal arquitetura, baseada em Redes Neurais com ligações Dinâmicas, apresenta reconhecimento invariante à posição e escala (limitadas). Modificações na arquitetura original foram apresentadas por BUHMANN et al. (1990).

4.3.3 UTILIZAÇÃO DE ATRIBUTOS INVARIANTES

Neste tipo de abordagem, após a extração de atributos invariantes, uma Rede Neural classificadora faz a determinação de classes dos objetos apresentados ao sistema, conforme mostrado na figura 4.9, a seguir.

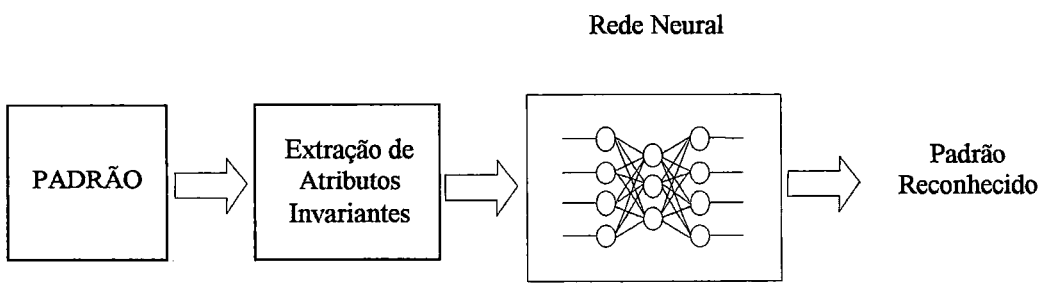


Figura 4.9. Abordagem utilizando Atributos Invariantes.

Atributos são propriedades mensuráveis dos objetos presentes na imagem e a representação de padrões através de atributos invariantes a transformações geométricas é um método de implementação do reconhecimento de padrões.

Os atributos podem ser globais (perímetro, área, momentos), locais (segmentos de reta, vértices) ou relacionais (regiões do objeto, distâncias) e sua escolha, na maioria das vezes, determina o desempenho do sistema de reconhecimento de padrões usando espaço de atributos invariantes (CHIN & DYER, 1986).

Normalmente, a escolha dos tipos de atributos utilizados pelo sistema depende dos tipos de objetos ou padrões a serem reconhecidos. Idealmente um atributo invariante teria valor constante, independente do grau de transformação sofrido pela imagem do objeto. Porém, na prática, diversas fontes de incerteza são acrescentadas ao modelo ideal, resultando, em uma variabilidade de valores medidos.

Diversas abordagens vem sendo propostas para a utilização de Redes Neurais em conjunto com Atributos Invariantes, envolvendo o uso de diversos modelos. As arquiteturas mais utilizadas englobam o Perceptron Multilayer e *Radial Basis Function* (CHTIOUI et al., 1996; GUPTA et al., 1989; BORGES et al., 1994; MARUNO, 1993;; LEJEUNE & SHENG, 1993; HWANG & TSENG, 1993; ANAND et al., 1993; WANG & COHEN, 1994; VARELLA et al., 1994, YAN & WU, 1994., BORGES et al., 1994, MITZIAS & MERTZIOS, 1994; VASCONCELOS, 1995). Outras abordagens incluem RNA do tipo counterpropagation (SEPEDA FILHO & STEMMER, 1994), GSN (CARVALHO, 1995), Redes Neurais *Constraint Satisfaction* (TSAO et al., 1993), Redes Neurais Recorrentes - *Recurrent Neural Networks* (WAARD, 1994), além de outras.

A seguir, apresenta-se uma visão geral de alguns atributos utilizados para este tipo de abordagem (COSTA, 1996).

Momentos descrevem quantidades numéricas em alguma distância de um ponto de referência ou eixo e podem ser utilizados para caracterizar uma imagem, binária ou em níveis de cinza, considerando esta como uma função de distribuição bidimensional de densidade. (PROKOP & REEVES, 1992; WONG et al., 1995).

A representação de toda informação presente numa imagem utilizaria um número infinito de valores de momento. Assim, para a implementação prática de sistemas de reconhecimentos que utilizem momentos, torna-se necessário determinar a ordem para a qual os momentos possam trazer informações que caracterizem a imagem.

A definição de momentos geométricos regulares tem a forma de uma projeção da função $f(x, y)$ que representa a imagem, em uma função monomial do tipo $x^p y^q$. O momento (p+q) é definido de acordo com a equação 4.5, a seguir.

$$m_{p,q} = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} x^p y^q . f(x,y).dxdy \qquad 4.5$$

para (p,q=0,1,2,.....)

Para uma imagem discretizada (NxM), o momento bidimensional é dado pela equação 4.6, a seguir.

$$m_{p,q} = \sum_{y=0}^{M-1}\sum_{x=0}^{N-1} x^p y^q . f(x,y) \qquad 4.6$$

para (p,q=0,1,2.....)

Utilizando a área da imagem,

$$m_{0,0} = \sum_{y=0}^{M-1}\sum_{x=0}^{N-1} f(x,y) \qquad 4.7$$

a equação 4.6 pode ser normalizada, de tal maneira que os momentos resultantes sejam invariantes a translação e escalonamento. Isto resulta na equação 4.8, abaixo.

$$\mu_{p,q} = \frac{1}{m_{0,0}^{1+(p+q)/2}} \sum_{y=0}^{M-1} \sum_{x=0}^{N-1} x^p y^q . f(x + \frac{m_{1,0}}{m_{0,0}}, y + \frac{m_{0,1}}{m_{0,0}}) \qquad 4.8$$

para (p,q=0,1,2....)

$\frac{m_{1,0}}{m_{0,0}} e \frac{m_{0,1}}{m_{0,0}}$ representam as coordenadas do centróide do objeto na imagem.

A combinação de momentos de segunda e terceira ordem resulta em atributos que são invariantes também à rotação dos padrões na imagem.

$$\phi_1 = \mu_{2,0} + \mu_{0,2}$$
$$\phi_2 = (\mu_{2,0} - \mu_{0,2})^2 + 4\mu_{1,1}^2$$
$$\phi_3 = (\mu_{2,0} - 3\mu_{1,2})^2 + (3\mu_{2,1} - \mu_{0,3})^2$$
$$\phi_4 = (\mu_{3,0} + \mu_{1,2})^2 + (\mu_{2,1} - \mu_{0,3})^2$$
$$\phi_5 = (\mu_{3,0} - 3\mu_{1,2})(\mu_{3,0} + \mu_{1,2}).\left[(\mu_{3,0} + \mu_{1,2})^2 - 3(\mu_{2,1} + \mu_{0,3})^2\right] +$$
$$+(3\mu_{2,1} - \mu_{0,3})(\mu_{2,1} + \mu_{0,3}).\left[3(\mu_{3,0} + \mu_{1,2})^2 - (\mu_{2,1} + \mu_{0,3})^2\right]$$
$$\phi_6 = (\mu_{2,0} - \mu_{0,2})\left[(\mu_{3,0} + \mu_{1,2})^2 - (\mu_{2,1} + \mu_{0,3})^2\right] + 4\mu_{1,1}(\mu_{3,0} + \mu_{1,2})(\mu_{0,3} + \mu_{2,1})$$

4.9

A utilização do logaritmo do módulo das funções ϕ_1 a ϕ_6 proporciona atributos representativos das imagens que estão sendo utilizadas no sistema de reconhecimento.

Assim,

$\log|\phi_k|$, $k = 1,2.....6$ representam os atributos invariantes da imagem.

O reconhecimento de padrões invariante à posição, escala e rotação, pode ser obtido pela utilização de invariantes de momentos em conjunto com Redes Neurais Artificiais. A utilização destes momentos, no entanto, envolve alguns problemas: são sensíveis ao ruído, não permanecem invariantes quando a imagem apresenta mudança de escala diferente para os eixos x e y e demandam muito tempo de cálculo (RAVEENDRAN et al., 1993; LI, 1991).

SUGISAKA & TESHNEHLAB (1993) apresentaram um sistema de reconhecimento de objetos, invariante às transformações geométricas (rotação, translação e escala), usando momentos invariantes em conjunto com Redes Neurais artificiais. A figura 4.10, a seguir, mostra esquematicamente o sistema.

Através da combinação de Redes Neurais e Momentos Invariantes, o sistema pode reconhecer padrões independentemente de transformações geométricas.

A simulação foi feita utilizando uma rede neural com 50 unidades na camada hidden. O número de interações de aprendizado é diferente para cada padrão. Os testes realizados mostraram a eficiência do sistema, já que boa parte da computação pesada relacionada ao cálculo dos Momentos Invariantes é eliminada.

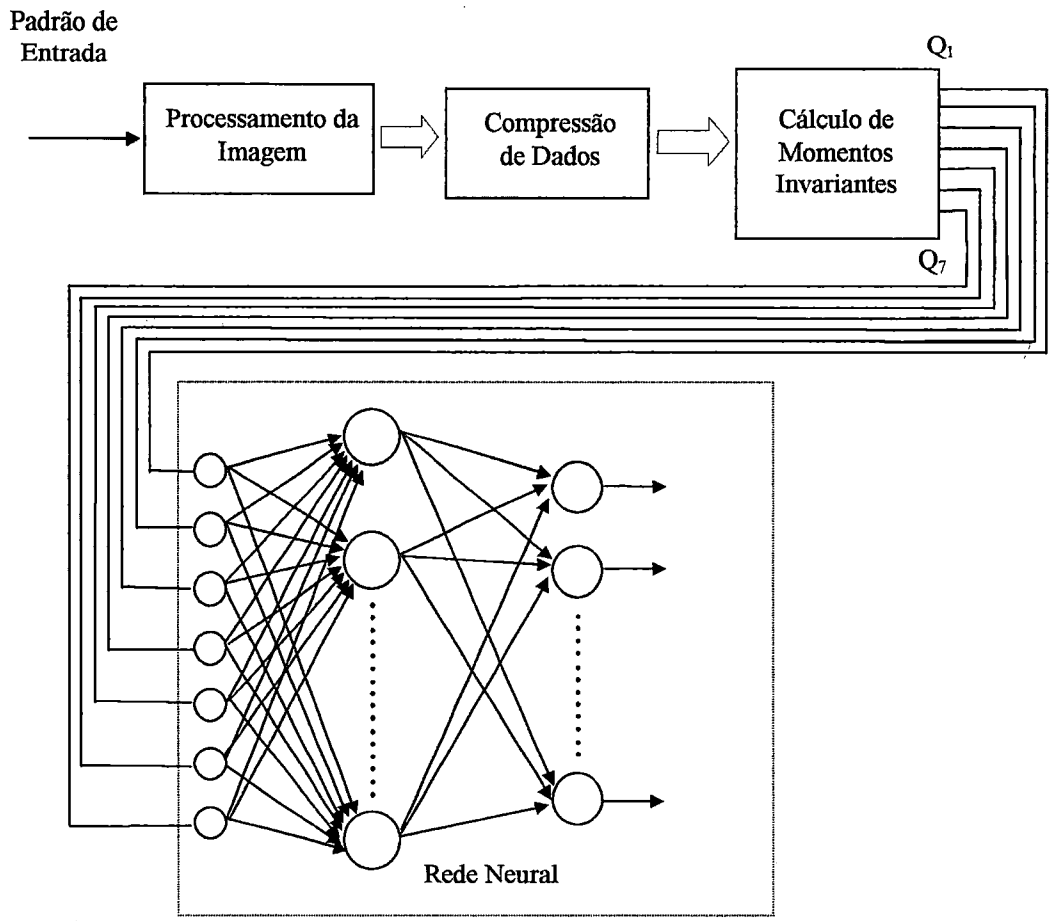


Figura 4.10. Sistema de Reconhecimento de Padrões (SUGISAKA & TESHNEHLAB (1993).

Momentos de Zernike

Momentos de Zernike são um conjunto de polinômios complexos, os quais formam um conjunto ortogonal em um círculo unitário. Tais momentos são baseados em funções ortogonais e suas magnitudes não variam com a mudança de orientação do padrão, podendo ser utilizados como atributos invariantes à rotação. São mais fáceis de calcular do que os momentos regulares, apresentam menor redundância de informação e possibilitam a restauração da imagem a partir destes valores (PROKOP & REEVES, 1992).

O momento complexo de Zernike Znl é definido como (COSTA, 1996):

$$Z_{nl} = \frac{(n+1)}{\pi} \int_0^{2\pi} \int_0^{\infty} V_{nl}(r,\theta) * f(r,\theta) r dr d\theta \qquad 4.9$$

onde * representa o conjugado complexo.

Os polinômios de Zernike de ordem n são dados por:

$$V_{nl}(r,\theta) = R_{n,l}(r) e^{il\theta} \qquad 4.10$$

sendo $0 \leq 1 \leq n$, $n-1$ par.

O polinômio radial é dado por:

$$R_{nl}(r) = \sum_{k=l}^{n} B_{nlk} . r^k \qquad 4.11$$

Os coeficientes de Zernike para (n - k) par são:

$$B_{nlk} = \frac{(-1)^{(n-k)/2} [(n+k)/2]!}{[(n-k)/2]! [(k+l)/2]! [(k-l)/2]!} \qquad 4.12$$

A utilização destes Momentos de Zernike proporciona invariância à rotação, necessitando, portanto, de imagens com posição e escala normalizadas.

Descritores de Fourier

Descritores de Fourier permitem a representação de padrões utilizando o contorno dos objetos. A partir de um contorno fechado, pode-se traçar duas curvas descrevendo as variações do contorno nos eixos x e y: x(n) e y(n), onde n = 0, 1, ..., N-1 são os pontos do contorno do objeto.

Para um contorno fechado,

$$u(n) = x(n) + jy(n) \qquad 4.13$$

o qual é periódico, com período N. Representando u(n) através da série discreta de Fourier:

$$u(n) = \frac{1}{N} \sum_{k=0}^{N-1} a(k) \exp\left(\frac{j2\pi kn}{N}\right)$$

$$a(k) = \sum_{n=0}^{N-1} u(n) \exp\left(\frac{-j2\pi kn}{N}\right)$$

Os coeficientes complexos a(k) são os descritores de Fourier do contorno.

Tais atributos tem pouca variabilidade em relação às transformações geométricas. O principal problema é a grande sensibilidade ao ruído.

4.3.4 OUTRAS TÉCNICAS

Análise de Componentes Principais (***Principal Components Analysis***)

ROMDHANI (1997) descreveu a utilização de PCA (*Principal Components Analysis*) no reconhecimento de faces, o qual pode ser descrito como um problema particular do Reconhecimento de padrões. A seguir, é mostrada uma formulação geral do problema.

A partir de imagens de uma cena, identificar uma ou mais pessoas nesta cena, usando uma base de dados de faces armazenadas. A solução de tal problema pode ser subdividida em três estágios:

- segmentação das cenas;
- extração de features da região da face;
- decisão.

A segmentação é obtida através da conexão, através da transformada de Hough, dos elementos de borda da face.

A extração de features envolve features de dois tipos: features holísticos (onde cada feature é uma característica da face) e features parciais (cabelo, nariz, boca, etc.). Para features parciais, algumas medidas em pontos cruciais da face são feitas, enquanto que para fatures holísticos, a técnica utilizada trata sempre a face como um todo. PCA é uma técnica de extração de features holísticos.

No terceiro estágio, uma decisão é tomada em função dos dados obtidos dos estágios anteriores. Tal decisão pode ser obtida através de três maneiras diferentes:

- identificação, na qual são obtidos labels de indivíduos;
- reconhecimento de uma pessoa, onde deve ser decidido se o indivíduo já foi visto;
- categorização, na qual uma face deve ser designada a uma dada categoria.

A imagem de uma face pode ser visualizada como um vetor. Se as dimensões da imagem são de largura **w** e altura **h** pixels, o número de componentes deste vetor será **w*h**. Cada pixel é codificado por um componente do vetor. A construção deste vetor é obtida através de uma simples concatenação, como mostrado na figura 4.11.

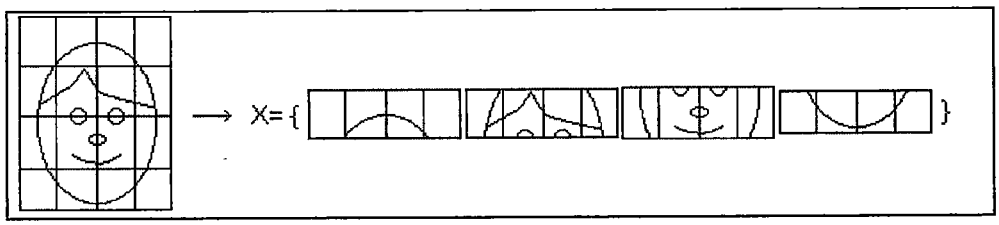


Figura 4.11. Formação do vetor face a partir da imagem ROMDHANI (1997).

Espaço Imagem

O vetor face, descrito anteriormente, pertence a um espaço, denominado espaço imagem, o qual é composto por todas as imagens cujas dimensões são **w** por **h** pixels. A base do espaço imagem é composta pelos vetores mostrados a seguir.

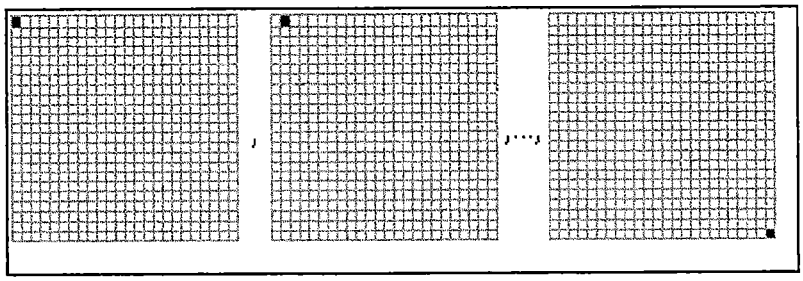


Figura 4.12. Base do Espaço Imagem ROMDHANI (1997).

Todas as faces tem dois olhos, uma boca, um nariz, etc., localizados no mesmo lugar. Portanto, todos os vetores face estão localizados em um cluster limitado do espaço imagem, como mostrado na figura 4.13, abaixo.

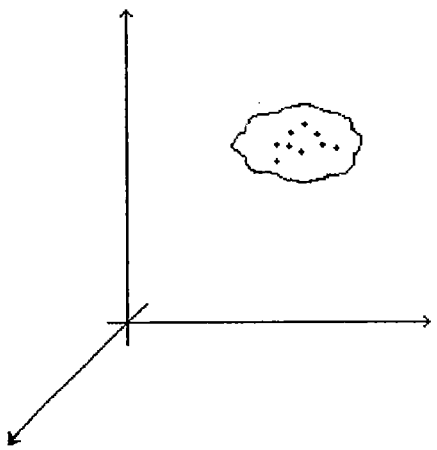


Figura 4.13. Espaço Imagem e face cluster ROMDHANI (1997).

Consequentemente, o espaço de imagem total não é um espaço ótimo para a descrição de face. A tarefa apresentada aqui busca construir um espaço o qual melhor descreva estas faces. Os vetores base deste espaço de face são chamados de componentes principais.

A dimensão do espaço de imagem é **w*h**. De fato, todos os pixels de uma face não são relevantes, e cada pixel depende de sua vizinhança. Assim, a dimensão do espaço de face é menor que a dimensão do espaço de imagem

O objetivo do método apresentado aqui, Análise de Componentes Principais, é reduzir a dimensão de um conjunto ou espaço tal que a nova base descreva melhor os "modelos" típicos do conjunto.

No presente caso, os "modelos" são um conjunto de faces de treinamento. A nova base será construída por combinação linear. Os Componentes nesta base de espaço de faces serão não correlacionados e maximizarão a variância obtida para as variáveis originais.

Principal Components Analysis (PCA) objetiva capturar a variação total no conjunto de faces de treinamento, e expressar esta variação através de poucas variáveis. Tal redução de dimensão é importante. De fato, uma observação descrita por poucas variáveis é mais fácil de manipular e de entender do que se fosse definida por uma quantia enorme de variáveis. E, quando muitas observações ou faces tem que ser processadas, a redução de dimensionalidade é de primordial importância.

PCA calcula a base de um espaço o qual é representado por seus vetores de treinamento. Os vetores base computados por PCA são na direção da maior variância dos vetores de treinamento. Estes vetores base são computados pela solução de um eigen problem, e como tal os vetores base são eigenvectors. Estes eigenvectors são definidos no espaço imagem. Eles podem ser visualizados como imagens e aqui serão referenciados como eigenfaces.

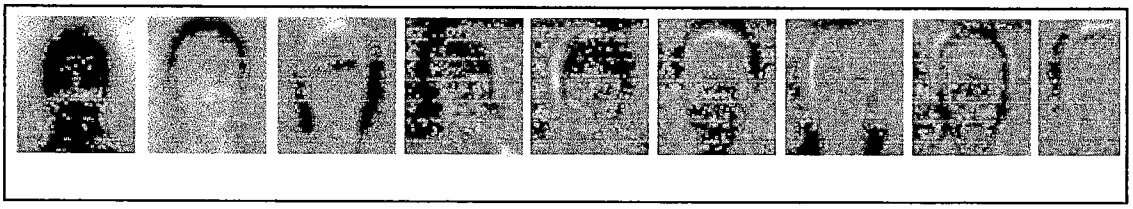


Figura 4.14. Primeiras eigenfaces ROMDHANI (1997).

A primeira eigenface é a face média, enquanto o restante das eigenfaces representa variações desta face média. A direção da maior variação (da média) dos vetores de treinamento é descrita pela segunda eigenface. A direção da segunda maior variação (da média) dos vetores de treinamento é descrita pela terceira eigenface, e assim por diante.

Cada eigenface pode ser visualizada como um *feature*. Quando uma face particular é projetada no espaço de face, seu vetor no espaço de faces descreve a importância de cada um destes *features* na face. A figura 4.15, a seguir, descreve o processo.

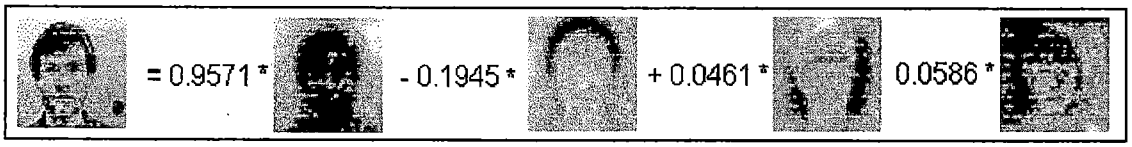


Figura 4.15. Uma face desenvolvida no espaço de faces ROMDHANI (1997).

Neste exemplo, uma face é desenvolvida no espaço de faces, sendo descrita por seus coeficientes de eigenface (ou pesos). Por conveniência, o vetor peso é normalizado. Como a imagem desenvolvida no espaço de face é de fato uma face, o peso da primeira eigenface é muito alto, quase igual a unidade. O valor dos pesos decresce com o incremento do número de eigenfaces. Isto está em conformidade com a definição de eigenfaces. De fato, PCA busca a direção das maiores variações. A primeira eigenface relata a variação máxima, a segunda para a segunda variação máxima, etc.

Processo de Geração

A figura 4.16, abaixo, mostra esquematicamente o que PCA faz. Ele toma as faces de treinamento com entrada e produz as eigenfaces como saída. Obviamente, o primeiro passo de qualquer experimento é computar as eigenfaces. Uma vez isto feito, o processo de identificação e categorização pode começar.

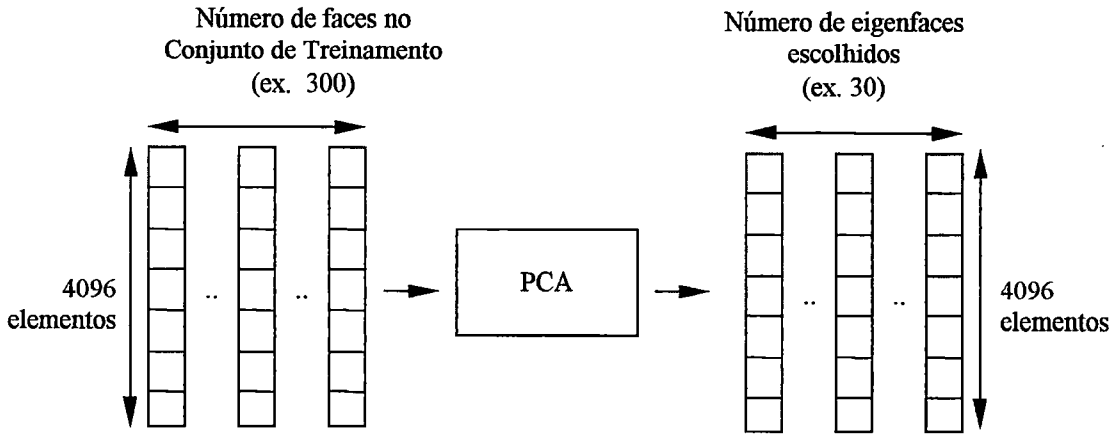


Figura 4.16. Processo de Geração de Eigenfaces ROMDHANI (1997).

Reconstrução

Uma vez computadas as eigenfaces, cada face no espaço de imagens pode ser visualizada no espaço de faces. A transformação do espaço de imagem para o espaço de faces é relativamente simples.

Sendo:

- E a matriz das primeiras eigenfaces, onde a primeira coluna é a primeira eigenface e assim por diante;
- f_I uma face no espaço de imagens, e
- f_F a mesma face no espaço de faces.

$$\mathbf{f}_F = \mathbf{f}_I * \mathbf{E}^T \qquad 4.15$$

Esta é uma transformação em que a dimensionalidade do espaço imagem é maior que a dimensionalidade do espaço face. Assim, a transformação introduz um erro o qual pode ser observado em uma face reconstruída. O processo de reconstrução é realizado tomando a transformação inversa da equação anterior.

$$\mathbf{f}_I = \mathbf{f}_F * \mathbf{E} \qquad 4.16$$

Como exemplo, a figura 4.17, a seguir, mostra uma face e sua reconstrução.

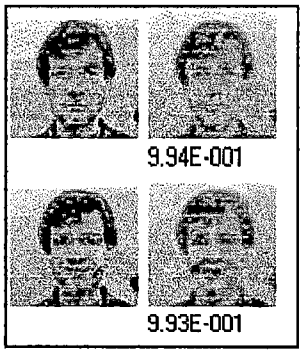


Figura 4.17. Face original e sua reconstrução. O erro de reconstrução é mostrado abaixo de cada face ROMDHANI (1997).

A magnitude da diferença entre a face original e sua reconstrução, denominada erro de reconstrução, é facilmente calculado. Fazendo f'_I ser a face reconstruída da face original f_I, e, o erro de reconstrução:

$$\mathbf{f'}_I = \mathbf{E}^T * \mathbf{E} * \mathbf{f}_I \text{ and } \varepsilon = |\, \mathbf{f}_I - \mathbf{f'}_I \,| \qquad 4.17$$

Como os vetores são normalizados, a distância cosseno pode ser utilizada, ao invés da distância Euclidiana:

$$\varepsilon = 1 - \cos(\mathbf{f'}_I, \mathbf{f}^T_I) \qquad 4.18$$

$$\varepsilon = 1 - \mathbf{f'}_I * \mathbf{f}^T_I \qquad 4.19$$

desde que $\mathbf{f'}_I$ e $\mathbf{f}^T_I$ sejam normalizados.

Processo de Identificação

Uma vez calculadas as eigenfaces, o espaço de face está povoado com faces conhecidas. Normalmente estas faces são tomadas de um conjunto de treinamento. Cada face conhecida é transformada no espaço de faces e seus componentes armazenados em memória.

Neste estágio o processo de identificação pode começar. Uma face desconhecida é apresentada ao sistema. O sistema a projeta no espaço de faces e calcula sua distância de todas as faces armazenadas. A face é identificada como sendo aquela mais próxima no espaço de faces. Existem vários métodos para calcular a distância entre vetores multidimensionais. Aqui, uma forma de distância Euclidiana foi escolhida. De fato, é o cosseno do ângulo entre as duas faces que é calculado. Como as faces são normalizadas (magnitude dos vetores igual a 1), numa comparação, a distância cosseno e distância Euclidiana são idênticas. A vantagem é que a distância cosseno pode ser calculada mais eficientemente.

SHANG & BROWN (1994) utilizaram PCA para classificação de imagens, utilizando redes neurais *Multi-layer Feedforward*, através de uma nova arquitetura do sistema de classificação.

YOU & FORD (1994) propuseram um sistema de reconhecimento de objetos, invariante a transformações, baseado em quatro sub-redes: a primeira implementa a Transformada de Radon (JAIN, 1989), a segunda proporciona capacidades de invariância à translação e escala, através de uma rede *maximum-pick-up*. A terceira proporciona features invariantes a rotação, através da Transformada Rápida (BURKHARDT & MULLER, 1990). A quarta rede, de reconhecimento, é implementada através de uma rede do tipo *multilayer perceptron*. Os resultados obtidos, segundo os autores, são comparáveis àqueles obtidos através da utilização de momentos de Zernike.

Outras abordagens envolvendo atributos invariantes utilizam o algoritmo de Karhuen-Loeve (HILAI & RUBINSTEIN, 1994).

OUTRAS ABORDAGENS

Redes Neurais *Adaptive Resonance Theory* (ART) são modelos biologicamente inspirados, implementam o processamento cooperativo e competitivo, através da formação de agrupamentos e são treinadas sem supervisão. Aplicações deste modelo em tarefas de reconhecimento de padrões são descritas em RAJAPAKSE & ACHARYA (1990) e SRINIVASA & JOUANEH (1992).

A utilização de uma estrutura neural para o reconhecimento de alvos, utilizando a integração de dois modelos neurais (MLP e Maxnet modificada), com propriedades de invariância à translação é descrita por MUNDKUR & DESAI (1991).

Uma estrutura integrada, apresentando propriedades de invariância geométrica, é apresentada por YOU & FORD. (1993). Tal estrutura é composta por quatro estágios. Tais estágios utilizam, respectivamente, Transformada de Radon, Rede *maximum-pick-up* e Rede MLP para prover invariância geométrica e estimar ângulos de rotação. O sistema, segundo os autores, apresenta boa performance quando comparado a outras abordagens.

MÖSCHEN & HOSTICKA (1993) apresentam uma visão geral das possibilidades de implementação em Hardware de arquiteturas para sistemas de reconhecimento de padrões. As aplicações envolvendo necessidades de processamento em tempo real são apresentadas e avaliadas.

PHAN & BAYRO-CORROCHANO (1994) descrevem um método de agrupamento de padrões, baseado no algoritmo de Kohonen e na estrutura de *perceptron multilayer*. O algoritmo de Kohonen funciona como um simples procedimento de *labelling* aplicado ao conjunto de dados de treinamento, para dividi-lo em clusters. Os dados de clusters são então utilizados para treinar uma rede *perceptrons* de três camadas usando a técnica de *backpropagation*. Os autores comprovaram, experimentalmente, que tal abordagem tem desempenho superior aos modelos ART.

KIM & YANG (1994) apresentaram um modelo de Rede Neural, AINET (*Adaptive Inference Network*) o qual tem capacidades de inferência lógica e aprendizado, para atividades de reconhecimento de padrões. O modelo híbrido de reconhecimento de imagens apresentado, consiste de dois estágios: extração de feature e reconhecimento. O modelo foi avaliado e os resultados experimentais provaram que os sistema proposto apresentou desempenho melhor do que o apresentado utilizando *Multilayer Perceptrons*.

HEMMINGER & RAEZ (1994) descrevem a utilização de uma Rede de Hopfield na implementação do reconhecimento de padrões invariante a escala e rotação. O sistema trabalha bem na presença de ruído e possui bom desempenho de reconhecimento.

PRICE et al. (1994) descrevem classificadores, usando redes neurais, nos quais saídas probabilísticas são fornecidas. O sistema proposto particiona o problema original de classificação em dois subproblemas, envolvendo duas classes: para cada par de classes, redes neurais são treinadas usando somente os dados destas duas classes. A combinação das saídas destas redes neurais fornece as probabilidades para as decisões de classes envolvidas no processo de classificação.

Implementações ópticas de Redes Neurais, em sistemas de reconhecimento de padrões invariante a transformações geométricas vem sendo realizadas e representam uma alternativa aos problemas de tempo de processamento e treinamento (BERGERON et al., 1995; UANG et al., 1994; KAKIZAKI et al., 1994).

Uma nova arquitetura para o reconhecimento de padrões invariante a rotações foi proposta por SAWAI (1994), utilizando Rede Neural Axialmente Simétrica. Esta arquitetura, composta de cinco camadas: uma camada de entrada, duas camadas *hidden* e duas camadas de saída é capaz também de detectar o ângulo de rotação do padrão. Os autores citaram a necessidade de integração do sistema em uma arquitetura modular, capaz de reconhecer padrões com distorções na escala e transladados.

KUNG & TAUR (1995) propuseram uma Rede Neural do tipo *Decision Based* - DBNN (*Decision Based Neural Network*) a qual combina a regra de aprendizado do *perceptron* e a estrutura de rede hierárquica não linear. Tal modelo foi utilizado em aplicações de classificação de imagens. Os autores concluíram que tal abordagem é um caminho mais natural e com melhores resultados do que outras abordagens.

4.4 SISTEMAS DE VISÃO COMPUTACIONAL PARA ROBÔS

LI & ZHANG (1986) Descreveram um sistema de visão inteligente para robô, no qual é priorizado um esquema de cooperação entre hardware e software para obter metas específicas, tais como processamento em tempo real, relação custo/benefício, etc. Processamento paralelo é utilizado. O algoritmo utilizado no sistema é baseado na extração de características por operações de convolução no domínio espacial. Uma visão esquemática do sistema é mostrada abaixo.

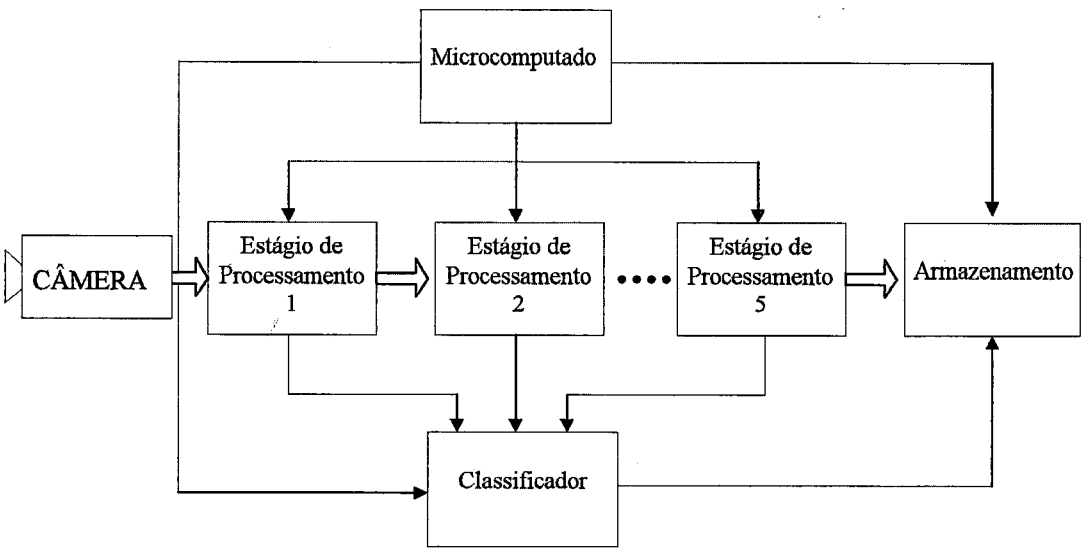


Figura 4.18. Sistema de Visão para Robô (LI & ZHANG, 1986)

O sistema é composto por uma seqüência de estágios processadores, sendo que cada estágio realiza um processamento simples em uma imagem. Os autores descrevem aplicações do sistema em sensoreamento remoto e operações de soldagem, sendo que as taxas de reconhecimento correto do sistema chegam a 97%.

HASHIMOTO et al. (1992) descreveu um sistema de controle para robô manipulador o qual usa informação visual para posicionar e orientar sua garra. O sistema de controle integra diretamente dados visuais no processo de servoacionamento. A rede neural é utilizada para determinar a mudança nos ângulos das juntas necessários para obter a posição e orientação desejados. A figura 4.19, a seguir, mostra o sistema.

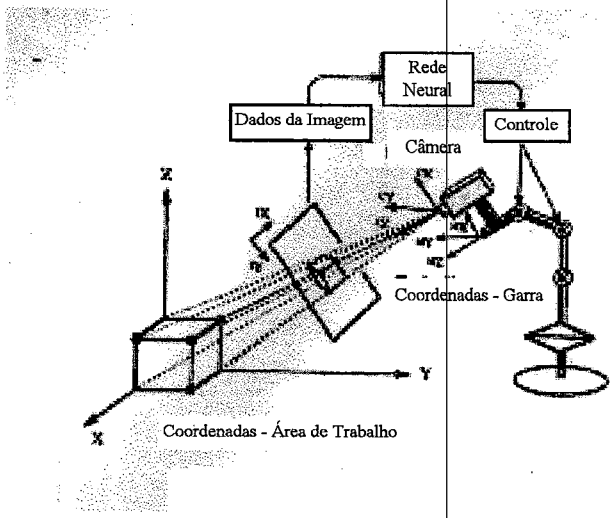


Figura 4.19. Configuração do sistema (HASHIMOTO et al., 1992).

O sistema utiliza duas Redes Neurais: uma "rede neural global" a qual é utilizada para aprender uma área grande de trabalho da garra e outra "rede neural local" a qual é utilizada para aprender a área na vizinhança da peça de trabalho. Estas redes, em conjunto, aprendem a relação não linear entre a imagem de entrada e os sinais de controle para as mudanças nos ângulos das juntas. Resultados experimentais comprovam a efetividade do esquema de controle utilizado, para robôs com diversos graus de liberdade.

COLOMBO et al. (1994) apresentou um sistema de visão artificial, baseado em redes neurais, o qual incorpora o conceito de comportamento atentivo, presente no sistema visual humano, o qual faz uma compressão de dados pela utilização de mecanismos de seleção atentiva. O sistema faz a integração de características sensoriais e semânticas dos dados visuais de entrada produzem o comportamento atentivo o qual pode variar levando em conta a tarefa sendo realizada.

MARTINETZ & SCHULTEN (1993) descrevem uma Rede Neural para a tarefa de aprender a coordenação visual-motora de uma garra de robô. Eles utilizaram o algoritmo de Mapa de Kohonen, o qual trabalha com dados provenientes de duas câmeras. Tais câmeras fornecem um par de coordenadas 2-D do objeto a ser manipulado. A Rede então é ensinada com as transformações entrada/saída, ou seja coordenadas de câmera para ângulos de junta apropriados para o posicionamento da garra do robô.

MOTAVALLI & BAHR (1993) descrevem a integração de um sistema de visão numa célula baseada em robôs. O sistema monitora ferramentas, através de dois parâmetros importantes nas operações de usinagem: condições de corte e identificação de tipo de ferramenta, além de realizar a identificação das peças. Os autores descrevem diversos métodos de monitoração, além do algoritmo utilizado no sistema, o qual analisa a proporção do número de pontos de borda não pertencentes às regiões de corte em relação àqueles pertencentes às regiões de corte na imagem da ferramenta para identificação de pontos de desgaste. A identificação de peças é feita utilizando a Transformada Generalizada de Hough e é composto pela fase de aprendizado, onde as características das peças são aprendidas e armazenadas num arquivo do microcomputador, e pela fase de reconhecimento, onde as características são comparadas. Uma vista esquemática do sistema é mostrada na figura 4.20, a seguir.

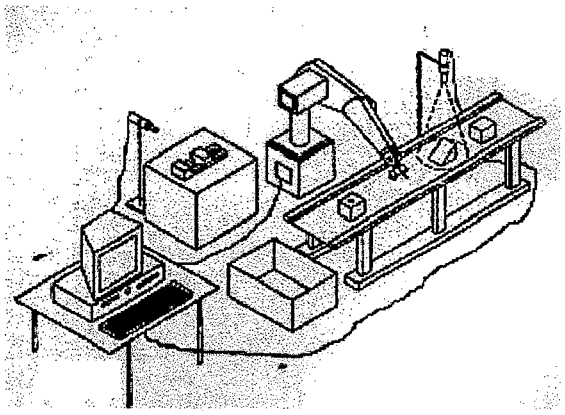


Figura 4.20. Sistema de Visão para Monitoração de Ferramentas de Usinagem (MOTAVALLI & BAHR, 1993).

OLSSON & GRUBER (1993) apresentam um sistema de inspeção de tecidos, baseado em Redes Neurais. A figura 4.21, a seguir, traz o diagrama de blocos do sistema

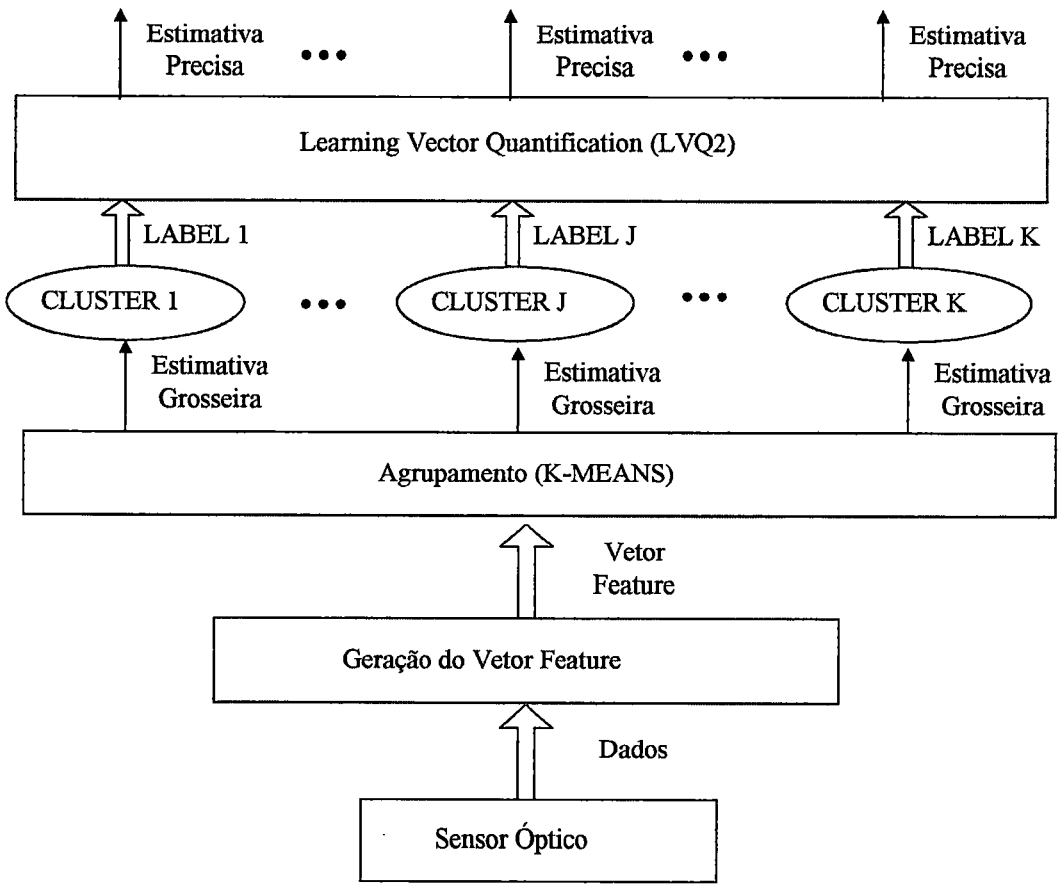


Figura 4.21. Sistema de Inspeção de Tecido (OLSSON & GRUBER, 1993).

O processo de inspeção neste tipo de aplicação, conforme os autores, envolve a análise de $5x10^6$ pixels por segundo, o que resulta em taxas de dados de entrada da ordem de 15Gbps, claramente impossíveis de serem manipuladas por computação tradicional. Assim, diversos métodos de inspeção da superfície do tecido são apresentados: intensidade, textura e dispersão de luz em diversos ângulos. Este último método, de dispersão de luz, é utilizado pelo sistema, sendo utilizada uma Câmera de CCD para aquisição da mesma. A imagem obtida é segmentada, para obter multi-segmentos, cada um relacionado a uma porção de distribuição angular da luz dispersa. A saída de cada segmento (a qual é proporcional intensidade da luz dispersa na superfície do segmento) é utilizada como um feature. O conjunto de features, obtida do conjunto de segmentos, é examinado por uma Rede Neural (LVQ2 - learning vector quantification), a qual foi previamente treinada para reconhecer a natureza da superfície..

Os resultados obtidos mostraram que o sistema foi capaz de classificar falhas na superfície do tecido, com erros em torno de 2,5%. Os autores também concluíram que melhorias nestes resultados poderiam ser obtidos através de um vetor feature de menor dimensionalidade, além de uma boa quantidade de exemplos no processo de aprendizado.

HOU et al. (1993) descreveram um sistema de inspeção visual para montagem de dispositivos eletrônicos, utilizando rede neural do tipo *backpropagation,* composto por um estágio de pré-processamento, no qual é feita a detecção de borda da imagem da peça, gerando o contorno (1 pixel de espessura) da peça a ser inspecionada. Estes dados são então processados, através de um módulo baseado na Transformada de Houg, no qual são detectados segmentos de reta do conjunto de elementos de borda gerados anteriormente, realizando uma compressão significativa do conjunto de dados originais. Através de uma Rede Neural *Backpropagation,* os dados obtidos através da Transformada de Hough são analisados e uma tomada de decisão é feita, baseada em dados de treinamento prévio da mesma. Resultados experimentais mostraram que o sistema tem uma maior acuracidade de detecção de padrões quando comparado ao método de *template matching*.

Uma área interessante para aplicação de sistema de inspeção visual automática, a de tubos não ferrosos, é apresentada por CHARLTON (1993). Esta área normalmente envolve a interpretação manual das peças e o uso de redes neurais resulta em um sistema automático, com grande capacidade de reconhecimento e baixíssimo índice de erro.

McMICHAEL (1994) utilizou um classificador neural (Parzen/Soft Perceptron) em um sistema de inspeção visual automática, em ambiente de FMS, aplicando o conceito de B*ayesian real time network* para trabalhar com requisitos de invariância a translação, rotação e escalonamento (TRS), utilizando como pré-processador a transformação poligonal, a qual fornece informações sobre linhas e círculos em imagens bi-dimensionais.

RAY & MAJUMDER (1994) desenvolveram um sistema de visão artificial, baseado em Redes Neurais, para reconhecer e localizar objetos 3-D parcialmente ocultos presentes em uma imagem. Utilizaram um modelo de Rede Neural binária bi-dimensional de Hopfield, sendo que os dados de entrada (modelos de objetos 3-D) são gerados usando o conceito de perspectiva canônica e provaram que o sistema apresenta diversas vantagens comparado aos que utilizam métodos tradicionais.

Outra área onde o uso de redes neurais vem sendo investigado é no reconhecimento de defeitos em cerâmicas. STINSON et al. (1994) utilizaram uma Rede Neural *Backpropagation* para reconhecer padrões de superfície, através de feixe de laser. Os resultados mostraram a eficiência da abordagem de Redes Neurais, quando comparada aos métodos tradicionais utilizados na inspeção de produtos da indústria cerâmica.

KARATHANASSI et al. (1996) apresenta um sistema de visão artificial para o controle de qualidade de soluções farmacêuticas, utilizando redes neurais artificiais.

COULOT, et al. (1996) apresentam um sistema o qual permite controlar objetos metálicos pelas medidas obtidas através de visão artificial. O sistema consiste de uma câmera simples e o algoritmo é baseado na relação entre as orientações da superfície do objeto e o nível de tons de cinza dos pixels correspondentes. Esta relação permite que o perfil do objeto seja reconstruído e que as dimensões do mesmo possam ser controladas. Os autores também tratam da avaliação de acuracidade de medida, da origem de erros e das limitações do algoritmo.

SONG & CHANG (1996) apresentam o estudo experimental de um controlador neural para robô manipulador, o qual pode rastrear um objeto móvel usando informação visual. O sistema proposto integra dados visuais (câmera de CCD) em uma Rede Neural Artificial, eliminando a necessidade da transformação de coordenadas globais para coordenadas de robô.

YU et al. (1996) apresentam um sistema de visão artificial aplicado em Veículos Guiados Automaticamente - AGV's. Duas imagens são tomadas, ao mesmo tempo, da cena em questão, por duas câmeras e então transmitidas ao Computador. Através de um algoritmo próprio, a distância entre o alvo e câmera é calculada.

TIANXU et al. (1996) descrevem o reconhecimento automático de objetos. Baseado no domínio de conhecimento e princípios de organização perceptual, um algoritmo heurístico de dois estágios é proposto, o qual extrai e identifica pequenos objetos de localização desconhecida. Resultados experimentais mostraram a efetividade e praticidade do método proposto.

HWANG et al. (1996) apresenta um sistema híbrido de processamento de imagens, utilizando uma rede neural artificial para aumentar a robustez do processo. O sistema é composto de três procedimentos: pré-rede, rede de segmentação e pós-rede. O esquema de treinamento da rede e o desempenho de classificação foram analisados e apresentados.

Uma área de bastante interesse, a qual relaciona visão computacional, técnicas de Redes Neurais e controle de robôs é o Rastreamento Visual. Tais sistemas utilizam algoritmos que endereçam a informação visual de objetos 3D no espaço 2D, combinando técnicas de visão computacional que detectam e medem o movimento com estratégias simples de controle. A arquitetura básica de tais sistemas é mostrada na figura 4.22, a seguir.

Neste modelo, a interatividade entre informação visual e informações de controle é evidente, resultando em movimentos controlados por um modelo dinamicamente controlado por informações visuais.

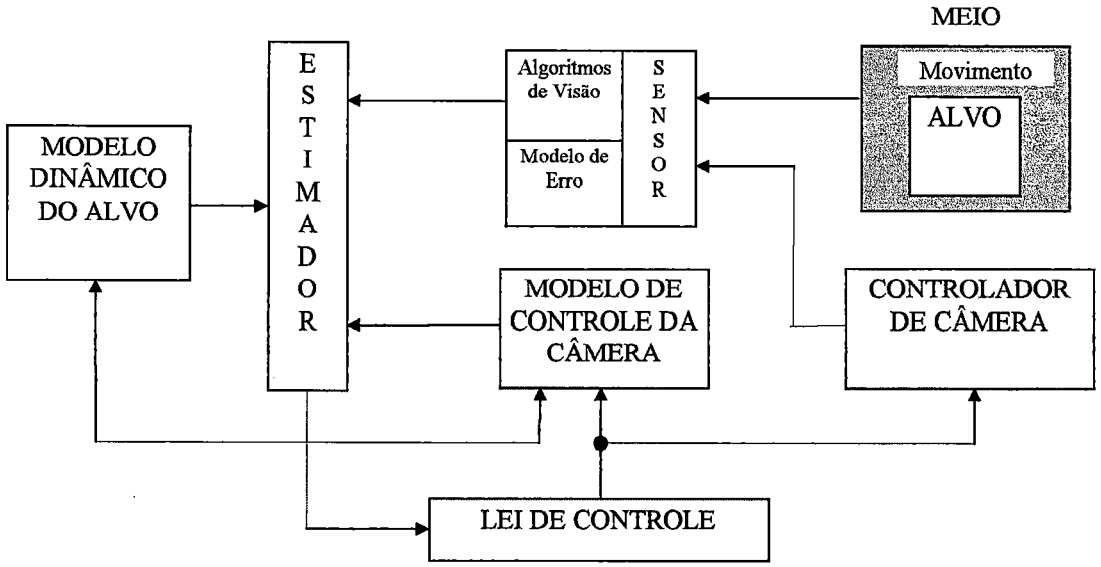


Figura 4.22. Arquitetura básica de um sistema de Rastreamento de Alvo (PAPANIKOLOPOULOS et al., 1993).

Sistemas de rastreamento visual, utilizando diversos modelos de Redes Neurais, são apresentados por INIGO et al. (1993), DROR et al. (1995) e COOPERSTOCK & MILIOS (1993). Estes trabalhos descrevem sistemas que perseguem um determinado alvo, através da utilização de modelos de Redes Neurais Artificiais, utilizados em robôs e que podem ser utilizados em diversas atividades, dentre as quais as de montagem automatizada.

ROSANDICH (1997) propôs uma nova arquitetura, utilizando redes neurais, desenvolvida especificamente para reconhecimento de padrões bi-dimensionais. Esta rede utiliza um conceito de similaridade, baseado na distância Hausdorff, para determinar o grau de semelhança entre um padrão de entrada e uma representação aprendida. O uso deste conceito leva a um comportamento mais consistente com o desempenho humano quando comparado com outras abordagens. Resultados mostram a efetividade do novo modelo.

FORESTI & PIERONI (1997) apresentaram um sistema de reconhecimento de objetos baseado em dois estágios de Redes Neurais, permitindo o reconhecimento 3D. O primeiro estágio extrai características locais das superfícies, as quais são utilizadas pelo segundo estágio na classificação. Estas características locais representam as classes de superfícies utilizadas no estágio de reconhecimento.

Um sistema de reconhecimento de imagens, baseado no modelo de Hopfield de Rede Neural foi apresentado por YOUNG et al. (1997). O modelo utiliza várias camadas de redes neurais em cascata, sendo que cada camada codifica características distintas do objeto, com diferentes resoluções. Resultados experimentais são apresentados e avaliados.

TSAI et al. (1998) desenvolveram um sistema de visão artificial, baseado em Redes Neurais, para avaliação de rugosidade de superfície em peças usinadas, o qual permite a inspeção da superfície sem qualquer contato com a mesma. Resultados experimentais apresentados mostraram que o uso da técnica de Redes Neurais proporcionou flexibilidade ao sistema de inspeção de rugosidade da superfície das peças.

PAUL et al. (1998) utilizaram neurônios ADALINE para implementar um sistema de reconhecimento de imagens, com tempo de treinamento extremamente baixo. O sistema, utilizando um modelo *fuzzy* de ADALINE, apresentou características de invariância geométrica (translação, escala, rotação e tamanho das imagens). Os resultados apresentados comprovam a viabilidade do sistema.

A característica de invariância geométrica também foi apresentada por KUNCHEV & TOPALOVA (1998), em um sistema de reconhecimento de objetos baseado em Backpropagation, o qual utiliza três estágios de Rede Neural e por SIM & DAMPER (1997), com um sistema utilizando mapas de Kohonen.

JONSSON et al. (1998) apresentam um sistema de visão de robô utilizando Redes Neurais e Máquinas de Estado em esquema de cooperação. O sistema é capaz de navegar de forma autônoma, baseado nas informações do sistema visual. O controlador baseado em Redes Neurais recebe a imagem de entrada sem qualquer tipo de pré-processamento e produz a saída adequada para os motores do robô.

Outros trabalhos, envolvendo sistemas de visão ativa de robôs e de reconhecimento de objetos podem ser vistos em SHARMA & SRINIVASA (1998), BRAUNL (1997), SIANG & MING (1997), PAULI (1998), WONG et al. (1998) e SAITO & KIMURA (1996).

4.5 CONSIDERAÇÕES FINAIS

Neste capítulo foi feita uma introdução à Visão Computacional, através da descrição dos principais conceitos relacionados ao tema. O problema do Reconhecimento de Padrões foi então apresentado, através da descrição das principais abordagens possíveis. Foi dada ênfase especial ao problema do Reconhecimento de Padrões utilizando Redes Neurais Artificiais, o qual foi descrito em detalhes, além da apresentação de trabalhos recentes realizados na área. O capítulo é finalizado com a apresentação de trabalhos relativos à implementação de Visão Computacional para Robôs.

No capítulo 5, a seguir, conceitos apresentados neste capítulo e nos capítulos anteriores serão utilizados para a implementação de um sistema de visão para robô, a ser utilizado em operações de montagem automatizada.

5. SISTEMA DE RECONHECIMENTO DE PEÇAS, BASEADO EM REDES NEURAIS, PARA ROBÔ DE MONTAGEM

5.1. INTRODUÇÃO

O presente trabalho teve como objetivo o desenvolvimento de um sistema capaz de reconhecer imagens, as quais estão relacionadas às aplicações envolvendo manipulação de peças (robôs, manipuladores, etc.). Basicamente, buscou-se um sistema que fosse capaz de reconhecer peças mecânicas, independente de sua posição, escala ou ângulo de rotação. Tal requisito implicou na construção de invariância geométrica no sistema, através da implementação de um módulo de pré-processamento dentro do módulo de reconhecimento.

Outro requisito levado em conta no desenvolvimento do sistema foi a possibilidade de trabalhar com grandes imagens de entrada, ou seja cenas compostas por 512 x 512pixels, 1024 x1024 pixels, ou maiores. Para tal, um esquema de *coarse coading* foi utilizado, possibilitando a utilização de imagens com tais dimensões, além de economia de memória e maior agilidade de processamento.

Finalmente, o principal requisito imposto no desenvolvimento do sistema foi o de que ele deveria ser apropriado para o processamento paralelo, particularmente para ser utilizado na arquitetura CPAD desenvolvida por OSHIRO (1998).

Para alcançar tais objetivos, foi feito um planejamento envolvendo o desenvolvimento de módulos. Tais módulos, atuando de forma integrada, realizam as diversas funções relativas à aquisição, processamento e reconhecimento das imagens a serem utilizadas pelo sistema de manipulação, operando com grandes imagens no campo de entrada, as quais podem variar geometricamente neste campo, ou seja, podem estar em posições, escalas e orientações diferentes.

Os módulos foram implementados utilizando como plataforma um Microcomputador Pentium, no qual foi acoplado um conjunto Câmera CCD/Frame Grabber. A figura abaixo mostra esquematicamente o sistema.

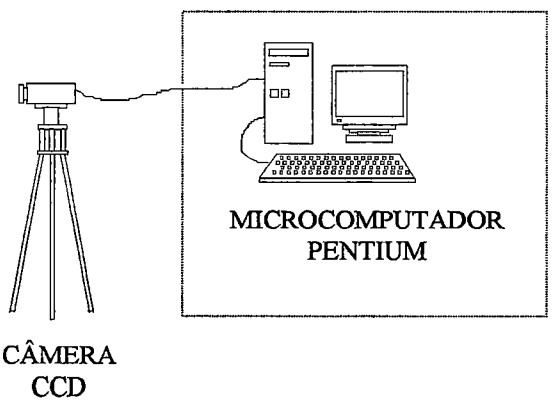


Figura 5.1. Configuração do Sistema.

No sistema desenvolvido, a Câmera está ligada ao Microcomputador, através de uma Frame Grabber. No microcomputador, através da utilização de módulos de processamento, a imagem é capturada (pela Frame Grabber) e, após processamento, códigos representando a classe de objeto reconhecida são gerados. Estes códigos são utilizados pelo mecanismo de manipulação como parâmetros de controle de movimento.

Na figura a seguir é mostrada esta configuração.

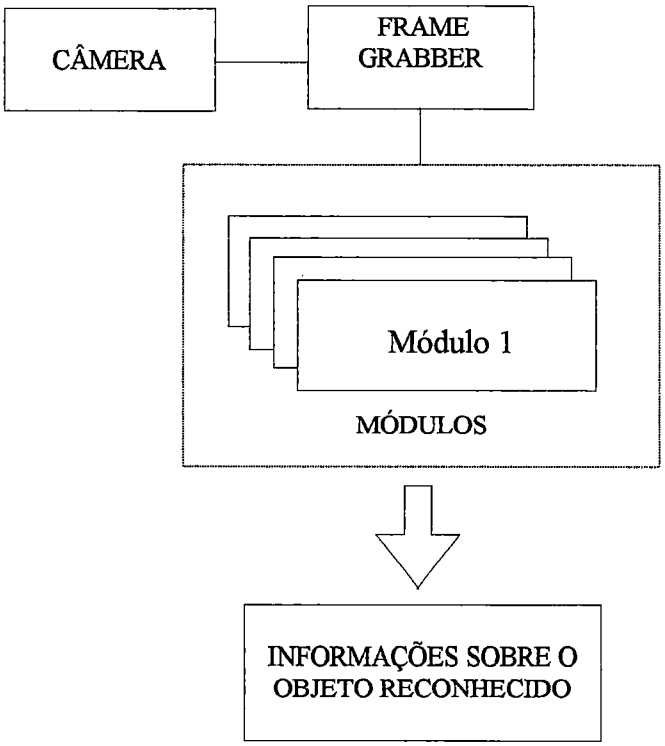


Figura 5.2. Funcionamento do Sistema.

Baseado nesta proposta inicial de sistema, foram implementados diversos módulos, com funções específicas, os quais realizam tarefas referentes ao processamento e reconhecimento da imagem obtida através da Câmera. A descrição da implementação de tais módulos é mostrada a seguir.

5.2. ARQUITETURA DO SISTEMA DE RECONHECIMENTO DE PEÇAS

O sistema básico para o reconhecimento de peças, como mostrado na figura 5.3, é constituído por quatro módulos:

- Módulo de Aquisição de Dados;
- Módulo de Segmentação / Extração de Características;
- Módulo de Reconhecimento:
- Módulo de Controle.

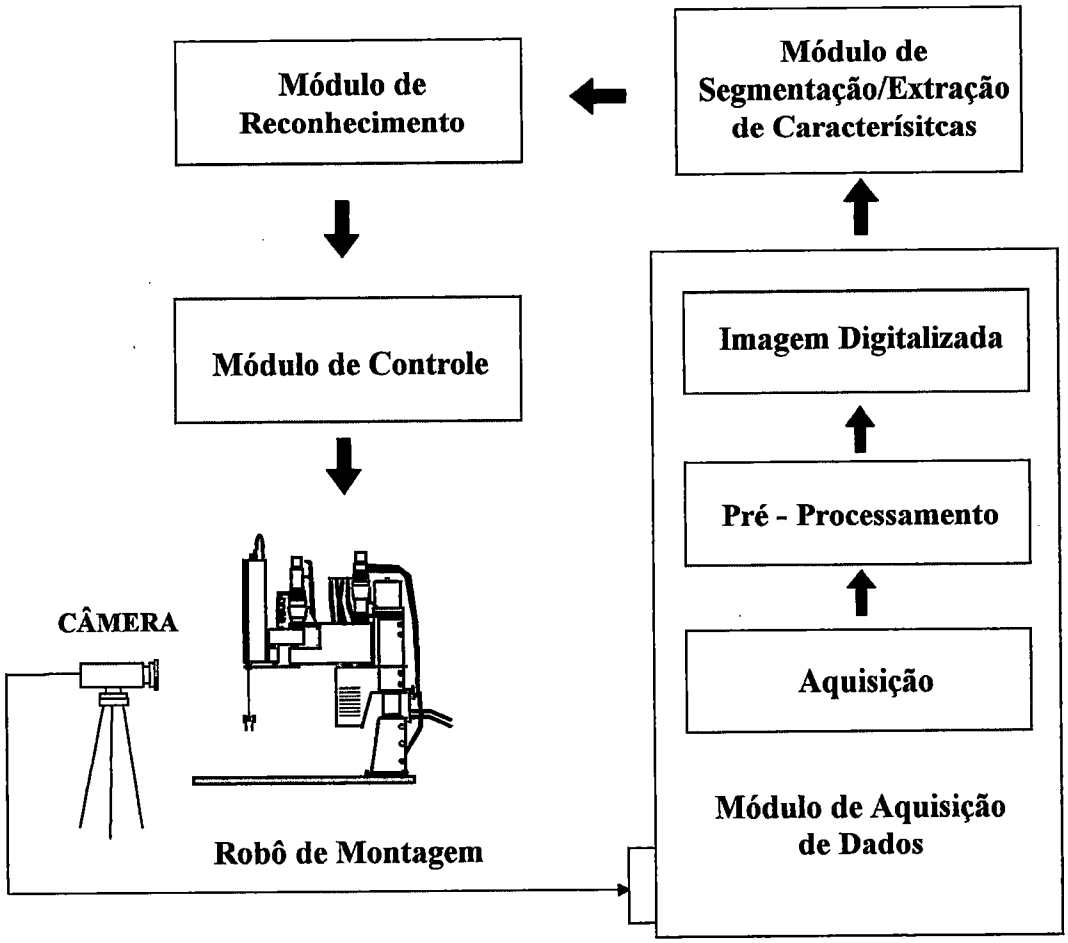


Figura 5.3: Módulos do Sistema.

Imagens de 128 x 128 pixels de diversas peças mecânicas fazem parte do banco de dados utilizado neste trabalho e são apresentadas nos ítens subsequentes. A seguir, alguns exemplos destas imagens são mostrados.

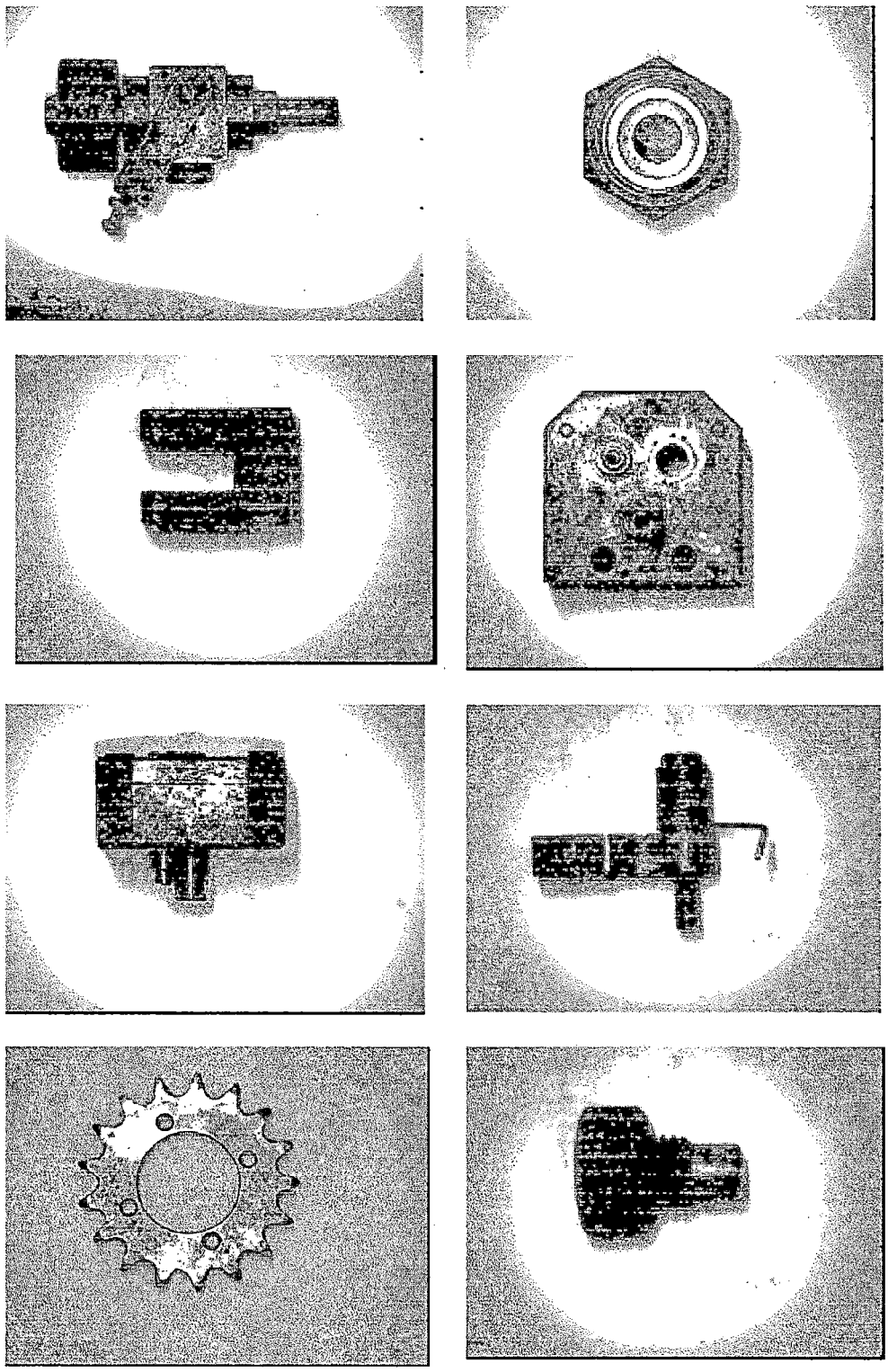


Figura 5.4. Exemplos de imagens reais utilizadas pelo sistema.

5.3 Descrição dos Módulos

5.3.1 Módulo de Aquisição de Dados

É o responsável pela aquisição dos dados de imagem (analógicos), o quais serão pré-processados resultando em dados digitalizados.

Após o processo de aquisição, a imagem é filtrada, com o objetivo de ressaltar características da imagem original, o que resultará em maior facilidade de reconhecimento para os módulos subsequentes.

A aquisição da imagem é feita através de uma Frame Grabber, à qual é acoplada uma câmera de CCD.

Em conjunto com a Frame Grabber, foi instalado um pacote de *software* específico para o tratamento das informações obtidas através da Frame Graber, denominado ***Global Lab***, o qual além de permitir o gerenciamento da mesma, possibilita a manipulação de imagens, além de algumas operações sobre as mesmas. Assim, as operações realizadas nas imagens neste módulo, de Aquisição de Dados/Pré-Processamento, foram todas baseadas em rotinas presentes no ambiente ***Global Lab***. Algumas destas rotinas são mostradas a seguir.

Rotinas que manipulam informações sobre buffers (da Frame Grabber e do usuário):

buffer_new
buffer_free
buffer_get_info
buffer_set_info
buffer_getpix

Estas rotinas permitem a alocação de buffers, acesso a informações sobre a imagem armazenada no mesmo, além da transferência do conteúdo do buffer para uma região de memória pré-definida. A estrutura básica do tipo buffer_info (geral) é apresentada a seguir.

```
struct buffer_info {
        int classno;
        struct rect bounds;
        int bits_per_pixel;
        long base_addr;
        int pixels_per_row;
        int capabilities;
        int display;
        int check1;
        int check2;
        int check3;
        int check4;
        int checkbits;
        HANDLE hMono;
        HANDLE hColor;
        int update;
        char name[40];
};
```

Após alocada nos buffers da Frame Grabber e/ou memória, a imagem pode ser manipulada através das rotinas de pré processamento.

Filtro: a fim de suavizar a imagem e corrigir pequenas distorções, aplica-se o filtro Mediana, cuja rotina básica é mostrada no Apêndice IV. Adicionalmente, outros filtros também podem ser aplicados (diversos tipos de filtros foram adicionados ao módulo, os quais podem ser utilizados como rotinas de pré-processamento da imagem original).

5.3.2 Módulo de Segmentação/Extração de Características

É o responsável pela compressão de dados da imagem original, priorizando informações relevantes e descartando as demais informações contidas na imagem.

Para tal foram desenvolvidas as rotinas descritas a seguir.

Binarização: A binarização é um processo de segmentação dos objetos contidos na imagem e é baseada em uma mudança dos níveis de cinza de acordo com uma função na qual seu conjunto imagem possui apenas dois valores possíveis. A binarização separa a informação (objetos) do resto da imagem. Após esta operação, teremos uma imagem que possui apenas dois níveis de cinza, o "0", preto, que indica a presença de objetos, e o "G-1", que indica o fundo da imagem (background), o qual normalmente não tem importância nas tarefas de reconhecimento de imagens. A rotina implementando tal função foi desenvolvida baseada no seguinte algoritmo:

```
Parâmetro: Valor_limiar
For 1=1 to N
  For j+1 to M
    If imagem[i][j] < Valor_limiar
        imagem[i][j] = 0;
    Else
        imagem[i][j] = G-1;
  end
end
```

O valor do limiar varia de acordo com as classes de objetos a serem reconhecidos pelo sistema e foram determinados empiricamente.

Introdução de Ruído: Com o objetivo de tornar o sistema tolerante a ruído, foi introduzido ruído nas imagens utilizadas para treinamento do sistema (degradação das imagens) com uma quantidade controlada de pixels. Este ruído é introduzido segundo o algoritmo a seguir.

```
float ruido_desejado;
int i,j;
{
for(i=0;i<NX,i++)
    for(j=0;j<Ny<j++)
        if(rand()%<ruido_desejado)
            inverter_pixel(i,j,imagem);
}
```

A figura 5.5, a seguir, ilustra o funcionamento do módulo de Segmentação/Extração de Características, através de exemplos de imagens processadas.

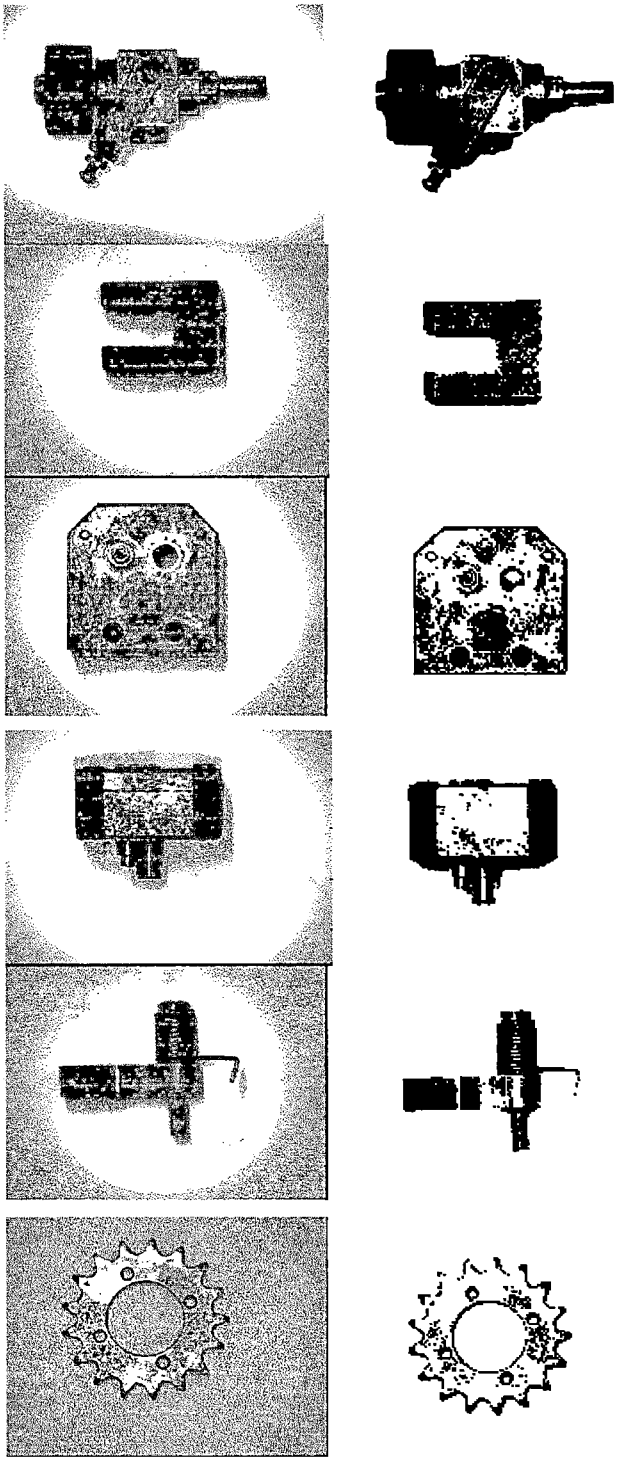


Figura 5.5. Imagens Reais e Imagens binarizadas.

5.3.3 Módulo de Reconhecimento

O módulo de Reconhecimento é o responsável pela classificação das imagens apresentadas ao sistema. Para implementação deste módulo, dois sub-módulos foram definidos:

- Sub-módulo Pré-Processador;
- Sub-módulo Classificador.

O **sub-módulo Pré-Processador**, referenciado a partir de agora somente como **Pré-Processador** tem como objetivo principal proporcionar correções de rotação, escala e translação antes do processo de classificação, proporcionando assim características de invariância geométrica ao sistema. Tal invariância pode ser obtida se o Pré-Processador apresentar as seguintes propriedades (YÜCEER & OFLAZER, 1993):

- O Pré-Processador deve mapear os padrões distorcidos por transformações ou ruído para um padrão de saída razoavelmente estável;
- O algoritmo de mapeamento deve ser de fácil computação, ou seja, não deve aumentar a complexidade do sistema global de reconhecimento.

A seleção de boas características para padrões complexos é uma tarefa impraticável ou algumas vezes impossível. O problema torna-se mais complexo ainda quando não há conhecimento a priori sobre os padrões a serem classificados. No entanto, um sistema para extrair automaticamente as características úteis é essencial. Tal tarefa é realizada por Redes Neurais Artificiais durante o processo de treinamento e será vista no bloco de classificação.

A função do Pré-Processador, como mostrado anteriormente, é proporcionar invariância rotacional, em escala e translação na imagem de entrada. Tanto a entrada como a saída do Pré-Processador são imagens na forma de mapa de pixels.

A estrutura do Pré-Processador tem três blocos cascateados, a saber: Bloco de Rotação, Bloco de Escala e Bloco de Translação, conforme mostrado na figura 5.6, a seguir.

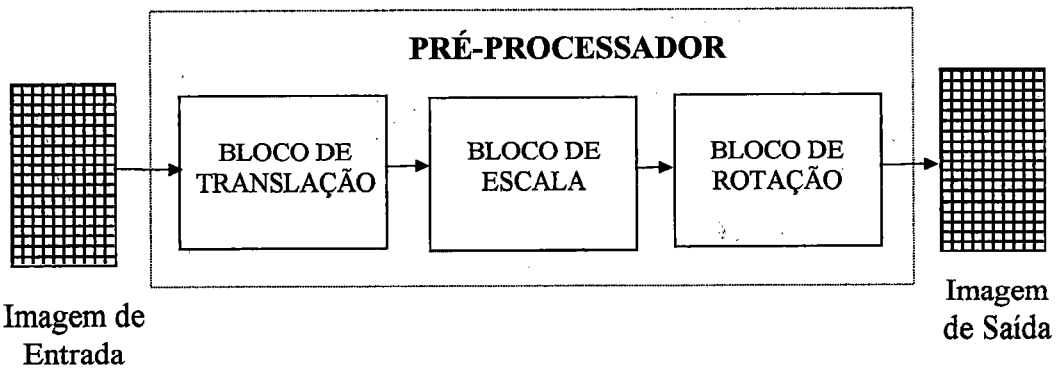


Figura 5.6. Estrutura do Pré-Processador (YÜCEER & OFLAZER, 1993).

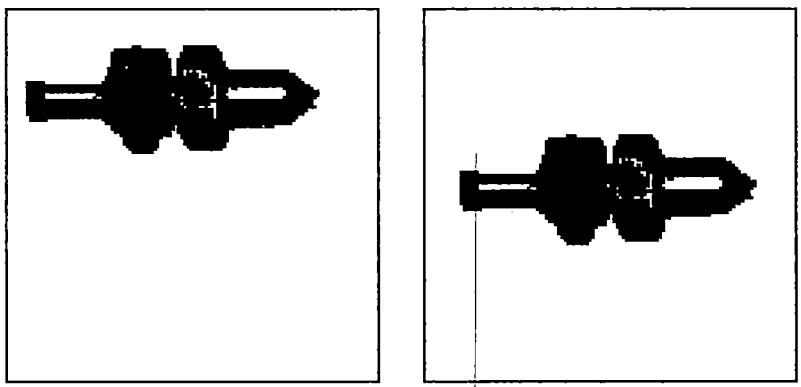
O Bloco de Rotação mantém invariância rotacional, o Bloco de Escala mantém invariância ao escalonamento e o Bloco de Translação mantém invariância translacional.

A ordem na qual o blocos são cascateados é determinada pelas dependências funcionais entre os blocos, sendo o primeiro bloco o de translação, seguido pelo de escala e finalmente pelo de rotação. Como as operações de escalonamento e rotação necessitam de um ponto (pivô) apropriado, o Bloco de Translação é posicionado antes dos dois outros blocos. A origem da imagem de saída deste bloco será o ponto pivô para os blocos de Escala e Rotação.

A seguir, é feita uma descrição dos blocos.

Bloco de Translação

O bloco de Translação mantém a invariância translacional computando o centro de gravidade do padrão e transladando a imagem de tal forma que o centro de gravidade coincida com a origem. A imagem resultante é então passada ao Bloco de Escala. O centro de gravidade é calculado pela média das coordenadas x e y dos pixels "1", conforme mostrado a seguir.



(a) (b)

Figura 5.7. Imagem de Entrada (a) e Imagem após passar pelo Bloco de Translação (b).

Para a implementação deste bloco, foi utilizada a seguinte formulação:

Seja P o número de pixels "1" na imagem,

$$P = \sum_{i=1}^{N} \sum_{j=1}^{N} f(x_i, y_j) \qquad (5.1)$$

Assim, o Centro de Gravidade (x_{av},y_{av}) será dado por:

$$x_{av} = \frac{1}{P}\sum_{i=1}^{N}\sum_{j=1}^{N} f(x_i, y_j).x_i \quad , \quad y_{av} = \frac{1}{P}\sum_{i=1}^{N}\sum_{j=1}^{N} f(x_i, y_j).y_j \qquad (5.2)$$

onde *f(x,y)* fornece o valor do pixel para as coordenadas (x,y), ou seja, 0 ou 1.

A função de mapeamento, para que a invariância translacional possa ser obtida é dada por:

$$f_T(x_i, y_j) = f(x_i + x_{av}\ , y_j + y_{av}) \qquad (5.3)$$

Como neste trabalho as imagens são de 128 x 128 pixels, a forma como a função de mapeamento é implementada é mostrada a seguir, com uma grade de 128 x 128 posições (pixels).

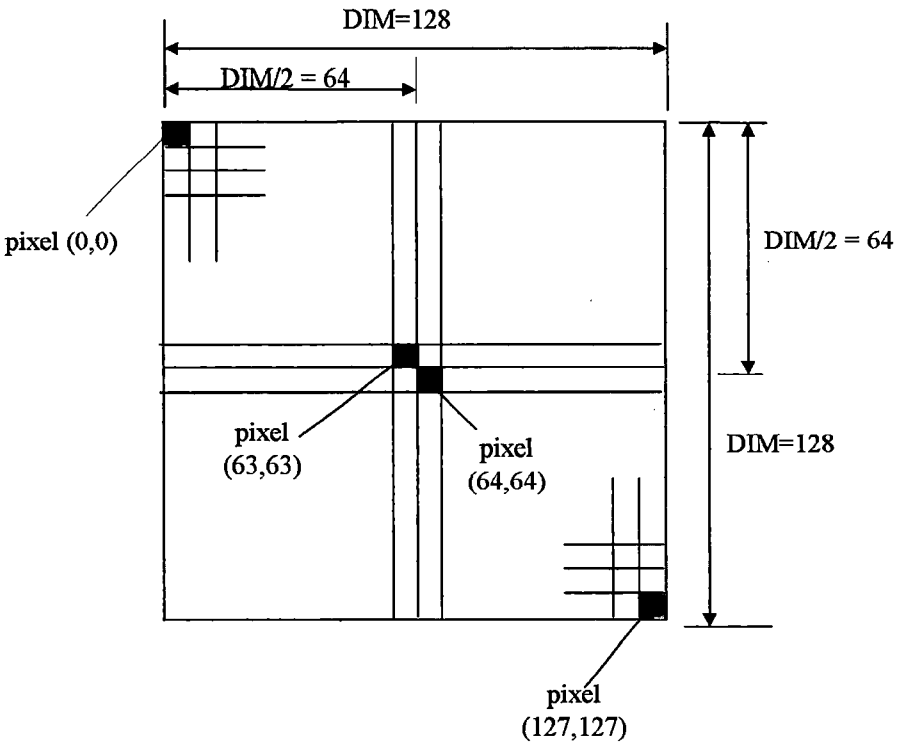


Figura 5.8. Grade de Imagem.

Para o cálculo do centro de gravidade da figura presente na grade, foi utilizado um sistema de coordenadas cuja origem está no centro da grade, ou seja, nas coordenadas originais (DIM/2, DIM/2), onde DIM é a dimensão da grade (no presente trabalho esta dimensão foi fixada em 128).

Portanto, em função deste sistema de coordenadas, foram definidos quatro quadrantes na grade original, conforme mostrado na figura a seguir.

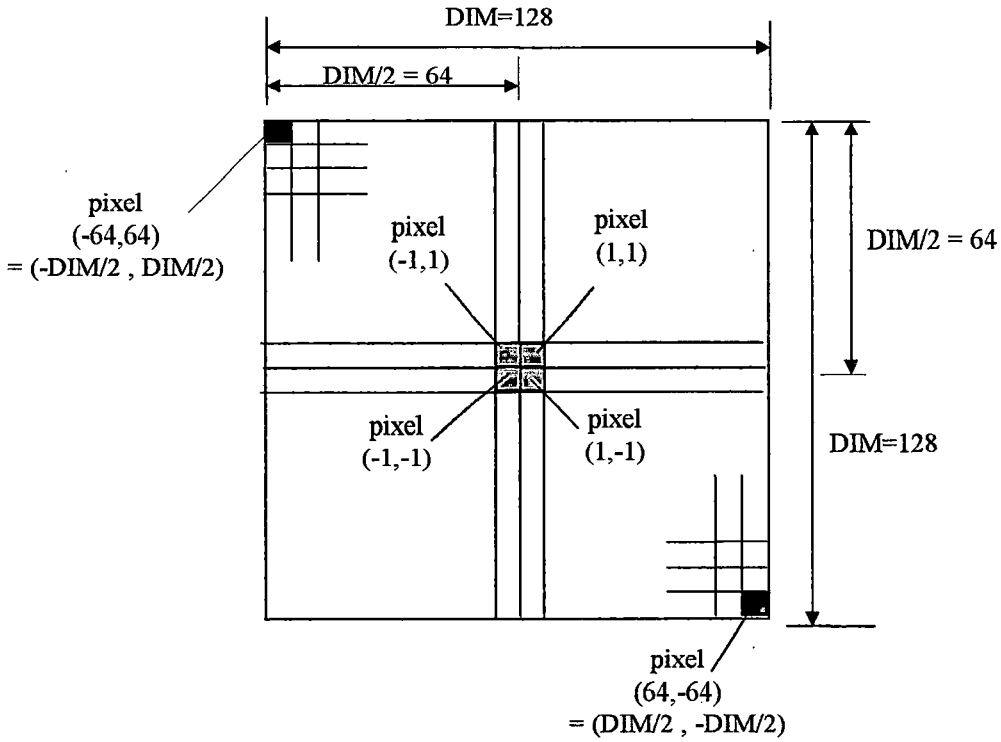


Figura 5.9. Sistema de Coordenadas baseado na origem da grade.

Com este sistema de coordenadas, o cálculo do centro de gravidade foi feito através de rotina como a mostrada a seguir.

```
for(i = 0; i < DIM; i++)
{
        for(j=0; j<DIM; j++)
        {
                if (A[i][j]=1)
                {
                        p++;                    // número de pixels "1"
                        Xav = (i-DIM/2) + Xav;
                        Yav = (DIM/2-j) + Yav;
                }
        }
}

Xav = int((1/p)*Xav);     // coordenadas do centro de gravidade no sistema
Yav = int((1/p)*Yav);              de coordenadas baseado no centro da grade

x = Xav-DIM/2;            // coordenadas reais do centro de gravidade
y = Yav-DIM/2;
```

Após calculadas as coordenadas do centro de gravidade, a saída do bloco de Translação é fornecida, mapeando os pixels originais (da imagem de entrada) para uma nova imagem, na qual o centro de gravidade da imagem original seja coincidente com o centro da grade. Tal mapeamento pode ser visto através da rotina mostrada a seguir.

```
for(i = 0; i < DIM; i++)
{
        for(j = 0; j < DIM; j++)
        {
                if (A[i][j]==1)
                        {
                        Xt=(i-DIM/2)-Xav;
                        Yt=(DIM/2-j)-Yav;
                        it=int(DIM/2+Xt);
                        jt=int(DIM/2+Yt);
                        B[it][jt] = A[i][j];
                        }
        }
}
```

Assim, a imagem resultante, e que será a imagem de entrada para o bloco de Escala, é uma imagem com as mesmas dimensões da original mas que foi processada de tal forma que a figura originalmente presente foi transladada para uma posição em que seu centro de gravidade coincida com o centro da grade.

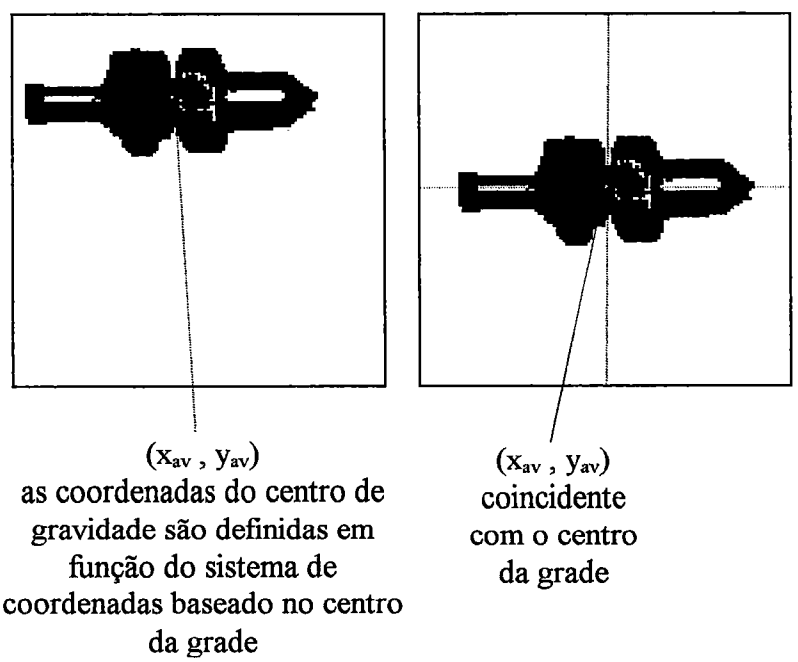


Figura 5.10. Imagem de saída do Bloco de Translação.

Bloco de Escala

O bloco de Escala mantém a invariância de escalonamento modificando a escala da imagem de entrada (proveniente do bloco de translação) de tal maneira que o raio médio para os pixels "1" seja igual a uma fração pré-determinada da grade, ou seja, qualquer que seja o tamanho da figura presente na imagem de entrada, ela será re-mapeada para uma imagem de saída do bloco, a qual conterá a figura da imagem de entrada com um tamanho igual a uma fração da grade. Neste trabalho, esta fração foi fixada em um quarto do tamanho da grade (128/4).

O raio de um determinado pixel é definido como o tamanho da reta que o liga à origem (ponto central da grade). Assim, o raio médio pode ser calculado como segue.

$$r_{av} = \frac{1}{\sum_{i=1}^{N}\sum_{j=1}^{N} f_T(x_i , y_j)} \sum_{i=1}^{N}\sum_{j=1}^{N} f_T(x_i , y_j).\sqrt{(x_i^2 + y_j^2)} \qquad (5.4)$$

O fator de escala, s, é calculado como:

$$s = \frac{r_{av}}{R} \qquad (5.5)$$

onde R é a fração da grade (128/4 neste trabalho).

A função de mapeamento, ou seja, aquela que produz a nova imagem, com a figura original escalada, é mostrada a seguir.

$$f_{TS}(x_i, y_j) = f_T(s.x_i , s.y_j) \qquad (5.6)$$

Através desta equação, os pixels da imagem de saída são mapeados levando em conta os pixels correspondentes na imagem de entrada (proveniente do bloco de translação). Assim, um pixel da imagem original pode gerar um conjunto de pixels na imagem de saída (no caso de aumento de tamanho da imagem original). Por outro lado, diversos pixels da imagem de entrada podem dar origem a um número menor de pixels ou mesmo um único pixel da imagem de saída (no caso de redução de tamanho da imagem original). Este tipo de mapeamento envolve propriedades de interpolação, como pode ser visto na equação 5.6, a qual descreve os pixels da imagem de saída, em função dos pixels da imagem de entrada e levando em conta a utilização de um fator de escala.

A figura a seguir mostra o resultado de mapeamento fornecido pelo bloco de escala.

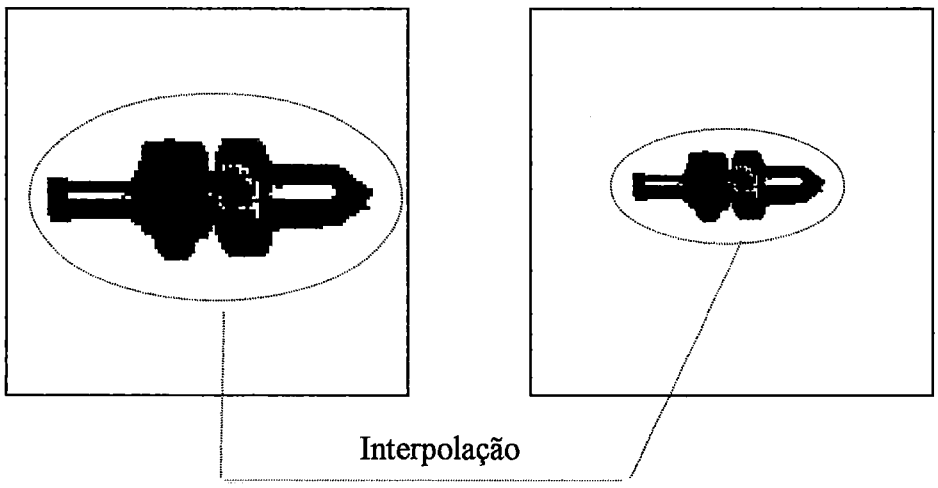


Figura 5.11. A imagem original (proveniente do bloco de translação) após processamento pelo bloco de escala.

Como mostrado na figura anterior, a imagem original foi mapeada para a imagem de saída de tal maneira que esta possui as mesmas características da original, com um fator de escala diferente. Através deste bloco, todas as imagens presentes na entrada do sistema gerarão imagens de saída no bloco de escala com o mesmo rio médio. Esta característica proporciona melhores resultados no estágio de classificação.

As rotinas implementadas para realizar esta operação, ou seja, a função de mapeamento do bloco de escala foram baseadas na rotina mostrada a seguir.

```
for(i=0; i< DIM; i++)
 {
                for (j=0; j< DIM; j++)
                {
                        if (B[i][j]==1)
                                {
                                k++;
                                z = (i-DIM/2)*(i-DIM/2);
                                w = (DIM/2-j)*(DIM/2-j);
                                m = z + w;
                                Rav = (sqrt(m)) + Rav;
                                }
                }
 }
Rav = (1/p)*Rav;
```

```
// Calculo do fator de escala

        R = DIM/4;
        s = (1/(0.25*DIM))*Rav;


        // Funcao de mapeamento

        for(i = 0; i < DIM; i++)
        {

                for(j = 0; j < DIM; j++)
                {
                        if (B[i][j]==1)
                                {
                                 p1++;
                                 sx = int(s*(i-DIM/2));
                                 sy = int(s*(DIM/2-j));

                                 A[DIM/2+sx][DIM/2-sy] = 1;
                                 }
                }
}
```

Bloco de Rotação

O bloco de rotação mantém invariância rotacional rotacionando a imagem de tal maneira que a direção de máxima variância coincida com o eixo x. Esta transformação, baseada na transformação de Karhunen-Loéve utiliza os seguintes conceitos (YUCEER & OFLAZER, 1993): dado um conjunto de vetores, o *eigenvector* que corresponde ao maior *eingenvalue* da matriz covariância calculada do conjunto de vetores, aponta para a direção de máxima variância. Tal propriedade pode ser utilizada para manter invariância rotacional já que a detecção da direção de máxima variância também revela o ângulo de rotação.

Utilizando-se de vetores 2D formados pelas coordenadas dos pixels "1" na imagem, os *eigenvalues* podem ser calculados como segue (YUCEER & OFLAZER, 1993).

Sendo

$$\begin{bmatrix} m_x \\ m_y \end{bmatrix} = \frac{1}{\sum_{i=1}^{N}\sum_{j=1}^{N} f_{TS}(x_i, y_j)} \sum_{i=1}^{N}\sum_{j=1}^{N} f_{TS}(x_i, y_j) \begin{bmatrix} x_i \\ y_j \end{bmatrix} \qquad (5.7)$$

$$P = \sum_{i=1}^{N} f_{TS}(x_i, y_j) \tag{5.8}$$

$$T_{xx} = \sum_{i=1}^{N} \sum_{j=1}^{N} f_{TS}(x_i, y_j).x_i^2 \tag{5.9}$$

$$T_{yy} = \sum_{i=1}^{N} \sum_{j=1}^{N} f_{TS}(x_i, y_j).y_j^2 \tag{5.10}$$

$$T_{xy} = \sum_{i=1}^{N} \sum_{j=1}^{N} f_{TS}(x_i, y_j).x_i\, y_j \tag{5.11}$$

A matriz covariância pode então, após simplificações, ser definida como:

$$C = \left(\frac{1}{\sum_{i=1}^{N}\sum_{j=1}^{N} f_{TS}(x_i, y_j)} \sum_{i=1}^{N} \sum_{j=1}^{N} f_{TS}(x_i, y_j)\, x \begin{bmatrix} x_i \\ y_j \end{bmatrix} \begin{bmatrix} x_i \\ y_j \end{bmatrix}^T\right) - \begin{bmatrix} m_x \\ m_y \end{bmatrix} \begin{bmatrix} m_x \\ m_y \end{bmatrix}^T \tag{5.12}$$

Como foi mantida a invariância à translação nos blocos anteriores, m_X e m_Y são zero. Após eliminar o termo médio antes da matriz, a matriz covariância torna-se:

$$C = \begin{bmatrix} T_{xx} & T_{xy} \\ T_{xy} & T_{yy} \end{bmatrix} \tag{5.13}$$

Em função do que foi definido anteriormente e, levando em consideração as simplificações feitas, os valores de seno e cosseno do ângulo de rotação da figura presente na imagem de entrada (imagem de saída do bloco de escala) podem ser definidos como segue.

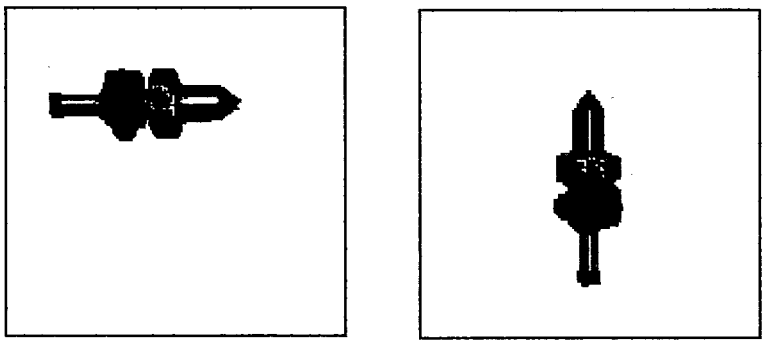
$$\operatorname{sen}\theta = \frac{(T_{yy} - T_{xx}) + \sqrt{((T_{yy} - T_{xx})^2 + 4.T_{xy}^2)}}{\sqrt{(8.T_{xy}^2 + 2.(T_{yy} - T_{xx})^2 + 2.(T_{yy} - T_{xx})\sqrt{((T_{yy} - T_{xx})^2 + 4T_{xy}^2)})}} \tag{5.14}$$

$$\cos\theta = \frac{2.T_{xy})}{\sqrt{(8.T_{xy}^2 + 2.(T_{yy} - T_{xx})^2 + 2.(T_{yy} - T_{xx})\sqrt{((T_{yy} - T_{xx})^2 + 4T_{xy}^2)})}} \tag{5.15}$$

Em função destes valores, de seno e cosseno, a função que faz o mapeamento da imagem de entrada para a imagem de saída, do bloco de rotação, pode ser descrita como segue.

$$f_{TSR}(x_i, y_j) = f_{TS}(\cos\theta.x_i - \text{sen}\,\theta.y_j\ ,\ \text{sen}\,\theta.x_i + \cos\theta.y_j) \qquad (5.16)$$

A figura 5.12, a seguir, mostra as imagens de entrada e de saída do bloco de rotação.



(a) (b)

Figura 5.12. Imagem de entrada (a) e imagem de saída do bloco de rotação.

Como pode ser visto na figura anterior, o bloco de rotação rotaciona a figura até que sua orientação coincida com o eixo *x*.

A implementação destas funções foi baseada na rotina mostrada a seguir.

```
//cálculo de Txx, Tyy e Txy

        for(i = 0; i < DIM; i++)
        {
                for(j = 0; j < DIM; j++)
                {
                 if (A[i][j] ==1)
                        {
                         Txx = ((i-DIM/2)*(i-DIM/2)) + Txx;
                         Tyy = ((DIM/2-j)*(DIM/2-j)) + Tyy;
                         Txy = ((i-DIM/2)*(DIM/2-j)) + Txy;
                        }
                }

        }
        //cálculo de senO e cosO
        senO = (Tyy - Txx) + (sqrt(((Tyy - Txx)*(Tyy - Txx)) + (4*(Txy*Txy))));
        senO = senO / (sqrt((8*(Txy*Txy)) + (2*(Tyy - Txx)*(Tyy - Txx)) + (2*(Tyy -
                Txx)*(sqrt(((Tyy - Txx)*(Tyy - Txx)) + (4*(Txy*Txy)))))));

        cosO = 2*Txy;
        cosO = cosO / (sqrt((8*(Txy*Txy)) + (2*(Tyy - Txx)*(Tyy - Txx)) + (2*(Tyy -
                Txx)*(sqrt(((Tyy - Txx)*(Tyy - Txx)) + (4*(Txy*Txy)))))));
```

```
//Funcao de Mapeamento

Xt=0;
Yt=0;

for(i = 0; i < DIM; i++)
{
        for(j = 0; j < DIM; j++)
        {
                if (A[i][j]==1)
                        {
                        Xt=(i-(DIM/2));
                        Yt=((DIM/2)-j);
                        rotx = int((cosO*Xt)-(senO*Yt));
                        roty = int((senO*Xt)+(cosO*Yt));
                        Xt=rotx;
                        Yt=roty;
                        B[(Xt+(DIM/2))][((DIM/2)-Yt)]= 1;
                        }
        }
}
```

A seguir, apresenta-se um exemplo das imagens resultantes dos blocos de pré-processamento descritos anteriormente.

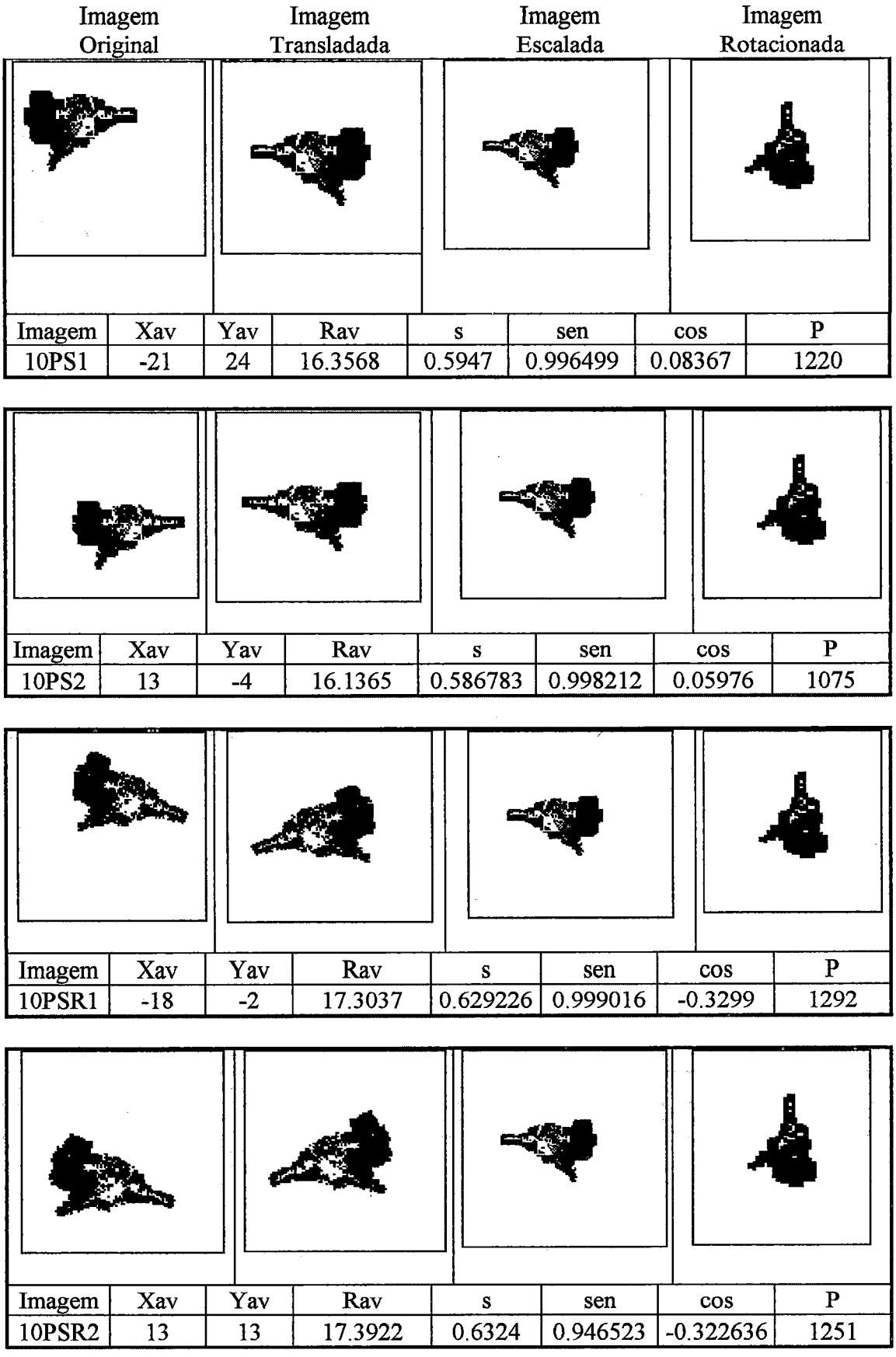


Imagem	Xav	Yav	Rav	s	sen	cos	P
10PS1	-21	24	16.3568	0.5947	0.996499	0.08367	1220

Imagem	Xav	Yav	Rav	s	sen	cos	P
10PS2	13	-4	16.1365	0.586783	0.998212	0.05976	1075

Imagem	Xav	Yav	Rav	s	sen	cos	P
10PSR1	-18	-2	17.3037	0.629226	0.999016	-0.3299	1292

Imagem	Xav	Yav	Rav	s	sen	cos	P
10PSR2	13	13	17.3922	0.6324	0.946523	-0.322636	1251

Figura 5.13. Exemplos de imagens resultantes dos Blocos de Pré-Processamento.

Sub-Módulo de Classificação

O sub-módulo de classificação, cuja função é a classificação das imagens resultantes do sub-módulo de pré-processamento, é composto por um bloco codificador (gerador de *coarse coading*) e um bloco de reconhecimento, baseado no modelo GSN de Rede Neural, o qual incorpora modificações propostas por MARTINS (1994), as quais foram discutidas no Capítulo 3.

A figura a seguir mostra esquematicamente a estrutura do sub-módulo de classificação.

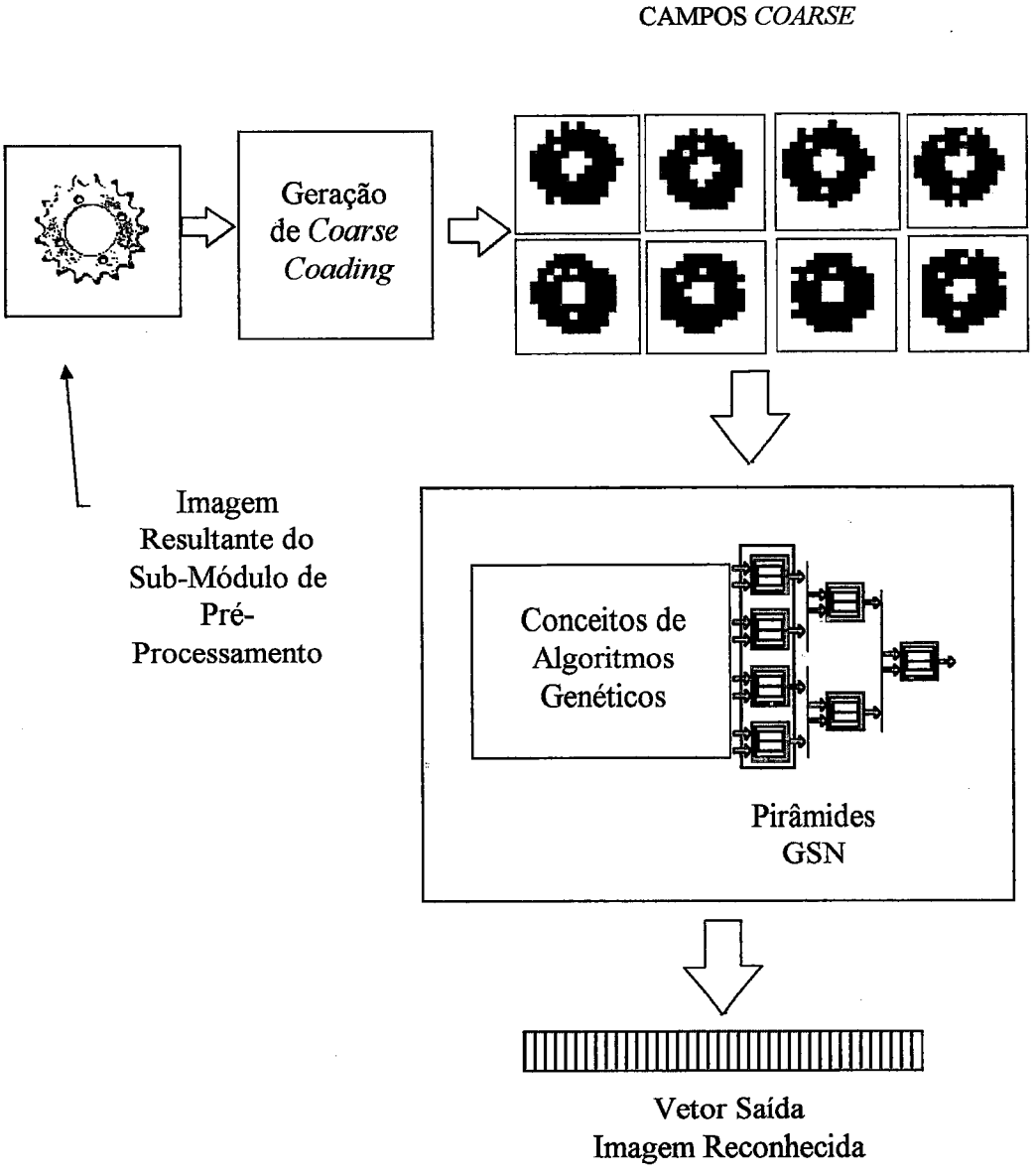


Figura 5.14. Sub-Módulo de Classificação.

Como mostrado na figura anterior, o sub-módulo de classificação é composto basicamente por um bloco gerador de *Coarse Coading* e um bloco de reconhecimento, baseado no modelo GSN.

O bloco gerador de *Coarse Coading* foi incorporado ao sistema com o objetivo de viabilizar a utilização de imagens com maiores dimensões (512 x 512, 1024 x1024, 2048x2048 pixels, etc.) as quais estão presentes em tarefas práticas de manipulação de peças reais.

A técnica de *Coarse Coading*, como descrita em SPIRKOVSKA & REID (1993) é uma variação da técnica de representação distribuída, na qual cada característica é representada por um padrão de atividade sobre muitas unidades. Usando unidades que são muito grosseiramente ajustadas, poucas unidades podem codificar muitas características precisamente. O número máximo de caraterísticas é determinado pela densidade e grau de sobreposição das unidades do campo receptivo.

Codificar uma imagem com um conjunto de imagens geradas através de *Coarse Coading* incrementa significativamente o tamanho do campo de entrada possível em um sistema de classificação de imagens.

O algoritmo de coarse coading utiliza campos sobrepostos para representar um campo de entrada composto de pixels menores. Esta técnica funciona de forma análoga aos campos da retina, os quais permitem uma hiperacuracidade controlada. A figura 5.15, a seguir, mostra esquematicamente a técnica.

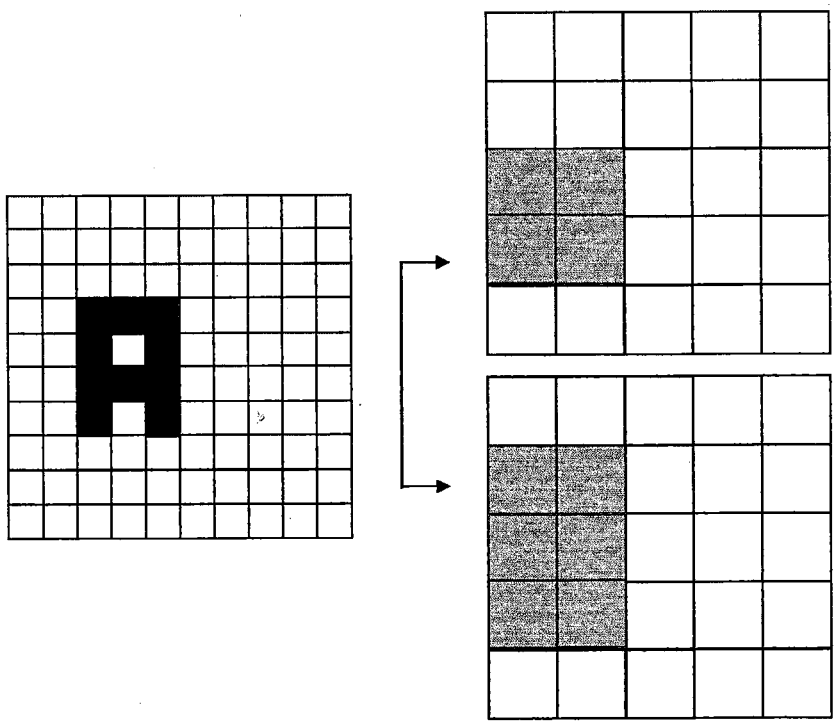


Figura 5.15. Geração de Campos *Coarse*.

A figura anterior mostra um campo de entrada de tamanho 10 x 10 pixels. Através de sobreposição de campos os quais possuem entre si um *offset* obtém-se dois campos de 5 x 5 *coarse* pixels. Cada campo é composto por pixels com o dobro da largura do campo de entrada. Esta transformação implica em que cada pixel da imagem original pode ser representado por um único conjunto de *coarse* pixels.

A figura 5.16, a seguir, mostra esquematicamente a sobreposição dos campos.

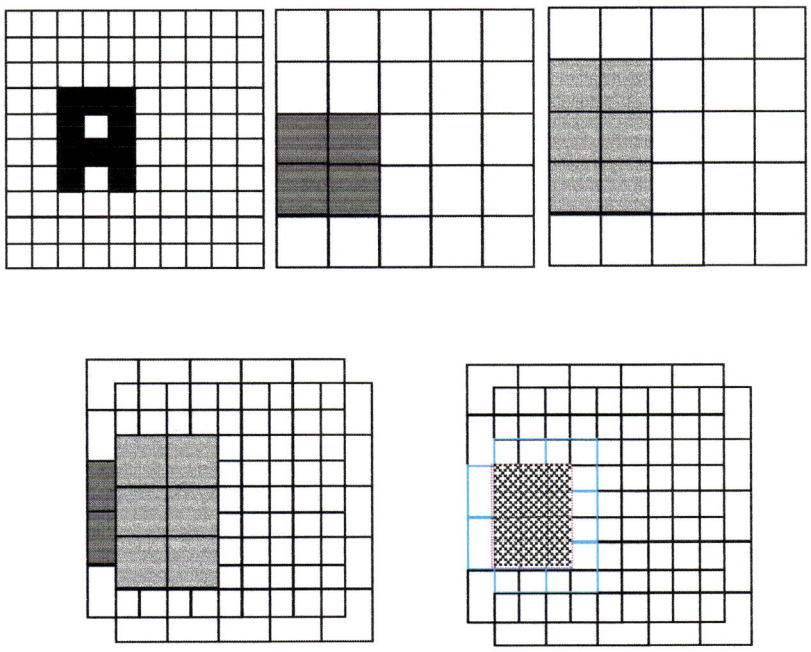


Figura 5.16. Sobreposição de Campos *Coarse*.

Como mostrado na figura anterior, através da sobreposição de campos de *coarse* pixels, é possível obter a imagem originalmente codificada.

Usando esta técnica, a relação entre o tamanho do campo de entrada (IFS), tamanho do campo *coarse* (CFS) e o número de campos *coarse* (n) em cada dimensão é dada por (SPIRKOVSKA & REID, 1993):

$$IFS = (CFS * n)$$

No presente trabalho, como são utilizadas imagens de 128 x 128 pixels, optou-se pela geração de 8 campos de 16 x 16 *coarse* pixels. Assim, cada imagem proveniente do sub-módulo de pré-processamento será representada, na saída do bloco gerador de coarse coading por 8 imagens de 16 x 16 pixels. Estas imagens são utilizadas para o treinamento da rede neural (quando da fase de aprendizado) e para teste (quando da fase de reconhecimento).

A geração dos coarse coadings foi baseada nas rotinas descritas a seguir.

```
// Gerando o primeiro coarse coading
 Cc=0;
 Cl=0;
 linha=0;
 coluna=0;
 contador=0;
        for (i=9; i<126; i+=8)
        {
                for (linha=0; linha<7; linha++)
                {
                        for (coluna=i-8; coluna<i; coluna++)
                                {
                                if (R[linha][coluna]==1)
                                  contador=contador+1;
                                 }

                }

                        if (contador>=1)
                                A[Cl][Cc]=1;
                        else
                                A[Cl][Cc]=0;
                        Cc=Cc+1;
                        contador=0;
}
 Cc=1;
contador=0;

for (j=15; j <126; j+=8)
        {
        for (i=9; i<126; i+=8)
        {
                for (linha=j-8; linha<j; linha++)
                        {
                                for (coluna=i-8; coluna<i; coluna++)
                                        {
                                                if (R[coluna][linha]==1)
                                                        {
                                                        contador=contador+1;
                                                        }
                                        }
                        }
                                if (contador>=1)
                                        A[Cl][Cc]=1;
                                else
                                        {
                                        A[Cl][Cc]=0;
                                        }
                        Cc=Cc+1;
                        contador=0;
        }
         Cc=0;
         Cl=Cl+1;
        }

        // gerando o segundo coarse coading
```

```
Cl=0;
Cc=0;
contador=0;
for (j=8; j <126; j+=8)
        {
        for (i=10; i<126; i+=8)
        {
                for (linha=j-8; linha<j; linha++)
                        {
                                for (coluna=i-8; coluna<i; coluna++)
                                        {
                                                if (R[coluna][linha]==1)
                                                        contador=contador+1;
                                        }
                        }
                                if (contador>=1)
                                        B[Cl][Cc]=1;
                                else
                                        {
                                        B[Cl][Cc]=0;
                                        }
                        Cc=Cc+1;
                        contador=0;
        }
        Cc=0;
        Cl=Cl+1;
        }
```

Os campos *coarse coading* gerados para cada imagem são apresentados à Rede Neural para treinamento.

Conforme descrito anteriormente, foram utilizadas diversas imagens geometricamente transformadas (posição, escala e rotação) além de imagens com ruído e imagens padrão para o treinamento da Rede Neural GSN. Assim, o funcionamento do sistema pode ser descrito como segue.

Um conjunto de imagens (128 x 128 pixels) é apresentado ao sistema, para treinamento com imagens conhecidas. Cada uma destas imagens irá gerar 8 imagens de 16 x 16 pixels (campos *coarse*) as quais ensinadas à Rede Neural. Uma vez que imagens geometricamente transformadas são pré-processadas e geram imagens que são muito próximas das imagens originais não transformadas, estas imagens (transformadas) gerarão campos *coarse* muito parecidos com os gerados pelas imagens não transformadas e já prendidos pela Rede.

Durante a fase de teste, sempre que uma imagem é apresentada ao sistema, o bloco de pré-processamento gera uma imagem a qual é então apresentada ao bloco de geração de *coarse coading*. Neste bloco, oito imagens são geradas e apresentadas à Rede para Reconhecimento.

Com conhecimento prévio dos campos *coarse* pertencentes às imagens de treinamento, a Rede Neural classifica a imagem.

Bloco de Reconhecimento

O bloco de reconhecimento de imagens foi implementado utilizando o modelo GSN de Rede Neural, incorporando as modificações propostas por MARTINS (1994) e descritas no Capítulo 3.

A estrutura utilizada foi do tipo Hierárquico, como mostrado na figura 3.21.

A estratégia para otimização da escolha das redes iniciais para o classificador GSN, além das modificações propostas para otimização dos códigos de saída e controle dinâmico da ordem de apresentação dos exemplos foram utilizadas na construção do classificador baseado em GSN.

A figura 5.17, a seguir, ilustra esquematicamente o bloco de reconhecimento.

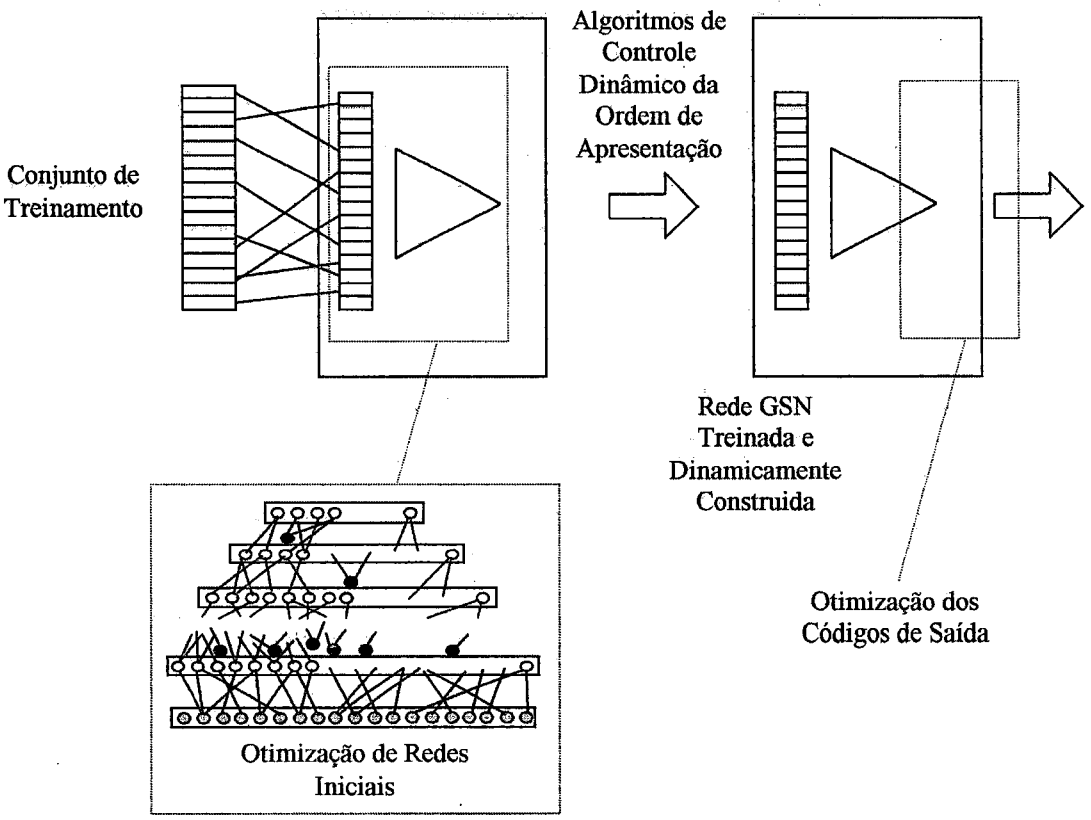


Figura 5.17. Bloco de Reconhecimento.

As rotinas que implementam as modificações do modelo GSN descritas podem ser vistas no anexo 4.

Através desta arquitetura, o treinamento é realizado de forma dinâmica, ou seja, algum tipo de comportamento cooperativo é imposto entre as redes componentes do classificador durante a apresentação das imagens de treinamento.

Na implementação das rotinas de teste do módulo de reconhecimento, foram utilizados os seguintes parâmetros:

MinDif - valor de threshold que controla a fração do conjunto de treinamento que deve ser considerada - assume valores entre 0, 5, 10 e 20;

Ciclos de Recombinações/*Resampling* - entre 20, 40 e 60 ciclos;

Número de pirâmides por bit dos códigos de saída: 3, 5 e 11;

Número de entradas por pirâmides: 32, 64 e 128.

5.3.4 Módulo de Controle

O módulo de controle tem como função, avaliar variáveis da imagem de entrada e fornecer dados sobre a mesma, os quais serão utilizados pelo mecanismo de manipulação das peças. Estes dados envolvem as seguintes variáveis:

- Posição real da peça em relação à plataforma de montagem;
- Ângulo de posicionamento da peça na plataforma;
- Medidas da peça a ser manipulada.

Assim, através da avaliação destas grandezas, o módulo de controle gera como saída uma relação de parâmetros, os quais controlarão o posicionamento da garra do dispositivo de manipulação.

A estratégia para implementação do módulo foi baseada nos dados obtidos durante a fase de pré-processamento, ou seja, dos blocos de translação, escala e rotação.

As coordenadas do centro de gravidade e dos pontos extremos da peça a ser manipulada são utilizadas para avaliação de seu tamanho e distância em relação a um ponto de referência localizado na plataforma de trabalho. A avaliação do ângulo de rotação permite girar a garra de forma precisa para o correto posicionamento e pega.

A figura 5.18, a seguir, mostra esquematicamente a geração dos parâmetros para o módulo de controle, baseadas em uma sugestão de possível configuração da plataforma de trabalho.

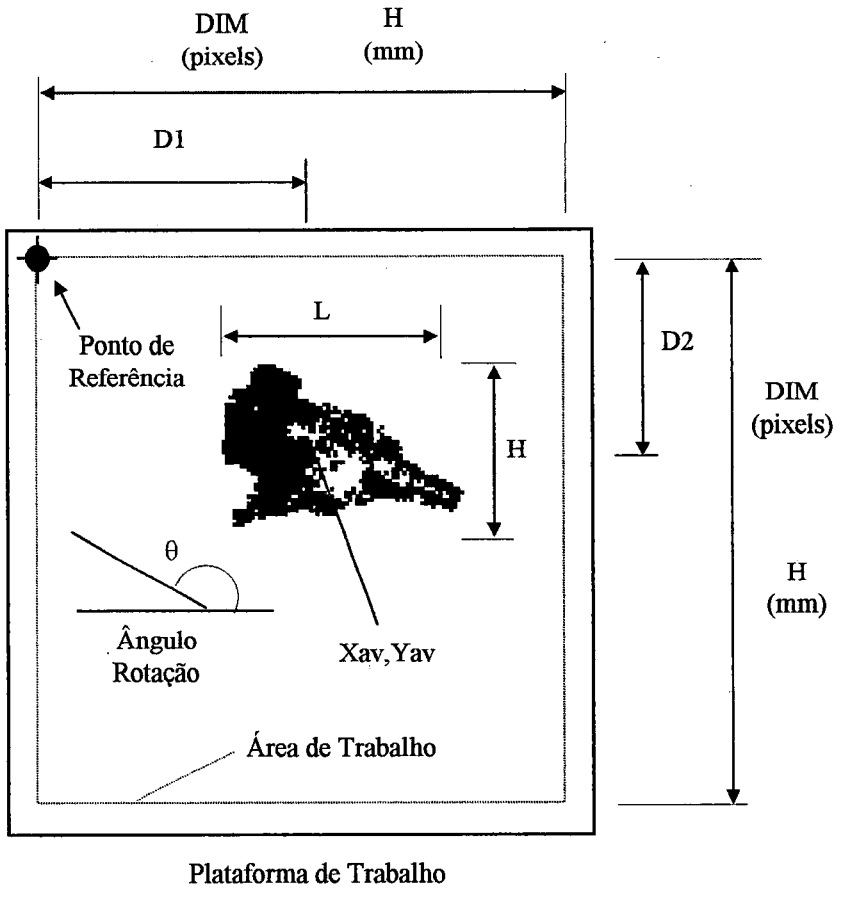


Figura 5.18. Dados utilizados no Módulo de Controle.

Através da figura anterior, podemos descrever a geração dos dados de saída do Módulo de Controle.

O Ponto de Referência, o qual deve ser uma identificação de um ponto conhecido da plataforma de trabalho, proporciona as medidas relativas, de acordo com o tamanho da área de trabalho, a qual é definida em quantidade de pixels (128 x 128, 512 x 512, 1024 x124, etc.). Portanto, a precisão das medidas fornecidas pelo Módulo será definida como:

$$\textit{Precisão}:\ 1\ \text{pixel} = \frac{H(mm)}{DIM(pixels)}$$

Assim, as distâncias do centro de gravidade (coordenadas Xav,Yav) ao ponto de referência podem ser definidas como:

D1 = Xav (pixels) . *Precisão* (mm)
D2 = Yav (pixels) . *Precisão* (mm)

Portanto, a distância (mm) entre o centro de gravidade da peça e o ponto de referência é dada por:

$$D = \sqrt{D1^2 + D2^2} \text{ (mm)} \qquad (5.17)$$

Da mesma forma, as dimensões da peça, L e H podem ser calculadas como:

L = Distância entre pontos extremos da peça na direção x (pixels) . *Precisão* (mm)

H = Distância entre pontos extremos da peça na direção y (pixels) . *Precisão* (mm)

Através destas medidas e da utilização do ângulo de rotação (obtido através do bloco de rotação no estágio de pré-processamento) pode-se realizar o correto posicionamento da garra.

Portanto, a precisão de posicionamento está diretamente relacionada ao tamanho da grade de imagem (área de trabalho) em pixels. Quanto maior a quantidade de pixels, maior a precisão de posicionamento.

A figura 5.19 mostra o funcionamento do módulo, esquematicamente.

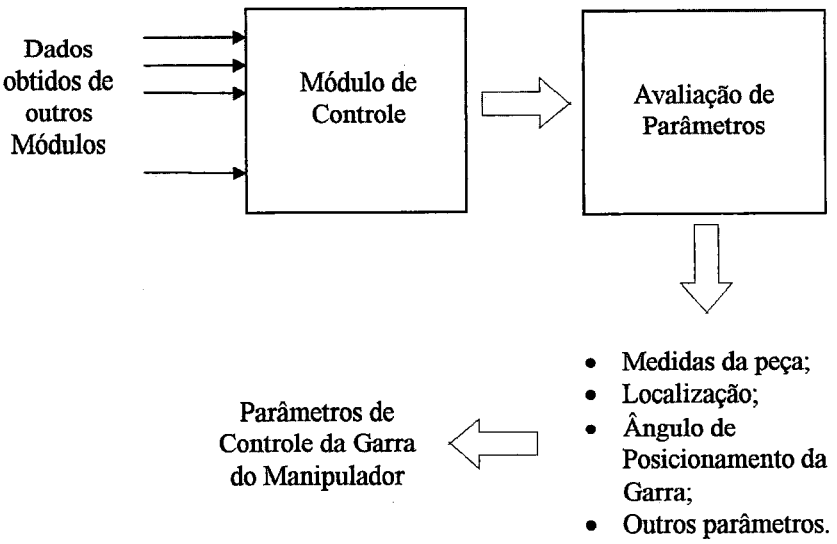


Figura 5.19. Módulo de Controle.

As rotinas para geração dos parâmetros de controle da garra do manipulador não foram implementadas, já que dependem da implementação prática de uma plataforma de trabalho e da definição do tipo de manipulador a ser utilizado. Tais rotinas seguem o mesmo padrão utilizado nos demais módulos, com pequenas modificações.

5.4 RESULTADOS EXPERIMENTAIS

A seguir, são apresentados resultados experimentais obtidos nos diversos blocos componentes do sistema.

Sub-Módulo Pré-Processador

O conjunto de imagens mostrado a seguir mostra as imagens resultantes dos blocos de translação, escala e rotação do Pré-Processador, para um determinado universo de imagens.

São mostrados, também, os valores obtidos para o Centro de Gravidade (Xav, Yav, em coordenadas baseadas no centro da grade), para o Raio Médio (Rav), para o fator de escala (s), para seno e cosseno (sen, cos) e para o numero total de pixels com valor igual a 1 na imagem (P).

Tais resultados foram obtidos utilizando imagens de 128x128 pixels.

Imagem Binarizada	Imagem Transladada	Imagem Escalonada	Imagem Rotacionada

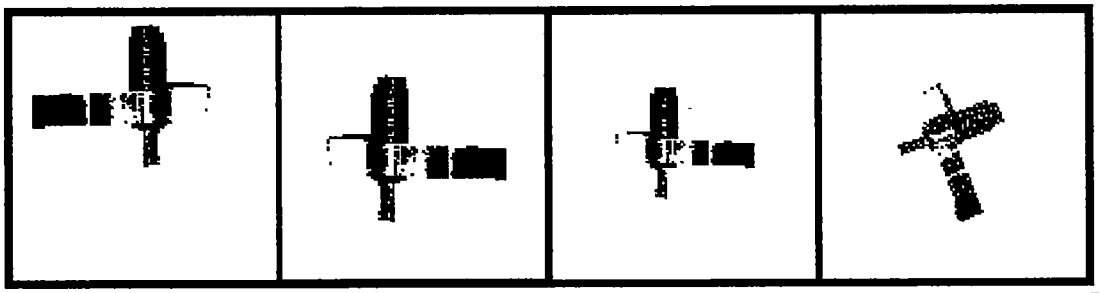


Imagem	Xav	Yav	Rav	s	sen	cos	P
19P1	-22	9	21.18	0.77	0.9256	0.3783	1179

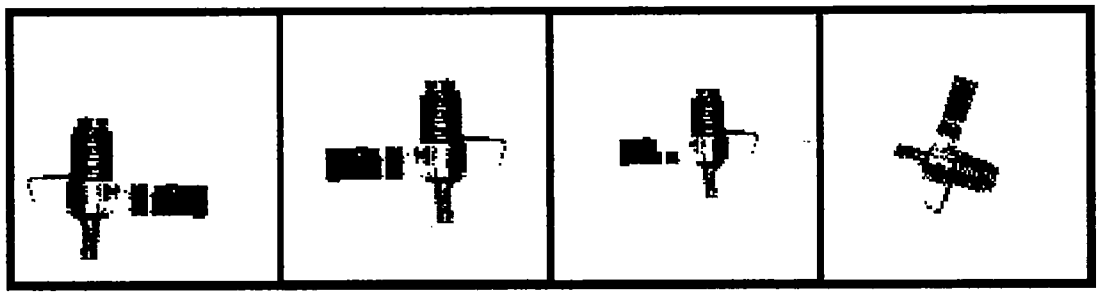


Imagem	Xav	Yav	Rav	s	sen	cos	P
19PR1	15	12	21.48	0.78	0.9456	-0.3251	1196

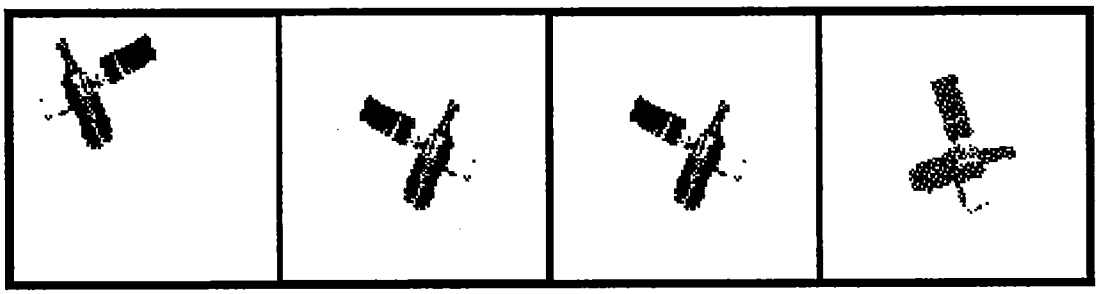


Imagem	Xav	Yav	Rav	s	sen	cos	P
19PRS1	-27	19	15.88	0.57	0.5940	0.8044	656

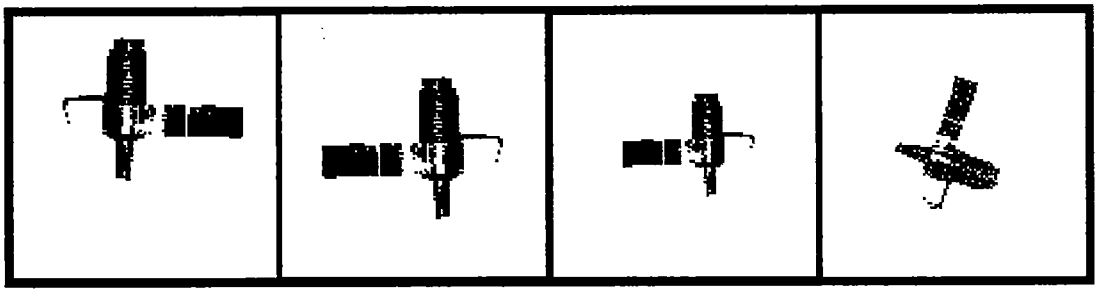


Imagem	Xav	Yav	Rav	s	sen	cos	P
19PR2	-18	-3	21.15	0.76	0.9368	-0.3497	1223

Figura 5.20. Imagens Pré-Processadas - Exemplo1.

Imagem Binarizada	Imagem Transladada	Imagem Escalonada	Imagem Rotacionada

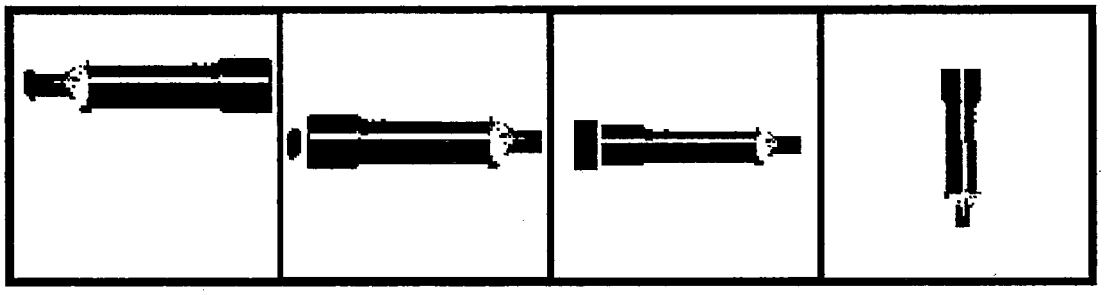


Imagem	Xav	Yav	Rav	s	sen	cos	P
25P1	-25	-8	26.51	0.96	0.9999	0.0117	1627

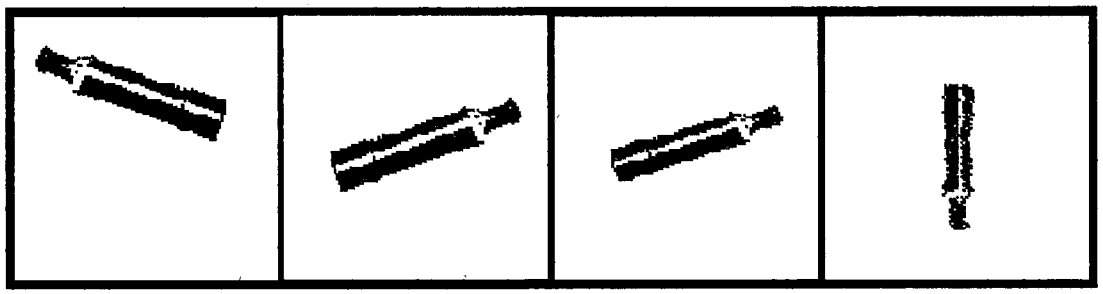


Imagem	Xav	Yav	Rav	s	sen	cos	P
25PRS1	-23	2	21.18	0.77	0.9458	-0.3247	970

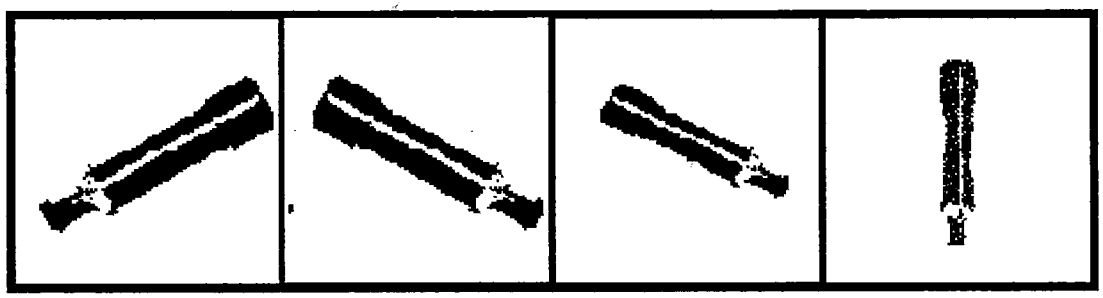


Imagem	Xav	Yav	Rav	s	sen	cos	P
25R1	0	-10	27.27	0.99	0.8828	0.4695	1662

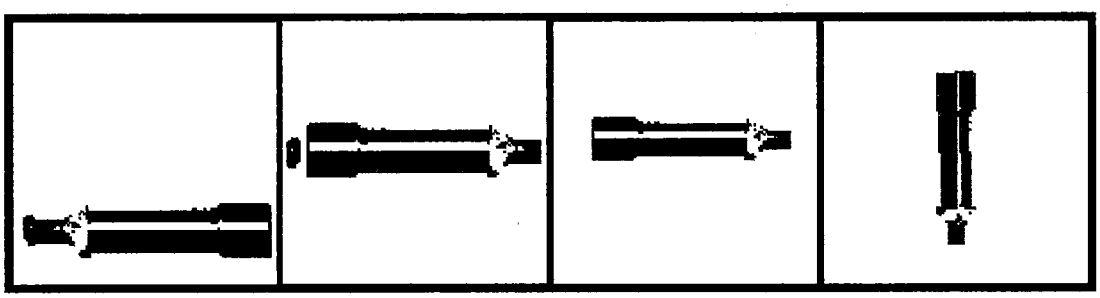


Imagem	Xav	Yav	Rav	s	sen	cos	P
25P2	34	-8	27.05	0.98	0.9999	0.0058	1565

Figura 5.21. Imagens Pré-Processadas - Exemplo 2.

Imagem Binarizada	Imagem Transladada	Imagem Escalonada	Imagem Rotacionada

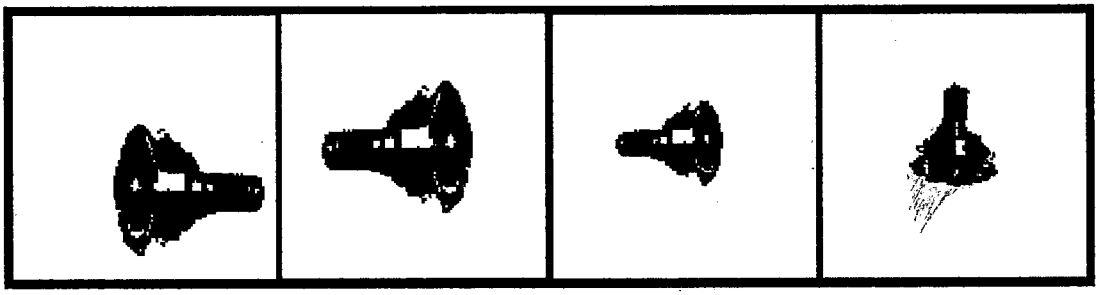


Imagem	Xav	Yav	Rav	s	sen	cos	P
13P1	18	-12	19.23	0.69	0.9967	-0.0810	1808

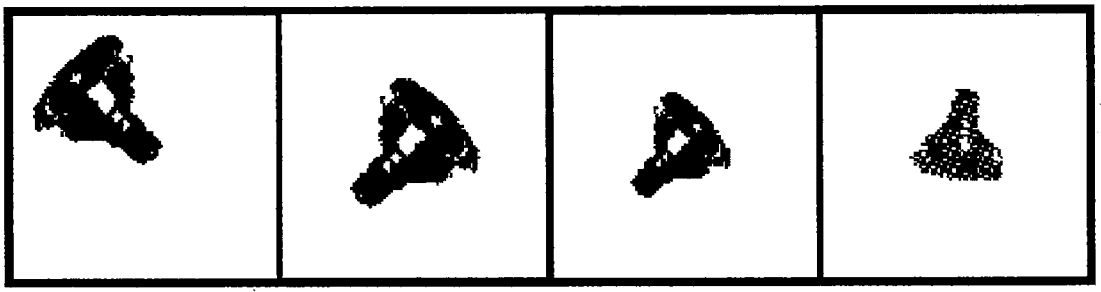


Imagem	Xav	Yav	Rav	s	sen	cos	P
13PRS1	-19	20	17.16	0.62	0.6283	-0.7779	1441

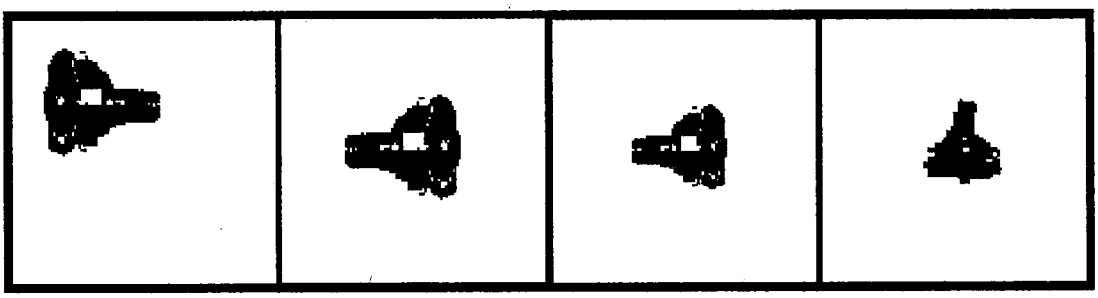


Imagem	Xav	Yav	Rav	s	sen	cos	P
13PS1	-20	23	15.43	0.56	0.9965	-0.0832	1224

Imagem	Xav	Yav	Rav	s	sen	cos	P
13R1	-1	3	19.56	0.71	0.9639	0.2659	1927

Figura 5.22. Imagens Pré-Processadas - Exemplo 3.

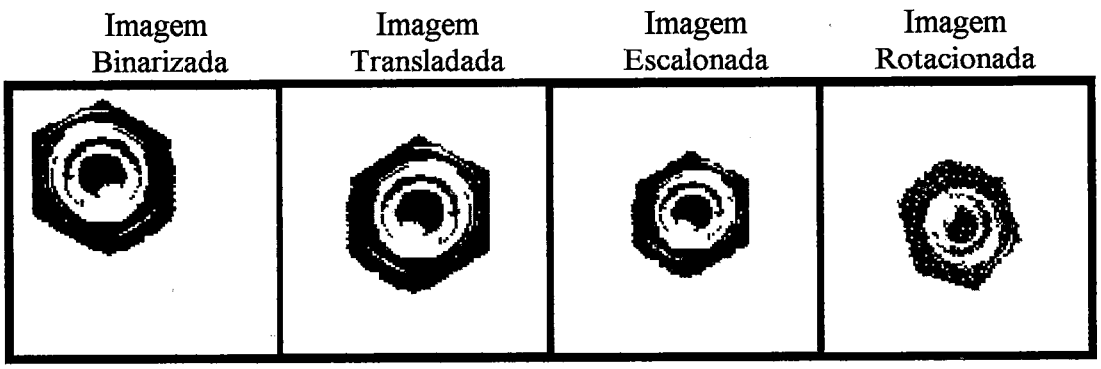


Imagem	Xav	Yav	Rav	s	sen	cos	P
12P1	-17	16	23,26	0.84	0.9627	-0.2702	2087

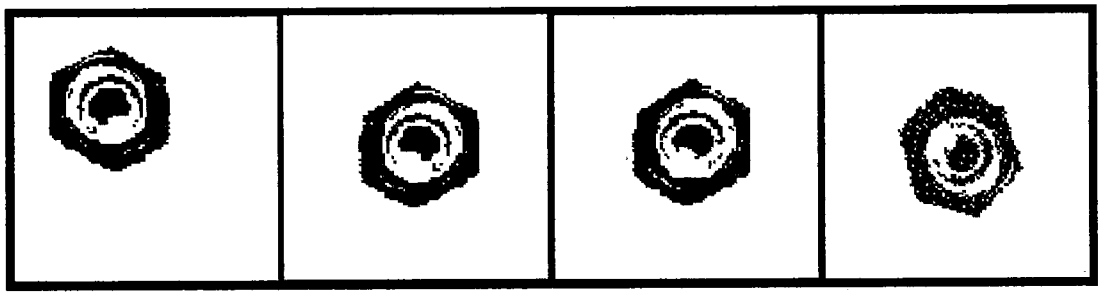


Imagem	Xav	Yav	Rav	s	sen	cos	P
12PS2	-17	14	18.97	0.69	0.9778	-0.2094	1411

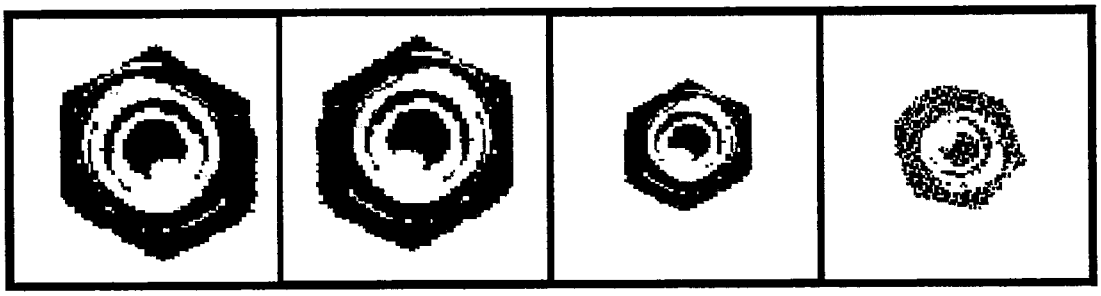


Imagem	Xav	Yav	Rav	s	sen	cos	P
12S1	3	-6	31.82	1.15	0.9688	-0.2477	3901

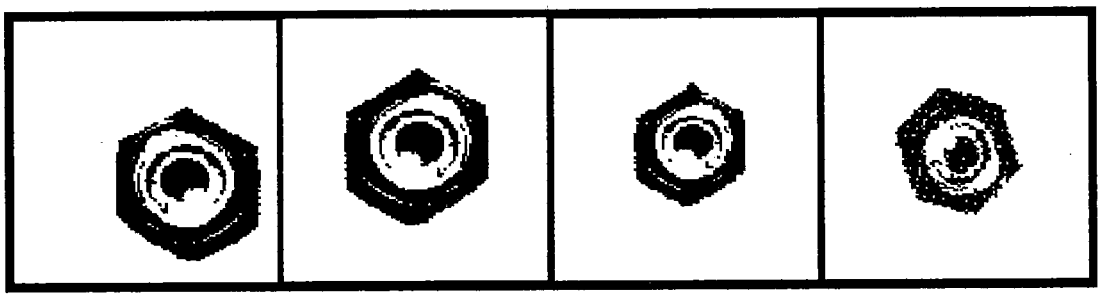


Imagem	Xav	Yav	Rav	s	sen	cos	P
12P2	15	-19	23.32	0.8482	0.9657	-0.2595	2132

Figura 5.23. Imagens Pré-Processadas - Exemplo 4.

Imagem Binarizada	Imagem Transladada	Imagem Escalonada	Imagem Rotacionada

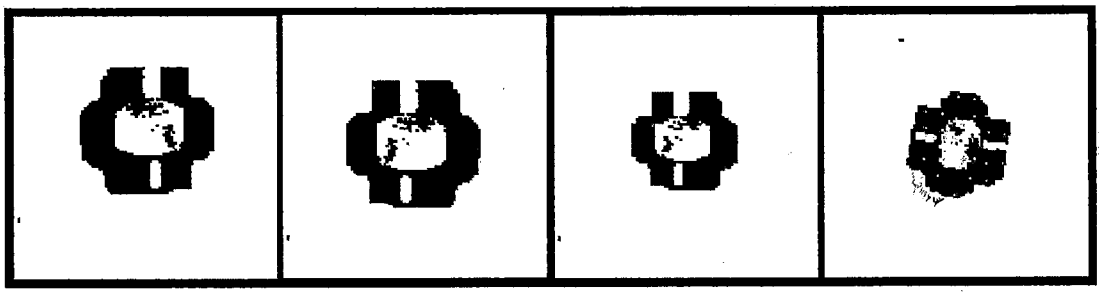


Imagem	Xav	Yav	Rav	s	sen	cos	P
11R1	-7	0	21.84	0.79	0.9901	-0.1397	1846

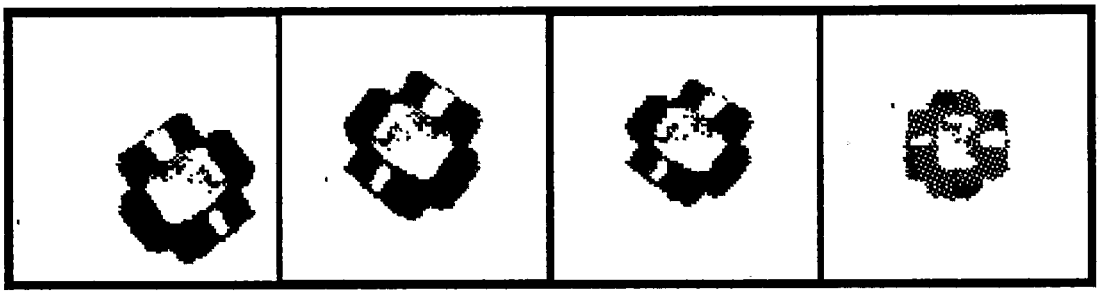


Imagem	Xav	Yav	Rav	s	sen	cos	P
11RP1	18	-17	22.58	0.82	0.8067	0.5909	1844

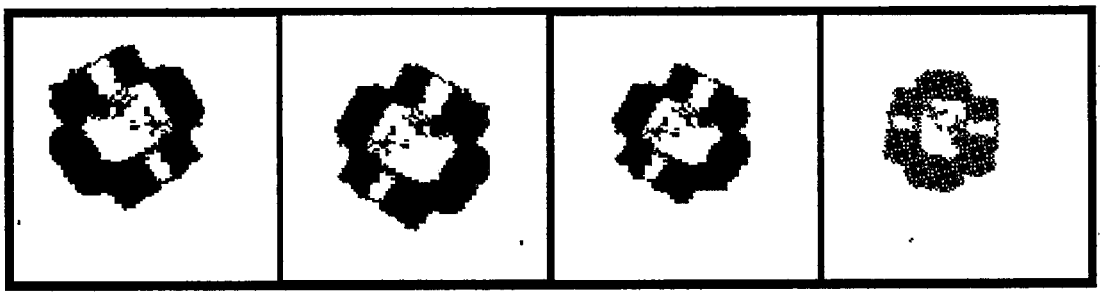


Imagem	Xav	Yav	Rav	s	sen	cos	P
11RS1	-9	9	24,88	0.90	0.8966	0.4426	2391

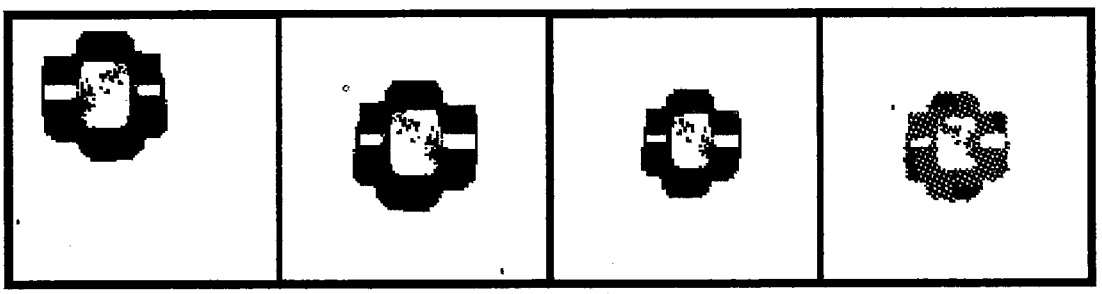


Imagem	Xav	Yav	Rav	s	sen	cos	P
11P1	-22	17	21.07	0.7664	0.0127	0.9999	1753

Figura 5.24. Imagens Pré-Processadas - Exemplo 5.

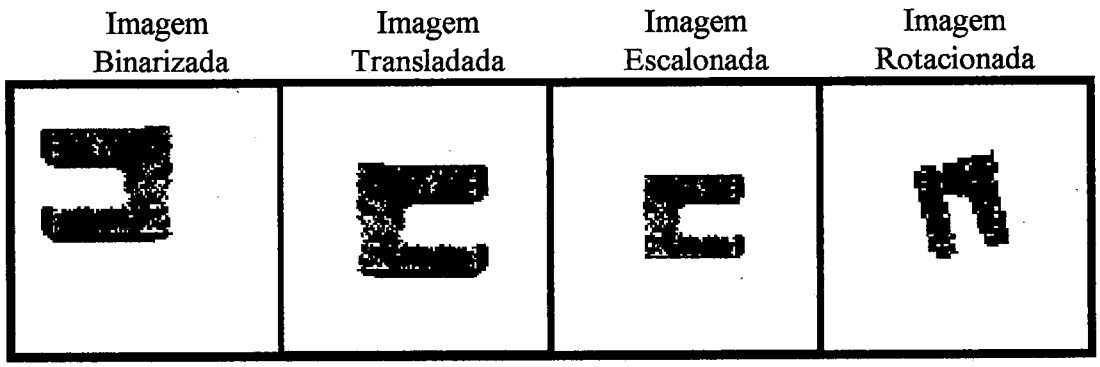


Imagem	Xav	Yav	Rav	s	sen	cos	P
16P1	-17	14	21.66	0.78	0.9995	-0.0307	1887

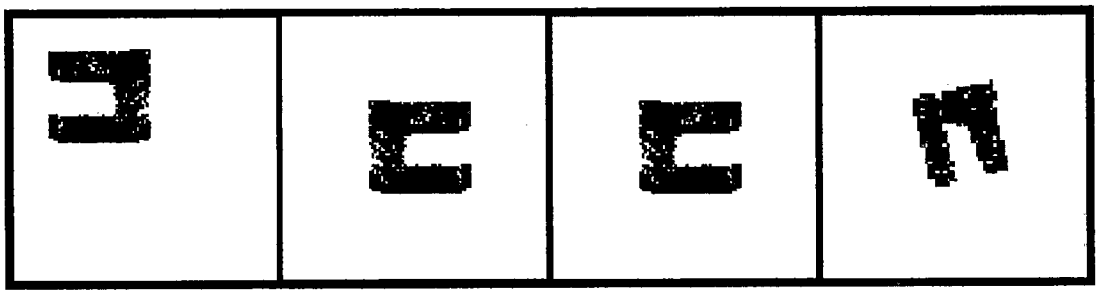


Imagem	Xav	Yav	Rav	s	sen	cos	P
16PS1	-23	17	17.28	0.62	0.9794	0.2016	1187

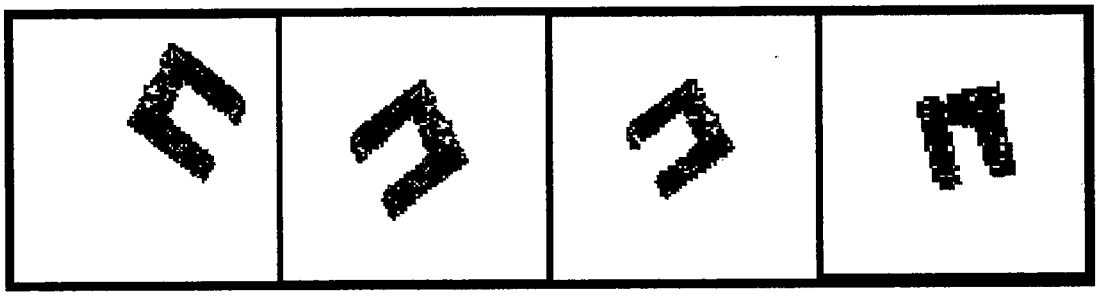


Imagem	Xav	Yav	Rav	s	sen	cos	P
16PSR1	-17	-16	17.71	0.64	0.0526	0.9986	1136

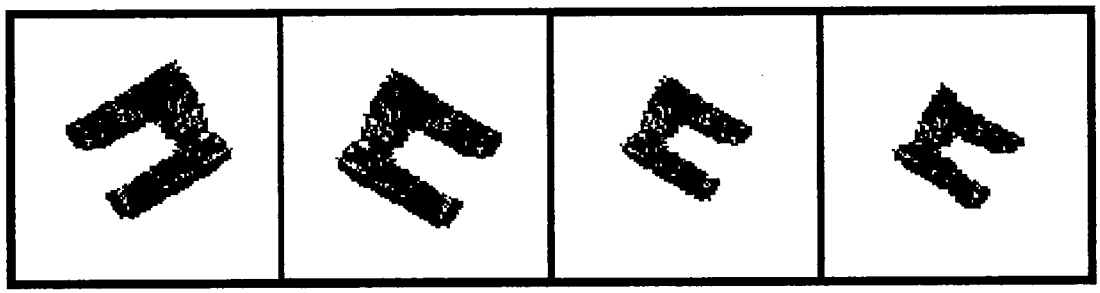


Imagem	Xav	Yav	Rav	s	sen	cos	P
16R1	-5	-3	21.94	0.79	0.0072	0.9999	1938

Figura 5.25. Imagens Pré-Processadas - Exemplo 6.

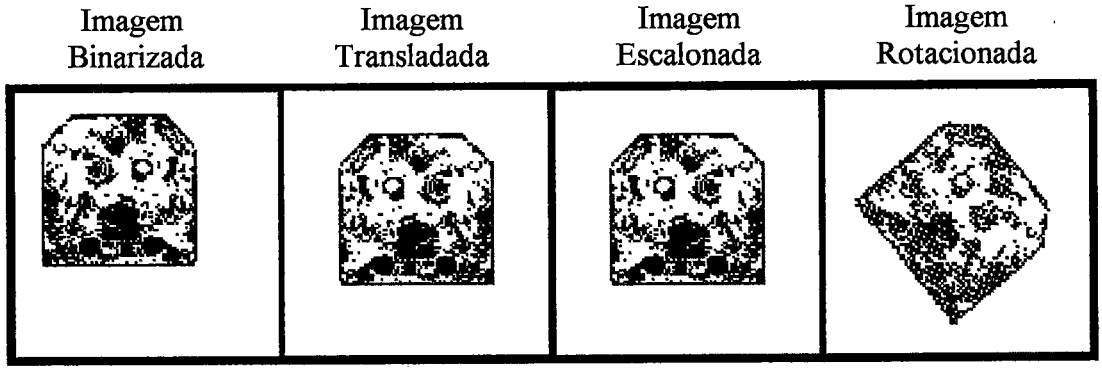


Imagem	Xav	Yav	Rav	s	sen	cos	P
17P1	-9	12	25.24	0.91	0.7729	0.6350	1898

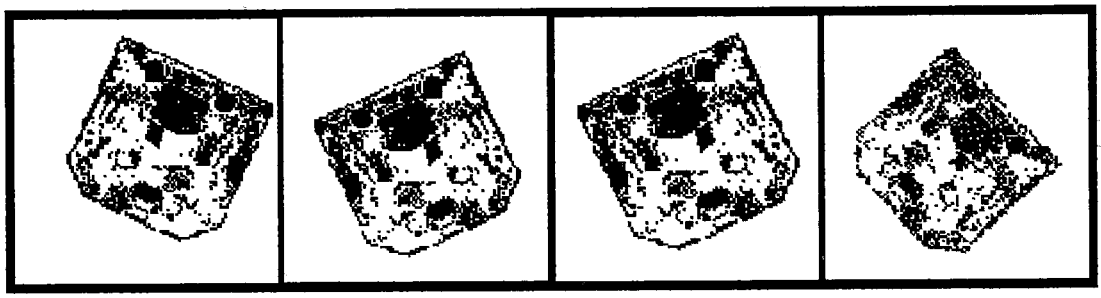


Imagem	Xav	Yav	Rav	s	sen	cos	P
17PR1	-8	-11	27.23	0.99	0.9568	0.2906	2278

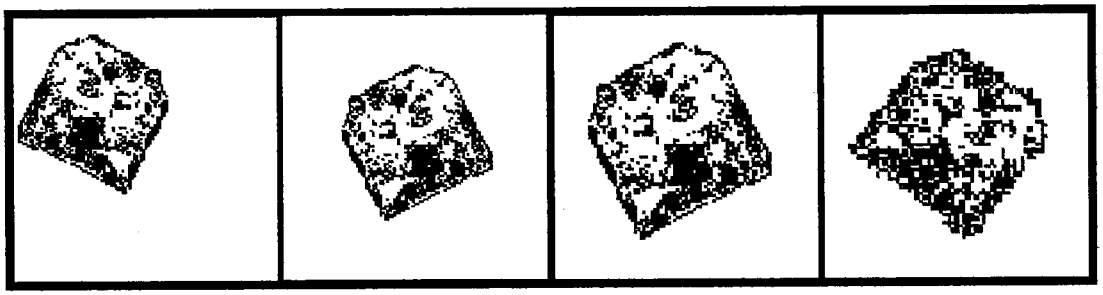


Imagem	Xav	Yav	Rav	s	sen	cos	P
17PSR1	-13	22	21.06	0.7660	0.9002	0.4354	1305

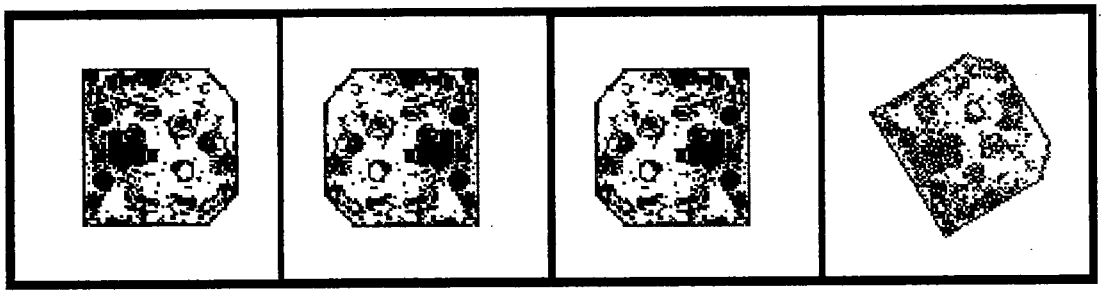


Imagem	Xav	Yav	Rav	s	sen	cos	P
17R1	-1	0	26.08	0.95	0.4986	-0.8668	2013

Figura 5.26. Imagens Pré-Processadas - Exemplo 7.

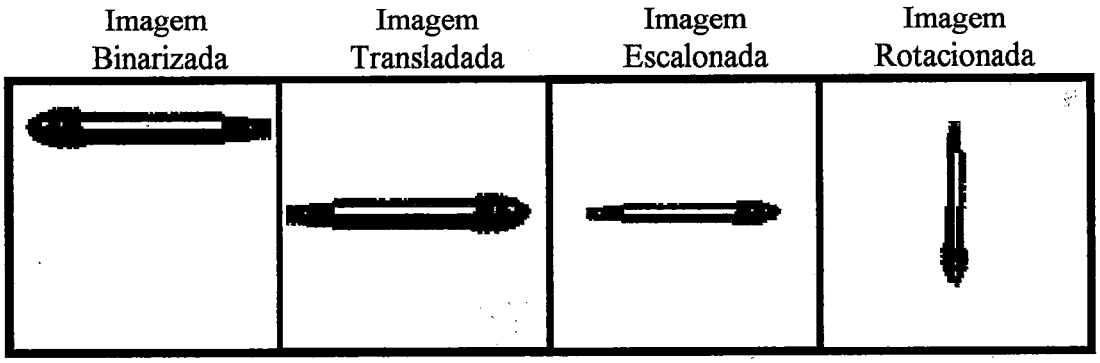


Imagem	Xav	Yav	Rav	s	sen	cos	P
24P1	-38	1	27.01	0.98	0.9999	-0.0060	1214

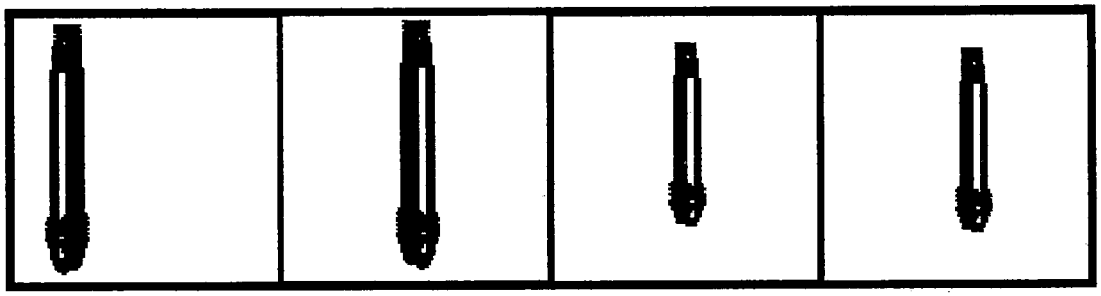


Imagem	Xav	Yav	Rav	s	sen	cos	P
24PR1	1	33	27.94	1.01	0.0036	0.9999	1380

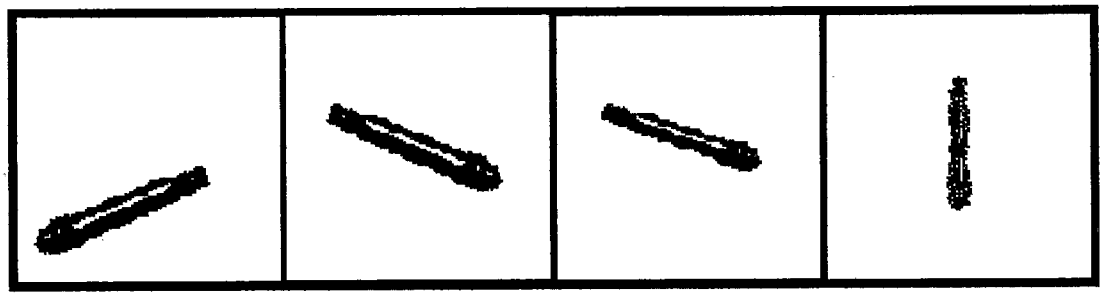


Imagem	Xav	Yav	Rav	s	sen	cos	P
24PRS1	26	13	20.39	0.74	0.9283	0.3717	828

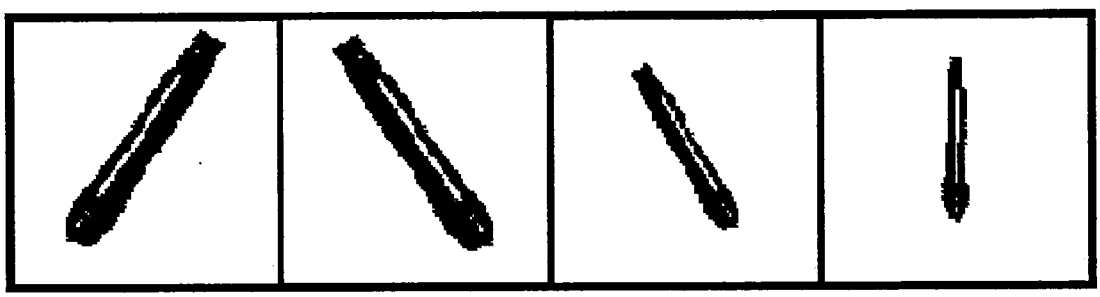


Imagem	Xav	Yav	Rav	s	sen	cos	P
24R1	-3	1	27.95	1.02	0.5613	0.8276	1530

Figura 5.27. Imagens Pré-Processadas - Exemplo 8.

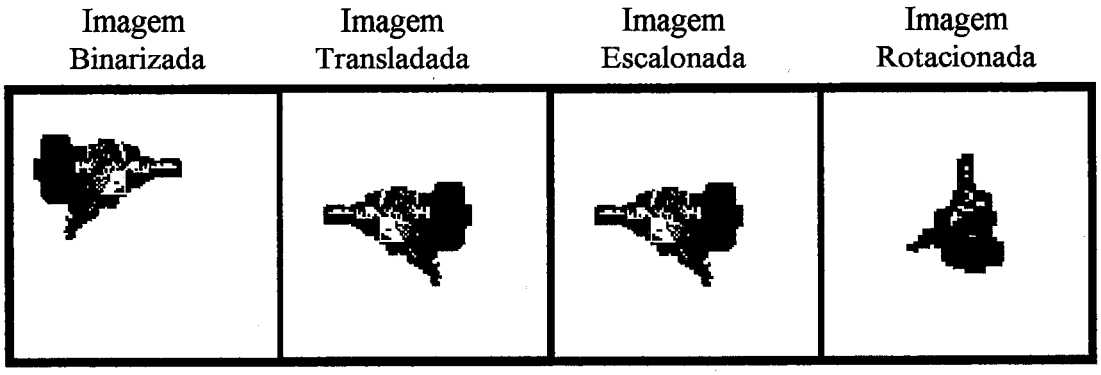


Imagem	Xav	Yav	Rav	s	sen	cos	P
10PS1	-21	24	16.35	0.59	0.9964	0.0836	1220

Imagem	Xav	Yav	Rav	s	sen	cos	P
10PSR1	-18	-2	17.30	0.63	0.9490	0.3299	1292

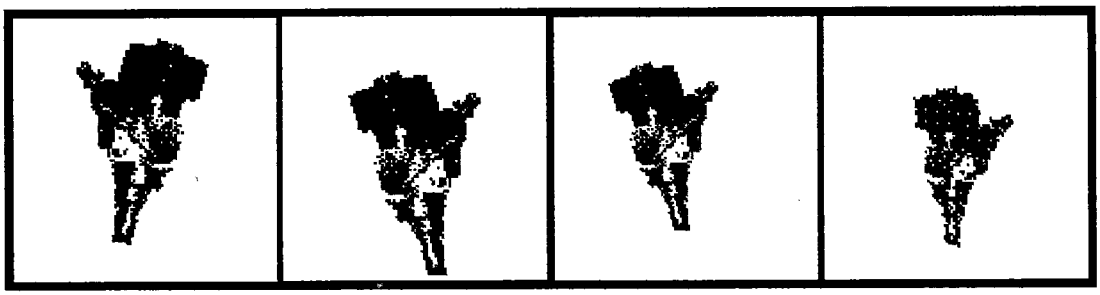


Imagem	Xav	Yav	Rav	s	sen	cos	P
10R105	-14	1	21.82	0.79	0.2028	0.9792	1996

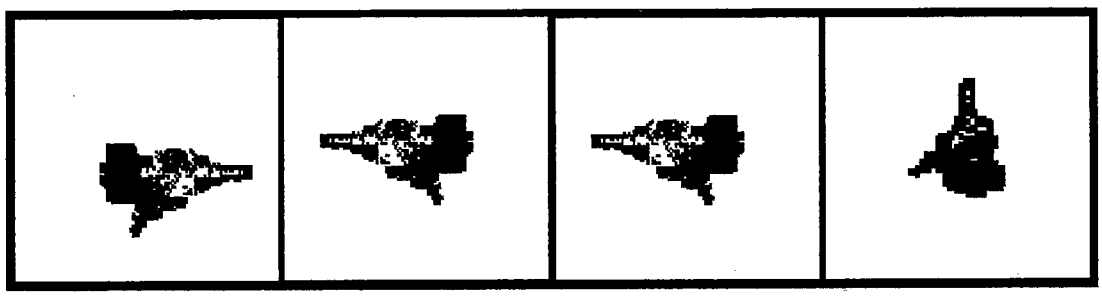


Imagem	Xav	Yav	Rav	s	sen	cos	P
10PS2	13	-4	16.13	0.59	0.9982	0.0597	1075

Figura 5.28. Imagens Pré-Processadas - Exemplo 9.

Imagem Binarizada	Imagem Transladada	Imagem Escalonada	Imagem Rotacionada

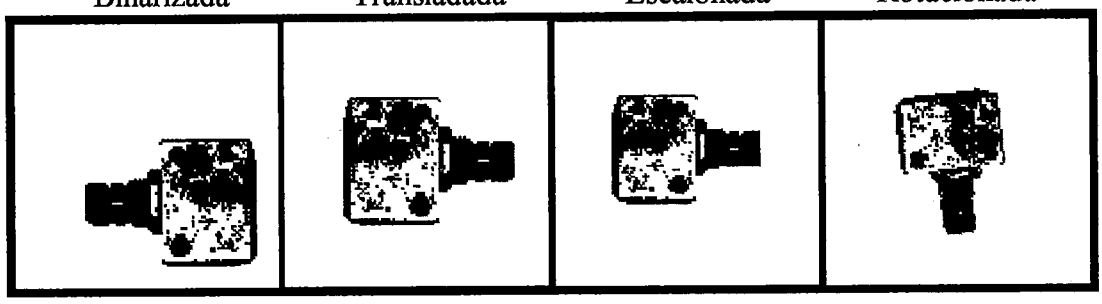


Imagem	Xav	Yav	Rav	s	sen	cos	P
29P1	17	-16	23.20	0.84	0.9937	0.1119	1842

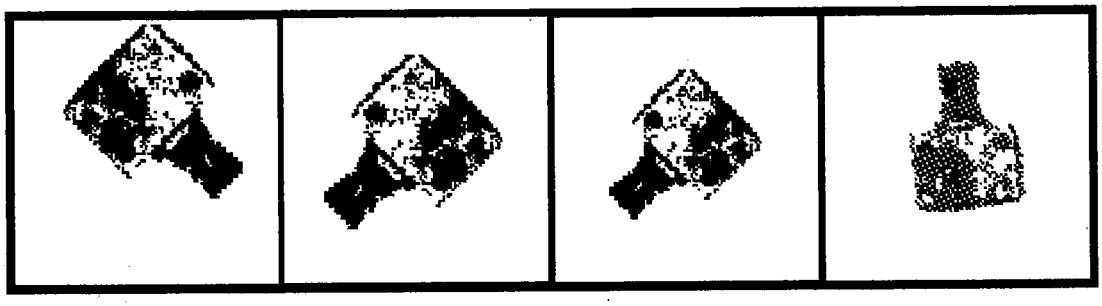


Imagem	Xav	Yav	Rav	s	sen	cos	P
29PR1	-14	-2	23.65	0.86	0.7844	-0.6201	1923

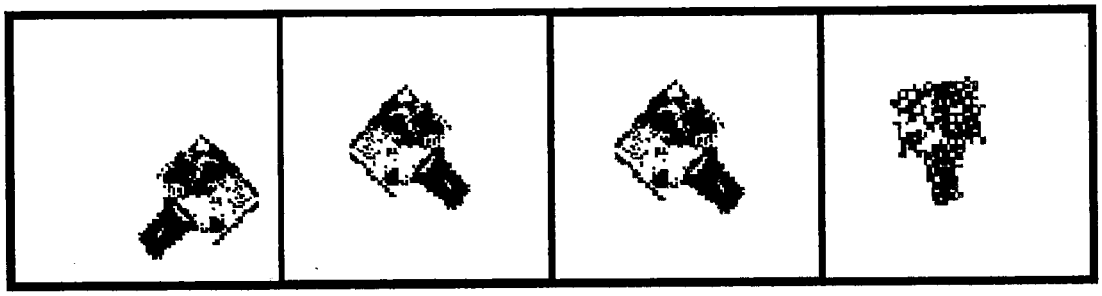


Imagem	Xav	Yav	Rav	s	sen	cos	P
29PRS1	20	-20	16.53	0.6012	0.6462	0.7631	927

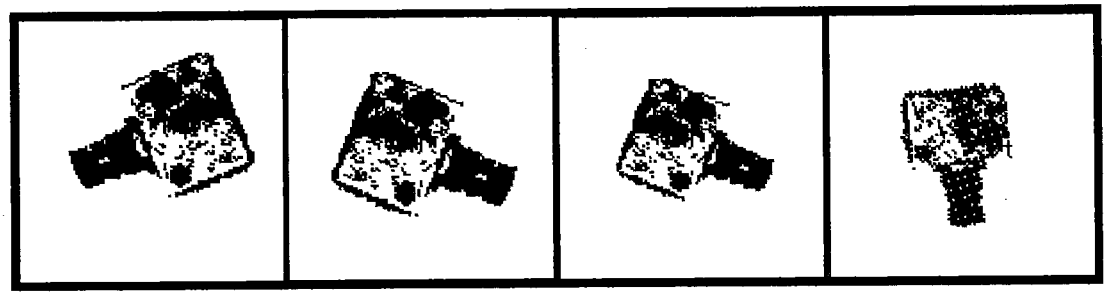


Imagem	Xav	Yav	Rav	s	sen	cos	P
29R1	-9	-6	23.16	0.84	0.8831	0.4690	1888

Figura 5.29. Imagens Pré-Processadas - Exemplo 10.

Imagem Binarizada	Imagem Transladada	Imagem Escalonada	Imagem Rotacionada

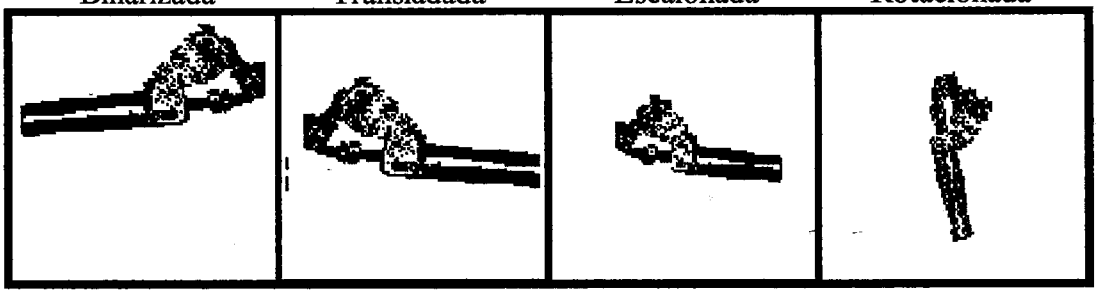


Imagem	Xav	Yav	Rav	s	sen	cos	P
2P1	-23	-5	28.65	1.04	0.9698	0.2437	1568

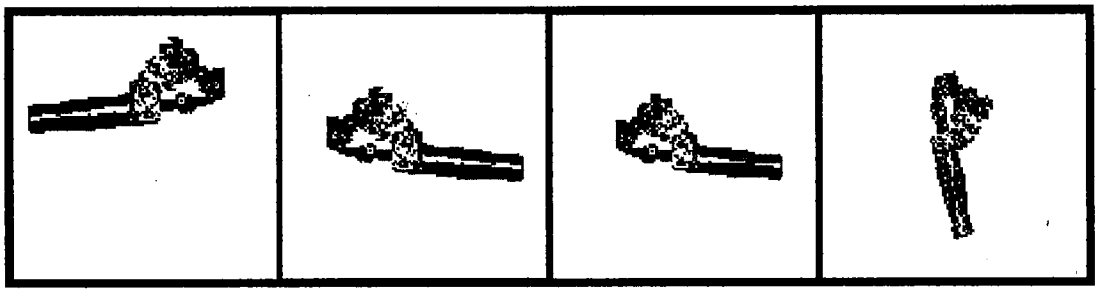


Imagem	Xav	Yav	Rav	s	sen	cos	P
2PS1	-22	3	23.27	0.84	0.9571	0.2898	1114

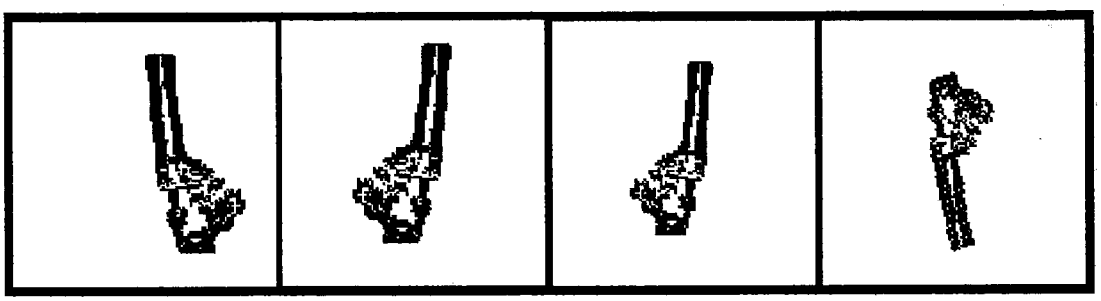


Imagem	Xav	Yav	Rav	s	sen	cos	P
2PSR1	4	-17	23.92	0.87	0.2919	-0.9564	1127

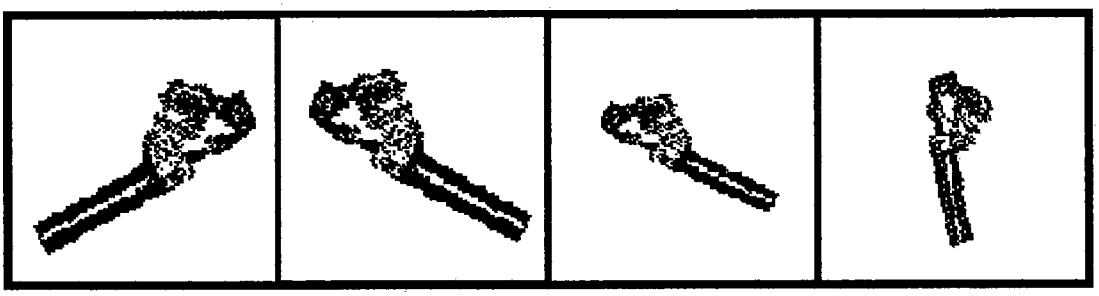


Imagem	Xav	Yav	Rav	s	sen	cos	P
2R1	4	-4	28.80	1.05	0.7867	0.6173	1570

Figura 5.30. Imagens Pré-Processadas - Exemplo 11.

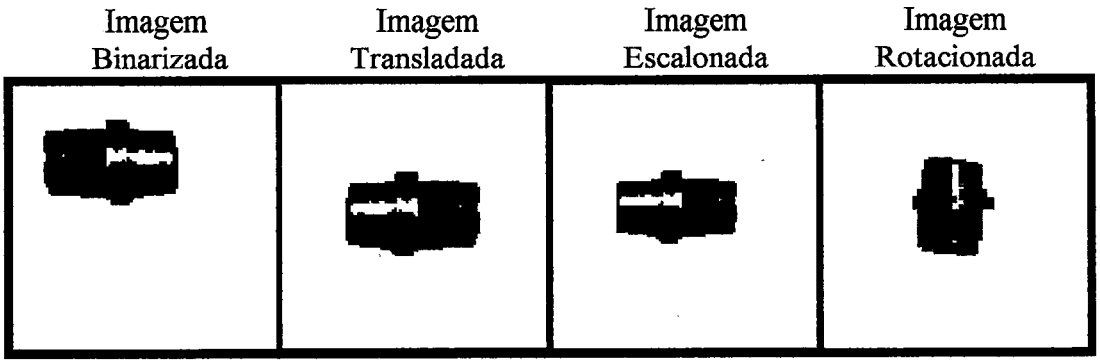


Imagem	Xav	Yav	Rav	s	sen	cos	P
30P1	-23	15	17.11	0.62	0.9996	-0.0283	1573

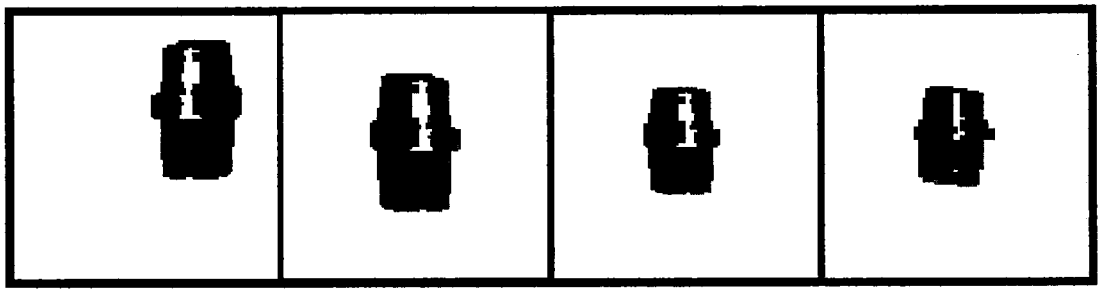


Imagem	Xav	Yav	Rav	s	sen	cos	P
30P2	-15	-22	17.97	0.65	0.0363	0.9993	1736

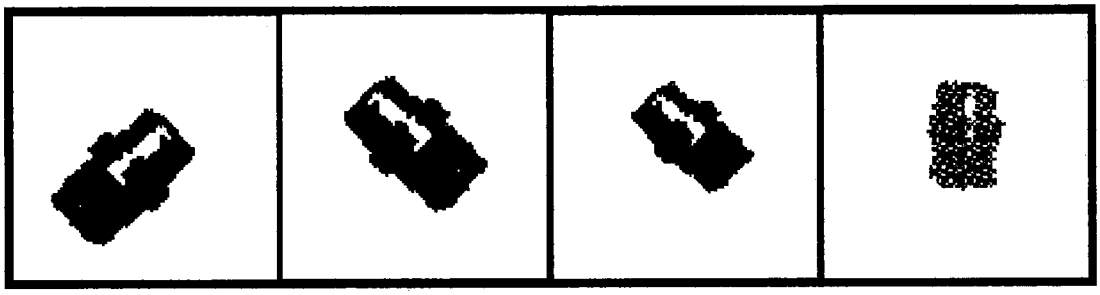


Imagem	Xav	Yav	Rav	s	sen	cos	P
30PR1	14	9	18.16	0.66	0.7844	0.6563	1740

Figura 5.31. Imagens Pré-Processadas - Exemplo 12.

Imagem Binarizada	Imagem Transladada	Imagem Escalonada	Imagem Rotacionada

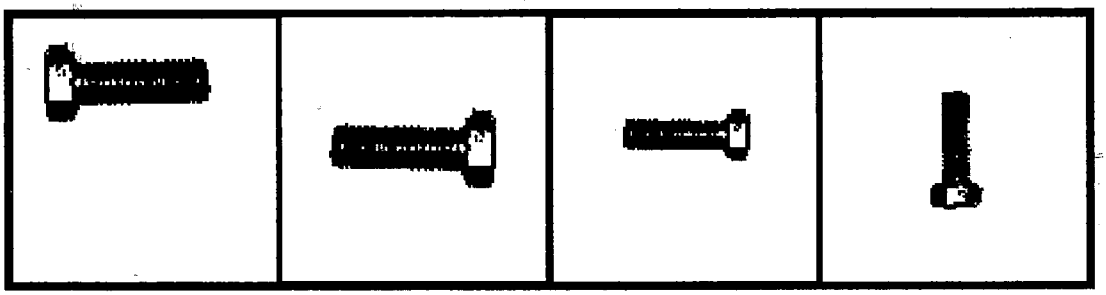


Imagem	Xav	Yav	Rav	s	sen	cos	P
31P1	-29	8	19.67	0.71	0.9997	0.0206	1288

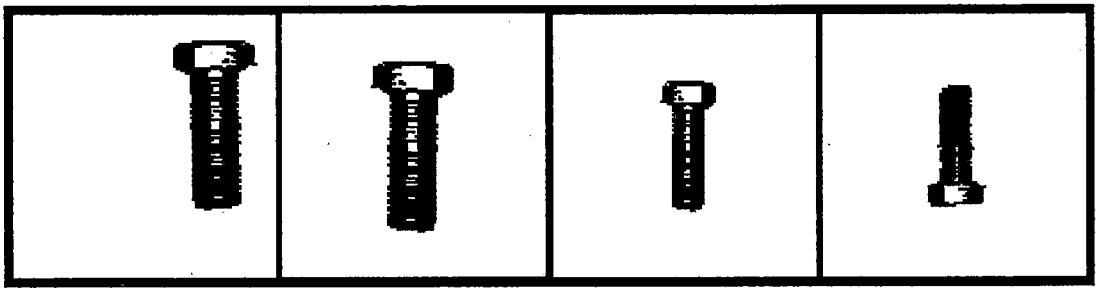


Imagem	Xav	Yav	Rav	s	sen	cos	P
31PR1	-10	-30	20.59	0.75	0.0242	-0.9997	1415

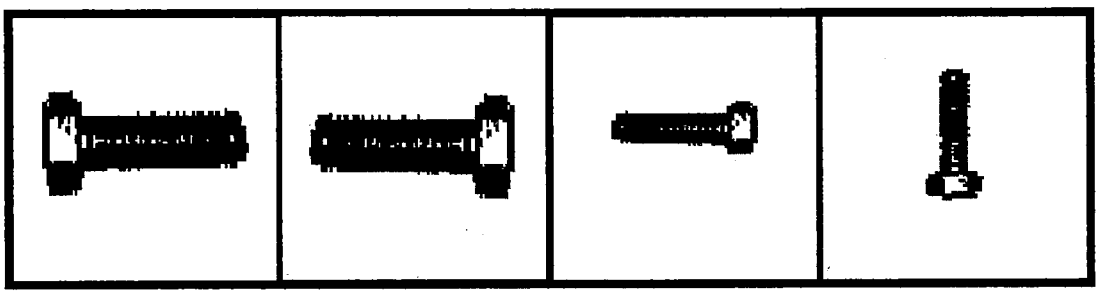


Imagem	Xav	Yav	Rav	s	sen	cos	P
31S1	-2	1	24.49	0.8907	0.9998	0.0203	2084

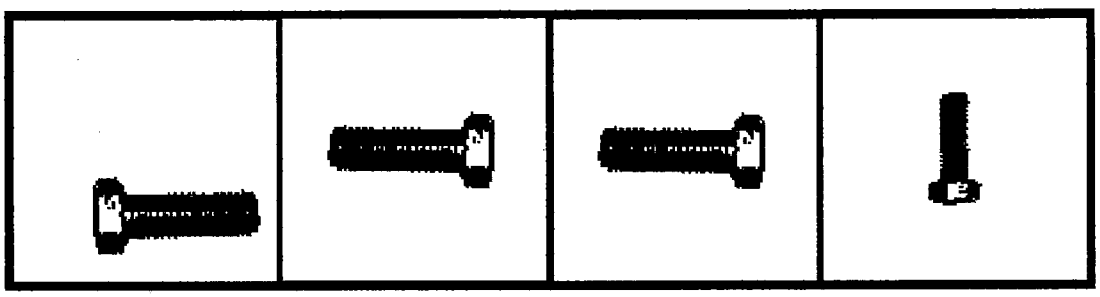


Imagem	Xav	Yav	Rav	s	sen	cos	P
31P2	28	-12	19.65	0.71	0.9994	0.0322	1297

Figura 5.32. Imagens Pré-Processadas - Exemplo 13.

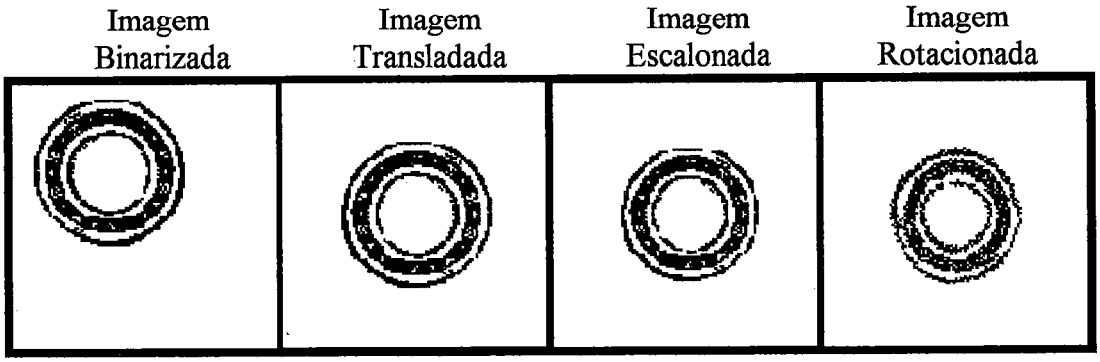


Imagem	Xav	Yav	Rav	s	sen	cos	P
7P1	-19	14	25.09	0.91	0.9488	-0.3157	1436

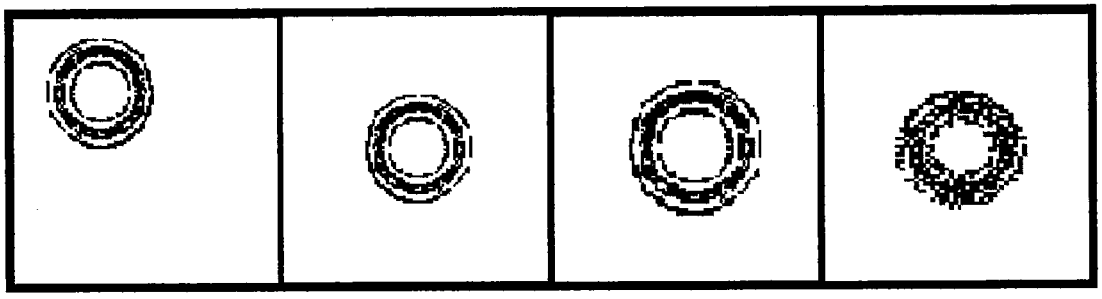


Imagem	Xav	Yav	Rav	s	sen	cos	P
7PS1	-25	18	18.35	0.67	0.6656	-0.7463	750

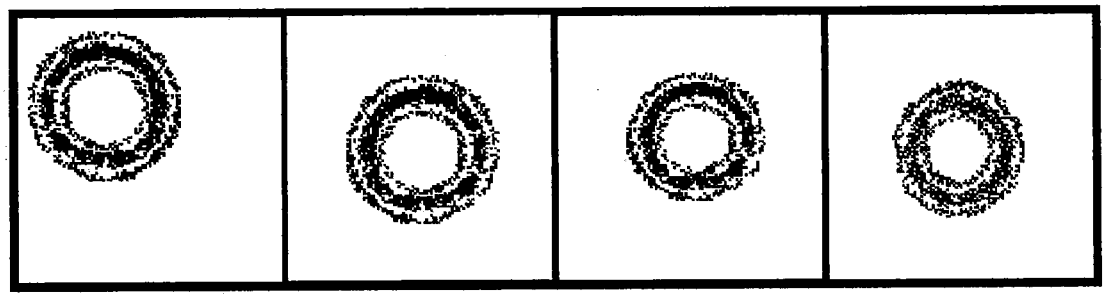


Imagem	Xav	Yav	Rav	s	sen	cos	P
ruid7P	-20	18	25.08	0.91	0.9256	-0.3785	1409

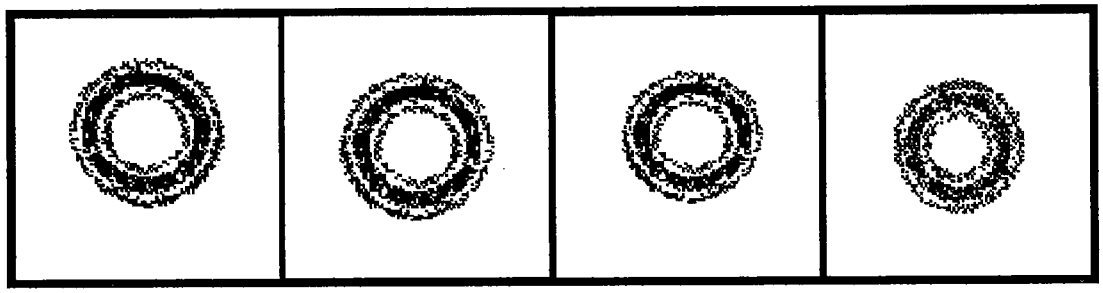


Imagem	Xav	Yav	Rav	s	sen	cos	P
ruid7t	-7	0	24.98	0.91	0.9067	-0.4218	1465

Figura 5.33. Imagens Pré-Processadas - Exemplo 14

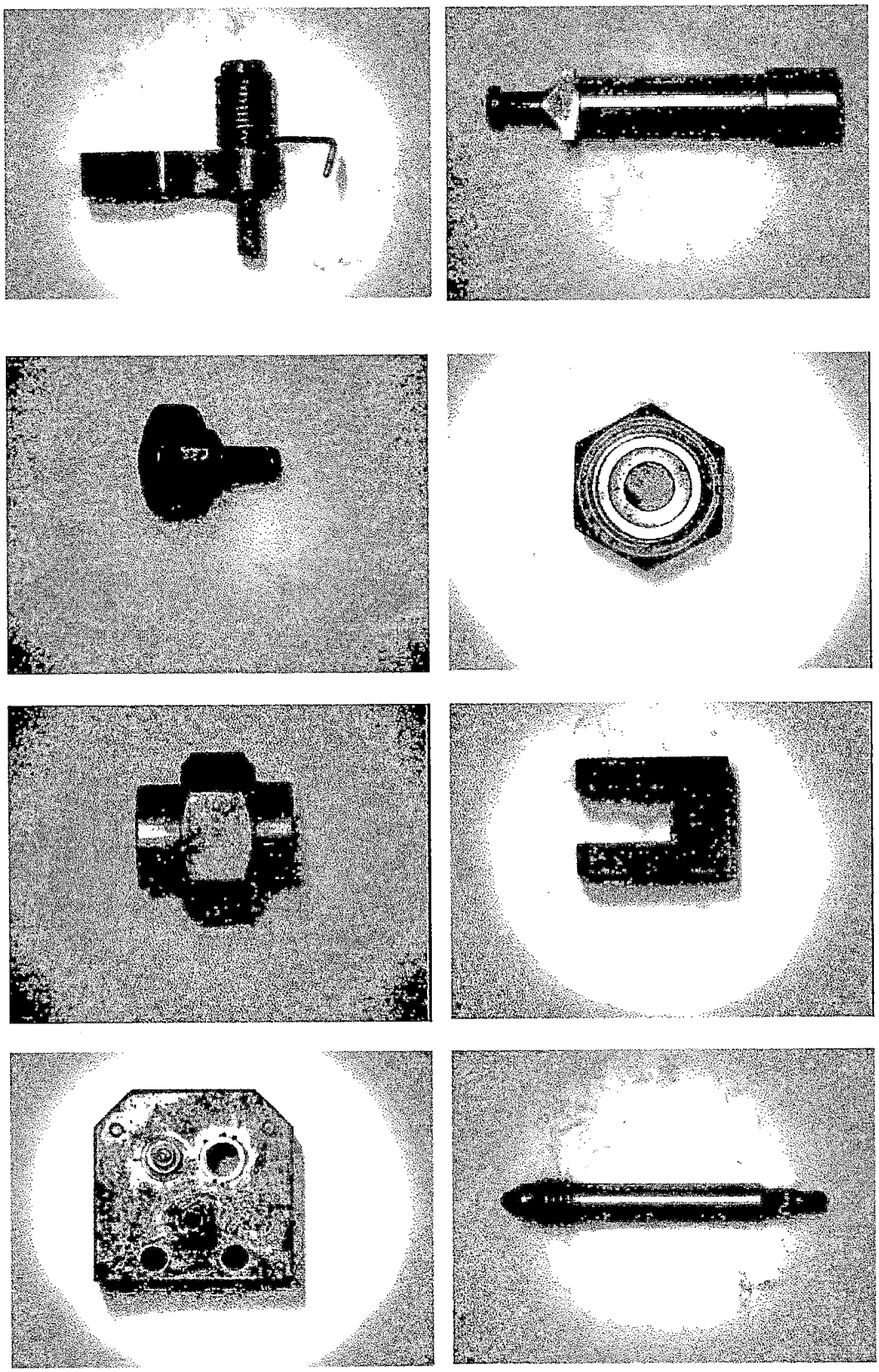


Fig 5.34. Imagens reais utilizadas - Exemplo1.

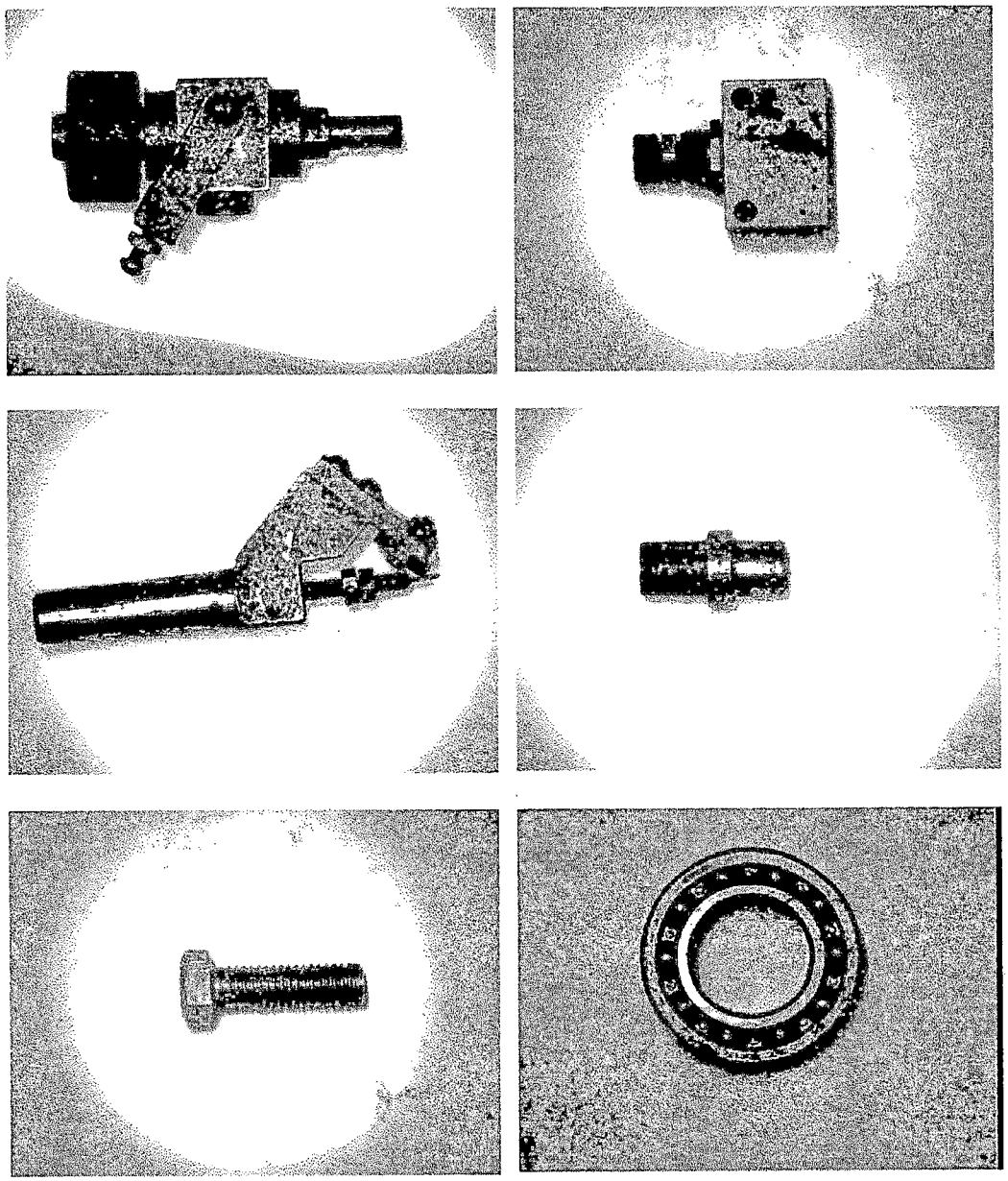


Figura 5.35. Imagens Reais utilizadas - Exemplo 2.

Bloco Gerador de Coarse Coading

As imagens apresentadas a seguir ilustram as imagens de saída do bloco gerador de coarse *coading*, para cada imagem de entrada. As imagens de entrada são as resultantes do modulo anterior (Pré-Processamento), com 128x128 pixels e as imagens resultantes (8 imagens) são de 16x16 pixels.

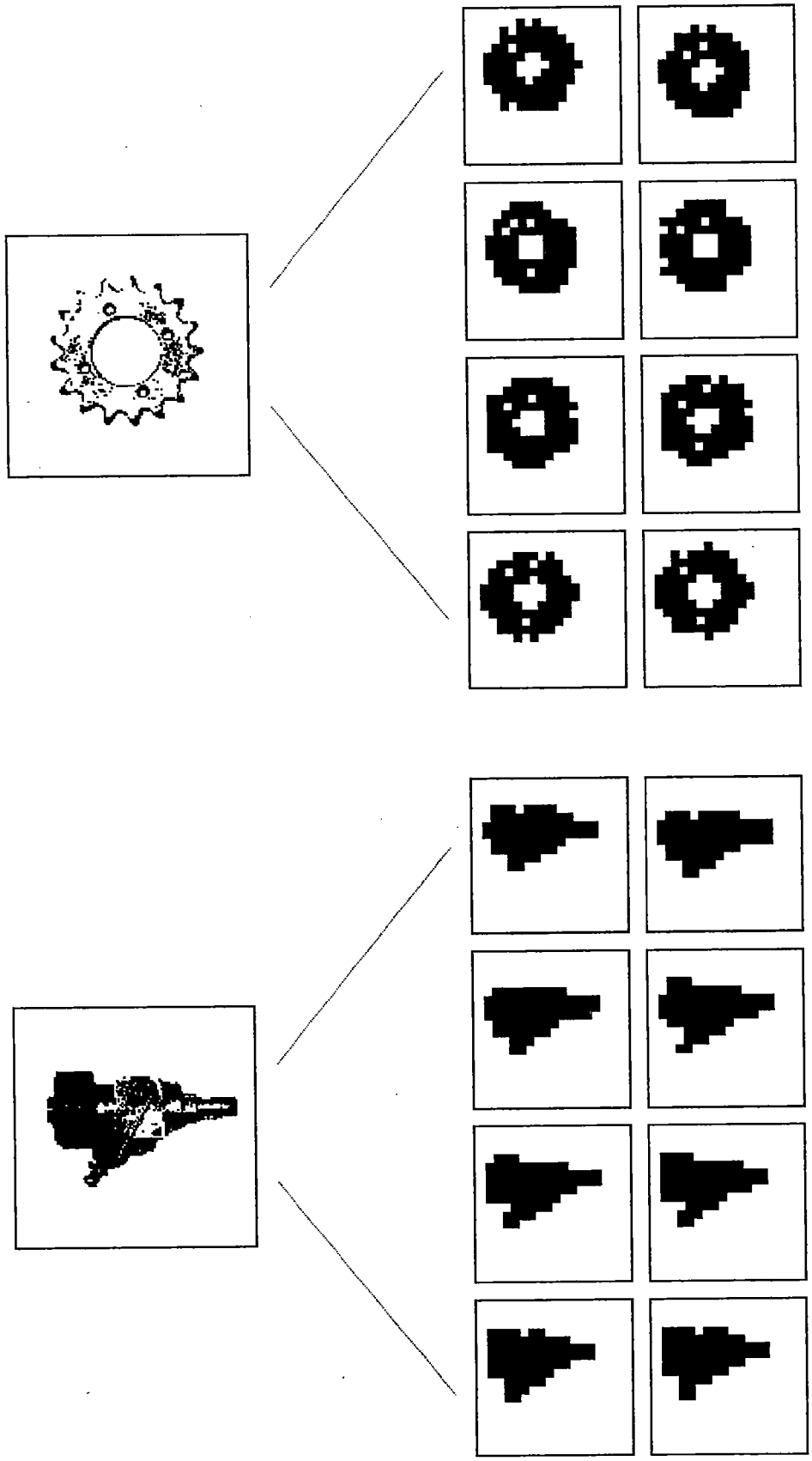


Figura 5.36. Geração de Campos Coarse - Exemplo 1.

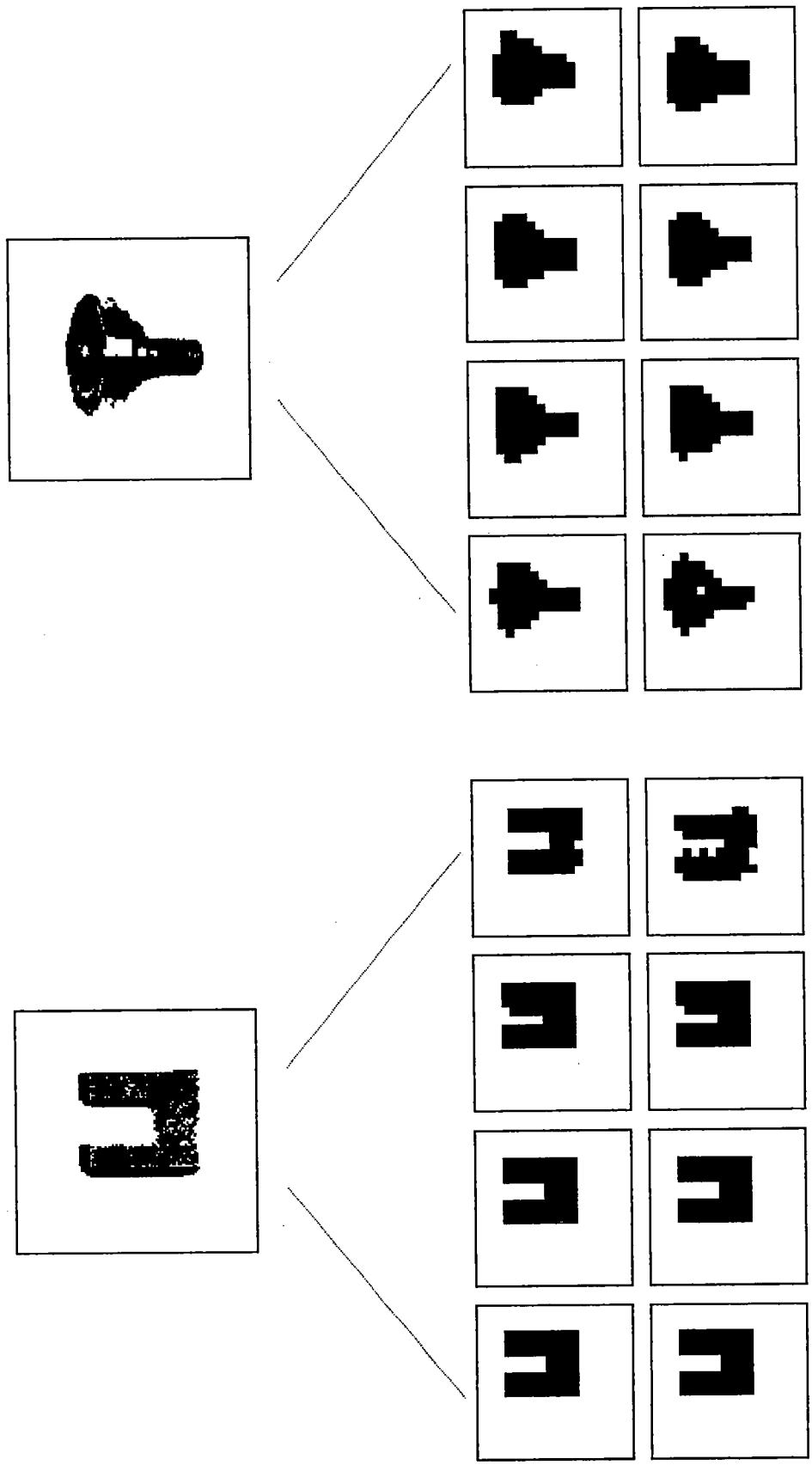


Figura 5.37. Geração de Campos Coarse - Exemplo 2.

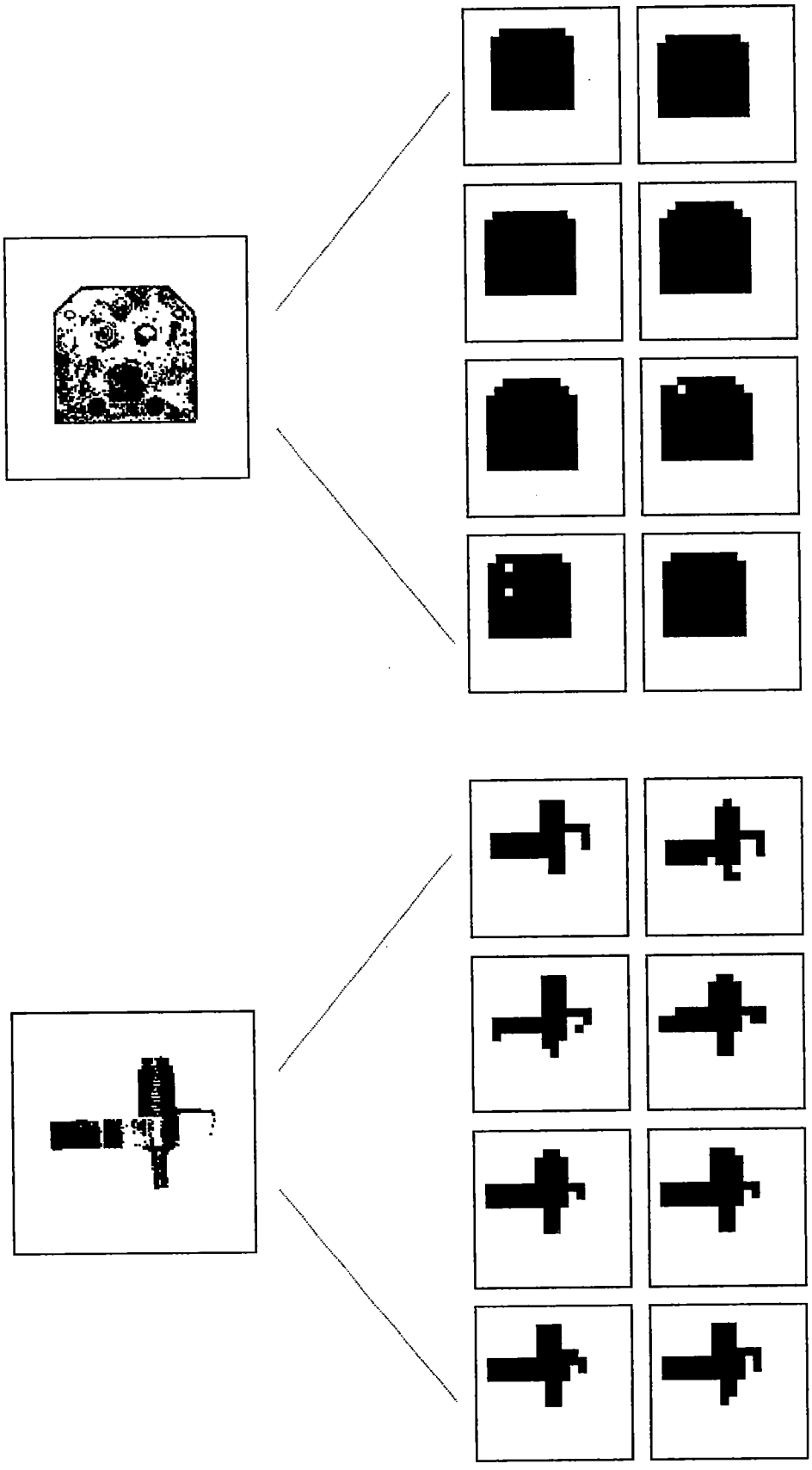


Figura 5.38. Geração de Campos Coarse - Exemplo 3.

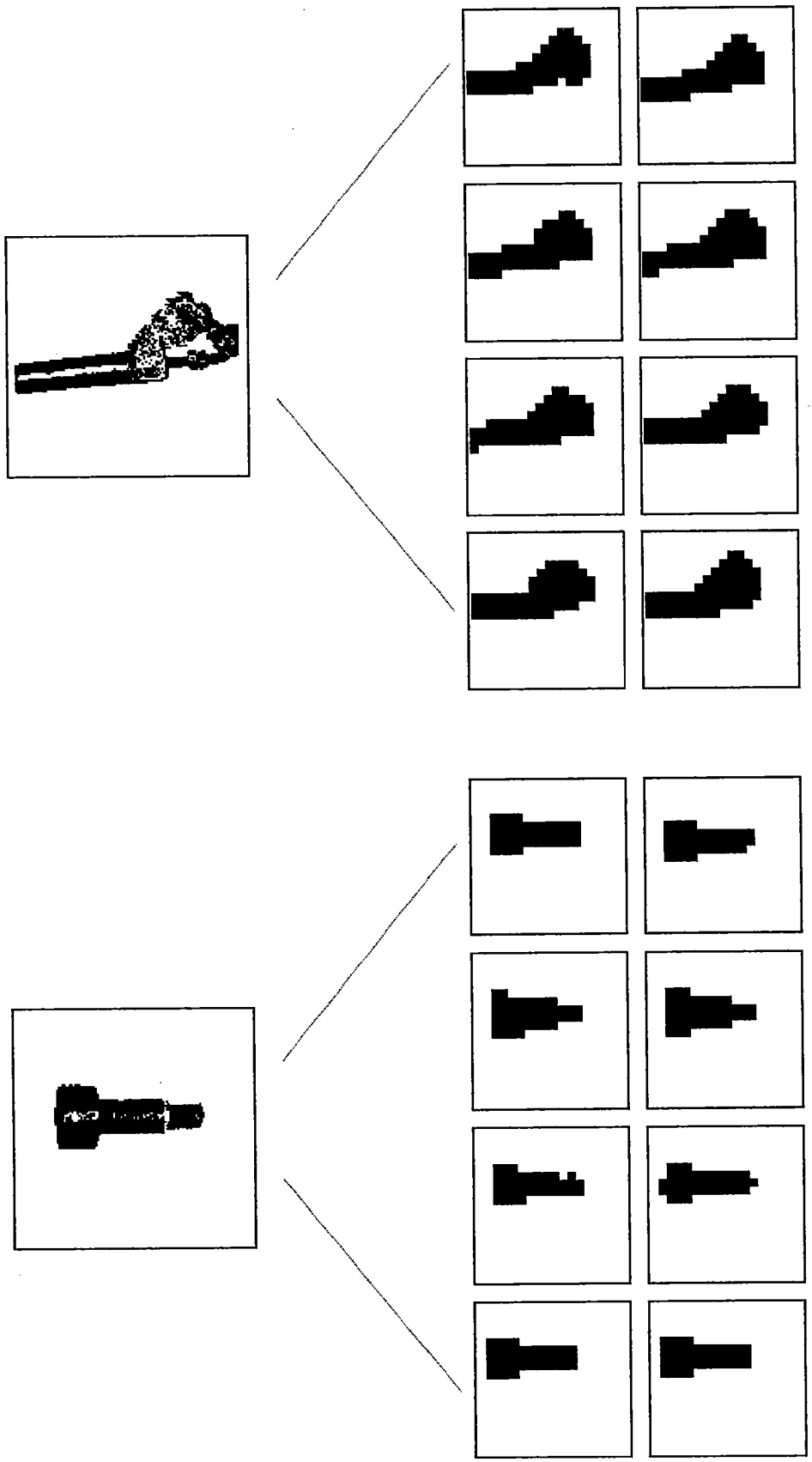


Figura 5.39. Geração de Campos Coarse - Exemplo 4.

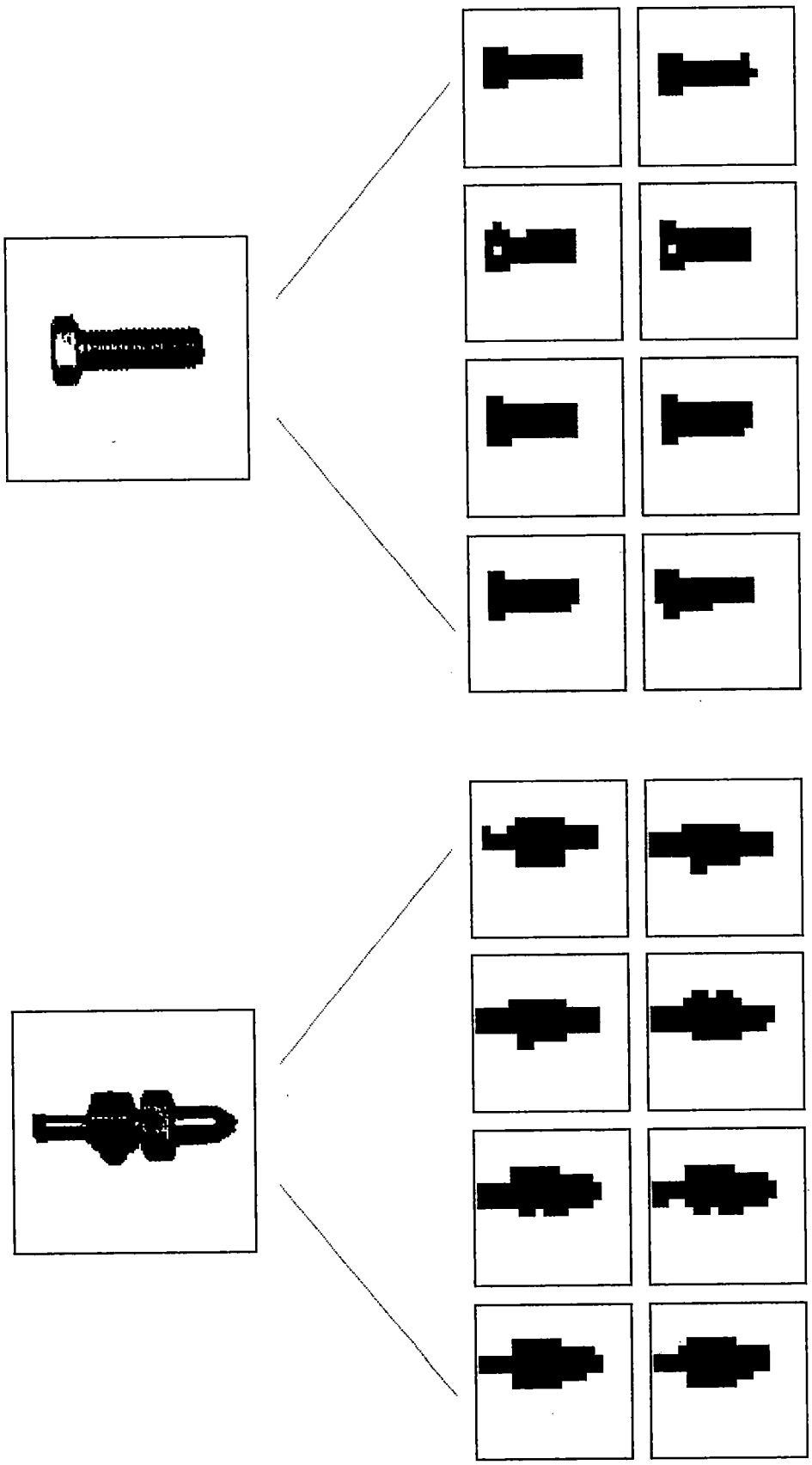


Figura 5.40. Geração de Campos Coarse - Exemplo 5.

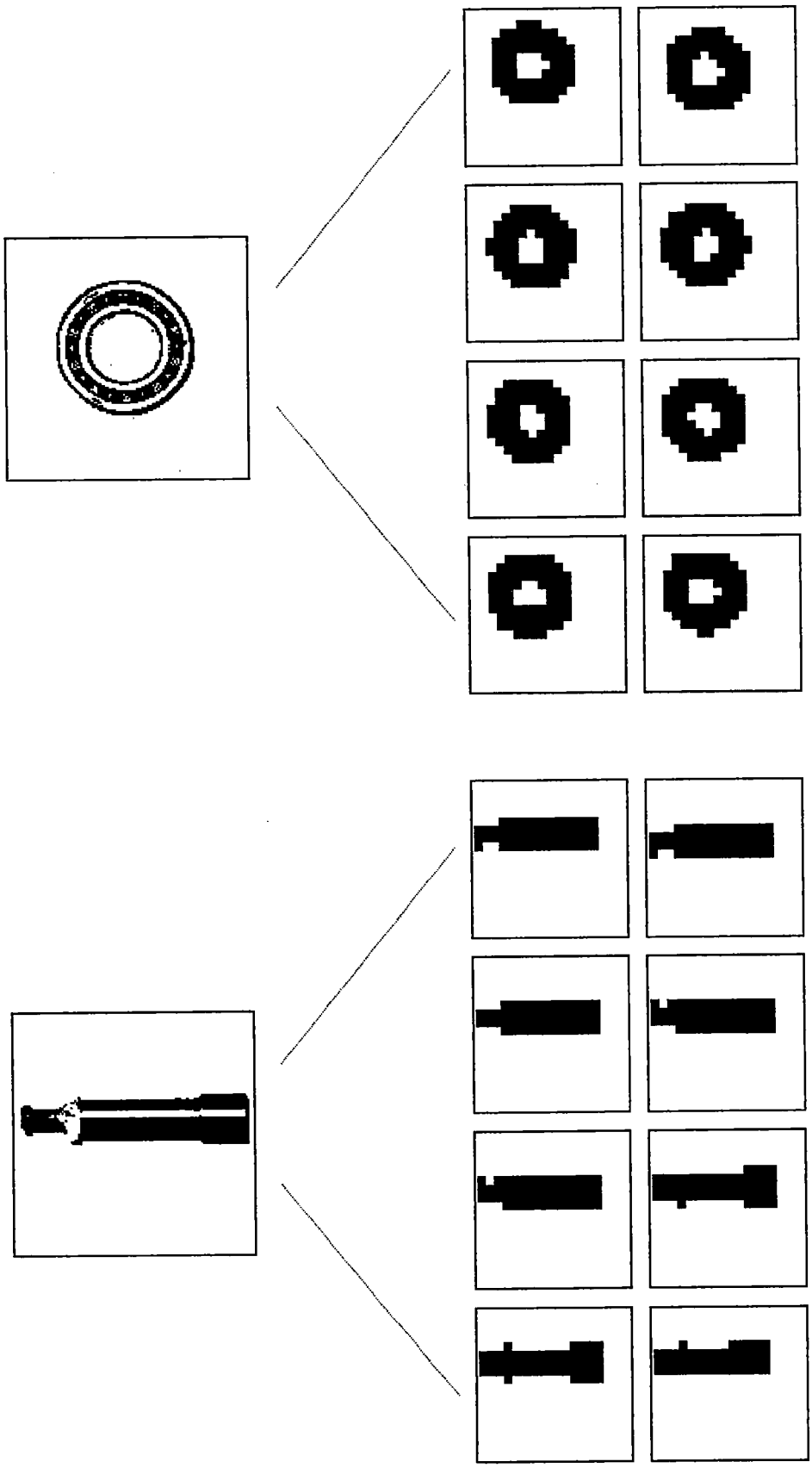


Figura 5.41. Geração de Campos Coarse - Exemplo 6.

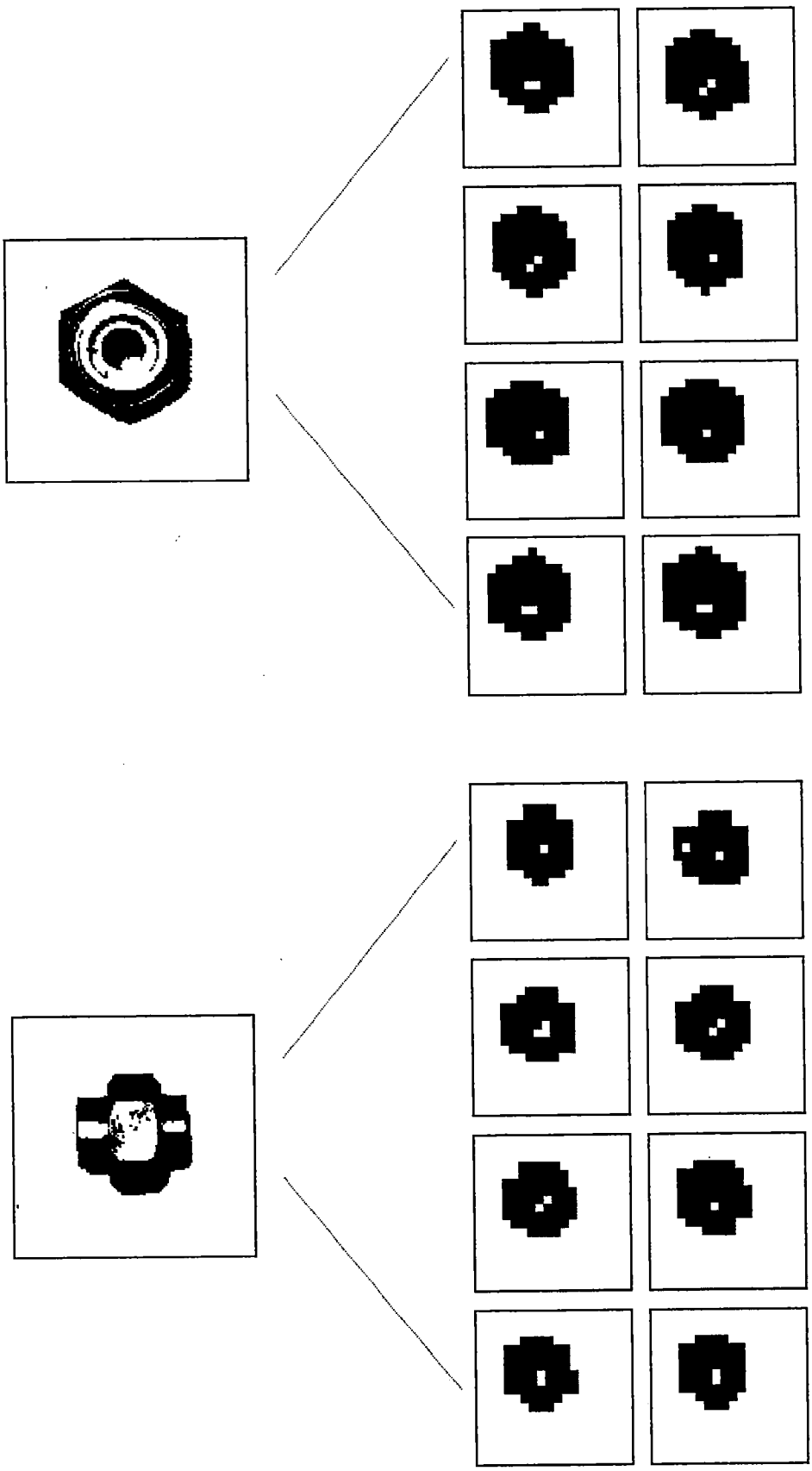


Figura 5.42. Geração de Campos Coarse - Exemplo 7.

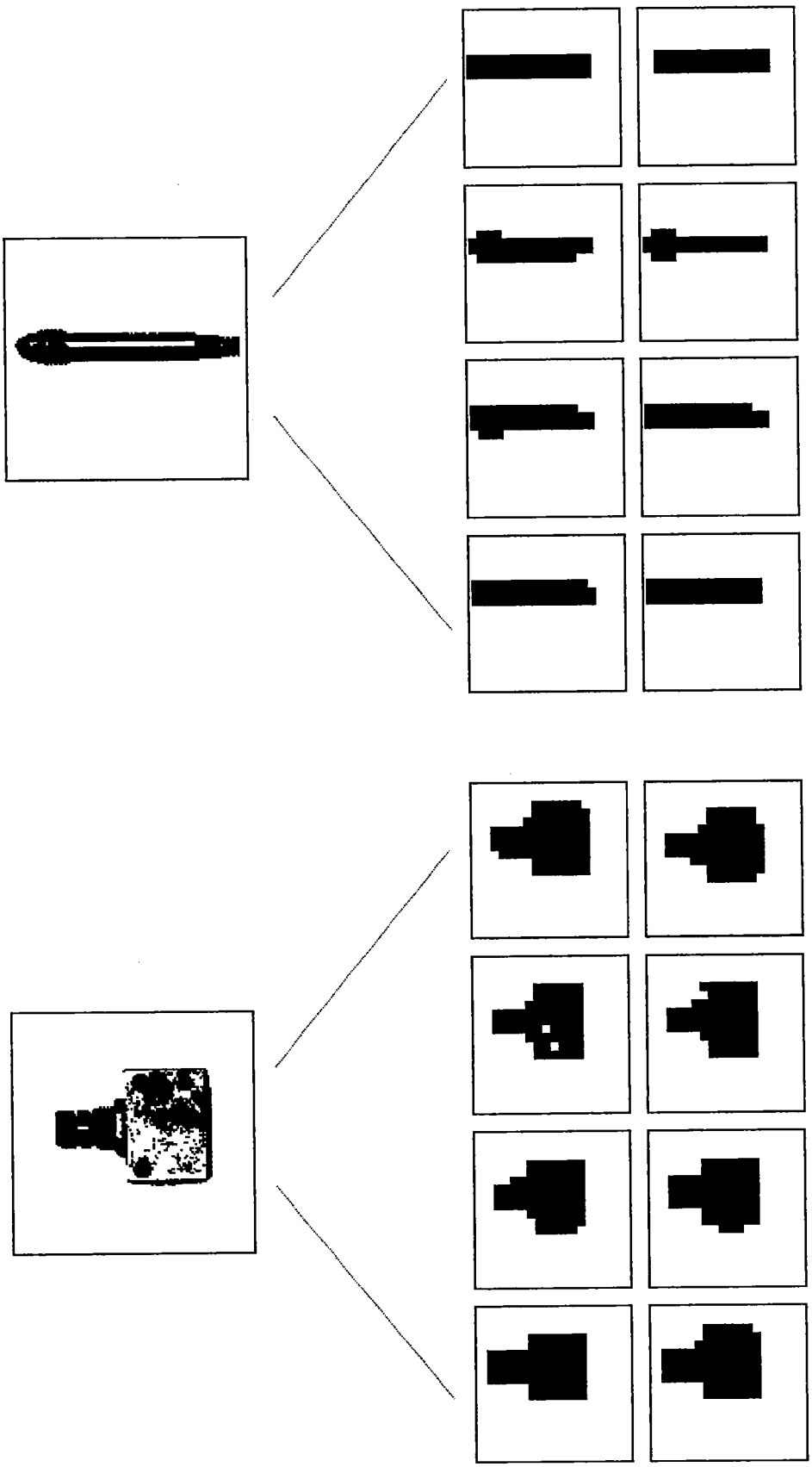


Figura 5.43. Geração de Campos Coarse - Exemplo 8.

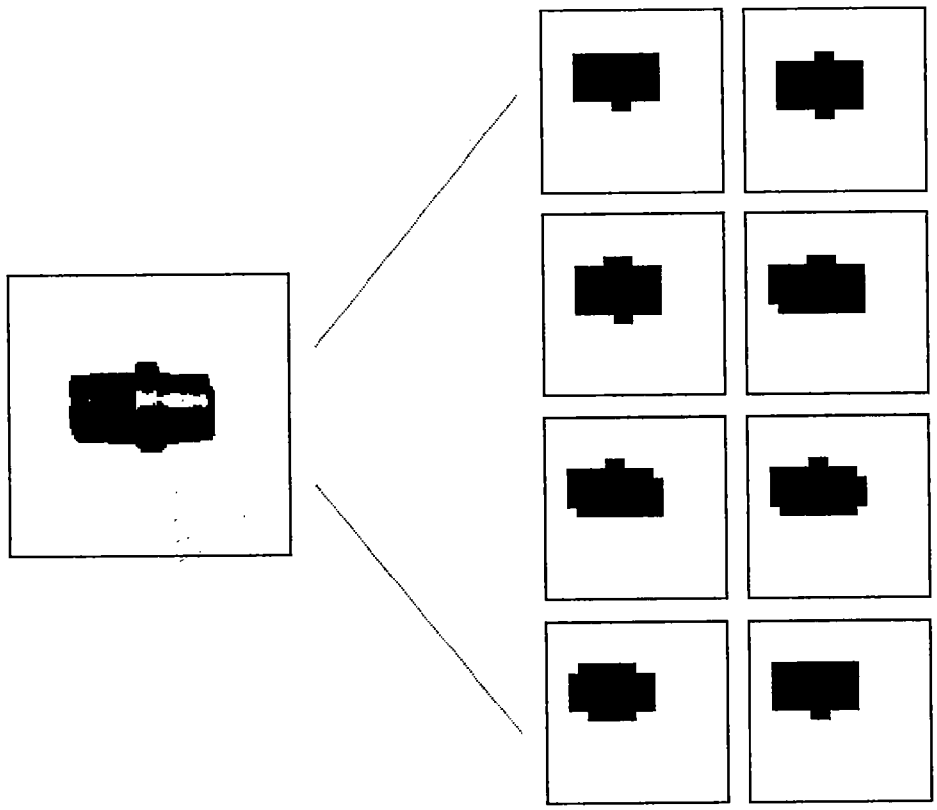


Figura 5.44. Geração de Campos Coarse - Exemplo 9.

Bloco de Reconhecimento

Foram feitos testes utilizando um conjunto de imagens para treinamento (20% do total) e imagens transformadas geometricamente para teste. A seguir, apresentam-se os resultados (taxa de reconhecimento) para diversas configurações do sistema, ou seja, utilizando 3, 5 e 11 pirâmides por bit dos códigos de saída e 25, 27, 32, 64, 81, 125 e 128 entradas por pirâmide.

Os padrões de entrada para este bloco são as imagens de campos coarse geradas. Assim, cada padrão e representado pelos seus 8 campos coarse de 16x16 pixels.

Cada lote testado (de um total de 100 lotes) é composto por 100 padrões de imagens (campos coarse).

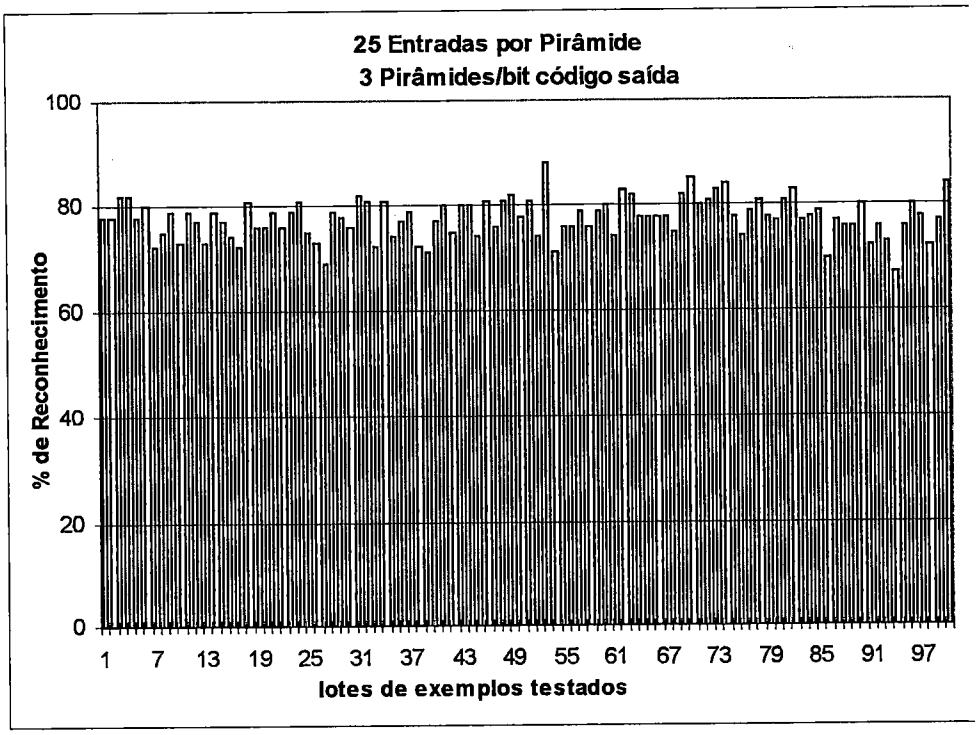


Figura 5.45. Taxa de Reconhecimento - 25 Entradas por Pirâmide e 3 Pirâmides por bit de código de saída.

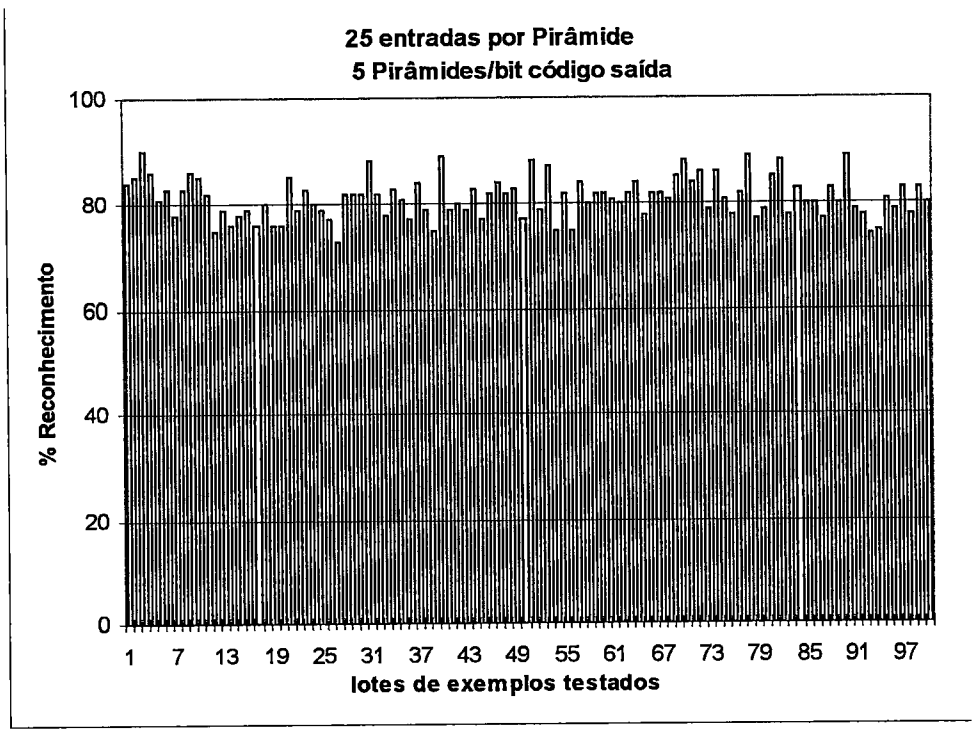


Figura 5.46. Taxa de Reconhecimento - 25 Entradas por Pirâmide e 5 Pirâmides por bit de código de saída.

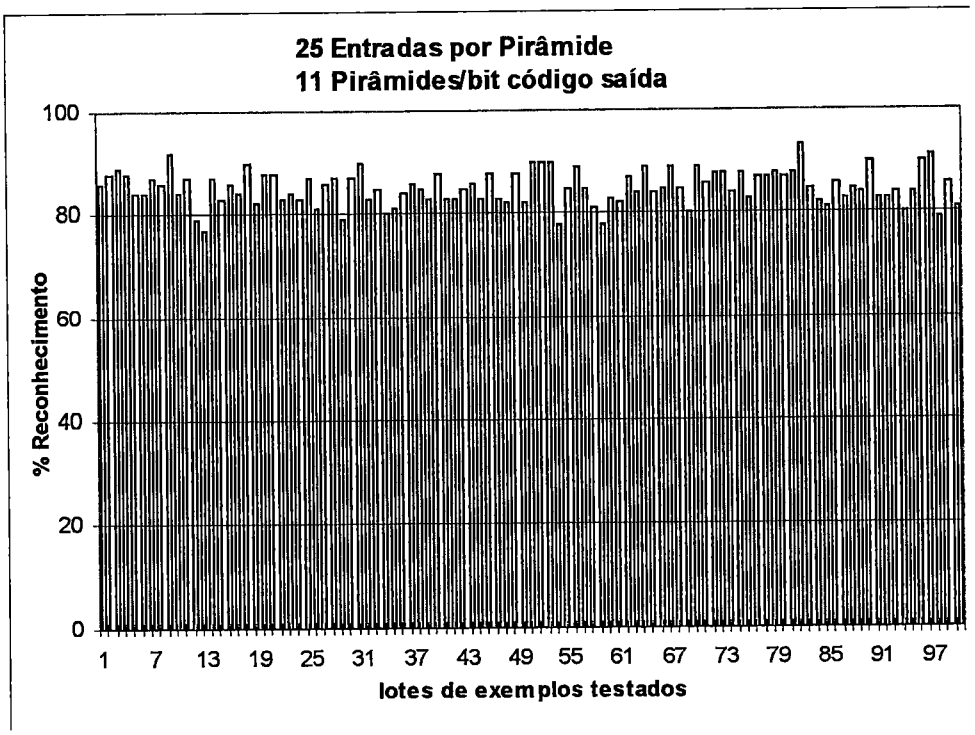


Figura 5.47. Taxa de Reconhecimento - 25 Entradas por Pirâmide e 11 Pirâmides por bit de código de saída.

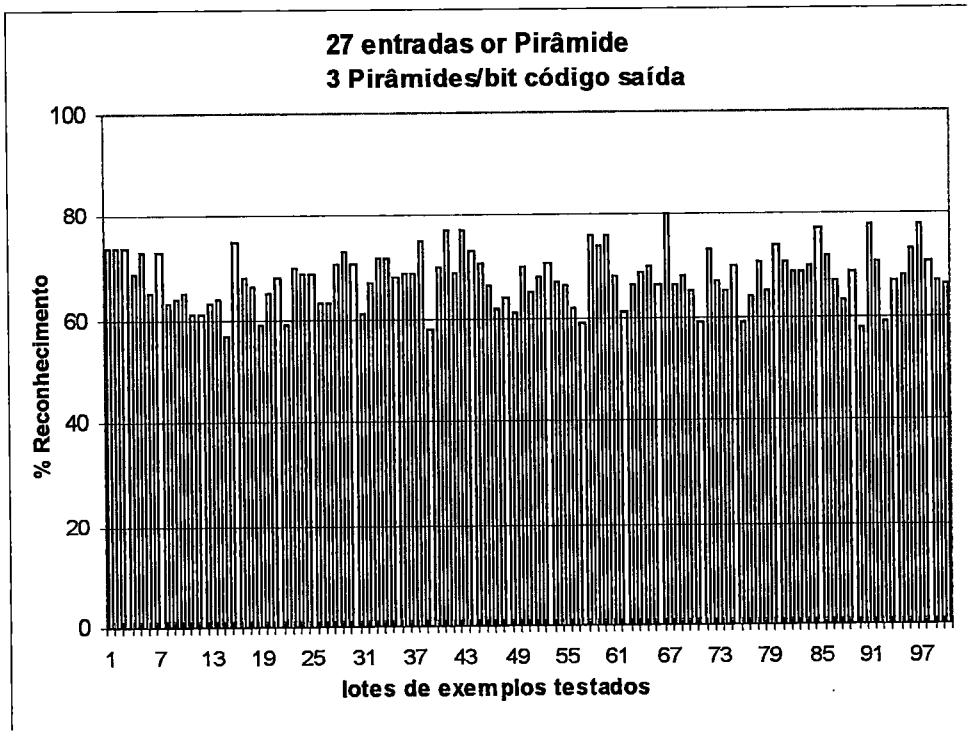


Figura 5.48. Taxa de Reconhecimento - 27 Entradas por Pirâmide e 3 Pirâmides por bit de código de saída.

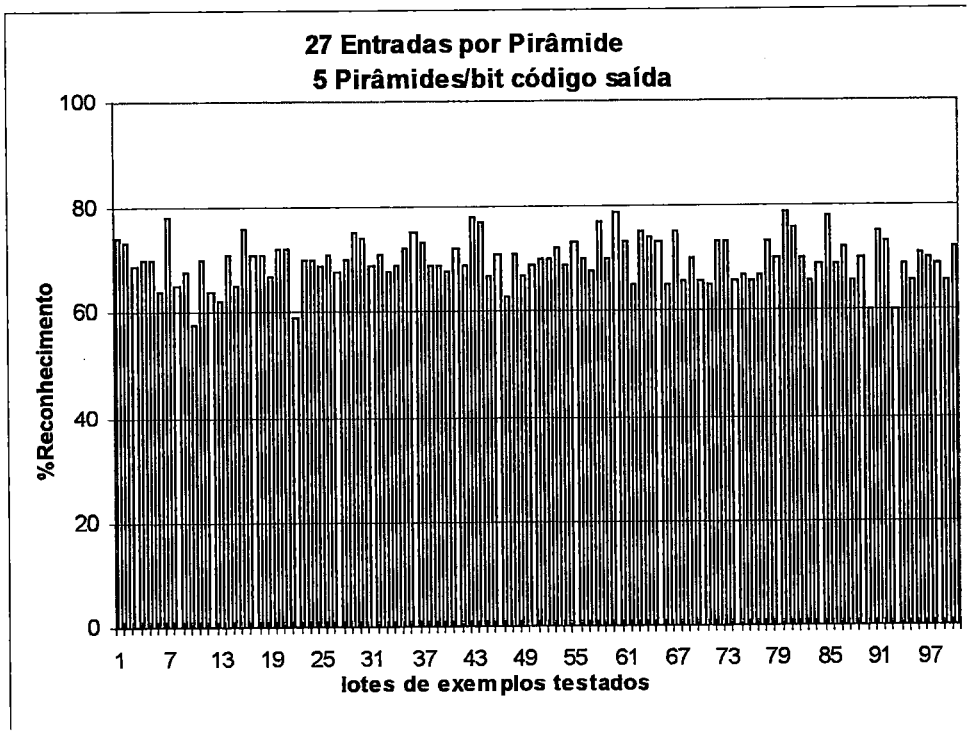


Figura 5.49. Taxa de Reconhecimento - 27 Entradas por Pirâmide e 5 Pirâmides por bit de código de saída.

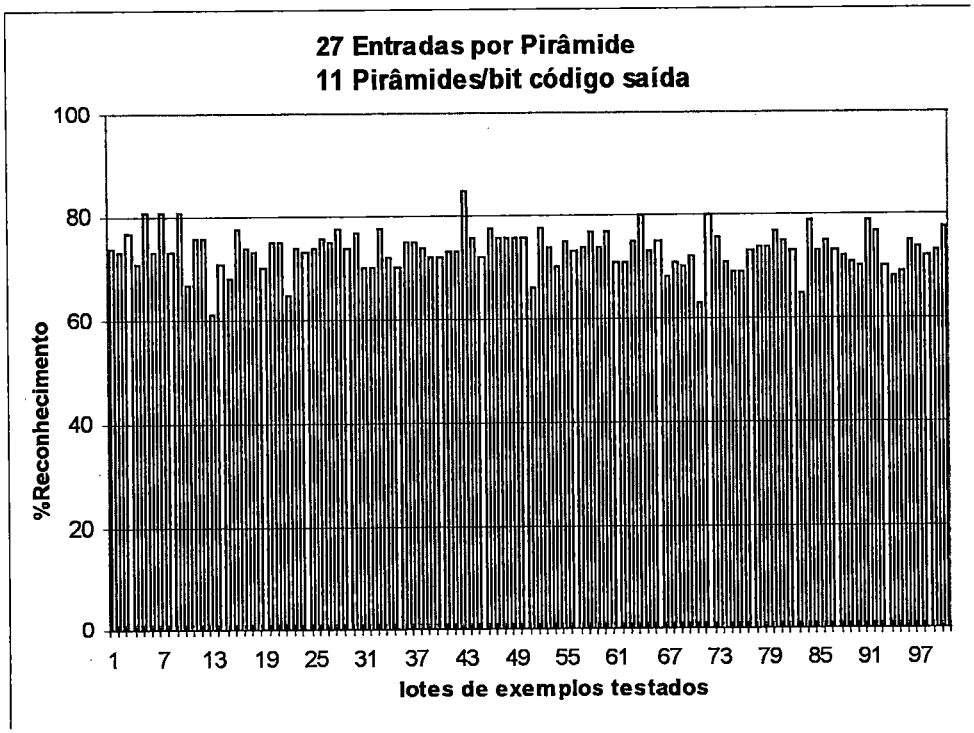


Figura 5.50. Taxa de Reconhecimento - 27 Entradas por Pirâmide e 11 Pirâmides por bit de código de saída.

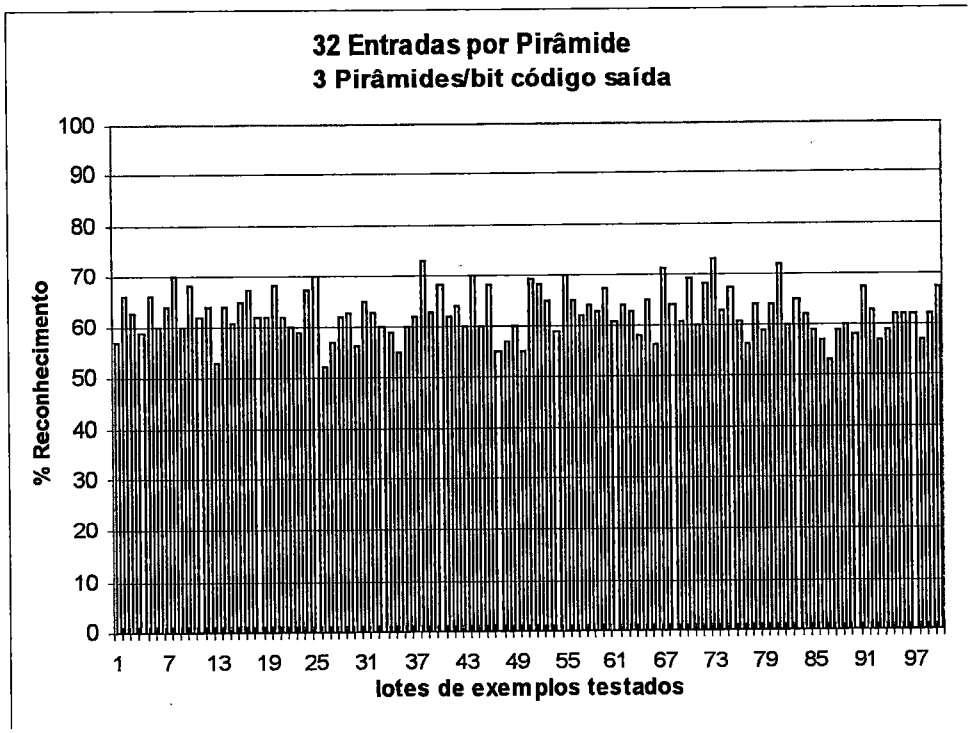


Figura 5.51. Taxa de Reconhecimento - 32 Entradas por Pirâmide e 3 Pirâmides por bit de código de saída.

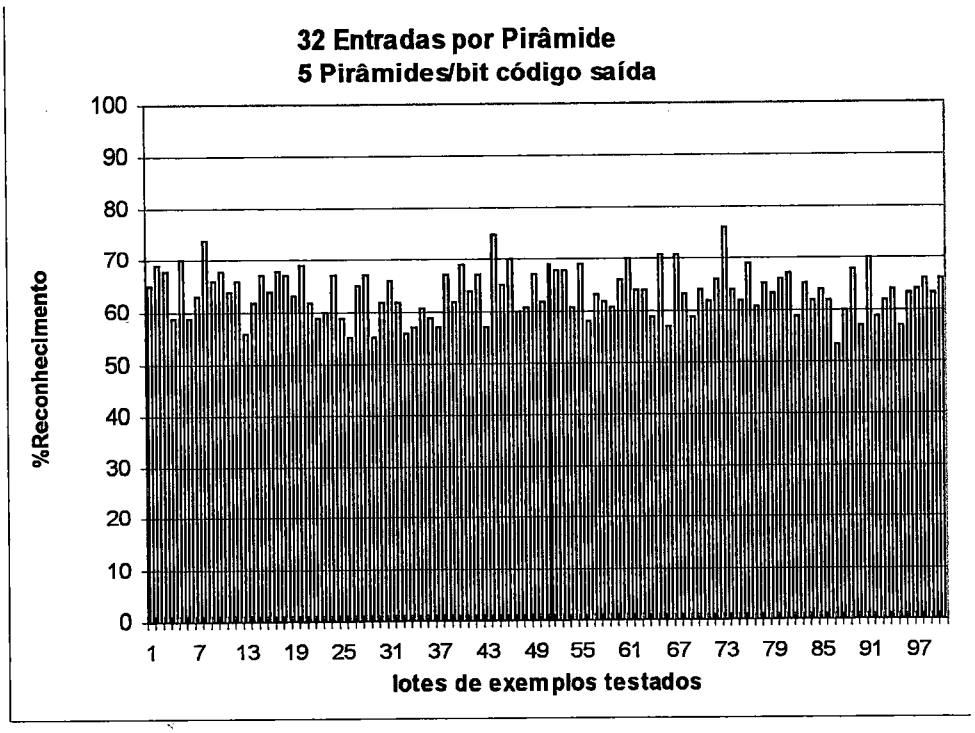


Figura 5.52. Taxa de Reconhecimento - 32 Entradas por Pirâmide e 5 Pirâmides por bit de código de saída.

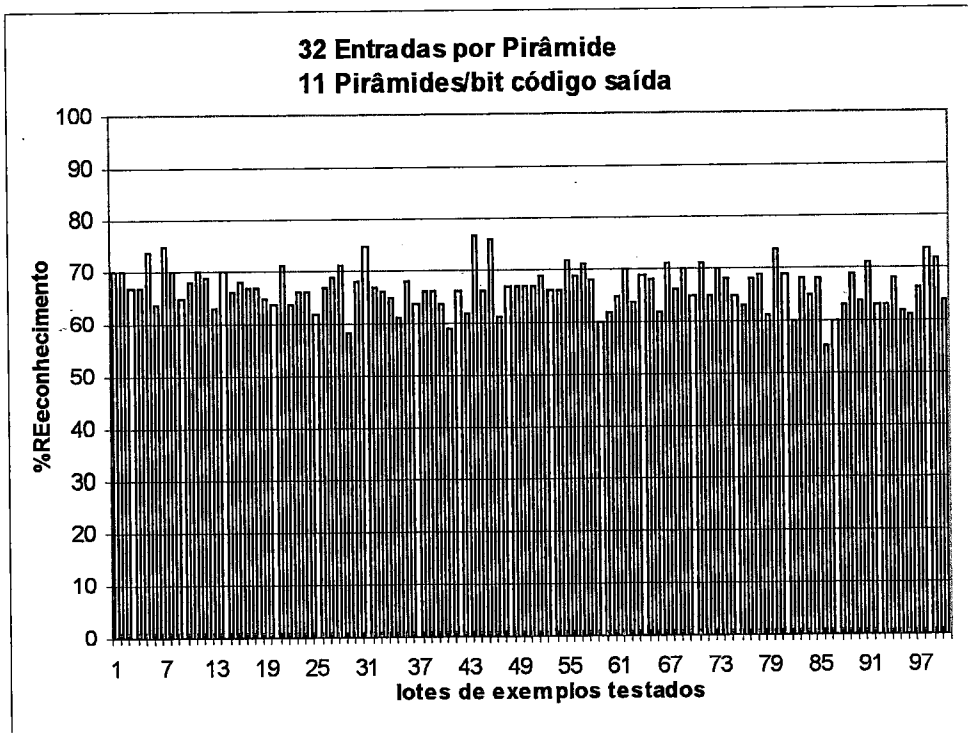


Figura 5.53. Taxa de Reconhecimento - 32 Entradas por Pirâmide e 11 Pirâmides por bit de código de saída.

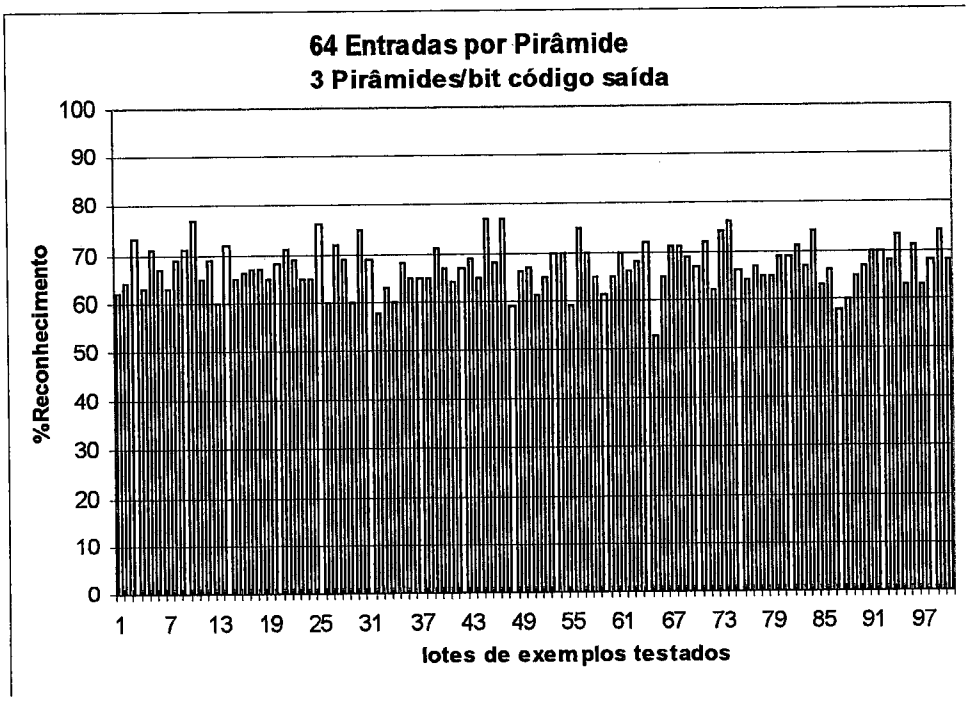


Figura 5.54. Taxa de Reconhecimento - 64 Entradas por Pirâmide e 3 Pirâmides por bit de código de saída.

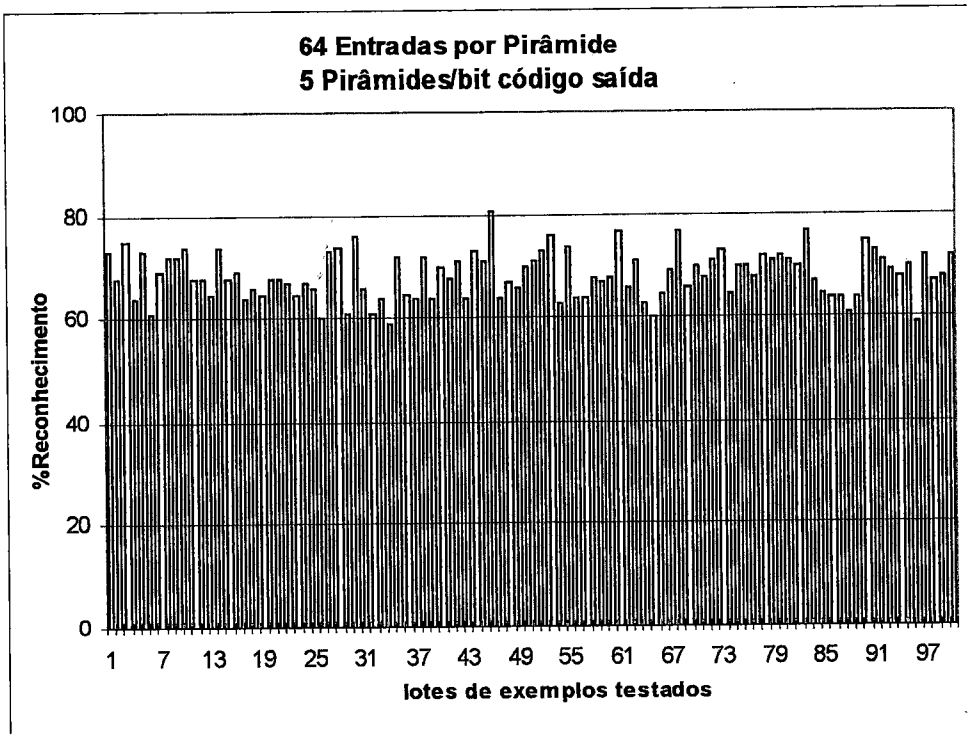


Figura 5.55. Taxa de Reconhecimento - 64 Entradas por Pirâmide e 5 Pirâmides por bit de código de saída.

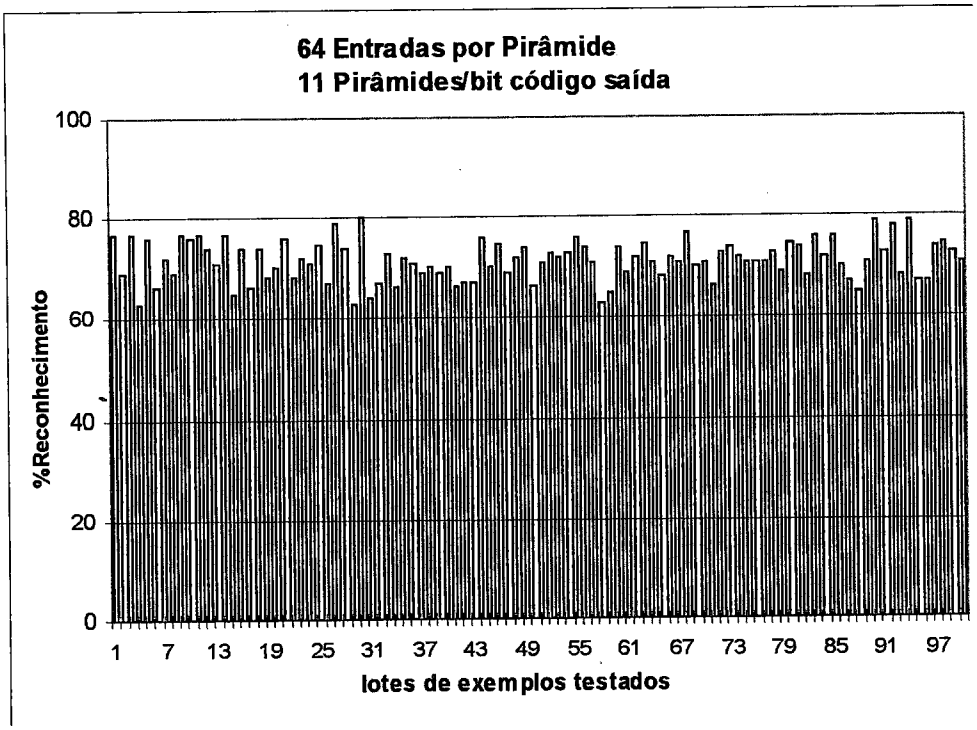


Figura 5.56. Taxa de Reconhecimento - 64 Entradas por Pirâmide e 11 Pirâmides por bit de código de saída.

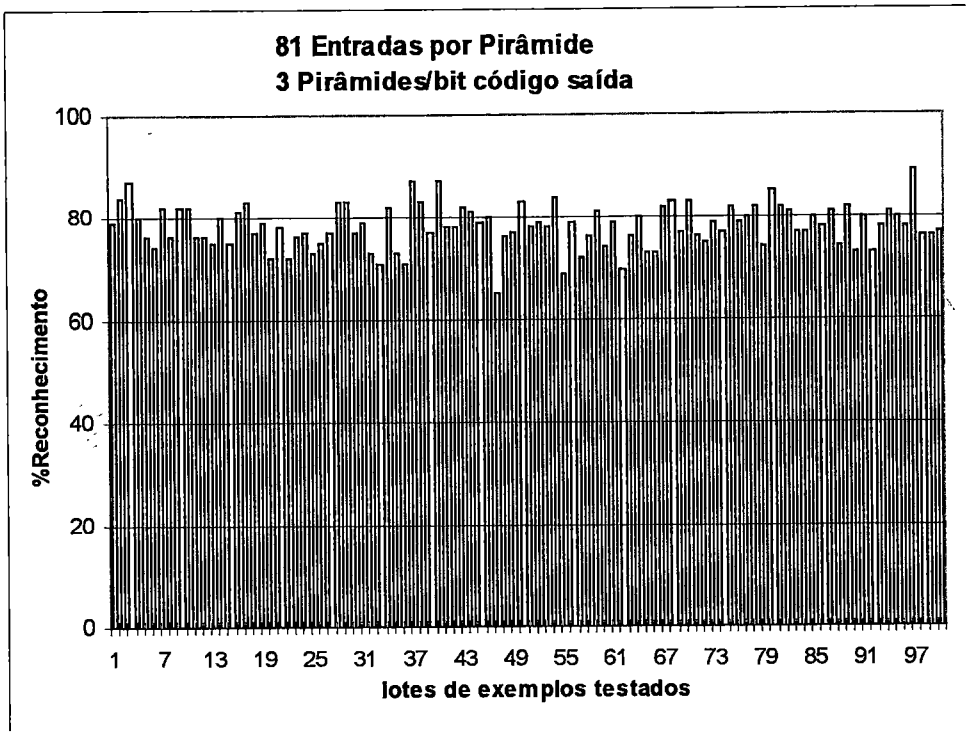


Figura 5.57. Taxa de Reconhecimento - 81 Entradas por Pirâmide e 3 Pirâmides por bit de código de saída.

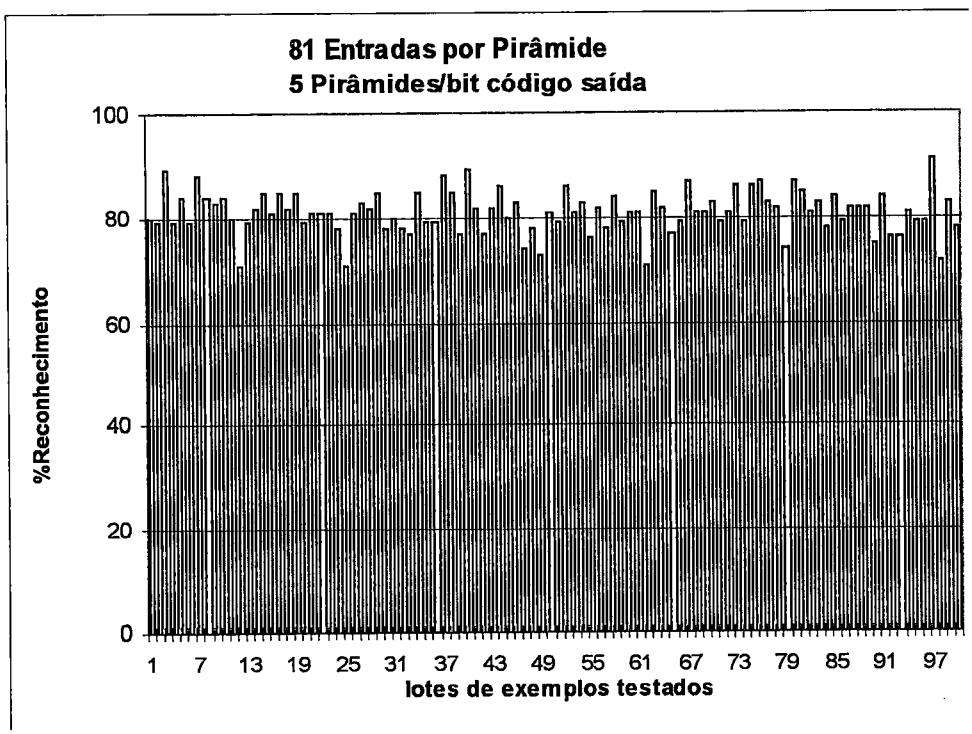


Figura 5.58. Taxa de Reconhecimento - 81 Entradas por Pirâmide e 5 Pirâmides por bit de código de saída.

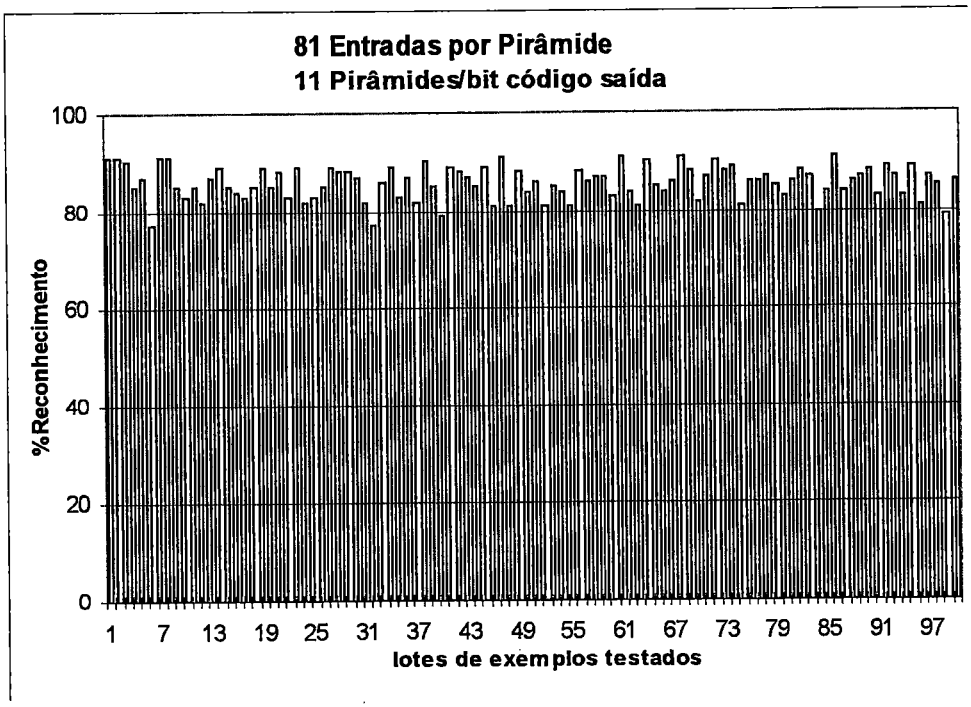


Figura 5.59. Taxa de Reconhecimento - 81 Entradas por Pirâmide e 11 Pirâmides por bit de código de saída.

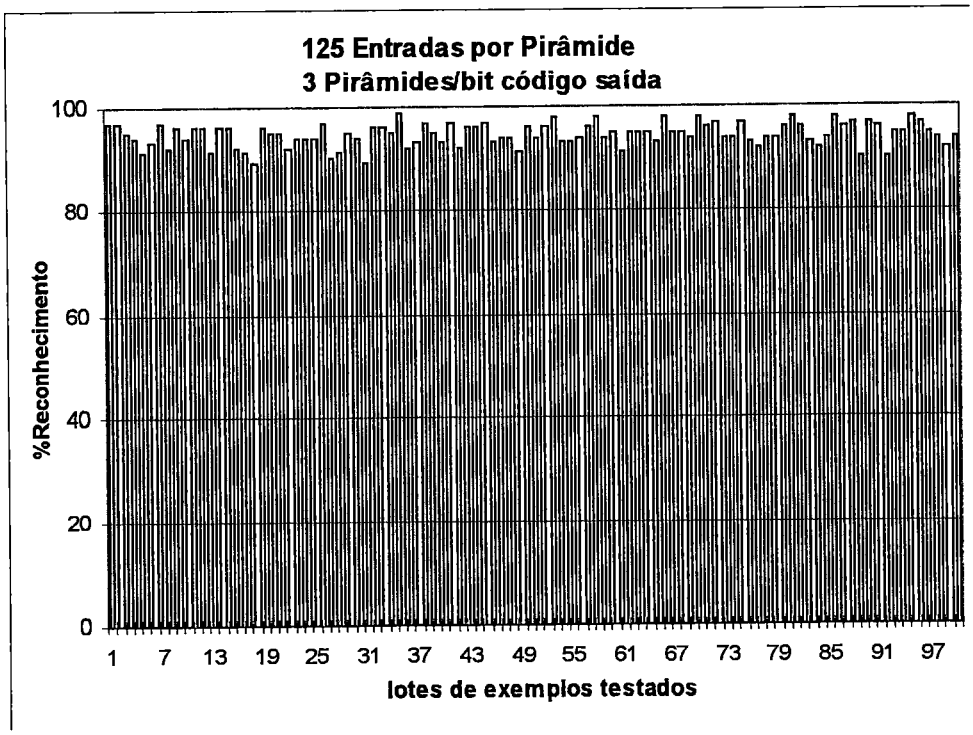


Figura 5.60. Taxa de Reconhecimento - 125 Entradas por Pirâmide e 3 Pirâmides por bit de código de saída.

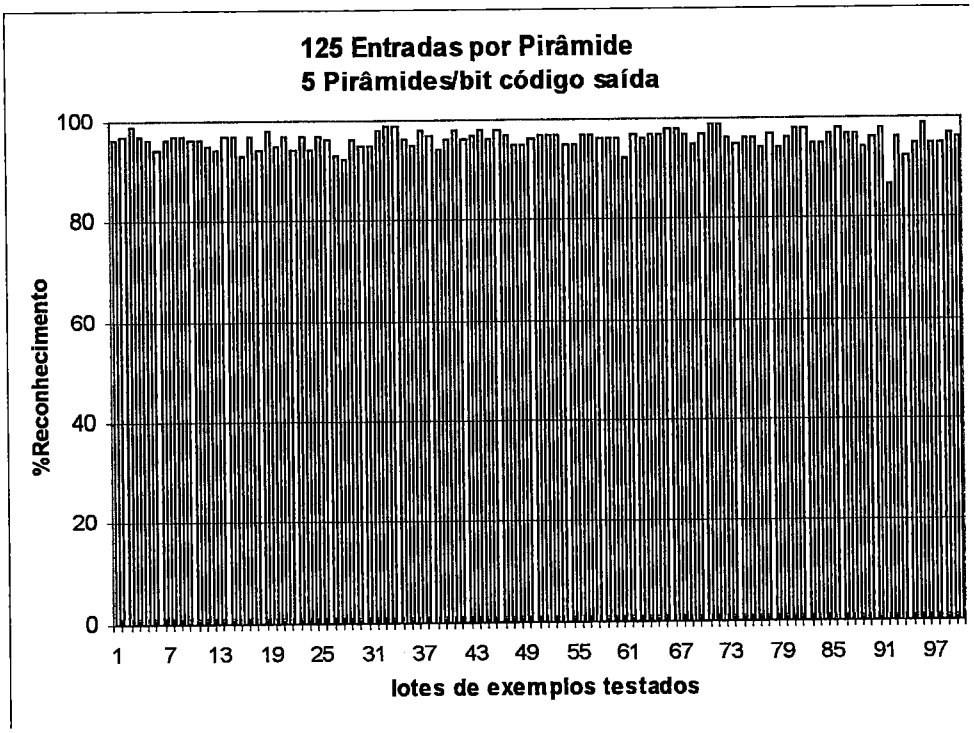


Figura 5.61. Taxa de Reconhecimento - 125 Entradas por Pirâmide e 5 Pirâmides por bit de código de saída.

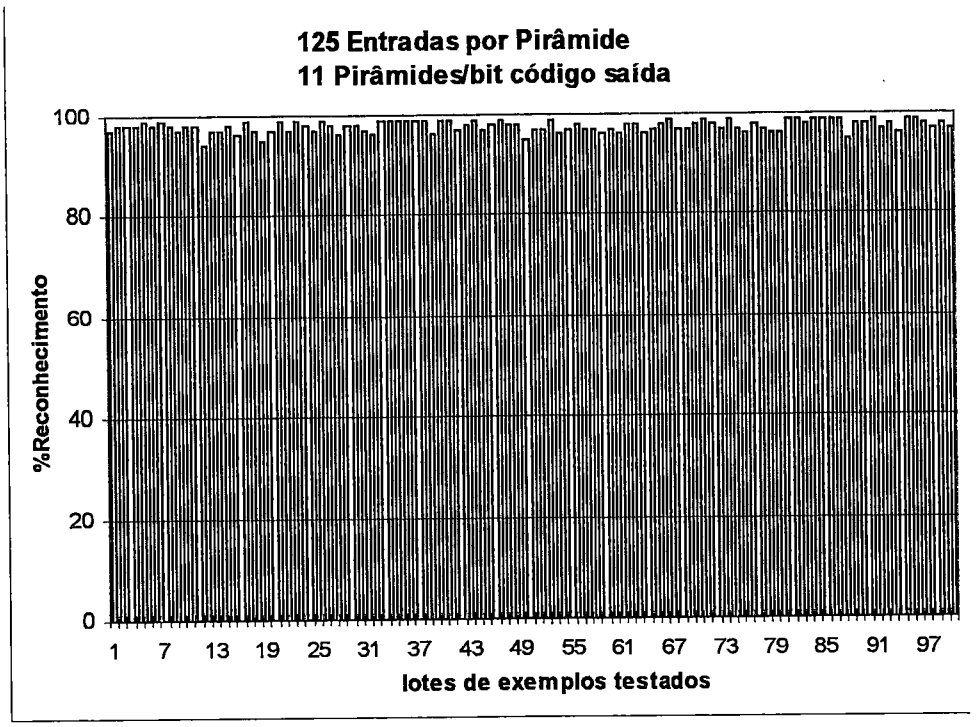


Figura 5.62. Taxa de Reconhecimento - 125 Entradas por Pirâmide e 11 Pirâmides por bit de código de saída.

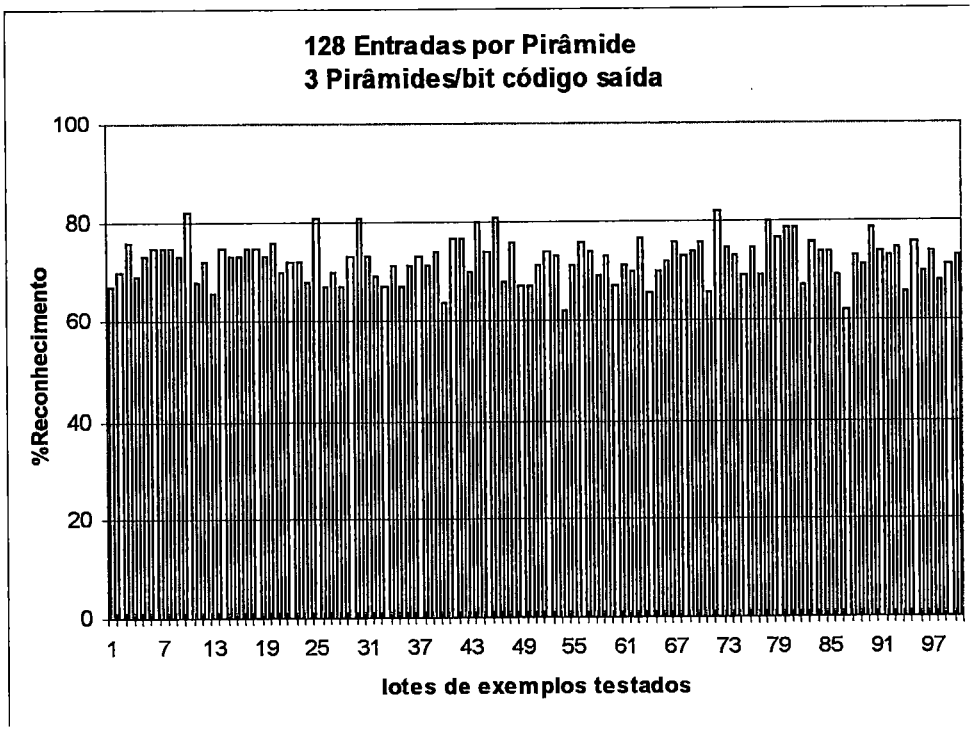


Figura 5.63. Taxa de Reconhecimento - 128 Entradas por Pirâmide e 3 Pirâmides por bit de código de saída.

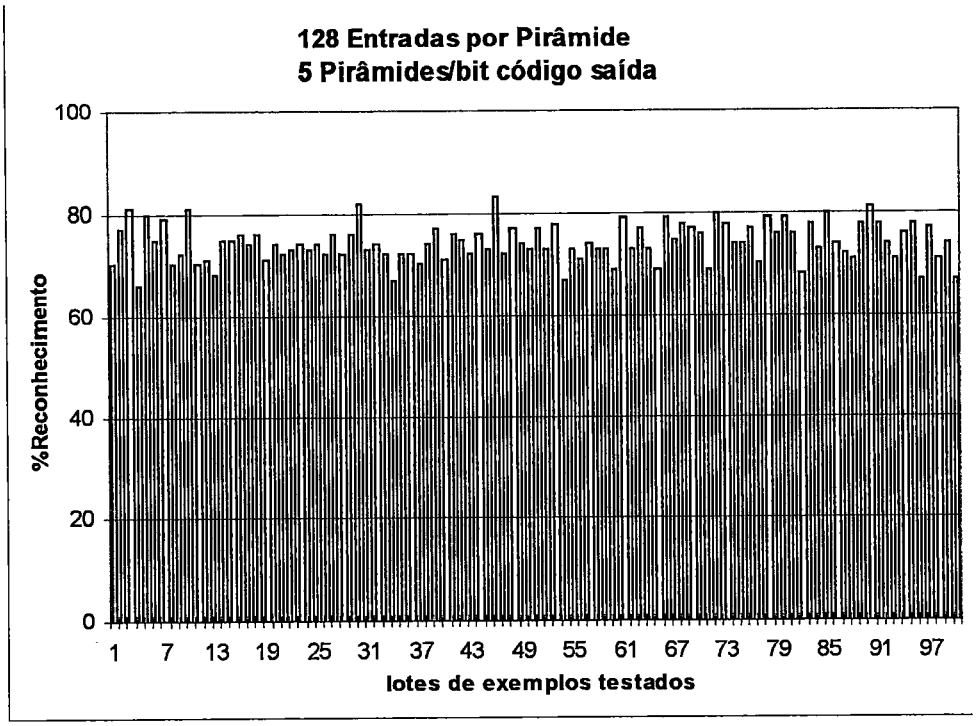


Figura 5.64. Taxa de Reconhecimento - 128 Entradas por Pirâmide e 5 Pirâmides por bit de código de saída.

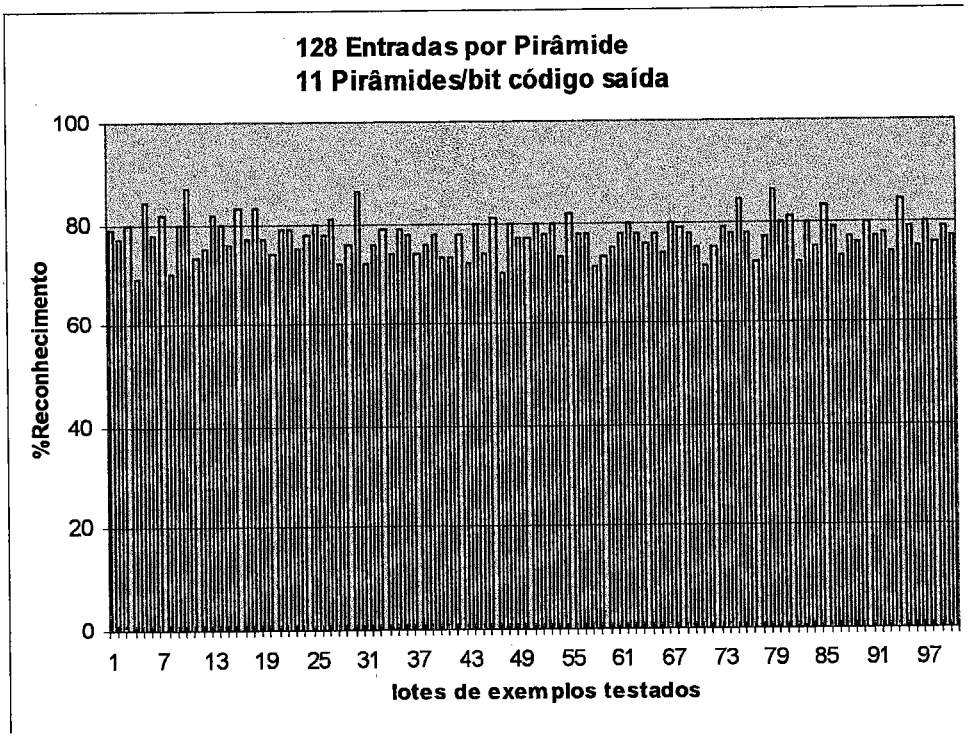


Figura 5.65. Taxa de Reconhecimento - 128 Entradas por Pirâmide e 11 Pirâmides por bit de código de saída.

Análise dos resultados:

Como mostrado nas figuras anteriores, o desempenho de reconhecimento melhora com o aumento do número de pirâmides por bit do código de saída. Esta melhora é mais acentuada para o caso de 25, 81 e 125 entradas por pirâmide. O desempenho de reconhecimento aumenta sensivelmente com o aumento do número de entradas por pirâmide, de tal maneira que o melhor desempenho é obtido com 125 entradas. Uma excessão é o caso de 128 entradas, que tem desempenho inferior aos de 25 e 81 entradas por pirâmide.

Tempos de Processamento

O tempo de execução envolvido na tarefa de reconhecimento de uma imagem de entrada para o sistema é constituído pela soma do tempo de aquisição (aquisição mais filtragem), tempo de pré-processamento (blocos de translação, escala e rotação), tempo de geração de coarse coadings e tempo de classificação pela Rede Neural.

Os tempos de processamento descritos a seguir são diretamente relacionados à velocidade do processador utilizado no sistema. Para efeito de comparação, apresenta-se a seguir os resultados obtidos através de um Microcomputador Pentium - 133MHz.

O tempo de aquisição + filtragem é o maior gargalo do sistema, já que para uma janela de 3x3, a filtragem do tipo mediana leva em torno de 0.1 segundos (tempo médio).

O tempo de pré-processamento é diretamente relacionado ao número de pixels "1" presentes na imagem. O tempo total é equivalente a:

9P + 2G adições + 5P multiplicações + P cálculos de Raiz

onde:

P = número de pixels "1" na imagem;
G = número de pixels na imagem;

Para as imagens utilizadas neste trabalho os tempos de pré-processamento obtidos foram:

- Bloco de translação: 0.017 segundos por padrão, em media;
- Bloco de escala: 0.015 segundos por padrão, em media;
- Bloco de rotação: 0.018 segundos por padrão, em media.

O tempo de geração de campos *coarse* foi de 0.005 segundos por campo. Portanto, para cada padrão o tempo de geração de campos coarse foi de 8x0.005=0.04 segundos.

O tempo de treinamento obtido para os padrões utilizados neste trabalho variou de 18 a 33 segundos, em função dos parâmetros utilizados na configuração da rede.

O tempo de classificação obtido, para cada padrão foi de 0.005 segundos em media.

O tempo total de processamento é proporcional ao tamanho da imagem de entrada (maior gargalo na filtragem). As operações de pré-processamento, no entanto, não comprometem o requisito de operação em tempo real para imagens maiores (512 x 512, 1024 x 1024, etc.).

5.5 UTILIZAÇÃO DA ARQUITETURA CPAD NA IMPLEMENTAÇÃO DO SISTEMA DE RECONHECIMENTO DE IMAGENS

Como descrito no Capítulo 1, o objetivo deste trabalho foi o desenvolvimento de um sistema de reconhecimento de peças mecânicas o qual pudesse ser implementado através da arquitetura CPAD desenvolvida por OSHIRO (1998) e descrita em detalhes no Anexo 3. Assim, descreve-se a seguir a proposta de implementação do sistema desenvolvido sob a arquitetura em questão.

Para que o sistema de reconhecimento de imagens utilize as potencialidades da arquitetura CPAD, são necessários 5 módulos básicos, dispostos como na figura a seguir.

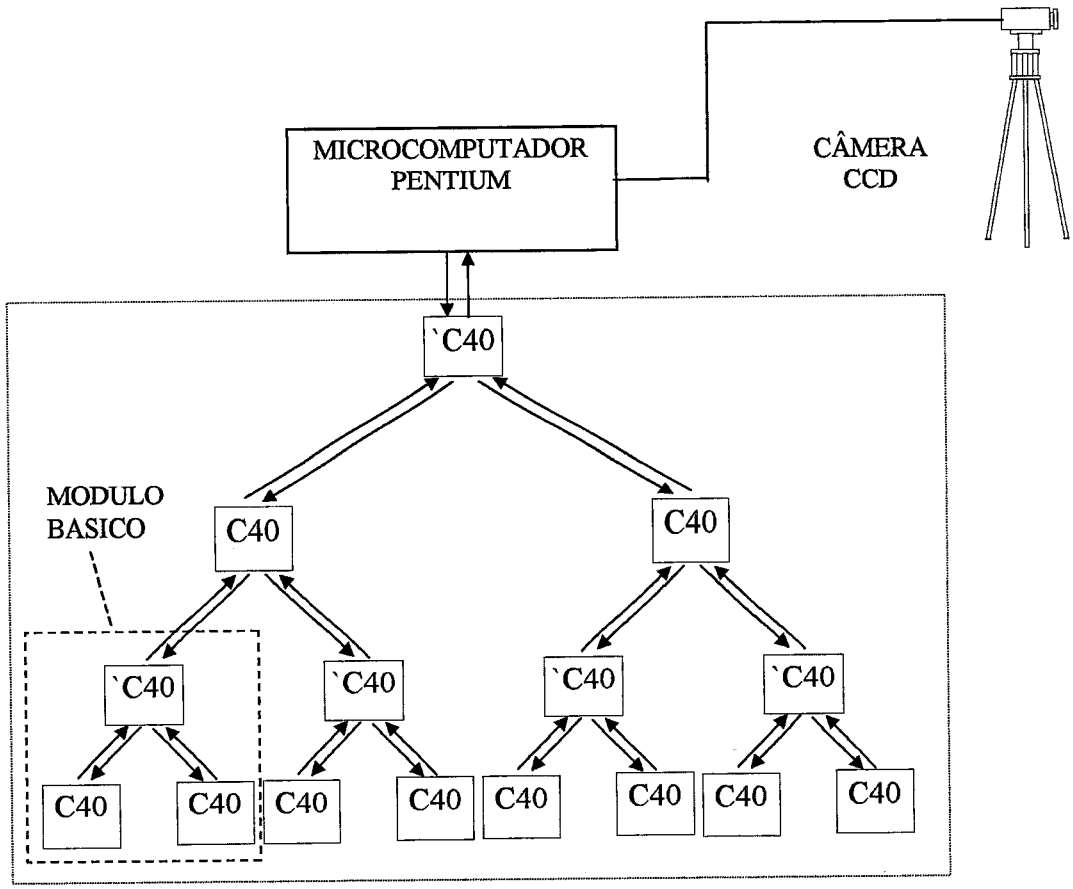


Figura 5.66. Implementação do sistema de reconhecimento de imagens através da arquitetura CPAD.

Através da utilização desta arquitetura, os módulos componentes do sistema devem ser distribuídos em elementos processadores (módulos básicos) de tal maneira que as tarefas de pré-processamento e reconhecimento da imagem e as rotinas do módulo de controle sejam realizadas em paralelo.

Basicamente, o sistema deverá operar da seguinte maneira:

- As tarefas de Pré-Processamento e Geração de Coarse Coading deverão ser concorrentes às tarefas de geração de dados de saída do módulo de controle; portanto deverão estar em módulos diferentes;
- Após gerados os campos coarse, cada um dos oito elementos de processamento (elementos mais distantes do Microcomputador) deverá se encarregar de testar uma das oito imagens, o que resulta em oito saídas em paralelo para a mesma imagem de entrada (128 x128 pixels) apresentada ao sistema. Por comparação do tipo *winner takes all*, uma saída é produzida, representando a imagem de entrada;
- Quando o sistema está na fase de treinamento, todos os campos coarse são ensinados a todos os oito elementos de processamento.

Tal implementação pode trabalhar com grandes imagens (1024x1024, 2048x2048 pixels) já que cada elemento processador da camada mais distante do Microcomputador pode avaliar, em paralelo, 1/8 dos campos coarse gerados. Assim, trabalhando com imagens de 2048x2048 pixels (altíssima definição), cada processador avaliaria duas imagens de 128x128 pixels (utilizando 16 campos coarse por imagem). Caso o número de campos coarse fosse 8, cada processador avaliaria uma imagem de 256x256 pixels.

Outro aspecto importante a ser considerado é que, processadores intermediários processam os dados referentes ao módulo de controle, paralelamente às tarefas de classificação das imagens, o que adequa o sistema aos requisitos de tempo real colocados como condição principal no desenvolvimento do sistema.

Os tempos de transmissão entre processadores, como avaliados por OSHIRO (1998) são bastante apropriados à aplicação descrita.

A figura 5.67, a seguir, mostra esquematicamente a operação do sistema.

Como mostrado na figura, cada processador da última camada é treinado com todos os campos *coarse* gerados a partir da imagem original. Durante a fase de teste, cada processador irá reconhecer um campo *coarse*, sendo geradas pelo sistema, portanto, 8 saídas em paralelo, representando os 8 campos *coarse* gerados. Estas saídas são então comparadas e, utilizando um esquema *winner takes all*, a saída com maior incidência representa a imagem de entrada.

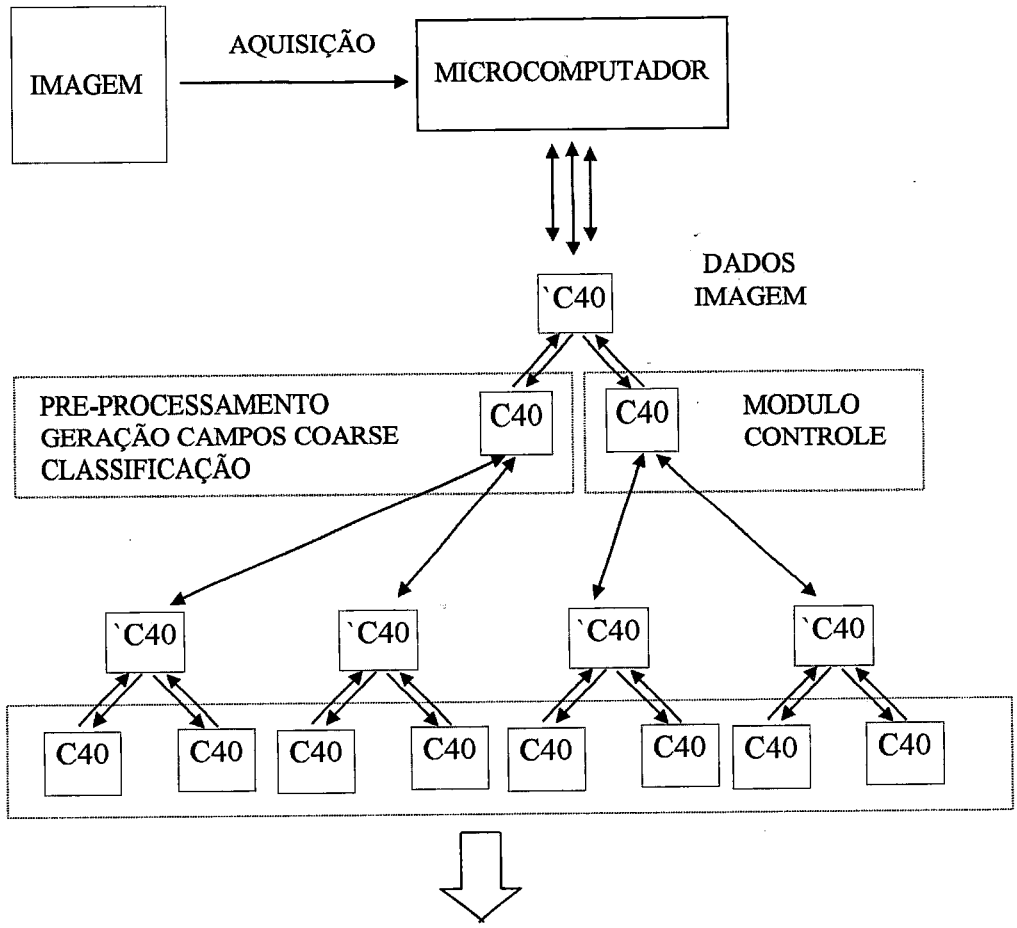


Figura 5.67 - Operação do sistema.

Como mostrado na figura anterior, cada elemento processador (C40) da base da arquitetura paralela em árvore mostrada possui um módulo classificador como o descrito anteriormente, o qual é treinado com todos os campos *coarse* de todas as imagens manipuladas pelo sistema. Assim, cada processador gera uma saída independente para a imagem apresentada ao sistema, ou seja, todos os elementos processadores da base (8 elementos C40) geram saídas (imagem reconhecida) para os campos *coarse* gerados para a imagem de entrada (8 campos os quais são processados por 8 processadores). Esta forma de processamento permite a avaliação utilizando o esquema de *winners take all*, para as saídas dos elementos processadores (8 saídas).

A saída com maior incidência representa a imagem reconhecida.

5.6 CONSIDERAÇÕES FINAIS

Neste capítulo foi apresentado o sistema de reconhecimento de imagens proposto, através de seus módulos e sub-módulos, além da estrutura geral de funcionamento. Os principais resultados experimentais obtidos foram mostrados, relativos a cada bloco componente do sistema. O desempenho de classificação das imagens foi então apresentado e analisado.

No capítulo 6, a seguir, as principais conclusões obtidas do presente trabalho serão apresentadas, além de sugestões de trabalhos futuros

6. CONCLUSÕES

6.1 CONSIDERAÇÕES INICIAIS

No capítulo anterior apresentou-se o projeto de sistema de reconhecimento de peças mecânicas, para robô de montagem, baseado em redes neurais artificiais, com concepção de projeto voltada para a implementação paralela, especificamente para a Arquitetura CPAD. Foram apresentados os diversos componentes do sistema, assim como os principais resultados experimentais obtidos com o sistema proposto.

Foi dada ênfase especial à apresentação das imagens resultantes de cada um dos blocos componentes do estágio de Pré-Processamento, assim como à operação do bloco gerador de campos *coarse*. Outro aspecto evidenciado foi a taxa de reconhecimento obtida com diversos exemplos apresentados ao sistema, exemplos estes compostos de imagens transformadas geometricamente e corrompidas com ruído. Evidenciou-se portanto, o aspecto de invariância geométrica incorporado ao sistema. Além disso, foram avaliados os tempos de processamento envolvidos.

6.2 CONCLUSÕES

O objetivo deste trabalho foi implementar um sistema de reconhecimentos de objetos, invariante a transformações geométricas, o qual pudesse ser utilizado em aplicações de montagem automatizada. Foi mostrado, com os resultados obtidos, que o sistema pode trabalhar com imagens distorcidas ou ruidosas. Tais imagens podem estar transladadas, escalonadas ou rotacionadas em relação as imagens originalmente ensinadas ao sistema. Diversos aspectos relativos ao desenvolvimento teórico e experimental foram abordados.

Os tempos de processamento envolvidos demonstraram a viabilidade de utilização do sistema em aplicações de tempo real, baseados na utilização da Arquitetura CPAD.

Através das taxas de reconhecimento obtidas para os campos *coarse* gerados (algumas superiores a 99%), pode-se garantir o reconhecimento de 100% das imagens apresentadas ao sistema, já que a classificação final é baseada no esquema *winner takes all* sobre os oito campos *coarse* testados.

Em função dos objetivos inicialmente traçados para a implementação do sistema e dos resultados experimentais obtidos, são citadas, a seguir, as principais conclusões obtidas do presente trabalho.

O sistema apresenta a característica de invariância geométrica ao tratar com as imagens de entrada, ou seja, tolera imagens de peças transladadas, escaladas e rotacionadas em relação à posição original dos exemplos utilizados para treinamento. Isto possibilita a operação em meios não controlados, ou seja, as peças a serem manipuladas podem apresentar-se ao sistema de forma diferente daquela originalmente aprendida. Para tarefas de manipulação em tempo real, esta característica representa um grande passo em relação à automação total da montagem.

Grandes imagens de entrada podem ser utilizadas (128x128, 256x256, 512x512, 1024x1024, 2048x2048, etc.) sendo que a limitação está apenas na sensibilidade da filtragem inicial, a qual toma um maior tempo de processamento para imagens mais complexas. O esquema de codificação *coarse*, implementado no sistema, mostrou-se bastante eficaz em sua tarefa de compressão de dados, ou seja, através desta técnica, os tamanhos de imagens citados anteriormente passam a ser factíveis em sistemas de tempo real, já que estes são os tamanhos normalmente requeridos em tarefas práticas envolvendo manipulação precisa de diversas famílias de peças.

O sistema pode trabalhar com diversas classes de peças, de diversos tamanhos e formas, além de poder trabalhar com imprecisões de posicionamento das mesmas.

Através do módulo de controle, o sistema gera os parâmetros principais para o manipulador. Isto permite uma total integração do sistema, a nível de *hardware* embarcado em manipuladores ou robôs de montagem.

O sistema apresenta tempos de processamento apropriados aos requisitos de tempo real impostos pela aplicação em questão;

O custo de implementação de um sistema prático, baseado nesta arquitetura, é relativamente baixo, limitado apenas pela velocidade de processamento requerida.

Os objetivos definidos inicialmente foram alcançados, o que justifica algumas propostas de desenvolvimento futuro, descritas a seguir.

6.3 PROPOSTAS PARA TRABALHOS FUTUROS

Através do sistema proposto e implementado neste trabalho e, utilizando a plataforma de hardware desenvolvida por OSHIRO (1998), pode-se projetar um sistema de alto desempenho para ser utilizado em mecanismos de manipulação. Os principais trabalhos nesta área deverão envolver:

- Construção completa do CPAD, com o número apropriado de processadores para a implementação do sistema em questão;
- Otimização das rotinas de *software*, as quais foram desenvolvidas apenas para teste e não para operação em tempo real;
- Implementação de novos módulos de *software*, os quais poderão contornar problemas de tempo de processamento, como aqueles apresentados pelo filtro utilizado no módulo de aquisição de dados;
- Implementação de um protótipo industrial, realizando testes de operação em ambientes industriais;
- Migração da estratégia de operação 2D para a operação 3D, possibilitando, futuramente, a inclusão de movimento nas peças a serem manipuladas.

REFERENCIAS BIBLIOGRÁFICAS

AGNETIS, A.; DROR, M.; VAKHARIA, A. J.; ROSSI, F. (1996). Tool handling and scheduling in a two-machine flexible manufacturing cell. *IIE Transactions*, vol 28, n. 5.

AGOSTINHO, O. L. (1989). *Sistemas Flexíveis de Manufatura*, Apostila - EESC, USP.

AKELLA, S.; MASON, M. T. (1998). Parts orienting with partial sensor information. *Proceedings-IEEE-International-Conference-on-Robotics-and-Automation*. vol 1, IEEE, Piscataway, NJ, USA, p 557-564.

ALEKSANDER, I.; THOMAS, W. V.; BOWDEN, P. A. (1984). Wisard: a radical step forward in image recognition. *Sensor Review*, 4(3):120-124.

ALEKSANDER, I. (1967). Adaptive systems of logic networks and binary memories. *Proceedings of the Spring Joint Computer Conference*.

ALEKSANDER, I.; MAMDANI, E. H. (1968). Microcircuit learning nets: Improved recognition by means of pattern feedback. *Electronics Letters*, 4(20):425-426.

ALEKSANDER, I. (1989a). *The logic of Connectionist Systems*. North Oxford Academic, 1989.

ALEKSANDER, I. (1989b). *Ideal neurons for neural computers*. Elsevier, 1989.

ALEXIEF, J. L.; KERKENI, N.; ANGUE, J. C. (1996) General method for inspection based on artificial vision of plan products with random or organized texture.

AMIRAT, Y.; PONTNAU, J.; BABACI, S.; FRANCOIS, C. (1995). Flexible assembly cell integrating a parallel manipulator for accurate automatic assembly tasks. *IEEE/IAS Conference on Industrial Automation and Control.* Proceedings, USA.

ANAND, R.; MEHPOTRA, K.; MOHAN, C. K.; RANKA, S. (1993). Analysing images containing multiple sparse patterns with Neural networks. *Pattern Recognition*, Vol 26, n.11.

ARAI, T.; OSUMI, H.; FUKUOKA, T.; MORIYAMA, K. A (1995). Cooperative Assembly System using two Manipulators with precise positioning devices. *Annals of the CIRP*, Vol 44, January.

ARMSTRONG, W. W.; GECSEI, J. (1979). Adaptation algorithms for binary tree networks. *IEEE Transactions on Systems, Man and Cybernetics*, vol 9.

AUSTIN, J.; STONHAM, T. J. (1987). An associative memory for use in image recognition and occlusion analysis. *Image and vision computing*, vol 5, 1987, p 251-261.

BERGERON, A.; ARSENAULT, H. H.; GINGRAS, D. (1995). Single-rail translation-invariant optical associative memory. *Applied Optics*, vol 34, n. 2.

BLEDSOE, W. W.; BROWNING, I. (1966). *Pattern recognition and reading by machine*. L Uhr, editor, Pattern Recognition. Wiley.

BORGES, D. L.; ORR, M. J.; FISHER, R. B. (1994). A Radial Basis Function Neural Network for Parts Identification of three dimensional shapes. *VII SIBGRAPI*, Anais.

BOTELHO, S. S. C.; DO VALLE, S. E. ; UEBEL, L. F.; BARONE, D. (1996). High speed neural control for robot navigation. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol 3, 1996, IEEE, Piscataway, NJ, USA, p 1956-1959.

BOWMAKER, R. G.; COGHILL, G. G. (1992). Improved recognition capabilities for goal seeking neuron. *Electronic Letters*, n. 28.

BOHN, R.; JAIKUMAR, R. (1988). *The Dynamic Approach: an Alternative Paradigm for Operations Management*. Harvard Business Scholl.

BRADY, M. L.; RAGHAVAN, R.; SLAWNY, J. (1989). Probabilistic cellular automata in pattern recognition. *International Joint Conference on Neural Networks*, vol I, pages 177-182, Washington, D.C., June, IEEE.

BRAUNL, T. (1997). Improv and EyeBot real-time vision on-board mobile robots. *Proceedings of the Annual Conference on Mechatronics and Machine Vision in Practice -MViP*, 1997, IEEE, Los Alamitos, CA, USA. p 131-135.

BUHMANN, J.; SCHULTEN, K. (1988). Invariant pattern recognition by means of fast synaptic plasticity. *IEEE Conference on Neural Networks*, Proceedings, San Diego.

BUHMANN, J.; LADES, M.; VON DER MALSBURG, C. (1990). Size and distortion invariant object recognition by hierarchical graph matching. *International Joint Conference on Neural Networks*, Proceedings.

BURKHARDT, B.; MULLER, X. (1990). On invariant sets fo a certain class of fast translation-invariant-transforms. *IEEE Transactions on Acoustic Speech Signal Processing*, vol 28.

BUZZACOTT, J. A.; SHANTHIKUMAR, J. G. (1980). Models for understanding Flexible Manufacturing Systems. *AIIE Transactions*, December .

CAMARINHA-MATOS, L. M.; LOPES, L. S.; BARATA, J. (1996). Integration in supervision of Flexible Assembly Systems. *IEEE Tansactions on Robotics and Automation*, vol 12, n. 2.

CARVALHO, A. C. P. L. F. (1994). *Towards an integrated boolean neural network for image recognition*. PhD thesis. University of Kent at Canterbury.

CARVALHO, A. C. P. L. F. (1995). A modular neural architecture for pattern recognition. *II Simpósio Brasileiro de Redes* Neurais, Anais.

CARVALHO, A. C. P. L. F.; BISSET, D.; FAIRHURST, M. (1997). Training algorithms for GSN super f neural networks. *Proceedings of the Workshop on Cybernetic Vision, 1997, IEEE Comp Soc*, Los Alamitos, CA, USA, p 74-79.

CHAO, T. H. (1992). Automatic target recognition using a feature-based optical neural network. *Optical Pattern Recognition III*, Proceedings.

CHAO, T. H.; STONER, W. W. (1993). Optical implementation of a feature-based neural network with application to automatic target recognition. *Applied Optics*, vol 32, n. 8.

CHARLTON, P. C. (1993). Investigation into the suitability of an neural network classifier for use in automated tube inspection system. *British Journal of Non Destructive Testing*. Vol 35, n. 8.

CHETTY, O.; V. KRISHNAIAH; GNANASEKARAN, O. C. (1996). Modelling, simulation and scheduling of flexible assembly systems with coloured Petri Nets. *International Journal of Advanced Manufacturing Technology*, vol 11, n. 6.

CHIN, R. T.; DYER, C. R. (1986). Model-based recognition in robot vision. *Computing Surveys*, vol 18, n. 1, 1986.

CHTIOUI, Y.; BERTRAND, D.; DEVAUX, M. F.; BARBA, D. (1996). Application of a hybrid neural network for discrimination of seeds by artificial vision. *Proceedings of the International Conferecne on Tools with Artificial Intelligence*.

COLLET, S.; SPICER, R. J. (1995). Improving productivity through cellular manufacturing. *Production and Inventory Management Journal*, vol 36, n. 1.

COLOMBO, C.; RUCCI, M.; DARIO, P. (1994). Attentive behavior in an anthropomorfic robot vision system, *Robotics and Automation Systems*, vol 12.

COSTA, J. A. F. (1996). *Sistema de Reconhecimento de Padrões Visuais Invariante a Transformações Geométricas Utilzando Redes Neurais Artificiais de Múltiplas Camadas*. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo.

COOPERSTOCK, J. R.; MILIOS, E. E. (1993). Self-supervised learning for docking and target reaching. *Robotics and Autonomous Systems*, vol 11.

COULOT, C.; KOHLER-HEMMERLIN, S.; DUMONT, C.; LAMALLE, B. (1996). Dimensional control of metallic objects by artificial vision, *Industrial Electronics Conference* , vol 2.

CRUZ, V.; CRISTOBAL, G.; MICHAUX, T.; BARQUIN, S. (1989). Invariant image recognition using a multi-network neural model. *International Joint Conference on Neural Networks*, Proceedings.

DAVIES, J. L.; GILL, K. F. (1993). Machine Vision and Automated Assembly. *Mechatronics*, Vol 3, n.4.

DAVIS, L. (1991). *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, 1991.

DOYLE, J. R. (1990). Supervised learning in n-tuple neural networks. *International Journal Man-Machine Studies*, n. 33, pag 21-40.

DROR, I. E.; ZAGAESKI, M.; MOSS, C. F. (1995) Three-Dimensional Target Recognition via Sonar: A Neural Network Model. *Neural Networks*, vol 8, n. 1.

EGBELU, P. J.; TANCHOCO, J. M. A. (1984). Characterization of Automatic Guided Vehicle Dispatching Rules. *International Journal of Production Research*, vol 22.

EL-MARAGHY, H. A.; SYED, A.; CHU, H. (1993). A 2-D mapping approach for monitoring and diagnosing the execution of robotic assembly tasks. *Annals of the CIRP*. vol 42, January.

FAIRHURST, M. C.; FILHO, E.; BISSET, D. L. (1991a). Adaptive pattern recognition using goal-seeking neurons. *Pattern Recognition Letters*, n. 12, pag 131-138, March, 1989.

FAIRHURST, M. C.; FILHO, E.; BISSET, D. L. (1990). Networks of Boolean goal seeking nerons for pattern recognition. *International Symposium on Neural Networks for Sensory and Motor Systems*, Neuss, F. R. G., March.

FAIRHURST, M. C.; FILHO, E.; BISSET, D. L. (1990a). A goal seeking Boolean neural network. *Neural Computing Meeting 90*, London, UK, April 1990. British Neural Network Society.

FAIRHURST, M. C.; CARVALHO, A. de; COWLEY, K. D.; BISSET, D. L. (1991). A parallel approach to the real-time classification of handprinted characters. *Proceedings of the IWFHR-2 International Workshop*, pag 93-100, Toulouse, France, September.

FAIRHURST, M. C.; FILHO, E.; BISSET, D. L. (1992). Analysis of saturation problem in ram-based neural networks. *Electronic Letters*, vol 28, n. 4, pag 345-347, February.

FAIRHURST, M.C. (1988). *Computer vision for robotic systems.* Prentice Hall.

FANG, W. C.; SHEU, B. J.; CHEN, O. T. C.; CHOI, J. (1992). A VLSI Neural Processor for Image Data Compression using Self-Organization Networks. *IEEE Transactions on Neural Networks*, vol 3, n. 3.

FELDMANN, K.; GERHARD, M. (1995). Direct Soldering of Electronic Components on Molded Devices. *Annals of the CIRP*, vol 44, January.

FELDMANN, K.; COLOMBO, A. W. (1998). Material flow and control sequence specification of flexible production systems using coloured Petri nets. *International Journal of Advanced Manufacturing Technology*, vol 14, n. 10, 1998, p 760-774.

FERNANDES, C. J. G. (1985). Stability properties inherent to digital neural networks. *Proceedings of Cognitiva 85*, 1:8.

FILHO, E. (1990). *Investigation of Boolean Neural Networks based on a Novel Goal-Seeking Neuron.* PhD thesis. University of Kent, 1990.

FORESTI, G. L.; PIERONI, G. G. (1997). 3D Object recognition by neural trees. *IEEE International Conference on Image Processing*, vol 3, 1997, IEEE Comp Soc, Los Alamitos, CA, USA, p 408-411.

FUKUDA, T.; KURIHARA, T.; SHIBATA, T.; TOKITA, M.; MITSUOKA, T. (1991). Application of Neural Network-Based Servo Controller to position, force and stabbing control by robotic manipulator. *JSME International Journal*, vol 34, n. 2.

FUKUMI, M.; OMATSU, S.; TAKEDA, F.; KOSAKA, T. (1992). Rotation-invariant neural pattern recognition system with application to coin recognition, *IEEE Transactions on Neural Networks*, vol 3, n. 2.

FUKUSHIMA, K. (1975). Cognitron: a self-organizing multilayered neural network. *Biological Cybernetics*, vol 20, 1975, p 121-136.

FUKUSHIMA, K. (1980). Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, *Biological cybernetics*, vol 36, n. 4.

FUKUSHIMA, K. (1983). Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEE Transactions on Systems, Man and Cybernetics*, SM-13(5), p. 826-834.

FUKUSHIMA, K. (1989). Analysis of the process of visual pattern recognition by the neocognitron. *Neural Networks*, vol 2, n. 6.

FUKUSHIMA, K. (1985). Cognitron: a self-organizing multilayered neural network. *Biological Cybernetics*, 20, p. 121-136.

FUKUSHIMA, K. (1992). Character recognition with neural networks. *Neurocomputing*, vol 4, n. 5, 1992, p 221-233.

FUNABIKI, N.; KITAMICHI, J.; NISHIKAWA, S. (1998). Evolutionary Neural Network approach for module orientation problems. *IEEE Transactions on Systems, Man and Cybernetics*, Part-B: Cybernetics, vol 28, n. 6, p 849-855.

GARRET, T. (1989). CIM Cost Justification and Analysis. *Control*, p. 26 - 34, june.

GARZOTTO, A. (1992). The NAND-Net. *Proceedings of the ICANN 92 Conference*, p. 1419-1422, Brighton, UK, September, Elsevier.

GILES, G. L.; MAXWELL, T. (1987). Learning invariances and generalization in higher-order neural networks. *Applied Optics*, vol 26.

GILES, G. L.; GRIFFIN, R. D.; MAXWELL, T. (1988). Encoding geometric invariance in higher-order neural networks. *Neural Information Processing Systems*.

GONZALES, R. C.; WINTZ, P. (1987). *Digital Image Processing*. Addison-Wesley Publishing Company.

GORSE, D.; TAYLOR, J. G. (1988). On the equivalence and properties of noisy neural and probabilistic ram nets. *Physics Letters A*, 131, p. 326-332.

.GREWAL, S.; TRAN, P.; BHASKARE, A. (1993). Assembly Planning Software. *Annals of the CIRP*, Vol 44, January.

GROSSBERG, S. (1976a). Adaptive pattern classification and universal recoding: I. parallel development and coding of neural feature detectors. *Biological Cybernetics*, 23, p. 121-134.

GROSSBERG, S. (1976b). Adaptive pattern classification and universal recoding: II. Feedback, expectation, olfaction, illusions. *Biological Cybernetics*, 23, p. 187.

GROOVER, M. P. (1987). *Automation, Production Systems and Computer-Integrated Manufacturing*. Prentice-Hall.

GRUVER, W. A. (1994). Intelligent Robotics in Manufacturing, Service and Rehabilitation: An Overview. *IEEE Transactions on Industrial Electronics*, vol 41, n. 1, February.

GUPTA, L.; SAYEH, M. R.; TAMMANA, R. (1989). A neural network approach to robust shape classification. *Pattern Recognition*, vol 23, n. 6.

HASHIMOTO, H.; KUBOTA, T.; SATO, M.; HARASHIMA, F. (1992). Visual control of robotic manipulator based on neural networks. *IEEE Transactions on Industrial Electronics*, vol 3, n. 6.

HEBB, D. O. (1949). *The organization of Behavior*. Wiley.

HEMMINGER, T. L.; RAEZ, C. A. (1994). Using a Hopfield network for rotation and sacle independent pattern recognition. *IEEE International Conference on Neural Networks*, Proceedings, USA.

HILAI, R.; RUBINSTEIN, J. (1994). Recognition of rotated images by invariant Karhunen-Loeve expansion. *Journal Optic Society*, vol 11, n.5.

HOPFIELD, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, USA, p. 2554-2558, Aprill.

HOSOKAWA, M.; OMATU, S.'FUKUMI, M. (1989). New apprach for pattern recognition by neural networks with scramblers. *International Joint Conference on Neural Networks*, Proceedings, Washington.

HOU, T. H.; LIN, L.; SCOTT, P. D. (1993). A neural network-based automated inspection system with an application to surface mount devices. *International Journal of Production Research*, vol 31, n. 5.

HOVLAND, G. E.; MCCARRAGHER, B. J. (1997). Combining force and position measurements for the monitoring of robotic assembly. *IEEE International Conference on Intelligent Robots and Systems*, vol 2, 1997, IEEE, Piscataway, NJ, USA, p 654-660.

HWANG, H.; PARK, B.; NGUYEN, M.; CHEN, Y. R. (1996). Hybrid image processing for robust extraction of lean tissue on beef cut surface. *Proceedings of SPIE - The International Society for Optical Engineering*, vol 2665.

HWANG, J. N.; TSENG, Y. H. (1993). 3D motion estimation using simple perspective sparse range data via surface reconstruction neural networks. *International Conference on Neural Networks*, Proceedings.

INIGO, R. M.; HIMES, G. S.; NARATHONG, C. (1993). A Neural Network Visual-Tracking System. *Computers Electronics Engineering*, vol 19, n. 4.

JAIKUMAR, R.; SHIRLEY, G. (1987). *Technological Niches and Competitive Development*. Harvard Business Scholl, May.

JAIN, A. K. (1989). *Fundamentals of Digital Image Processing*, Prentice Hall.

JONSSON, M.; WIBERG, P. A.; WICKSTRÖM, N. (1998) *Robot Neuro - Vision based navigation experiments using a State Machine and Neural Networks in close Cooperation*. Halmstad University, Halmstad, Sweden, 1998.

KAKIZAKI, S.; HORAN, P.; ARIMOTO, A.; SAKO, H.; SAITO, A.; KUGIYA, F. (1994). Optical implementation of a translation-invariant second-order neural network for multiple-pattern classification. *Applied Optics*, vol 33, n. 35.

KAN, W. K.; ALEKSANDER, I. (1987). A probabilistic logic neuron network for associative learning. *Proceedings of the IEEE First International Conference on Neural Networks*, volume II, p. 541-548, San Diego, California, June, IEEE.

KANERVA, P. (1988). *Sparse Distributed Memory*. MIT Press.

KARATHANASSI, V.; IOSSIFIDIS, C.; ROKOS, D. (1996). Application of machine vision techniques in the quality control of pharmaceutical solutions. *Computers in Industry*, vol 32, n. 2.

KARPUSHCHENKO, V. N.; LIVINTSEV, V. V. (1985). Integrated Mechanization and Automation of Warehousing and stockpiling Facilities. *Soviet Energy Technology*, vol. 5.

KAUFFMAN, S. A. (1969). Metabolic stability and epigenesis in randomly connected genetic nets, *Journal of Theoretical Biology*, 22, p. 37-467.

KELLER, J. M.; GADER, P.; TAHANI, H.; CHIANG, J. H.; MAHAMED, M. (1994). Advances in fuzzy integration for pattern recognition. *Fuzzy Sets and Systems*, vol 65, n 2-3.

KIM, H. J.; YANG, H. S. (1994). A Neural Network capable of learning and inference for visual pattern recognition. *Pattern Recognition*, vol 27, n. 10.

KIM, J.; CHO, H.; KIM, S. (1997). Visual sensing system with multi-views for flexible parts assembly. *International Conference on Advanced Robotics - Proceedings - ICAR. 1997*, IEEE, Piscataway, NJ, USA, p 979-984.

KOH, J.; SUK, M.; BHANDARKAR, S. M. (1995). A Multilayer Self-Organizing Feature Map for Range Image Segmentation. *Neural Networks*, vol 8, n. 1.

KOH, I.; CHOI, J. H. (1997). Initial requirements to the optimal performance of systems modeled by Timed Place Petri Nets. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol 4, 1997, IEEE, Piscataway, NJ, USA,. p 3233-3238.

KOHONEN, T. (1989). *Self-Organization and Associative Memory*. Springer Verlag.

KOHONEN, T. (1988). The neural phonetic typewriter. *Computer*, 21(3), p. 11-22.

KORCYL, A.; LEBKOWSKI, P.; SAWIK, T. (1995). Selection of assembly sequences and balancing workloads in a flexible assembly line. *IEEE Symposium on Emerging Technologies & Factory Automation,* vol 3.

KOUVELIS, P.; CHIANG, W. C. (1996). Optimal and heuristic procedures for row layout problems in automated manufacturing systems. *Journal of the Operacional Research Society*, vol 47, n. 6.

KOWALCZYK, A.; FERRA, H. L. (1994). Developing Higher-Order Networks with empirically selected units. *IEEE Transactions on Neural Networks*, vol 5, n. 5.

KUNCHEV, R.; TOPALOVA, I. (1998). Invariant recognition system for 2D objects based on three-stage BPNN structure. *Electronics Letters*, vol 34, n. 10, May 14, 1998, p 974-976.

KUNG, S. Y.; TAUR, J. S. (1995). Decision-Based Neural Networks with Signal/Image Classification Applications. *IEEE Transactions on Neural Networks*, vol 6, n. 1.

KUPRIYANOV, K. V.; KORENEV, A. I.; KOZLOV, G. V.; LASHCHENKO, I. D. (1985). In-Plant Storage and Stacking Systems. *Soviet Energy Technology*, vol 8.

LADES, M.; VORBRUGGEN, J. C.; BUHMANN, J.; LANGE, J.; MALSBURG, R. P.; KONEN, W. (1993). Distortion invariant object recognition in the dynamic link architecture. *IEEE Transactions on Computers*, vol 42, n. 3.

LEJEUNE, C.; SHENG, Y. (1993). Optoneural system for invariant pattern recognition, *Journal of Physics*, vol 71, n. 9-10.

LI, Z.; ZHANG, D. (1991). An Intelligent Vision System for Robot. *Applications of Artificial Intelligence on Engineering Problems - 1st International Conference*, Southampton University, UK, vol 1, April.

LIN, C.; YEH, J. M.; DING, J. R. (1998). Design of random inspection rate for a flexible assembly system: A heuristic genetic algorithm approach. *Microelectronics and Reliability*, vol 38, n. 4, Apr 1998, p 545-551.

LING, A. C.; CHANG, T. C. (1993). An integrated approach to automated assembly planning for three -dimensional mechanical products. *International Journal Production Research*, vol 31, n. 5, p. 1201-1227.

LIPPMANN, R. P. (1987). An introduction to computing with neural nets. *IEEE ASSP Magazine*, 4-22, April.

LINSKER, R. (1986a). From basic network principles to neural architecture: Emergence of spatial-opponent cells. *Proceedings of the National Academy of Sciences*, USA, p. 7508-7512.

LINSKER, R. (1986b). From basic network principles to neural architecture: Emergence of orientation-selective cells. *Proceedings of the National Academy of Sciences*, USA, p. 8390-8394.

LINSKER, R (1986c). From basic network principles to neural architecture: Emergence of orientations columns. *Proceedings of the National Academy of Sciences*, USA, p. :8779-8783.

LU, S.; SZETO, A. (1993). Hierarchical artificial neural networks for edge enhancement. *Pattern recognition*, vol 26, n. 8.

McMICHAEL, D. W. (1994). Decision-theoretic approach to visual inspection using neural networks. *IEEE Proceedings on Visual Image Signal Processing*, vol 141, n. 4.

MAKINO, H. (1995). Estimation of Production Rate in Flexible Assembly Systems. *Anal of the CIRP*, vol 44, January.

MAREN, A. J.; HARSTON, C. T.; PAP, R. M. (1990). *Handbook of neural computing applications*, San DIego, Academic Press.

MARTINETZ, T.; SCHULTEN, K. (1993). A neural network for robot control cooperation between neural units as a requeriment for learning. *Computers Elect. Engng*, vol 19, n. 4.

MARTINS, W. (1994). *The Improvement of Classifiers based on Weightless Neural Networks*. PhD thesis. University of York.

MARTLAND, D. (1987a). Behaviour of autonomous (synchronous) Boolean networks, *Proceedings of the first IEEE Conference on Neural Networks*, San Diego II, p. 243-250.

MARTLAND, D. (1987b). Auto-associative pattern storage using synchronous Boolean networks, *Proceedings of the first IEEE Conference on Neural Networks*, San Diego II, p. 355-366.

MARUNO, S.; IMAGAWA, T.; KOHDA, T.; SHIMEKI, Y. (1993). Object recognition system using temporal pattern recognition with quantizer neuron chip. *International Joint Conference on Neural Networks*, Proceedings.

MASTERS, T. (1995). *Advanced ALgorithms for Neural Networks*, John Wileey.

MATHER, T.W.; GARNER, H. E. (1989). Implementing CIM in a Existing Facility, *Hydrocarbon Processing*, p. 43 - 49, January.

McCULLOCH, W.S.; PITTS, W. H. (1943). A logical calculus of the ideas immanent in nervous activity. *Bull Math Biophys*, 5, p. 115-133.

MEREDITH, J. R.; HILL, M. M. (1987). Justifying New Manufacturing System: a Managerial Approach. *Sloan Management Review*, vol 28, n.4.

MEYER, J. D. (1988). Applications of robots. *International Encyclopedia of Robotics: Applications and Automation.* R. Dorf and S. Nof, Eds. New York: Wiley Interscience.

MICHALEWICZ, Z. (1992). *Genetic ALgorithms + Data Structures = Evolution Programs.* Springer-Verlag, 1992.

MINSKY, M.; PAPERT, S. (1969). *Perceptrons.* MIT Press, Cambridge.

MITZIAS, D. A.; MERTZIOS, B. G. (1994). Shape recognition with neural classifier based polygon approximation tecnique. *Pattern Recognition*, vol 27, n. 3.

MOH, J.; SHIH, F. (1995). A general purpose model for image operations based on multilayer perceptrons. *Pattern Recognition*, vol 28, n. 7.

MÖSCHEN, J.; HOSTICKA, B. J. (1993). Programmable Hardware-Architecture for real-time pattern-recognition systems. *4th Internatinal Conference on Image Processing and its Applications.*

MOTAVALLI, S.; BAHR, B. (1993). Automated tool monitoring system using vision system for a robotic cell. *Robotics and Computer-Integrated Manufacturing,* vol 10, n. 4.

MUNDKUR, P. Y.; DESAI, U. B. (1991). Automatic target recognition using image and network decomposition. *International Joint Conference on Neural Networks.* Proceedings.

MYERS, C.; ALEKSANDER, I. (1988). Learning algorithm for plrobabilistic neural nets. *Proceedings of the First INSS Annual Meeting*, p. 205-209.

NAYAR, S. K.; NENE, S. A.; MURASE, H. (1996). Subspace methods for robotic vision. *IEEE Transactins on Robotic Automation*, vol 12, n. 5.

NAJJARI, H.; STEINER, S. S. (1995). Knowledge-based object recognition technique for robots assembly application. *Proceedings of the SPIE*, vol 2588, USA.

NEATHWAY, P.; CANTUA, F. (1998). Pick-and-place machine analysis. *SMT-Surface-Mount-Technology-Magazine.* vol 12, n. 10, p 54-58.

NEWMAN, T. S.; JAIN, A. K. (1995). A survey of Automated Visual Inspection. *Computer vision and Image Understanding*, vol 61, n. 2.

NGUYEN, W.; MILLS, J. K. (1996). Multi-robot control for flexible fixtureless assembly of flexible sheet metal auto body parts. *IEEE International Conference on RObotics and Automation*, Proceedings, vol 3.

NOF, S. Y.; WHISTON, A. B.; BULLERS, W. I. (1980). Control and Decision Support in Automatic Manufacturing Systems. *AIIE Transactions*, vol 12, n.2.

NOF, S. Y.; RAJAN, V. N. (1993). Automatic Generation of Assembly Constrainsts and Cooperation Task Planning. *Annals of the CIRP*, vol 42, January.

ODED, B. ; ODED, M. (1986). Cooperation Among Flexible Manufacturing Systems. *IEEE Journal of Robotics and Automation*, vol RA-2, n.1, March.

OLSSON, L. J.; GRUBER, S. (1993). Web Process Inspection using Neural Classification of Scattering Light. *IEEE Transactions on Industrial Electronics*, vol 40, n. 2.

OSHIRO, O. T. (1998). Uma arquitetura paralela para o controle de máquina-ferramenta de ultraprecisão. Tese de Doutorado. EESC - USP.

PAL, N. R.; PAL, P.; BASU, A. K. (1993). A new shape representation scheme and its application to shape discrimination using a neural network. *Pattern Recognition*, vol 26, n. 4.

PAPANIKOLOPOULOS, N. P.; KHOSLA, P. K.; KANADE, T. (1993). Visual tracking of a moving target by a Câmera mounted on a robot: a combination of control and vision. *IEEE Transactions on Robotics and Automation*, vol 9, n. 1.

PARKER, J. R. (1994). *Practical Computer Vision Using C*, Wiley.

PATARNELLO, S.; CARNEVALI, P. (1987). Learning networks of neurons with Boolean logic. Europhysics Letters, 4(4), p. 503-508.

PAUL, B.; KONAR, A.; MANDAL, A. K. (1998). Image recognition with fuzzy ADALINE neurons. *IEEE World Congress on Computational Intelligence -IEEE International Conference on Neural Networks Conference* - Proceedings, vol 1, 1998, IEEE, Piscataway, NJ, USA, p 747-752.

PAULI, J. (1998). Learning to recognize and grasp objects. *Machine Learning*, vol 31, n. 1-3, Apr-Jun 1998, p 239-258.

PHAM, D. T.; BAYRO-CORROCHANO, E. J. (1994). Self-Organizing neural-network-based pattern clustering method with fuzzy outputs. *Pattern Recognition*, vol 27, n. 8.

PHAM, D. T.; BAYRO-CORROCHANO, E. J. (1994). Neural classifiers for automated visual inspection. *Institution of Mechanical Engineers, Journal of Automated Engineering*, Proceedings, Part D.

PIERSON, R. A. (1984). Adapting horizontal material handling systems to Flexible Manufacturing Systems. *Industrial Engineering*, 16, n. 3.

PLATERO, C.; FERNANDEZ, C.; CAMPOY CEVERA, P.; ARACIL R. (1996). Surface analysis of cast aluminum by means of artificial vision and AI-based techniques. *Proceedings of the SPIE*, USA.

PLESCHBERGER, T. E.; HITOMI, K. (1993). Flexible final-assembly sequencing method for a JIT manufacturing environment. *International Journal of Production Research*, vol 31, n. 5, p. 1189-1199.

POURBOGHRAT, F. (1993). Adaptive neural controller design ofr robots. *Computers Electronic Engineering*, vol 19, n. 4.

PRICE, D.; KNERR, S.; DREYFUS, G. (1994). Pairwise Neural Network classifiers with probabilistic outputs. *Neural Information Processing Systems.*

PROKOP, R. J.; REEVES, A. P. (1992). A survey of moment-based techniques for unoccluded object representation and recognition. *Graphical Models and Image Processing*, vol 54, n. 5.

RABELO, L. C.; AVULA, X. J. R. (1991). Intelligent control of robotic arm using hierarchical neural network systems. *IEEE.*

RAJAPAKSE, J.; ACHARYA, R. (1990). Hierarchical neural architecture for visual pattern recognition and reconstructioin. *Proceedings of SPIE*, v. 1246.

RAMPERSAD, H. K. (1993). The DFA house. *Assembly Automation*, vol 13, n. 4, December.

RAMPERSAD, H. K. (1994a). *Integrated and Simultaneous Design for Robotic Assembly.* John Wiley, Chichester, November.

RAMPERSAD, H. K. (1994b). A concentric design process. *Advanced Summer Institute in Co-operative Intelligent Manufacturing Systems, Proceedings of the ASI 94*, Patras, Geece, June.

RAMPERSAD, H. K. (1994c). Integral and simultaneous design of robotic assembly systems. *Third International Conference on Automation, Robotics and Computer Vision*, Singapore, November.

RAMPERSAD, H.K. (1995). State of the art in robotic Assembly. *Industrial Robots*, vol 22, n. 2.

RAVEENDRAN, P.; JEGANNATHANS, S.; OMATU, S. (1993). New regular moment invariants to classify elongated and contracted images. *International Journal Conference on Neural Networks*, vol 3, 1993, p 2089-2092.

RAVINDRAN, A.; FOOTE, B.L.; BADIRU, A.; LEEMIS, L.; WILLIANS, L. (1988). Mechanized Material Handling Systems Desing and Routing. *Computers Industrial Engineering*, vol. 14, n.3.

RAY, K. S.; MAJUMDER, D. D. (1994). Application of Hopfield neural networks and canonical perspectives to recognize and locate partially occluded 3-D objects. *Pattern Recognition Letters*, vol 15.

REID, M. B.; SPIRKOVSKA, L.; OCHOA, E. (1989). Simultaneous position, scale and rotation invariant pattern classification using third-order neural networks. *International Journal Neural Networks*, vol 1.

ROMERO, R.A.F. (1993). *Otimização de Sistemas através de Redes Neurais Artificais*. Campinas. Tese (Doutorado) - Faculdade de Engenharia Elétrica, Universidade Estadual de Campinas - UNICAMP.

ROMDHAMI, S. (1997). *Face Recognition using Principal Components Analysis*, University of Glasgow, 1997.

ROSANDICH, R. G. (1997). HAVNET: a new neural network architecture for pattern recognition. *Neural Networks*, vol 10, n. 1.

ROTH, N.; SCHNEIDER, B. (1993). Clean Room Industrial Robot for Handling and Assembly in Semiconductor Industry. *Annals of the CIRP*, vol 42, January.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. (1986). Learning internal representations by error propagation. *Parallel Distributed Processing*. Cambridge, MA, MIT Press, vol 1.

SACKETT, P. .J.; FAN, I. .S. (1989). The Control of a Flexible Manufacturing System by Short-Term Goal Identification. *International Journal of Advanced Manufacturing Technology*, vol 4.

SAITO, H.; KIMURA, M. (1996). Shape modeling of multiple objects from shading images using genetic algorithms. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol 4, 1996, IEEE, Piscataway, NJ, USA, p 2463-2468.

SAWAI, H. (1994). Axially Symmetric Neural Network Architecture for Rotation-Invariant Pattern Recognition. *IEEE International Conference on Neural Networks*, Proceedings, USA.

SCHIMIDT, W. A. C. (1993). Pattern recognition properties of various feature spaces for higher order neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol 15, n. 8.

SEPEDA FILHO, I. H.; STEMMER, M. R. (1994). Reconhecimento de peças utilizando redes neurais. *I Congresso Brasileiro de Redes* Neurais, Itajubá - MG, Anais.

SHANG, C.; BROWN, K. (1994). Principal features-based texture classification with neural networks. *Pattern Recognition*, vol 27, n. 5.

SHARMA, S. M.; VISWANADHAM, N. (1996). Simulated annealing approach to machine loading problem in flexible assembly systems. *IEEE International Conference on Robotics and Automation.* Proceedings, vol 4, USA.

SHARMA, R.; SRINIVASA, N. (1998). Framework for active vision-based robot control using neural networks. *Robotica*, vol 16, n. pt 3, May-Jun 1998, p 309-327.

SHIN, K. G.; CUI, X. (1991). Collision avoidance in a multiple-robot system using intelligent control and neural networks. *Proceedings of the 30th Conference on Decision and Control.* Brighton, England.

SHIRVAIKAR, M. V.; TRIVEDI, M. M. (1995). A Neural Network Filter to detect small targets in High Clutter Backgrounds. *IEEE Transactions on Neural Networks,* vol 6, n. 1.

SIANG, K. S.; MING, Y. T. (1997). Enhancing flexibility of vision-based robots using an artificial neural network approach. *Integrated Manufacturing Systems,* vol 8, n. 1, 1997, p 43-49.

SIM, H.; DAMPER, R. I. (1997). Two-dimensional object matching using Kohonen maps. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol 1, 1997, IEEE, Piscataway, NJ, USA, p 620-625.

SKONECZNY, S.; SZOLTAKOWSKI, J.; STAJNIAK, A. (1995). Face recognition by higher-order neural networks. *Proceedings of the SPIE*, vol 2564.

SMAGT, P. van der; GROEN, F.; KRÖSE, B.; (1993). *Robot hand-eye coordination using neural networks.* Technical Paper, University of Amsterdam, October.

SONG, K. T.; CHANG, J. M. (1996). Experimental study on robot visual tracking using a neural controller. *Industrial Electronics Conference*, vol 3.

SPIRKOVSKA, L.; REID, M. B. (1990). Connectivity strategies for higher-order neural networks applied to pattern recognition. *Internationl Joint Conference on Neural Networks*, Proceedings, San Diego, CA.

SPIRKOVSKA, L.; REID, M. B. (1992a). *Fast Learning and Invariant Object Recognition.* John Wiley & Sons, p. 153 - 168.

SPIRKOVSKA L.; REID, M. B. (1992b). Robust position, scale and rotation invariant object recognition using higher-order neural networks, *Pattern Recognition*, vol 25, n. 9.

SPIRKOVSKA, L.; REID, M. B. (1993). Coarse Coading Higher-Order Neural Networks for PSRI Object Recognition. *IEEE Transaction on Neural Networks*, vol 4, n. 2.

SPIRKOVSKA, L.; REID, M. B. (1994). Higher-order neural networks appled to 2D and 3D object recognition. *Machine Learning*, vol 15.

SRINIVASA, N.; JOUANEH, M. (1992). A neural network model for invariant pattern recognition. *IEEE Transactions on Signal Processing*, vol 40, n. 6.

SRINIVASAN, V.; BHATIA, P.; ONG, S. H. (1994). Edge detection using a neural network. *Pattern Recognition*, vol 27, n. 12.

STRAVOS, P.; LISBOA, P. J. G. (1992). Translation, rotation, and scale invariant pattern recognition by hygher-order neural networks and moment classifiers. *IEEE Transactions on Neural Networks*, vol 3, n. 2, 1992, p 241-251.

SONG, K. T.; CHANG, J. M. (1996). Experimental study on robot visual tracking using neural controller. *IECON*, Proceedings, vol 3.

SUGISAKA, M.; TESHNEHLAB, M. (1993). Fast pattern recognition by using moment invariants computation via artificial neural networks. *Control-Theory and Advanced Technology*, Vol 9, n. 4.

TAI, H. M.; WANG, J.; ASHENAYI, K. (1992). A neural network-based tracking control system. *IEEE Transactions on Industrial Electronics*, vol 39, n. 6, December.

TAKANO, M.; KANAOKA, T.; SKRZYPEK, J.; TOMITA, S. (1994). A note on a higher-order neural network for distortion invariant pattern recognition. *Pattern Recognition Letters*, vol 15.

TIANXU, Z.; NONG, S.; GUOYOU, W.; XIAOWEN, L. (1996). Effective method for identifying small objects on a complicated background. *Artificial Intelligence in Engineering*, vol 10, n. 4.

TIRAKIS, A.; DELOPOULOS, A.; KOLLIAS, S. (1994). Invariant Image Recognition using triple correlations and Neural Networks. *IEEE International Conference on Neural Networks*, Proceedings, USA.

TLALE, N. S.; MAYOR, R.; BRIGHT, G. (1998). Intelligent gripper using low cost industrial sensors. *IEEE-International-Symposium-on-Industrial-Electronics*, vol 2, IEEE, Piscataway, NJ, USA, p 415-419

TRUXEL, S. K.; ROGERS, S. K.; KABRISKY, M. (1988). The use of neural networks in PSRI recognition. *Proc. Joint International Conference Neural Networks*, San Diego, CA, July 24-27.

TSAI, D. M.; CHEN, J. J.; CHEN, J. F. (1998). Vision system for surface roughness assessment using neural networks. *International Journal of Advanced Manufacturing Technology*, vol 14, n. 6, 1998, p 412-422.

TSAO, C. K. E.; LIN, W. C.; CHEN, C. T. (1993). Constraint satisfaction neural networks for image recognition. *Pattern Recognition*, vol 26, n. 4.

TZAFESTAS, S. G.; STAMOU, G. B. (1997). Concerning automated assembly: Knowledge-based issues and a fuzzy system for assembly under uncertainty. *Computer-Integrated-Manufacturing-Systems*, vol 10, n. 3, p 183-192.

UANG, C. M.; YIN, S.; ANDRES, P.; REESER, W.; YU, F. T. S. (1994). Shift-invariant interpattern association neural network. *Applied Optics*, vol 33, n. 11.

VAILLANT, R.; MONROCQ, C.; LE CUN, Y. (1994). Original approach for the localisation of objects in images. *IEE Proceedings - Vis. Image Signal Process*, vol 141, n. 4.

VARELLA, L. E. S.; PASSOS, E. P. L.; SANTOS, M. A.; ARAÚJO, R. L. (1994). Uma solução back-propagation invariante a escala, rotação e translação prara o reconhecimento de caracteres. *I Congresso Brasileiro de Redes Neurais*, Itajubá - MG.

VASCONCELOS, G. C. (1995). Redes Neurais e Reconhecimento de padrões. *II Sipósio Brasileiro de Redes Neurais*, São Carlos.

VILLALOBOS, L.; MERAT, F. (1995). Learning Capability Assesment and Feature Space Optimization for Higher-Order Neural Networks. *IEEE Transactions on Neural Networks*, vol 6, n. 1.

VON DER MALSBURG, C. (1988). Object recognition in dynamical network architecture, *Neural Networks*, vol 1.

VON DER MALSBURG, C. (1973). An automation with brain-like properties. *Kybernetcs*, 14, p. 85-100.

WAARD, W. P. (1994). Neural techniques and postal code detection. *Pattern Recognition Letters*, vol 15.

WALKER, C. C. (1971). Behavior of a class of complex systems: the effect of system size on properties of terminal cycles. *Journal of Cybernetics*, voll, 1971, p 55-67.

WANG, J. Y.; COHEN, F. S. (1994). 3-D object recognition and shape estimation from image countours using B-spline, shape invariant matching, and neural networks, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol 16, n. 1.

WANG, L.; SUN, L. (1996). Automatic target recognition using higher order neural network. *IEEE Proceedings of the National Aerospace and Electronics Conference*, USA, vol 1.

WANG, C.; CHANG, E. (1997). Neural network and striped lighting pattern-based autograsping technology for flexible robotic assembly. *Journal of Robotic Systems*, vol 14, n. 7, Jul 1997, p 559-569.

WIDROW, B.; WINTER, R. G.; BAXTER, R. A. (1988). Layered neural nets for pattern recognition. *IEEE Transactions on Acoustic, Speech and SIgnal Processing*, vol 36.

WILLSHAW, D. J.; BUNEMAN, O. P.; LONGUET-HIGGINS, H. C. (1969). Non-holographic associative memory. *Nature*, n. 222, p. 960-962.

WONG, W. H.; SIU, W. C.; LAM, K. M. (1995). Generation of moment invariants and their uses for character recognition. *Pattern Recognition Letters*, vol 16, n. 2.

WONG, H.; LEU, M.C. (1993). Adaptive Genetic ALgorithm for Optimal Printed Circuit Board Assembly Planning. *Annals of the CIRP*, vol 42, January.

WONG, A.; RONG, L.; LIANG, X. (1998). Robot vision: Model synthesis for 3D objects. *Innovations in Theory, Practice and Applications - IEEE International Conference on Intelligent Robots and Systems*, vol 3, 1998, IEEE, Piscataway, NJ, USA, p 1820-1827.

WU, C. M.; JINAG, B. C.; WUS, C. H. (1991). Using neural networks for robot positioning control. *Robotics & Computer-Integrated Manufacturing*, vol 10, n. 3.

WUENCHE, A. (1992). Basis of attraction in disordered networks. *Proceedings of the ICANN 92 Conference*, Brighton, UK, September, Elsevier.

XING, G.; FELTHAM, R. (1994). Pyramidal Neural Networking for mammogram tumour pattern recognition. *IEEE International Conference on Neural Networks*, Proceedings, vol 6.

YAN, H.; WU, J. (1994). Character and line extraction from color map images using a multi-layer neural network. *Pattern Recognition Letters*, vol 15.

YOU, S. D.; FORD, G. E. (1993). Network model for invariant object recognition and rotation angle estimation. *International Joint Conference on Neural Networks*, Proceedings.

YOU, S. D.; FORD, G. E. (1994). Network model for invariant object recognition. *Pattern Recogniton Letters*, vol 15, n. 8.

YOUNG, S. S.; SCOTT, P. D.; NASRABADI, N. M. (1997). Object recognition using multilayer Hopfield neural network. *IEEE Transactions on Image Processing*, vol 6, n. 3, Mar 1997, p 357-372.

YU, F.; CHEN, H.; WANG, K. (1996). Vision system with dual CCD cameras for automatic guided vehicle *Guangdianzi Jiguang/Optronics Lasers*, vol 7, n. 4.

YUCEER, C.; OFLAZER, K. (1993). A rotation, scaling and translation invariant pattern classification system. *Pattern Recognition*, v. 26, n. 5.

ZHANG, W.; HASEGAWA, A.; ITOH, K.; ICHIOKA, Y. (1991). Error backpropagation with minimum-entropy weights: a technique for better generalization on 2D shift invariants. *International Joint Conference on Neural Networks*, Proceedings, p. 645-648.

ZIRSK, B. I. (1983). Flexibility is key to Automated Material Transport System for Manufacturing Cells. *Industrial Engineering*, vol 15, n.11, p. 58 - 64.

APÊNDICE I

TEOREMA FUNDAMENTAL DOS ALGORITMOS GENÉTICOS

O Teorema Fundamental dos Algoritmos Genéticos (AG) representa, em termos simples, uma combinação direta entre cadeias de caracteres que melhor representam as propriedades de uma população. Através da troca de blocos de esquemas, objetiva-se a construção de novas populações nas quais, por intermédio de mecanismos de cruzamento, reprodução e mutação, as características desejadas vão sendo sucessivamente melhor representadas.

As cadeias de caracteres representam uma coleção de objetos binários e podem ser representadas da seguinte forma:

$A = 0111000$ ou $A = a_1a_2a_3a_4a_5a_6a_7$

Os genes estão formados pelos elementos a_i das cadeias de caracteres. Agora considera-se o esquema H, construído sobre o alfabeto {0,1,*}:

$H = *11*0**$

onde o caráter (*) significa qualquer valor binário {0,1}.

As propriedades básicas dos esquemas estão indicadas a seguir e são úteis no processo de discussão e de classificação de similaridades na análise de conjuntos de cadeias de caracteres. Através delas, é possível avaliar os efeitos do uso dos operadores genéticos em blocos pré-construídos dentro de uma população.

i) Ordem (O(H)): Representa o número de posições fixadas do esquema. Por exemplo, no esquema H, O(H) = 3, corresponde aos dois "1" e ao "0".

ii) Comprimento de definição δ(H): Distância entre a primeira e a última posição especificadas para uma cadeia de caracteres. Por exemplo, no esquema H, δ(H) = 5-2 = 3, porque temos o primeiro "1" ocupando a posição dois e o último "0" ocupando a posição cinco.

Seja m = m(H,t) a representação de um exemplo de esquema particular H, contido em uma população A(t), no tempo t. Durante o processo reprodutivo, uma cadeia de caracteres é copiada de acordo com sua aptidão ou, mais especificamente, uma cadeia de caracteres $A_j(t)$ é selecionada com probabilidade:

$$p_i = \frac{f_i}{\sum_j f_j} \qquad \text{(I.1.1)}$$

onde f_i representa o valor da função avaliação para a cadeia de caracteres i.

Após um processo de geração de uma nova população, o número esperado de indivíduos representativos de um esquema H, no tempo (t+1), é dado por:

$$m(H, t+1) = \frac{f(H)}{\hat{f}} m(H, t) \qquad \text{(I.1.2)}$$

onde $f(H)$ e $\hat{f}$ representam a aptidão das cadeias de caracteres representativos do esquema H e a aptidão média das cadeias de caracteres existentes na população no tempo t, respectivamente.

Assumindo que um esquema apresenta uma aptidão acima da média em $c\hat{f}$ unidades, então, a equação B.1.2 pode ser reescrita:

$$m(H,t+1) = \frac{\left(\hat{f}+c\hat{f}\right)}{\hat{f}} m(H,t) \qquad (I.1.3)$$
$$= (1+c)m(H,t)$$

Partindo de t = 0 e supondo c um valor constante, tem-se:

$$\begin{aligned} m(H,1) &= (1+c)m(H,0) \\ m(H,2) &= (1+c)^2 m(H,0) \\ &\vdots \\ &\vdots \\ m(H,t) &= (1+c)^t m(H,0) \end{aligned} \qquad (I.1.4)$$

Assim, um esquema acima da média deverá crescer exponencialmente com o tempo (em contrapartida, aqueles que se situam abaixo da média tenderão a desaparecer). Vale dizer que o valor de "c" é a taxa de crescimento para o esquema considerado.

O cruzamento representa, na realidade, uma troca de informação aleatória entre cadeias de caracteres, proporcionando a criação de novas estruturas sem que ocorra uma ruptura na estratégia de alocação fornecida pela reprodução. Paralelamente, o cruzamento tem o efeito de aumentar exponencialmente a proporção de alguns esquemas verificados em uma dada população. Assim, por exemplo, consideramos:

$$A = 0\ 1\ 1\ |\ 1\ 0\ 0\ 0$$
$$H_1 = *\ 1\ *\ |\ *\ *\ *\ 0$$
$$H_2 = *\ *\ *\ |\ 1\ 0\ *\ *$$

H_1 e H_2 são representados na cadeia de caracteres A. Considerando um cruzamento efetuado entre a terceira e a quarta posição (|), o esquema H_2 vai sobreviver. Entretanto, nada se pode afirmar sobre o esquema H_1. Para uma cadeia de caracteres de tamanho L, existem (L-1) pontos de cortes possíveis. A probabilidade (P_d) de um esquema H_j ser destruído é:

$$p_d = \frac{\delta(H_j)}{L-1} \qquad (I.1.5)$$

Considere-se, agora, a probabilidade de um esquema sobreviver. Essa probabilidade poderá ser obtida a partir de análises realizadas sobre um processo de destruição de um esquema: se o cruzamento é realizado através de processos aleatórios, então, P_c é a probabilidade de um esquema ser escolhido para um dado cruzamento. A probabilidade de um esquema escolhido ser destruído é conhecida e é menor ou igual a $\frac{\delta(H_j)}{L-1}$. Logo, a probabilidade de o esquema a ser escolhido e destruído será menor ou igual a $p_c \frac{\delta(H_j)}{L-1}$, onde obtém-se a probabilidade de sobrevivência para um esquema:

$$p_s \geq 1 - p_c \frac{\delta(H_j)}{L-1} \qquad \text{(I.1.6)}$$

O número esperado de elementos de um esquema particular na próxima geração considerando o efeito combinado de reprodução e de cruzamento e assumindo independência entre as operações de reprodução e cruzamento, é dado por:

$$m(H, t+1) \geq m(H, t) \frac{f(H)}{\hat{f}} [1 - p_c \frac{\delta(H)}{L-1}] \qquad \text{(I.1.7)}$$

Considera-se mutação como sendo uma alteração aleatória de uma única posição e admite-se que a sua ocorrência se dá com probabilidade p_m. Assim, para uma única posição, a probabilidade de sobrevivência é igual a 1- p_m. Se um dado esquema sobrevive, isso implica que cada uma de suas posições fixas, (O(H)), também, sobrevive. Então, para um esquema com O(H) = n, obtém-se a seguinte probabilidade de sobrevivência:

$$p_s' = (1 - p_m) \ldots (1 - p_m) = (1 - p_m)^n = (1 - p_m)^{O(H)} \qquad \text{(I.1.8)}$$

Expandindo-se o termo $(1-p_m)^{O(H)}$, através de coeficientes binomiais, para valores de p_m pequenos, pode-se desprezar os termos de grau superior a 1 e obtém-se a seguinte aproximação:

$$p_s' = 1 - O(H) p_m \qquad \text{(I.1.9)}$$

Assim, combinando-se os efeitos da reprodução, cruzamento e mutação, o valor esperado de cópias que um dado esquema H irá receber na próxima geração é dado por:

$$m(H,t+1) \geq m(H,t)\frac{f(H)}{\hat{f}}[1-p_c\frac{\delta(H)}{L-1}][1-O(H)p_m]$$

$$m(H,t+1) \geq m(H,t)\frac{f(H)}{\hat{f}}[1-O(H)p_m-\frac{\delta(H)}{L-1}p_c+\frac{\delta(H)}{L-1}p_cO(H)p_m]$$

(II.1.10)

Ignorando os produtos cruzados (probabilidade desprezível de ocorrência), vem:

$$m(H,t+1) \geq m(H,t)\frac{f(H)}{\hat{f}}[1-\frac{\delta(H)}{L-1}p_c-O(H)p_m] \qquad \text{(II.1.11)}$$

Se a ordem do esquema O(H) for pequena, então, o produto $O(H)p_m$ também o será. Adicionalmente, considerando para um esquema uma probabilidade pequena de ser escolhida e destruída, o efeito conjugado referente à sua presença, em uma geração futura, depende apenas de $\frac{f(H)}{\hat{f}}$. Ou seja, para esquemas de alto grau de ajuste, comprimento pequeno e baixa ordem, o número esperado de indivíduos de esquema nas gerações subseqüentes cresce exponencialmente. Tal resultado é conhecido como o Teorema Fundamental dos Algoritmos Genéticos.

O quadro do desempenho do Algoritmo Genético (AG) fica muito mais claro com a perspectiva fornecida pelos esquemas. Esquemas curtos, de baixa ordem e de alta aptidão são amostrados, reconhecidos e reamostrados para formar cadeias de caracteres de ajustes potencialmente mais elevados. Esse procedimento torna a busca de uma solução ótima, mais eficiente, diminuindo a quantidade de processamento.

APÊNDICE II

ARQUITETURA CPAD

II. ARQUITETURA CPAD

Um dos principais problemas envolvendo o uso de controladores com múltiplas CPUs é que eles são dedicados para uma determinada classe de aplicações, não apresentando versatilidade suficiente para utilização situações específicas.

A seguir, apresenta-se a definição de arquiteturas paralelas voltadas para sistemas de controle industrial, proporcionando o hardware básico para o desenvolvimento de toda uma família de controladores de alto desempenho. São apresentados os blocos arquiteturais básicos e como eles podem ser associados para atender os requisitos de desempenho e modularidade necessários para uma ampla gama de aplicações.

Definição da Topologia e do Número de Processadores

Em função da existência de uma grande quantidade de topologias de arquiteturas paralelas que podem ser construídas à partir de uma única família de processadores, deve-se escolher uma topologia física de interligação dos processadores que seja a mais adequada para a aplicação específica, para que o melhor desempenho seja obtido. Esta escolha, no entanto, pode prejudicar o desempenho da arquitetura para outras aplicações. Algumas topologias são bastante gerais, como a árvore binária e o hipercubo, podendo ser utilizadas com sucesso em uma ampla gama de aplicações.

A topologia escolhida foi a de árvore binária, a qual proporciona economia de memória e boa generalização. A figura II.1, a seguir, ilustra tais vantagens.

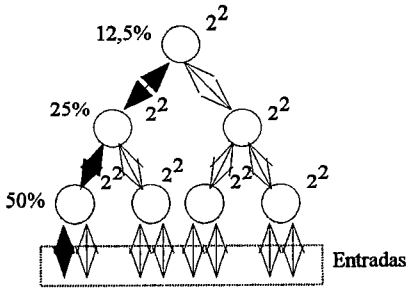


Figura II.1. Arquitetura em árvore com 7 nós de processamento

Com sete nós de processamento e duas entradas obtem-se $7.2^2 = 28$ (quantidade de memória). Caso fosse utilizado apenas um nó teríamos $2^8 = 64$ (quantidade de memória).

Hardware do CPAD

A escolha de uma arquitetura MIMD com memória distribuída permite maior facilidade de distribuição física dos elementos de processamento e facilidade de implementação. Além disso, a velocidade de transmissão de dados entre os elementos de processamento proporcionada por sistemas de rede modernos é mais que suficiente para as aplicações previstas.

O CPAD (Controlador Paralelo com Alto Desempenho) foi definido para atender aos requisitos de processamento para aplicações em controladores de máquinas ferramentas e sistemas de reconhecimento de imagens. No CPAD, podem existir até 15 nós de processamento dispostos em árvore binária com 4 níveis e dois canais de comunicação entre cada nó, sendo um dedicado para o recebimento e outro para o envio de dados.

Além dos nós, um microcomputador Pentium é utilizado como servidor de arquivos, para carga do sistema e como interface com o usuário. Esse microcomputador é conectado ao primeiro nó através de dois canais paralelos, da mesma maneira como é feito entre os demais nós do sistema. A figura II.2, a seguir, ilustra a arquitetura CPAD.

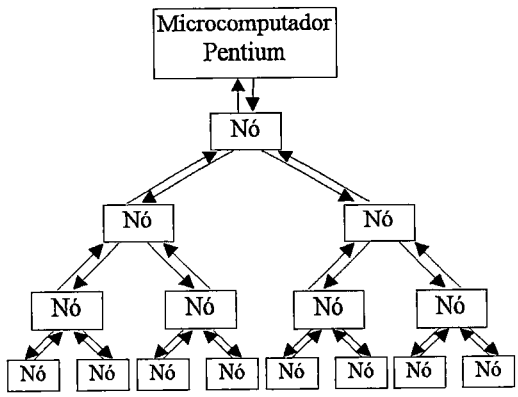


Figura II.2. Arquitetura do controlador paralelo

A escolha da arquitetura com topologia em árvore binária facilita a transmissão e busca de dados entre o primeiro e o último nível hierárquico. Essa característica vem ao encontro das necessidades das principais aplicações previstas para CPAD. Outra razão dessa escolha é a modularidade apresentada pelo sistema, que pode ter seu desempenho computacional facilmente aumentado pela inserção de novos nós na árvore.

O nó de processamento do CPAD é mostrado na figura II.3 e possui 4 elementos básicos: o processador, canais de comunicação, memória local com capacidade acima de 4Mbytes e interfaces de entrada e saída de dados.

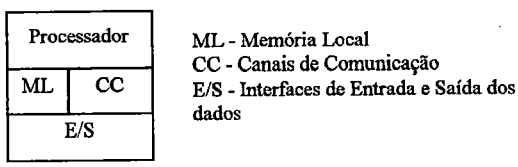


Figura II.3. Componentes dos Nós de Processamento

Na prática os nós do CPAD podem ser implementados com qualquer tipo de processador. Os dois processadores disponíveis comercialmente e que melhor se encaixam neste projeto são o Transputer T9000 da INMOS e o TMS320C40 da Texas Instruments. Esses dois processadores incorporam características paralelas on-chip para conseguir alto desempenho individual. Além disso, esses processadores oferecem características extras para expansão *off-chip* do paralelismo.

Organização dos Nós Processadores

Os nós processadores ou elementos de processamento constituem as unidades básicas para a construção da arquitetura CPAD. Além dos Elementos de Processamento (EP), cada nó é formado por canais de comunicação e memórias locais (RAM dinâmica).

Os canais de comunicação geralmente limitam o desempenho das arquiteturas paralelas pela sua baixa capacidade de transmissão de dados em relação ao desempenho do processador. No caso da arquitetura CPAD, que é uma arquitetura em árvore binária, o número mínimo de canais de comunicação é três e a velocidade desejada desses canais é a maior possível. Quanto mais velozes forem os canais de comunicação maior será o desempenho do sistema.

As memórias utilizadas em cada nó do CPAD possuem a capacidade de 4M bytes. A escolha baseou-se no custo das memórias disponíveis comercialmente, as mesmas utilizadas nos microcomputadores Pentium. Conforme a necessidade, pode-se aumentar facilmente a sua capacidade.

Processadores Adequados para o CPAD

O desempenho dos computadores com arquitetura paralela é bastante dependente da velocidade de processamento e de transmissão dos dados entre os processadores. No mercado existem muitos processadores capazes de processar em altíssima velocidade tais como: 80486, i860, 68040 e Pentium. Nenhum possui características especiais para transmissão de dados. Os três processadores que melhor preenchem essas características são: o Transputer 805, o Transputer T9000 e o DSP TMS320C40.

DSP TMS320C40

As principais características do TMS320C40 para processamento paralelo são: as seis portas de comunicação que permitem a comunicação direta entre os processadores, os seis canais de DMA que possibilitam entrada e saída concorrentes para maximizar o desempenho da CPU e o alto desempenho apresentando 275 MOPS, 50 MFLOPS, 25MIPS, 320MBYTES/S com ciclo de 40 ns (TEXAS, 1993).

O TMS320C40 e o T9000 se igualam no número de contadores de 32 bits de propósito geral e na existência de um sistema *autoloader*, cujas funções são recepção, armazenamento e execução de programas e/ou dados do computador hospedeiro em uma área da memória previamente definida.

O TMS320C40 é superior ao T9000 em relação à transmissão de dados entre processadores. O processador T9000 possui quatro canais seriais que podem operar a 100 Mbits/s e o processador TMS320C40 possui 6 canais paralelos que podem operar a 120 Mbytes/s. A taxa de transmissão agregada do TMS320C40 é 14,4 vezes maior que a taxa do T9000.

Há dois outros bons motivos para adotar o processador TMS320C40. O primeiro é o apoio técnico e científico que a empresa norte-americana Texas fornece aos pesquisadores e projetistas de computadores. O segundo é a longa tradição que a Texas tem na fabricação de circuitos integrados, garantindo o suprimento de peças e a evolução da arquitetura.

Ambiente de Desenvolvimento para o 'C40

Para a construção do 'C40 foram utilizadas ferramentas básicas que trabalham no ambiente DOS:

- **Compilador C:** traduz arquivos fontes em linguagem C para arquivos fonte em linguagem assembler, seguindo o padrão ANSI;
- **Assembler:** traduz arquivos fontes em linguagem assembly em arquivos objeto em linguagem de máquina;
- **Archiver:** permite juntar vários procedimentos individuais em uma única chamada de biblioteca, além de apagar, extrair, substituir e adicionar membros na biblioteca;
- **Linker:** combina arquivos objeto em um único módulo objeto, que pode ser alocado na memória do sistema e executado pelo TMS320C40;
- **Hex30:** converte os arquivos objeto tipo COFF para os formatos Intel, Tektronix, Texas e ASCII, para que possam ser gravados em memórias EPROMs;
- **Simulador:** permite a execução de programas desenvolvidos em linguagem C e/ou Assembly, verificar os registradores do processador e variáveis no microcomputador PC.

A figura II.4, a seguir, mostra a conexão entre os computadores PC e 'C40 e a maneira que as ferramentas de software são utilizadas para a geração do programa executável pelo 'C40.

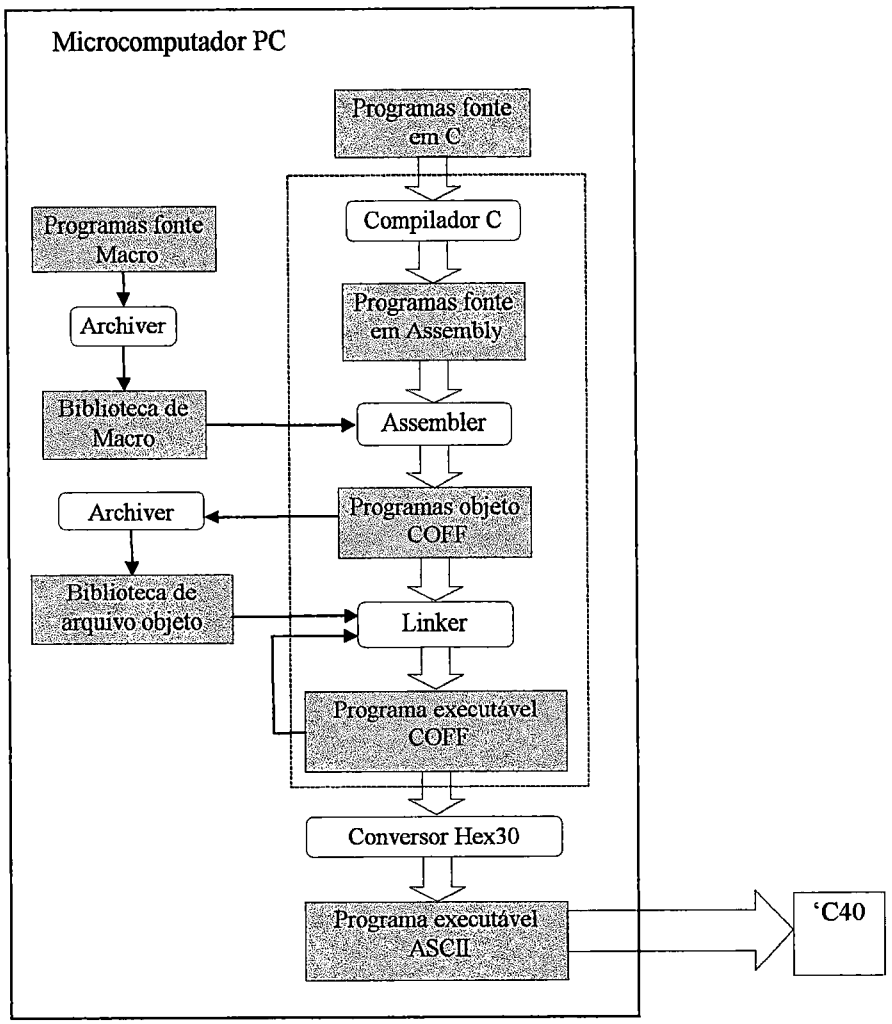


Figura II. 4. Fluxo de desenvolvimento de programas para o 'C40

A figura II.5, a seguir, ilustra as etapas de desenvolvimento do projeto do 'C40.

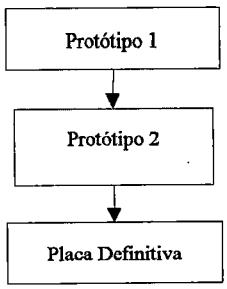


Figura II.5. Fases do projeto

Protótipo 1 - Construção e testes de hardware e software. Para que o microcomputador Pentium e o C40 se comunicassem foi desenvolvida uma placa, padrão ISA, com interface paralela baseada no CI 8255 da Intel, conforme diagrama de blocos mostrado na figura II.6.

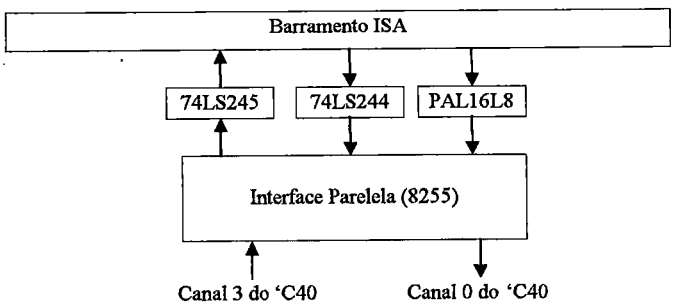


Figura II.6. Interface para comunicação entre o 'C40 e o Pentium

Protótipo 2 - Construção de três placas com apenas um processador TMS320C40, memórias dinâmicas, canais paralelos e circuito de decodificação e "refresh" baseados em CPLDs. Através dessas placas foram feitos os testes de comunicação C40-C40, processamento paralelo, circuito de "refresh" das memórias dinâmicas, circuito de aquisição e envio de dados.

Implementação do CPAD

O CPAD é formado por vários nós processadores chamados de 'C40. O nome 'C40 é devido à adoção do processador TMS320C40. Nessa implementação cada nó processador dispõe de um processador 'C40 e memória local

O Módulo Básico de Processadores

A arquitetura CPAD é formada basicamente por módulos, como esquematizado na figura II.7. Cada módulo é constituído de um nó pai e dois nós filhos. As principais razões dessa configuração são: permitir uma pequena distância entre os canais dos processadores, reduzir o número de canais que são interligados com fios e tornou mais compacta a implementação do sistema.

Não existem diferenças entre os módulos e eles podem ser encaixados em qualquer posição no bastidor, que será descrito no próximo sub-item. A hierarquia entre os módulos é determinada pela ligação do nó filho de um módulo com o nó pai de outro. O meio físico de comunicação é provido por um barramento no bastidor ao qual os módulos estão conectados.

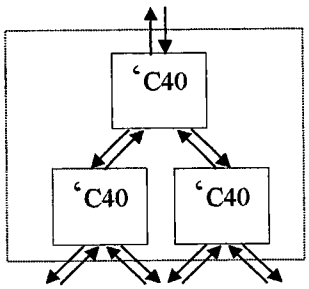


Figura II.7. Módulo básico da arquitetura CPAD

Fisicamente, o controlador pode ser alojado em um bastidor com 14 posições sendo que a primeira posição é a da fonte de alimentação, 5 posições são ocupadas pelos módulos de processadores e 8 são reservadas para possíveis expansões ou hardware adicional (conversores analógico-digitais, portas de entrada/saída, etc). A figura II.8 esquematiza a disposição dos módulos no bastidor.

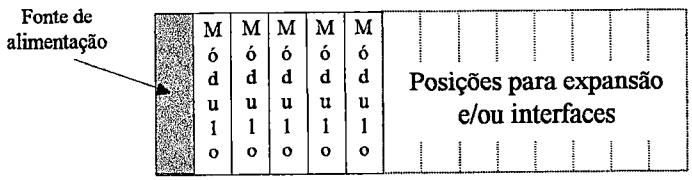


Figura II.8. Vista frontal do bastidor

O projeto inicial dos elementos de processamento do CPAD foi desenvolvido com circuitos integrados (CIs) convencionais, como mostra o diagrama de blocos na figura II.9. No bloco (1) estão os seis canais de comunicação que transmitem e recebem informações dos outros 'C40, no bloco (2) está o oscilador de clock de 50 MHz e no bloco (3) está o circuito de reset manual. Caso o módulo fique "travado", o operador reinicia-o apertando o botão de reset.

No bloco (4) está o processador TMS320C40. No bloco (5) estão as memórias dinâmicas, cuja capacidade é de 4 Mbytes. No bloco (6) está o decodificador de endereços, cuja função é a habilitação dos endereços da memória dinâmica e no bloco (7) está o circuito de "refresh" necessário para a sua operação. No bloco (8) está o circuito de tratamento de interrupções, cuja função é a de avisar o processador que algum dispostivo externo está tentando enviar dados ou disparar a execução de alguma tarefa.

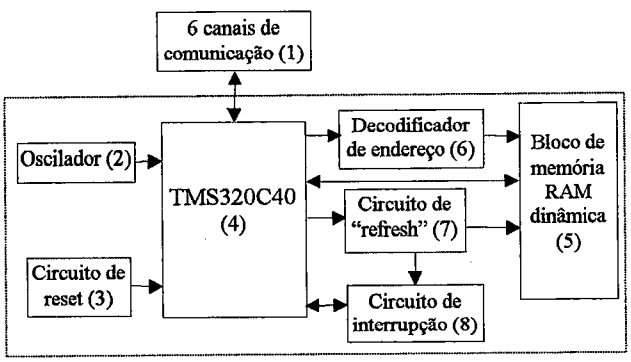


Figura II.9. Diagrama de blocos do 'C40 sem CPLD

Para simplificar a construção e diminuir o tamanho físico do sistema, optou-se pelo emprego de CPLDs (*Complex Programmable Logic Device*) em substituição os blocos 6, 7 e 8 como mostra a figura II.10.

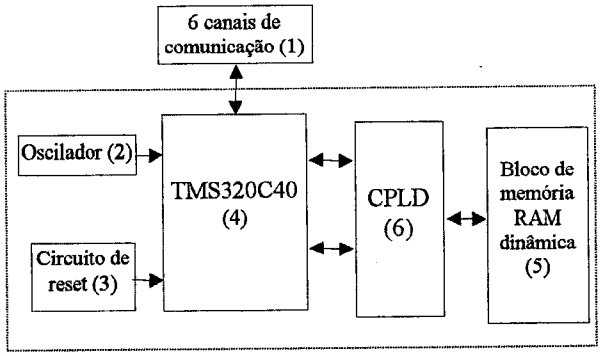


Figura II.10. Diagrama de blocos do 'C40 com CPLD

APÊNDICE III

PRINCIPAIS ROTINAS DE SOFTWARE DESENVOLVIDAS

Arquivos de Manipulação de Imagens

```
/************************
 PCX.c - visualizacao de arquivos pcx.
*************************/
#include <mem.h>
#include <math.h>
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include "Palette.h"
#include "PCX.h"
/************** Macros ****************/
#define     RGB_RED                     0
#define     RGB_GREEN          1
#define     RGB_BLUE           2
#define     HIGHCOLOUR(r,g,b) (((((unsigned int)r >> 3) & 0x1f) << 10) + \
                               ((((unsigned int)g >> 3) & 0x1f) << 5) + \
                               (((unsigned int)b >> 3) & 0x1f))
#define SCREEN_WIDTH 320
#define SCREEN_HEIGHT 200
/************** Tipos ********************/
typedef struct PCX_HEAD
 {
 Byte manufacturer;
 Byte version;
 Byte encoding;
 Byte bits_per_pixel;
 Short  xmin,ymin;
 Short  xmax,ymax;
 Short  hres;
 Short  vres;
 Byte palette[48];
 Byte reserved;
 Byte colour_planes;
 Short  bytes_per_line;
          Short  palette_type;
          Byte filler[58];
          } PCX_head;
/************ Variaveis Locais ************/
Byte    palette[768]; /* Palette do PCX. */
FILE    *file;      /* Arquivo PCX. */
PCX_head head;      /* header do PCX. */
Byte    bits;       /* bits por pixel. */
Short   bytes;      /* bytes por linha. */
Short   width,height; /* Largura e profundidade. */
Short   n_colors;   /* Numero de cores. */
Short   map_color[256];
Short   pcx_x, pcx_y; /* Coordenadas de desenho iniciais. */
Int     scl_width;
Int     scl_height;   /* Tamanho do pcx scalado. */
Double  scl_dy;     /* Razao entre o pcx scalado e o real. */
Double  scl_dx;     /* Razao entre o pcx scalado e o real. */

/* Palette default de 16 cores. */
static Byte pcxpalette[48]=
 {
 0x00,0x00,0x0E,0x00,0x52,0x07,0x2C,0x00,
 0x0E,0x00,0x00,0x00,0xF8,0x01,0x2C,0x00,
 0x85,0x0F,0x42,0x00,0x21,0x00,0x00,0x00,
 0x00,0x00,0x6A,0x24,0x9B,0x49,0xA1,0x5E,
 0x90,0x5E,0x18,0x5E,0x84,0x14,0xD9,0x95,
 0xA0,0x14,0x12,0x00,0x06,0x00,0x68,0x1F
 };

/* Mascaras. */
Byte masktable[8]={0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01};
Byte bittable[8]= {0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};

/********************** PCX ***************/
/*********************
```

```
            Desenyha um pixel na tela
*********************/
void put_pixel(Int x, Int y, Byte d)
{
asm {
push es
push ax
push dx
mov ax, 0A000h
mov es, ax
mov ax, y;
mov dx, 320
mul dx
add ax, x
mov bx, ax
mov al, d
mov [es:bx], al
pop dx
pop         ax
pop es
}
}

/*************************
            Acha a cor de palette real
mais proxima do cor PCX.
            A paleta real esta definida na
variavel global palette.
            Retorna o indice da cor desta paleta
**************************/
Int get_color(Int color)
{
Int  c, i;
Int  r, g, b;
Double d, dif;
Double dr, dg, db;
            if (map_color[color]>=0)
                        return map_color[color];
r=palette[color*3];
g=palette[color*3+1];
b=palette[color*3+2];
c=pal_index;
dif=sqrt(sqr(r-pal[pal_index].r)+
                                                sqr(g-pal[pal_index].g)+
                                                sqr(b-pal[pal_index].b));
            for (i=1+pal_index; i<pal_index+pal_size; i++)
                        {
                        dr=(Double)sqr(r-pal[i].r);
                        dg=(Double)sqr(g-pal[i].g);
                        db=(Double)sqr(b-pal[i].b);
                        d=sqrt(dr+dg+db);
    if (d<dif)
      {
      dif=d;
                                                c=i;
      }
   }
map_color[color]=c;
return c;
}
/*************************
  Acha a cor de palette real
mais proxima do cor RGB.
            A paleta real esta definida na
variavel global palette.
            Retorna os valores em RGB
**************************/
Int get_rgb_color(Byte r, Byte g, Byte b)
{
Int  c, i;
Double d, dif;
Double dr, dg, db;
c=pal_index;
dif=abs(r-pal[c].r) +
```

```
                                abs(g-pal[c].g) +
                        abs(b-pal[c].b);
                for (i=1+pal_index; i<pal_index+pal_size; i++)
                        {
  dr=abs(r-pal[i].r);
  dg=abs(g-pal[i].g);
  db=abs(b-pal[i].b);
  d=dr+dg+db;

   if (d<dif)
    {
    dif=d;
    c=i;
    }
  }
return c;
}

/************************
 Le uma linha PCX e a grava
no buffer de linha: line.
************************/
void read_pcx_line(Byte *line, FILE *file)
{
Short n;
Short c,i;
n=0;
        do
        {
        c=fgetc(file) & 0xff;
                if((c & 0xc0) == 0xc0)
                        {
                        i=c & 0x3f;
                        c=fgetc(file);
                                        if (bits==8)
                                                while(i--) line[n++]=get_color(c);
                                        else
                                                while (i--) line[n++]=c;
                        }
                else
                        if (bits==8)
                                line[n++]=get_color(c);
                        else line[n++]=c;
                } while(n < bytes);


/*************************
        Escreve uma linha do PCX na tela
*************************/
void write_pcx_line(Int y, Byte *buffer)
{
Int  i;
Byte *d;
d=buffer;
        for (i=0; i<width; i++, d++)
                put_pixel(i+pcx_x, y+pcx_y, *d);
}


/***************
        Abre um arquivo pcx.
Inicializa o seu header.
        Retorna 0 caso nao consiga abrir o arquivo
***************/
Int load_pcx(Int px, Int py, Char *name)
{
extern void set_palette(Int, Int, Byte*);
Short  y, j;
Short  i;
Byte   *line_buffer, *extra_buffer;
file=fopen((char *)name,"rb");
        if (!file)
                return 0;
```

```
/* Le header. */
fread(&head, 1, sizeof(PCX_head), file);

        if (head.manufacturer!=10)
  {
  fclose(file);
                return 0;
  }

 for (j=0; j<256; j++)
  map_color[j]=-1;

/* Define coordenadas iniciais. */
pcx_x=px;
pcx_y=py;

/* Define largura e profundidade. */
width=head.xmax-head.xmin+1;
height=head.ymax-head.ymin+1;
 if (width+pcx_x>SCREEN_WIDTH)
  width=SCREEN_WIDTH-pcx_x;
 if (height+pcx_y>SCREEN_HEIGHT)
  height=SCREEN_HEIGHT-pcx_y;

// Centraliza a figura
        if (pcx_x==CENTER_PCX_X)
                pcx_x=(SCREEN_WIDTH-width)/2;
        if (pcx_y==CENTER_PCX_Y)
                pcx_y=(SCREEN_HEIGHT-height)/2;
        if(head.bits_per_pixel==8 && head.colour_planes==3) bits=24;
        else if(head.bits_per_pixel==1) bits=head.colour_planes;
 else bits=head.bits_per_pixel;

 /* Define a  palette. */
        if(bits==8 && head.version>=5)
                {
                fseek(file,-769L,SEEK_END);

                        if(fgetc(file)==12)
                                {
                                        if(fread(palette,1,768,file) != 768)
                                                {
                                                fclose(file);
                                                return 0;
                                                }
                                }
                        else
                                memcpy(palette,head.palette,48);

                set_palette(0, 255, palette);
                }
        else if (head.version == 3)
                memcpy(palette,pcxpalette,48);
        else
                memcpy(palette,head.palette,48);

/* Define o comprimento de linha. */
        if(bits > 1 && bits <= 4)
                {
    if(head.bits_per_pixel==4 && head.colour_planes==1)
      bytes=head.bytes_per_line;
    else bytes=head.bytes_per_line*bits;
  }
 else if (bits==24)
  bytes=head.bytes_per_line*3;
 else bytes=head.bytes_per_line;

n_colors= 2 << (bits-1);

/****** Fim da carga do header do PCX. ******/
/* numero de bits em uma linha. */
//n=(width+7)/8;

/* Aloca buffer de linha. */
```

```
        line_buffer=(Byte*) malloc(nmax(bytes, width));
extra_buffer=(Byte*) malloc(nmax(bytes, width));

/* Restaura posicao. */
fseek(file,(Dword)sizeof(PCX_head),SEEK_SET);

/* Inicializa linha. */
        for (y=0; y<height; y++)
                {
                /* Le uma linha PCX. */
                read_pcx_line(line_buffer, file);

                        switch (bits)
                                {
                                case 1:
                                /* Linha de 1 bit depth. */
                                        for (j=0; j<head.bytes_per_line; j++)
                                                for (i=0; i<8; i++)
                                                        extra_buffer[j*8+i]=
                                                                255*((line_buffer[j] &
masktable[i])>>(7-i));

    write_pcx_line(y, extra_buffer);
    break;

    case 4:
     if (head.bits_per_pixel == 4 && head.colour_planes == 1)
      {
       for (i=0; i<head.bytes_per_line; i++)
        {
        extra_buffer[i*2]=get_color(line_buffer[i] & 0xf);
        extra_buffer[i*2+1]=get_color((line_buffer[i] & 0xf0) >> 4);
        }

                                                write_pcx_line(y, extra_buffer);
                                                }
                                        else
                                                {
                                                        for (i=0; i<head.bytes_per_line; i++);
                                                }
    break;
   case 8:
                        /* eight-bit lines go as is */
                        write_pcx_line(y, line_buffer);
    break;

   case 24:
          for (j=0;j<width;++j)
     extra_buffer[j]=(line_buffer[j]/64)*64+
             (line_buffer[j+head.bytes_per_line]/32)*8+
             line_buffer[j+head.bytes_per_line*2]/32;

                                        write_pcx_line(y, extra_buffer);
    break;
        }

  }
/* Libera memoria. */
free(extra_buffer);
free(line_buffer);

fclose(file);

return 1;
}


/*********************************

        Palette.c - palette de cores

*********************************/
```

```
#include "Palette.h"

/***************** Variaveis Globais ****************/

Short   pal_index;   // Indice inicial da palette do jogo
Short   pal_size;    /* tamanho da palette do jogo. */
RGB     pal[MAX_COLORS];    /* Palette do Jogo. */


/************** Funcoes **********/


void set_palette(Int color_0, Int color_1, Byte *p)
{
Int i;
Byte c, r, g, b;

        for (i=color_0; i<=color_1; i++)
                {
                r = p[3*i+0];
                g = p[3*i+1];
                b = p[3*i+2];
                pal[i].r = r;
                pal[i].g = g;
                pal[i].b = b;
                c = i;

                asm {
                push ax
                push bx
                push cx
                push dx

                mov  dx, 03c8H
                mov  al, c
                out   dx, al        // seleciona cor ( c )

                mov  dx, 03c9H      // Seleciona registrador da palette

                mov   al, r
                shr   al, 2
                out   dx, al        // Seta Red
                mov   al, g
                shr   al, 2
                out   dx, al        // Seta Green
                mov   al, b
                shr   al, 2
                out   dx, al        // Seta Blue

                pop   dx
                pop   cx
                pop   bx
                po    ax
                }
                }
}


void init_palette(void)
{
Int  i, j, k;
Byte p[768];

        for (i=0; i<4; i++)
                for (j=0; j<8; j++)
                        for (k=0; k<8; k++)
                                {
                                p[3*(i*64+j*8+k)+0]=i*64;
                                p[3*(i*64+j*8+k)+1]=j*32;
                                p[3*(i*64+j*8+k)+2]=k*32;
                                }
```

```
pal_index=0;
pal_size=256;
set_palette(0, 255, (Byte *)p);
}
```

Filtro Mediana

```
#include <owl/pch.h>
#include "it-pi.h"
template <class Type>
void MinMax(Type list[9], int &minV, int &maxV, bool used[9]) {
  for (minV = 0 ; used[minV] ; minV += 1) ;
  maxV = minV;
  for (int i = minV+1 ; i < 9; i += 1) {
    if (!used[i]) {
      if (list[i] <  list[minV]) minV = i;
      if (list[i] >= list[maxV]) maxV = i;
    }
  }
}
template <class Type>
Type Median(Type list[9]) {
  int minV, maxV;
  bool used[9];
  memset(used, 0, sizeof used);
  MinMax(list, minV, maxV, used); used[minV] = used[maxV] = true;
  MinMax(list, minV, maxV, used); used[minV] = used[maxV] = true;
  MinMax(list, minV, maxV, used); used[minV] = used[maxV] = true;
  MinMax(list, minV, maxV, used); used[minV] = used[maxV] = true;
  MinMax(list, minV, maxV, used);
  return list[minV];
}
template <class Type>
class MedianHelper {
public:
  MedianHelper(const RWDib &, RWDib &);
};
template <class Type>
MedianHelper<Type>::MedianHelper(const RWDib &src, RWDib &dst) {
  int xPad = (src.ActualWidth() - src.Width()*sizeof(Type)) / sizeof(Type);
  Type list[9];
  Type *pSrc = (Type *)src.GetBits();
  Type *pDst = (Type *)dst.GetBits();
  int iX = src.Width();
  int iY = src.Height();
  int rowInc = src.ActualWidth() / sizeof(Type);
  pSrc += rowInc;
  pDst += rowInc;
  for (int y = 1 ; y < iY-1 ; y += 1) {
    int col;
    *pDst = *pSrc;
    list[0] = *(pSrc-rowInc);
    list[3] = *(pSrc      );
    list[6] = *(pSrc+rowInc);
    pSrc++;
    pDst++;
    list[1] = *(pSrc-rowInc);
    list[4] = *(pSrc      );
    list[7] = *(pSrc+rowInc);
    col = 2;
    for (int x = 1 ; x < iX-1 ; x += 1) {
      pSrc++;
      list[col  ] = *(pSrc-rowInc);
      list[col+3] = *(pSrc      );
      list[col+6] = *(pSrc+rowInc);
      *pDst = Median(list);
      pDst++;
      col += 1;
      if (col == 3) col = 0;
```

```
    }
*pDst = *pSrc;
    pDst++;
    pSrc++;
    pSrc += xPad;
    pDst += xPad;
  }
}
static void FilterOneImage(const RWDib &oldDib, RWDib &newDib) {
  int bpp = oldDib.Depth();
  if (bpp == 8) {
    MedianHelper<unsigned char>(oldDib, newDib);
  } else if (bpp == 16) {
    MedianHelper<unsigned short>(oldDib, newDib);
  } else if (bpp == 32) {
    MedianHelper<unsigned long>(oldDib, newDib);
  }
}
RWDib *_stdcall _export TheRoutine(ImageClientWindow *inWin) {
  RWDib *result = 0;
  int nImages = GetNumberOfImages(inWin);
  if (nImages == 0) {
    inWin->MessageBox("There is nothing to do.");
  } else if (GetSpecificImage(inWin, 0)->IsGrayScale()) {
    try {
      if (nImages == 1) {
        RWDib *inDib = GetSpecificImage(inWin, 0);
        result = new RWDib(*inDib);
        FilterOneImage(*inDib, *result);
      } else {
        TWindow *newStack = DisplayNewStack(inWin);
        for (int image = 0 ; image < nImages ; image += 1) {
          RWDib *oldImage = GetSpecificImage(inWin, image);
          RWDib *newImage = new RWDib(*oldImage);
          FilterOneImage(*oldImage, *newImage);
          AddImageToStack(newStack, newImage);
        }
      }
    } catch (TXGdi &) {
      inWin->MessageBox("Impossivel alocar a imagem resultante");
    }
  } else {
    inWin->MessageBox("O filtro pode somente trabalhar com imagens com 8, 16, e 32 bit de escala de cinza");
  }
  return result;
}
```

Rotinas de Pré-Processamento e Geração de Campos Coarse

```
#include<iostream.h>
#include<time.h>
#include<math.h>
#define DIM 128
#include<stdio.h>

void main()
{
        //declarações
        int A[DIM][DIM];
        int B[DIM][DIM];

        int i, j, R, k=0, x=0, y=0, z=0, w=0, m=0, sx=0, sy=0;

        float p = 0, TEMP1=0, TEMP2=0, p1=0, rotx=0, roty=0, Xav=0,Xt=0, Yt=0, it=0, jt=0, s=0, Rav=0, Yav=0, Txx=0,
Tyy=0, Txy=0, senO, cosO, Rx, Ry, Sx, Sy;
```

```
// rotina de calculo de Xav e Yav

                    for(j=0; j<DIM; j++)
                    {
                            ch = fgetc(in);
                     //     cout<<"\n A["<<i<<","<<j<<"] = "<< ch;
                            if (ch=='0')
                                    A[i][j]=0;
                            else
                            {
                                    A[i][j]=1;
                                    p++;
                                    Xav = (i-DIM/2) + Xav;
                                    Yav = (DIM/2-j) + Yav;
                            }
                    }
            }
            Xav = int((1/p)*Xav);
            Yav = int((1/p)*Yav);
            x = Xav-DIM/2;
            y = Yav-DIM/2;



            }

// primeira transformacao (translacao)
            for(i = 0; i < DIM; i++)
            {

                    for(j = 0; j < DIM; j++)
                    {
                            if (A[i][j]==1)
                                    {
                                    Xt=(i-DIM/2)-Xav;
                                    Yt=(DIM/2-j)-Yav;
                                    it=int(DIM/2+Xt);
                                    jt=int(DIM/2+Yt);
                                    B[it][jt] = A[i][j];
                                    }
                    }
            }

// segunda transformacao (escala)
  for(i=0; i< DIM; i++)
  {
                    for (j=0; j< DIM; j++)
                    {
                            if (B[i][j]==1)
                                    {
                                    k++;
                                    z = (i-DIM/2)*(i-DIM/2);
                                    w = (DIM/2-j)*(DIM/2-j);
                                    m = z + w;
                                    Rav = (sqrt(m)) + Rav;
                                    }
                    }
  }
            Rav = (1/p)*Rav;
                    R = DIM/4;
            s = (1/(0.25*DIM))*Rav;

            for(i = 0; i < DIM; i++)
            {
                    for(j = 0; j < DIM; j++)
                    {
                            if (B[i][j]==1)
                                    {
                                     p1++;
                                     sx = int(s*(i-DIM/2));
                                     sy = int(s*(DIM/2-j));
                                     A[DIM/2+sx][DIM/2-sy] = 1;
                            }
                    }
```

```
        }
// terceira transformacao (rotacao)

        //cálculo de Txx, Tyy e Txy

        for(i = 0; i < DIM; i++)
        {
                for(j = 0; j < DIM; j++)
                {
                 if (A[i][j] ==1)
                        {
                        Txx = ((i-DIM/2)*(i-DIM/2)) + Txx;
                        Tyy = ((DIM/2-j)*(DIM/2-j)) + Tyy;
                        Txy = ((i-DIM/2)*(DIM/2-j)) + Txy;
                        }
                }
        }

//cálculo de senO e cosO

        senO = (Tyy - Txx) + (sqrt(((Tyy - Txx)*(Tyy - Txx)) + (4*(Txy*Txy))));
        cout<<"\n seno1 ="<< senO;
        senO = senO / (sqrt((8*(Txy*Txy)) + (2*(Tyy - Txx)*(Tyy - Txx)) + (2*(Tyy - Txx)*(sqrt(((Tyy - Txx)*(Tyy - Txx)) +
(4*(Txy*Txy)))))));

        cosO = 2*Txy;
        cosO = cosO / (sqrt((8*(Txy*Txy)) + (2*(Tyy - Txx)*(Tyy - Txx)) + (2*(Tyy - Txx)*(sqrt(((Tyy - Txx)*(Tyy - Txx)) +
(4*(Txy*Txy)))))));
        cosO =-cosO;

        Xt=0;
        Yt=0;

        for(i = 0; i < DIM; i++)
        {
                for(j = 0; j < DIM; j++)
                {
                        if (A[i][j]==1)
                                {
                                Xt=(i-(DIM/2));
                                Yt=((DIM/2)-j);
                                rotx = int((cosO*Xt)-(senO*Yt));
                                roty = int((senO*Xt)+(cosO*Yt));
                                Xt=rotx;
                                Yt=roty;
                                B[(Xt+(DIM/2))][((DIM/2)-Yt)]= 1;
                                }
                }
        }
}
```

// Geração de Campos Coarse

```
#include<iostream.h>
#include<time.h>
#include<math.h>
#define DIM 16
#include<stdio.h>

void main()
{
        //declarações
        int A[DIM][DIM];
        int B[DIM][DIM];
        int C[DIM][DIM];
        int D[DIM][DIM];
        int E[DIM][DIM];
        int F[DIM][DIM];
        int G[DIM][DIM];
                int H[DIM][DIM];
```

```
int R[126][126];
int l, k, i, j, Cl, Cc, lin, col, contador, linha, coluna;

//leitura da imagem
for(k = 0; k < 128; k++)
{
        for(l=0; l<128; l++)
        {
        ch = fgetc(in);
                if (ch=='0')
                        R[l][k]=0;
                else
                        R[l][k]=1;
        }
}


// Gerando o primeiro coarse coading

Cc=0;
Cl=0;
linha=0;
coluna=0;
contador=0;

        for (i=9; i<128; i+=8)
        {
                        for (linha=0; linha<7; linha++)
                        {
                                for (coluna=i-8; coluna<i; coluna++)
                                        {
                                        if (R[linha][coluna]==1)
                                         {
                                                contador=contador+1;
                                         }
                                        }
                        }
                                if (contador>=1)
                                        A[Cl][Cc]=1;
                                else
                                        A[Cl][Cc]=0;
                         Cc=Cc+1;
                        contador=0;
        }
        Cc=1;
contador=0;

 for (j=15; j <128; j+=8)
        {
         for (i=9; i<128; i+=8)
         {
                for (linha=j-8; linha<j; linha++)
                        {
                                for (coluna=i-8; coluna<i; coluna++)
                                        {
                                                if (R[coluna][linha]==1)
                                                        {
                                                        contador=contador+1;
                                                        }
                                        }
                        }
                                if (contador>=1)
                                        A[Cl][Cc]=1;
                                else
                                        {
                                        A[Cl][Cc]=0;
                                        }

                        Cc=Cc+1;
                        contador=0;
         }
         Cc=0;
```

```
            Cl=Cl+1;
            }

// gerando o segundo coarse coading

Cl=0;
Cc=0;
contador=0;

 for (j=8; j <128; j+=8)
            {
            for (i=10; i<128; i+=8)
            {
                        for (linha=j-8; linha<j; linha++)
                                    {
                                                for (coluna=i-8; coluna<i; coluna++)
                                                            {
                                                                        if (R[coluna][linha]==1)
                                                                                    {
                                                                                    contador=contador+1;
                                                                                    }
                                                            }
                                    }
                                                if (contador>=1)
                                                            B[Cl][Cc]=1;
                                                else
                                                            {
                                                            B[Cl][Cc]=0;
                                                            }


                                    Cc=Cc+1;
                                    contador=0;
            }
            Cc=0;
            Cl=Cl+1;

            }



// gerando o terceiro coarse coading


Cl=0;
Cc=0;
contador=0;

 for (j=9; j <128; j+=8)
            {
             for (i=11; i<128; i+=8)
             {

                        for (linha=j-8; linha<j; linha++)
                                    {
                                                for (coluna=i-8; coluna<i; coluna++)
                                                            {
                                                                        if (R[coluna][linha]==1)
                                                                                    {
                                                                                    contador=contador+1;

                                                                                    }
                                                            }
                                    }

                                                if (contador>=1)
                                                            C[Cl][Cc]=1;

                                                else
                                                            {
                                                            C[Cl][Cc]=0;
                                                            }
```

```
                    Cc=Cc+1;
                    contador=0;
        }
        Cc=0;
        Cl=Cl+1;

        }

// gerando o quarto coarse coading


Cl=0;
Cc=0;
contador=0;

for (j=10; j <128; j+=8)
        {
        for (i=12; i<128; i+=8)
        {

                for (linha=j-8; linha<j; linha++)
                        {
                                for (coluna=i-8; coluna<i; coluna++)
                                        {
                                                if (R[coluna][linha]==1)
                                                        {
                                                        contador=contador+1;

                                                        }
                                        }
                        }

                                if (contador>=1)
                                        D[Cl][Cc]=1;

                                else
                                        {
                                        D[Cl][Cc]=0;
                                        }


                    Cc=Cc+1;
                    contador=0;
        }
        Cc=0;
        Cl=Cl+1;

        }



// gerando o quinto coarse coading

Cl=0;
Cc=0;
contador=0;

for (j=11; j <128; j+=8)
        {
        for (i=13; i<128; i+=8)
        {

                for (linha=j-8; linha<j; linha++)
                        {
                                for (coluna=i-8; coluna<i; coluna++)
                                        {
                                                if (R[coluna][linha]==1)
                                                        {
                                                        contador=contador+1;

                                                                }
```

```
                                        }
                                }

                                        if (contador>=1)
                                                E[Cl][Cc]=1;

                                        else
                                                {
                                                E[Cl][Cc]=0;
                                                }


                                Cc=Cc+1;
                                contador=0;
        }
        Cc=0;
        Cl=Cl+1;

        }


// gerando o sexto coarse coading

Cl=0;
Cc=0;
contador=0;

for (j=12; j <128; j+=8)
        {
        for (i=14; i<128; i+=8)
        {

                for (linha=j-8; linha<j; linha++)
                        {
                                for (coluna=i-8; coluna<i; coluna++)
                                        {
                                                if (R[coluna][linha]==1)
                                                        {
                                                        contador=contador+1;

                                                        }
                                        }
                                }

                                        if (contador>=1)
                                                F[Cl][Cc]=1;

                                        else
                                                {
                                                F[Cl][Cc]=0;
                                                }


                                Cc=Cc+1;
                                contador=0;
        }
        Cc=0;
        Cl=Cl+1;

        }


// gerando o setimo coarse coading


Cl=0;
Cc=0;
contador=0;
```

```
for (j=13; j <128; j+=8)
        {
        for (i=15; i<128; i+=8)
        {

                for (linha=j-8; linha<j; linha++)
                        {
                                for (coluna=i-8; coluna<i; coluna++)
                                        {
                                                if (R[coluna][linha]==1)
                                                        {
                                                        contador=contador+1;

                                                        }
                                        }
                        }

                                if (contador>=1)
                                        G[Cl][Cc]=1;

                                else
                                        {
                                        G[Cl][Cc]=0;
                                        }


                                Cc=Cc+1;
                                contador=0;
        }
        Cc=0;
        Cl=Cl+1;

        }

// gerando o oitavo coarse coading
Cl=0;
Cc=0;
contador=0;

 for (j=14; j <128; j+=8)
        {
        for (i=16; i<128; i+=8)
        {

                for (linha=j-8; linha<j; linha++)
                        {
                                for (coluna=i-8; coluna<i; coluna++)
                                        {
                                                if (R[coluna][linha]==1)
                                                        {
                                                        contador=contador+1;

                                                        }
                                        }
                        }

                                if (contador>=1)
                                        H[Cl][Cc]=1;

                                else
                                        {
                                        H[Cl][Cc]=0;
                                        }


                                Cc=Cc+1;
                                contador=0;
        }
        Cc=0;
        Cl=Cl+1;
        }
 }
```

Programa de Teste do Classificador com Modelo GSN Modificado

```
/*-------------------------------------------------------------------------*/
/*----------------------------Including Headers----------------------------*/

/*#include<conio.h>*/
#include<malloc.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>

/*-------------------------------------------------------------------------*/
/*-----------------------------------Defines-------------------------------*/

#define Ntotal          400/*Number of patterns that can be used*/
#define ctONE            3/*0x...0011*/
#define ctZERO           0/*0x...0000*/
#define ctUNDF           1/*0x...0001*/
#define ctEMPTY          0xaaaa/*See InputVM/EM*/
#define NExamples        100/*The ideal is 100*/
#define NClasses        2
#define NBits           2
/*#define NTrain         200/*Ideal is 160*/
/*#define NEval          400/*Ideal is 800*/
#define Max_Allocatade   260/*The total of memory allocated for neurons*/

/*-------------------------------------------------------------------------*/
/*---------------------------Declaring Types-------------------------------*/

typedef struct InternalType/*Data type to store behavior and adress of a neiron*/
{
 unsigned long x1, x2;/*64(32*2) bits used in the case of fan-in equal to 5.*/
}InternalType;


typedef struct tgsn/*Struct for the GSN 'till 5 inputs"*/
{
 InternalType bhv,adr;/*Behavior and Adress*/
 short level;/*Level of the neuron on the Tree(Leaf is 0(Zero))*/
 int sons;
 struct tgsn *b,*s;/*brother and son*/
}tgsn;


/*-------------------------------------------------------------------------*/
/*---------------------------Prototypes ----------------------------------*/

InternalType ITAND(InternalType a,InternalType b);
InternalType ITOR(InternalType a,InternalType b);
int ITNUNS(InternalType a);
InternalType ITShift(InternalType t1,int length,int direction);

void GsnTree(void);
void GsnTreeAux(tgsn *Tx);
unsigned int PositionEM(tgsn *Tx,unsigned int Es);
unsigned int InputEM(tgsn *Tx,short Ex);
unsigned int EvalEM(tgsn *Tx, short Ex);
unsigned int Which_Input(long bhv,short Ex);
unsigned int InputVM(tgsn *Tx,short Ex);
unsigned int EvalVM(tgsn *Tx, short Ex);
unsigned int PositionVM(tgsn *Tx,unsigned int Es);
tgsn *NewBrother(short Level,unsigned int num);
tgsn *NewSon(short Level,unsigned int num);
tgsn *NewNode(void);
void KillTree(void);
void Alloc(void);
void Store(tgsn *Tx,unsigned int D,short verificado);
unsigned int MelhorPosicao(tgsn *Tx,unsigned int D);
unsigned int FormatoEnt(InternalType Aux,short NF);
unsigned int Nrandom(short X);
```

```
void OutputResults(short rep);
void ValidaArmazena(void);
void Simula(void);

void Standard(void);
void Open_Files(void);

short Code[NClasses][NBits] =
{{3,0},{3,3}};
/*
{
{ 3,3,3,3,0,0,0,0,0,3,3,3,0,0,0,0},
{ 3,0,3,0,0,3,0,3,3,0,3,0,0,3,0,3},
{ 3,3,0,0,0,0,3,3,3,3,0,0,0,0,3,3},
{ 3,3,0,0,3,3,0,0,3,3,0,0,3,3,0,0},
{ 3,3,0,0,0,3,3,0,0,0,0,0,0,3,3,0},
{ 3,3,3,3,3,3,3,3,0,0,0,0,0,3,3,0},
{ 3,0,0,0,3,0,3,0,0,0,0,3,0,3,0,3},
{ 3,3,0,0,3,3,0,0,0,0,3,0,3,0,3,3},
{ 3,0,0,0,3,0,0,3,0,3,3,0,0,3,3,0},
{ 3,3,3,3,0,0,0,0,0,0,0,0,3,3,3,3}
};*/

short Sons = 0;
short ent_arv=0,arv_bit=0;
char Name[50];
FILE  *arq_ger;
FILE  *arq_sem;
tgsn *T;
short ResMatrix[Ntotal][NClasses];
short SeedsMatrix[15][11][3];
short Bit=0;
int Indice_Alocacao = 0;
short pixel=0;
unsigned short desejada=0;
long Indc_Aux;
char  InpMatrix[Ntotal][32];
InternalType MaskMatrix[4][5][4];
short Ones[256] = /* how many ones are there in a byte? */
{0,1,1,2,1,2,2,3,1,2,2,3,2,3,3,4,1,2,2,3,2,3,3,4,2,3,3,4,3,4,4,5,1,2,2,
 3,2,3,3,4,2,3,3,4,3,4,4,5,2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,1,2,2,3,2,3,
 3,4,2,3,3,4,3,4,4,5,2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,2,3,3,4,3,4,4,5,3,
 4,4,5,4,5,5,6,3,4,4,5,4,5,5,6,4,5,5,6,5,6,6,7,1,2,2,3,2,3,3,4,2,3,3,4,
 3,4,4,5,2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,
 6,3,4,4,5,4,5,5,6,4,5,5,6,5,6,6,7,2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,3,4,
 4,5,4,5,5,6,4,5,5,6,5,6,6,7,3,4,4,5,4,5,5,6,4,5,5,6,5,6,6,7,4,5,5,6,5,
 6,6,7,5,6,6,7,6,7,7,8};
tgsn *ptNeurons[Max_Allocatade]; /*points to the first element*/
short VAux[Max_Allocatade];
short  s1=0,s2=0,s3=0;
char Nome_ent[40];
int NTrain;
int NEval;


/*-------------------------------------------------*/
int main(int argc, char *argv[])
{
 /*clrscr();*/
 if(argc!=5)
            {
            printf("Uso correto: <aplicativo> <entrada> <saida> <NTrain> <Neval>");
            exit(1);
            }
 strcpy(Nome_ent,argv[1]);
 strcpy(Name,argv[2]);
 NTrain = atoi(argv[3]);
 NEval = atoi(argv[4]);
 Standard();
 return (0);
}

/*-----------------------------------------------------Standard*/
void Standard()
```

```
{
short i,j,k,q,cont=0;
short nent[4]={3,2,1,2};
short N_ent[4][3] = {{32,64,128},{27,81,0},{64,0,0},{25,125,0}};

short tree_level[4][3] = {{5,6,7},{3,4,0},{3,0,0},{2,3,0}};
short NArvBit [3]={3,5,11};
Open_Files();
Alloc();
for(Sons=2;Sons<6;Sons++)/*2->5*/
        {
        for(i=0;i<nent[Sons-2];i++)/*0->nent[Sons-2]*/
                {
                ent_arv = N_ent[Sons-2][i];
                for(j=0;j<3;j++)/*0->2 - Controle de piramides por bit*/
                        {
                        arq_ger = NULL;
                        arv_bit = NArvBit[j];
                        if (arq_sem != NULL)
                                arq_sem = freopen("seeds.dat", "r+b", arq_sem);
                        else
                                arq_sem = fopen("seeds.dat", "r+b");
                        if (arq_sem == NULL)
                                exit(1);
                        for(k=0;k<NExamples;k++)
                                {
                                memset(ResMatrix,0,sizeof(ResMatrix));
                                fread(SeedsMatrix, sizeof(SeedsMatrix), 1, arq_sem);
                                for(q=0;q<arv_bit;q++)
                                        {
                                        for(Bit=1;Bit<NBits;Bit++)
                                                {
                                                s1 = SeedsMatrix[Bit-1][q][0];
                                                s2 = SeedsMatrix[Bit-1][q][1];
                                                s3 = SeedsMatrix[Bit-1][q][2];
                                                T = NewSon(tree_level[Sons-2][i],1);
                                                GsnTree();
                                                ValidaArmazena();
                                                /*printf("\r%d",cont);*/
                                                Simula();
                                                Indice_Alocacao = 0;
                                                pixel = 0;
                                                cont++;
                                                }/*fim dos bits*/
                                        }/*fim das arvores*/
                                OutputResults(k);
                                cont = 0;
                                }/*fim dos exemplos*/
                        }/*fim da variacao do numero de arvores*/
                }/*fim da variacao de entradas por arvores*/
        }/*fim da variacao do numero de filhos*/
 KillTree();
}


/*-------------------------------------------------------------Simula*/

void  Simula()
{
short k=0, j=0;
unsigned int saida=0;
for(k = 0; k < NEval; k++)
        {  /*leaving false examples out!*/
         saida = EvalEM(T, k);
         for (j = 0; j < NClasses; j++)
                {
                        if (saida == 1)
                                ResMatrix[k][j]++;
                        else if (Code[j][Bit] == saida)
                                ResMatrix[k][j] += 2;
                }
        }
}
```

```
/*----------------------------------------------------------Valida Armazena*/

void ValidaArmazena(void)/*a = 0, sem expansao, 1, com expansao*/
{
short kn=0;
for (kn = 0; kn < NTrain; kn++)
        {
         desejada = Code[kn % NClasses][Bit];
         if((abs(EvalVM(T, kn) - desejada) <= 2))
                        Store(T, desejada,0);
        }
}


/*-----------------------------------------------------Output Results*/

void OutputResults(short rep)

{
int SucT,PtST,FaiT,CCS,CPS,CCF,K,J,I,Amax;
arq_ger = NULL;
CCS = 0;
CPS = 0;
CCF = 0;
for (K = 0; K < NEval; K++)
        {
         I = 0;
         Amax = 0;
         for (J = 0; J < NClasses; J++)
                        {
                         if (ResMatrix[K][J] > Amax)
                                {
                                 Amax = ResMatrix[K][J];
                                 I = 0;
                                }
                         else if (ResMatrix[K][J] == Amax)
                                 I++;
                        }
         if (I == 0)
                {
                        if (ResMatrix[K][K % NClasses] == Amax)
                                CCS++;
                        else
                                CCF++;
                }
         else
                        CPS++;
         if (K == NTrain-1)
                {
                        SucT = CCS;
                        PtST = CPS;
                        FaiT = CCF;
                        CCS=CPS=CCF=0;
                }
        }
if (arq_ger != NULL)
        arq_ger = freopen(Name, "a+", arq_ger);
else
        arq_ger = fopen(Name, "a+");
if (arq_ger == NULL)
         exit(1);
fprintf(arq_ger,"%4d%4d%4d%4d%4d%4d%4d%4d%4d%4ld\n",ent_arv,arv_bit,CCS,CPS,CCF,SucT,PtST,
         FaiT,Indc_Aux);
if (arq_ger != NULL)
         fclose(arq_ger);
arq_ger = NULL;
/*printf("%4d%4d%4d%4d%4d%4d%4d%4d%4d%4d%4d%4d\n",rep+1,ent_arv,arv_bit,CCS,CPS,CCF,SucT,PtST,
          FaiT,Indc_Aux);*/
```

```
if (arq_ger != NULL)
        fclose(arq_ger);
}



/*----------------------------------------------OPEN FILES*/

void Open_Files(void)

{
if (arq_ger != NULL)
        arq_ger = freopen(Nome_ent, "r+b", arq_ger);
else
        arq_ger = fopen(Nome_ent, "r+b");
if (arq_ger == NULL)
        {
        printf("\nError opening %s",Nome_ent);
        exit(1);
        }
fread(InpMatrix, sizeof(InpMatrix),1,arq_ger);
if (arq_ger != NULL) fclose(arq_ger);
        arq_ger = NULL;
if (arq_sem != NULL)
        arq_sem = freopen("seeds.dat", "r+b", arq_sem);

else
        arq_sem = fopen("seeds.dat", "r+b");
if (arq_sem == NULL)
        {
        printf("\nError opening seeds.dat");
        exit(1);
        }
if (arq_ger != NULL) fclose(arq_ger);
        arq_ger = NULL;
if (arq_ger != NULL)
        arq_ger = freopen("msktip.msk", "r+b", arq_ger);
else
        arq_ger = fopen("msktip.msk", "r+b");
if (arq_ger == NULL)
        {
        printf("\nError opening msktip.mask");
        exit(1);
        }

fread(MaskMatrix, sizeof(MaskMatrix), 1,arq_ger);

}


/*----------------------------------------------INPUTVM*/

unsigned int InputVM(tgsn *Tx,short Ex)

{
tgsn *Aux=NULL;
unsigned int Res;
Res = (EvalVM(Tx,Ex))|(ctEMPTY<<2);
Aux = Tx->b;
do
 {
  Res = (Res<<2)|(EvalVM(Aux,Ex));
  Aux = Aux->b;
 }
while (Aux != NULL);
return(Res);
}



/*----------------------------------------------EvalVM*/
```

```
unsigned int EvalVM(tgsn *Tx, short Ex)

{
if (Tx->s == NULL)
        return(Which_Input(Tx->bhv.x2,Ex));
else
        return(PositionVM(Tx,InputVM(Tx->s,Ex)));

}
/*-------------------------------------------------------------*/
/*-----------------------PosicaoVM-----------------------------*/


unsigned int PositionVM(tgsn *Tx,unsigned int Es)
{
unsigned int Res;
int i = 0;
Tx->adr.x1 = 0xffffffffL;
Tx->adr.x2 = 0xffffffffL;
do
 {
  Tx->adr = ITAND(Tx->adr,MaskMatrix[(Es) & (3)][i][Tx->sons-2]);
  Es = Es>>2;
  i++;
 }
while(((Es) & (3))!=(2));
if ((((Tx->bhv.x1) & (Tx->adr.x1)) == 0)
        &&
        (((Tx->bhv.x2) & (Tx->adr.x2)) == 0))/*Tx.a and Tx.b*/
                Res = 0;
else if((((Tx->bhv.x1) & (Tx->adr.x1)) == Tx->adr.x1)
                &&
                (((Tx->bhv.x2) & (Tx->adr.x2)) == Tx->adr.x2))/*Tx.a and Tx.b*/
                Res = 3;
else
                Res = 1;


return Res;
}


/*-------------------------------------------------------ITShift*/

InternalType ITShift(InternalType t1,int length,int direction)
{
InternalType t = {0,0};
unsigned long aux;
t = t1;
if(length>=32)
  {
        if(direction)/*right*/
         {
                t.x2 = t.x1;
                t.x1 = 0;
                t.x2 >>=length-32;
         }
        if(!direction)
         {
                t.x1 = t.x2;
                t.x2 = 0;
                t.x1 <<=length-32;
         }
  }
else if(length<32)
  {
        if(direction)/*right*/
         {
                aux = t.x1;
                t.x1 >>= length;
                t.x2 >>= length;
                aux  <<= 32-length;
                t.x2 |= aux;
```

```
            }
            if(!direction)/*left*/
            {
                        aux = t.x2;
                        t.x1 <<= length;
                        t.x2 <<= length;
                        aux >>= 32-length;
                        t.x1 |= aux;
            }
 }
return(t);
}
/*-----------------------------------------------------------ITAND*/

InternalType ITAND(InternalType a,InternalType b)
{
InternalType Res={0,0};
Res.x1 = ((a.x1) & (b.x1));
Res.x2 = ((a.x2) & (b.x2));
return(Res);
}


/*-----------------------------------------------------------ITOR*/
InternalType ITOR(InternalType a,InternalType b)
{
InternalType Res={0,0};
Res.x1 = ((a.x1) | (b.x1));
Res.x2 = ((a.x2) | (b.x2));
return(Res);
}


/*-----------------------------------------------------------ITNUNS*/

int ITNUNS(InternalType a)
{
short Res=0;
unsigned long aux=0;
aux = a.x1;
do
 {
            if ((aux%2)!=0)
                        Res++;
            aux /= 2;
 }
while(aux>=1);
aux = a.x2;
do
 {
            if ((aux%2)!=0)
                        Res++;
            aux /= 2;
 }
while(aux>=1);
return(Res);
}

/*-----------------------------------------------------------Store*/
void Store(tgsn *Tx,unsigned int D,short verificado)
{
unsigned int P;
tgsn *Aux=NULL;
if(Tx->s != NULL)
            {
            P = MelhorPosicao(Tx,D);
            Store(Tx->s,P,0);
                        if((Tx->b!=NULL)&&(!verificado))
                                    {
                                    Aux = Tx->b;
                                    do
```

```
                                        {
                                        D = D>>2;
                                        Store(Aux,D,1);
                                        Aux = Aux->b;
                                        }
                                while(Aux != NULL);
                                }
        }
}



/*-------------------------------------------------------Melhor Posicao*/

unsigned int MelhorPosicao(tgsn *Tx,unsigned int D)
{
int ent[] = {0x0,0x3  ,0xc  ,0xf  ,0x30 ,0x33 ,0x3c ,0x3f ,0xc0 ,0xc3,
                                        0xcc ,0xcf ,0xf0 ,0xf3 ,0xfc ,0xff ,0x300,0x303,0x30c,
                                        0x30f,0x330,0x333,0x33c,0x33f,0x3c0,0x3c3,0x3cc,
                                        0x3cf,0x3f0,0x3f3,0x3fc,0x3ff};
short C1,C0,CU;
int ENT_UM[32];
int ENT_ZERO[32];
int ENT_IND[32];
int i,Res,Aux;
int chosen;
InternalType AuxI;
memset(ENT_UM,0,sizeof(ENT_UM));
memset(ENT_ZERO,0,sizeof(ENT_ZERO));
memset(ENT_IND,0,sizeof(ENT_IND));

C1 = 0;
C0 = 0;
CU = 0;
AuxI = Tx->adr;
        for(i=0;i<(int)pow(2,Tx->sons);i++)
         {
                Res = PositionEM(Tx,(ent[i]|(2<<(Tx->sons*2))));
                if((Res==ctUNDF)&&(ITNUNS(ITAND(AuxI,Tx->adr))))
                 {
                 ENT_IND[CU] = ent[i];
                 CU++;
                 }
                if((Res==ctZERO)&&(ITNUNS(ITAND(AuxI,Tx->adr))))
                 {
                 ENT_ZERO[C0] = ent[i];
                 C0++;
                 }
                if((Res==ctONE)&&(ITNUNS(ITAND(AuxI,Tx->adr))))
                 {
                 ENT_UM[C1] = ent[i];
                 C1++;
                 }
         }
        if ((D&3)==3)
         {
                if(C1)
                 {
                 chosen = Nrandom(C1);
                 Aux = ENT_UM[chosen];/*escolhe usando C1*/
                 }
                else
                 {
                 chosen = Nrandom(CU);
                 Aux = ENT_IND[chosen];
                 Res = PositionEM(Tx,(Aux|(2<<(Tx->sons*2))));
                 Tx->bhv = ITOR(Tx->bhv,Tx->adr);
                 }/*escolhe usando CU*/
         }
        else
         {
                if(C0)
```

```
                    {
                     chosen = Nrandom(C0);
                     Aux = ENT_ZERO[chosen];/*escolhe usando C0*/
                     }
                    else
                     {
                     chosen = Nrandom(CU);
                     Aux = ENT_IND[chosen];
                     Res = PositionEM(Tx,Aux|(2<<(Tx->sons*2)));
                     Tx->adr.x1 = ~Tx->adr.x1;
                     Tx->adr.x2 = ~Tx->adr.x2;
                     Tx->bhv = ITAND(Tx->bhv,Tx->adr);
                     Tx->adr.x1 = ~Tx->adr.x1;
                     Tx->adr.x2 = ~Tx->adr.x2;
                     }

                    /*escolhe usando CU*/
          }
Res = PositionEM(Tx,Aux|(2<<(Tx->sons*2)));
return(FormatoEnt(Tx->adr,Tx->sons));
}


/*-------------------------------------------------------*/

unsigned int Nrandom(short X)

{ short Z, k;
 double RGenCte = 3.0899e-5;
 int x1=1204;
 int x2=3001;
 int x3=2653;
 k = x1 / 206;
 x1 = (x1 - k * 206) * 157 - k * 21;
 if (x1 < 0)
          x1 += 32363;
 k = x2 / 217;
 x2 = (x2 - k * 217) * 146 - k * 45;
 if (x2 < 0)
          x2 += 31727;
 k = x3 / 222;
 x3 = (x3 - k * 222) * 142 - k * 133;
 if (x3 < 0)
          x3 += 31657;
 Z = x1 - x2;
 if (Z > 706)
          Z -= 32362;
 Z += x3;
 if (Z < 1)
          Z += 32362;
 return (unsigned int)(Z * RGenCte * X);
}


/*-------------------------------------------------------Formato ent*/
unsigned int FormatoEnt(InternalType Aux,short NF)
{
int i,j;
unsigned int Form=0;
for(i=0;i<4;i+=3)
          {
          for(j=0;j<=NF-1;j++)
                    {
                     if((((Aux.x1) & (MaskMatrix[i][j][NF-2].x1)) == Aux.x1)&&(((Aux.x2) & (MaskMatrix[i][j][NF-2].x2))
== Aux.x2))
                              {
                              Form = Form|(i<<(2*(NF - 1 - j)));
                              }
                    }
          }
return(Form);
}
```

```
/*-------------------------------------------------------------ALLOC*/

void Alloc(void)

{
int i=0;
for(i=0;i<Max_Allocatade;i++)
        {
         ptNeurons[i] = (tgsn *)malloc(sizeof(tgsn));
        }
}




/*-------------------------------------------------------------New son*/


tgsn *NewSon(short Level,unsigned int num)
{
tgsn *neuron = NULL;
neuron = NewNode();
neuron->level = Level;
neuron->sons = Sons;
if(num >1)
        neuron->b = NewBrother(Level,num-1);
else if(num == 1)
        neuron->b = NULL;
if(!Level)
        neuron->s = NULL;
else
        neuron->s = NewSon(Level-1,Sons);
return (neuron);
}


/*-------------------------------------------------------------New Brother*/

tgsn *NewBrother(short Level,unsigned int num)
{
tgsn *neuron = NULL;
neuron = NewNode();
neuron->sons = Sons;
neuron->level = Level;
if(num>1)
        neuron->b = NewBrother(Level,num-1);
else if(num == 1)
        neuron->b = NULL;
if(!Level)
  neuron->s = NULL;
else
  neuron->s = NewSon(Level-1,Sons);
return (neuron);
}


/*-------------------------------------------------------------New Node*/

tgsn *NewNode(void)
{
tgsn *neuron = NULL;
if (Indice_Alocacao == Max_Allocatade)
        {
         printf("\nThe maximum has already been alocated");
         exit(1);
        }
neuron = ptNeurons[Indice_Alocacao];
neuron->bhv.x1 = 0x55555555L;
```

```
neuron->bhv.x2 = 0x55555555L;
neuron->level = 0;
Indice_Alocacao++;
return (neuron);
}


/*-----------------------------------------------------------------Kill Tree*/

void KillTree(void)
{
short i=0;
for(i=0;i<Max_Allocatade;i++)
        free(ptNeurons[i]);
}



/*-----------------------------------------------------------------GsnTree*/

void GsnTree(void)
{
short n,Z,k;
double RandCte = 256 * 3.0899e-5;
for (n = 0; n < Max_Allocatade; n++)
        { /* exclusive/portable random generator! */
        k = s1 / 206;
        s1 = (s1 - k * 206) * 157 - k * 21;
        if (s1 < 0)
                        s1 += 32363;
        k = s2 / 217;
        s2 = (s2 - k * 217) * 146 - k * 45;
        if (s2 < 0)
                        s2 += 31727;
        k = s3 / 222;
        s3 = (s3 - k * 222) * 142 - k * 133;
        if (s3 < 0)
                        s3 += 31657;
        Z = s1 - s2;
        if (Z > 706)
                        Z -= 32362;
        Z += s3;
        if (Z < 1)
                        Z += 32362;
        VAux[n] = (short)(Z * RandCte);
        /*printf("%6d",VAux[n]);*/
        }
GsnTreeAux(T);
}


/*-----------------------------------------------------------------GsnTreeAux*/
void GsnTreeAux(tgsn *Tx)
{
if(Tx->b != NULL)
        GsnTreeAux(Tx->b);
if(Tx->s != NULL)
        GsnTreeAux(Tx->s);
else if(Tx->s == NULL)
 {
        Tx->bhv.x2 = (unsigned long int)VAux[pixel];
        pixel++;
 }
}


/*-----------------------------------------------------------------Which_Input*/
unsigned int Which_Input(long bhv,short Ex)
{
unsigned char  MaskBit[8] = { 0x80, 0x40, 0x20, 0x10, 0x8, 0x4, 0x2, 0x1 };
```

```
if ((InpMatrix[Ex][(unsigned int)bhv/8] & MaskBit[(unsigned int)bhv & 7]) == 0)
        return ctZERO;
else
        return ctONE;
}


/*-----------------------------------------------------------------INPUTVM*/
unsigned int InputEM(tgsn *Tx,short Ex)
{
tgsn *Aux=NULL;
unsigned int Res;
Res = (EvalEM(Tx,Ex))|(ctEMPTY<<2);
Aux = Tx->b;
do
 {
 Res = (Res<<2)| (EvalEM(Aux,Ex));
 Aux = Aux->b;
 }
while (Aux != NULL);
return(Res);
}

/*-----------------------------------------------------------------EvalEM*/

unsigned int EvalEM(tgsn *Tx, short Ex)
{
if (Tx->s == NULL)
        return(Which_Input(Tx->bhv.x2,Ex));
else
        return(PositionEM(Tx,InputEM(Tx->s,Ex)));
}


/*-----------------------------------------------------------PositionEM*/

unsigned int PositionEM(tgsn *Tx,unsigned int Es)
{
unsigned int NA,N1;
int i = 0;
Tx->adr.x1 = 0xffffffffL;
Tx->adr.x2 = 0xffffffffL;
do
 {
 Tx->adr =ITAND(Tx->adr,MaskMatrix [(Es) & (3)][i][Tx->sons-2]);
 Es = Es>>2;
 i++;
 }
while(((Es) & (3))!=(2));
NA = ITNUNS(Tx->adr);/*NA/2 = NA>>1 no. de enderecados*/
N1 = ITNUNS(ITAND(Tx->bhv,Tx->adr)); /*N1 = no. de uns dos enderecados*/
if (N1 == (NA/2))
        return 1;
else if (N1>(NA/2))
        return 3;
else
        return 0;
}
```