UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Uma Metodologia para o Desenvolvimento de Aplicações de Visão Computacional utilizando um projeto conjunto de *Hardware* e *Software*

por

ROLF FREDI MOLZ

Tese submetida à avaliação, como requisito parcial
para a obtenção do grau de Doutor
em Ciência da Computação

Prof. Dr. Paulo M. Engel
Orientador

Prof. Dr. Fernando G. Moraes
Co-Orientador

Porto Alegre, setembro 2001.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Molz, Rolf Fredi

Uma Metodologia para o Desenvolvimento de Aplicações de Visão Computacional utilizando um projeto conjunto de *Hardware* e *Software* / por Rolf Fredi Molz. - Porto Alegre: PPGC da UFRGS, 2001.

80 f.: il.

Tese (Doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2001. Orientador: Engel, Paulo M. Co-Orientador: Moraes, Fernando G.

1. Visão computacional. 2. Computação configurável. 3. Implementação *hardware*/*software* 4. Redes neurais artificiais. I. Engel, Paulo M. II. Moraes, Fernando G. III. Título.

Agradecimentos

Ufa ! ! !

É inacreditável alcançar este momento tão importante na minha vida. Momento este que se iniciou com os meus estudos de Engenharia Elétrica na UFSM. Desde então, tive três grandes degraus a galgar : graduação, mestrado e doutorado.

Este último representa para mim uma grande vitória. Vitória essa que seria impossível sem a ajuda e o apoio de grandes pessoas que me auxiliaram, me auxiliam, e tenho a certeza de que sempre me auxiliarão.

A estas pessoas é que eu não posso deixar de agradecer neste momento.

Inicio esta seqüência de merecidos agradecimentos pela minha família. Agradeço a minha esposa Simoni pelo amor e paciência mantida durante este longo período. Agradeço a minha mãe e ao meu pai (*em memória*) pelo eterno amor e apoio ao estudo. Por fim, agradeço ao meu irmão pelo apoio e incentivo à conclusão deste doutorado.

Agradeço aos meus orientadores, Paulo Martins Engel e Fernando Gehm Moraes, que sempre confiaram no meu trabalho e na minha vontade de crescer. Agradeço ao Fernando Moraes pelo apoio durante o período em que ele também esteve na França e pelo incentivo no experimento dos diversos vinhos e queijos durante este período.

Agradeço também aos professores membros desta banca por terem aceitado a fazerem parte desta e assim fazem-me sentir honrado por tais presenças.

Agradeço a todos os funcionários do Instituto de Informática e da Biblioteca pelo esmero e disposição que sempre colocaram em seus serviços.

Para homenagear as pessoas que me auxiliaram durante o período em que estive na França segue um trecho em francês.

J’aimerai être reconnaissant pour l’aide fourni par: Jean-Marie DANDONNEAU et Laurent PRUNET dans le cadre de mon travail. J’aimerai aussi remercier les professeurs Michel ROBERT et Lionel TORRES pour leur soutien pendant l’exécution de ce travail.

Agradeço a Deus, sem o qual nada seria possível, pela vida, pela esperança e pelas bênçãos.

Por fim, agradeço a todas as pessoas que ajudaram, direta ou indiretamente, durante a realização deste doutorado.

A todos vocês o meu muito obrigado !

Sumário

Lista de Abreviaturas **6**
Lista de Figuras **7**
Lista de Tabelas **9**
Resumo **10**
Abstract **11**
1 Introdução **12**
1.1 Apresentação 12
1.2 Motivação e Escopo do Trabalho 12
1.3 Objetivos e Definição do Problema 12
1.4 Fundamentação Teórica 13
1.4.1 Dispositivos Configuráveis **13**
1.4.1.1 Fundamentação do Conceito de FPGA 14
1.4.1.2 Tipos de FPGAs 15
1.4.2 Visão Computacional **17**
1.5 Estado da Arte 19
1.6 Estrutura da Tese 23
2 Proposta de uma Metodologia para Implementação de Sistemas de Visão Computacional **25**
2.1 Introdução 25
2.2 A Metodologia Proposta 25
2.2.1 Pré-processamento **26**
2.2.2 Extração de Feições **33**
2.2.2.1 Zoning 33
2.2.2.2 Momentos Invariantes 34
2.2.2.3 Descritores de Fourier 35
2.2.3 Classificação **37**
2.3 Relacionamento entre o estado-da-arte e o problema 39
2.4 Conclusões 39
3 Implementação do Sistema Proposto **41**
3.1 Introdução 41
3.2 Ambientes de Prototipação 41
3.3 Partição do Sistema 43
3.4 Descrição Funcional do Sistema 45
3.4.1 Particionamento da Memória Externa **46**
3.4.2 Interface Core **47**
3.4.3 ADM Core **49**
3.4.3.1 Filtro Gaussiano 50
3.4.3.2 Direção/Detecção 50
3.4.4 Lines Core **51**
3.4.5 Neural Core **52**
3.4.6 DSP **56**
3.5 Conclusões 57
4 Resultados **58**

4.1 Introdução 58
4.2 Filtro Gaussiano 58
4.3 Resultados do Processo de Classificação 60
4.3.1 Reconhecimento baseado em palavras 61
4.3.2 Reconhecimento baseado em letras 62
4.3.2.1 Extração de Feições e Classificação 66
4.3.2.1.1 Método dos Descritores de Fourier 69
4.4 Conclusões 70
5 Conclusões 72
Anexo 1 Publicações 75
Bibliografia 76

Lista de Abreviaturas

AR – Auto-Regressivo
ART– Adaptive Resonance Theory
ASIC – Application Specific Integrated Circuit
CAD– Computer Aided Design
CLB – Configurable Logic Block
DCT – Discret Cosine Transform
DFT – Discret Fourier Transform
DMA – Direct Memory Acess
DSP – Digital Signal Processor
FFT – Fast Fourier Trasnform
FIFO – First In First Out
FPGA – Field Programmable Gate Array
GPP – General Purpose Processor
LIRMM – Logic Inside Reconfigurable Micro Machine
MLP – Perceptrom de Múltiplas Camadas (*Multi-Layer Perceptron*)
Pixel – Elemento de Imagem (*Picture Element*)
RAM – Random Access Memory
RBF – Função de Base Radial (*Radial Basis Function*)
RNA – Rede Neural Artificial
SIFT – Scale Invariant Feature Transform
ULA – Unidade Lógica e Aritmética

Lista de Figuras

FIGURA 1.1 – Esquema de uma unidade funcional utilizando LUT. 15
FIGURA 1.2 – Esquema de uma unidade funcional baseada em multiplexadores. 16
FIGURA 1.3 – Estrutura conceitual de um FPGA. 17
FIGURA 1.4 – Hierarquia das tarefas computacionais em visão computacional. 19
FIGURA 2.1 – Diagrama geral dos blocos do sistema proposto. 26
FIGURA 2.2 - Tarefas da etapa de pré-processamento. 27
FIGURA 2.3 – Máscara utilizada para a determinação da direção. 29
FIGURA 2.4 – (a)Exemplo do cálculo da direção. (b)Janela de dimensão 3x3. 29
FIGURA 2.5 – Processo de detecção de borda 30
FIGURA 2.6 – Linhas obtidas usando o algoritmo de localização de end-point com restrições. 32
FIGURA 2.7 – Formas retangulares obtidas. 32
FIGURA 2.8 – Zoning de um caractere em níveis de cinza. (a) Uma grade 4x3 superposta. (b) Média do nível de cinza de cada zona. 33
FIGURA 2.9 – A borda detectada de um dado caractere. Os pontos (x0, y0) e (x1, y1) são (arbitrariamente) os dois primeiros pontos da seqüência. 36
FIGURA 3.1 – Placa de Prototipação LIRMM. 42
FIGURA 3.2 – Ambiente de Prototipação APTIX. 43
FIGURA 3.3 – Partição proposta entre *hardware* e *software* para o sistema. 44
FIGURA 3.4- (a)Diagrama em blocos do sistema. (b)Fluxo de dados/informações. 46
FIGURA 3.5- Diagrama do particionamento da memória externa. 47
FIGURA 3.6- Programa que envia uma determinada parte da imagem ao FPGA. 49
FIGURA 3.7 – Modelo do neurônio adotado. 52
FIGURA 3.8 – Função de ativação. 54
FIGURA 3.9 – Diagrama em blocos do neurônio utilizado para o processo de classificação. 55
FIGURA 3.10 – Demonstração da técnica de *bit-slice*. 55
FIGURA 3.11 – Bloco *Neural Core*. 56
FIGURA 4.1 – Imagem Original (a) e Imagem Filtrada em *Software* sem Adequações (b). 58
FIGURA 4.2 – Imagens Filtradas em *Hardware* (a) e em *Software* com Adequações (b). 59
FIGURA 4.3 – Histograma da diferença entre Imagens Filtradas por *Software* com e sem adequação. 60
FIGURA 4.4 – Histograma da diferença entre Imagens Filtradas por *Software* sem adequação e por *Hardware*. 60
FIGURA 4.5 – Letras sintéticas binárias utilizadas no treinamento da RNA. 62
FIGURA 4.6 – Imagens a serem reconhecidas. 64
FIGURA 4.7 – Novo diagrama geral de blocos. 64
FIGURA 4.8 – Exemplo de separação dos caracteres. 65

FIGURA 4.9 – Exemplo de ajustes efetuados nos caracteres. 66
FIGURA 4.10 – Exemplo da extração de contorno utilizada pelos Descritores de Fourier.....69
FIGURA 5.1 – Provável versão SoC.....73

Lista de Tabelas

TABELA 1 – Resumo do Estado-da-arte. 24
TABELA 2 - Coeficientes de ponderação *w(p,q).* 28
TABELA 3 - Diferenças e semelhanças. 39
TABELA 4 – Resultados comparativos para a propagação neuronal..... 45
TABELA 5 – Resultados comparativos para a filtragem gaussiana. 45
TABELA 6 – Comandos disponíveis (DSP - FPGA). 48
TABELA 7 – Comandos internos. 49
TABELA 8 - Coeficientes de ponderação *w(i,j)* adequados. 50
TABELA 9 – Codificação adotada para os padrões de entrada e função de ativação. ... 53
TABELA 10- Resultados obtidos. 61
TABELA 11- Resultados obtidos. 61
TABELA 12- Resultados obtidos. 62
TABELA 13 – Resultados do reconhecimento de letras com o método Zoning com máscara 4x3. 67
TABELA 14 – Resultados do reconhecimento de letras com o método Zoning com máscara 4x4. 67
TABELA 15 – Resultados do reconhecimento de letras com a utilização de 4 momentos invariantes. 68
TABELA 16 – Resultados do reconhecimento de letras com a utilização de 7 momentos invariantes. 68
TABELA 17 – Resultados do reconhecimento de letras com a utilização de 5 descritores..... 69
TABELA 18 – Resultados do reconhecimento de letras com a utilização de 6 descritores..... 69
TABELA 19 – Resultados do reconhecimento de letras com a utilização de 10 descritores..... 70

Resumo

As tarefas de visão computacional incentivam uma significativa parte da pesquisa em todas as áreas científicas e industriais, entre as quais, cita-se a área voltada para o desenvolvimento de arquiteturas de computadores. A visão computacional é considerada um dos problemas mais desafiadores para a computação de alto desempenho, pois esta requer um grande desempenho, bem como um alto grau de flexibilidade. A flexibilidade é necessária pois a visão computacional abrange aplicações em que há diferentes tarefas a serem realizadas com diferentes necessidades de desempenho. Esta flexibilidade é particularmente importante em sistemas destinados a atuar como ambientes experimentais para novas técnicas de processamento visual ou para a prototipação de novas aplicações.

Computação configurável tem demonstrado, por meio de exemplos implementados pela comunidade científica, fornecer uma boa relação entre alto desempenho e flexibilidade necessária para a implementação de diferentes técnicas utilizadas na área de visão computacional. Contudo, poucos esforços de pesquisa têm sido realizados na concepção de sistemas completos visando a solução de um problema de visão computacional, incluindo ambos os requisitos de *software* e de *hardware*.

O principal objetivo deste trabalho é mostrar que as técnicas e tecnologias disponíveis na área de computação configurável podem ser empregadas para a concepção de um sistema capaz de implementar um grande número de aplicações da área de visão computacional na pesquisa e no ambiente industrial. Entretanto, não é escopo deste trabalho implementar um sistema de computação que seja suficiente para abordar os requerimentos necessários para todas as aplicações em visão computacional, mas os métodos aqui introduzidos podem ser utilizados como uma base geral de implementação de várias tarefas de visão computacional.

Este trabalho utiliza ambientes que permitem implementações conjuntas de *hardware* e *software*, pois os mesmos facilitam a validação das técnicas aqui apresentadas, por meio da implementação de um estudo de caso, sendo parte deste estudo de caso implementado em *software* e outra parte em *hardware*.

Palavras-chave: visão computacional, processamento de imagens, computação configurável, implementação *hardware*/*software*, redes neurais artificiais, FPGA.

TITLE: “A Methodology for Computer Vision Applications Using Hardware and Software Partition.”

Abstract

Computer vision tasks have motivated a significant part of the research in many scientific and industrial areas. Among these, development of advanced computer architectures. In addition, the computer vision is considered one of the most challenging problems for high performance computing because it requires a high computational power as well as a high degree of flexibility. Flexibility is needed because computer vision embraces applications where processes with different computing requirements are combined. This flexibility is particularly important in systems intended to act as experimental environments for testing new techniques for computer vision or prototyping new applications.

Configurable computing has proven to be a good trade-off between the high performance and the flexibility required for implementing different techniques used in computer vision. Nevertheless, few research efforts have been carried out in conceiving complete systems aimed at implementing vision applications, including both the required hardware elements and the software environments.

The goal of this work is to show that the techniques and the available technologies in the configurable computing area can be used for the conception of a system capable of mapping a great number of computer vision applications in research and industrial areas. However, this work does not have as goal to implement a computation system that is enough for all applications in computer vision. The methods introduced herein can be used as a general base for implementation of several tasks of computer vision.

This work uses platforms that allow hardware/software implementations. These platforms facilitate the validation of the techniques presented. This work presents a case study, being part of this study implemented in software and part in hardware.

Keywords: computer vision, image processing, configurable computing, *hardware*/*software* implementation, artificial neural network, FPGA.

1 Introdução

1.1 Apresentação

Visão tem sido um foco de atenção para pesquisadores desde o início da computação, pois este é um dos mais notáveis sistemas de percepção dos seres humanos. A pesquisa no campo de emulação das capacidades visuais, através do uso de computadores, estende-se até os dias de hoje e por isto tornou-se uma área de aplicação que compreende áreas como automação industrial, robótica e processamento de documentos.

Estas tarefas de visão são implementadas em computadores por meio de diferentes técnicas estudadas em disciplinas como: processamento de imagens, classificação de padrões e inteligência artificial. Além disto, as tarefas de visão computacional incentivam uma significativa parte da pesquisa em todas estas áreas citadas bem como em áreas voltadas para o desenvolvimento de arquiteturas de computadores e sensoriamento remoto. Por exemplo, visão computacional tem sido utilizada como um meio de validação para aplicações em processamento paralelo, e é considerada um dos problemas mais desafiadores para a computação de alto desempenho, pois esta requer um grande desempenho bem como um alto grau de flexibilidade. Flexibilidade é necessária pois a visão computacional abrange aplicações em que existem diferentes tarefas a serem realizadas com diferentes necessidades de desempenho. Esta flexibilidade é particularmente importante em sistemas destinados a atuar como ambientes experimentais para novas técnicas de processamento visual ou para a prototipação de novas aplicações.

1.2 Motivação e Escopo do Trabalho

A utilização da computação configurável tem demonstrado na prática fornecer uma boa relação entre alto desempenho e flexibilidade necessária para a implementação de diferentes técnicas utilizadas para a área de visão computacional [VIL97].

Além disto, a comunidade de pesquisa de computação configurável tem validado diferentes exemplos de tarefas voltadas para o processamento de imagens como aplicações para as pesquisas desenvolvidas desde o aparecimento dos primeiros dispositivos lógicos programáveis. Contudo, existem poucas publicações de pesquisa voltadas para a concepção de sistemas completos visando a solução de um problema de visão computacional, incluindo ambas as necessidades de *software* e de *hardware*. Por outro lado, muito do desenvolvimento da pesquisa de visão computacional é ainda realizada por ambientes experimentais baseados somente em *software* [MIN98]. Estes ambientes são utilizados tanto para o teste de novas técnicas bem como para a prototipação de sistemas completos, mas apresentam deficiências em relação ao desempenho necessário para aplicações que visem tempo-real.

1.3 Objetivos e Definição do Problema

Como salientado anteriormente, existe uma gama de áreas de aplicações que empregam técnicas de visão computacional. Embora acredite-se que nunca um sistema de computação será suficiente para preencher os requisitos necessários para todas as

aplicações em visão computacional, a metodologia aqui introduzida pode ser utilizada como uma base geral de implementação de várias tarefas de visão computacional.

Neste trabalho será proposta uma metodologia para o desenvolvimento de aplicações de visão computacional utilizando uma partição *hardware*/*software*, tanto como um ambiente experimental para a pesquisa em visão computacional, como também uma ferramenta de prototipação para aplicações objetivando alto desempenho (tempo-real).

Como um estudo de caso, este trabalho apresenta uma solução para a localização e classificação de placas que possuem um formato retangular. Este problema escolhido serve como base de estudo para o desenvolvimento de um sistema capaz de localizar e classificar placas de trânsito visando com isto auxiliar a tomada de decisões de um motorista de automóvel, por exemplo.

Como objetivos principais desta tese ressalta-se a proposta de uma metodologia completa para a solução do problema acima mencionado, bem como a portabilidade e o reuso das técnicas implementadas neste sistema, viabilizando outros requisitos de *software* e *hardware*. Esta portabilidade é caracterizada pela forma da descrição do sistema, sendo esta realizada nas linguagens C e VHDL. O reuso destas técnicas se dá através da possibilidade de se utilizar todos os métodos ou parte destes em outras tarefas de visão computacional. Com base neste reuso inicia-se o desenvolvimento de uma biblioteca de funções descritas em *hardware* visando solucionar problemas de visão computacional que visem alto desempenho.

Como um objetivo secundário, esta tese apresenta a implementação de Rede Neural Artificial (RNA) em *hardware* digital, demonstrando a potencialidade da mesma e o ganho de desempenho sobre implementações realizadas somente em *software.*

1.4 Fundamentação Teórica

Esta seção apresenta uma breve fundamentação teórica sobre os temas preponderantes neste trabalho. São abordados conceitos fundamentais sobre os dispositivos programáveis, onde é ressaltado principalmente o FPGA. A seguir, é apresentado o tema visão computacional. É importante salientar que a visão computacional é um tópico mais abrangente que o assunto aqui descrito, mas este trabalho aborda funções básicas que todo sistema de visão computacional deve dispor em uma de suas etapas.

1.4.1 Dispositivos Configuráveis

Nesta seção abordaremos alguns tópicos referentes a dispositivos configuráveis ou, como também são conhecidos, dispositivos reprogramáveis. Dentre os dispositivos existentes, este trabalho procurou dar ênfase ao dispositivo denominado de FPGA, *Field Programmable Gate Array*.

Este dispositivo tem seu lugar na contínua evolução da tecnologia de circuitos integrados de escala muito ampla (VLSI) com respeito a circuitos mais densos e mais rápidos. O FPGA, para muitas aplicações, já fornece um número adequado de transistores em um único circuito integrado.

O FPGA é um tipo relativamente novo de componente para a concepção de sistemas eletrônicos, particularmente aqueles que se utilizam dos princípios de implementação

digital. Ele consiste de uma matriz de blocos configuráveis e de uma rede de interconexão configurável, e como o nome sugere, esta configuração pode ser determinada no campo, isto é, pelo usuário final. A função específica de cada bloco e as conexões entre os blocos são definidas pelo usuário.

Pode-se notar que a habilidade de reconfigurar um componente eletrônico, seja para corrigir um erro ou para alterá-lo para outra aplicação, tem benefícios econômicos, mas como é sugerido em [OLD95], o impacto do conceito de FPGA é muito mais profundo, visto que, um sistema lógico pode ser dinamicamente reconfigurado de modo a se adequar a uma nova situação. Isto é um novo paradigma na computação, cujo potencial de aplicação está cada vez mais amplo.

1.4.1.1 Fundamentação do Conceito de FPGA

Durante os anos 60, os sistemas digitais eram construídos através do uso de pequenos circuitos integrados e transistores interconectados em uma placa de circuito impresso. Estes componentes com lógica transistor a transistor (TTL) eram utilizados na maioria dos projetos digitais. Isto significa que era comercialmente atraente fornecer uma grande variedade de tais componentes que fossem compatíveis eletricamente e que pudessem ser conectados entre si para gerar um determinado sistema. Entretanto, com a evolução tecnológica, tiveram que surgir componentes que não possuíam uma grande venda em todos os níveis comerciais, o que levou os produtores destes componentes a descontinuarem a produção destes em um nível LSI.

Novas estruturas foram necessárias para se utilizar a vantagem da tecnologia VLSI, onde ressaltam-se três estruturas [OLD95]:

1. Microprocessadores / Memórias: microprocessadores e memórias são atraentes aos produtores de componentes, visto que estes podem ser utilizados para sistemas programáveis resultando em vendas com alto volume. Entretanto, microprocessadores não são aconselhados para aplicações que necessitam de computação rápida de funções simples.

2. Memórias Programáveis Somente de Leitura (PROM) e Matrizes de Lógica Programável (PAL): qualquer função lógica combinacional com um número fixo e limitado de entradas e de saídas pode ser implementada como uma tabela de consulta, mais conhecida como *lookup table*, em uma PROM.

3. Circuitos integrados de aplicação específica (ASIC): estes circuitos, como o próprio nome salienta, são específicos para cada aplicação, o que levou os fabricantes a desenvolver bibliotecas básicas que pudessem ser interligadas como em um projeto realizado através do uso de uma placa de circuito impresso.

A tecnologia do FPGA pode ser vista como uma extensão lógica destas três estruturas. As três principais classes de FPGAs são as seguintes [OLD95]:

1. *Computational Logic Arrays*: estes dispositivos estendem o paradigma processador / memória pela implementação de algoritmos a nível de portas em uma estrutura reprogramável.

2. *Structured PALs*: estes dispositivos estendem o paradigma PROM / PAL pelo uso da densidade da tecnologia VLSI para criar um dispositivo de propósito mais geral capaz de implementar a funcionalidade de muitas PALs simples interconectadas em uma placa de circuito impresso.

3. *Channeled FPGAs*: estes dispositivos estendem o paradigma ASIC pela produção de uma estrutura programável em campo com capacidades similares e com uma interface com o usuário. Além disto, tal tecnologia evita o alto custo da implementação e também o tempo de ciclo do projeto requerido para um componente fabricado em tecnologia dedicada.

Estas três famílias de dispositivos são sintetizadas com o auxílio de *softwares* específicos e a despeito de suas diferentes filosofias, há uma forte similaridade entre elas e, também, uma clara classe de produtos configuráveis.

1.4.1.2 Tipos de FPGAs

Há vários tipos de FPGAs que se diferenciam pelas suas unidades funcionais, as quais são baseadas em muitas técnicas. Dentre elas, temos:

RAM *lookup tables (LUT)*: nesta estrutura as variáveis de entrada são utilizadas para selecionar um determinado valor de uma memória RAM que possui valores pré-carregados representando a saída da tabela verdade da função a ser implementada. Um exemplo pode ser visto na FIGURA 1.1.

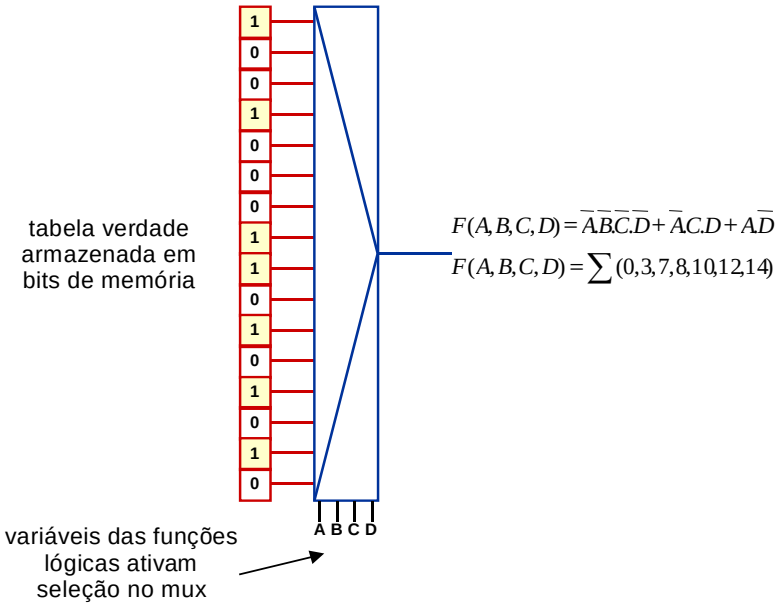


FIGURA 1.1 – Estrutura de uma unidade funcional utilizando LUT.

Baseado em Multiplexador: este estilo de unidade funcional está baseado no fato de que todas as funções de duas variáveis de entrada podem ser implementadas por um simples multiplexador dois para um (2:1) [BET99], FIGURA 1.2. Estas unidades funcionais implementam o que se denomina de Gerador Universal de Funções Lógicas (ULG).

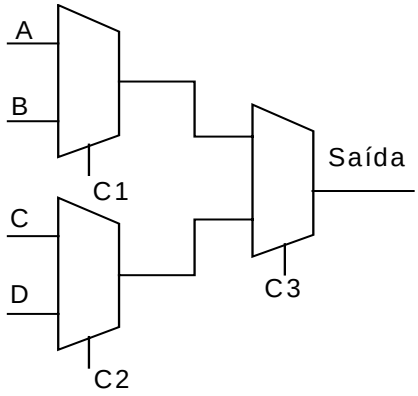


FIGURA 1.2 – Esquema de uma unidade funcional baseada em multiplexadores.

Função Fixa: a unidade funcional fornece uma função fixa simples. Este tipo de unidade funcional possui como vantagem a simplicidade e um atraso pequeno por estágio. Porém, como desvantagem, tem-se o grande número de unidades funcionais que devem ser interligadas para se obter uma função descrita pelo usuário, trazendo com isto problemas de roteamento.

Atualmente, podem ser encontrados no mercado três tipos básicos de FPGAs classificados de acordo com a tecnologia empregada na programabilidade: SRAM, EPROM ou EEPROM. Os FPGAs baseados em SRAM têm programação volátil e portanto devem ser reprogramados a cada vez que o sistema for alimentado [ADA97].

Conceitualmente, um dispositivo do tipo FPGA reprogramável é uma estrutura simples, FIGURA 1.3. Este dispositivo possui uma área de armazenamento de controle na forma de uma memória RAM estática (SRAM). Unidades funcionais, representadas pelos retângulos, e fios, representados pelo conjunto de linhas no sentido horizontal e vertical, são programados pela escrita de um determinado código na área de armazenamento de controle. As famílias de FPGA são diferenciadas pelas suas arquiteturas internas, pelas granularidades com relação às unidades funcionais e pela organização do roteamento interno.

Como os atuais circuitos são definidos pelo conteúdo de uma área de armazenamento de controle volátil, as aplicações utilizando FPGA possuem as seguintes características:

1. **Implementação Rápida**: como a área de armazenamento de controle programa a fiação necessária para a interconexão entre os blocos funcionais, o tempo de implementação de uma dada aplicação é curto, da ordem de milisegundos. Isto é uma clara forma de distinção em relação aos projetos convencionais realizados em VLSI, onde decorrem muitas semanas entre o projeto e a implementação real. Esta característica cria oportunidade para novas metodologias de projeto.
2. **Reconfiguração Dinâmica**: com algumas arquiteturas de FPGA pode ser realizada a reprogramação em tempo de execução. Portanto, um mesmo FPGA pode ser reutilizado para diferentes funções em tempo de execução. Estes circuitos podem, em princípio, se auto modificarem.
3. **Segurança de Projeto**: Uma configuração de FPGA se desfaz quando o chip é desligado. Há uma faixa de aplicações onde este nível adicional de segurança do projeto é importante.

4. **Programabilidade em campo**: sistemas construídos com o uso de FPGAs podem ser atualizados no campo, seja por um operador ou até mesmo remotamente, para corrigir possíveis erros ou adicionar novas funcionalidades.

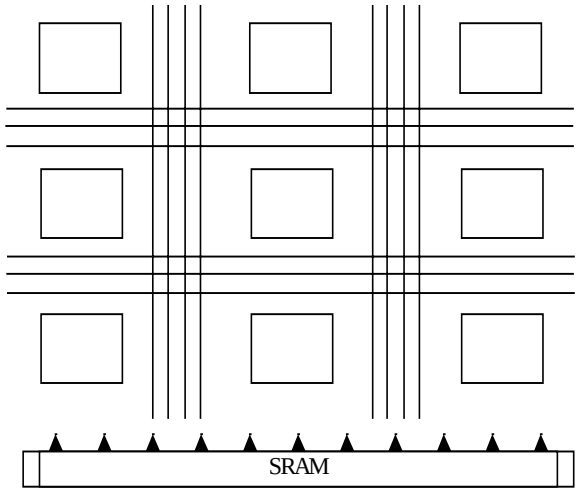


FIGURA 1.3 – Estrutura conceitual de um FPGA.

1.4.2 Visão Computacional

O sentido da visão pode ser considerado o meio mais eficiente que o ser humano dispõe para captar as informações originadas no ambiente que o cerca. Algumas aplicações de imagens e alguns tipos de imagens requerem uma interação visual bastante intensa. A capacidade humana para processar e interpretar imensas quantidades de dados de natureza visual motiva o desenvolvimento de técnicas e dispositivos, de modo a estender esta sua capacidade e sensibilidade ainda mais. O conhecimento de como reage o sistema visual humano e de algumas técnicas disponíveis para melhor adequar a imagem à aplicação são importantes para poder explorar mais eficientemente os recursos de sistemas de processamento de imagens. A utilização da imagem na forma digital torna possível o seu processamento computacional, aumentando sua qualidade.

O interesse em métodos de Processamento Digital de Imagens surgiu, principalmente, da necessidade de melhorar a qualidade da informação pictorial para interpretação humana. Uma das primeiras aplicações das técnicas de processamento digital foi a melhoria de ilustrações de jornais enviados por cabo submarino entre Londres e New York por volta de 1920 [GON92]. Mas as técnicas de processamento digital de imagens evoluíram em meados dos anos 60 com o advento de computadores digitais e com o programa espacial norte-americano. Em 1964 as imagens da lua transmitidas pela sonda Ranger 7 foram processadas por um computador para corrigir vários tipos de distorções inerentes à câmara de TV a bordo. As técnicas de processamento usadas nesta época serviram de base para o realce e restauração de imagens de outros programas espaciais posteriores, como as expedições tripuladas da série Apollo, por exemplo.

Adotando-se como uma evolução do processamento de imagens tem-se a Visão Computacional. Esta área, bem como a área de processamento de imagens, aborda os assuntos algorítmicos e computacionais associados com a aquisição, processamento e entendimento da imagem, que incorporam os princípios que delineiam as capacidades visuais [ALO91]. Atribui-se como diferença básica entre processamento de imagens e visão computacional o fato de que a primeira é uma tarefa de mais baixo nível, além de

que, a segunda está relacionada com a percepção da imagem enquanto que a primeira é uma tarefa que possibilita esta percepção.

Historicamente, a visão computacional tem sido uma ciência experimental, onde as soluções e pesquisas desenvolvidas para aplicações específicas foram obtidas por métodos de tentativa e erro. Entretanto, a tendência das últimas duas décadas foi no fornecimento de bases teóricas para vários tópicos importantes desta área.

A primeira metodologia completa para a visão computacional foi proposta por David Marr em 1982, no livro *"Vision"* [MAR82]. Marr definiu visão como *"um processo que produz uma descrição, a partir de imagens do mundo externo, que é útil ao visualizador e não repleta de informações irrelevantes*". A palavra "processo" refere-se ao mapeamento das diferentes representações de uma cena, presente no mundo externo, obtidas a partir das matrizes dos valores de intensidade de brilho aos diferentes padrões que descrevem esta cena. As matrizes são obtidas nos primeiros estágios da visão computacional, enquanto que os diferentes padrões são obtidos nos últimos estágios da mesma.

O processo desta definição de visão é comumentemente implementado como um encadeamento de diferentes tarefas no sistema. Na FIGURA 1.4 estas tarefas estão divididas em três classes de acordo com o seu propósito [TRI89], [CHO92], [GON92].

As tarefas do processamento de ***Baixo Nível*** manipulam funções para propósitos como: condicionamento de imagens (remoção de ruídos, melhoramento de contraste, etc.) ou extração de suas características (bordas, regiões, etc.). Estas tarefas resultam no particionamento (segmentação) da imagem em suas partes significativas, as quais correspondem às linhas ou objetos existentes nesta cena.

O processamento pertencente ao ***Nível Intermediário*** está principalmente interessado com a análise das feições (características) obtidas no baixo nível objetivando produzir dados simbólicos necessários ao processamento de alto nível. Dados simbólicos incluem a representação das bordas e linhas. O processamento neste nível tenta construir uma união destas bordas e linhas para a extração de entidades significativas, como por exemplo, formar retângulos a partir de linhas.

O processamento de ***Alto Nível*** envolve a interpretação final da cena, objetivando o reconhecimento dos objetos e a análise das estruturas relacionais. Para tanto, é comum utilizar-se um paradigma baseado em modelo. Por exemplo, um modelo de um carro pode possuir as descrições das rodas, portas, etc., e as restrições descrevendo seus relacionamentos. O processamento de alto nível também possui o controle das interfaces entre o sistema e seu ambiente.

Estes três níveis não são fixos, havendo uma barreira virtual que depende muito da aplicação final. Por exemplo, o processamento de nível intermediário pode conter a etapa de extração de feições para um nível superior (processamento de alto nível) realizar a classificação dos vetores de características calculados no nível anterior.

Como pode ser visto na FIGURA 1.4, a visão computacional é composta por uma série de estágios que manipulam diferentes tipos de dados (pixels, números em ponto fixo ou ponto-flutuante) de diferentes tamanhos (resolução em quantidade de bits).

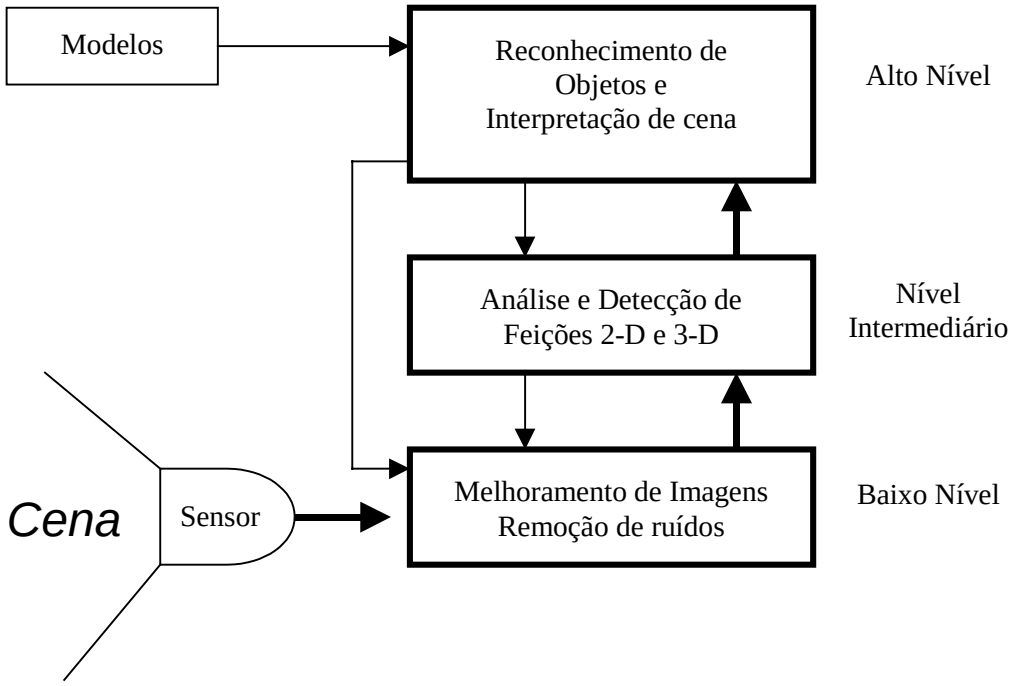


FIGURA 1.4 – Hierarquia das tarefas computacionais em visão computacional.

1.5 Estado da Arte

Nesta seção, apresentamos uma descrição sucinta de alguns trabalhos relevantes envolvendo sistemas de visão computacional para localização e/ou classificação de objetos. Estes trabalhos foram escolhidos por representarem o estado da arte na área, além de apresentarem detalhes sobre a implementação e o desempenho alcançado, que permitem uma comparação, mesmo que qualitativa, com a solução proposta nesta tese.

O trabalho apresentado em [GAV99] é um sistema instalado em um carro para a detecção de sinais de trânsito ou de pedestres. O algoritmo de detecção é baseado no cálculo da Transformada de Distância e de um algoritmo de busca hierárquica. O reconhecimento é realizado por um pré-treinamento de um clusterizador do tipo K-médias. O trabalho contempla como uma última etapa uma rede neural do tipo RBF para verificação e conseqüente diminuição do número de respostas falsas positivas. As imagens utilizadas para teste são imagens em 256 tons de cinza com um tamanho de 360x288 pixels. Esta implementação foi realizada em um Pentium dual MMX 450MHz e possui um desempenho entre 10 a 15 quadros por segundo para o reconhecimento de placas. Para o reconhecimento de pedestres o desempenho cai para o intervalo entre 1 a 5 quadros por segundo. Por fim, este sistema perde um pouco o desempenho na presença de objetos com oclusão.

O artigo de [SIM98] também aborda a detecção e reconhecimento de objetos. Porém, este trabalho contempla o reconhecimento de objetos com algum grau de oclusão. O algoritmo realiza uma etapa de pré-processamento com: segmentação da imagem, delineamento do contorno, suavização do contorno e por fim uma aproximação em polígonos. A etapa de reconhecimento é realizada por mais 6 sub-etapas, onde em uma destas, há a utilização de "m" redes neurais do tipo MLP (*Multi-layer Perceptron*). O valor de “m” é o número de objetos diferentes na base de conhecimento. Cada MLP

possui uma camada de entrada com 2 neurônios, uma camada escondida com 30 neurônios e uma camada de saída com X neurônios, onde X é o número de feições a serem consideradas no objeto. As imagens utilizadas para teste são imagens em 256 tons de cinza com um tamanho de 640x512 pixels. O desempenho obtido é de uma imagem em menos de 3 minutos. Toda a implementação foi realizada em *software* e o maior custo de tempo no algoritmo se dá na etapa de pré-processamento.

Em [TSA99] o autor aborda a detecção e reconhecimento de objetos com um certo grau de oclusão. A etapa de pré-processamento é responsável pela detecção de pontos principais. Após há a geração de modelos Auto-Regressivos (AR) ortogonais e o reconhecimento é realizado pela análise da distância euclidiana entre o modelo gerado e os modelos AR pertencentes aos padrões pré-armazenados. O trabalho não aborda o tipo de implementação, o ambiente de implementação e nem o desempenho obtido.

O trabalho apresentado em [FOR99] realiza a detecção de objetos após a supressão do fundo da cena (*background*), suposto conhecido e que deve ser atualizado caso a implementação seja em ambiente aberto. Esta detecção é realizada com a esqueletização estatística. A etapa de reconhecimento é através da comparação do objeto esqueletizado com os modelos previamente armazenados. Por fim, este trabalho também realiza a perseguição ao modelo localizado. A implementação foi idealizada completamente em *software*, onde a plataforma utilizada foi um Pentium II 300MHz. O desempenho obtido é em média 3 quadros por segundo. Este desempenho é para toda a tarefa; contudo, há um percentual de 75% deste tempo, destinado para a operação de extração de feições e para o reconhecimento. A extração de feições é realizada pela operação de esqueletização.

Em [LOW99], temos o reconhecimento de objetos com um certo grau de oclusão. A detecção é realizada por meio de uma estrutura piramidal em dois níveis, onde cada estágio desta pirâmide é formado por transformações na imagem de entrada. Um primeiro nível é formado pela diferença entre gaussianas e o outro nível é formado pela interpolação. A partir desta pirâmide calculam-se pontos máximos e mínimos, estes por sua vez, denominados pelo autor como SIFT. O reconhecimento é realizado por meio da comparação entre pontos característicos (SIFT) entre a imagem de entrada e os objetos previamente armazenados. As imagens testes são de 384x512 pixels. A implementação foi realizada em *software* e foi executada em uma Sun Sparc 10. O tempo de todo o tratamento e reconhecimento é em torno de 1,5 segundos. O autor comenta que 0,9 segundos são destinados para a geração da estrutura piramidal e 0,6 segundos para a etapa de reconhecimento.

O trabalho proposto em [RUC97] tem o objetivo de realizar a localização de objetos com um certo grau de oclusão. A localização e conseqüente reconhecimento se faz por meio da procura da menor distância de Hausdorff. A implementação é em *software* e foi testada sobre Sun Sparc Server 1000 - 8 processadores, onde para o exemplo de uma caixa apresentado no artigo, o processo leva aproximadamente 7 minutos para a localização.

O trabalho apresentado em [JOR96] propõe uma metodologia para a localização de placas de perigo instaladas em *containers* no porto de Esbjerg, Dinamarca. A etapa de localização é realizada por meio da seleção de todos os objetos que possuem formas semelhantes às placas de perigo. A classificação é realizada por uma rede neural do tipo

RAM. A implementação foi realizada em *software* e testada sobre uma máquina 486 - 66 MHz. O tempo para localização e classificação é em torno de 10 segundos por imagem.

O trabalho proposto em [MIN98] aborda um sistema para tratamento de imagens em tempo real. É implementado numa placa para o tratamento de imagens que possui um barramento PCI. Como exemplo de operação, ele demonstra uma aplicação para o reconhecimento de placas de carro na entrada de estacionamentos. Os carros estão parados e todos possuem aproximadamente a mesma distância até a câmera; com isto não há problemas de escalonamento. A etapa de localização e detecção da placa é realizada por heurísticas e é implementada em *hardware* (FPGA - 10K100 Altera). O reconhecimento da placa é realizado em *software* e implementado em computador PC por meio de uma rede neural *feedforward* de cinco camadas (600x241x42x100x34). O desempenho para localização e reconhecimento da placa é de 30 quadros por segundo.

O trabalho [ADO99] propõe o uso de redes neurais para a localização e classificação de marcas para a navegação (*landmarks*) visualizadas por robôs móveis situados em ambientes fechados. A procura por marcas ocorre somente em regiões possíveis, as quais são selecionadas por um analisador de tela (*screener*). O ponto negativo é que as marcas devem possuir o mesmo ângulo e devem estar em posições pré-estabelecidas da imagem. A implementação é realizada em *software* e possui um desempenho de 2,5 quadros por segundo para imagens de 256x256 pixels. O trabalho também apresenta uma comparação de tempos de execução para implementações em plataformas diferentes. As plataformas utilizadas foram: Sparc20, Pentium, Intel486, PAPRICA (ASIC). Esta última, realiza processamento paralelo e assim produz os resultados mais rapidamente.

O trabalho apresentado em [HAN99] propõe um neurochip analógico implementado como um ASIC para o processamento de imagem. O trabalho apresenta a construção de três diferentes tipos de redes neurais sem qualquer modificação estrutural do circuito. O autor apresenta como resultado a simulação do comportamento do operador lógico XOR. A taxa de saída do neurochip é de $12x10^6$ sinapses/segundo. A tecnologia utilizada para a implementação foi um processo CMOS analógico de 1,2 μm, com duas camadas de polisilício.

O trabalho em [SAN99] apresenta a nova versão do chip PAPRICA, que é um ASIC para processamento geral. Ele foi desenvolvido a partir de um processo CMOS de tecnologia 0,8μm, com uma camada de polisilício. É apresentada uma aplicação onde são utilizados 4 chips para realizar a tarefa de reconhecimento do montante expresso em cheques. A tarefa é realizada por meio de redes *neuro-fuzzy*, mas o trabalho não contempla a descrição desta parte, somente os resultados obtidos em comparação com PC e Sparc 10.

Em [KRA97], o autor apresenta uma metodologia para a geração de ASICs para o processamento de imagens em tempo-real. Apresenta, ao final do artigo, uma análise do resultado de algumas técnicas de processamento de imagens com o tempo de resposta do ASIC bem como a tecnologia de implementação deste ASIC.

O trabalho apresentado em [MAH97] realiza a extração de linhas e seus *end-points* (pontos terminais) por meio da Transformada de Hough em uma placa contendo FPGAs

em paralelo. Este artigo não apresenta uma descrição detalhada da placa. O tempo obtido para extração dos parâmetros de linhas em imagens de 256x256 pixels é de 48ms.

O trabalho proposto em [SUD98] explora a implementação de um algoritmo para esqueletização de uma imagem binária de 240x240 pixels. É realizada a comparação qualitativa entre a implementação em *software* e em *hardware*. Não são comentados resultados quanto ao desempenho das duas implementações.

Em [DAL97], tem-se a extração de pontos dominantes e contornos de objetos. Esta implementação é uma arquitetura VLSI (ASIC tecnologia 0,8µm) e um FPGA para o controle de dados e interface. A extração dos pontos dominantes é feita por meio da utilização de filtros FIR. Entretanto, o artigo não aborda claramente o desempenho e nem o tamanho da imagem utilizada para testes.

O trabalho proposto em [ALZ97] explora a segmentação em *hardware*. Este trabalho apresenta uma variação na etapa de filtragem para se adequar às restrições de *hardware*. O processo foi implementado em um ASIC com tecnologia de 0,8µm, com dupla camada de metal. O desempenho anunciado é de 30 quadros por segundo para imagens do tamanho de imagens VGA.

Em [BOS99], concebe-se uma estratégia para implementar uma convolução em um dispositivo reconfigurável (FPGA). O desempenho atingido é 14 vezes superior ao desempenho obtido por um *software* aplicado em um DSP C-40. O trabalho mostra um estudo de ocupação de FPGAs e a implementação é realizada em dois FPGAs XC4013 (consumo total de 962 CLBs).

O trabalho apresentado em [ROB94] apresenta a concepção de um projeto para a classificação de padrões. A técnica é implementada em ASIC, tecnologia 1,2µm, e em FPGA. O desempenho obtido para a implementação ASIC é de um ponto classificado em menos de 100ns.

Em [TOR98] é apresentada uma implementação de um filtro para a detecção de bordas pelo método de Canny-Deriche. A implementação é realizada em um ASIC com um processo de 1,0µm. O desempenho é de 30ns por pixel.

Por fim, em [WIA2000] aborda-se uma análise comparativa entre três metodologias para a implementação de operações de convolução sobre uma dada imagem. Nestes trabalhos são abordadas as metodologias de *Look-Up-Table*, Aritmética Distribuída e Convolução sem Multiplicação. A terceira técnica é também conhecida como sendo Convolução por Deslocamento e conforme citado no artigo, é a melhor técnica dentre as três para a maioria dos casos.

A TABELA 1 apresenta, de forma resumida, os trabalhos relevantes publicados na literatura. No Capítulo 2 serão analisados quatro destes trabalhos, escolhidos segundo o critério de afinidade com os objetivos desta proposta.

1.6 Estrutura da Tese

Apresenta-se a seguir a estrutura para esta tese. O Capítulo 2 apresenta a metodologia proposta para a solução do problema, bem como uma análise comparativa, situando este trabalho no contexto do estado-da-arte.

Seguindo a estrutura deste trabalho, o Capítulo 3 apresenta a implementação do sistema com as devidas justificativas para cada bloco componente, bem como o ambiente de prototipação. O Capítulo 4 é dedicado aos resultados obtidos. Por fim, no Capítulo 5 são apresentadas as conclusões.

TABELA 1 – Resumo do Estado-da-arte.

Referência	Função	Metodologia	Implementação	Desempenho
[GAV99]	Detecção e reconhecimento de sinais de trânsito e de pedestres	Transformada de Distância (*Distance Transforms*) e árvore hierárquica são utilizados para a detecção. Reconhecimento por RBF	*Software*	10-15 quadros / seg (360 x 288 pixels). Pedestres: 1 -5 quadros / seg.
[SIM98]	Detecção e reconhecimento de objetos com oclusão	Estágios de pré-processamento. Classificação por MLP	*Software*	3 minutos / imagem
[TSA99]	Reconhecimento de objetos com oclusão	Detecção por pontos dominantes. Reconhecimento pela distância euclidiana	*Software*	
[FOR99]	Detecção e reconhecimento de objetos	Detecção por extração de fundo e esqueletização. Reconhecimento por base de dados	*Software* Pentium II - 300MHz	3 quadros / seg
[LOW99]	Detecção e reconhecimento com oclusão	Filtro gaussiano como pré-processamento. Extração de pontos dominantes. Reconhecimento por combinação de SIFTs.	*Software* Sun Sparc 10	1 imagem (384 x 512 pixels) em 2 Seg
[RUC97]	Localização de objetos	Distância Hausdorff	*Software* 8 processadores Sun SPARC Server 1000	Exemplo caixa = +- 7min.
[JOR96]	Detecção e classificação de sinais de perigo	Detecção e classificação por pré-processamento e uma rede neural do tipo RAM	*Software* Pentium 486-66MHz	10 seg (detecção e classificação)
[MIN98]	Detecção e reconhecimento de placas de veículos	Detecção por pré-processamento e reconhecimento por MLP	Pré-processamento por FLEX10K100 Reconhecimento *software*	30 quadros / seg
[ADO99]	Detecção e reconhecimento de *Landmarks*	Reconhecimento por rede neural do tipo MLP	Comparação entre: PC, Sparc20, PAPRICA.	400ms / imagem (2,5 quadros / seg.)
[HAN99]	Somente arquitetura neural		ASIC tecnologia 1,2µm	$12x10^6$ sinapses / seg.
[SAN99]	Processamento genérico de imagens		4 chips PAPRICA-3 tecnologia 0,8µm	
[KRA97]	Processamento de imagens em tempo-real por ASICs		Vários ASICs	
[MAH97]	Extração de linhas pela Transformada de Hough		Placa contendo FPGAs	
[SUD98]	Esqueletização de imagens	Comparação entre *hardware* e *software*		
[DAL97]	Pontos dominantes e delineamento de objetos		ASIC (0,8µm) e FPGA	
[ALZ97]	Segmentação em tempo-real		ASIC tecnologia 0,8µm	30 quadros/seg
[BOS99]	Convolução		FPGA (2 x XC4013)	14 x mais rápido do que DSP C-40
[ROB94]	Classificação de padrões		ASIC tecnologia 1,2µm e FPGA	ASIC < 100ns / ponto
[TOR98]	Detecção de borda		ASIC tecnologia 1,0µm	30ns / pixel
[WIA2000]	Operações de Convolução	Comparação de várias metodologias para convoluções	FPGA	

2 Proposta de uma Metodologia para Implementação de Sistemas de Visão Computacional

2.1 Introdução

Este trabalho propõe uma metodologia para o desenvolvimento de aplicações de visão computacional realizando a integração de um DSP e um Dispositivo Lógico Programável representado por um FPGA. Esta proposta abordará uma metodologia completa, isto é, compreendendo todas as tarefas necessárias para a obtenção de uma solução para o problema alvo. As tarefas envolvidas compreendem: pré-processamento, extração de feições e classificação. O conjunto de métodos empregados para a solução deste problema compõe um sistema completo.

Esta metodologia propõe a utilização de diferentes formas de implementação, ora em *hardware* (FPGA) ora em *software* (DSP), visando encontrar o compromisso ideal entre desempenho e flexibilidade.

A integração entre DSP e FPGA justifica-se pela obtenção de uma partição que melhora o desempenho geral do sistema quando comparado a uma implementação puramente realizada em DSP; tal partição reduz tanto o tempo necessário para o produto chegar ao mercado como o custo final do projeto quando comparado a uma implementação puramente realizada em ASIC. Assim, este sistema utilizando um projeto conjunto de *hardware* e *software* apresenta uma boa relação entre desempenho e custo.

Como salientado no Capítulo 1, esta metodologia foi implementada e validada utilizando como estudo de caso o problema da localização e classificação de formas retangulares. Estas formas retangulares podem ser sinais de trânsito, marcas para navegação (*landmarks*) em robótica, placas de veículos, etc.

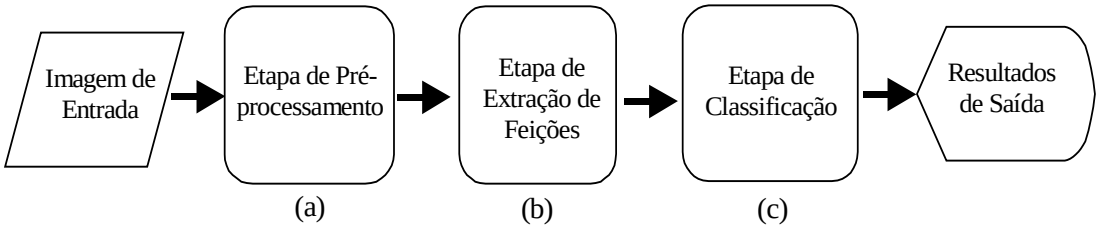
A seguir será apresentada a proposta para a solução do problema como um todo. Esta explicação não abordará conceitos de implementação, deixando estes conceitos para o Capítulo 3.

2.2 A Metodologia Proposta

A metodologia proposta é composta por três blocos. Estes blocos realizam tarefas nos três níveis da área de processamento de imagens, que são: baixo, intermediário e alto nível [GON92]. As tarefas a serem realizadas são: segmentar formas desejadas, que aqui serão formas retangulares ou formas quase retangulares; extrair feições que representam o conteúdo dos retângulos segmentados e, por fim, classificar o conteúdo destes retângulos.

A FIGURA 2.1 apresenta um diagrama em blocos da metodologia proposta, ressaltando os diferentes níveis de processamento e tipos de implementação. O bloco (a), correspondente ao baixo nível, é composto pela etapa de pré-processamento. O bloco (b) corresponde ao nível intermediário consistindo da etapa relacionada à extração de feições. Finalmente, o bloco (c) representa o alto nível e é responsável pela etapa de classificação das feições extraídas no bloco anterior. Além destes, a FIGURA 2.1 apresenta dois blocos externos ao sistema, que representam as tarefas de “Imagem de Entrada” e “Resultados de Saída”. O primeiro bloco, “Imagem de Entrada”, representa

toda a parte de aquisição da imagem, enquanto que o segundo bloco, “Resultados de Saída”, representa as tarefas após a etapa de classificação, por exemplo, tomada de decisão a partir dos dados classificados.



(a) – baixo nível

(b) – nível intermediário

(c) – alto nível

FIGURA 2.1 – Diagrama geral dos blocos da metodologia proposta.

2.2.1 Pré-processamento

Após uma imagem digitalizada ter sido adquirida, o primeiro passo é o pré-processamento, que corresponde ao processamento de baixo nível. Um grande número de técnicas de pré-processamento são disponíveis na prática. Estas incluem melhoramento da imagem, filtragem de ruído, isolamento de regiões, correção geométrica, restauração, reconstrução e segmentação. As técnicas de melhoramento da imagem podem ser classificadas em dois grandes grupos: domínio espacial e domínio das freqüências. As técnicas que utilizam o domínio espacial são baseadas na manipulação direta dos valores dos tons (intensidade luminosa) associados aos pixels; já no domínio das freqüências, as técnicas são baseadas na modificação da representação da imagem pela Transformada de Fourier. Na técnica de manipulação direta dos tons associados aos pixels o melhoramento em qualquer ponto na imagem pode depender somente do valor associado a este ponto ou pode depender do valor associado a este ponto e de seus pontos vizinhos. Este tipo de técnica faz uso de máscaras ou janelas que definem a região de vizinhança de um dado pixel.

A metodologia proposta utiliza a técnica de manipulação direta dos valores de pixels associados a um determinado ponto levando em consideração a sua vizinhança. Com isso, a técnica utilizada é realizada no domínio espacial e não das freqüências.

A FIGURA 2.2 apresenta, em forma de níveis, a descrição das tarefas empregadas na etapa de pré-processamento do sistema. No primeiro nível encontram-se descritas as macrotarefas a serem realizadas sob uma forma mais generalizada. O segundo nível apresenta a decomposição da macrotarefa de segmentação nas tarefas necessárias para que o sistema obtenha seus resultados. Por fim, o terceiro nível apresenta todas as subtarefas que são realizadas durante o pré-processamento do sistema.

Este terceiro nível mostrado na FIGURA 2.2 apresenta um conjunto de tarefas que formam uma possível solução para uma gama de aplicações voltadas para a área de visão computacional. Aplicações que envolvam tarefas de localização de segmentos de retas em uma imagem, por exemplo, podem vir a utilizar diretamente a implementação descrita neste trabalho.

As tarefas “j” e “k” presentes na FIGURA 2.2 estão dentro de um retângulo tracejado pois são necessárias para a solução do problema de localização e classificação de formas retangulares. Caso o problema difira deste, estas tarefas deverão ser substituídas pelas tarefas necessárias para a solução do problema em particular. Estas duas tarefas são implementadas em *software* para prover uma maior flexibilidade ao sistema além de prover uma maior facilidade de implementação. Contudo, as mesmas podem ser facilmente implementadas em *hardware* pois, como será visto mais adiante, não há o envolvimento de operações complexas, havendo, na sua grande maioria, operações de comparação.

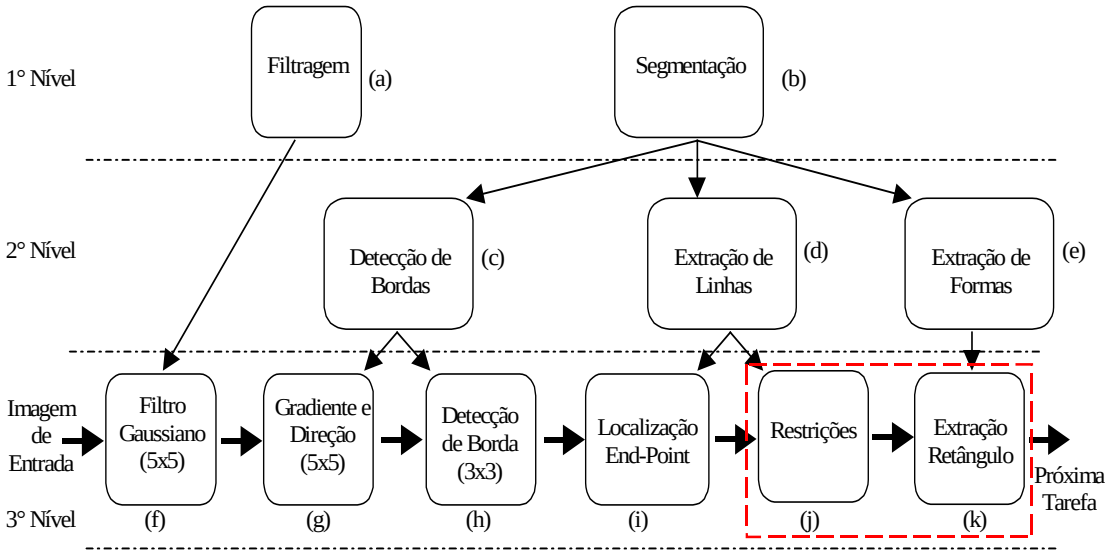


FIGURA 2.2 - Tarefas da etapa de pré-processamento.

A etapa de pré-processamento (FIGURA 2.2a e 2.2b) é composta por uma tarefa de filtragem seguida por uma segmentação. Estas tarefas têm como objetivo realizar o processo de localização de formas geométricas conhecidas a priori. Na literatura, pode-se encontrar vários métodos para a localização de formas geométricas, entre eles têm-se: análise de textura [LIB97], lógica fuzzy [HOE97], características locais [LOW99], características globais de formas desejadas (aproximação por polígonos [YAN98] e detecção de pontos dominantes [TSA99]) e morfologia matemática [FOR99]. Neste trabalho será empregado o método de características globais de formas desejadas.

É importante ressaltar que para cada tipo de problema pode-se adotar uma técnica diferente de localização de formas ou objetos. A técnica aqui escolhida é a que mostrou ser a mais adequada levando-se em consideração a facilidade de implementar parte desta etapa em *hardware* e a facilidade de se realizar a implementação conjunta de *hardware* e *software*. Esta definição deu-se através de leituras de trabalhos que empregam os métodos acima descritos e comparando-os em relação a uma possível implementação em *hardware* do mesmo.

A tarefa de filtragem é implementada por meio da aplicação de um filtro gaussiano sobre a imagem de entrada (FIGURA 2.2f). Este filtro atua como uma função suavizadora, devendo ser utilizada principalmente para minimizar os ruídos provenientes do processo de digitalização da imagem e da introdução de sombras e luzes que diferem de uma imagem para outra. Neste trabalho utiliza-se uma janela de

5x5 pixels para determinar a região de vizinhança em relação a um dado pixel central. A Equação (1) descreve o comportamento do filtro.

$$g'(x,y) = \frac{1}{D}\sum_{i=0}^{N-1}\sum_{j=0}^{M-1} w(i,j) * g(x + j - (M-1)/2, y + i - (N-1)/2) \qquad (1)$$

Onde: $w(i, j)$ = coeficientes de ponderação;
$g(x, y)$ = valor do tom associado ao pixel (x , y);
$g'(x, y)$ = novo valor do tom associado ao pixel (x, y);
M, N = largura e altura da janela de vizinhança (geralmente números ímpares);
D = somatório dos coeficientes de ponderação.

Abaixo, na TABELA 2, apresenta-se a matriz dos valores utilizados para os coeficientes de ponderação $w(i,j)$ da equação (1). Estes valores são obtidos através da Equação (2).

$$W_p = \frac{1}{d_p} \qquad (2)$$

Onde: W_p = coeficiente de ponderação do pixel p;
d_p = distância entre o pixel p e o centro da máscara (FIGURA 2.3);

TABELA 2 - Coeficientes de ponderação $w(p,q)$.

0,353	0,447	0,5	0,447	0,353
0,447	0,707	1	0,707	0,447
0,5	1	2	1	0,5
0,447	0,707	1	0,707	0,447
0,353	0,447	0,5	0,447	0,353

A macrotarefa de segmentação é realizada por meio das seguintes tarefas: detecção de bordas, extração de linhas e extração de formas. Estas tarefas são mostradas na FIGURA 2.2 itens “c”, “d” e “e”.

No trabalho apresentado em [HEA98], vários métodos de detecção de bordas são apresentados e também é realizada uma análise comparativa entre os métodos. Os trabalhos [HEA98] e [PAR97] relatam que o método de Canny é o que apresenta melhores resultados entre os diversos algoritmos estudados. Entretanto, este método possui um alto custo computacional, o que deve ser considerado no momento de desenvolvimento de aplicações visando um alto desempenho em relação ao tempo de resposta. Uma solução alternativa é apresentada em [ALZ97], onde é proposto um método adequado para a implementação em *hardware*. Este método realiza a detecção de bordas utilizando o método do gradiente e da direção que cada pixel possui em relação a uma dada região de vizinhança (FIGURA 2.2 itens “g” e “h”).

O procedimento do cálculo do gradiente e da direção para um dado pixel tomado como referência se dá através de uma máscara que está apresentada na FIGURA 2.3. Essa máscara é centrada no pixel de interesse, p_{ij}, para determinar sua direção relativa aos seus vizinhos. A direção relativa deste pixel será determinada dentre quatro possíveis: diagonal negativa – Dn, (dir=1), vertical – V, (dir=2), diagonal positiva – Dp, (dir=3) e horizontal – H, (dir=4). Os passos que realizam a determinação desta direção são:

1) Determinação dos valores necessários para o cálculo da diferença absoluta utilizando a máscara da FIGURA 2.3. Estes valores são calculados a partir dos tons de cinza obtidos na imagem.

$$V_s = V_s(1) + V_s(2), V_i = V_i(1) + V_i(2)$$
$$H_d = H_d(1) + H_d(2), H_e = H_e(1) + H_e(2)$$
$$Dp_s = Dp_s(1) + Dp_s(2), Dp_i = Dp_i(1) + Dp_i(2)$$
$$Dn_s = Dn_s(1) + Dn_s(2), Dn_s = Dn_s(1) + Dn_s(2)$$

2) Cálculo das diferenças absolutas:

$$V = | V_s - V_i |$$
$$H = | H_d - H_e |$$
$$Dp = | Dp_s - Dp_i |$$
$$Dn = | Dn_s - Dn_i |$$

3) Determinação do gradiente do pixel p_i e de sua direção relativa aos seus vizinhos:

$$gra = max\{V, H, Dp, Dn\}/2$$

$$dir = dir(min\{V, H, Dp, Dn\})$$

$Dp_s(2)$		$V_s(2)$		$Dn_s(2)$
	$Dp_s(1)$	$V_s(1)$	$Dn_s(1)$	
$H_e(2)$	$H_e(1)$	$\boldsymbol{p_{ij}}$	$H_d(1)$	$H_d(2)$
	$Dn_i(1)$	$V_i(1)$	$Dp_i(1)$	
$Dn_i(2)$		$V_i(2)$		$Dp_i(2)$

FIGURA 2.3 – Máscara utilizada para a determinação da direção.

O valor obtido da menor diferença absoluta nas quatro direções determina a escolha da direção do pixel, p_{ij}, pois esta é freqüentemente paralela à direção da borda. Para ilustrar isto, consideremos a presença de uma borda horizontal onde a diferença do nível de cinza é igual a 100 (FIGURA 2.4a). O resultado do cálculo, para o pixel hachurado de tom 0, dos valores de diferenças absolutas nas quatro direções é: Dn=200, V=200, Dp=200, H=0. Isto indica que a borda geralmente assume a mesma direção da menor diferença absoluta. O mesmo princípio pode ser generalizado para os outros três tipos de bordas.

Uma vez determinado o cálculo sobre todos os pixels pertencentes à imagem, teremos que cada pixel p_{ij} será representado por dois números, correspondentes ao gradiente e à direção. Por exemplo, (100,4) significa que este pixel possui um gradiente de 100 e representa uma borda horizontal (*dir*=4).

100	100	100	100	100	100	100
100	100	100	100	100	100	100
100	100	100	100	100	100	100
0	0	0	**0**	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

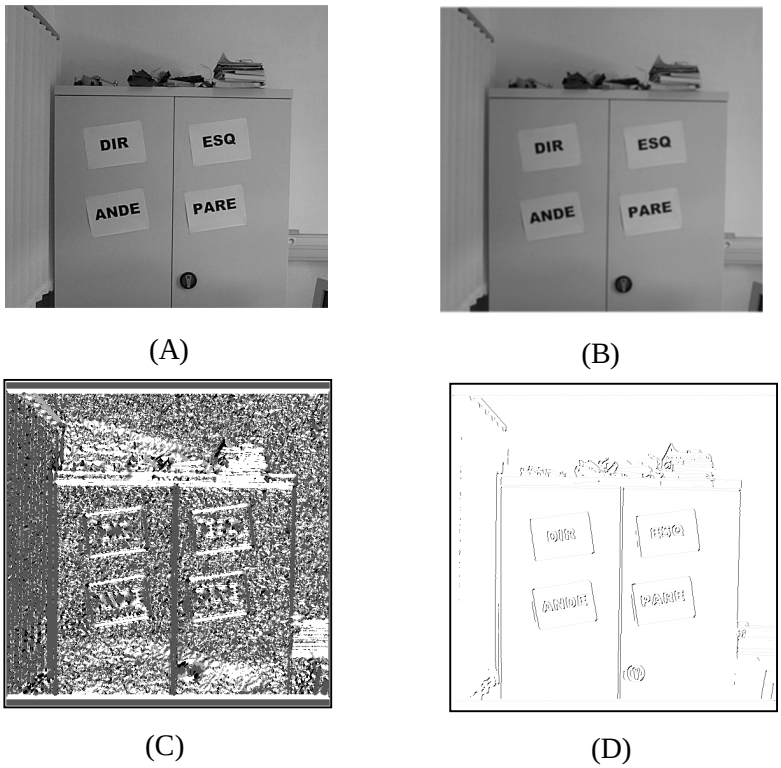
P1	P2	P3
P4	P5	P6
P7	P8	P9

FIGURA 2.4 – (a)Exemplo do cálculo da direção. (b)Janela de dimensão 3x3.

Para que este resultado seja útil nos próximos estágios de processamento é necessário obter bordas de largura de somente um pixel. A próxima tarefa (Figure 2.2h) realiza simultaneamente a detecção da borda e afinamento da mesma, produzindo bordas com espessura de um pixel. É importante salientar que nesta etapa existe um valor de limiar (*threshold*) que varia conforme a luminosidade da imagem. Portanto, torna-se importante o conhecimento deste valor antes da implementação. Usualmente este valor é obtido por meio de simulações realizadas em *software*.

O algoritmo que realiza esta tarefa é baseado em comparações feitas entre os elementos de uma janela de dimensão 3x3 (FIGURA 2.4b), levando-se em consideração as respectivas direções relativas dos pixels pertencentes a esta janela, bem como o valor de gradiente em cada um destes pixels. Para melhor exemplificar a operação deste algoritmo, consideremos a matriz apresentada na FIGURA 2.4a como sendo parte da imagem alvo. O pixel de valor 100, que está hachurado, possui uma direção relativa horizontal (Dn=200, V=200, Dp=200, H=0, logo *dir* = 4), indicando que a borda é horizontal. Conforme o algoritmo implementado, a direção e o gradiente deste pixel (que é o pixel P5 na FIGURA 2.4b) serão comparados com os gradientes de P2 e P8. Se for verificado uma determinada relação (P5 ≥ P2 E P5 > P8) então o pixel P5 é tido como borda. Com este algoritmo, pode-se verificar que há o afinamento da borda na FIGURA 2.4a. O algoritmo completo, bem como as respectivas relações de verificação de pertinência ou não a uma determinada direção da borda estão descritos em [ALZ97].

A FIGURA 2.5 ilustra estes três passos. A FIGURA 2.5a apresenta a imagem original. A FIGURA 2.5b mostra a imagem obtida após o filtro gaussiano. A FIGURA 2.5c apresenta a imagem com todas as direções calculadas. Por fim, a FIGURA 2.5d apresenta as bordas detectadas e com espessura de somente um pixel.



(A) (B)

(C) (D)

FIGURA 2.5 – Processo de detecção de borda.

Os processos descritos até este ponto (itens f, g e h da FIGURA 2.2) não localizam as linhas. O algoritmo somente detecta a existência de linhas (bordas), com alguma dimensão e algum ângulo dentro da imagem. Para se obter a localização exata destas linhas, dois diferentes métodos podem ser utilizados:

> *Localização de Vértices*. Máscaras de convolução pré-definidas são aplicadas sobre a imagem para se obter a localização dos vértices. Este processo somente obtém os ângulos retos ou com valores muito próximos a estes. Este método não pode ser utilizado na prática em nosso sistema, pois o processo de aquisição da imagem introduz ruídos, tornando muitos dos ângulos retos em ângulos não retos. Entretanto, existem outras aplicações que utilizam este método, tais como [MOK98], [CHA99] e [FRE77].
>
> *Localização de End-point*. Esta técnica é apresentada por [COC95]. Este método pode ser facilmente implementado em *hardware*, e possui um bom desempenho em relação à velocidade de processamento, pois realiza somente operações de comparação e busca em memória. Entretanto, uma etapa de detecção de borda eficiente é necessária para se obter bons resultados do processo de localização de *end-point*. Este algoritmo implementa a tarefa apresentada na FIGURA 2.2i. Este algoritmo é aplicado sobre as bordas detectadas e devidamente afinadas. Para ser realizada esta tarefa, efetua-se uma busca na imagem (FIGURA 2.5d) de cima para baixo e da esquerda para a direita, de um pixel que possua uma das quatro direções relativas possíveis (dir = 1 ou 2 ou 3 ou 4). A seguir, esta direção é perseguida, enquanto a mesma persistir, em uma vizinhança verificada por meio de uma matriz 3x3. Cada pixel visitado é marcado para evitar nova comparação. Este laço persiste até não existir mais pixels não marcados na imagem. A saída deste algoritmo nos fornece as posições iniciais e finais de cada linha, sendo que com estas informações pode-se calcular o ângulo e o comprimento de cada linha existente na imagem.

As duas próximas tarefas (itens “j” e “k” na FIGURA 2.2) são necessárias para a solução do problema proposto como estudo de caso, que é a localização e classificação de formas retangulares. Caso o problema seja diferente, como por exemplo, reconhecimento de impressões digitais, estas duas tarefas deverão ser substituídas pelas tarefas necessárias para a solução deste problema em particular. Objetivando uma maior flexibilidade do sistema, estas tarefas foram desenvolvidas em *software*.

A tarefa descrita pelo item “j” da FIGURA 2.2 é a aplicação de restrições: dimensão mínima e ângulo máximo.

A restrição de dimensão mínima existe por causa da ocorrência de *gaps* (espaços entre dois pixels consecutivos) no processo de segmentação. Uma alternativa para diminuir estes *gaps* seria a aplicação do operador morfológico de fechamento sobre a imagem [FAC96].

A restrição de ângulo máximo também é ligada à existência destes *gaps*. Tem sido verificado, por experiências em *software*, que a partir de 50° até 130° (exceção para 90°) o processo de segmentação produz vários *gaps* em uma mesma linha dificultando

as demais tarefas. Por causa desta restrição, este algoritmo não é capaz de localizar linhas cujo ângulo seja superior a 50° (exceção para 90°).

No final da tarefa de aplicação destas restrições, obtém-se uma imagem com linhas que respeitam as restrições impostas. A FIGURA 2.6 mostra a imagem obtida por este algoritmo.

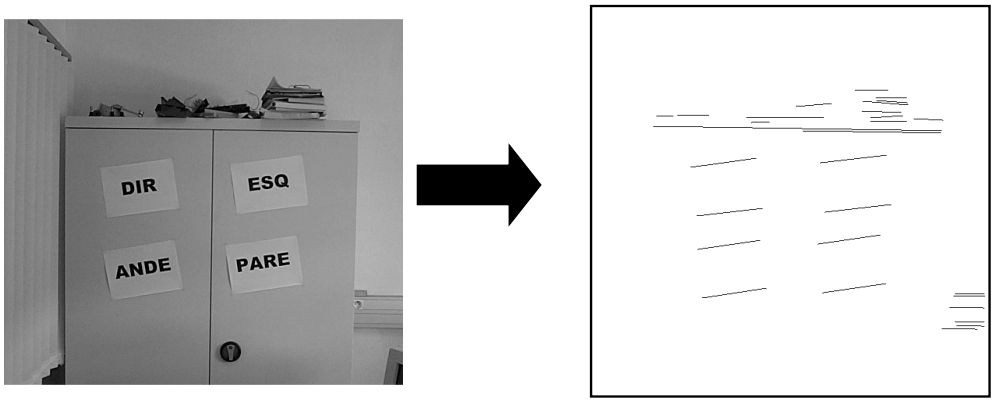


FIGURA 2.6 – Linhas obtidas usando o algoritmo de localização de end-point com restrições.

Finalmente, estas linhas obtidas devem ser associadas a possíveis formas retangulares (item “k” da FIGURA 2.2). Este procedimento é realizado pela procura de similaridades entre as linhas obtidas. As similaridades levadas em conta são: comprimentos das linhas, ângulos das linhas e posições iniciais relativas ao eixo *X*.

Com este procedimento de associação de linhas, obtêm-se dois lados da forma retangular. Assim, pode-se calcular os demais lados ou trabalhar com base em dois lados somente. A FIGURA 2.7 mostra o resultado do algoritmo completo sobre a imagem original (FIGURA 2.5a). Observa-se na FIGURA 2.7 a existência de formas retangulares tidas como ruído do processo descrito. Contudo, estes ruídos serão descartados pelo processo de classificação, havendo assim a desconsideração destas formas retangulares indesejáveis.

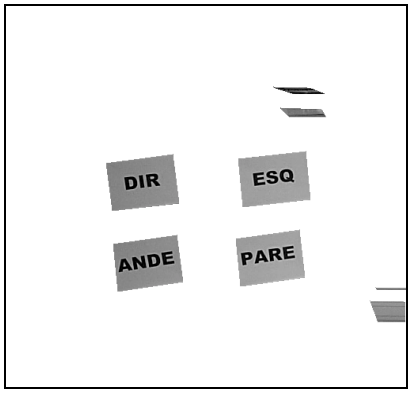


FIGURA 2.7 – Formas retangulares obtidas.

2.2.2 Extração de Feições

A próxima etapa de processamento corresponde ao nível intermediário. A tarefa neste nível consiste em realizar um mapeamento do processamento realizado no baixo nível, etapa de pré-processamento, para entidades significativas do próximo nível, etapa de classificação.

Sistemas de reconhecimento de padrões geralmente consideram um espaço de características dentro do qual um vetor de observação é mapeado. Com este vetor de observação mapeado, pode-se verificar a qual classe ele pertence. O propósito da extração de feições é a redução da quantidade de dados, ou redução da dimensionalidade dos dados, obtida por meio da observação de certas características ou propriedades que distinguem os padrões de entrada. Na extração de feições transforma-se um vetor de observação em um vetor de características empregando alguma função ortogonal ou não-ortogonal, de modo a se obter um espaço de características não correlacionado.

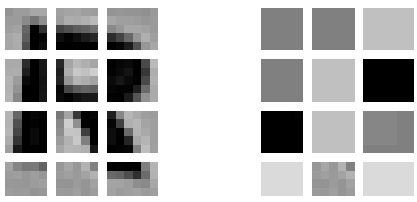
Uma grande variedade de métodos para extração de feições tem sido desenvolvida. Entre as técnicas mais comuns que são utilizadas incluem-se a FFT, momentos invariantes, distribuição de Chord, polinomiais ortogonais e funções de Gabor. Transformadas ortogonais como Walsh, Walsh Hadamard e DCT, que são empregadas para a compressão de dados, também são utilizadas para a extração de feições [KUL94].

Uma vez que são obtidas as feições que melhor representam os padrões de entrada, a tarefa de classificação ou reconhecimento se reduz à realização de uma partição do espaço de características.

Na Seção seguinte serão descritas algumas técnicas de extração de feições que podem ser utilizadas para a solução do presente problema. As técnicas são: Zoning, Momentos Invariantes e Descritores de Fourier.

2.2.2.1 Zoning

Este método consiste na sobreposição de uma grade *n* x *m* na imagem em que se quer obter a extração de feições, por exemplo, um dado caractere. Em cada zona *n* x *m* a média do tom de cinza é computada, resultando em um vetor de características de tamanho *n* x *m* (FIGURA 2.8). Este método não é invariante à iluminação. Ocorrem problemas também quando a imagem está sob efeito de rotação, o que, neste caso, requer uma etapa de pré-processamento.



(a) (b)

FIGURA 2.8 – Zoning de um caractere em níveis de cinza. (a) Uma grade 4x3 superposta. (b) Média do nível de cinza de cada zona.

Além do problema da variação da iluminação, este método também é susceptível a variação do *background* (fundo da imagem). Este método é aconselhável para ambientes controlados, como por exemplo, reconhecimento de caracteres (OCR). Como uma

vantagem deste método perante os demais que serão apresentados, pode-se ressaltar a facilidade de implementação em *hardware*, pois as operações matemáticas envolvidas não são complexas.

2.2.2.2 Momentos Invariantes

Esta técnica define um conjunto de sete funções invariantes a momentos (translação, escala e rotação) para os padrões de entrada. A principal desvantagem da técnica de invariância a momentos é que não há garantia de que estas sete funções formem um conjunto completo de descritores para os padrões de entrada. Entretanto, para várias aplicações práticas, por exemplo, reconhecimento de aeronaves por meio de uma RNA [KUL94], estas sete funções foram suficientes para a diferenciação entre os padrões de entrada.

Um padrão bidimensional ou imagem pode ser representado por uma matriz quadrada *n* por *n*, onde cada elemento é o tom associado ao pixel (*f(x,y)*). Os pixels são assumidos como tendo uma altura e comprimento unitário e com isto suas posições podem ser especificadas por meio de coordenadas inteiras x_t, y_t, t=1,2,…,n. Assim sendo, o jk-ésimo momento será:

$$M_{jk} = \sum_{l=1}^{n}\sum_{m=1}^{n}(x_l)^j (y_m)^k f(x_l, y_m) \qquad (3)$$

Para j, k = 0,1,2,3, etc.

Se além disto, considerar-se que os pixels são valores binários (imagem preto e branco) com “zeros” e “uns”, tem-se:

$$M_{jk} = \sum_{A}(x)^j (y)^k \qquad (4)$$

Onde o somatório compreende todos os elementos em A, isto é, todos os pixels “pretos” ou “uns” da matriz. Da equação (4) pode-se observar que M_{00} = a área do padrão, ou seja, o número de pixels “pretos” ou “uns” na área considerada. O centro de gravidade (também chamado de média ou média aritmética) dos padrões é determinado por:

$$\bar{x} = \frac{M_{10}}{M_{00}} \quad \text{e} \quad \bar{y} = \frac{M_{01}}{M_{00}} \qquad (5)$$

Os momentos tornam-se invariantes à translação quando estes são comparados com o centro de gravidade, ou seja:

$$\overline{M}_{jk} = \sum_{A}(x-\bar{x})^j (y-\bar{y})^k \qquad (6)$$

Estes momentos, momentos centrais, podem ser normalizados para invariância a escala, sendo calculados como:

$$\mu_{jk} = \frac{\overline{M}_{jk}}{M_{00}^{\gamma}} \qquad (7)$$

Onde $\gamma = \frac{j+k}{2} + 1$.

Hu em 1962 [HU62] desenvolveu as seguintes sete funções de momentos centrais que são invariantes a diferenças rotacionais e de escala. São elas:

$$\phi_1 = (\mu_{20} + \mu_{02}) \tag{8}$$

$$\phi_2 = (\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2 \tag{9}$$

$$\phi_3 = (\mu_{30} - 3\mu_{12})^2 + (3\mu_{21} - \mu_{03})^2 \tag{10}$$

$$\phi_4 = (\mu_{30} + \mu_{12})^2 + (\mu_{21} + \mu_{03})^2 \tag{11}$$

$$\begin{aligned}\phi_5 = {} & (\mu_{30} - 3\mu_{12})(\mu_{30} + \mu_{12}) \cdot \left[(\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2\right] \\ & + (3\mu_{21} - \mu_{03})(\mu_{21} + \mu_{03}) \cdot \left[3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2\right]\end{aligned} \tag{12}$$

$$\phi_6 = (\mu_{20} - \mu_{02})\left[(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2\right] + 4\mu_{11}(\mu_{30} + \mu_{12})(\mu_{21} + \mu_{03}) \tag{13}$$

$$\begin{aligned}\phi_7 = {} & (3\mu_{21} - \mu_{03})(\mu_{30} + \mu_{12}) \cdot \left[(\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2\right] \\ & - (\mu_{30} - 3\mu_{12})(\mu_{21} + \mu_{03}) \cdot \left[3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2\right]\end{aligned} \tag{14}$$

Como se verificou por meio destas equações descritas acima, este método possui dificuldades de se implementar em *hardware*, pois ele requer várias operações matemáticas complexas. Contudo, [MOR99] apresenta uma alternativa para este método de extração de feições implementada em *hardware*.

2.2.2.3 Descritores de Fourier

Este método consiste na obtenção de um conjunto de coeficientes que representam uma dada borda. Observa-se que é necessário o uso de uma rotina para extrair o contorno de cada objeto a ser descrito.

A FIGURA 2.9 mostra uma borda de N pontos no plano *xy* de uma dada imagem exemplo. Começando de um ponto arbitrário (x_0, y_0), pode-se encontrar os pares ordenados (x_0, y_0), (x_1, y_1), (x_2, y_2), ..., (x_{N-1}, y_{N-1}) ao longo da borda (por exemplo) no sentido anti-horário. Essas coordenadas podem ser expressas na forma $x(k) = x_k$ e $y(k) = y_k$. Com essa notação, a própria borda pode ser representada como uma seqüência de coordenadas s(k) = [x(k), y(k)], para k = 0,1,2, ... N-1. Além disso, cada par ordenado pode ser tratado como um número complexo da forma

$$s(k) = x(k) + jy(k) \tag{15}$$

para k = 0, 1, 2, ..., N-1. Ou seja, o eixo x é tratado como o eixo real, enquanto que o eixo y é tratado como o eixo imaginário de uma seqüência de números complexos. Embora a interpretação da seqüência tenha sido reformulada, a própria natureza da

borda não foi alterada. Obviamente, essa representação possui uma grande vantagem: ela reduz um problema de duas dimensões a uma só dimensão.

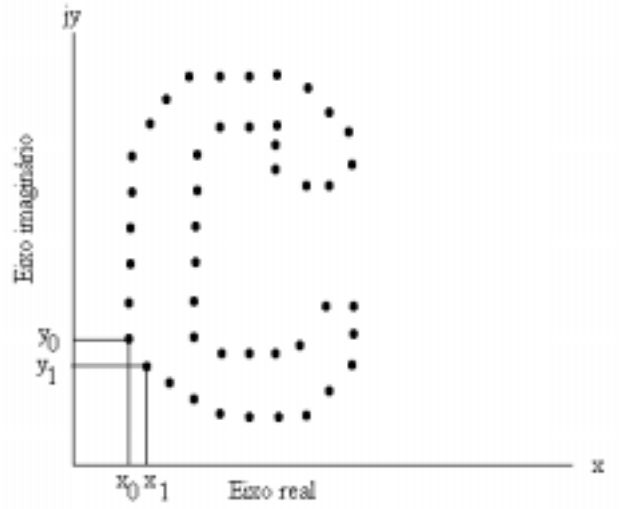


FIGURA 2.9 – A borda detectada de um dado caractere. Os pontos (x0, y0) e (x1, y1) são (arbitrariamente) os dois primeiros pontos da seqüência.

A transformada discreta de Fourier (DFT) de s(k) é definida como:

$$a(u) = \frac{1}{N}\sum_{k=0}^{N-1} s(k)\exp[-j2\pi uk / N] \qquad (16)$$

para u = 0, 1, 2, ... N-1. Os coeficientes complexos a(u) são chamados de *descritores de Fourier* da borda dada. A transformada inversa de Fourier de a(u) reconstrói s(k), ou seja:

$$s(k) = \sum_{u=0}^{N-1} a(u)\exp[\,j2\pi uk / N] \qquad (17)$$

para k = 0, 1, 2, ..., N-1. Suponha, entretanto, que no lugar de todos os valores a(u), apenas os primeiros M coeficientes sejam usados. Isso é equivalente a igualar a(u) = 0 para u > M-1 na Equação (16). O resultado é a seguinte aproximação de s(k):

$$\hat{s}(k) = \sum_{u=0}^{M-1} a(u)\exp[\,j2\pi uk / N] \qquad (18)$$

para k = 0, 1, 2, ..., N-1. Assim sendo, tem-se apenas os M primeiros coeficientes para a obtenção de cada componente s(k). Com isto, há o mesmo número de pontos na borda, mas houve uma redução da quantidade de termos empregados na reconstrução de cada um destes pontos. Se o número de pontos da borda for grande, então M é geralmente selecionado como sendo uma potência de 2, de maneira que um algoritmo de FFT possa ser usado no cálculo dos descritores. Cabe salientar que na transformada de Fourier os componentes de alta-freqüência geralmente estão relacionados a detalhes finos, enquanto que os componentes de baixa-freqüência determinam a forma global. Portanto, quanto menor for M, mais detalhes são perdidos na fronteira.

2.2.3 Classificação

A última etapa corresponde ao processamento de alto nível (FIGURA 2.1c). Fazem parte deste nível de processamento o reconhecimento de objetos e a interpretação da cena (FIGURA 1.4).

A tarefa de classificação consiste em associar rótulos a um dado objeto baseado na informação fornecida pelos seus descritores (vetor de características). Um classificador pode ser implementado a partir de técnicas de aprendizagem supervisionada ou não supervisionada. Na categoria supervisionada, os classificadores aprendem com a ajuda de conjuntos de treinamento, enquanto que na categoria não-supervisionada, os classificadores aprendem sem a ajuda de conjuntos de treinamento previamente classificados.

Dentre os vários tipos de classificadores existentes, há os classificadores que se utilizam das técnicas implementadas em RNAs. Para a solução do problema apresentado neste trabalho, resolveu-se implementar a fase de classificação através do uso de RNA, pois é um dos objetivos. Além disto, as redes neurais são conhecidas por apresentarem bons resultados quando operando como classificadores.

Redes Neurais Artificiais representam uma boa alternativa para classificadores. Modelos de RNAs com algoritmo de aprendizado tal como o *backpropagation* estão sendo utilizados como classificadores supervisionados; já as redes neurais auto-organizáveis ou ART, com aprendizado competitivo, são utilizadas como classificadores não-supervisionados.

As RNAs têm sido largamente utilizadas nos mais diversos campos científicos e industriais. Elas podem ser empregadas para solucionar uma grande variedade de problemas que são difíceis de serem resolvidos por métodos tradicionais. As redes neurais são especialmente interessantes para problemas onde a capacidade de generalização é muito importante.

Embora RNAs tenham sido muitas vezes implementada em *software*, versões em *hardware* estão ganhando importância. Implementações em *software* possuem a vantagem da facilidade de implementação, mas possuem a desvantagem quanto ao desempenho, em relação a velocidade de processamento. As versões em *hardware* geralmente são mais difíceis de serem implementadas e consomem um longo tempo para isto, mas em compensação, possuem um desempenho muitas vezes superior às versões em *software*.

Alguns parâmetros restritivos devem ser observados para qualquer implementação de uma RNA, seja em *software* ou *hardware*:

- A topologia da rede (*feedforward* ou *feedback*);
- O número de entradas e saídas;
- O número de neurônios;
- O número de pesos sinápticos por neurônio;
- O número de camadas;
- O algoritmo de aprendizagem;
- Operações em ponto-flutuante.

Para implementações em *hardware*, deve-se considerar também:

- A tecnologia empregada (analógica, digital ou híbrida);
- A representação numérica para os sinais de entrada, pesos sinápticos, saída e função de ativação da saída (ponto fixo ou ponto flutuante);
- A precisão para os sinais de entrada, pesos sinápticos, saída e função de ativação da saída (número de bits).

Para o caso de uma RNA implementada em *hardware*, a tecnologia empregada define o custo, desempenho e configuração da mesma:

- Implementação analógica possui um bom desempenho e baixo custo, mas esta é difícil de implementar devido às tensões de referência necessárias, além das dificuldades encontradas devido aos valores dos pesos sinápticos;
- Implementação digital possui um alto custo e menor desempenho quando comparado à implementação analógica, mas este tipo de implementação, digital, é mais rápida de ser obtida devido ao grande número de ferramentas de CAD existentes.

Ambas metodologias, analógica e digital, possuem uma topologia fixa após a implementação, resultando em uma RNA adequada somente para a aplicação alvo original. Dispositivos configuráveis (FPGA) podem ser empregados para implementar redes neurais em *hardware* ([MOL99][FIG98][FIG99][ROS98][CLO94]). Além disso, a implementação digital usando FPGA permite a redefinição da topologia neural utilizando o mesmo *hardware*. A desvantagem de uma implementação por meio do uso de FPGA em relação aos dispositivos ASIC é o desempenho. FPGAs normalmente são mais lentos do que os ASICs.

Na implementação de RNAs, deve-se considerar para a fase de aprendizado uma boa precisão numérica para se obter os pesos sinápticos. Se uma pequena precisão numérica for utilizada, durante a fase de aprendizado, a RNA não convergirá a um valor de erro aceitável e esta gerará erros durante a fase de propagação. Por outro lado, o custo computacional na fase de propagação está associado fundamentalmente com a velocidade de processamento. Esta velocidade é necessária devido ao grande volume de operações de soma e multiplicação que devem ser executadas. A precisão numérica desta fase também depende intimamente da precisão adotada durante a fase de aprendizado [MOL99].

Como mencionado acima, uma forma ideal de implementação de RNA é a utilização de *software* e *hardware* trabalhando em conjunto. A implementação de aritmética em ponto-flutuante, muito utilizada durante a fase de aprendizado, em dispositivos programáveis (FPGA, por exemplo) é proibitiva devido ao espaço necessário para a sua implementação. Portanto, a fase de aprendizado é uma excelente candidata a ser implementada em *software* (processador de propósito geral – GPP). Sendo implementada em *software*, a fase de aprendizado permite o uso da representação em ponto-flutuante, melhorando, com isto, a precisão numérica das operações, e garantindo assim, uma boa convergência a valores aceitáveis de erro. Por outro lado, a fase de propagação pode ser implementada em *hardware* pois isso aumentaria a velocidade de processamento. Utilizando um *hardware* dedicado muitas operações do tipo multiplica-acumula podem ser realizadas em paralelo.

2.3 Relacionamento entre o estado-da-arte e o problema

Esta seção visa apresentar os trabalhos que implementam todas as tarefas necessárias para a obtenção de uma dada resposta frente a um problema particular e similar ao tratado por este trabalho. Com base neste estudo, identificamos nove trabalhos que são próximos do objetivo deste trabalho. São eles : [GAV99], [SIM98], [TSA99], [FOR99], [LOW99], [RUC97], [JOR96], [MIN98], [ADO99]. Dentre estes nove, citam-se os artigos de [GAV99], [JOR96], [MIN98] e [ADO99] como sendo os que possuem uma maior afinidade ao objetivo esperado neste trabalho.

Em [ADO99] há a necessidade de que as marcas estejam em lugares pré-definidos, além de ser processado em *software*. É feita uma análise com o uso do PAPRICA, mas este é conectado a um *host*. Em [MIN98], o carro necessita estar a uma distância fixa e conhecida. Esta limitação quanto a distância permite aplicar uma regra de proporção que facilita uma rápida detecção dos números da placa. O sistema foi desenvolvido como sendo parte em *software*, instalado em PC, e parte em *hardware*, FPGA. Em [JOR96] o sistema foi implementado totalmente em *software*. Por último, em [GAV99] o sistema também foi implementado totalmente em *software*, portanto não se beneficia de qualquer grau de paralelismo que pode ser aplicado a todas as aplicações de processamento de imagens.

Como se pode observar, o trabalho aqui proposto é semelhante aos artigos [GAV99], [JOR96], [MIN98] e [ADO99] em termos de métodos empregados tanto no baixo nível como no alto nível de processamento, contudo, a metodologia de implementação das técnicas é diferente, como será visto mais adiante, apresentando uma alternativa para a obtenção de sistemas completos e visando um ganho de desempenho em termos de tempo de resposta aliado a uma possível minimização do sistema. Por fim, todos estes trabalhos, inclusive o proposto, não suportam qualquer grau de oclusão nos objetos a serem localizados e ou classificados.

A TABELA 3 apresenta as diferenças e semelhanças entre os quatro trabalhos afins e o trabalho aqui proposto. Esta tabela considera as técnicas utilizadas para o processamento de imagens realizado no alto e baixo nível.

TABELA 3 - Diferenças e semelhanças.

Referência	**Baixo nível**	**Alto nível**
[ADO99]	Detecção de borda	Transformada de Distância
[MIN98]	Segmentação	RNA
[JOR96]	Detecção de borda	RNA
[GAV99]	RNA	RNA
Este Trabalho	Segmentação	RNA

2.4 Conclusões

Este capítulo apresentou de forma teórica a metodologia que será utilizada para a solução do problema proposto e que será implementada no capítulo seguinte.

As técnicas referentes à etapa de pré-processamento que são implementadas em *hardware* foram escolhidas pela facilidade de implementação e por proporcionarem bons resultados. Contudo, torna-se evidente que estas técnicas não representam boas

alternativas para todas as condições atmosféricas (nevoeiro, chuva, neve, dentre outras) possíveis, visto que para isto necessitar-se-iam de muitas etapas adicionais para se corrigir os efeitos atmosféricos.

Uma das principais características do sistema proposto é a utilização de uma rede neural para a tarefa de classificação. A RNA pode ser treinada para reconhecer determinados objetos em uma cena, a partir de um conjunto de exemplos. Como abordado anteriormente, a fase de aprendizado pode ser implementada em *software*, aumentando a flexibilidade do sistema, enquanto a fase de reconhecimento, pode ser implementada em *hardware*, aumentando o desempenho do sistema em relação à velocidade do processamento.

Por fim, foi apresentado uma comparação entre o trabalho aqui proposto e alguns trabalhos relevantes encontrados na literatura científica internacional.

3 Implementação do Sistema Proposto

3.1 Introdução

Este Capítulo descreve a implementação da metodologia proposta e discutida no Capítulo 2, visando uma possível solução do problema de localização e classificação de formas retangulares. Para isto, este capítulo abordará o ambiente de prototipação empregado para realizar os ensaios, a partição do sistema entre as partes *software* e *hardware*, e por fim, a descrição funcional do sistema. Nesta última parte, são mencionados os blocos implementados bem como a comunicação entre esses blocos.

3.2 Ambientes de Prototipação

Como será visto, este trabalho contempla a implementação da metodologia proposta em plataformas de prototipação rápida. Tal decisão deve-se a:

1) Rápida prototipagem. Pelo próprio nome da plataforma de prototipação rápida. Assim, consegue-se um maior número de testes para se obter uma melhor arquitetura em um reduzido tempo.
2) Disponibilidade. As duas placas utilizadas estavam disponíveis para utilização em um curto intervalo de tempo, permitindo assim um desenvolvimento mais rápido.
3) Segurança. As placas já se encontram devidamente validadas, facilitando o processo de localização de falhas no sistema implementado.

Para o desenvolvimento deste trabalho foram utilizados dois tipos diferentes de plataformas de prototipação. A primeira plataforma foi empregada durante o início das pesquisas; nesta foram implementadas duas arquiteturas neurais para estudo de viabilização da implementação de RNAs em *hardware/software* ([MOL99] e [MOL2000]). A segunda plataforma foi empregada para a implementação da metodologia proposta no capítulo anterior e que será descrita neste capítulo.

Primeiramente foi utilizada uma placa desenvolvida no Laboratório de Informática, Robótica e Microeletrônica de Montpellier, a qual é denominada de LIRMM [TOR97].

Esta placa foi desenvolvida de modo a se obter um ambiente de prototipação rápido baseado em um processador de sinais (DSP), para a prototipação de *software*, e dois circuitos configuráveis (FPGA), para a prototipação de *hardware*.

O DSP é o processador TMS 320C40 da Texas, que possui uma unidade lógica e aritmética (ULA) e uma comunicação inter-processador bem conhecidas. A ULA está baseada em uma unidade de ponto flutuante de ciclo único, com somador e multiplicador distintos. Estas duas operações podem, portanto, ser realizada em paralelo. Este DSP utiliza seis portas de comunicação para realizar comunicações inter-processadores de alta velocidade, sendo que cada porta está habilitada a realizar transferências a uma taxa de 20 Mbytes/s de dados bidirecionais.

Os dois FPGAs utilizados são da Xilinx (XC4013). Estes FPGAs possuem, de forma individual, 576 CLBs (13.000 portas lógicas equivalentes) e 208 pinos que podem ser interligados aos componentes periféricos.

Para aumentar a flexibilidade da placa LIRMM, os dois FPGAs estão interconectados por um barramento de 34 bits, permitindo com isto, particionar o *hardware* necessário

entre os dois FPGAs. Além disto, para incorporar uma maior flexibilidade, há uma memória RAM estática local de 128Kbytes para cada um dos FPGAs. A frequência do *clock* pode ser ajustada para 20Mhz ou 40Mhz.

Na figura 3.1 apresenta-se a placa de prototipação rápida LIRMM.

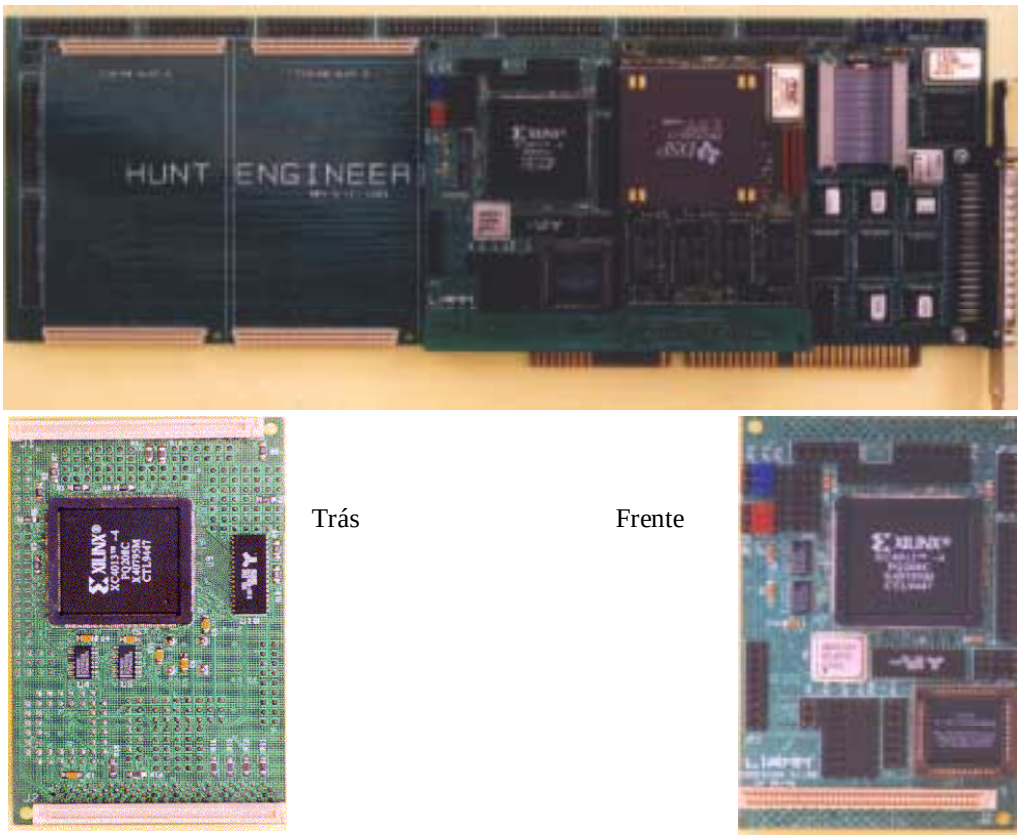


FIGURA 3.1 – Placa de Prototipação LIRMM.

Num segundo momento, foi utilizado o ambiente de prototipação APTIX [APT99]. O ambiente APTIX é uma nova geração de plataformas de prototipação. Este ambiente é composto basicamente por uma placa organizada como uma matriz de conectores fêmeas permitindo conectar grande parte dos circuitos integrados existentes, tais como, FPGAs, microcontroladores, DSPs, conversores A/D e D/A, memórias, etc. Para este trabalho, uma placa APTIX do tipo MP3 foi utilizada. Esta placa possui 2 módulos com FPGAs da Altera (Flex 10k100, 100.000 gates equivalentes em cada módulo), 1 módulo DSP (D950 core - ST Microelectronics) e 2 módulos de memória de 256K palavras por 4bits. Os módulos de FPGA podem utilizar dois sinais de clock, 16 MHz ou 25 MHz. A FIGURA 3.2 apresenta o ambiente de prototipação APTIX.

Como pode-se observar pela FIGURA 3.2, este ambiente de prototipação possui além da placa de prototipação, um conjunto de facilidades para se realizar o processo de testes e localização de falhas. Tais processos são realizados com o auxílio do osciloscópio e do analisador de sinais. A placa APTIX, por meio de seu *software* instalado na estação de trabalho, programa o analisador lógico para capturar os sinais necessários para um determinado conjunto de testes.

O computador PC é utilizado para o desenvolvimento, compilação e *download* do *software* a ser executado pelo DSP.

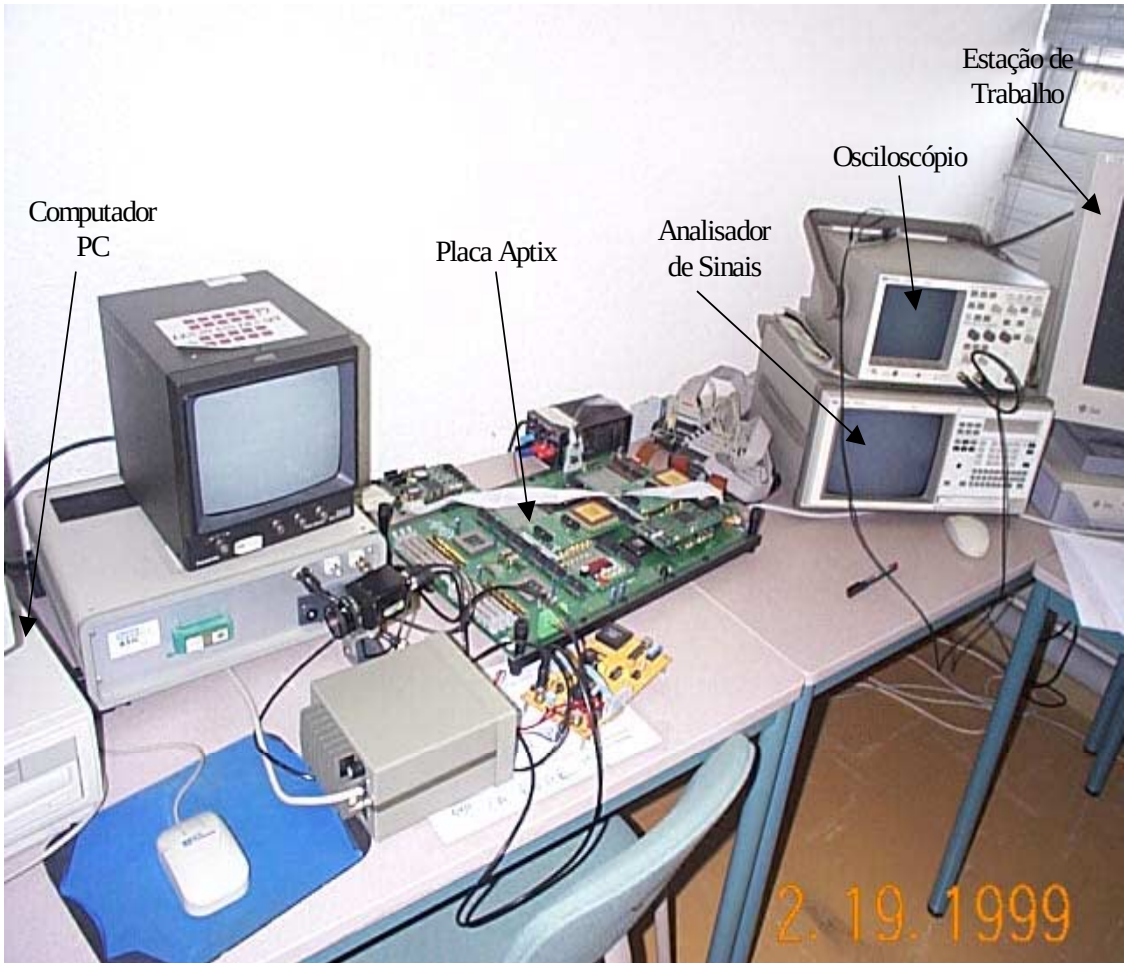


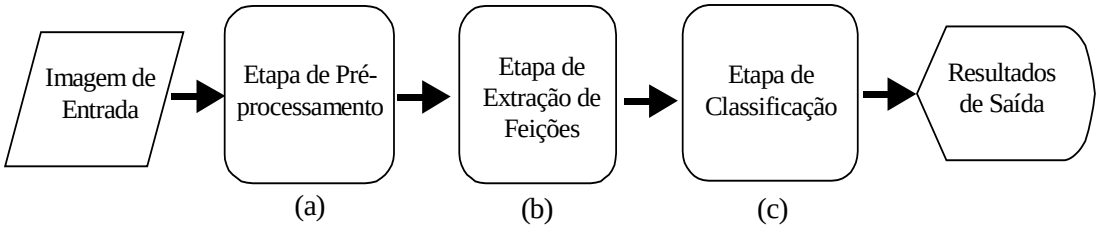
FIGURA 3.2 – Ambiente de Prototipação APTIX.

Como estes dois ambientes de prototipação permitem implementações conjuntas de parte do sistema em *software* e outra parte do mesmo em *hardware*, há a necessidade de programação em formas distintas. O processador de sinais é programado após a geração e avaliação de um código gerado em linguagem C. Os circuitos configuráveis (FPGA) são programados por meio de ferramentas de mapeamento de *hardware* e síntese lógica (Synopsys, Xilinx, Altera).

3.3 Partição do Sistema

Nesta Seção é apresentado o particionamento do sistema proposto no Capítulo 2. Como será abordado posteriormente, o sistema é implementado através da integração entre *software* (processador DSP) e *hardware* (dispositivo programável FPGA). Uma cuidadosa partição entre as partes *hardware* e *software* do sistema pode resultar em uma implementação com melhor desempenho que uma solução puramente *software*, realizada em DSP, e com menor custo final que uma implementação em circuito dedicado, ASIC. Logo, uma implementação conjunta *hardware*/*software* representa uma boa relação entre desempenho e custo.

Na FIGURA 3.3, apresenta-se novamente o diagrama geral do sistema em blocos, salientando-se a partição proposta entre implementações em *hardware* e em *software*. De um modo geral, o particionamento foi realizado pensando-se nas facilidades de implementação de cada um dos algoritmos discutidos no capítulo anterior, buscando-se um ganho de desempenho.



(a) – hardware/software

(b) – software

(c) – hardware

FIGURA 3.3 – Partição proposta entre *hardware* e *software* para o sistema.

O bloco (a) é composto pela etapa de pré-processamento. Este bloco é implementado em *hardware/software*. Parte deste bloco é implementada em *hardware* através de um FPGA, permitindo utilizar o máximo de operações em paralelo para se aumentar o desempenho do sistema, em termos de velocidade de processamento. Este mesmo bloco, possui partes implementadas em *software* objetivando uma maior flexibilidade do mesmo. O bloco (a) compreende a implementação das técnicas descritas na Seção 2.2.1.

O bloco (b) consiste da etapa relacionada à extração de feições. Esta etapa é implementada em *software* através de um DSP, o que permite operações em ponto flutuante. Uma implementação, destas operações em ponto flutuante, em *hardware* extrapolaria a capacidade atual dos FPGAs disponíveis. Operações em ponto fixo, com uma precisão limitada a poucos bits de precisão (em torno de 10 bits) poderia ser uma alternativa viável de implementar esta tarefa em FPGAs de alta densidade de integração. Mais adiante será discutido um exemplo deste tipo de implementação. Este bloco compreende a implementação das técnicas descritas na Seção 2.2.2.

Finalmente, o bloco (c) é responsável pela etapa de classificação das feições extraídas no bloco anterior. Este bloco é implementado em *hardware* permitindo alta velocidade de processamento pela realização de operações em paralelo. O bloco (c) compreende a implementação das técnicas descritas na Seção 2.2.3.

Para validar esta partição proposta, realizou-se um estudo comparativo de desempenho entre algumas tarefas realizadas por este trabalho. Para estas comparações foram utilizados o FPGA, o DSP D950 existente na placa APTIX, o simulador do DSP TMS320C6201 e o Pentium MMX. Os programas utilizados para os dois DSPs e para o Pentium foram escritos em linguagem C e compilados através dos compiladores fornecidos pelos fabricantes.

A TABELA 4 apresenta os resultados comparativos para a propagação do neurônio sem a função de ativação. As simulações foram executadas com duas configurações diferentes: pesos em ponto-fixo e em ponto-flutuante. A TABELA 5 mostra os

resultados obtidos para a filtragem gaussiana. Neste caso as simulações foram executadas com duas configurações diferentes: valores de máscara em ponto-fixo e em ponto flutuante.

TABELA 4 – Resultados comparativos para a propagação neuronal.

Descrição	**FPGA (25MHz) 40ns**	**DSP D950 (*) (143MHz) 7ns**	**DSP TMS320C6201 (*) (200MHz) 5ns**	**Pentium MMX (*) (233MHz) ≈4.3ns**
Pesos em ponto-fixo	200ns (5 ciclos)	7,7µs (1040 ciclos)	360ns (72 ciclos)	1,06µs (246 ciclos)
Pesos em ponto-flutuante	Não Implementado	53,3µs (7195 ciclos)	12,5µs (2514 ciclos)	1,9µs (441 ciclos)

*: O programa foi compilado a partir do código C.

TABELA 5 – Resultados comparativos para a filtragem gaussiana.

Descrição	**FPGA (25MHz) 40ns**	**DSP D950 (*) (143MHz) 7ns**	**DSP TMS320C6201 (*) (200MHz) 5ns**	**Pentium MMX (*) (233MHz) ≈4.3ns**
Valores em ponto-fixo	2,04µs (51 ciclos)	26µs (3512 ciclos)	8,3µs (1662 ciclos)	6,1µs (1418 ciclos)
Valores em ponto-flutuante	Não Implementado	180µs (24344 ciclos)	41µs (8196 ciclos)	12µs (2790 ciclos)

*: O programa foi compilado a partir do código C.

Estas tabelas demonstram que realizando-se as tarefas aqui propostas em *hardware* obtem-se um desempenho bem superior a uma implementação puramente em *software* dos mesmos algoritmos. Por meio destas tabelas observa-se que, quanto ao bloco de classificação (propagação neuronal), há um desempenho 14 vezes (72 ciclos de clock do DSP dividido por 5 ciclos de clock do FPGA) maior que o DSP TMS320C6201. Com relação ao bloco do filtro gaussiano há um desempenho 32 vezes (1662 ciclos de clock do DSP dividido por 51 ciclos de clock do FPGA) maior que o DSP TMS320C6201.

3.4 Descrição Funcional do Sistema

Nesta Seção será abordada a arquitetura para o sistema proposto. Será descrito cada bloco que compõe o sistema e a comunicação entre eles.

A FIGURA 3.4a apresenta o diagrama em blocos correspondente à implementação do sistema. Este diagrama apresenta uma parte implementada em FPGA e uma outra parte implementada em DSP. A parte implementada em FPGA contém quatro *cores* desenvolvidos: *Interface Core*, *ADM Core*, *Lines Core* e *Neural Core*. A parte implementada em software (DSP) contempla algumas tarefas da etapa de pré-processamento e a tarefa de extração de feições.

A FIGURA 3.4b apresenta o fluxo de dados/informações entre o FPGA e o DSP. Cada seta indica a existência de um fluxo de dados/informações entre os respectivos blocos. O conteúdo do retângulo tracejado indica a decomposição do *ADM Core*. A descrição deste *core* será realizada mais adiante.

A seguir, cada um destes blocos acima mencionados será comentado, bem como a implementação em *software* no DSP e o particionamento da memória externa.

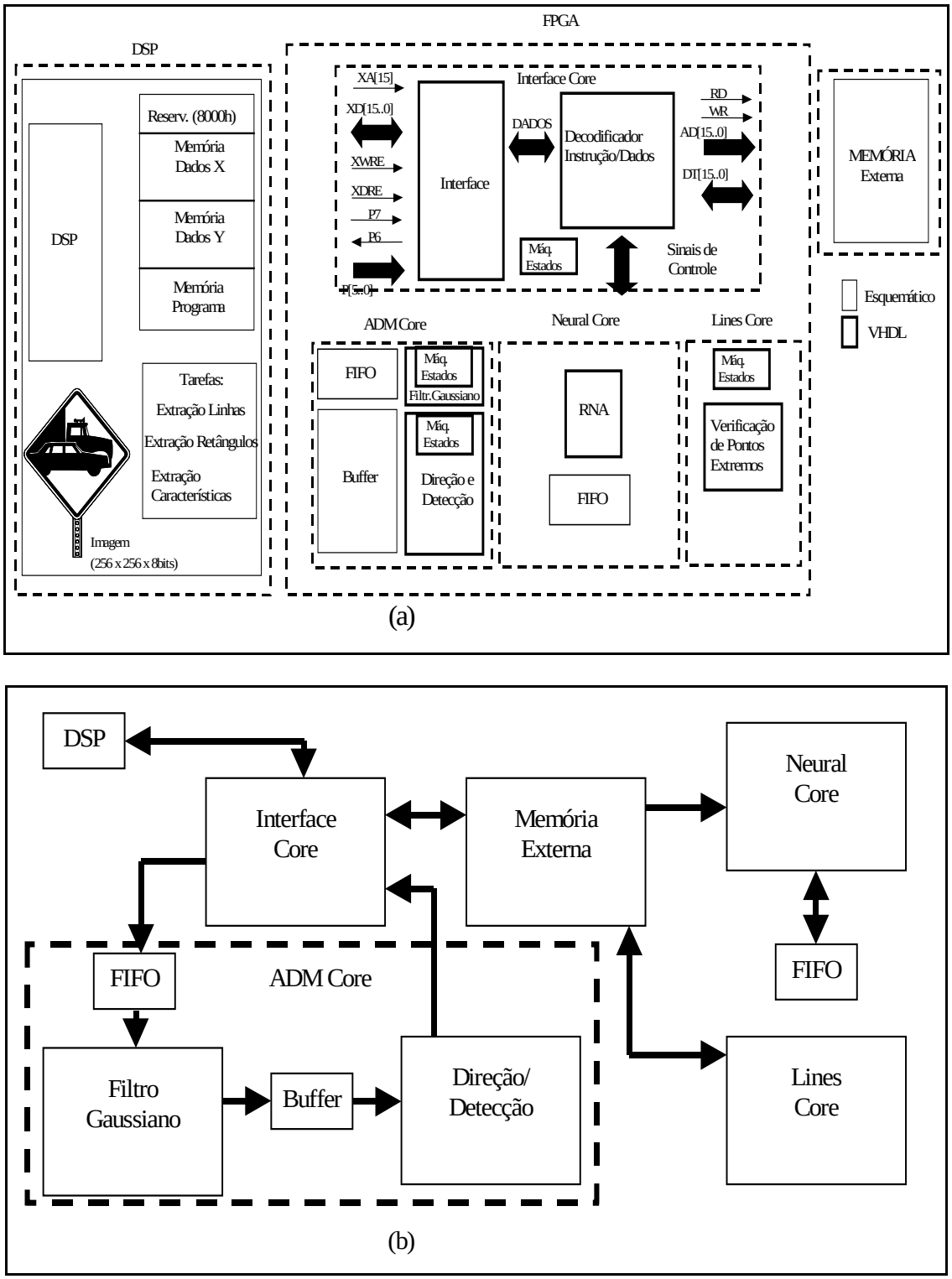


FIGURA 3.4- (a)Diagrama em blocos do sistema. (b)Fluxo de dados/informações.

3.4.1 Particionamento da Memória Externa

Como abordado na descrição do ambiente de prototipação APTIX, há 2 módulos de memória de 256K palavras de 4 bits cada um. Para a implementação do sistema proposto, uniu-se estes dois módulos, obtendo-se com isto, uma memória disponível de 256K palavras por 8 bits. Esta união vem a facilitar as tarefas de armazenamento e leitura de dados da memória, pois a imagem tratada é de 8 bits por pixel (256 tons de cinza).

A memória externa foi dividida em cinco segmentos de 64Kbytes. A imagem original é armazenada no primeiro segmento de memória externa (primeiros 64Kb). Com isto, o tamanho máximo da imagem a ser tratada é de 256x256 pixels de 8bits cada um. O segundo segmento é utilizado para armazenar a imagem segmentada, isto é, com as bordas detectadas e afinadas para um pixel.

O segmento seguinte, terceiro, é utilizado para auxiliar o processo de localização de *end-points*. Esta parte contém os endereços dos pixels, resultantes do processo de segmentação, que são diferentes de zero. O quarto segmento da memória externa, que não possui 64Kb, é utilizado para armazenar as coordenadas iniciais e finais de cada linha. Por fim, o quinto segmento, que também não possui 64Kb, é utilizado pelo bloco *Neural Core* para implementar a RNA. O quarto segmento mais o quinto segmento possuem 64Kb.

A FIGURA 3.5 apresenta o particionamento implementado sobre a memória externa disponível na placa de prototipação rápida APTIX.

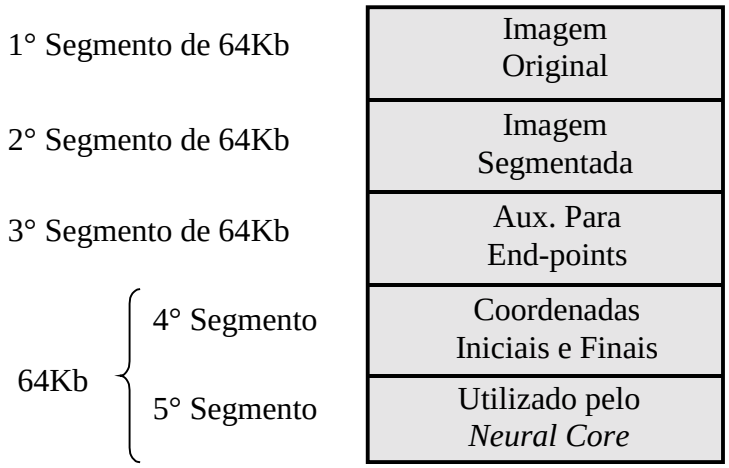


FIGURA 3.5- Diagrama do particionamento da memória externa.

Uma alternativa para aumentar o desempenho, ou seja, diminuir o tempo de acesso à memória, seria utilizar os dois bancos existentes como independentes (barramentos de endereços e dados independentes). Devido à restrição de tempo de projeto, esta alternativa, apesar de realizável, não foi implementada.

3.4.2 Interface Core

O bloco *Interface Core* realiza toda a comunicação necessária entre o FPGA e o DSP, DSP e a memória externa e parte da comunicação existente entre o FPGA e a memória externa.

O DSP efetua leitura de dados do FPGA por meio do barramento de dados do DSP. Para enviar dados ao FPGA, o DSP os envia pela sua porta paralela. Este método, que certamente não é o melhor, foi utilizado devido a problemas de sincronismo existentes entre o DSP e o FPGA. Esta falta de sincronismo deve-se ao fato de que o *software*, que realiza o roteamento dos sinais na placa APTIX, pode vir a definir caminhos diferentes para o barramento de dados e para o barramento de endereços que conectam o DSP ao FPGA para uma possível comunicação. Assim sendo, têm-se tempos de atraso diferentes, o que causa a falta de sincronismo no momento do envio de dados do DSP

ao FPGA. Como já comentado, a alternativa escolhida foi realizar a comunicação pela porta paralela, pois assim, pode-se utilizar parte dos bits da porta paralela para dados (6 bits) e a parte restante dos bits da porta paralela para sinais de sincronização (2 bits). Torna-se evidente que com isto tem-se a formação de um gargalo no sistema.

O DSP envia dados e/ou comandos. A TABELA 6 apresenta os comandos disponíveis no bloco *Interface Core*. Estes comandos são decodificados pela máquina de estados existente neste bloco.

TABELA 6 – Comandos disponíveis (DSP - FPGA).

Comandos	**Descrição**
01 xxxx	Escreve xxxx em uma variável auxiliar
10 xxxx	Concatenar xxxx com dado da variável auxiliar. Gravação deste dado resultante na memória externa (endereço é incrementado após o comando)
11 xxxx	Para ativar *Neural Core*
00 0001	Reseta ponteiro para memória externa
00 0010	FPGA lê memória externa e disponibiliza dado lido ao DSP (endereço é incrementado após o comando)
00 0011	Ajusta ponteiro para o 2° bloco de 64K da memória externa
00 0100	Ajusta ponteiro para o 3° bloco de 64K da memória externa
00 0101	Ajusta ponteiro para gravar 2° parte da imagem
00 0110	Ajusta ponteiro para gravar 3° parte da imagem
00 0111	Ajusta ponteiro para gravar 4° parte da imagem
00 1000	Habilita mostrar 1° bloco de 64k no vídeo VGA
00 1001	Desabilita mostrar 1° bloco de 64k no vídeo VGA
00 1111	Habilita blocos internos (os diferentes Cores) do FPGA

Como pode ser observado pelos comandos “00 0101”, “00 0110”, “00 0111” da TABELA 6, a imagem é dividida em 4 partes. Isto se deve ao fato que o DSP possui somente 8,5K palavras de 16bits para a memória de dados. Assim, a imagem é dividida em 4 partes iguais de 64 linhas x 256 colunas. A FIGURA 3.6 apresenta o programa que envia a parte desejada da imagem para o FPGA. Este programa deve ser utilizado para enviar a imagem original para o primeiro segmento da memória externa antes de se ativar os blocos implementados em FPGA (com exceção para o bloco *Interface Core* que está sempre ativado).

O programa da FIGURA 3.6 realiza um laço que percorre a memória de dados do DSP realizando a leitura de 16 bits, que representam 2 pixels da imagem. Cada dado lido do DSP é dividido nos dois pixels e, após, cada pixel é novamente dividido por dois, obtendo-se com isto palavras de 4 bits. Isto é necessário pois o DSP envia ao FPGA 4 bits de um dado pixel por vez. Os primeiros 4 bits são enviados do DSP ao FPGA por meio do comando “01 xxxx” e os 4 bits restantes são enviados através do comando “10 xxxx”. A função *comando(int)* é responsável pela geração dos sinais da porta paralela do DSP.

```
comando(0x0001); // gera sinais na porta paralela
for(x=0;x<0x0280;x++){ // tamanho da imagem
  pixel = *(arq + x); // leitura dos dois pixels da memória do DSP
  for(lsb=1;lsb<=2;lsb++){
    aux = 0;
    if (lsb == 1){
      aux = pixel >> 8; //aquisição dos 8 bits msb
      *p1 = aux & 0x00FF;
      aux = *p1;
      *p3 = (aux & 0x000F) + 0x0010; // composição do comando “01 xxxx”
      comando(*p3); // envia comando/dado
      aux = *p1;
      *p4 = ((aux & 0x00F0)>>4) + 0x0020; // comando “10 xxxx”
      comando(*p4); // envia comando/dado
    }
    else {
      aux = pixel & 0x00FF; //aquisição dos 8 bits lsb
      *p2 = aux;
      aux = *p2;
      *p3 = (aux & 0x000F) + 0x0010; composição do comando “01 xxxx”
      comando(*p3); // envia comando/dado
      aux = *p2;
      *p4 = ((aux & 0x00F0)>>4) + 0x0020; // comando “10 xxxx”
      comando(*p4); // envia comando/dado
    }
  }
}
```

FIGURA 3.6- Programa que envia uma determinada parte da imagem ao FPGA.

A comunicação existente entre os blocos internos do FPGA e o *Interface Core* é realizada pelo modo de requisição, isto é, se um bloco deseja se comunicar com o *Interface Core*, este envia um sinal requerendo um período de comunicação com o *Interface Core*. Assim que o *Interface Core* estiver habilitado a tratar esta comunicação, esse envia um sinal reconhecendo esta requisição solicitada pelo bloco requerente. Assim sendo, em cada requisição feita, o bloco interno espera pelo reconhecimento do *Interface Core*. Após o bloco ter sido reconhecido, este envia o comando ou espera por um dado. Há somente um bloco interno que envia comandos, este é o bloco Direção/Detecção que pertence ao bloco *ADM Core*. Os comandos disponíveis, ditos comandos internos, são:

TABELA 7 – Comandos internos.

Comandos	**Descrição**
0100	Envia dados para a memória externa (ponteiro é incrementado de uma unidade)
0101	Envia dados para a memória externa (ponteiro é incrementado de cinco unidades)

O Filtro Gaussiano, que faz parte do *ADM Core*, não envia comandos, apenas requisições. Estas requisições são entendidas pelo *Interface Core* como um comando fixo. Este comando indica para o *Interface Core* ler dados da memória externa e gravá-los na FIFO existente no *ADM Core* (FIGURA 3.4).

3.4.3 ADM Core

O *ADM Core* é composto pelos seguintes blocos: Filtro Gaussiano e Direção/Detecção. Além disto, este bloco possui um *buffer* e uma FIFO. Tal composição pode ser observada na FIGURA 3.4. Este *core* está organizado como um *pipeline*, visando um aumento do desempenho do sistema.

Entre os blocos Filtro Gaussiano e Direção/Detecção está implementado um *buffer* (uma memória interna de duplo acesso). Este possui uma capacidade de armazenamento equivalente a 1280 pixels (5 linhas de 256 colunas cada linha). A função deste *buffer* é armazenar os dados calculados pelo Filtro Gaussiano e disponibilizá-los para o bloco que calcula a direção relativa de cada pixel (Direção/Detecção). Assim, o bloco Direção/Detecção inicia após o Filtro Gaussiano ter processado os primeiros 5 pixels da quinta linha do *buffer*. Este bloco, Direção/Detecção, utiliza o mesmo *buffer* para efetuar os seus procedimentos e, após isto, escreve os resultados na memória externa.

Por fim, há uma pilha FIFO implementada entre as comunicações realizadas pelo Filtro Gaussiano e o *Interface Core*. Esta FIFO é responsável pela sincronização entre estes dois blocos evitando-se perdas de dados decorrentes das diferenças entre as frequências de operação de cada um dos blocos.

3.4.3.1 Filtro Gaussiano

Após o FPGA ter sido habilitado pelo DSP a iniciar o tratamento de solicitações internas, o primeiro bloco a solicitá-las é o Filtro Gaussiano. Como comentado, esta requisição indica ao *Interface Core* que este deve preencher a FIFO com os pixels da imagem que foram previamente armazenados no primeiro segmento da memória externa. O Filtro Gaussiano monitora a FIFO esperando um valor mínimo de trabalho e inicia o processamento quando este nível é alcançado. Enquanto a FIFO estiver vazia, o Filtro Gaussiano permanece em estado de espera (*wait state*). O pixel resultante deste processo de cálculo é armazenado no *buffer*.

O cálculo efetuado por este bloco utiliza uma adaptação do filtro apresentado no Capítulo 2. Os valores obtidos para os pesos deste filtro foram adequados para valores múltiplos da potência de dois, pois desta forma, se permitiu que a operação de multiplicação do peso pelo respectivo valor do pixel seja implementada por meio de operações de deslocamento. Com esta adequação, os valores dos pesos empregados são mostrados na TABELA 8. Na Seção 4.2 será apresentada uma análise desta alteração dos valores dos coeficientes de ponderação do filtro Gaussiano.

TABELA 8 - Coeficientes de ponderação *w(i,j)* adequados.

0,25	0,5	0,5	0,5	0,25
0,5	0,75	1	0,75	0,5
0,5	1	2	1	0,5
0,5	0,75	1	0,75	0,5
0,25	0,5	0,5	0,5	0,25

3.4.3.2 Direção/Detecção

A computação da direção relativa do pixel (baseada em uma janela de 5x5) e a detecção da borda (baseada em uma janela de 3x3) são tarefas implementadas pelo bloco Direção/Detecção. Este bloco é aplicado após o Filtro Gaussiano. Com base no resultado da direção obtida, o bloco Direção/Detecção realiza sucessivas leituras, no *buffer*, das respectivas posições (baseado em uma janela de 3x3) para produzir uma borda de largura igual a um pixel. Finalmente, este bloco envia o resultado ao *Interface Core*. Juntamente com este resultado segue um dos comandos internos (TABELA 7), o

qual indica ao bloco *Interface Core* como incrementar o seu ponteiro interno que controla os dados gravados no segundo segmento da memória externa.

O *Interface Core* envia este dado à segunda parte da memória externa. Este dado é armazenado em 4 bits, pois, como foi discutido anteriormente, as direções relativas são codificadas com valores 1, 2, 3 ou 4.

Por fim, se o dado recebido for diferente de zero, o *Interface Core* escreve no terceiro segmento da memória externa, o endereço em que este dado foi gravado no segundo segmento da mesma. Assim sendo, a gravação auxiliar visa facilitar o algoritmo implementado pelo *Lines Core*.

3.4.4 Lines Core

O *Lines Core* inicia após o término do bloco Direção/Detecção. Com a habilitação do *Lines Core* o *Interface Core* torna disponíveis os barramentos de dados e de endereços da memória externa. Tal tarefa pode ser entendida como um acesso direto à memória (DMA). Por um lado, o acesso direto a memória acelera o processo, mas por outro, ele indisponibiliza os barramentos de dados e de endereços aos demais blocos, impedindo uma possível paralelização entre o *Lines Core* e o envio dos dados ao DSP para posterior processamento.

Como comentado anteriormente, este algoritmo é aplicado sobre as bordas detectadas e devidamente afinadas, as quais se encontram armazenadas no segundo segmento da memória externa. Como um outro recurso, o *Lines Core* utiliza os dados armazenados no terceiro segmento da memória externa para acelerar o processo de busca e comparação dos dados.

Como descrito no bloco *ADM Core*, mais precisamente na Seção 3.4.3.2, os valores codificados referentes às direções relativas de cada pixel da imagem são armazenados no segundo segmento da memória, e o endereço em que foi armazenado este pixel é armazenado no terceiro segmento da memória. Pode-se dizer que os endereços do terceiro segmento são ponteiros para elementos do segundo segmento de memória.

Esta técnica foi implementada pois uma imagem que possui suas bordas devidamente detectadas e afinadas é considerada como uma matriz esparsa, logo, a maioria dos dados nela existentes são nulos. Com base nisto, tem-se um acesso indexado que conduz diretamente aos pixels que possuem uma direção relativa válida para o processamento.

Assim sendo, o *Lines Core* executa a leitura de um endereço a partir do terceiro segmento de memória de um dado que deve ser tratado. A partir deste endereço, o *Lines Core* realiza a leitura do respectivo dado (pixel base) a ser processado. Com este pixel base definido, este bloco inicia um processo de perseguição da mesma direção codificada neste pixel base. Esta perseguição é implementada por meio de uma matriz 3x3. Cada pixel visitado é marcado, setando-se o bit mais significativo, para evitar uma nova comparação. Este laço persiste até não existir mais pixels não marcados na imagem. A saída deste algoritmo nos fornece as posições iniciais e finais de cada linha existente no segundo segmento de memória. Estas posições são armazenadas no quarto segmento de memória e estes serão lidos pelo DSP para realizar os demais processos.

3.4.5 Neural Core

Nesta seção será descrita a técnica utilizada para a implementação da RNA do tipo MLP com aprendizado por *backpropagation*. Tal RNA será implementada na plataforma discutida neste trabalho. Este modelo de RNA foi escolhido pelo fato de que uma rede do tipo MLP é efetivamente um classificador universal, possuindo como ponto negativo a sua não adaptabilidade, isto é, o seu aprendizado não é incremental.

A operação de multiplica-acumula é exigida para se calcular o valor (*net*) de cada neurônio, Equação (19). Esta operação é o gargalo quando implementam-se RNAs em FPGA, pois ela requer um grande número de dispositivos lógicos programáveis. Após o *net* ter sido calculado, o valor de saída do neurônio (O_i) é obtido por meio de uma função de ativação. Uma função sigmóide ($f(net_i)$) pode ser utilizada para uma rede neural *feedforward* com aprendizado por *backpropagation* (FIGURA 3.7).

$$net_i = \sum_{j=0}^{n} w_{ij} * I_j \qquad (19)$$

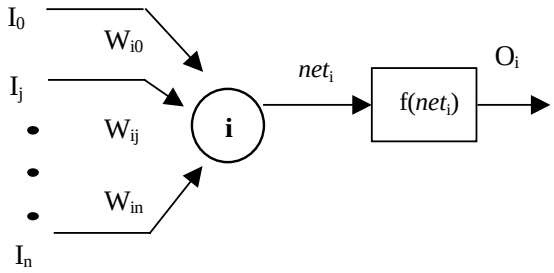


FIGURA 3.7 – Modelo do neurônio adotado.

Para reduzir o tamanho do circuito resultante, deve-se substituir as operações do tipo multiplica-acumula por simples operações de adição ou de subtrações. Os seguintes métodos podem ser utilizados:

- Multiplicação serial, onde somente um multiplicador é implementado [GSC94], com isto, obtém-se um circuito pequeno, mas com um desempenho pobre. Neste método, os padrões de entrada são serializados para se realizar a multiplicação por meio de bit e não por meio de palavra.
- Regra da aritmética distribuída. Esta implementação usa a estrutura interna do FPGA, isto é, as tabelas de consulta (*lookup tables*) [ROS98]. A desvantagem de tal método é que um dos números deve ser constante.
- Operações de deslocamento [CLO94]. Este método requer que os números envolvidos nas operações sejam múltiplos da potência de dois[1]. Este método possui um bom desempenho, e permite que problemas simples possam ter a fase de aprendizado implementada em FPGA. O problema-chave deste método é adequar a melhor precisão tal que o erro da saída permaneça mínimo [MOL99].

Neste trabalho, a multiplicação é realizada por meio do deslocamento de bits seguido por somas parciais. O objetivo é obter convergência com um mínimo de área implementável e de erro.

[1] Este intervalo causa assimetria entre os números adotados, por exemplo, 0.5, 0.25, 0.125, 0.0625.

No trabalho apresentado em [CLO94] a codificação assimétrica (valores codificados como sendo múltiplos da potência de dois) dos padrões de entrada (I_j na Equação (19)) e da função de ativação induz a erros nos valores de saída dos neurônios. Neste trabalho, assumiu-se um tipo de codificação diferente. Os números são representados com um formato em ponto-fixo de 4 bits, Equação (20). Nesta codificação, 1 bit é utilizado para o sinal (s) e os demais três bits são utilizados para a parte fracionária (f).

$$número = \begin{cases} -1^s * 0, f & para\ \ f \neq 0 \\ -1^s & para\ \ f = 0 \end{cases} \tag{20}$$

A TABELA 9 ilustra este método de codificação. Estes números estão codificados em binário e abrangem o intervalo fechado entre –1 e +1.

TABELA 9 – Codificação adotada para os padrões de entrada e função de ativação.

Valor Real	Codificação Proposta
-1	1000b
-0,125	1001b
-0,250	1010b
-0,375	1011b
-0,500	1100b
-0,625	1101b
-0,750	1110b
-0,875	1111b
0,125	0001b
0,250	0010b
0,375	0011b
0,500	0100b
0,625	0101b
0,750	0110b
0,875	0111b
1	0000b

A letra " b " após as seqüências de uns e zeros indica a notação binária

Os valores dos pesos sinápticos (w_{ij} na Equação (19)) são representados por um número em ponto-fixo de 10 bits, onde 1 bit é utilizado para representar o sinal, 4 bits são usados para representar a parte inteira e 5 bits representam a parte fracionária. Esta partição de bits para a codificação foi definida por meio de simulações. Com isto, esta codificação para os pesos sinápticos abrange a faixa de valores entre 15,96875 e -15,96875. Salienta-se que é muito importante esta faixa de valores estar bem definida, pois caso contrário, pode levar o neurônio à saturação, impedindo uma boa fase de propagação.

Embora as funções de ativação do tipo de limiar serem mais fáceis de se implementar, estas não são adequadas para as tarefas de reconhecimento de padrões. A função de ativação sigmóide pode ser implementada como uma função do tipo linear por partes (*piecewise*), com valores discretos fixos em função do valor *net* do neurônio [MOL99]. Neste trabalho a função de ativação é implementada por meio de *lookup tables*

(utilizando com isto a estrutura interna do FPGA). A FIGURA 3.8 apresenta a curva resultante da função de ativação utilizada.

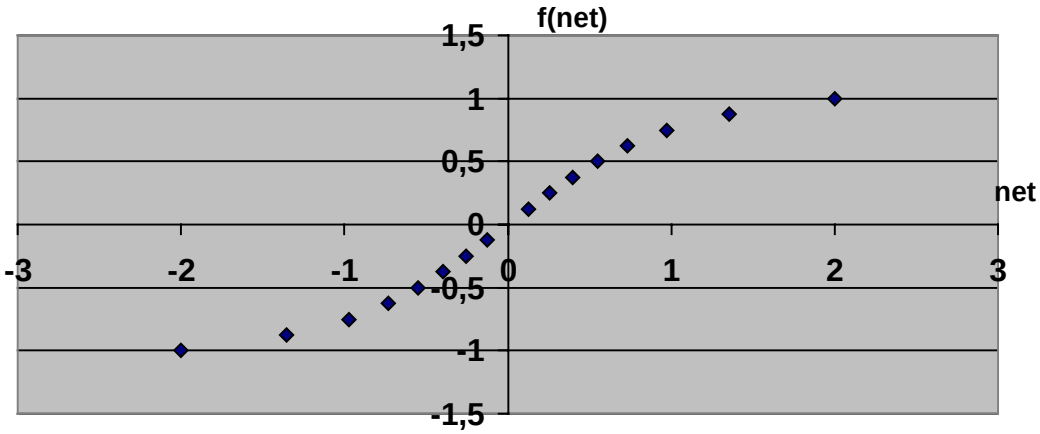


FIGURA 3.8 – Função de ativação.

Como foi mencionado anteriormente, a última parte do sistema, etapa de classificação, é realizada pela RNA. Esta RNA é implementada em *hardware* (FPGA) garantindo um bom desempenho devido às operações paralelizadas [MOL99].

A FIGURA 3.9 apresenta o diagrama em blocos de um único neurônio implementado. Este neurônio é controlado pelo DSP, o qual envia os padrões de entrada e os pesos sinápticos. Após o envio dos dados, o bloco *Weights* dispara o bloco *Delay*, sendo que o mesmo, por meio de um registrador de deslocamento, ocasiona um retardo no sistema. Este retardo (*delay*) fornece um tempo para o circuito combinacional do bloco *Neuron* realizar as operações de multiplicação e soma necessárias para a obtenção do valor do *net*. Após este tempo gerado pelo bloco *Delay*, o mesmo ativa o cálculo da função de ativação que está implementada no bloco *Neuron*. Por fim, o bloco *Delay* fornece um sinal indicando que a operação terminou e que o valor de resposta pode ser recuperado, isto é, lido do bloco *Neuron*.

A parte do circuito combinacional, utilizada para o cálculo do valor do *net*, é implementada por meio de um conjunto de 11 operações de deslocamento (responsáveis pela multiplicação entre os pesos e os padrões de entrada) e por um conjunto de 4 estágios de somadores, que são responsáveis pela operação de somatório existente na equação de cálculo do *net*.

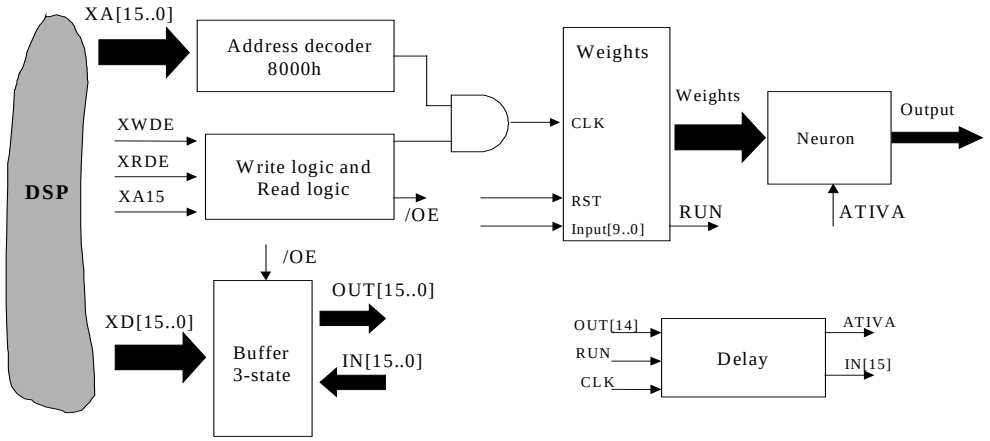


FIGURA 3.9 – Diagrama em blocos do neurônio utilizado para o processo de classificação.

A FIGURA 3.9, representa o processamento neural de somente um neurônio. Contudo, a etapa de classificação para as formas retangulares necessita de no mínimo uma RNA com duas camadas (camada escondida e camada de saída), sendo que a primeira, camada escondida, deve possuir em torno de 10 neurônios e a seguinte, camada de saída, deve possuir ao menos 4 neurônios. Assim sendo, a implementação da rede neural completa foi realizada por meio da técnica de *bit-slice*, técnica empregada para compor uma RNA a partir de um neurônio ou um pequeno conjunto de neurônios. Esta técnica pode ser melhor entendida por meio da seqüência de imagens mostradas na FIGURA 3.10. Cada imagem mostrada nesta figura representa uma mesma camada, sendo entendida como uma camada virtual. Em determinados estados da máquina de estados (FIGURA 3.10 itens a-f), o neurônio realmente implementado representa um determinado neurônio presente nesta camada virtual.

Este processamento foi implementado por meio de uma máquina de estados e memórias auxiliares que armazenarão os padrões de entrada, os pesos sinápticos enviados pelo DSP e os resultados intermediários existentes entre uma camada e outra.

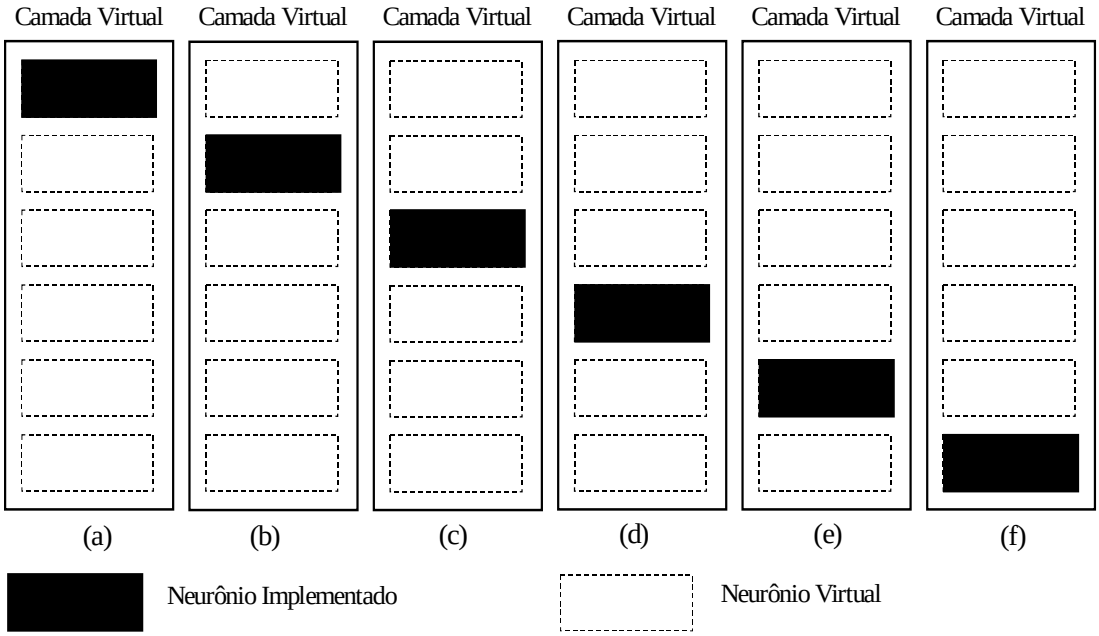


FIGURA 3.10 – Demonstração da técnica de *bit-slice.*

Após o DSP realizar a etapa de extração de feições, ele envia os padrões de entrada da RNA à memória externa. Estes dados são armazenados no quinto segmento de memória. Neste segmento encontram-se também os pesos sinápticos de cada um dos neurônios da RNA. Além destes dados, neste mesmo segmento, existe a descrição topológica da RNA a ser utilizada para o processo de classificação. Esta descrição topológica é a quantificação dos neurônios existentes em cada uma das camadas, sendo que esta descrição deve ser encerrada com o número 0 (zero). Este valor é utilizado pelo bloco de controle para verificar o fim da RNA.

Os valores resultantes, obtidos como respostas dos neurônios de uma dada camada intermediária, são armazenados em uma memória interna (FIFO), pois os mesmos são posteriormente utilizados como padrões de entrada da camada seguinte. A FIGURA 3.11 apresenta o bloco *Neural Core*.

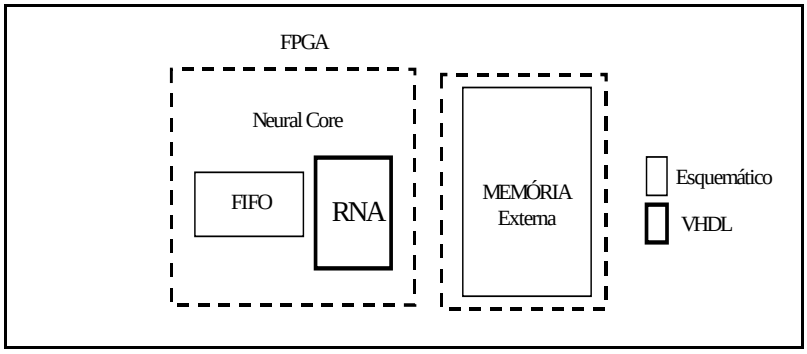


FIGURA 3.11 – Bloco *Neural Core.*

O módulo de classificação utiliza 1943 LE (*Logic Elements*), consumindo 38% do FPGA 10K100. Este grande consumo de elementos programáveis é devido ao processamento em paralelo, isto é, o neurônio processa 11 padrões de entrada em paralelo. Como a RNA está descrita como sendo uma máquina de estados que evolui conforme o pulso de clock, a ferramenta de síntese estimou uma frequência máxima de operação 20,40MHz. Supondo que a RNA tenha 14 neurônios (10 na camada escondida e 4 neurônios na camada de saída) a frequência obtida será de 1MHz. Se os demais blocos apresentados neste capítulo executassem suas tarefas em tempo-real (60 imagens por segundo, sendo que cada imagem representa um quadro), a RNA poderia classificar 16.667 formas retangulares extraídas de um determinado quadro.

3.4.6 DSP

O DSP executa as tarefas correspondentes à parte da extração de linhas (Aplicação de Restrições - tarefa descrita no item “j” da FIGURA 2.2), a extração da forma (tarefa descrita no item “k” da FIGURA 2.2) e por fim, a extração de feições.

A aplicação das restrições diz respeito à verificação da dimensão mínima e do ângulo máximo admitido. Estas restrições foram abordadas no Capítulo 2.

A extração da forma é realizada pela associação das linhas contidas no quarto segmento da memória externa em possíveis formas retangulares. Este procedimento é realizado pela procura de similaridades entre as linhas obtidas. As similaridades levadas em conta são: comprimentos das linhas, ângulos das linhas e posições iniciais relativas ao eixo *X*. Como o quarto segmento da memória externa somente contém as coordenadas iniciais e

finais, o DSP realiza o cálculo do comprimento da linha e também do seu ângulo relativo ao eixo horizontal.

Esta associação de linhas fornece dois lados da forma retangular. Assim, pode-se calcular os demais lados ou trabalhar em somente dois lados.

Além da extração da forma, o DSP realiza a extração de feições. Isto se deve à necessidade de um grande montante de operações matemáticas em ponto-flutuante. O trabalho [TRI96] apresenta alguns métodos para extração de feições. Estes valores calculados (vetor de características) são enviados ao *Neural Core* implementado em *hardware* para o processo de classificação.

Neste trabalho foram utilizados alguns métodos de extração de feições. É importante ressaltar que qualquer método pode ser empregado, pois este é implementado em *software* e, além disto, as formas retangulares já se encontram devidamente localizadas e definidas na imagem alvo. Supondo que um determinado método de extração de feições necessite de uma etapa diferente de processamento antes de ser aplicado, a mesma pode ser desenvolvida em *software*. Com isto, pode-se obter uma grande flexibilidade do sistema.

Por fim, existem métodos de extração de feições implementados em *hardware*, por exemplo, a implementação em *hardware* do método de momentos invariantes [MOR99] e DCT [NAY99].

3.5 Conclusões

Neste capítulo foram abordadas as técnicas empregadas na implementação da metodologia proposta e discutida no Capítulo 2. Para isto, este capítulo abordou o ambiente de prototipação empregado para se realizar testes de viabilidade e também a plataforma de prototipação utilizada para se implementar as técnicas descritas neste capítulo. Além disso, este capítulo apresentou a partição do *hardware*/*software* adotada, e por fim, a descrição funcional do sistema. Nesta última parte, são mencionados os blocos implementados bem como a comunicação entre esses blocos.

Pode-se salientar que as técnicas aqui apresentadas visam fornecer flexibilidade ao sistema para que um outro sistema, se adicionado de suas particularidades, possa utilizar os blocos aqui descritos.

4 Resultados

4.1 Introdução

Este capítulo abordará resultados significativos que comprovam a eficiência e flexibilidade da metodologia proposta para a solução do problema estudado neste trabalho. Primeiramente serão comentados resultados que mostram a viabilidade da implementação do filtro gaussiano em FPGA e, por fim, serão apresentados resultados do processo de extração de feições bem como de classificação das formas retangulares obtidas pelo método apresentado neste trabalho.

4.2 Filtro Gaussiano

Esta seção apresentará uma análise comparativa entre a aplicação do filtro gaussiano adotado neste trabalho e o filtro gaussiano sem a correção dos seus pesos. O objetivo desta seção é apresentar a viabilidade da substituição das operações de multiplicação envolvidas no processo de filtragem por simples operações de deslocamento. Com isto, ganha-se em desempenho e simplicidade da descrição final.

A FIGURA 4.1a apresenta a imagem original e a FIGURA 4.1b mostra a imagem filtrada por meio de um filtro gaussiano sem adequações nos seus pesos (pesos extraídos da TABELA 2 – seção 2.2.1).

(a) (b)

FIGURA 4.1 – Imagem Original (a) e Imagem Filtrada em *Software* sem Adequações (b).

A FIGURA 4.2 apresenta os resultados obtidos por meio de uma implementação em *hardware* (FIGURA 4.2a) e em *software* (FIGURA 4.2b) da imagem processada pelo filtro gaussiano com pesos adaptados (pesos extraídos da TABELA 8 – seção 3.4.3.1), respectivamente.

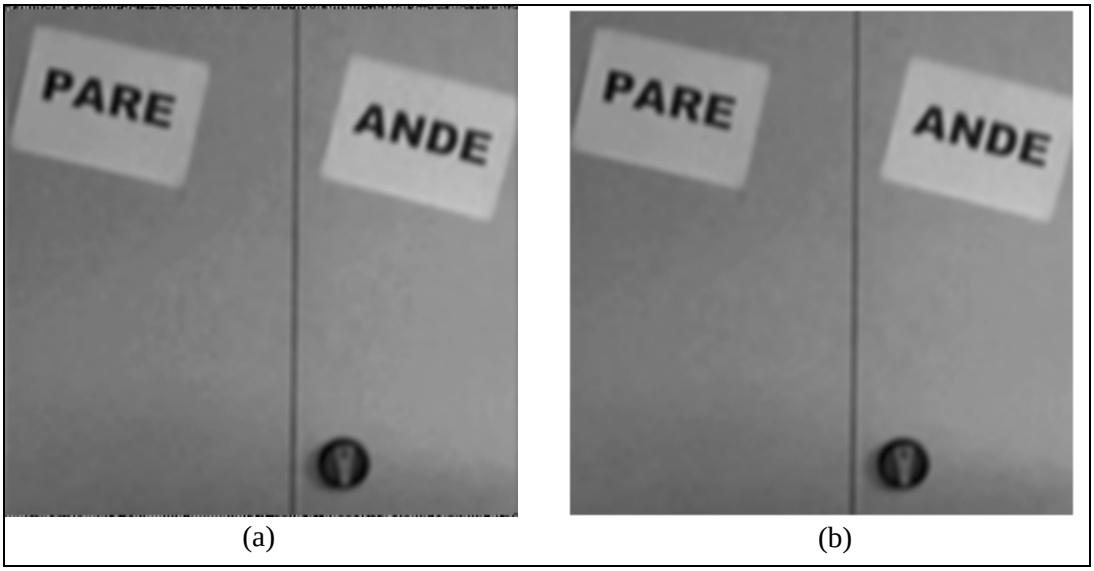


FIGURA 4.2 – Imagens Filtradas em *Hardware* (a) e em *Software* com Adequações (b).

A frequência obtida para o processamento realizado pela implementação em *hardware* (clock = 16MHz) do Filtro Gaussiano foi de 2,07 quadros/segundo, enquanto que a implementação do filtro executando em *software* em uma máquina AMD-K6-2 de 333MHz foi de 1,5 quadros/segundo. A FIGURA 4.3 apresenta o histograma obtido pela subtração entre a imagem filtrada em *software* sem adequação dos valores dos pesos (FIGURA 4.1b) que são aplicados no processo de filtragem e a imagem filtrada em *software* com adequação dos pesos (FIGURA 4.2b). A FIGURA 4.4 apresenta o histograma obtido pela subtração entre a imagem filtrada em *software* sem adequação dos valores dos pesos (FIGURA 4.1b) e a imagem filtrada em *hardware* (FIGURA 4.2a). Por ser uma operação de subtração entre as duas imagens, quanto mais próximo todo o histograma estiver do nível 0 (preto) mais semelhante é a imagem, indicando um menor erro decorrente da modificação imposta.

Por meio deste histograma, pode-se observar o pequeno erro introduzido pela adequação dos pesos, permitindo com isto um ganho de desempenho desta etapa. A degradação da resposta apresentada na FIGURA 4.4 se deve ao fato de que o algoritmo de filtragem inicia a partir da terceira linha da imagem, com isto, há uma quantidade de pixels com valores incorretos sendo contabilizada no histograma. Da mesma forma, as últimas duas linhas da imagem processada em *hardware* também não são tratadas.

Por fim, estes resultados conferem com o resultado obtido em [MOL2000a], onde foi realizada uma análise através do valor do PSNR (*Peak Signal Noise Ratio*).

FIGURA 4.3 – Histograma da diferença entre Imagens Filtradas por *Software* com e sem adequação.

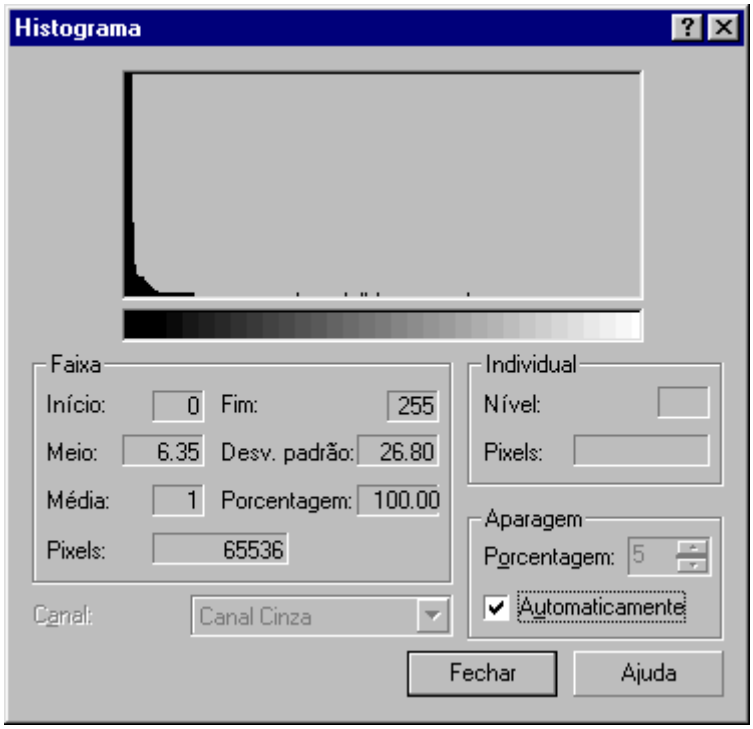


FIGURA 4.4 – Histograma da diferença entre Imagens Filtradas por *Software* sem adequação e por *Hardware.*

4.3 Resultados do Processo de Classificação

Nesta seção serão mostrados os resultados da etapa de extração de feições e da etapa de classificação. Estes resultados foram obtidos por meio de duas diferentes metodologias.

A primeira metodologia aborda a classificação da palavra contida na forma retangular localizada, enquanto que a segunda metodologia realiza a classificação baseada nas letras contidas na forma retangular obtida pelos processos de localização abordados neste texto. Os respectivos resultados serão apresentados, bem como o procedimento de obtenção dos mesmos. Esta seção irá demonstrar mais uma vez a flexibilidade existente neste trabalho.

4.3.1 Reconhecimento baseado em palavras

Esta primeira alternativa de extração de feições e de classificação das mesmas propõe a extração da forma retangular como um todo. Isto significa que todo o conteúdo desta forma retangular é submetido à extração de feições como sendo um único objeto.

Neste trabalho foram utilizados os momentos invariantes obtidos de cada amostra para a tarefa de aprendizado e reconhecimento. A RNA utilizada para o reconhecimento foi composta por 10 neurônios na camada escondida e 4 neurônios para a camada de saída. Não é necessário implementar a camada de entrada da RNA, visto que os dados já estão devidamente acondicionados. A TABELA 10 apresenta os resultados.

Para a tarefa de classificação, foram consideradas 38 formas retangulares representando os seguintes sinais: PARE, ANDE, DIR, ESQ. Entre estas 38 amostras, 20 amostras foram utilizadas para a etapa de aprendizado e as 18 amostras restantes foram empregadas para a tarefa de classificação (etapa de teste). É importante ressaltar que as 38 amostras utilizadas foram obtidas através de diferentes escalas e rotações.

A etapa de aprendizado foi realizada em *software* empregando-se ponto-flutuante. Para estes primeiros resultados, a etapa de teste também foi realizada em *software* empregando-se ponto-flutuante.

TABELA 10- Resultados obtidos.

Número de momentos usados	**Iterações da fase de aprendizado**	**Números de classificações erradas**	**%**
4	159303	10	55%
5	112788	10	55%
6	81675	10	55%
7	70735	10	55%

Para obter melhores resultados foram utilizados os momentos invariantes associados ao número de letras existente em cada palavra. Por exemplo, a palavra PARE possui 4 letras. Este valor que representa a quantidade de números existentes em uma determinada palavra foi obtido através da análise do histograma. A TABELA 11 apresenta os resultados.

TABELA 11- Resultados obtidos.

Característica	**Iterações da fase de aprendizado**	**Números de classificações erradas**	**%**
letras + 3 momentos	58764	5	27%
letras + 5 momentos	71408	6	33%

Pode-se observar que o melhor resultado obtido foi através do uso do número de letras contidas em uma determinada palavra associado a três momentos invariantes. Estes padrões de entrada foram então submetidos ao processo de classificação (etapa de teste) em ponto-fixo. A TABELA 12 apresenta os resultados.

TABELA 12- Resultados obtidos.

Números de classificações erradas			**%**		
Ponto-flutuante	Ponto-fixo Parte Fracionária : 4 bits	Ponto-fixo Parte Fracionária : 5 bits	Ponto-flutuante	Ponto-fixo Parte Fracionária : 4 bits	Ponto-fixo Parte Fracionária : 5 bits
5	5	6	27%	27%	33%

É importante observar a pequena variação do número de classificações erradas entre uma rede neural operando em ponto-flutuante e em ponto-fixo. Esta pequena variação habilita a implementação da rede neural em FPGA sem degradar a resposta do sistema. Além disso, o sistema ganhará em desempenho devido à natureza paralela do neurônio.

4.3.2 Reconhecimento baseado em letras

Esta segunda alternativa de extração de feições e de classificação das mesmas propõe o reconhecimento de cada letra contida na forma retangular localizada. Tal alternativa visa aumentar o percentual de classificações corretas em comparação com os resultados obtidos na seção anterior. Os resultados aqui demonstrados foram extraídos de [ERP2000].

A rede neural, representando a etapa de classificação, utilizada em todos os testes que serão apresentados, possui 10 neurônios na camada escondida e 9 neurônios na camada de saída. O número de neurônios da camada de entrada varia conforme o tamanho do vetor de características.

O treinamento da rede neural foi efetuado com a utilização das 27 letras sintéticas binárias mostradas abaixo (FIGURA 4.5):

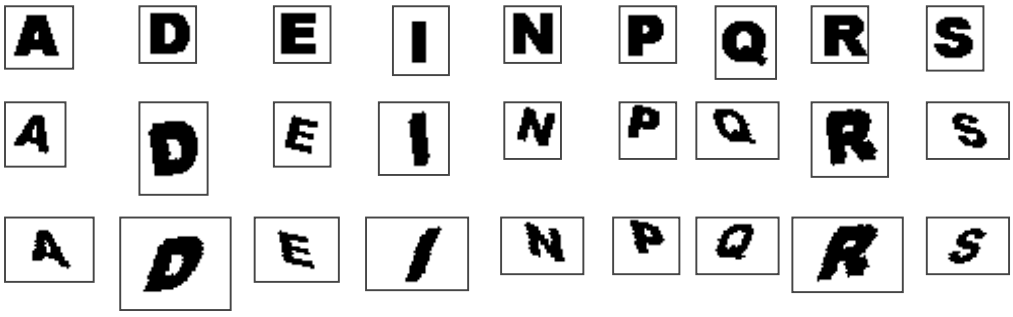


FIGURA 4.5 – Letras sintéticas binárias utilizadas no treinamento da RNA.

Estas letras foram escolhidas pois representam as letras das palavras que devem ser reconhecidas. Elas estão distribuídas em 10 imagens, que contêm 30 formas retangulares representando os sinais: PARE, ANDE, DIR e ESQ. Cabe salientar que a orientação e a escala destas imagens variam. As imagens estão mostradas na FIGURA 4.6.

(A)

(B)

(C)

(D)

(E)

(F)

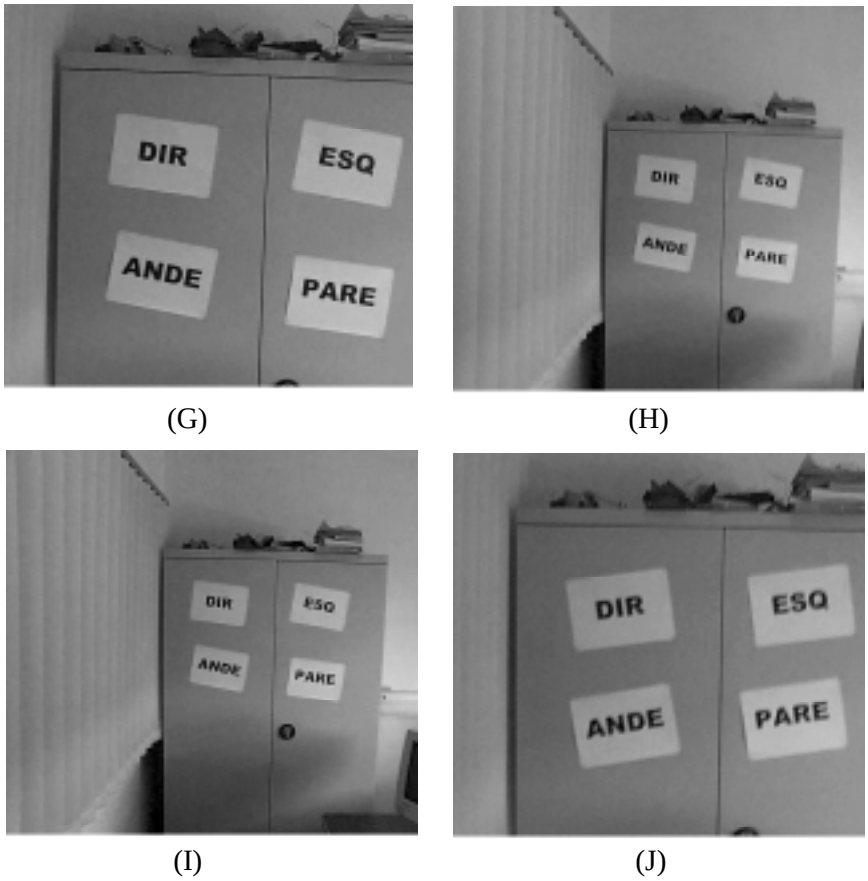


FIGURA 4.6 – Imagens a serem reconhecidas.

As formas retangulares foram localizadas através do método descrito no Capítulo 2. A técnica de detecção e separação dos caracteres contidos nas formas retangulares é uma abordagem baseada no procedimento de crescimento de regiões, utilizado na área de Processamento de Imagens.

Esta técnica é facilmente implementável pela simples inclusão de uma tarefa de extração de caracteres (letras) após a etapa de pré-processamento. Tal operação pode ser visualizada na FIGURA 4.7.

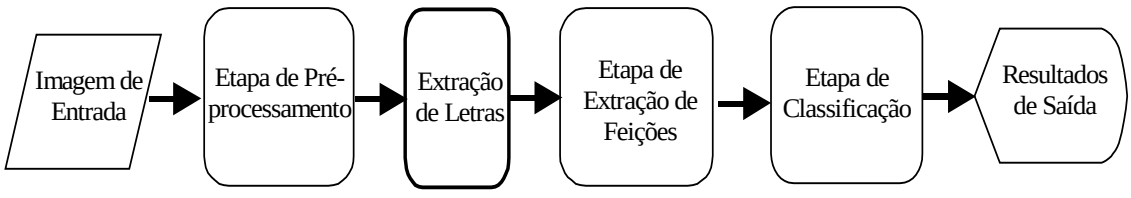


FIGURA 4.7 – Novo diagrama geral de blocos.

A técnica funciona da seguinte forma: depois de realizada a associação de retas em possíveis formas retangulares (item “k” da FIGURA 2.2), projeta-se uma linha que atravessa a imagem seguindo a mesma orientação destas retas. Isso é feito pois nesta linha sempre deverá existir pelo menos uma pequena parte de cada caractere, o que já é suficiente para o perfeito funcionamento da técnica. O primeiro ponto suficientemente

próximo a preto (neste trabalho foram considerados todos os pixels com nível de cinza menor que 121) é um indicativo de que está começando um novo caractere. Esse ponto então, é armazenado e sua vizinhança (conectividade-8) é verificada com o intuito de observar se algum dos vizinhos também está próximo de preto. Esse processo é recursivo e a cada vez que um pixel atende ao critério de similaridade esta posição é marcada para que o algoritmo não a percorra novamente. Isso é feito até que nenhum pixel atenda mais aos critérios. Ao final do processo, a rotina retorna uma matriz contendo a coluna e a linha de cada pixel pertencente a um mesmo caractere.

Este processo é demonstrado utilizando-se a FIGURA 4.6A. Note que nas imagens mostradas (FIGURA 4.8) o processo de detecção de retângulos já está realizado, sendo que sobre estas imagens é que as próximas etapas serão realizadas.

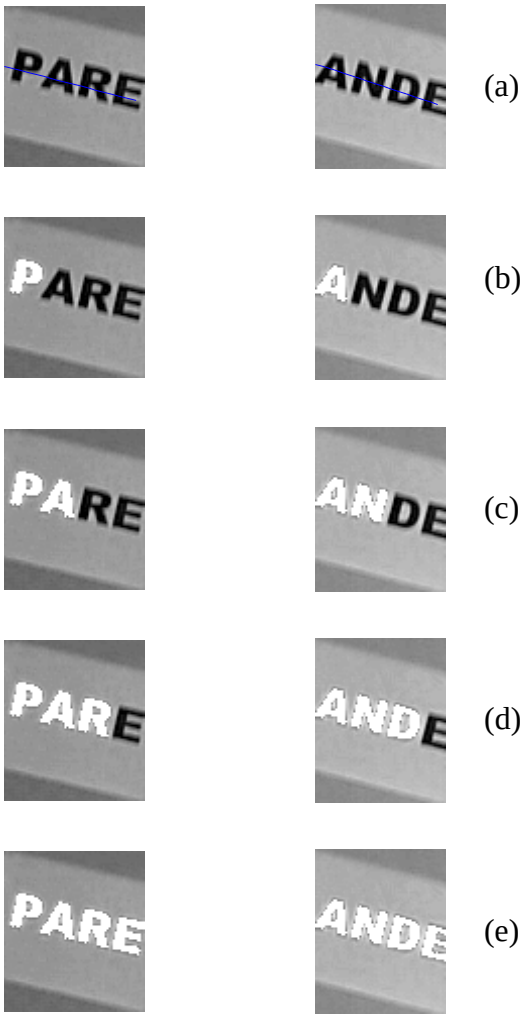


FIGURA 4.8 – Exemplo de separação dos caracteres.

A FIGURA 4.8a mostra a linha que cruza o centro da imagem. Nas demais imagens (4.8b até 4.8e) o processo de extração é mostrado. Esta técnica é válida para qualquer quantidade de caracteres.

Antes de os caracteres serem apresentados à RNA para a classificação, estes passam por uma rotina que ajusta o tamanho da subimagem a uma área um pouco maior do que o caractere propriamente dito. Isso é feito subtraindo-se a menor coluna encontrada na subimagem, descontando-se dois pixels (para deixar uma pequena borda) de todos as outras colunas. O processo ocorre de maneira equivalente para as linhas. As imagens resultantes desse processamento são binárias. O processo, considerando-se novamente a letra E da palavra ANDE (FIGURA 4.8e), é demonstrado abaixo (FIGURA 4.9).

(a) (b)

FIGURA 4.9 – Exemplo de ajustes efetuados nos caracteres.

Com esta imagem (FIGURA 4.9) é possível visualizar claramente qual o objetivo da rotina de ajuste. Também percebe-se que apesar de não terem sido estudados neste trabalho métodos de pré-processamento, algumas rotinas tiveram de ser desenvolvidas.

Por fim, é importante salientar que este algoritmo de extração de letras pode vir a ser desenvolvido em *hardware*, pois como pôde-se observar, a técnica realiza operações de comparação e busca em memória. Tais operações são excelentes candidatas a serem implementadas em *hardware.*

A seguir será apresentada uma análise dos resultados obtidos com alguns dos métodos de extração de feições existentes na literatura científica.

4.3.2.1 Extração de Feições e Classificação

O trabalho apresentado por [ERP2000] mostrou uma análise comparativa entre três métodos de extração de feições. Foram abordados os seguintes métodos: Zoning, Descritores de Fourier e Momentos Invariantes. Este trabalho teve como um dos seus objetivos, realizar um estudo destes métodos de extração de feições visando analisar qual método seria o mais adequado para a tarefa de classificar letras. Este trabalho utilizou as tarefas de localização aqui descritas no Capítulo 2.

Observando-se os resultados, pôde-se concluir que o método dos Descritores de Fourier é o mais apropriado para a tarefa de extração de feições deste trabalho. Para tanto, será abordado de forma sucinta os resultados dos outros dois métodos e após haverá uma seção específica para o método dos Descritores de Fourier.

As Tabelas 13 e 14 apresentam os resultados do método de Zoning. A TABELA 13 mostra os resultados para uma máscara de tamanho 4x3 enquanto que a TABELA 14 mostra os resultados para uma máscara de tamanho 4x4.

Pode-se observar nas Tabelas 13 e 14 que o desempenho obtido com a aplicação do método Zoning foi extremamente baixo. Isso ocorre principalmente devido ao fato de que este método não possui invariância à rotação.

Para uma melhora nos resultados deste método poder-se-ia utilizar duas alternativas principais. A primeira é a realização da compensação da rotação na imagem objetivando deixar todas as letras com a mesma orientação. Isso é possível pois o ângulo de rotação pode ser facilmente calculado. A outra alternativa seria a utilização de um conjunto de treinamento mais extenso, abordando diversas possibilidades de rotação para que a RNA pudesse discernir com mais exatidão.

TABELA 13 – Resultados do reconhecimento de letras com o método Zoning com máscara 4x3.

FIGURA 5.6		***Palavras a serem Reconhecidas***				***Acertos***	***Percentual***
		DIR	ESQ	PARE	ANDE		
(a)	**Palavras Reconhecidas**			PAES	SNSQ	3/8	37,50%
(b)				RPSI	SPPE	1/8	12,50%
(c)		SIQ	ESQ			4/6	66,67%
(d)		IIR	SSA			3/6	50,00%
(e)		QPP	SQI			0/6	0,00%
(f)		QQE	NSQ	PAQE	ANQN	7/14	50,00%
(g)		SPN	NND	SAQE	ANSS	4/14	28,57%
(h)		QSA	PQE	PASQ	ANQP	4/14	28,57%
(i)		ASP	QSS	PAPS	ANAP	5/14	35,71%
(j)		SSR	SSQ	SASE	ANSS	7/14	50,00%

TABELA 14 – Resultados do reconhecimento de letras com o método Zoning com máscara 4x4.

FIGURA 5.6		***Palavras a serem Reconhecidas***				***Acertos***	***Percentual***
		DIR	ESQ	PARE	ANDE		
(a)	**Palavras Reconhecidas**			DDRP	RNQA	2/8	25,00%
(b)				RPAD	RRDD	0/8	0,00%
(c)		DDR	PSD			3/6	50,00%
(d)		QAR	DPD			1/6	16,67%
(e)		DDS	ASD			2/6	33,33%
(f)		DRN	PSQ	PESS	DNDE	7/14	50,00%
(g)		QDA	EDQ	RRRS	DAQE	4/14	28,57%
(h)		DDD	PPA	PSAA	DNPP	3/14	21,43%
(i)		DSP	PSA	PSSS	DRDP	4/14	28,57%
(j)		RAR	DDD	RDPR	DDPD	1/14	7,14%

Nas Tabelas 15 e 16 pode-se observar os resultados provenientes da aplicação do método dos Momentos Invariantes. Estas tabelas contemplam a aplicação de 4 e 7 momentos invariantes, respectivamente.

Observando-se essas tabelas, constata-se uma taxa de reconhecimento extremamente baixa. Podemos observar que a presença dos 7 momentos tornou os resultados melhores do que com 4, mas não o suficiente para alcançar o próximo método a ser descrito.

TABELA 15 – Resultados do reconhecimento de letras com a utilização de 4 momentos invariantes.

FIGURA 5.6		***Palavras a serem Reconhecidas***				***Acertos***	***Percentual***
		DIR	ESQ	PARE	ANDE		
(a)	**Palavras Reconhecidas**			PPDD	PQQD	1/8	12,50%
(b)				PPDD	AQQD	2/8	25,00%
(c)		QDD	QQQ			1/6	16,67%
(d)		QQD	DDQ			1/6	16,67%
(e)		QQD	DQQ			1/6	16,67%
(f)		QDD	DQQ	QPDQ	ANQD	3/14	21,43%
(g)		QQD	DQQ	PPDD	ANQD	4/14	28,57%
(h)		QQD	DNQ	QPDQ	ANQD	3/14	21,43%
(i)		QID	DQQ	QPDQ	AQQD	3/14	21,43%
(j)		QID	DDQ	PPDD	ARQD	4/14	28,57%

TABELA 16 – Resultados do reconhecimento de letras com a utilização de 7 momentos invariantes.

FIGURA 5.6		***Palavras a serem Reconhecidas***				***Acertos***	***Percentual***
		DIR	ESQ	PARE	ANDE		
(a)	**Palavras Reconhecidas**			NASE	QNQQ	3/8	37,50%
(b)				NARE	ANQS	5/8	62,50%
(c)		QIN	SSS			2/6	33,33%
(d)		QDN	ESQ			3/6	50,00%
(e)		QRN	ESQ			3/6	50,00%
(f)		QRN	ESQ	NQNS	ANQE	6/14	42,86%
(g)		QIR	ESQ	NQNE	ANQE	9/14	64,29%
(h)		QRS	SPQ	NQNS	ANQS	3/14	21,43%
(i)		QDN	SSQ	NQNE	ANQS	5/14	35,71%
(j)		QRR	ESQ	NQNS	ANQE	7/14	50,00%

4.3.2.1.1 Método dos Descritores de Fourier

Esta seção aborda o método que apresentou o melhor desempenho entre os três métodos de extração de feições utilizados para estudo. Para o funcionamento deste método foi necessário o desenvolvimento de uma rotina que extrai o contorno de cada caractere. Este contorno é somente externo, não abrangendo o interior das letras (FIGURA 4.10). A função desenvolvida retorna o contorno em uma forma complexa sendo que a parte real representa as colunas e a parte imaginária representa as linhas da imagem.

(a) (b)

FIGURA 4.10 – Exemplo da extração de contorno utilizada pelos Descritores de Fourier.

O número de descritores indica quantos coeficientes da DFT serão utilizados. Estes coeficientes são normalizados pelo primeiro coeficiente. Além disso, somente a magnitude é utilizada, descartando-se a fase.

TABELA 17 – Resultados do reconhecimento de letras com a utilização de 5 descritores.

FIGURA 5.6		***Palavras a serem Reconhecidas***				***Acertos***	***Percentual***
		DIR	ESQ	PARE	ANDE		
(a)	**Palavras Reconhecidas**			PARE	ANDE	8/8	100,00%
(b)				PARE	ANDE	8/8	100,00%
(c)		DIR	ESQ			6/6	100,00%
(d)		DIR	ESQ			6/6	100,00%
(e)		DIR	ESQ			6/6	100,00%
(f)		DIR	ESQ	PARE	ANDE	14/14	100,00%
(g)		DIR	ESQ	PARE	ANDE	14/14	100,00%
(h)		DIR	ERQ	PADE	ANDE	12/14	85,71%
(i)		DIR	ESQ	PARE	ANDE	14/14	100,00%
(j)		DIR	ESQ	PARE	ANDE	14/14	100,00%

TABELA 18 – Resultados do reconhecimento de letras com a utilização de 6 descritores.

FIGURA 5.6		***Palavras a serem Reconhecidas***				***Acertos***	***Percentual***
		DIR	ESQ	PARE	ANDE		
(a)	**Palavras Reconhecidas**			PARE	ANDE	8/8	100%
(b)				PARE	ANDE	8/8	100%
(c)		DIR	EQQ			5/6	83,33%
(d)		DIR	EQQ			5/6	83,33%
(e)		DIR	EQQ			5/6	83,33%
(f)		DIR	EQQ	PARE	ADDE	12/14	85,71%
(g)		DIR	EQQ	PARE	ANDE	13/14	92,86%
(h)		DQR	ESQ	PAAE	ANDE	12/14	85,71%
(i)		DIR	ESQ	PPRE	ANDE	13/14	92,86%
(j)		DIR	EQQ	PARE	ANDE	13/14	92,86%

TABELA 19 – Resultados do reconhecimento de letras com a utilização de 10 descritores.

FIGURA 5.6		***Palavras a serem Reconhecidas***				***Acertos***	***Percentual***
		DIR	ESQ	PARE	ANDE		
(a)	**Palavras Reconhecidas**			PPRE	ANDS	6/8	75,00%
(b)				PARE	ANDE	8/8	100,00%
(c)		DIR	NSD			4/6	66,67%
(d)		DIR	ESQ			6/6	66,67%
(e)		DIR	ESQ			6/6	100,00%
(f)		DIR	NSQ	QARE	ANDE	12/14	85,71%
(g)		DIR	ESQ	PPRN	PNDE	11/14	78,57%
(h)		DDR	ESQ	PARN	ANDD	11/14	78,57%
(i)		DIR	NSQ	PARE	ANDS	13/14	92,86%
(j)		DIR	ESQ	PPRN	ANDE	12/14	85,71%

Observando-se as Tabelas 17, 18 e 19 constata-se a excelente taxa de reconhecimento obtida pelo método dos Descritores de Fourier. Também é possível observar que a utilização de 5 descritores é a melhor pois somente em um caso não atingiu 100% de reconhecimento.

Isso indica que apenas 5 coeficientes já são o suficiente para capturar a forma de um objeto simples como são as letras dos sinais. Entretanto, quando se aumenta o número de descritores objetivando aumentar os detalhes finos das formas tem-se um decaimento na taxa de reconhecimento, apesar de os valores obtidos não serem péssimos. Este decaimento pode ser devido ao método considerar ruídos na imagem.

4.4 Conclusões

O sistema completo está consumindo 4600 (≈ 92100 gates) Logic Elements (10K100 da Altera possui 4992 Logic Elements). A ferramenta de timing tem fixado a frequência máxima de operação em 9,2MHz para o FPGA 10K100gc503-4. Utilizando-se o FPGA 10K100gc503-3 a ferramenta de timing estimou uma frequência máxima de 11,2MHz. Como o tamanho da imagem utilizada é de 256x256 (65536 pixels/imagem) e o gargalo do sistema é o *ADM Core* (mais precisamente o Filtro Gaussiano) com 51 ciclos de clock, o sistema tem um desempenho de 2,7 quadros/seg em um FPGA do tipo 10K100gc503-4. Usando-se dispositivos mais rápidos como Virtex (da Xilinx) ou APEX (da Altera) a frequência de operação pode facilmente ser melhorada, permitindo chegar ao processamento em tempo-real.

Além disto, com FPGAs de maior densidade, obter-se-á um maior paralelismo no nível do Filtro Gaussiano. Com isto, teremos um menor número de ciclos de clock necessário e conseqüente aumento no desempenho do sistema.

Com relação à etapa de extração de feições, tem-se que a primeira alternativa apresentada é menos eficiente que a segunda, pois a mesma se utiliza do reconhecimento feito com base nos caracteres e não em palavras. Com a utilização de letras diminuiu-se a incidência de ruídos existentes na área em que estão as respectivas palavras.

Por fim, dentre os métodos apresentados na segunda alternativa, o método dos Descritores de Fourier obteve os melhores resultados. Evidencia-se, também, que este método possui um custo computacional mais elevado que os demais.

5 Conclusões

Este trabalho propõe uma metodologia para a implementação de sistemas voltados à área de visão computacional, tendo como estudo de caso um sistema completo para a localização e classificação de formas retangulares.

No decorrer do mesmo, foram fundamentados todos os módulos necessários bem como a comunicação entre os mesmos. O sistema está completamente funcional em *software*. A implementação dos blocos em *hardware* está em progresso, tendo sido todos os blocos implementados de forma isolada. Contudo, o bloco *ADM Core* está implementado em conjunto, pois foi necessário para a simulação do fluxo de informações entre os seus componentes (Filtro Gaussiano e Direção/Detecção) e entre este bloco e o *Interface Core*.

Além disso, foram apresentados resultados abrangendo a fase de classificação, levando em consideração alguns métodos de extração de feições. Estes métodos demonstram a flexibilidade do sistema, pois estes são implementados em *software*. Dentre os métodos aqui apresentados, o método dos Descritores de Fourier apresentou resultados muito satisfatórios, resultando em quase 100% de acertos em todas as placas.

Como um dos objetivos que foram atingidos por este trabalho, está a adoção de uma RNA do tipo MLP como o bloco responsável pela classificação. Esta rede apresenta bons resultados que dependem, de forma clara, de como é feita a extração de feições. Além disto, a forma em que a mesma foi implementada (*bit-slice*) permite a sua configuração. Tem-se como restrições a quantidade de padrões de entrada que cada neurônio pode possuir. Com o avanço da tecnologia, esta quantidade de padrões de entrada pode ser ampliada, bem como a quantidade de neurônios, realizando, com isto, um *bit-slice* de mais de um neurônio por vez.

A RNA proposta está implementada em FPGA e é viável. Salienta-se que esta viabilidade se dá somente durante a fase de classificação, pois a solução adotada não permite que seja realizada a fase de aprendizado em FPGA. Acredita-se que para a fase de aprendizado ser feita em FPGA, a RNA deve possuir uma resolução de bits mais elevada, o que pode ser inviabilizado por causa do espaço disponível nos FPGAs. Levando-se em conta que os FPGAs de hoje suportam microprocessadores implantados, é aconselhável utilizar estes para a fase de aprendizado da RNA. Por fim, o bloco de classificação possui um tempo de resposta adequado a operações de alto-desempenho.

Para validar a proposta deste trabalho foram apresentados resultados que demonstram o desempenho superior desta implementação. O bloco de classificação possui um desempenho 14 vezes (72 ciclos do DSP dividido por 5 ciclos do FPGA) maior que o DSP TMS320C6201 e o filtro gaussiano tem um desempenho 32 vezes (1662 ciclos DSP dividido por 51 ciclos FPGA) maior que o DSP TMS320C6201. Foi levado em consideração o número de ciclos, pois não temos como comparar freqüências de clock que são muito diferentes. A questão da programação utilizada, do DSP e do microprocessador, pode levantar uma questão com relação a estes valores obtidos. Contudo, o número de ciclos da implementação em *hardware* do Filtro Gaussiano pode também ser reduzido, visto que não foi empregado nenhum método para paralelizar esta operação. Com a paralelização do Filtro Gaussiano, que é um dos gargalos do sistema,

poderia-se obter tempos de resposta muito melhores. Esta paralelização depende tão somente do *hardware* disponível.

A frequência máxima do sistema, 2,7 quadros por segundo, certamente será melhorada com a utilização de FPGAs do tipo Virtex (Xilinx) ou APEX (Altera), pois a frequência máxima de operação é dependente da tecnologia.

A metodologia apresentada não contempla a localização de formas oclusas. Para tanto, seriam necessários outros métodos de tratamento, tais como os apresentados em [LOW99], [TSA99], [YAN98], e [COC95].

A metodologia aqui proposta pode ser vista como o início de uma biblioteca de funções de processamento de imagens implementadas em FPGA. Estas funções podem ser incorporadas em outras visando a solução de um determinado problema. Salienta-se a portabilidade dos blocos descritos, visto que foram implementados em VHDL. Com estas funções, pode-se iniciar um poderoso ambiente, para o desenvolvimento de sistemas de processamento de imagens customizado.

Com o uso de FPGAs pode-se empregar o conceito da reconfiguração para gerar outros sistemas que utilizem o mesmo fluxograma de pré-processamento e classificação. Por exemplo, o algoritmo de extração de formas retangulares pode ser alterado para extrair qualquer forma geométrica. Esta alternativa seria ideal para a detecção e classificação de sinais de trânsito, habilitando um sistema de auxílio ao motorista.

Por fim, este sistema pode vir a ser implementado em um ASIC dedicado ou um FPGA (por exemplo Excalibur - Altera), caracterizando assim, um *system-on-a-chip* (SoC) ou um *system-on-a-programmable-chip* (SoPC) para a área de processamento de imagem. A FIGURA 5.1 apresenta uma versão provável para SoC ou SoPC.

O desempenho necessário para processamento de tempo-real é obtido pela implementação de algumas das operações críticas em paralelo, utilizando os FPGAs. Esta implementação *hardware*/*software* constitui um sistema autônomo, capaz de executar todas as tarefas necessárias para, por exemplo, a localização e classificação de formas retangulares que constitui o estudo de caso implementado neste trabalho.

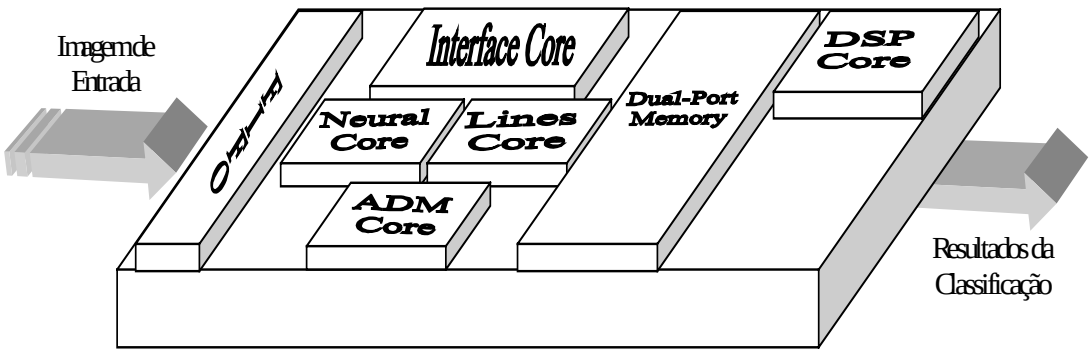


FIGURA 5.1 – Provável versão SoC.

Para completar este sistema, pode-se conectá-lo a um circuito integrado CIS (*CMOS image sensor*) obtendo com isto a inclusão da tarefa de aquisição das imagens.

Comparando-se esta metodologia com os trabalhos afins apresentados no Capítulo 2 (Seção 2.3 – TABELA 3) pode-se concluir que: embora tenha se obtido um baixo desempenho com relação ao tempo de reposta, a metodologia apresentada neste trabalho pode ser implementada em dispositivos maiores e mais rápidos permitindo com isto uma sensível melhora no seu desempenho. Este trabalho apresentou uma metodologia fácil de se implementar em uma partição *hardware* e *software* e, que permite uma extensão para outras aplicações. Além disto, esta metodologia propõe um sistema completo em uma partição *hardware* e *software* de dimensões reduzidas. Isto facilita o desenvolvimento de sistemas embarcados. Com relação a qualidade das respostas obtidas pela etapa de classificação, este trabalho apresentou um método que fornece uma alta taxa de acerto. Com isto, este trabalho demonstra a flexibilidade desejada, pois é permitida a troca do método de extração de feições, habilitando com isto a classificação de outros objetos que são melhores classificados por outros métodos de extração de feições. Ainda em relação a flexibilidade deste trabalho, a forma extraída pode ser alterada mantendo grande parte do sistema inalterado.

Com base na TABELA 3 pode-se observar que o trabalho com maior afinidade é o trabalho apresentado por [MIN98]. Este trabalho se baseia no conhecimento da distância do objeto, o que facilita a etapa de pré-processamento. Além disto, em [MIN98] não há o emprego de uma técnica para extração de caraterística visando uma diminuição da quantidade de dados e por conseguinte uma diminuição dos padrões de entrada da rede neural. Por fim, este trabalho, [MIN98], utiliza uma grande rede neural que realiza todo a etapa de classificação em *software*.

Já o trabalho apresentado aqui, possui uma etapa de pré-processamento adequada para localizar formas retangulares com uma variedade de distâncias permitidas. Além disto, este trabalho apresenta uma alternativa de métodos de extração de feições que fornece flexibilidade ao sistema e uma diminuição do número de padrões de entrada da rede neural, ganhando com isto, um maior desempenho em termos de velocidade de processamento. Por fim, a rede neural está implementada em *hardware*.

Para finalizar, este trabalho gerou uma série de publicações que abrangem as diferentes etapas do seu desenvolvimento. No Anexo I está apresentada a lista destas publicações.

Anexo 1 Publicações

Publicação	Título	Data	Local	Pág.
Multi-Conference on Systemics, Cybernetics and Informatics (SCI2000)	Codesign of Fully Parallel Neural Network for a Classification Problem	23-26 July	Florida/ USA	601-605
Computação Reconfigurável: Experiências e Perspectivas (CORE2000)	Estudo da viabilidade de implementação de um sistema de localização e reconhecimento de objetos com uso de RNAs implementadas em FPGA	10-11 August	Marília/ SP Brazil	226-235
13th Symposium on Integrated Circuits and Systems Design. (SBCCI2000)	Design of a Classification System for Rectangular Shapes Using a Co-Design Environment	18-23 September	Manaus/ Brazil	281-286
XV Conference on Design of Circuits and Integrated Systems (DCIS2000)	A fast prototyping neural network model for image classification	21-24 November	Montpellier/ France	836-841
IP Based Design'2000 Workshop (IWLA2000)	Neural Network prototyping approach for Localization and Classification in Image Processing	14 - 15 December	Grenoble/ France	
9th ACM International on Field-Programmable Gate Arrays (FPGA2001)	System Prototyping dedicated to neural network real-time image processing	11-13 February	Monterey/ California	

Bibliografia

[ADO99] ADORNI, G.; GORI, M.; MORDONINI, M. Just-in-time Landmarks Recognition. **Real-Time Imaging**, [S.l.], v.5, p.95-107, 1999.

[ADA97] ADÁRIO, Alexandro M. S. **Uma Implementação em FPGA de um Processador de Vizinhança para Aplicação em Imagens Digitais**. São Paulo: Instituto de Informática, UNICAMP, 1997. 73 p. Dissertação de Mestrado.

[ALO91] ALOIMONOS, Y.; ROSENFELD, A. Computer Vision. **Science**, [S.l.], v.253, p.1249-1254, 1991.

[ALZ97] ALZAHRANI, F.M.; CHEN, T. A Real-Time Edge Detector: Algorithm and VLSI Architecture. **Real-Time Imaging**, [S.l.], v.3, p.363-378, 1997.

[APT99] APTIX Corporation. Disponível em : <www.aptix.com>. Acesso em : 11 jul. 1999.

[BET99] BETZ, Vaughn et al. **Architecture and CAD for Deep-Submicron FPGAs**. USA : Kluwer Academic Publishers, 1999.

[BOS99] BOSI, B.; BOIS, G.; SAVARIA, Y. Reconfigurable Pipelined 2-D Convolvers for Fast Digital Signal Processing. **IEEE Trans. on Very Large Scale Integration (VLSI) Systems**, New York, v.7, n.3, 1999.

[CHA99] CHABAT, F.; YANG, G.Z.; HANSELL, D.M. A corner orientation detector. **Image and Vision Computing**, [S.l.], v.17, p.761-769, 1999.

[CHO92] CHOUDHARY, A.; RANKA, S. Parallel Processing for Computer Vision and Image Understanding. **Computer,** Los Alamitos, v.25, n.2, p.7-10, 1992.

[CLO94] CLOUTIER, Jocelyn; SIMARD, P.Y. Hardware Implementation of the Backpropagation without Multiplication. In: INTERNATIONAL ON MICROELECTRONICS FOR NEURAL NETWORKS AND FUZZY SYSTEMS, 4., 1994, Turim, It. **Proceedings...** Los Alamitos: IEEE Computer Society, 1994. p.46-55.

[COC95] COCQUEREZ, J.; PHILIPP, S. et al. **Analyse d'images:** filtrage et segmentation. Paris: Masson, 1995.

[DAL97] DALLAIRE, S.; TREMBLAY, M.; POUSSART, D. VLSI Architectures for the Embedded Extraction of Dominant Points on Object Contours. In: COMPUTER ARCHITECTURES FOR MACHINE PERCEPTION, 1997, Como, It. **Proceedings ...** [S.l.:s.n.] , 1997. p.20-22.

[ERP2000] ERPEN, Luis Renato. **Estudo de Métodos de Extração de Feições**. Porto Alegre: CPGCC da UFRGS, 2000. 35p. Trabalho Individual.

[FAC96] FACON, J. **Morfologia Matemática :** teoria e exemplos. Curitiba: Ed. Universitária Champagnat da PUC-PR, 1996.

[FIG98] FIGUEIREDO M.; GLOSTER C. Implementation of a Probabilistic Neural Network for Multi-Spectral Image Classification on a FPGA CustomComputing Machine. In: BRAZILIAN SYMPOSIUM ON NEURAL NETWORKS, SBRN, 5., 1998, Belo Horizonte. **Proceedings...** Los Alamitos: IEEE Computer Society, 1998.

[FIG99] FIGUEIREDO M. et al. Extending NASA's Data Processing to Spacecraft. **Computer**, New York, v.32, n.6, p.115-118, 1999.

[FOR99] FORESTI, G.L. Object Recognition and Tracking for Remote Video Surveillance. **IEEE Trans. on Circuits and Systems for Video Technology**, [S.l.], v.9, n.7, p. 1045-1062, 1999.

[FRE77] FREEMAN, H. On the encoding of arbitrary geometric configurations. **IEEE Trans. Electron. Comput**, [S.l.], v. 26, p. 297-303, 1977.

[GAV99] GAVRILA, D.M. ; PHILOMIN, V. Real-Time Object Detection for "Smart" Vehicules. In: INTERNATIONAL CONFERENCE ON COMPUTER VISION, ICCV, 1999, Corfu, Greece. **Proceedings...** [S.l.: s.n.], 1999. v.1.

[GON92] GONZALEZ, Rafael; WOODS, Richard E. **Digital Image Processing.** New York : Addison-Wesley Publishing Company, 1992.

[GSC94] GSCHWIND, M.; VALENTINA, S.; MAISCHBERGER, O. Space Efficient Neural Net Implementation. In: INTERNATIONAL ACM/SIGDA WORKSHOP ON FIELD PROGRAMMABLE GATE ARRAYS, California, USA. **Proceedings...** [S.l.: s.n.], 1994.

[HAN99] HAN, Gunhee; SÁNCHEZ-SINENCIO, Edgar. A Flexible and Expendable Neuroimage Processor Architecture. **IEEE Transactions on Circuits and Systems - I**: Fundamental Theory and Applications, New York, v.46, n.9, p.1055-1063, 1999.

[HEA98] HEATH, M. et al. Comparison of Edge Detectors. **Computer Vision and Image Understanding**, San Diego, v.69, n.1, p.38-54, 1998.

[HOE97] HOEPPNER, F. Fuzzy Shell Clustring Algorithms in Image Processing: Fuzzy C-Rectangular and 2-Rectangular Shells. **IEEE Transactions on Fuzzy Systems**, New York, v.5, n.4, p.599- 613, 1997.

[HU62] HU, M.K. Visual pattern recognition by moment invariants. **IRE Transactions on Information Theory**, [S.l.], v.8, p.28-32, 1962.

[JOR96] JORGENSEN, T.M.; CHRISTENSEN, S.S.; ANDERSEN, A.W. Detecting danger labels with RAM-based neural networks. **Pattern Recognition Letters**, Amsterdam, v.17, p.399-412, 1996.

[KRA97] KRALJIC, I.C.; QUENOT, G.M.; ZAVIDOVIQUE, B. Investigating real-time validation of Real-Time Image Processing ASICs. In: COMPUTER ARCHITECTURES FOR MACHINE PERCEPTION, 1997, Como, It. **Proceedings...** [S.l.: s.n.], 1997.

[KUL94] KULKARNI, Arun D. **Artificial Neural Networks for Image Understanding**. USA: Van Nostrand Reinhold, 1994.

[LIB97] LIBERMANN, F. **Classificação de Imagens Digitais por Textura usando Redes Neurais**. Porto Alegre: CPGCC da UFRGS, 1997. 86p. Dissertação de Mestrado.

[LOW99] LOWE, David G. Object Recognition from Local Scale-Invariant Features. In: INTERNATIONAL CONFERENCE ON COMPUTER VISION, 1999, Corfu, Greece. **Proceedings...** [S.l.: s.n.], 1999.

[MAH97] MAHMOUD, M.; NAKANISHI, M.; OGURA, T. Hough transform implementation on a reconfigurable highly parallel architecture. In: COMPUTER ARCHITECTURES FOR MACHINE PERCEPTION 1997, Como, It. **Proceedings...** [S.l.: s.n.], 1997.

[MAR82] MARR, D. **Vision:** a Computational Investigation into the Human representation and Processing of Visual Information. New York: Freeman, 1982.

[MIN98] MINGO, Ferran L. **Configurable Computing for Real-Time Vision**. Bellaterra: Universitat Autônoma de Barcelona – UAB, 1998. 180p. Tese de Doutorado.

[MOK98] MOKHTARIAN, F.; SUOMELA, R. Robust Image Corner Detection Through Curvature Scale Space. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, New York, v.20, n.12, p.1376-1381, 1998.

[MOL99] MOLZ, R. F.; ENGEL, P.M. ; MORAES, F.G. Uso de um ambiente codesign para a implementação de redes neurais. In: CONGRESSO BRASILEIRO DE REDES NEURAIS, CBRN, 4., 1999, São José dos Campos, Brasil. **Anais...** [S.l.: s.n.], 1999. p.13-18.

[MOL2000] MOLZ, Rolf F. et al. Codesign of Fully Parallel Neural Network for a Classification Problem. In: WORLD MULTICONFERENCE ON SYSTEMICS, CYBERNETICS AND INFORMATICS, SCI, 4., Florida, USA. **Proceedings...** Orlando:IIIS,2000. p.601-605.

[MOL2000a] MOLZ, Rolf Fredi. **Hardware/Software implementation of a Neural Network Model for Localization and Classification Systems**.

Montpellier (França): Université de Montpellier II (LIRMM), 2000. 42p. (Rapport de Recherche n.00136).

[MOR99] MORENO, E.G. et al. Pipeline Architectures for Computing 2-D Image Moments. In: DESIGN OF CIRCUITS AND INTEGRATED SYSTEMS CONFERENCE, DCIS, 14., Palma de Mallorca, Spain. **Proceedings...** [S.l.: s.n.], 1999. p.169-174.

[NAY99] NAYAK, S.S.; MECHER, P.K. High Throughput VLSI Implementation of Discrete Orthogonal Transforms Using Bit-Level Vector-Matrix Multiplier. **IEEE Trans. on Circuits and Systems – II**: Analog and Digital Signal Processing, New York, v.46, n.5, p.655-658, 1999.

[OLD95] OLDFIELD, John V; DORF, Richard C. **Field Programmable Gate Arrays**. New York : John Wiley & Sons, 1995.

[PAR97] PARKER, J.R. **Algorithms for Image Processing and Computer Vision**. New York : John Wiley & Sons, 1997.

[ROB94] ROBERT, M. et al. Architectures for a Real Time Classification Processor. In: CONFERENCE ON CUSTOM INTEGRATED CIRCUITS, 1994. **Proceedings...** [S.l.: s.n.], 1994.

[ROS98] ROSADO, Alfredo et al. A new hardware architecture for a general-purpose neuron based on distributed arithmetic and implemented on FPGA devices, In: INTERNATIONAL SYMPOSIUM ON ENGINEERING OF INTELLIGENT SYSTEMS, 1998. **Proceedings...** [S.l.: s.n.], 1998. p.321-326.

[RUC97] RUCKLIDGE, William. Efficiently Locating Objects using the Hausdorff Distance. **International Journal of Computer Vision,** [S.l.], v.24, n.3, p.251-270, 1997.

[SAN99] SANSOÉ, C.; GREGORETTI, F.; REYNERI, L.M. A Neuro-Fuzzy Real-Time Image Processing System. In: EUROMICRO CONFERENCE, 25., 1999, Milan, It. **Proceedings...** [S.l.: s.n.], 1999.

[SIM98] SIM, H.C.; NG. G.S. Recognition of Partially Ocluded Objects with Back-Propagation Neural Network. **Intern. Journal of Pattern Recognition and Artificial Intelligence**, [S.l.], v.12, n.5, p.645-660, 1998.

[SUD98] SUDHA, N.; NANDI, S. A Parallel Skeletonization Algorithm and its VLSI Architecture. In: INTERNATINAL CONFERENCE ON HIGH PERFORMANCE COMPUTING, 5., Madras, India. **Proceedings...** [S.l.: s.n.], 1998.

[TOR97] TORRES, Lionel; PILLEMENT, S.; ROBERT, Michel. LIRMM: Prototyping Plataform for Hardware/Software Codesign. **Journal of**

Current Issue in Electronic Modelling, [S.l.], v.9, p.49-59, 1997.

[TOR98] TORRES, L. et al. A Recursive Digital Filter Implementation for Noisy and Blurred Images. **Real Time Imaging Journal**, [S.l.], v.4, n.3, p.181-191, 1998.

[TRI89] TRIVEDI, M. M.; ROSENFELD, A. On Making Computers “see”. **IEEE Trans. Systems Man. Cyber.**, New York, v.19, n.6, p.1333-1335, 1989.

[TRI96] TRIER, D.; JAIN, A.K.; TAXT, T. Feature Extraction Methods for Character Recognition – A Survey. **Pattern Recognition**, [S.l.], v.29, n. 4, p.641-662, 1996.

[TSA99] TSANG, K.M.; TO, F.W. Recognition of Partially Occluded Objects using an Orthogonal Complex AR Model Approach. **Intern. Journal of Pattern Recognition and Artificial Intelligence**, [S.l.], v.13, n.1, p.85-107, 1999.

[VIL97] VILLASENOR, John; MANGIONE-SMITH, W.H. Configurable Computing. **Scientific American**, [S.l.], p.55-59, Sept. 1997.

[WIA2000] WIATR, Kazimierz; JAMRO, Ernest. Implementation Image Data Convolutions Operations in FPGA Reconfigurable Structures for Real-Time Vision Systems. In: INTERNATIONAL CONFERENCE ON INFORMATION TECHNOLOGY: CODING AND COMPUTING, ITCC, 2000. **Proceedings...** [S.l.:s.n.], 2000.

[YAN98] YANG, Fan. **Traitement automatique d'images de visages algorithmes et architecture**. France : U.F.R. Sciences et Techniques - Université de Bourgogne, 1998. 181p. Tese de Doutorado.