SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 24.09.2001

Assinatura:

Cláudio Luiz Luri

**Projeto de uma rede neural usada no
reconhecimento de gestos por robôs móveis
utilizando-se computação reconfigurável**

Denis Fernando Wolf

Orientador: *Prof. Dr. Eduardo Marques*

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP – São Carlos
Setembro/2001

A Comissão Julgadora:

Prof. Dr. Eduardo Marques

Handwritten signature of Eduardo Marques in black ink, written over a horizontal line.

Profa. Dra. Roseli Aparecida Francelin Romero

Handwritten signature of Roseli Aparecida Francelin Romero in black ink, written over a horizontal line.

Profa. Dra. Anna Helena Reali Costa

Handwritten signature of Anna Helena Reali Costa in black ink, written over a horizontal line.

*"Vá tão longe quanto possa ver. Quando chegar lá,
você poderá ver ainda mais longe."*

Thomas Carlyle

*Este trabalho é dedicado aos meus pais,
Eloi José Wolf e Maria Rosa Wolf.*

Agradecimentos

Aos meus pais Eloi e Roseli, e ao meu irmão William, pela dedicação, carinho e incentivo em todos esses anos.

Ao meu orientador e grande amigo Prof. Dr. Eduardo Marques, por toda dedicação, atenção e orientação, não só acadêmica mas também pessoal e profissional.

A Prof. Dra. Roseli Romero, pela ajuda e atenção durante todo o desenvolvimento desse trabalho.

Ao Gedson do LABIC, pela ajuda com os testes e comparações de desempenho.

A todos os companheiros do LCR, pelos momentos de estudo e diversão compartilhados.

Aos funcionários do ICMR sempre prestativos e atenciosos.

Ao CNPq pelo auxílio financeiro.

Resumo

A Robótica é uma área de pesquisa que tem evoluído muito nas ultimas décadas, e tem recebido grande atenção pela comunidade científica. As principais limitações dos sistemas robóticos atuais estão no controle dessas máquinas complexas. É importante que sejam desenvolvidos algoritmos inteligentes o suficiente para controlar os robôs de maneira apropriada, e que esses algoritmos sejam executados de maneira eficiente. A utilização de hardware para a execução desses algoritmos é uma alternativa interessante para se conseguir um bom desempenho. Essa dissertação apresenta a implementação de uma rede neural artificial, utilizada no reconhecimento de gestos por robôs móveis. Para tanto, é utilizada a tecnologia FPGA, que permite o desenvolvimento de hardware com rapidez, eficiência e baixo custo. Após o desenvolvimento, são apresentadas comparações de desempenho e conclusões sobre as implementações.

Abstract

Robotics is a research area that has been growing in the last decades, and it has been receiving much attention by the scientific academy. The main limitations of the current robotics systems remain in the control of those complex machines. It is important to develop algorithms intelligent enough to handle the robots in an appropriate way, and these algorithms must be executed in an efficient way too. The utilization of hardware implementation is an interesting manner to acquire good performance in the algorithms execution. This work presents a hardware model of an artificial neural network used in the gesture recognizing by mobile robots. Thus, FPGA technology has been used. It allows quickness, efficiency, and low prices. After the hardware development, some performance comparisons and conclusions about the implementation are shown.

Sumário

Capítulo 1 Introdução.....	1
Capítulo 2 A Tecnologia FPGA.....	4
2.1 O Projeto de Circuitos Digitais	4
2.2 A Evolução dos PLDs	6
2.2.1 Os FPGAs Altera.....	8
Capítulo 3 A Computação Reconfigurável.....	12
3.1 Introdução.....	12
3.2 Sistemas Computacionais.....	12
3.2.1 Sistemas de Computação Reconfigurável	15
3.3 O Kit de Desenvolvimento Excalibur	17
3.3.1 O Processador Nios	18
3.3.2 Compilador GNUPro	20
3.3.3 Quartus II – Ferramenta de Desenvolvimento	20
3.3.4 Placa de Desenvolvimento	20
3.4 O Paradigma SOC – <i>System On a Chip</i>	21
Capítulo 4 Redes Neurais Artificiais.....	24
4.1 Introdução.....	24
4.2 O Neurônio Biológico	25
4.3 O Modelo de Neurônio de McCulloch e Pitts	26
4.4 Aprendizado	28
4.4.1 Aprendizado Supervisionado	29
4.4.2 Aprendizado por Reforço	29

4.4.3 Aprendizado Não-Supervisionado	30
4.4.4 Aprendizado por Competição	30
4.5 Os Modelos Perceptron e Adaline.....	31
4.6 As Redes MLP (Multi Layer Perceptron)	32
4.7 Redes Neurais Aplicadas a Robótica	33
4.7.1 O Projeto ALVINN.....	34
4.7.2 Planejamento de Percurso	35
4.7.3 Mapeamento de Ambiente e Posicionamento	36
Capítulo 5 Robôs Móveis	38
5.1 Introdução.....	38
5.2 Algoritmos de Robótica	40
5.3 Desafios da Robótica.....	41
5.3.1 Navegação	41
5.3.2 Reconhecimento	41
5.3.3 Aprendizado	43
5.3.4 Cooperação.....	43
5.4 O Projeto ARMOSH	44
5.4.1 Objetivos do Projeto ARMOSH.....	44
5.4.2 Interfaces Homem/Robô	45
5.4.3 Navegação	46
5.4.4 Relevância do Projeto ARMOSH	47
Capítulo 6 Desenvolvimento do Hardware.....	49
6.1 Objetivos	49
6.1.1 O Reconhecimento de Gestos	49
6.2 O Desenvolvimento de Redes Neurais em Hardware	51
6.2.1 Dificuldades na Implementação	53
6.2.2 Representação Numérica de Ponto Flutuante.....	54
6.2.3 Funções não lineares	54
6.2.4 Capacidade dos Dispositivos de Hardware	58
6.2.5 Integrando o processador Nios com o Hardware Dedicado	59

6.3 Implementação de uma Etapa de Desenvolvimento Utilizando-se o Conjunto de Dados Iris	60
6.3.1 Otimização de Desempenho.....	64
6.3.2 Utilização da Técnica de Pipelines	64
6.3.3 A Integração com o Processador Nios	69
6.4 Implementação da Rede Neural Utilizada no Reconhecimento de Gestos por Robôs Móveis.....	71
6.4.1 O modelo Sequencial do Neurônio na RNG	73
6.4.2 O Modelo Completo da RNG	77
6.4.3 Reutilização do Hardware Implementado	78
6.4.4 A RNG com o Processador Nios.....	78
6.5 Discussão dos Resultados Obtidos.....	82
6.5.1 Análise de Desempenho	82
6.5.1 Análise de Erro.....	85
Capítulo 7 Conclusões e Trabalhos Futuros.....	88
7.1 Conclusões	88
7.2 Trabalhos Futuros.....	89
Referências Bibliográficas	91

Capítulo 1

Introdução

Um robô inteligente é uma máquina capaz de extrair informações do ambiente e usar seu conhecimento sobre o mundo para se mover de maneira segura com propósito definido [Arkin, 1998].

Segundo a literatura, o primeiro robô móvel construído é o *Shakey*, desenvolvido em 1968 na Universidade de Stanford. Ele tinha uma variedade enorme de sensores, incluindo uma câmera de vídeo e sensores de toque. O *Shakey* realizava tarefas de navegação entre as salas do laboratório além de empurrar caixas e evitar obstáculos. Todo o processamento era realizado por um computador DEC PDP-10 externo, o qual recebia e enviava mensagens para o robô através de sinais de rádio [Vieira, 1999].

Depois de algumas décadas de pesquisa e desenvolvimento tecnológico, a área de robótica apresenta uma considerável evolução. Atualmente, os robôs móveis são capazes de realizar tarefas que a algum tempo atrás seriam consideradas pura ficção científica. Em 1996 o robô *Sojourner* [Nasa, 2001] foi enviado a Marte para obter e enviar informações sobre o planeta vermelho para os cientistas da Terra. Competições entre robôs que jogam futebol são comuns, e contam com a participação de universidades de todo o mundo. No futuro, pretende-se ter equipes de robôs humanóides jogando futebol contra humanos [Robocup, 2001].

Mesmo com toda a evolução dos componentes eletrônicos e dos processos de fabricação mecânica, a área de robótica ainda apresenta muitos desafios e uma infinidade de questões em aberto.

Atualmente, a maior dificuldade na obtenção de robôs móveis eficientes consiste na parte inteligente dos mesmos, ou seja, os algoritmos de controle. Esses algoritmos devem ser capazes de lidar com situações altamente complexas e dinâmicas [Arkin, 1998].

Tão importante quanto o desenvolvimento desses algoritmos inteligentes é a execução eficiente dos mesmos. Diante do grande e cada vez mais complexo volume de dados recebidos pelos sensores, é de suma importância que se tenha um sistema de processamento capaz de manipular e interpretar todos esses dados com o desempenho necessário. Desempenho esse que muitas vezes não pode ser alcançado através da execução dos algoritmos em processadores convencionais. Dentro desse contexto, implementações diretamente em hardware se fazem bastante interessantes, pois apresentam um desempenho muito superior quando comparados com softwares executados em processadores de propósito geral.

A tecnologia dos dispositivos FPGA (*Field Programmable Gate Arrays*) permite que sistemas computacionais sejam desenvolvidos diretamente em hardware a um custo muito baixo e com excelente desempenho.

O objetivo desse trabalho consiste no desenvolvimento de um hardware dedicado de uma rede neural do tipo MLP (*Multi Layer Perceptron*) utilizada no reconhecimento de gestos por robôs móveis.

A pesquisa original, na qual foi baseado esse trabalho, foi desenvolvida por Waldherr [Waldherr,1999] na Carnegie Mellon University - EUA. Nessa pesquisa, foi desenvolvida uma interface homem/robô através de gestos dinâmicos. As imagens captadas pelo robô eram processadas, as informações sobre os gestos eram extraídas e interpretadas por uma rede neural. O processamento de imagens é uma tarefa que requer um grande esforço computacional, e nem sempre os sistemas computacionais convencionais são capazes de processar esse grande volume de informações com o desempenho desejado. Esse trabalho visa o desenvolvimento de um circuito semelhante a rede neural utilizada na pesquisa de Waldherr, uma vez que a rede neural representava o trecho do sistema que demandava o maior esforço computacional. Após implementado o modelo de hardware, são realizadas comparações de desempenho entre as diferentes implementações (software e hardware).

A dissertação de mestrado aqui apresentada está organizada da seguinte forma:

- **Capítulo 2:** É apresentada a tecnologia FPGA, com um breve histórico e uma análise dos dispositivos disponíveis pelos principais fabricantes.
- **Capítulo 3:** São Discutidos os sistemas de Computação Reconfiguráveis, as características principais dessa forma de computação e quais as perspectivas da área.
- **Capítulo 4:** São apresentadas as Redes Neurais Artificiais, um breve histórico de seu desenvolvimento e aplicações.
- **Capítulo 5:** São apresentados Robôs Móveis, com um breve histórico de sua evolução e uma discussão sobre os atuais desafios da robótica.
- **Capítulo 6:** É apresentado o desenvolvimento desse trabalho de mestrado com os resultados obtidos.
- **Capítulo 7:** São apresentadas as conclusões e trabalhos futuros.

Capítulo 2

A Tecnologia FPGA

2.1 O Projeto de Circuitos Digitais

A grande revolução tecnológica das últimas décadas tem resultado em grandes mudanças em praticamente todas as áreas da computação. Áreas como a transmissão e o armazenamento de dados têm evoluído de maneira considerável nos últimos tempos. Hoje em dia é possível transmitir imagens e sons, de qualidade bastante razoável, para qualquer lugar do mundo através da internet. Há poucas décadas atrás, esse tipo de operação fazia parte apenas de filmes de ficção científica e atualmente faz parte do cotidiano de grande parte das pessoas.

A área de sistemas digitais vem acompanhando a evolução tecnológica e, conseqüentemente, tem apresentado mudanças bastante significativas. Essas mudanças têm possibilitado a construção de circuitos digitais cada vez mais compactos e eficientes. Os antigos transistores definitivamente deram lugar à tecnologia VLSI (*Very Large Scale Integration*). A maneira pela qual esses circuitos são projetados também vem passando por grandes mudanças. As tabelas e folhas de projeto têm sido substituídas por ferramentas EDA (*Electronic Design Automation*), que permitem que projetos digitais sejam desenvolvidos de forma mais simples e rápida. As linguagens de descrição de hardware (HDLs) permitem que projetos complexos sejam desenvolvidos de maneira organizada e eficiente. Linguagens como o VHDL (*VHSIC Hardware Description Language*, onde VHSIC significa *Very High Speed Integrated Circuit*) [Yalamanchili, 2001] e o Verilog [Coffman, 2000] têm se consolidado no meio acadêmico e industrial. Essas linguagens se

assemelham muito com conhecidas linguagens de programação como C e Pascal, possibilitando bastante flexibilidade e organização no desenvolvimento de hardware. Existem também ferramentas específicas para síntese lógica, as quais utilizam técnicas de inteligência artificial para realizar a síntese e o roteamento do hardware projetado da maneira mais eficiente possível [Aragão, 1997].

Com a evolução, os produtos de natureza tecnológicos têm apresentado um ciclo de vida cada vez menor. Dessa forma, é de suma importância que os produtos sejam desenvolvidos no tempo mais curto possível. Todas essas modernas metodologias para o projeto de hardware se fazem extremamente necessárias, contribuindo para que esses objetivos sejam atingidos.

Basicamente, o projeto de sistemas digitais pode ser realizado utilizando-se circuitos totalmente customizados, como os ASICs (*Application Specific Integrated Circuits*), ou semicustomizados: PLDs (*Programmable Logic Devices*) e MPGAs (*Mask-Programmable Gate Arrays*), como pode ser visto no diagrama da Figura 2.1.

Figura 2.1: Tecnologias para o projeto de sistemas digitais

A tecnologia ASIC requer máscaras específicas para cada projeto, resultando em um tempo de desenvolvimento longo e custos extremamente altos. Quando o volume de produção é alto, o custo é amortizado. Após o desenvolvimento da máscara, o projeto não pode ser mais modificado. Da mesma maneira, depois de construído o chip, esse não pode sofrer modificações.

Entre as tecnologias de dispositivos semicustomizados, se destacam os MPGAs e os PLDs.

Os MPGAs apresentam um processo de fabricação agilizado pelo uso de máscaras genéricas de módulos pré-projetados, mas ainda necessitam máscaras específicas para a interconexão dos módulos. O projeto é usualmente facilitado por uma biblioteca de células, oferecendo um tempo de desenvolvimento mais curto e custos mais baixos em relação aos ASICs.

Os PLDs apresentam como principal característica a capacidade de programação (configuração) por parte do próprio usuário, facilitando bastante a mudança de projetos. Esse tipo de dispositivo apresenta um ciclo de projeto curto associado a um baixo custo. Os PLDs podem ser classificados como SPLDs (*Simple PLDs*), CPLDs (*Complex PLDs*) e FPGAs, como será mostrado a seguir [Aragão, 1997].

2.2 A Evolução dos PLDs

A memória PROM (*Programmable Read-Only Memory*), foi o primeiro tipo de circuito programável pelo usuário, o qual podia implementar circuitos lógicos. As linhas de endereço eram utilizadas como entradas do circuito lógico e as linhas de dados como saídas desses circuitos [Brown, 1996].

Os PLAs (*Programmable Logic Arrays*), introduzidos pela Philips no início dos anos 70, foram os primeiros dispositivos criados especificamente para a implementação de circuitos lógicos. Esse tipo de dispositivo apresentava alto custo de fabricação e baixo desempenho. Ambas as desvantagens eram resultado de dificuldades na fabricação dos

planos lógicos, o que introduzia significantes atrasos na propagação dos sinais elétricos. A tecnologia PAL (*Programmable Array Logic*) foi então desenvolvida para superar algumas deficiências dos seus antecessores PLAs, apresentando um custo mais baixo e desempenho mais alto em relação aos PLAs.

Todos os pequenos PLDs, tais como: PLAs, PALs e similares, são agrupados em uma única categoria de dispositivos chamada: SPLDs (*Single PLDs*). A dificuldade em aumentar a capacidade da arquitetura PLD é causada devido ao fato que a estrutura de planos lógicos programáveis se torna altamente complexa com o aumento do número de entradas. A única maneira viável de produzir dispositivos com maior capacidade seria então integrar múltiplos SPLDs em um único chip, e prover interconexões programáveis para conectar os diversos PLDs. Atualmente existem muitos produtos comerciais utilizando essa estrutura básica, os quais são chamados CPLDs (*Complex PLDs*) [den Bout, 1999]. Os CPLDs provêem capacidade lógica de centenas de dispositivos SPLDs típicos (Figura 2.2).

Figura 2.2: Estrutura de um CPLD.

A necessidade de uma tecnologia que combinasse versatilidade, desempenho e alta capacidade motivou então o projeto de dispositivos programáveis conhecidos como FPGAs (*Field-Programmable Gate Arrays*). A principal diferença dessa tecnologia em relação a

dos MPGAs consiste no fato de que as interconexões são feitas via comutadores programáveis eletricamente, assim como nos PLDs. A programação de um FPGA é realizada pelo próprio usuário final [Aragão, 1997].

A arquitetura básica de um FPGA consiste de um arranjo 2-D de blocos lógicos. A comunicação entre os blocos é realizada através de recursos de interconexão. A borda externa desse arranjo consiste em blocos especiais capazes de realizar operações de entrada e saída (I/O). Uma arquitetura típica de um FPGA pode ser visualizada na Figura 2.3.

Figura 2.3: Estrutura básica de um FPGA.

Os FPGAs combinam o desempenho dos *gate arrays* com a versatilidade dos PLDs, e têm sido responsáveis por grandes mudanças no modo em que os circuitos digitais são projetados [Aragão, 1997].

2.2.1 Os FPGAs Altera

Existem diversos fabricantes de FPGAs, tais como: Altera, Xilinx, Actel, Atmel, AMD, etc. Neste trabalho serão apresentadas apenas as FPGAs da Altera, as quais foram utilizadas no desenvolvimento das implementações.

As mais modernas famílias de FPGAs desenvolvida pela Altera são: as APEX e APEX II. Os chips APEX e APEX II apresentam um altíssimo nível de integração, o que permite que sistemas completos sejam implementados em um único dispositivo. Através da arquitetura MultiCore, as famílias APEX acentuam o desempenho dos PLDs que a antecederam, possibilitando flexibilidade de projeto e eficiência para aplicações SOC (*System on a Chip*) de alto desempenho.

Os dispositivos da família APEX apresentam densidade de até 2.400.000 portas lógicas e até 442 kbits de memória *on-chip* (Tabela 2.1), enquanto os APEX II apresentam até 7 milhões de portas lógicas e até 1524 kbits de memória (Tabela 2.2). Os dispositivos APEX são compatíveis com arquiteturas PCI de 64bits e 66MHz enquanto os APEX II suportam o padrão PCI-X de 133MHz. O nível de integração dos chips da família APEX varia de 0.22 μ m a 0.15 μ m (a família APEX II apresenta dispositivos com nível de integração 0.15 μ m) [Altera, 2001].

Feature	EP20K200E	EP20K400	EP20K400E	EP20K600E	EP20K1000E	EP20K1500E
Maximum system gates	526,000	1,052,000	1,052,000	1,537,000	1,772,000	2,392,000
Typical gates	200,000	400,000	400,000	600,000	1,000,000	1,500,000
LEs	8,320	16,640	16,640	24,320	38,400	51,840
ESBs	52	104	104	152	160	216
Maximum RAM bits	106,496	212,992	212,992	311,296	327,680	442,368
Maximum macrocells	832	1,664	1,664	2,432	2,560	3,456
Maximum user I/O pins	376	502	488	588	708	808

Tabela 2.1: Características dos FPGAs APEX da Altera

A chamada arquitetura MultiCore dos chips APEX combina: LUTs (*Look Up Tables*), termos produto e blocos de memória. As LUTs são utilizadas para implementar funções que necessitam de registradores. Os blocos de memória são utilizados na implementação de RAMs e ROMs. Os termos produto são utilizados na implementação de lógica computacional de alto desempenho como máquinas de estado. Esses dois diferentes tipos de estrutura são combinados em uma arquitetura integrada, eliminando a necessidade de diversos chips e simplificando a implementação de projetos complexos (Figura 2.3).

Feature	EP2A15	EP2A25	EP2A40	EP2A70	EP2A90
Maximum gates	1,900,000	2,750,000	3,000,000	5,250,000	7,000,000
Typical gates	600,000	900,000	1,500,000	3,000,000	4,000,000
LEs	16,640	24,320	38,400	67,200	89,280
RAM ESBs	104	152	160	280	372
Maximum RAM bits	425,984	622,592	655,360	1,146,880	1,523,712
True-LVDS channels	36 (1)	36 (1)	36 (1)	36 (1)	36 (1)
Flexible-LVDS™ channels (2)	56	56	88	88	88
True-LVDS PLLs (3)	4	4	4	4	4
General-purpose PLL outputs (4)	8	8	6	8	8
Maximum user I/O pins	492	607	735	1,080	1,140

Tabela 2.2: Características dos FPGAs APEX II da Altera

Os ESBs (*Embedded System Blocks*) são elementos de grande importância na arquitetura MultiCore. Os 2048 bits programáveis de cada ESB podem ser configurados como dual-port RAM, ROM ou CAM (memórias endereçadas pelo conteúdo). A memória RAM implementada nos chips APEX suporta operações em até 255MHz (as memórias ROM suportam operações até 230MHz) enquanto nos dispositivos APEX II, esses valores podem chegar a até 1GHz [Altera, 2001].

Figura 2.3: Estrutura interna das famílias APEX e APEX II.

A Arquitetura MultiCore é organizada em diversas estruturas chamadas MegaLab. Cada MegaLab é composto por um conjunto de LABs (*Logic Array Blocks*), um ESB e as interconexões necessárias (Figura 2.4). Cada LAB é composto por 10 elementos lógicos – LEs (Figura 2.5), os quais consistem nas unidades mais básicas presentes na estrutura MultiCore.

Figura X: Estrutura do MegaLab

Figura X: Elemento Lógico da Arquitetura MultiCore

Capítulo 3

A Computação Reconfigurável

3.1 Introdução

Os sistemas computacionais podem ser implementados, basicamente, de duas maneiras: utilizando-se hardware ou software, os quais são executados em processadores de propósito geral. Mais recentemente, no entanto, os FPGAs têm se apresentado como uma alternativa bastante interessante, possibilitando que sistemas altamente versáteis e eficientes sejam desenvolvidos.

A utilização de FPGAs nos sistemas computacionais gera uma classe de arquiteturas chamada: Computação Reconfigurável. Segundo [Dehon, 1999], uma das principais características que distinguem essa classe de arquiteturas é a possibilidade de se customizar os dispositivos para a resolução de problemas específicos, mesmo depois da fabricação dos mesmos.

3.2 Sistemas Computacionais

Os *chips* que executam grande quantidade de funções diferentes, resolvendo uma série de diferentes problemas, normalmente, são referenciados como de uso geral (ou de propósito geral); entretanto, a velocidade desses dispositivos pode ser lenta em relação aos dispositivos com aplicação específica, que executam apenas um conjunto limitado de instruções.

Os famosos microprocessadores comerciais tais como: INTEL Pentium ou Motorola PowerPC, frequentemente encontrado nos computadores pessoais, são exemplos de sistemas com características de uso geral. Eles são capazes de executar um grande número de funções, atendendo assim um grande número de usuários que fazem uso de computadores pessoais para suas atividades. A utilização desse tipo de sistema é realizada através de softwares (programas de computador). O programador organiza as instruções que serão executadas pelo processador através de linguagens de programação.

São características dos sistemas desenvolvidos em software:

- Flexibilidade: Uma tarefa pode ser alterada, simplesmente, trocando-se o fluxo de instruções numa memória regravável.
- Baixa eficiência: A alta generalidade dos sistemas implica em execuções lentas causadas pelo overhead de instruções.
- Preços Baixos: Como diversas aplicações diferentes utilizam os mesmos dispositivos, esses podem ser fabricados em altíssimas escalas de produção.

Cada nova geração dos microprocessadores possibilita um aumento de desempenho na execução das aplicações. Entretanto, na maioria dos casos, a aplicação precisa ser recompilada para melhor usufruir das novas características desses processadores mais modernos [Gonçalves, 2000].

De uma forma oposta a esses sistemas, pode-se destacar os circuitos de hardware dedicado, conhecidos como ASICs (*Applications-Specific Integrated Circuits*), que em função de sua especificidade, os projetistas podem desenvolver *chips* menores, mais velozes, consumindo menos energia do que um processador de uso geral.

Durante o desenvolvimento de projetos de hardware, além da versatilidade e velocidade, os projetistas de hardware devem também levar em consideração as questões de custos dos sistemas. Se for considerado, por exemplo, um sistema implementado utilizando-se a tecnologia ASIC, que resolverá um problema específico, nada poderá ser modificado após o projeto ter sido concluído. Caso existam modificações a serem feitas no

projeto, todos os *chips* fabricados são descartados e novas unidades devem ser produzidas, implicando em custos extras.

As características básicas dos circuitos dedicados são:

- Falta de flexibilidade: Os sistemas são construídos com propósitos definidos, que não podem ser alterados.
- Eficiência: Os sistemas são fabricados para aplicações específicas, apresentando um excelente desempenho
- Preços altos: O custo de produção é altíssimo, principalmente quando são fabricados em pouca quantidade.

Uma nova opção em circuitos integrados, que oferece grande capacidade e desempenho, é a tecnologia FPGA. Os dispositivos FPGAs são circuitos de hardware que podem ser modificados (reconfigurados). Praticamente, em qualquer momento durante o uso, eles podem ser alterados, nas funções lógicas, nos blocos lógicos e nas conexões entre esses blocos, simplesmente enviando-se sinais de reconfiguração para o *chip*. Os blocos lógicos em um FPGA podem ser re-escritos e re-programados repetidamente, depois do *chip* ter sido produzido em uma fábrica [Dehon, 1999].

Os primeiros FPGAs necessitavam de vários segundos para alterar seus conteúdos, o que é razoável para verificar novos protótipos em um ciclo de desenvolvimento. Nos dias de hoje, já é possível que o hardware se adapte quase que continuamente em resposta a mudanças de sinais de entrada ou em respostas a sinais do meio externo.

Sistemas computacionais podem fazer uso dos elementos reconfiguráveis de diferentes maneiras. A maneira mais comum de utilização desses dispositivos consiste em comutar o hardware para diferentes aplicações, de forma semelhante a encerrar um programa de software e iniciar outro.

São características dos sistemas reconfiguráveis:

- Flexibilidade: Os dispositivos podem ser configurados para realizar qualquer tarefa computacional.

- **Eficiência:** Por resolverem problemas altamente específicos, apresentam desempenho bastante satisfatório.
- **Preços baixos:** Os dispositivos são fabricados em escalas muito altas pois, serão reconfigurados pelo usuário para resolverem problemas específicos.

A Computação Reconfigurável ainda é uma área em desenvolvimento, diferente de arquiteturas de computadores tradicionais, aonde os conceitos vêm sendo testados e comprovados a muito tempo. Portanto, definir e validar todas essas definições e conceitos a respeito da computação reconfigurável se torna, atualmente, um desafio para os pesquisadores [Dehon, 1999].

Berry Shackleford [Beckett, 2000], projetista dos computadores HP para 2010, acredita que o avanço tecnológico permitirá que em 2010 haja bilhões de transistores em um único *chip*. O desafio dos projetistas de hardware será encontrar a arquitetura computacional que consiga extrair o maior desempenho possível, proporcionado por um nível de integração tão grande. Segundo Shackleford, a tecnologia FPGA certamente será um componente fundamental para que todo o poder computacional disponibilizado pelo avanço tecnológico possa ser efetivamente utilizado.

3.2.1 Sistemas de Computação Reconfigurável

Os sistemas de computação reconfigurável são plataformas onde a arquitetura pode ser modificada em tempo real para melhor se adequar à aplicação que será executada. Deste modo, o processador reconfigurável passa a trabalhar com uma arquitetura específica, desenvolvida exclusivamente para aquele determinado tipo de aplicação. Isso possibilita uma eficiência muito maior do que a normalmente encontrada em processadores de uso geral, já que o hardware é otimizado para executar os algoritmos necessários para aquela aplicação específica.

O hardware reconfigurável pode ser classificado em três categorias [Hauck, 1998], como mostra a Figura 3.1:

- Hardware Puro (hardware para aplicações específicas)
- Coprocessadores (*chips* específicos para determinadas tarefas)
- Computadores (plataformas de computação completamente reconfiguráveis)

Figura 3.1: Categorias de hardware reconfigurável

Atualmente, a forma de utilização mais difundida dos *chips* reconfiguráveis é como parte de sistemas que utilizam processadores tradicionais. Os *chips* reconfiguráveis atuam como coprocessadores, normalmente trabalhando com algoritmos de alto custo computacional. Dessa forma, eles aliviam a carga do processador principal, permitindo que este possa executar outras tarefas mais genéricas. A área de computação reconfigurável apresenta-se como uma tecnologia extremamente interessante tanto no campo industrial, como no campo acadêmico. São inúmeras as atuais aplicações de computação reconfigurável como, por exemplo: telefones celulares, controladores de dispositivos presentes em carros e aviões, robótica e hardwares específicos de alto desempenho para as mais diversas aplicações.

Outra área bastante interessante para o uso de dispositivos reconfiguráveis é a prototipação. Semelhante ao software, o desenvolvimento de hardware é realizado em etapas, as quais devem ser testadas e validadas para que prossiga o desenvolvimento do

projeto. Quando se tratam de sistemas de software, esses testes são realizados de maneira muito simples e eficiente utilizando-se ferramentas para depuração de código. Em projetos de hardware dedicado (ASICs) é inviável que se desenvolva protótipos com funcionalidade parcial para que o sistema seja validado em etapas, pois esse processo seria inadequadamente caro e demorado. A computação reconfigurável é uma alternativa bastante atraente para esse tipo de tarefa. Como os dispositivos FPGA podem ser configurados freqüentemente e muito rapidamente, os protótipos de hardware (partes dos projetos) podem ser implementados utilizando-se FPGA a um custo e tempo muito baixos.

3.3 O Kit de Desenvolvimento Excalibur

O kit de desenvolvimento Altera Excalibur é um conjunto de hardware (placa de desenvolvimento, dispositivo FPGA, periféricos e dispositivos de conexão) e software (ferramentas para desenvolvimento e testes) utilizados para o projeto de hardware [Altera, 2000].

O Excalibur possui todas as ferramentas que projetistas de hardware e software precisam para o desenvolvimento de sistemas de alto desempenho em Dispositivos de Lógica Programável.

A ferramenta de desenvolvimento Excalibur é composta, basicamente, pelos seguintes itens:

- Processador softcore Nios
- Compilador C/C++ GNUPro
- Software de desenvolvimento Quartus II
- Placa de desenvolvimento

3.3.1 O Processador Nios

O processador embutido Nios é um *softcore* de um processador RISC configurável, com um conjunto de instruções de 16 bits e um *datapath* selecionável de 32 ou 16 bits [Altera, 2000a]. Como o Nios foi desenvolvido especificamente para ser usado em projetos que utilizam lógica programável, ele se apresenta de maneira otimizada para ocupar uma reduzida área do dispositivo. Inicialmente sendo destinado para a família de dispositivos APEX da Altera, este processador ocupa apenas 12% da área de um dispositivo de 500 mil portas lógicas (EP20K200E). A família APEX já possui dispositivos com 7 milhões de portas lógicas, isso significa que há uma infinidade de recursos disponíveis para o restante dos sistemas lógicos que podem ocupar o FPGA. Diversos periféricos já estão disponíveis e em desenvolvimento para o NIOS, como timers, contadores, UART, PIO, controladores de disco, 10/100 Ethernet MAC e controladores SDRAM.

As possibilidades de uso do processador Nios são bastante amplas, e sua alta eficiência (até 50MIPS) possibilita também que múltiplas instâncias do processador possam ser usadas para aplicações de alto desempenho, como mostra a Figura 3.2.

Figura 3.2: Flexibilidade e escalabilidade dos processadores embutidos Nios.

O processador Nios pode ser parametrizado, permitindo que o usuário faça o balanço entre área de dispositivo ocupada, funcionalidade e desempenho. Apesar da configuração, o conjunto de instruções permanece o mesmo, possibilitando o uso de ferramentas de desenvolvimento de software padrões de mercado como compiladores C/C++. Um diagrama do Processador Nios pode ser visualizado na Figura 3.3.

Figura 3.3: Processador *softcore* Nios

Característica do *softcore* Nios:

- Barramento de dados de 16 ou 32bits
- Execução de uma instrução por ciclo de clock
- Suporte para memória *on-chip* e *off-chip*
- Desempenho de até 50 milhões de instruções por segundo (50 MIPS)
- Pode operar com um conjunto de até 512 registradores de 32 bits

Periféricos disponíveis juntamente com o Nios

- Receptor/transmissor assíncrono universal (UART)
- Entrada/saída paralela (PIO)
- Timers
- Interface de memórias para SRAM e FLASH

3.3.2 Compilador GNUPro

O compilador e *debugger* GNUPro da Cygnus, uma companhia da Red Hat, é uma ferramenta padrão, usada por fabricantes de software de todo o mundo. Ele é uma ferramenta de desenvolvimento C/C++, otimizada para o processador embutido Nios, o GNUPro proporciona um ambiente de desenvolvimento familiar aos profissionais da área, incluindo:

- Compilador otimizado C/C++
- Assembler GNU
- *Debugger* interno e outros utilitários

3.3.3 Quartus II – Ferramenta de Desenvolvimento

O software de desenvolvimento Quartus II possibilita aos projetistas o gerenciamento de projetos complexos, utilizando-se milhões de portas lógicas de maneira clara e organizada. O software Quartus II suporta soluções em nível de sistema com editoração de blocos, trabalho em grupo, suporte para megafunções e linguagens de descrição de hardware como o VHDL e o Verilog. Além disso, um sistema de análise lógica embutido permite aos projetistas verificar a funcionalidade e a temporização dos projetos desenvolvidos, observando sinais e valores internos do sistema.

3.3.4 Placa de Desenvolvimento

O Excalibur inclui ainda uma placa de desenvolvimento que proporciona aos usuários uma plataforma de hardware, através da qual podem iniciar o desenvolvimento de um sistema sobre o *chip* programável.

A placa de desenvolvimento (Figura 3.5) incorpora os seguintes componentes:

- FPGA APEX EP20K200E (equivalente a 500 mil portas lógicas)
- Memórias Flash (1M byte) e SRAM (256 Kbytes).

- Portas de comunicação paralela, serial (RS-232) e JTAG RS-232
- LEDs, chaves e um display de cristal liquido.

A placa de desenvolvimento pode ser expandida para suportar uma vasta gama de interfaces através de conectores de protótipos, conectores PCI ou SDRAM. O dispositivo FPGA (20k200E) que acompanha o kit de desenvolvimento pertence a família APEX da Altera, ele possui o equivalente a 526.000 portas lógicas e 106.496 bits de memória RAM.

Figura 3.5: Placa de desenvolvimento do kit Excalibur

3.4 O Paradigma SOC – *System On a Chip*

O Paradigma SOC, consiste no desenvolvimento de sistemas completos em um único dispositivo. Ou seja, em apenas um *chip* são embutidos o processador, memória, timers,

interfaces e todos os outros circuitos necessários para o funcionamento de um determinado sistema. Essa idéia tem sido amplamente adotada pela grande maioria de fabricantes da área de hardware. Devido ao seu excelente desempenho e a sua versatilidade, a tecnologia FPGA tem se mostrado perfeita para o desenvolvimento de sistemas SOC.

A ferramenta de desenvolvimento Excalibur oferece todos os recursos necessários para que sistemas SOC sejam desenvolvidos, incluindo um processador-*softcore*, memória, periféricos de I/O e um barramento *on-chip*. A placa de desenvolvimento também apresenta uma memória não volátil do tipo flash, onde os projetos podem ficar armazenados mesmo quando a placa do sistema é desconectada da alimentação elétrica. O processador Nios se comunica com os dispositivos periféricos através do PBM - Módulo de Barramento Periférico (Figura 3.6).

Implementação da Rede Neural que será discutida no Capítulo 6

Figura 3.6: Interface PBM

O PBM é gerado automaticamente pelo software *SOC builder*, baseado nos periféricos requisitados pelo usuário [Altera, 2000]. Ele realiza automaticamente:

- A codificação de endereço para cada periférico.
- A conexão do barramento de dados entre cada periférico e o processador Nios.
- O controle de interrupções para todos os periféricos.

Capítulo 4

Redes Neurais Artificiais

4.1 Introdução

As Redes Neurais Artificiais, ou simplesmente Redes Neurais, são conjuntos de uma ou mais unidades de processamento simples, chamadas de neurônios. Esquematicamente, as RNs (Redes Neurais) são representadas por grafos direcionados, onde os nós representam os neurônios e os arcos representam as interconexões existentes entre eles [Freeman, 1992]. Os neurônios são normalmente organizados em camadas e interligados por um grande número de conexões, geralmente unidirecionais. Em grande parte dos modelos de redes neurais, estas conexões estão associadas a pesos, os quais armazenam o conhecimento representado no modelo e servem para ponderar as entradas recebidas por cada um dos neurônios da rede. O funcionamento destas redes é inspirado em uma estrutura física concebida pela natureza: o cérebro humano.

A utilização das RNs na solução de problemas tem se mostrado bastante interessante, já que a forma como esses problemas são representados internamente pela rede e o paralelismo natural inerente à arquitetura das RNs criam a possibilidade de um desempenho superior ao dos modelos algorítmicos convencionais. A solução de problemas utilizando-se RNs é realizada inicialmente por uma fase de aprendizagem, na qual um conjunto de exemplos é apresentado para a rede, a qual extrai as características necessárias para assimilar a informação fornecida. Essas informações são utilizadas posteriormente para gerar respostas para o problema [Braga, 2000].

Basicamente, as principais características que tornam as redes neurais bastante interessante na resolução de problemas são:

- A capacidade de generalização das informações que lhe são apresentadas.
- A capacidade de aprendizado através de exemplos.

Através dessas duas características importantes, as RNs utilizam alguns exemplos que lhe são apresentados para extrair conhecimento e posteriormente fornecer respostas corretas para conjuntos de dados desconhecidos. Isso prova que as RNs não fazem apenas o mapeamento entre entradas e saídas de dados, mas também extraem informações de exemplos apresentados a elas.

4.2 O Neurônio Biológico

Os Neurônios biológicos são divididos em três partes distintas: o corpo da celular, os dendritos e o axônio (Figura 4.1). Os dendritos são responsáveis por receber os impulsos nervosos (informações) provenientes de outros neurônios e conduzi-los até o corpo celular, onde as informações são processadas, e novos impulsos são gerados. Estes impulsos são transmitidos aos dendritos dos outros neurônios através do axônio. O ponto de contato entre a terminação axônica de um neurônio e o dendrito de outro neurônio é onde ocorrem as sinapses. As sinapses funcionam como válvulas, capazes de controlar a transmissão de impulsos, isto é, o fluxo da informação através da rede neural. Se o percentual de impulsos recebidos em um intervalo curto de tempo é suficientemente alto, a célula é ativada, produzindo um impulso que é transmitido para as células seguintes. É dessa forma que ocorrem a maioria das operações realizadas pelo nosso cérebro. Através da operação em paralelo dos, aproximadamente, 100.000.000 neurônios, o nosso cérebro é capaz de realizar tarefas complexas.

Figura 4.1: Neurônio Biológico

4.3 O Modelo de Neurônio de McCulloch e Pitts

Segundo a literatura, o primeiro modelo artificial de um neurônio biológico foi proposto por Warren McCulloch e Walter Pitts em 1943 [Haykin, 1999]. Esse modelo é baseado no que se conhecia sobre o neurônio biológico até então. Matematicamente, o modelo de McCulloch e Pitts funciona da seguinte forma. Os n terminais de entrada (dendritos) são representados por $x_1, x_2, x_3, \dots, x_n$ o terminal de saída (axônio) é representado por y . Cada uma das entradas x_i do neurônio é associada a um valor de peso w_i , dessa forma simulando as sinapses. Os valores de w positivos representam sinapses excitatórias e os valores negativos, inibitórias. O resultado da sinapse de um determinado neurônio i é calculado através de $x_i * w_i$ (Figura 4.2).

Um neurônio biológico dispara quando a soma dos impulsos que ele recebe ultrapassa o seu limiar de excitação (*threshold*). O corpo do neurônio, por sua vez, é emulado por um mecanismo simples que faz a soma dos valores $x_i * w_i$ recebidos pelo neurônio (soma ponderada) e decide se o neurônio deve ou não disparar (saída igual a 0 ou 1), comparando a soma obtida com seu limiar ou *threshold*. No modelo MPC (McCulloch e Pitts), a ativação do neurônio é obtida através da aplicação de uma “função de ativação”,

que ativa ou não a saída, dependendo do valor da soma ponderada de suas entradas. Na descrição original do modelo MCP, a função de ativação é dada pela função de limiar descrita na equação 4.1. O nodo MCP terá então sua saída ativa quando:

$$\sum x_i * w_i \geq \theta \quad (4.1)$$

Onde n é o número de entradas do neurônio, w_i é o peso associado à entrada x_i e θ é o limiar (*threshold*) do neurônio.

Figura 4.2: Neurônio MCP

A partir do modelo MPC, foram derivadas algumas variações da função de transferência, sendo que algumas delas permitem uma saída que não necessariamente seja 0 ou 1. Na Figura 4.3 podem ser observadas algumas das funções de transferência mais constantemente utilizadas.

Figura 4.3: Funções de Transferência

4.4 Aprendizado

A etapa de aprendizagem é o passo inicial para a que uma rede neural seja utilizada na solução de um problema. Durante a aprendizagem, a rede cria uma representação particular do problema através da extração de informações a partir dos dados fornecidos a ela. Essa representação é mantida na forma de pesos associados as conexões entre os neurônios.

Basicamente, existem duas formas distintas de aprendizado. O aprendizado supervisionado e o não-supervisionado. Outros dois paradigmas bastante conhecidos são os de aprendizado por reforço (que é um caso particular de aprendizado supervisionado) e aprendizado por competição (que é um caso particular de aprendizado não-supervisionado) [Braga, 2000].

4.4.1 Aprendizado Supervisionado

O aprendizado supervisionado consiste em fornecer a rede um conjunto de dados de entrada e o valor das saídas correspondentes. Com base nos valores de entrada, a RN calcula o valor de saída. Caso esse valor seja diferente do valor desejado é calculado o erro (diferença) entre a saída obtida e a saída desejada. Com base nesse erro, a RN realiza atualizações nos pesos dos neurônios com o objetivo de que a saída obtida pela rede se aproxime ao máximo da saída desejada. Esse método é indicado quando existem diversos conjuntos de entrada e saída para que a RN possa obter informações suficientes para fazer uma boa generalização a partir dos dados apresentados (Figura 4.4).

Figura 4.4: Aprendizado Supervisionado

A principal desvantagem desse método de aprendizado é que nem sempre é possível se obter previamente o resultado de saída para todos os conjuntos de dados de entradas.

4.4.2 Aprendizado por Reforço

O aprendizado por reforço pode ser visto como um caso particular de aprendizado supervisionado. A principal diferença entre o aprendizado supervisionado clássico e o aprendizado por reforço consiste na atualização dos pesos, a qual não é baseada no cálculo da diferença entre a saída desejada e a saída obtida. No aprendizado por reforço, a única

informação de realimentação fornecida à rede é se uma determinada saída está correta ou não, isto é, não é fornecida à rede a resposta correta para o padrão de entrada. Com base nessas informações, a RN vai adquirindo conhecimento (aprendendo).

4.4.3 Aprendizado Não-Supervisionado

Nos algoritmos de aprendizado não-supervisionado, somente os padrões de entrada estão disponíveis para a rede, ao contrário do aprendizado supervisionado, cujo conjunto de treinamento possui pares de entrada e saída. A partir do momento em que a rede estabelece uma harmonia com as regularidades estatísticas da entrada de dados, desenvolve-se nela uma habilidade de formar representações internas para codificar características da entrada e criar novas classes ou grupos automaticamente. Este tipo de aprendizado só se torna possível quando existe redundância nos dados de entrada. Sem redundância seria impossível encontrar quaisquer padrões ou características nos dados de entrada.

4.4.4 Aprendizado por Competição

O aprendizado por competição é um caso particular de aprendizado não-supervisionado. A idéia nesse caso é que dado um padrão de entrada, deve-se fazer com que as unidades de saída disputem entre si para serem ativadas. Existe, portanto, uma competição entre as unidades de saída para decidir qual delas será a vencedora e, conseqüentemente, terá sua saída ativada e seus pesos atualizados no treinamento. As unidades de entrada são diretamente conectadas às unidades de saída, e estas últimas também podem estar ligadas entre si, via conexões laterais inibitórias ou negativas. A unidade de saída com maior ativação inicial terá maior chance de vencer a disputa das outras unidades, os quais perderão o poder de inibição ao longo do tempo sobre a unidade de maior ativação. Desse modo, a unidade mais forte fica ainda mais forte, e seu efeito inibidor sobre as outras unidades de saída torna-se dominante. Com o tempo, todas as outras unidades de saída ficarão completamente inativas, exceto a vencedora. Este tipo de inibição mútua também é conhecido como *winner takes all*.

4.5 Os Modelos Perceptron e Adaline

O perceptron é uma das formas mais simples de rede neural usada na classificação de padrões que sejam linearmente separáveis. Basicamente, o perceptron consiste em um único neurônio com pesos sinápticos ajustáveis. Segundo a literatura, a convergência do algoritmo de aprendizado do perceptron foi provada em 1962 por Rosenblatt [Haykin, 1999]. Uma rede perceptron (com um único neurônio) pode classificar apenas dois padrões distintos (que sejam linearmente separáveis). O algoritmo de aprendizado do perceptron pode ser visto na equação 4.2.

$$w_i(t+1) = w_i(t) + \eta \cdot e \cdot x_i(t) \quad (4.2)$$

onde:

$$e = d - y$$

O termo d representa a saída esperada da rede, enquanto y é a saída obtida pela rede. Portanto, e significa a diferença entre a saída esperada e a obtida, $w_i(t)$ é o valor de peso correspondente à entrada i no tempo t , η é o valor da taxa de aprendizado, e $x_i(t)$ representa o valor da entrada i no tempo t . Através dessa regra, todos os pesos devem ser corrigidos até que ocorra a convergência da rede, e esta apresenta os resultados desejados a todas as entradas.

Quase que simultaneamente ao modelo perceptron, surge o modelo adaline (*adaptive linear neuron*), proposto por Widrow e Hoff [Haykin, 1999]. A grande diferença entre os algoritmos de treinamento dos modelos perceptron e adaline consiste que no modelo adaline a correção dos pesos é baseada na saída linear (antes da função de ativação). Esse algoritmo de aprendizado (conhecido por regra delta) deu origem ao famoso algoritmo back-propagation, muito utilizado no treinamento de redes neurais multicamadas. A Figura 4.5 apresenta a rede neural adaline.

Figura 4.5: Modelo ADALINE

4.6 As Redes MLP (Multi Layer Perceptron)

Conforme já foi visto, as redes de uma só camada resolvem apenas problemas linearmente separáveis. A solução de problemas não linearmente separáveis passa pelo uso de redes com uma ou mais camadas intermediárias ou escondidas. Uma rede com uma camada intermediária pode implementar qualquer função contínua, e a utilização de duas camadas intermediárias pode implementar qualquer função. Um exemplo de rede MLP pode ser visto na Figura 4.6.

Figura 4.6: Rede Neural MLP

O algoritmo mais conhecido para treinamento destas redes é o algoritmo *back-propagation* [Haykin, 1999]. O algoritmo *back-propagation* é baseado na regra delta, sendo por isso também chamado de regra delta generalizada. Através desse algoritmo pode-se estimar o valor do erro nas camadas intermediárias da rede neural e, conseqüentemente, ajustar os pesos dos neurônios. Os ajustes são realizados utilizando-se o método dos gradientes, segundo a equação 4.3.

$$W_{ji}(t+1) = W(t) + \eta \cdot \delta_j(t) \cdot x_i(t) \quad (4.3)$$

onde:

$$\delta_j(t) = e(t) \cdot y(t) \cdot (1 - y(t)) \quad (\text{para a última camada})$$

$$\delta_j(t) = y(t) \cdot (1 - y(t)) \cdot \sum (\delta(t) \cdot W(t)) \quad (\text{para as camadas intermediárias})$$

A função de transferência ou ativação $y(t)$ é a função sigmoide, apresentada na Figura 4.3d.

4.7 Redes Neurais Aplicadas a Robótica

Existe uma infinidade de aplicações para as Redes Neurais. Dentre elas, podem ser citadas: o processamento de imagens, detecção de padrões e a robótica. Dentro da área da robótica, as Redes Neurais são de grande utilidade devido a sua capacidade de aprendizado e generalização. Alguns exemplos de RNs aplicadas a problemas de robótica são apresentados a seguir.

4.7.1 O Projeto ALVINN

O projeto ALVINN (Autonomous Land Vehicle in a Neural Network), desenvolvido por Pomerleau na Carnegie Mellon University, consiste em um veículo guiado automaticamente [Pomerleau, 1991]. O veículo possui uma câmera de vídeo que registra as imagens da pista diante do mesmo. Através do processamento dessas imagens obtêm-se a direção em que o veículo deve seguir. A interpretação das imagens e a sugestão da direção a ser seguida são realizadas por uma rede neural do tipo MLP (Figura 4.7).

Figura 4.7: Projeto ALVINN

Cada frame de imagem possui resolução de 30x32, sendo que cada um desses 960 pixels corresponde exatamente a um neurônio da camada de entrada da RN. Todos os 960 neurônios da primeira camada são conectados a 5 neurônios na camada intermediária que por sua vez, são conectados a 30 neurônios na camada de saída. Cada um dos neurônios de

saída representa uma direção à ser tomada pelo veículo. O neurônio mais à esquerda sugere que o veículo faça uma curva acentuada para a esquerda. O neurônio mais à direita sugere que o veículo faça uma curva acentuada para a direita. E os neurônios do centro sugerem que o veículo continue em frente (Figura 4.7).

Inicialmente a RN foi treinada com um conjunto de 1200 imagens de pista e suas respectivas reações desejadas. Para que o treinamento apresentasse bons resultados, era necessário que cada imagem fosse apresentada cerca de 30 a 40 vezes para a RN. O tempo de treinamento era de cerca de 6 horas para esse conjunto de dados. Apesar do aparente sucesso obtido com esse treinamento, havia alguns problemas quando o veículo era exposto a situações reais. Frequentemente, existiam muitas diferenças entre as imagens utilizadas para o treinamento da RN e as imagens reais, o que resultava em um desempenho mediano. A solução encontrada por Pomerlau foi realizar o treinamento da RN enquanto o veículo estava em movimento (guiado por um motorista), ou seja, a RN deveria aprender a imitar as reações do motorista diante das diversas situações captadas pela câmera. O sistema mantinha um conjunto de 200 imagens para realizar o aprendizado da RN. A partir de cada imagem obtida pela câmera, eram geradas 15 variações da mesma, que substituíam outras 15 imagens do conjunto de 200. Após cerca de 5 minutos de aprendizado, a RN já estava pronta para poder guiar o veículo automaticamente [Pomerlau, 1991].

4.7.2 Planejamento de Percurso

Em um trabalho desenvolvido por [Glasius, 1995], na Universidade de Nijmegen, uma RN do tipo Hopfield foi utilizada para realizar o planejamento do percurso que um robô móvel deve seguir dentro de um labirinto.

Inicialmente, foram apresentados à RN o mapa do labirinto, o ponto de partida do robô e o destino que o mesmo deveria atingir. O objetivo da RN era criar uma representação desse labirinto e encontrar um caminho que levasse o robô até o destino previsto. No caso de mais de uma possibilidade de percurso, o caminho mais curto deveria ser escolhido. A representação do mapa pode ser vista na Figura 4.8.

Figura 4.8: Representação do Mapa

A RN utilizada possui 2500 neurônios, organizados em uma matriz de 50x50. Os pontos S e T (Figura 4.8) representam, respectivamente, a posição inicial do robô e a posição de destino, a qual o mesmo deve alcançar. Os neurônios de cor escura representam os obstáculos que devem ser evitados pelo robô, e os neurônios de cor clara representam o caminho que deve ser percorrido pelo robô [Glasius, 1995].

4.7.3 Mapeamento de Ambiente e Posicionamento

O trabalho de [Marquès, 1996] apresenta um sistema de mapeamento de ambientes dinâmicos e posicionamento do robô dentro dos mesmos. Para tanto, foi utilizada uma Rede de Kohonem. As informações passadas à RN eram obtidas através do sistema odométrico do robô e de sensores (sonares e lasers) .

A Figura 4.9 mostra algumas etapas do processo de aprendizado. Quanto mais são realizadas as iterações no algoritmo de aprendizado, maior é a precisão da representação do mapa real pela RN.

Figura 4.9: Representação do mapa pela RN

O mapa real é visto na Figura 4.9 (a). As Figuras 4.9 (b), (c) e (d) mostram a representação da RN com 250, 375 e 500 iterações, respectivamente.

Muitos outros exemplos de aplicações de RNs em robótica podem ser encontrados na literatura [Omidvar, 1997]. A capacidade de aprendizado e representação de conhecimento torna as RNs muito interessantes na resolução de diversos problemas comuns à robótica como mapeamento, localização, processamento de imagens e etc.

Capítulo 5

Robôs Móveis

5.1 Introdução

Nos últimos anos tem-se observado um vertiginoso crescimento do potencial de sistemas robóticos. Numa primeira etapa houve um grande desenvolvimento na área de robótica industrial. A implantação no chão de fábrica de robôs industriais trouxe benefícios como a melhoria da eficácia, da qualidade, redução da mão-de-obra, além de mais eficiência, confiabilidade e redução de custos. Algumas vantagens adicionais incluem a capacidade de realizar tarefas para as quais os humanos teriam grande dificuldade e a remoção de humanos de tarefas em ambientes perigosos [Aragão, 2000]. A Figura 5.1 apresenta uma garra mecânica, um tipo de robô muito utilizado em fábricas para trabalhos pesados e precisos.

Figura 5.1: Robô Garra

Numa nova etapa de evolução, pesquisadores em robótica têm concentrado esforços na construção de Robôs Móveis, introduzindo as capacidades de mobilidade e autonomia aos sistemas robóticos, o que abre um vasto campo de novas aplicações (Figura 5.2).

Figura 5.2: Robôs Móveis

Atualmente, muitos institutos de pesquisa e empresas vêm trabalhando no desenvolvimento de sistemas robóticos que se assemelhem a seres vivos (humanos, cachorros etc.), proporcionando grandes desafios para os pesquisadores da área (Figura 5.3)

Figura 5.3: Robôs que imitam seres vivos

Geralmente, os problemas enfrentados no desenvolvimento de robôs móveis estão relacionados com a necessidade de interagir com os objetos físicos e entidades do ambiente. A plataforma deve ser capaz de navegar a partir de uma posição conhecida para uma nova localização desejada, evitando colisões com objetos fixos e móveis durante sua rota. Para funcionar em um ambiente desestruturado, desconhecido ou dinâmico, um robô móvel deve ser capaz de perceber as circunstâncias e gerar ações que são apropriadas para o ambiente e para os objetivos do sistema robótico [Medeiros, 1998].

5.2 Algoritmos de Robótica

Os algoritmos de robótica são, em sua maioria, descrições abstratas dos processos de percepção e interação com o ambiente. Enquanto os algoritmos computacionais tradicionais, normalmente, têm total controle e completo acesso aos dados que utilizam, os algoritmos de robótica são aplicados a objetos físicos do mundo real, os quais nem sempre podem ser previstos e descritos com precisão. Conseqüentemente, algoritmos de robótica possuem características clássicas da teoria de controle, normalmente ausentes em algoritmos computacionais convencionais. A complexidade do ambiente e suas possíveis variações para a realização da mesma tarefa também levantam questões combinatoriais, tais como encontrar um caminho livre de colisões com obstáculos e confrontar dados de sensores com um modelo padrão. Esta mistura singular de questões relativas a controle e algoritmos de programação são, praticamente, o núcleo da "Ciência Robótica" [Latombe, 1995].

Grande parte dos pesquisadores em robótica enfoca prioritariamente o desenvolvimento de algoritmos, pois consideram que a principal limitação para a construção de robôs está em como controlá-los, ou seja, como criar programas capazes de operar estas máquinas complexas [Brooks, 1986]. Mas, também é fundamental que esses algoritmos desenvolvidos sejam executados com a eficiência necessária para que seus objetivos sejam alcançados.

Muitos dos algoritmos utilizados em robótica trabalham com grande quantidade de informação e precisam ser executados com tempo de resposta curto. Um exemplo disto são os algoritmos que controlam navegação baseados em processamento de imagens. Muitas vezes, implementações diretamente em hardware desses algoritmos se fazem necessárias para que os robôs manipulem grandes quantidades de dados com um tempo de resposta satisfatório para os objetivos que foram propostos.

5.3 Desafios da Robótica

Muitos dos problemas na área de robótica ainda se encontram sem soluções definitivas. Algumas dessas questões em aberto são descritas a seguir [Jones, 1999].

5.3.1 Navegação

Um dos grandes problemas em aberto dentro da área da robótica é a navegação. Uma das prováveis dificuldades de se obter um sistema que permita a robôs móveis navegar de forma eficiente é a representação do ambiente. Existem diversas pesquisas na área que tentam apresentar maneiras eficientes de organizar dados, de forma a obter uma representação clara e precisa do ambiente em que o robô se encontra. Esses dados podem ser adquiridos através de diversos tipos de sensores, cada tipo com sua particularidade. Apesar de existirem tantas formas de se capturar dados a respeito do ambiente, e diversas maneiras de se organizar esses dados, a maioria dos sistemas de navegação ainda se apresentam extremamente limitados a situações e ambientes específicos. A necessidade de sistemas de navegação eficientes é uma das prioridades dentro da área de robôs móveis.

5.3.2 Reconhecimento

Outro problema que, juntamente com a navegação, vem sendo bastante abordado em recentes pesquisas na área é o reconhecimento. É fundamental que um robô possa interpretar de forma clara os dados que são obtidos através dos seus sensores. A localização

através de *Landmarks* (pontos de referência previamente conhecidos), por exemplo, é uma maneira do robô saber onde este se encontra dentro de um determinado contexto, sendo esse um dos passos fundamentais para que o problema de mapeamento de uma região possa ser obtido com êxito. Através dos *Landmarks*, o robô sabe exatamente de sua localização e com essa informação ele pode desenvolver ou atualizar mapas com precisão.

O reconhecimento de objetos também é um assunto de grande interesse entre os pesquisadores da área. Muitos robôs são construídos com o objetivo de realizar busca de determinados objetos. Por exemplo, um robô que deve fazer manutenção em um determinado equipamento precisa identificar esse equipamento com razoável precisão. O reconhecimento de objetos pode ser realizado de diversas maneiras e uma das mais comuns é através de câmeras de vídeo. Sendo o processamento de imagens uma tarefa de extrema complexidade computacional, é de grande utilidade que sejam desenvolvidos algoritmos eficientes para o processamento dessa informação e também, sistemas de hardware capazes de processar essas informações com a eficiência requerida.

A interface entre homens e robôs, que é outra importante área de pesquisa dentro da robótica, também consiste em um problema de reconhecimento. Essa interface pode ser realizada de diversas maneiras. Duas dessas maneiras que têm obtido destaque em pesquisas recentes são as interfaces através de voz e de gestos. Essas duas diferentes abordagens têm algumas características em comum. Ambas exigem o tratamento eficiente de um grande volume de dados que podem ser captados através de câmeras de vídeo e microfones. Normalmente, esses dados sofrem forte influência de ruídos o que acaba tornando muito difícil a interpretação correta desses sinais. É de fundamental importância que algoritmos inteligentes suficientemente para extrair informações úteis a partir de um grande volume de dados captados por esses sensores sejam desenvolvidos. Normalmente, métodos computacionais de Inteligência Artificial como, por exemplo, as Redes Neurais são utilizadas nesse tipo de tarefa, devido a sua notável capacidade de realizar o reconhecimento de padrões. Esse tipo de sistema constantemente requer alta capacidade de processamento para ser executado com sucesso.

5.3.3 Aprendizado

Com o passar do tempo, os sistemas robóticos vêm se tornando cada vez mais complexos e sofisticados. Os robôs móveis, a cada dia, vêm incorporando cada vez mais sensores (e esses têm adquirido cada vez mais acuidade) e atuadores (e esses têm adquirido cada vez mais precisão). Dessa forma, se faz extremamente necessário o desenvolvimento de algoritmos de aprendizado que utilizem toda essa informação de forma eficiente. Atualmente, existem muitos pesquisadores da área investigando o desenvolvimento de como esses algoritmos devem ser desenvolvidos para se obter o máximo de proveito de toda essa evolução dos periféricos que são acoplados aos robôs. Existem ainda muitas perguntas sem respostas dentro dessa área, tais como. Qual é a maneira mais eficiente de aprender através dos dados obtidos pelos sensores? Um robô pode aprender a re-calibrar seus sensores? Um robô pode aprender novos comportamentos por si só?

5.3.4 Cooperação

Dois ou mais robôs são mais eficientes do que apenas um? Quando utilizar mais de um robô para realizar uma determinada tarefa? A utilização de equipes de robôs para resolver determinados problemas é uma área de estudo que têm se tornado bastante interessante entre os pesquisadores. Equipes de robôs apresentam vantagens quando:

- O objetivo pode ser decomposto em tarefas que podem ser executadas em paralelo.
- Existe uma tarefa que seria impossível de ser realizada por apenas um agente (robô).
- Precisa-se de alta confiabilidade na execução de determinada tarefa. Dessa forma, é criado um sistema de tolerância a falhas com diversos robôs realizando a mesma tarefa.

Designar equipes de robôs para realizar determinadas tarefas é uma operação que exige complexos algoritmos responsáveis pela organização do grupo de robôs. Para que esses robôs realizem suas tarefas de forma satisfatória, normalmente, são necessários

complexos sistemas de comunicação entre os robôs. Nem sempre essa comunicação é explícita entre as máquinas, mas a troca de informações é fundamental para que a produtividade alcançada seja a maior possível.

Todos os problemas apresentados anteriormente ainda permanecem sem soluções definitivas, propondo grandes desafios para os pesquisadores que se aventuram nessa fascinante área de pesquisa que é a robótica.

5.4 O Projeto ARMOSH

O projeto ARMOSH - Aprendizado em Robôs Móveis via Software e Hardware [Romero, 2000] propõe o estudo e implementação de algoritmos de aprendizado de robôs móveis tanto no nível de software quanto no nível de hardware, visando uma comparação de desempenho dos principais algoritmos de aprendizado em ambas as implementações

5.4.1 Objetivos do Projeto ARMOSH

O projeto tem a finalidade de buscar novas alternativas na área, inicialmente através do estudo, trabalhando com implementações tanto no nível de software quanto no nível de hardware dos principais algoritmos de aprendizado de máquina, onde serão abordados:

- Algoritmos para navegação, levando-se em conta dependência do número de sensores do robô (sonar, infravermelho ou câmera de vídeo), definição do ambiente (fechado, aberto, desconhecido).
- Algoritmos que envolvem aprendizado via Programação Dinâmica, via Redes Neurais e aprendizado com Reforço.
- Algoritmos que envolvem reconhecimento de objetos via Redes Neurais, via técnicas clássicas de processamento de imagens tais como, forma, textura e cor, e em ambientes estáticos e dinâmicos.

O estudo levantado neste projeto será direcionado com vistas a sua evolução futura, através da construção de uma biblioteca dos algoritmos estudados em sua versão hardware/software e na montagem de um kernel, a longo de prazo, que controlará de forma inteligente a execução destes algoritmos, proporcionando grande versatilidade ao sistema, pois em tempo real, o sistema poderá decidir qual algoritmo é mais adequado para realização de uma determinada tarefa, utilizando ainda, a melhor implementação (via hardware/software) do algoritmo escolhido.

Em sua etapa inicial, o Projeto ARMOSH vem apresentando trabalhos nas áreas de interfaces homem/robô e navegação .

5.4.2 Interfaces Homem/Robô

Uma das áreas que tem chamado grande atenção dos pesquisadores é o desenvolvimento de interfaces homem/robô, porque a maioria dos robôs existentes não possui uma interface amigável. Portanto, a necessidade de interfaces homem/robô mais amigáveis tem sido claramente reconhecida. De uma forma geral, infelizmente, os modelos de interfaces ainda se mantêm longe de se conseguir estabelecer uma comunicação simples e eficaz com pessoas comuns (não treinadas previamente para interagir com o robô) [Romero, 2000]. Uma experiência muito interessante deste tipo de aplicação foi obtida pelo robô RHINO [Burgard, 1998] no museu *Deutsches Museum Bonn* na Alemanha. Os visitantes ficam no máximo 15 minutos em frente a um robô e seria indesejável que estes tivessem que aprender como interagir com um robô, ainda que a natureza da interface por gestos facilite este processo. Existe, claramente, a necessidade de uma interface que possibilite o robô interagir de uma forma mais efetiva e natural com o usuário. Portanto, a área de interfaces entre pessoas e robôs ainda se apresenta como um vasto e promissor campo para a pesquisa na área de robótica.

Atualmente, duas das mais importantes formas de interfaces entre o homem e os robôs são as interfaces através do reconhecimento de voz e através de gestos. Uma proposta recente desenvolvida por [Waldherr, 1999] permite que robôs móveis consigam reconhecer gestos dinâmicos; isto é, gestos que são definidos por movimentos e não apenas por gestos estáticos. Gestos dinâmicos, que são mais comumente usados para comunicação entre

pessoas, fornecem um grau de liberdade a mais no projeto de gestos. Além disso, eles reduzem as chances de classificação incorreta de determinados gestos. Devido aos excelentes resultados obtidos, a pesquisa realizada por Waldherr será utilizada como base para a pesquisa proposta nessa dissertação. Maiores detalhes serão apresentados no Capítulo 6.

5.4.3 Navegação

Navegação é a ciência e/ou tecnologia de encontrar a posição, o curso e a distância percorrida por um veículo qualquer. O sistema de navegação de robôs móveis necessita de um modelo de aprendizagem de direção com suporte para navegação, pois o sistema precisa saber onde se encontram os objetos para poder prever ou até mesmo planejar suas ações futuras. O modelo de aprendizagem tem como referência os dados obtidos a partir dos sensores de navegação (câmeras, sonares, etc), os quais contém as informações sobre o mundo real, auxiliando no processo de navegação [Scatena, 2000].

Um eficiente sistema de mapeamento é essencial para que a operação de navegação seja realizada com sucesso. Dentre os diversos algoritmos de navegação existentes, destacam-se os mapas baseados em grid e os mapas topológicos.

- **Mapas baseados em grid:** são mapas que são construídos a partir de matrizes de células, onde cada célula contém as medidas de certeza em que as áreas correspondentes estão ocupadas por objetos ou não (Figura 5.4 a).
- **Mapas topológicos:** o lugar reconhecido é representado como um grafo de lugares conectados. Neste mapeamento não é preciso haver um prévio conhecimento das coordenadas cartesianas (Figura 5.4 b).

Figura 5.4: Mapas (a) baseados em grid e (b) topológicos

Ambos os algoritmos descritos anteriormente apresentam vantagens e desvantagens. O mapeamento baseado em grid possui uma interpretação geométrica detalhada do ambiente. Porém, o seu custo computacional é muito alto. O mapeamento topológico possui um baixo custo computacional, mas não apresenta uma interpretação geométrica detalhada do ambiente [Duckett, 2000].

5.4.4 Relevância do Projeto ARMOSH

Apesar de haver um grande número de pesquisas em sistemas de navegação, interfaces e diversas outras áreas da robótica, ainda não existem soluções definitivas capazes de realizar as tarefas exigidas no mundo real com total eficiência e flexibilidade. A medida que a tendência para a utilização de robôs móveis aumenta, a necessidade de sistemas de navegação, de localização e de interfaces para interação homem/máquina também aumenta na mesma proporção.

Apesar de apenas alguns pontos de pesquisa terem sido levantados, pode-se notar que existe muito trabalho ainda a ser realizado na área de Robótica. O que é proposto no projeto ARMOSH é a construção de um sistema que disponha de diferentes metodologias de aprendizado de robô e de maneira inteligente. Para este sistema, inicialmente, será construída uma biblioteca dos algoritmos estudados em sua versão hardware/software. A

implementação de hardware dos algoritmos escolhidos usará a tecnologia de computação reconfigurável.

Num futuro próximo, pretende-se construir um kernel (Figura 5.5) que controlará de forma inteligente a execução dos algoritmos desta biblioteca para ser usado num robô móvel a ser construído no ICMC (Instituto de Ciências Matemáticas e de Computação) - USP. O fato da implementação de hardware estar usando a tecnologia de computação reconfigurável permitirá a construção de um robô projetado com circuitos integrados que são reconfiguráveis em tempo de execução (sem que o hardware seja desligado), dando ao robô a possibilidade de trabalhar com diversos tipos de hardware numa mesma implementação. Isto viabilizará a obtenção de um sistema mais flexível e de mais fácil adaptação a diferentes ambientes [Romero, 2000]. Por exemplo, com esta tecnologia, o robô terá condições de escolher quais serão os melhores algoritmos de navegação e visão (por exemplo) para aquele determinado ambiente que acabara de ser detectado, reconfigurando seu hardware automaticamente (Figura 5.5).

Figura 5.5: Kernel do Robô, responsável pelo controle dos FPGAs

Capítulo 6

Desenvolvimento do Hardware

6.1 Objetivos

O objetivo deste projeto de pesquisa é a implementação de um Hardware Dedicado de uma RN do tipo MLP, usada no reconhecimento de gestos por Robôs Móveis.

Como já foi mencionado anteriormente, essa dissertação está relacionada ao projeto ARMOSH, o qual irá realizar e avaliar implementações em hardware dos algoritmos de software utilizados na área de robótica.

6.1.1 O Reconhecimento de Gestos

O sistema de reconhecimento de gestos dinâmicos que foi utilizado como base para essa pesquisa, foi desenvolvido na *Carnegie Mellon University* por [Waldherr, 1999]. Esse sistema funciona da seguinte forma. O robô recebe imagens através de uma câmera de vídeo NTSC. Essas imagens são originalmente coloridas, têm resolução de 320x240 pixels (Figura 6.1a) e passam por um estágio de tratamento antes de serem efetivamente processadas. Esse pré-processamento consiste, inicialmente, na identificação da região onde se encontra o braço da pessoa que passa instruções para o robô. Essa região é extraída da imagem (agora com resolução de 10x10 pixels) e tem suas cores originais convertidas para tons de cinza. A partir dessa nova imagem obtida é realizado o processamento para a interpretação dos gestos presentes na imagem (Figura.6.1b).

(a)

(b)

Figura 6.1: (a) Imagem original captada pela câmera, (b) Imagem pré-processada para a análise do gesto.

Os diferentes comandos enviados ao robô são realizados de acordo com o ângulo formado pelo antebraço e o corpo e também pelo ângulo formado com o braço e o antebraço. A partir da imagem pré-processada, um algoritmo será capaz de identificar os dois ângulos desejados. A partir desses ângulos, pode-se determinar qual é o gesto que está sendo mostrado ao robô naquele determinado instante. Sabendo-se qual é o gesto que está sendo recebido, o robô pode então proceder de maneira adequada, conforme as instruções que lhe foram passadas. O algoritmo responsável pela análise da imagem e pela identificação dos dois ângulos necessários, o qual identifica o gesto recebido pelo robô, é baseado em uma RN do tipo MLP. Esse algoritmo é executado em um microcomputador on-board (laptop). Ou seja, um processador de propósito geral do tipo Intel Pentium é responsável pelo processamento dos dados de imagens.

A RN utilizada no processamento das imagens captadas pelo robô possui uma topologia de $100 \times 100 \times 60$, ou seja, 100 neurônios na camada de entrada, 100 neurônios na camada intermediária e 60 neurônios na camada de saída. Os 100 neurônios da camada de entrada recebem valores referentes aos 100 pixels (10×10) das imagens que devem ser processadas. Os resultados apresentados pelos 60 neurônios na camada de saída são

interpretados da seguinte maneira. Esses neurônios são separados em dois grupos de 30 neurônios. O resultado apresentado por cada grupo de 30 neurônios equivale a um determinado ângulo (entre o braço e o antebraço e entre o antebraço e o tronco), apresentado na imagem. Com as informações dos dois ângulos é possível determinar qual é o gesto que está sendo indicado para o robô e, conseqüentemente, qual a atitude que deve ser tomada pelo robô. Como por exemplo, um braço levantado significa parar o robô.

A maioria dos algoritmos de RNs utiliza um grande volume de dados e operações aritméticas, o que implica em um altíssimo custo computacional. Frequentemente, esses algoritmos são os responsáveis pela baixa eficiência dos sistemas computacionais devido a sua lentidão na execução. Uma forma de se obter um processamento mais eficiente e um melhor desempenho é através do desenvolvimento de circuitos de hardware específicos para a execução desses algoritmos. A pesquisa apresentada nessa dissertação é exatamente o projeto e o desenvolvimento de um circuito de hardware dedicado, equivalente à RN descrita anteriormente. Como grande parte dos algoritmos de robótica utilizam RNs, como já foi mostrado no Capítulo 4, essa pesquisa tem também o objetivo de contribuir para futuras implementações de algoritmos que envolvam RNs.

6.2 O Desenvolvimento de Redes Neurais em Hardware

O paralelismo natural e a simplicidade da maioria das operações matemáticas presentes nos algoritmos de RNs tornam esses algoritmos bastante interessantes para serem implementados em hardware. O hardware funciona de uma forma paralela, aproveitando o paralelismo inerente das RNs. Além disso, os sistemas implementados em hardware possuem excelente desempenho quando comparados com implementações dos mesmos algoritmos em processadores de propósito geral. Dessa forma, o desenvolvimento de RNs em hardware é um assunto que tem sido amplamente pesquisado e discutido na área científica. Existem vários trabalhos como: [Moreno, 1999], [Moreno, 1999a], [Grossi, 1995] e [Moerland, 1997] discutem as possibilidades de implementação de RNs em

hardware. Alguns trabalhos que implementam as RNs por hardware serão apresentados a seguir:

O projeto GANGLION [Cox, 1992], desenvolvido por pesquisadores da IBM em 1992, apresenta a implementação de uma RN do tipo MLP com topologia de 12x14x4. O GANGLION era utilizado para realizar segmentação de imagem. Devido às limitações da tecnologia FPGA, na época, foram utilizados dezenas de dispositivos FPGA na implementação do sistema, além de diversos dispositivos de memória para o armazenamento dos resultados parciais. Cada um dos 14 neurônios da camada intermediária era composto por dois *chips* FPGA Xilinx XC3090 e por duas unidades de memória PROM de 2 kbytes cada uma. A camada de saída também era formada por dois chips Xilinx XC3090 e por um *chip* Xilinx XC3042, além de duas unidades de memória PROM de 2kbytes cada uma. O sistema operava em uma velocidade de clock de 20 MHz e apresentava um desempenho incrível para aquela época. O sistema era capaz de processar 20 milhões de pixels por segundo, o que representava o dobro do que uma câmera era capaz de gerar naquela época.

O trabalho desenvolvido por Schoenauer [Schoenauer, 2000] na Universidade de Berlin, consiste na implementação de uma RN do tipo *pulse-coded*, também utilizada para segmentação de imagem, a qual foi simulada e obteve excelentes resultados. Foi simulado o comportamento de uma RN com 1000 neurônios. Os resultados obtidos com a simulação foram comparados a um algoritmo equivalente executado em uma máquina do tipo Alpha de 500 MHz. Os resultados da simulação do hardware apresentaram desempenho centenas de vezes superior ao software executado no processador Alpha.

O trabalho desenvolvido por Pérez-Urbe [Pérez-Urbe, 1996] consiste na implementação de uma RN do tipo FAST (*Flexible Adaptable-Size Topology*), utilizada na segmentação de imagens (Figura 6.2). A RN era composta por 6 neurônios e a implementação foi realizada com um microcontrolador 68331, 4 *chips* FPGA Xilinx XC4013 e 4 dispositivos de interligação (FPID I-Cube 320), apresentando ótimos resultados.

Figura 6.2: Exemplo de segmentação de imagem

Basicamente, existem dois tipos diferentes de implementação de RNs em hardware. O primeiro tipo envolve implementações que permitem que as RNs sejam treinadas pelo próprio hardware como em [Eldredge, 1994] e [Hikawa, 1995]. No segundo tipo, as RNs são treinadas fora do hardware e depois implementadas, como nos trabalhos apresentados anteriormente. Ou seja, o treinamento é realizado *off-line*. A implementação realizada durante essa pesquisa não realiza o treinamento da RN no próprio *chip*. É necessário que esse treinamento seja realizado previamente em outro sistema.

6.2.1 Dificuldades na Implementação

Existem algumas dificuldades encontradas no processo de implementação de algoritmos de RNs em hardware. As principais dificuldades encontradas durante as implementações realizadas foram:

- Representação numérica de ponto flutuante
- Implementação de funções não lineares
- Capacidade dos dispositivos de hardware

6.2.2 Representação Numérica de Ponto Flutuante

A maior parte dos algoritmos de RNs trabalha com números de ponto flutuante. Quando esses algoritmos são executados através de algoritmos de software não existem problemas para a representação desse tipo de número, pois, praticamente, todas as linguagens de programação possuem todas as ferramentas necessárias para se representar e realizar operações com números de ponto flutuante. Quando se trata de implementações em hardware, a utilização de números em ponto flutuante é uma tarefa não trivial.

A representação desses números se torna bastante complexa, quando comparada a de números inteiros, e as rotinas que realizam as operações básicas, necessárias para a manipulação desses números, tem custo computacional muito alto (o que implica em maior tempo de execução). As implementações de números de ponto flutuante em hardware também ocupam um espaço físico, dentro do dispositivo de hardware, muito maior do que implementações envolvendo números inteiros. Dessa forma, a solução encontrada para evitar esses problemas consistiu na conversão dos números de ponto flutuante, utilizados nas operações com a RN, para números inteiros, através de técnicas de normalização. Essa conversão, geralmente, implica em uma perda de precisão. O erro obtido quando se converte um número de ponto flutuante para uma representação inteira depende basicamente de quantos bits de precisão são utilizados. Por exemplo, um número decimal qualquer convertido para um inteiro de 16 bits, na maioria das vezes, apresentará um erro menor em relação ao mesmo número convertido para um inteiro de 8 bits. Por outro lado, quanto maior é o número de bits utilizado na conversão, maior também é o custo computacional das operações que envolvem esses números. Deve-se, então, procurar alcançar uma representação numérica que não comprometa os resultados finais e que seja também de menor custo computacional possível.

6.2.3 Funções não lineares

Os neurônios presentes nas RNs do tipo MLP apresentam, na maior parte das implementações, funções de ativação não linear, do tipo sigmóide (Figura 4.3d), a qual é descrita pela equação 6.1

$$F(x) = 1 / (1 - e^{-x}) \quad (6.1)$$

A função sigmóide consiste em uma série exponencial infinita, a qual não é apropriada para a implementação em hardware digital, uma vez que essa implementação é uma tarefa complexa, com custo computacional elevado e altíssima taxa de ocupação do dispositivo no qual a função é implementada. Dessa forma, foram estudadas algumas alternativas de funções que tivessem um comportamento semelhante ao da sigmóide, mas que fossem mais apropriadas para implementação em hardware digital e que possuísem desempenho satisfatório.

Três alternativas foram encontradas:

a) A utilização de *Look Up Tables*

Muitas implementações de RNs, tal como [Eldredge, 1994a], utilizam *Look Up Tables*, que consistem em tabelas de conversão. Essas tabelas relacionam valores de entrada com as respectivas saídas (aplicando-se a função sigmóide). Como a função original trabalha com valores contínuos é necessário que se faça alguma aproximação dos números que se deseja submeter à função, pois apenas alguns pares entrada/saída são representados na tabela. Quanto maior for o número de valores na tabela, maior será sua precisão, uma vez que são necessárias aproximações menores. A implementação da tabela de conversão é, normalmente, realizada com circuitos de memória. Dependendo da velocidade de acesso a essa memória, a utilização de tabelas de conversão pode implicar em atrasos relevantes durante a execução do sistema.

b) Aproximação por funções não-lineares

Segundo [Kwan, 1992], é possível realizar a aproximação da função sigmóide utilizando-se uma função não-linear de complexidade reduzida, a qual pode ser implementada em hardware com facilidade. A função proposta por Kwan é apresentada na Equação 6.2:

$$F(x) = \begin{cases} 1 & \text{para } L \leq x \\ Y & \text{para } -L < x < L \\ -1 & \text{para } -L \geq x \end{cases} \quad (6.2)$$

onde:

$$Y = \begin{cases} x(\beta - \theta x) & \text{para } -L < x < 0 \\ x(\beta + \theta x) & \text{para } 0 \leq x < L \end{cases}$$

e

$$\theta = 1/L^2$$

$$\beta = 2/L$$

Segundo Kwan, a melhor aproximação da função sigmóide é conseguida quando o valor de $L = 2$. A Figura 6.3 apresenta um gráfico que compara a função não-linear proposta por Kwan com a função sigmóide original.

Figura 6.3: Aproximação da função sigmóide.

c) Aproximação por funções lineares

Apesar de apresentar uma precisão menor em relação a solução apresentada anteriormente, a utilização de funções lineares para a aproximação da função sigmóide se mostra interessante devido a sua facilidade de implementação. A função linear, chamada de LPP (linear-por-partes), foi apresentada por Faria et al. em [Wolf, 2000]. A função LPP pode ser descrita da seguinte forma:

$$F(x) = \begin{cases} 1 & \text{para } 1 \leq x \\ Y & \text{para } -1 < x < 1 \\ -1 & \text{para } -1 \geq x \end{cases} \quad (6.3)$$

onde:

$$Y = x/\alpha$$

Através de testes realizados com a função LPP, chegou-se a conclusão de que o valor $\alpha = 5$ é o que mais se aproxima da função sigmóide original, como mostra a Figura 6.4.

Figura 7.4: Funções de Ativação sigmóide e linear-por-partes

Como a implementação da função LPP é de grande facilidade em hardware digital (em relação às outras soluções apresentadas), possui bom desempenho e uma aproximação bastante razoável (para os casos específicos onde ela foi testada), ela foi escolhida para o desenvolvimento dos modelos de RNs apresentados nesse trabalho de pesquisa.

6.2.4 Capacidade dos Dispositivos de Hardware

Pequenas RNs, com poucos neurônios, podem ser implementadas de forma paralela nos dispositivos FPGA disponíveis atualmente no mercado. Mas, infelizmente, os atuais dispositivos não conseguem comportar RNs de grande porte, como a RN proposta nesse trabalho de mestrado. Para implementar RNs complexas (como a da proposta) são necessárias algumas modificações na implementação.

Uma forma encontrada para que esse problema possa ser resolvido é através do desenvolvimento de implementação de apenas alguns poucos neurônios que sejam capazes de realizar todo o processamento da RN através de processamento seqüencial (Figura 6.5).

Figura 6.5: RN com implementação (a) paralela (b) seqüencial

Com a implementação de apenas um ou de alguns poucos neurônios, o algoritmo da RN passa a funcionar de uma forma seqüencial, onde os dados são enviados aos neurônios

em pequenas porções e são obtidos resultados parciais que vão sendo armazenados, para ao final da operação, obter-se o resultado final.

6.2.5 Integrando o processador Nios com o Hardware Dedicado

Em circuitos dedicados que operam de maneira seqüencial, como aquele que foi proposto anteriormente, é fundamental que haja uma perfeita sincronização entre as operações de processamento e os acessos à memória. A utilização de um elemento que garanta essa sincronização e que mantenha uma organização dos endereçamentos de memória realizados é de grande utilidade. O microprocessador Nios é uma alternativa interessante como elemento usado na realização dessa função. O Nios, como já foi descrito anteriormente, é um *softcore* de um microprocessador RISC de 32bits. Ele é ser utilizado para realizar as operações de entrada/saída, fazer os acessos necessários à memória e sincronizar o funcionamento circuito dedicado da RN. A Figura 6.6 apresenta um diagrama que mostra a integração do hardware da RN com o Nios.

Figura 6.6: Integrando o softcore Nios com o circuito da rede neural

O circuito responsável pelo processamento da RN será interconectado com o *softcore* Nios através da interface PBM. Como o Nios ocupa apenas uma pequena parte do dispositivo FPGA, é possível utilizar o restante do mesmo *chip* para a implementação do circuito dedicado correspondente a RN.

6.3 Implementação de uma Etapa de Desenvolvimento Utilizando-se o Conjunto de Dados Íris

Como forma de conhecer melhor as ferramentas e dispositivos disponíveis para a realização dessa pesquisa, uma RN, mais simples em relação à RN proposta, foi projetada e implementada em hardware utilizando-se o kit de desenvolvimento Excalibur.

A RN implementada é utilizada na resolução de um problema de classificação, semelhante a RN utilizada no reconhecimento de gestos por robôs móveis. O conjunto de dados utilizado para os testes da RN é o conjunto Íris, que consiste em um conjunto de dados muito popular e largamente utilizado em testes de algoritmos de aprendizado. O conjunto Íris é formado por informações referentes a 150 flores que são classificadas em 3 diferentes grupos. Cada uma das flores é caracterizada por 4 atributos, que são as informações utilizadas para se realizar a classificação.

O modelo de RN utilizado durante os testes e implementação foi o modelo MLP com uma topologia de 4x4x3; ou seja, 4 neurônios na camada de entrada, 4 neurônios na camada intermediária e três neurônios na camada de saída (Figura 6.7).

Figura 6.7: Rede Neural do Conjunto Íris

A etapa inicial da implementação do modelo de hardware da RNI (rede neural do conjunto Íris) foi o projeto de um neurônio. Cada neurônio recebe 4 valores de entrada e, após o processamento desses valores, apresenta uma saída. Como o papel desempenhado pelos neurônios da camada de entrada é apenas distribuir os valores de entrada para a camada intermediária, sem nenhuma operação aritmética sobre esses valores, a camada de entrada foi substituída por um conjunto de barramentos que distribui diretamente os sinais de entrada para a camada intermediária da RNI.

Cada um dos neurônios foi modelado da seguinte maneira. Cada uma das 4 entradas é multiplicada pelo valor do peso equivalente a aquela entrada. Os resultados das 4 multiplicações são somados com o valor do bias daquele neurônio e então o valor obtido é submetido à função de transferência. O resultado apresentado pela função de transferência equivale ao resultado do neurônio. A Figura 6.8 apresenta o modelo de um neurônio desenvolvido com a ferramenta de projeto de hardware Quartus II.

Figura 6.8: Neurônio da RNI

A RNI completa é formada por 7 neurônios idênticos. A RNI inteira pode ser visualizada na Figura 6.9.

Figura 6.9: Modelo da RNI completa

A implementação completa da RNI foi realizada em um chip APEX20k200E com 500.000 portas lógicas, das quais 50% foram utilizadas com o projeto da RNI. Esses números mostram a limitação de espaço físico dos dispositivos reconfiguráveis. É evidente que já existem dispositivos com maior capacidade, como foi apresentado no Capítulo 2, mas o espaço físico ainda é um dos principais problemas enfrentados quando deseja-se implementar circuitos de hardware mais complexos.

Como já foi descrito anteriormente, os valores de entradas, pesos, bias e resultados, os quais são utilizados no processamento da RNI, foram convertidos para números inteiros.

Foram utilizados 16 bits na representação desses valores, os quais atenderam plenamente os objetivos desejados, com 100% de eficiência na classificação. A Tabela 6.1 apresenta a diferença no tempo de execução do algoritmo da RNI com os dois diferentes tipos de representação numérica.

Implementação	Tempo de Execução
Ponto Flutuante	195 μ s
Inteiro	23 μ s

Tabela 6.1: Diferença de desempenho na conversão de ponto flutuante para inteiros

A Figura 6.10 apresenta as formas de onda obtidas durante a simulação do funcionamento da RNI.

Figura 6.10: Formas de onda obtidas na simulação da implementação em hardware

Os dados de entrada são apresentados à RNI no tempo $t=5\text{ns}$. O resultado do processamento é obtido no tempo $t=37\text{ns}$. Portanto, o tempo necessário para que a RNI realize o processamento dos dados é de aproximadamente 32 ns. Esse tempo de processamento é bastante significativo, uma vez que a execução do mesmo algoritmo em software é centenas de vezes mais lento, como mostra a tabela 6.2.

Implementação	Ambiente utilizado	Tempo de processamento
Software	PC – Pentium III 550MHz	23 μs
Hardware	FPGA APEX 20k200E	32 ns

Tabela 6.2: Comparação de desempenho entre implementações em software e hardware

6.3.1 Otimização de Desempenho

Analisando-se o hardware descrito anteriormente, observa-se que o paralelismo é explorado no processamento simultâneo das entradas da RNI. Obviamente, esse fato representa um avanço em relação às implementações em software, quando executadas por apenas um processador. No entanto, implementações em hardware apresentam outras possibilidades de exploração do paralelismo. Por exemplo, uma função ou procedimento de um algoritmo implementado em software permite que apenas um conjunto de entrada (dados de entrada) seja processado em um determinado instante. Ou seja, o algoritmo passa os dados durante a chamada da função, sendo necessário que essa função realize o processamento e apresente os resultados para que outro conjunto de dados possa ser processado. Nas implementações em hardware, essa limitação pode ser superada através do uso de pipelines [Wolf, 2000].

6.3.2 Utilização da Técnica de Pipelines

Através da utilização de técnicas de pipeline, o circuito pode ser dividido em estágios, e cada um desses estágios pode processar um diferente conjunto de dados,

possibilitando um maior grau de paralelismo durante a execução das operações. Conseqüentemente, um melhor desempenho é obtido do circuito.

A princípio, a RNI foi dividida em dois estágios de processamento representados pelas duas camadas (intermediária e de saída). É necessário que exista uma perfeita sincronização entre os dados processados pelos diferentes estágios do pipeline para que o circuito apresente resultados corretos. Essa tarefa de sincronização é realizada por sinais de clock. Os resultados do desempenho da RNI utilizando-se pipelines pode ser visualizado na tabela 6.3.

Implementação	Tempo de processamento	Tempo de Latência
Sem Pipeline	32 ns	-
Pipeline de 2 estágios	20 ns	42 ns

Tabela 6.3: Desempenho do hardware com pipeline de 2 estágios

O tempo de latência, que é exibido na tabela 6.3, refere-se ao tempo necessário para que o primeiro conjunto de dados seja processado pelo circuito. Esse tempo é, obviamente, maior do que o tempo de processamento padrão porque os dois estágios do pipeline ficam “vazios” até que o primeiro conjunto de dados seja apresentado. Após o processamento desse primeiro conjunto de dados, o pipeline passa a processar sempre 2 conjuntos de dados simultaneamente, melhorando assim o desempenho do circuito. A Figura 6.11 apresenta a implementação da RNI implementada com um pipeline de dois estágios e a Figura 6.12 apresenta as formas de onda obtidas durante a simulação do circuito.

Figura 6.11: RNI com pipeline de 2 estágios

Figura 6.12: Formas de onda obtidas do circuito com pipeline de 2 estágios

Pode ser percebido na figura que, além do circuito apresentar um melhor desempenho (menor tempo de resposta), o circuito apresenta também resultados sem ruídos (períodos de instabilidade), diferentes dos resultados apresentados na Figura 7.11 devido à ausência de ruídos. Portanto, o sinal apresentado na saída do circuito é constantemente correto.

Uma vez que o circuito foi dividido em 2 estágios, a mesma idéia foi utilizada para dividir cada um dos neurônios em 2 estágios de processamento, como pode ser visualizado na Figura 6.13.

Figura 6.13: Neurônios dividido em 2 estágios de processamento

Quando esses neurônios (com pipeline de dois estágios) são utilizados em um circuito de dois estágios, obtém-se um circuito com um total de quatro estágios. Dessa

forma, o circuito pode processar 4 diferentes conjuntos de dados ao mesmo tempo, possibilitando um bom ganho de desempenho em relação a implementação original. A Figura 6.14 mostra as formas de onda obtidas a partir de testes, nos quais o circuito possui um pipeline de 4 estágios.

Figura 6.14: Formas de onda obtidas do circuito com pipeline de 4 estágios

A tabela 6.4 apresenta uma comparação entre o desempenho obtido pelas três versões do circuito da RNI.

Implementação	Tempo de processamento	Tempo de Latência
Sem Pipeline	32 ns	-
Pipeline de 2 estágios	20 ns	42 ns
Pipeline de 4 estágios	10 ns	52 ns

Tabela 6.4: Desempenho das implementações com e sem o uso de pipeline

6.3.3 A Integração com o Processador Nios

A RNI implementada foi conectada ao processador Nios como mostra a figura 6.15.

Figura 6.15: Rede Neural com o processador Nios

Da forma como foi implementado, o circuito permite que a utilização do hardware da RNI seja feita através do processador Nios. Portanto, pode-se utilizar uma linguagem de alto nível (como a linguagem C) para fazer o uso da RNI. Supondo que o software que está sendo executado precise do processamento de um conjunto de dados pela RNI, é necessário que o software apenas realize algumas poucas instruções de I/O para que este obtenha o resultado do processamento. Se o processamento desses dados fosse realizado através da

maneira tradicional, seria necessária a execução de muitas linhas de código pelo processador. São necessárias mais de 100 linhas de código para se implementar um algoritmo equivalente à RNI desenvolvida no hardware.

Um exemplo de código em linguagem C que faria acesso a RNI pode ser visto na Tabela 6.16.

```
...  
out(#port1, #valor1);  
out(#port2, #valor2);  
out(#port3, #valor3);  
out(#port4, #valor4);  
  
result = in(#port5);
```

Tabela 6.16: Algoritmo de acesso à RNI.

Cada uma das instruções de envio de sinais para a RNI é executada em 4 ciclos de clock e as operações de leitura da RNI são executadas em 8 ciclos de clock. Na implementação testada, o processador Nios operava a uma frequência de 40MHz; portanto, cada ciclo de clock possuía 25ns. O processamento de um conjunto de dados pela RNI precisa de 24 ciclos de clock para ser executado (600ns). A implementação do mesmo algoritmo em software precisa de 173µs para processar um conjunto de dados. Assim, a implementação em hardware acessada pelo processador Nios é cerca de 218 vezes mais rápida que a implementação em software. A tabela 6.5 apresenta uma comparação de desempenho das diversas formas de implementação da RNI.

Implementação	Frequência	Tempo de processamento
Software (Nios)	40 MHz	173 μ s
Software (Nios) + Hardware	40 MHz	600 ns
Hardware sem Pipeline	100 MHz	32 ns
Hardware com pipeline de 2 estágios	100 MHz	20 ns
Hardware com pipeline de 4 estágios	100 MHz	10 ns

Tabela 6.5 Comparação de desempenho das diversas implementações da RN Íris

6.4 Implementação da Rede Neural Utilizada no Reconhecimento de Gestos por Robôs Móveis

Devido a grande diferença de topologia entre a rede neural referente ao conjunto de dados Íris (RNI) e a rede neural utilizada para reconhecimento de gestos por robôs móveis (RNG), o projeto do hardware específico para cada uma delas se apresenta de maneira diferente. A RNI possui uma topologia relativamente simples (4x4x3), enquanto que a RNG apresenta uma topologia com dimensões muito maiores (100x100x60). A RNI pode ser implementada de forma paralela, (como já foi descrito anteriormente, ocupando cerca de 50% do dispositivo APEX utilizado. Sabendo-se que cada neurônio da RNG é aproximadamente 25 vezes maior do que um neurônio da RNI, e que a RNG possui cerca de 20 vezes mais neurônios do que a RNI, fica claro que uma implementação paralela da RNG (semelhante a implementação da RNI) se torna uma opção inviável diante da capacidade do dispositivo no qual pretende-se realizar a implementação. A Figura 6.17 apresenta um modelo completo da RNG, onde cada um dos 100 neurônios da camada intermediária recebe 100 valores de entrada e 100 valores de peso, totalizando 10.000 valores de pesos. Os 100 resultados fornecidos por esses neurônios equivalem aos 100 valores de entrada que alimentarão os 60 neurônios da camada de saída, juntamente com mais 6.000 valores de pesos (100 para cada neurônio). Como pode ser observado, além do grande número de neurônios desta RN, as interconexões entre eles (barramentos de 100

linhas) também consistem em um problema bastante difícil de ser resolvido, dado o elevado número de linhas de dados envolvidas. implicam em dificuldades para a implementação em um são ocasionam

Figura 6.17: Diagrama completo da RNG

Como já foi apresentado em 6.2.4, o processamento seqüencial de dados tem se mostrado como uma alternativa interessante diante do problema de falta de capacidade dos dispositivos FPGAs. Obviamente, o processamento paralelo apresenta melhor desempenho quando comparado ao processamento seqüencial. Mas, diante do bom desempenho dos dispositivos FPGAs, as implementações que utilizam processamento seqüencial ainda apresentam desempenho extremamente satisfatório, quando comparado com o processamento realizado por software.

O primeiro passo para a implementação do modelo de hardware da RNG foi a análise numérica dos valores em ponto flutuante envolvidos (entradas e pesos). Após a realização de testes, segundo Faria et al. [Wolf, 2000] a melhor relação entre precisão e desempenho foi encontrada com os valores de entrada representados com 7 bits e os valores dos pesos representados por 16 bits. A escolha desses valores foi realizada de forma que o resultado final fosse o mais próximo possível dos resultados originais e que o menor número de bits fosse utilizado em cada uma das conversões de ponto flutuante para inteiro, pois o número de bits utilizado é diretamente proporcional ao esforço computacional necessário para se realizar as operações aritméticas sobre esses valores.

6.4.1 O modelo Seqüencial do Neurônio na RNG

O neurônio da RNG, diferente dos neurônios da RNI, funciona de forma seqüencial e segue o modelo apresentado em 6.2.4. O funcionamento de um neurônio da RNG pode ser sucintamente descrito como se segue. O neurônio recebe pares formados pelo valor de entrada (X) e pelo valor do peso correspondente (W). Cada um desses pares é processado e o valor de cada resultado parcial é acumulado. Quando todos esses pares forem processados, o resultado parcial obtido pelo acumulador é submetido a função de transferência e, por último, obtêm-se o valor do resultado final desse neurônio (Y), como pode ser visto na Figura 6.18.

Figura 6.18: Arquitetura simplificada de um neurônio da RNG.

Para um perfeito funcionamento, a entrada dos dados e o processamento dos mesmos, deve ser um processo corretamente sincronizado. Esse sincronismo é obtido através do uso de contadores e circuitos de clock. Os contadores, que recebem pulsos de *clock* do sistema, calculam de maneira adequada os endereços utilizados no acesso aos dados nas respectivas memórias. A Figura 6.19 mostra em detalhes todos os componentes de hardware utilizados no projeto do neurônio. Praticamente, todos os elementos necessários para o projeto desse hardware, como: multiplicadores, somadores divisores etc., estão presentes em uma biblioteca disponível no software Quartus II.

Figura 6.19: Esquema detalhado de um neurônio da RNG.

Após a implementação e teste do modelo de neurônio da RNG, foi desenvolvido então o modelo de uma camada completa da RNG, baseada no neurônio descrito anteriormente. Para realizar o teste de uma camada completa da RNG, foram utilizados memórias que continham os valores das entradas e pesos dos 100 neurônios da camada intermediária. Após o processamento, os resultados obtidos foram comparados com os resultados desejados, demonstrando a eficácia da implementação. A sincronização também foi realizada através de contadores conectados aos circuitos de memória. Esses contadores faziam os cálculos de endereçamento para que a RNG fosse alimentada com os valores corretos. Também foi necessária a utilização de um sistema de clock global para a coordenação de todas as operações. A Figura 6.20 apresenta uma camada completa da RNG.

Figura 6.20: Camada completa da RNG

No desenvolvimento de cada camada da RNG foram utilizadas 2 memórias do tipo RAM, contendo os valores dos pesos e entradas. A memória correspondente às entradas possui 100 valores de 7bits (as memórias possuem 8 bits para manter compatibilidade com os barramentos utilizados no restante do sistema), enquanto a memória correspondente aos pesos possui 10.000 valores de 16 bits. Foi também utilizada uma memória do tipo ROM para armazenar os valores dos bias de cada neurônio com 100 valores de bits (as memórias possuem 24 bits para manter compatibilidade com os barramentos internos do circuito). A implementação de uma camada completa da RNG ocupa aproximadamente 12% do chip APEX 20k200E de 500.000 portas lógicas. Após o desenvolvimento de uma camada completa, foi então possível desenvolver a RNG completa.

6.4.2 O Modelo Completo da RNG

Basicamente, a RNG é composta por duas camadas muito parecidas com a camada apresentada anteriormente. A única diferença está no número de neurônios de cada uma das camadas. Como já foi dito anteriormente, a camada de entrada não é representada no modelo de hardware, devido a ausência de operações aritméticas envolvendo os valores utilizados pela mesma. A camada de saída da RNG também possui 1 memória do tipo RAM para os valores de entrada (com 60 valores de 8 bits) e duas memórias do tipo ROM, para os pesos e bias, totalizando 10.060 valores de 16 bits. As duas camadas da RNG são interconectadas de forma que as saídas da camada intermediária são diretamente conectadas à memória que alimenta a camada de saída. Como não é possível realizar as operações de leitura e escrita simultâneas na memória, um novo circuito foi desenvolvido para realizar o sincronismo entre as duas operações. A Figura 6.21 mostra a RNG completa

Figura 6.21: RNG completa.

6.4.3 Reutilização do Hardware Implementado

Cabe ressaltar, que a implementação da RNG em hardware apresentada é capaz de se adequar a qualquer topologia de RN do tipo MLP. Uma vez que todos os neurônios presentes em cada camada da RNG original são representados por apenas um neurônio no modelo de hardware, basta alterar as configurações de sincronismo para que o mesmo neurônio em hardware processe um conjunto de valores correspondente a um diferente número de neurônios equivalentes em software. Portanto, através da configuração dos contadores que fazem o endereçamento de memória e, obviamente, do tamanho das neurônio em hardware processe um conjunto de valores correspondente a um diferente número de neurônios equivalentes em software.

É importante destacar que o neurônio implementado em hardware não será alterado para qualquer que seja a topologia de RN MLP. Apenas será necessária a substituição dos valores de entradas e pesos, obtidos após um novo treinamento, o qual deve ser realizado off-line em software. Portanto, a RNG construída poderá ser utilizada para a solução de novos problemas que envolvam RNs do tipo MLP.

6.4.4 A RNG com o Processador Nios

Apesar do ótimo desempenho da implementação em hardware da RNG, o hardware dedicado apresenta algumas desvantagens como:

- A interface direta com o hardware (entrada/saída de dados).
- A necessidade de um sistema que realize o sincronismo entre a entrada, saída e processamento dos dados.
- O treinamento é off-line, necessitando de uma um outra implementação em software para executa-lo.

Como já foi mencionado anteriormente, o processador *softcore* Nios pode realizar as tarefas citadas de uma forma bastante eficiente. O Nios possui interface paralela de até 32 bits e interface serial padrão RS-232, as quais são de grande utilidade, facilitando as

Na implementação da RNG que envolve o processador Nios com o hardware dedicado da RNG, parte do algoritmo da RN foi codificada em software (linguagem C), o qual foi compilado pelo compilador GNUPro e executado pelo processador Nios. Nessa implementação existe apenas uma camada da RNG implementada em software dedicado, a qual funciona como camada intermediária e como camada de saída da RN. Cabe ao processador Nios enviar os dados de entrada referentes a camada intermediária da RN, obter as respostas da mesma, enviar os dados de entrada da camada de saída da RN e, por último, obter os resultados finais do processamento da RN. Dessa forma, o controle da topologia da RN que está sendo processada fica a cargo do software. O algoritmo de controle da RNG pode ser visto na Figura 6.23.

```

**** camada intermediária da RNG ****
Para X = 0 até X = 99 // X = no. Do neurônio
    envia sinal de inicialização do neurônio X;
    Para X = 0 até X = 99
        prepara dados para ser enviado ao hardware dedicado;
        envia dados para o hardware dedicado;
        envia sinal de sincronismo para o hardware dedicado;
    obtém resultado do neurônio X;

**** camada de saída da RNG ****
Para X = 0 até X = 59 // X = no. Do neurônio
    envia sinal de inicialização do neurônio X;
    Para X = 0 até X = 99
        prepara dados para ser enviado ao hardware dedicado;
        envia dados para o hardware dedicado;
        envia sinal de sincronismo para o hardware dedicado;
    obtém resultado do neurônio X;

```

Figura 6.23: Algoritmo de controle da RNG

A etapa do algoritmo de controle da RNG que mais consome recursos computacionais é a troca de dados entre o processador Nios e o hardware dedicado. Para cada operação da RNG é necessário um conjunto de dados de 3 valores: entradas, pesos e sinais controle. Inicialmente, os 3 valores eram enviados em separado, consumindo um grande esforço computacional do sistema e, conseqüentemente, apresentando um baixo desempenho. A maneira encontrada para amenizar esse problema foi enviar os 3 valores distintos através de uma única operação de envio de dados. Ao receber esse conjunto de dados, o hardware separa e interpreta cada um dos valores, como mostra a Figura 6.2.4.

Figura 6.24: Envio de dados agrupados para o hardware

Devido a falta de um componente que se adequasse às necessidades do projeto, foi necessário o desenvolvimento de um modelo de acumulador diferente dos disponíveis na biblioteca do software Quartus II. O bloco acumulador adequado ao circuito foi desenvolvido utilizando-se a linguagem VHDL.

6.5 Discussão dos Resultados Obtidos

A discussão será abordada em dois aspectos diferentes: desempenho e erros obtidos.

6.5.1 Análise de Desempenho

Os dados da Tabela 6.6 mostram o desempenho de diversas implementações diferentes, inclusive da implementação original que era utilizada pelo Robô da CMU.

	Formas de Implementação	Plataforma	Frequência de clock	Tempo de processamento	Frames processados por segundo
A	Software Completo	Pentium MMX	200 MHz	58.8 ms	17
B	Software da RNG	Pentium III	866 MHz	1.43 ms	700
C	Hardware Dedicado	Excalibur com FPGA APEX 20k200E e memória flash	8 MHz	2.0 ms	500
D	Hardware Dedicado	Excalibur com FPGA APEX 20k200E e memória SDRAM	50 MHz	0.32 ms	3125
E	Hardware Dedicado	FPGA APEX II EP2A70	112MHz	0.14 ms	7142
F	Hardware Dedicado + Nios	Excalibur com FPGA APEX 20k200E	40 MHz	16.1 ms	62

Tabela 6.6: Comparação de desempenho das implementações da RNG.

A implementação A apresenta o desempenho do software original desenvolvido na CMU. Ela apresenta um desempenho relativamente baixo quando comparada às outras

implementações. Isso se deve, basicamente, pelo fato do algoritmo da RNG ser executado juntamente com outros algoritmos, tais como os algoritmos de navegação, controle e etc, simultaneamente no mesmo processador. Ou seja, o processador deve dividir sua capacidade de processamento entre todos os algoritmos executados.

Na implementação B o mesmo algoritmo da RNG foi executado em um processador mais moderno. Como o processador não realizava nenhuma outra tarefa a não ser o processamento da RNG foi obtido um ótimo desempenho no teste. No entanto, em uma situação real, provavelmente, com outros algoritmos sendo executados ao mesmo tempo pelo mesmo processador, o desempenho do algoritmo da RNG não seria tão grande quanto ao obtido. Infelizmente, o trabalho para colocar o software completo do Robô executando no Pentium III é uma tarefa bastante demorada e, deste modo, ela foi deixada como um trabalho futuro para efeito de comparações de desempenho mais completas.

A implementação C foi limitada a 8 MHz devido ao tempo de acesso da memória flash disponível no kit Excalibur, que é de 120 ns. Portanto, é interessante ressaltar que muitas vezes outros periféricos conectados aos FPGAs podem limitar o desempenho do sistema. Nesse caso específico, seria interessante substituir os *chips* de memória por outros mais eficientes.

No caso da implementação D, o desempenho foi limitado pelo próprio FPGA, em 50 MHz, já que o tempo de acesso da memória SRAM é de 10 ns, o que permite operações em até 100 MHz.

Na simulação E, o tempo de acesso à memória foi desconsiderado e, utilizou-se apenas um FPGA APEX II EP2A70, o chip mais recentemente disponível pela Altera.

A implementação F utiliza o hardware dedicado implementado juntamente com o processador Nios, o qual realiza operações de I/O.

A tabela 6.7 apresenta as relações de desempenho entre as diversas implementações.

	Implementações relacionadas	Relação de Desempenho
	C / B	0.71
	D / B	4.46
	E / B	10.2
	F / B	0.088

Tabela 6.7: Relações de desempenho entre as implementações

Analisando a Tabela 6.6 percebe-se que existe uma grande diferença entre as frequências de trabalho dos processadores de propósito geral e dos circuitos de hardware dedicado. Mesmo assim, os circuitos de hardware apresentam um desempenho satisfatório. Cabe lembrar que as implementações em hardware dedicado foram realizadas de forma sequencial. Implementações paralelas, como a da Rede Neural Íris, são muito mais eficientes, mas ocupam muito mais espaço do dispositivo de hardware. Mesmo assim, o hardware dedicado, implementado com um FPGA de ponta, apresentou-se 10.2 vezes mais rápido que o Pentium III 866 MHz (relação E/B). Entretanto, devido a limitação do padrão NTSC, a câmera de vídeo, que capta a imagem dos gestos, fornece apenas 30 frames por segundo (no modo não-entrelaçado, o mais comumente utilizado). Deste modo, qualquer sistema que funcione a uma taxa de processamento maior do que esse ficará ocioso; e assim não há necessidade em paralelizar a implementação em hardware.

Finalmente, a implementação F apresenta a melhor relação entre desempenho e versatilidade, mostrando-se como uma alternativa bastante interessante para a resolução do problema proposto. Além disso, a manipulação de entrada/saída dos dados é toda realizada pelo Nios, o que facilita bastante o projeto, dado o grande volume de dados envolvidos (16.000 valores de 16 bits).

Concluindo, as implementações C, D, E e F satisfazem plenamente os requisitos do sistema de processamento de imagens do Robô, e não há necessidade de implementações mais sofisticadas, dentro do atual contexto (imagens de 10x10 pixels com 30 fps).

6.5.1 Análise de Erro

Como consequência de todas as conversões e aproximações numéricas realizadas, os resultados finais apresentam um erro quando comparados aos resultados originais do software. As Figuras 6.25 e 6.26 apresentam uma comparação entre os resultados originais obtidos em software com os resultados obtidos com as implementações em hardware, respectivamente. Os erros obtidos pelas diferentes implementações foram calculados pelo software BPNN – *Backpropagation Neural Network*, desenvolvido pelo LABIC – ICMC/USP. O erro MSE da implementação em software foi de 0.1308 e o erro da implementação em hardware foi de 0.1413, como mostra a Figura 6.27.

Figura 6.25: Resultados obtidos com execução em software

Figura 6.26: Resultados obtidos com execução em hardware

Figura 6.27: BPNN – Software utilizado na análise de erro das implementações.

Caso fosse necessário obter uma implementação mais precisa, os valores numéricos em ponto flutuante envolvidos nas operações poderiam ser convertidos para uma representação inteira com maior número de bits, ou mesmo uma implementação dos operadores em ponto flutuante.

Capítulo 7

Conclusões e Trabalhos Futuros

7.1 Conclusões

A Computação Reconfigurável e a Robótica são duas áreas aparentemente distintas, porém tem diversas características em comum. Ambas são áreas recentes, onde existe um enorme campo de pesquisa em aberto. Esse trabalho apresenta uma união dessas duas áreas e mostra como implementações em hardware podem ser úteis para se conseguir um bom desempenho em algoritmos de robótica.

Como foi discutido nos capítulos anteriores, o desenvolvimento de hardware, atualmente, conta com ferramentas de grande eficiência, o que permite que sistemas extremamente complexos possam ser desenvolvidos de forma prática, organizada e rápida. A tecnologia FPGA permite que circuitos de hardware possam ser desenvolvidos com um custo baixíssimo, possibilitando que sejam desenvolvidos e testados protótipos com extrema facilidade e rapidez. Isso permite que os projetos desenvolvidos sejam efetivamente executados e não apenas testados.

A área de robótica trabalha com algoritmos complexos e de grande custo computacional, o que acaba, muitas vezes, restringindo suas aplicações. Como já foi discutido anteriormente, o desenvolvimento e a execução eficiente dos algoritmos são a principal limitação dessa área tão interessante. Dessa forma, é sempre importante que novas tentativas de se conseguir maior capacidade de processamento surjam e possibilitem o desempenho necessário para a execução dos algoritmos.

Essa dissertação mostra que a Computação Reconfigurável é uma excelente alternativa para se conseguir um bom desempenho associado a um baixo custo. O Projeto ARMOSH, do qual esse trabalho faz parte, pretende exatamente comparar diferentes implementações de algoritmos de robótica em software e em hardware. Cabe destacar, as características de reconfigurabilidade das implementações que utilizam a tecnologia FPGA. Ou seja, um robô implementado com esta tecnologia pode reconfigurar-se em tempo real (carregar um novo hardware dedicado no dispositivo FPGA) e adequar seus algoritmos para diferentes situações

Deste modo, este trabalho cumpriu esta meta e propiciou um aprendizado em metodologias avançadas para projeto de hardware, utilizando-se a nova e promissora tecnologia *System on Chip - SOC*.

Cabe ressaltar que esta RNG implementada será utilizada em uma nova ferramenta que está sendo desenvolvida no ICMC-USP denominada ARQUITECT-R: [Gonçalves, 2000, 2001, 2001a]. Este sistema utiliza a linguagem CES (*C for Embedded Systems*) [Thrun, 2000], e a partir dela pretende-se gerar automaticamente um hardware dedicado. No caso da linguagem especificar uma RN MLP, este trabalho será utilizado como referência para a implementação.

7.2 Trabalhos Futuros

As RNs são, freqüentemente, utilizadas como parte de algoritmos aplicados à robótica. Como a implementação apresentada nesse trabalho não se limita apenas a uma determinada topologia (conforme descrito no Item 6.4.3), essa implementação pode ser utilizada na solução de diversos problemas que envolvam RNs do tipo MLP. Então, uma proposta interessante para trabalhos futuros seria desenvolver uma ferramenta do tipo CAD que gere automaticamente o hardware de diversas topologias de RNs do tipo MLP. Pretende-se também desenvolver outros tipos de RNs que são freqüentemente utilizados em algoritmos de robótica como os mapas de Kohonen.

Outra sugestão de trabalho futura seria o desenvolvimento em hardware do circuito de pré-processamento das imagens captadas pelo robô.

Referências Bibliográficas

- [Altera, 2000] Altera Corp. *"Excalibur Backgrounder White Paper"*, in URL: <http://www.altera.com>, 2000.
- [Altera, 2000a] Altera Corp. *"Nios Softcore Embedded Processor"*, in URL: <http://www.altera.com>, 2000.
- [Altera, 2000b] Altera Corp. *"ARM-Based Embedded Processor Device Overview"*, in URL: <http://www.altera.com>, 2000.
- [Altera, 2000c] Altera Corp. *"MIPS-Based Embedded Processor Device Overview"*, in URL: <http://www.altera.com>, 2000.
- [Altera, 2001] Altera Corp. *"APEX Programmable Logic Device Family Data Sheets"*, in URL: <http://www.altera.com>, 2001.
- [Aragão, 1997] Aragão, A. C. O. S.; Marques, E., *"A Tecnologia FPGA"*, Relatório Técnico no. 60 – ICMC USP.
- [Aragão, 2000] Aragão, A.; Romero, R.; Marques, E., *"Computação Reconfigurável Aplicada à Robótica, in Computação Reconfigurável"*, Experiências e Perspectivas, pp.184-188, CORE-2000, Marília-SP, Agosto 2000.
- [Arkin, 1998] Arkin, R. C., *"Behavior-Based Robotics"*, MIT Press, 1998.
- [Beckett, 2000] Becket, J., *"As You Like It: New technology lets non-techies customize computers"*, HP Labs, 2000.
- [Braga, 2000] Braga A. P., Ludemir T. B., Carvalho A. C. P. L. F., *"Redes Neurais Artificiais Teoria e Aplicações"*, ed. LTC , 2000.

- [Brooks, 1986] Brooks, R.A., "*A Robust Layered Control System for a Mobile Robot*", IEEE Journal of Robotics and Automation, RA-2, pp. 14-23, 1986.
- [Brown, 1996] Brown S., Rode J. "*Architecture of FPGAs and CPLDs: A Tutorial*", IEEE Design and Test of Computers, vol. 13 No. 2, pp. 42-57, , 1996.
- [Burgard, 1998] Burgard, W., "*Experiences with na Interactive Museum Tour-Guide Robot*", TR no. CMU-CS-98-139, 1998.
- [Coffman, 2000] Coffman, K. "*Real world FPGA design with Verilog*", Prentice Hall, 2000.
- [Cox, 1992] Cox, C. E., Blanz, W. E., "*GANGLION-A Fast Field-Programmable Gate Array Implementation of a Connectionist Classifier*", IEEE Journal of Solid-State Circuits, Vol. 27 No.3, 1992.
- [den Bout, 1999] den Bout, D.V., "*The practical Xilinx designer lab book :version 1.5*", Prentice Hall, 1999.
- [Dehon, 1999] Dehon, A., Wawrzynek, J., "*Reconfigurable Computing: What, Why, and Design Automation Requirements*", Proceedings of the 1999 Design Automation Conference, pp. 610-615, June 1999.
- [Duckett, 2000] Duckett T., Saffloti A., "*Building globally consistent gridmaps from topologies.*" Center for Applied Autonomous Sensor Systems Department of Technology. The University Örebro, SE-70182, Örebro, Sweden.
- [Eldredge, 1994] Eldredge, J. G., Hutchings, B. L., "*RRANN: A Hardware Implementation of the Backpropagation Algorithm Using Reconfigurable FPGAs*", IEEE World Conference on Computational Intelligence, Orlando, EUA, 1994.
- [Eldredge, 1994a] Eldredge, J. G., Hutchings, B. L., "*Density Enhancement of a Neural Network Using FPGAs and Run Time Reconfiguration*", Proceedings of the IEEE Workshop on FPGAsfor Custom Computing Machines, pages 180-188, Los Alamitos, California, 1994.

- [Freeman, 1992] Freeman, J. A., Skapura, David. M. *"Neural Networks: Algorithms, Applications, and Programming Techniques"*, Addison-Wesley Publishing Company, Inc., 1992.
- [Fernandes, 2001] Fernandes, L. C., *"Implementação de Algoritmos para Localização de Robôs Móveis"*, Monografia de Qualificação ICMC/USP, 2001.
- [Glasius, 1995] Glasius R., Komoda A., Gielen S., *"Nural Network Dynamics for Path Planning and Obstacle Avoidance"*, Neural Networks 8(1), pp. 125-133, 1995.
- [Giorgini, 2001] Giorgini, A. L., *"Implementação de um controlador PID Digital para Robótica Baseado em Computação Reconfigurável"*, Dissertação de Mestrado ICMC/USP, 2001.
- [Gonçalves, 2000] Gonçalves, R. A., Wolf, D. F., Coelho, F. A. S., Teixeira, M. A., Ribeiro, A. L., Marques, E., *"ARCHITECT: Um sistema de Computação Reconfigurável"* Workshop de Computação Reconfigurável, pp. 42-45, Marília, 2000.
- [Gonçalves, 2000a] Gonçalves R. A., *"ARCHITECT-R: Uma Ferramenta para o Desenvolvimento de Robôs Móveis Reconfiguráveis"*, Monografia de Qualificação ICMC/USP, 2001.
- [Gonçalves, 2001] Gonçalves R. A., , Wolf, D. F., Rodrigues M. I., Osório, L. F., Moraes, P. A., Genuario, L. B., Ribeiro, A. A. L., Teixeira, M. A., Romero, R. A. F., Marques, E. *"ARCHITECT-R: A System for Reconfigurable Robots Design - An Overview and Initial Results"*, IFIP VLSI-SOC 2001 - International Conference on Very Large Scale Integration, 2001, Montpellier. França.
- [Grossi, 1995] Grossi, C., Posenato, R., Costa, M., Palmisano, D., Pasero E., *"Fast Prototyping for Hardware Neural Networks"*, Proc International Conference on Artificial Neural Networks (ICANN '95), Paris (F), October 9-13 1995, vol 1, pp 411-416.
- [Haykin, 1999] Haykin, S.; *"Neural Networks: A Comprehensive Foundation 2nd. edition"*, Macmillan Publishing Co:/ IEEE Press, 1999.

- [Hikawa, 1995] Hikawa, H., *"Implementation of Simplified Multilayer Neural Networks with On-chip Learning"*, Proceedings of 1995 IEEE International Conference on Neural Networks (IEEE ICNN'95), Vol. 4, pp.1633-1637, 1995.
- [Jones, 1999] Jones, J.L., Flynn, A. M., *"Mobile Robots: Inspiration to Implementation"*, 2nd. Edition, A.K.Peters Ltda, 1999.
- [Kwan, 1992] Kwan, H. K., *"Single Sigmoid-Like Activation Function Suitable for Digital Hardware Implementation"*, Electronics Letters, Vol. 28 No. 15, 1992.
- [Latombe, 1995] Latombe, J.C, *"Robot Algorithms"*, Algorithmic Foundations of Robotics, edited by Ken Goldberb, p. 1-18, A K Peter, Wellesley Massachusetts, 1995.
- [Marquès, 1996] Marquès, R., Zalama, E., Garcia-Bermejo, J. G., Perán, J. R. *"World Modeling and Position Estimation for a Mobile Robot Using Self-Organizing Networks"*, In Proceedings of the 4th. IFAC International Symposium on Intelligent Components and Instruments for Control Applications, 1996.
- [Medeiros, 1998] Medeiros, A.A D., *"A Survey of Control Architectures for Autonomous Mobile Robots"*, Journal of the Brazilian Computer Society, No. 3, Vol. 4, April, 1998.
- [Mezencio, 2001] Mezencio, R., *"Otimização de Algoritmos de navegação para robôs móveis, usando o modelo baseado em células e apoiado em Hardware reconfigurável."*, Monografia de Qualificação ICMC/USP, 2001.
- [Moerland, 1997] Moerland P., Fiesler E., *"Neural Network Adaptations to Hardware Implementations"*, Handbook of Neural Computation E1.2:1-13 Institute of Physics Publishing and Oxford University Publishing, New York, 1997.
- [Moreno, 1999] Moreno J. M., Cabestany J., Cantó E., Faura J., Insenser J. M., *"Dynamically Reconfigurable Strategies for Implementing Artificial Neural Networks Models in Programmable Hardware"*, Proceedings of the 6th Conference Mixed Design of Integrated Circuits and Systems (MIXDES'99),pp. 379-384, Kraków, Poland, June 1999.

- [Moreno, 1999a] Moreno J. M., Cabestany J., Cantó E., Faura J., Insenser J. M., *"The Role of Dynamic Reconfiguration for Implementing Artificial Neural Networks Models in Programmable Hardware"*, Engineering Applications of Bio-Inspired Artificial Neural Networks, J. Mira, J.V. Sánchez Andrés (eds.), pp. 85-94, Springer-Verlag, 1999.
- [Nasa, 2001] Nasa, *"Sojourner web page"*, in URL: www.jpl.nasa.gov/vrml/pathvrml.html, 2001.
- [Omidvar, 1997] Omidvar, O., Smagt, P., *"Neural Systems for Robotics"* Academic Press, 1997.
- [Pérez-Urbe, 1996] Pérez-Urbe A., Sanchez E., *"FPGA Implementation of an Adaptable-Size Neural Network"*, Proc. of VI International Conference on Artificial Neural Networks ICANN96, pp.29-31, 1996.
- [Pomerlau, 1991] Pomerlau, D. A. *"Efficient Training of Artificial Neural Networks for Autonomous Navigation"*, Neural Computation 3, pp. 88-97, 1991.
- [Ribeiro, 2001] Ribeiro, A. A. L. *"Estudo da Reconfigurabilidade das FPGAs"*, Monografia de Qualificação ICMC/USP, 2001.
- [Robocup, 2001] Robocup, *"Robocup web page"* in URL: <http://www.robocup.org>, 2001.
- [Romero, 2000] Romero, R. A., *"Aprendizado em Robôs Móveis Via Software e Hardware – ARMOSH"*, Projeto de Pesquisa do ICMC-USP, Março de 2000.
- [Scatena, 2000] Scatena, J. M. *"Implementação de Mapas Topológicos para Navegação de Robôs Móveis baseada em Computação Reconfigurável"*, Exame de Qualificação ICMC/USP, 2000.
- [Schoenauer, 2000] Schonauer, T., Atasoy, S., Mehrtash, N., Klar, H., *"Simulation of a Digital Neuro-Chip for Spiking Neural Networks"*, IEEE-INNS-ENNS International Joint Conference on Neural Networks, 2000.
- [Teixeira, 2001] Teixeira, M. A., *"Técnicas de Reconfigurabilidade das FPGAs da Família APEX – Altera"*, Monografia de Qualificação ICMC/USP, 2001.

- [Thrun, 2000] Thrun S. *"Towards programming tools for robotics that integrate probabilistic computation and learning"*. Technical report, Carnegie Mellon, 2000.
- [Vieira, 1999] Vieira, R. S., *"Protótipo de um Sistema Monitoramento Remoto Inteligente"*, Dissertação de Mestrado, Universidade Federal de Santa Catarina, 1999.
- [Waldherr, 1999] WALDHERR, S., ROMERO, R. A., THRUN, S., *"Gesture-Based Interface for Human-Robot Interaction"*, Autonomous Robots, vol. 9, pp.151-173, 2000.
- [Wolf, 2001] Wolf D. F., Faria G., Romero R. A. F., Marques E., Teixeira M. A. Ribeiro A. A. L., Fernandes L. C., Scatena J. M., Mezencio R., *"A Pipeline Hardware Implementation for an Artificial Neural Network"*, aceito para publicação no ENIA Encontro Nacional de Inteligência Artificial – XXI Congresso da Sociedade Brasileira de Computação, Fortaleza, 2001.
- [Wolf, 2001a] Wolf, D. F., Romero, R. A. F., Marques, E., *"An Adjustable Size Implementation for an Artificial Neural Network"*, aceito para publicação no IC-AI'2001 International Conference in Artificial Intelligence, Las Vegas, 2001.
- [Wolf, 2001b] Wolf, D. F., Romero, R. A. F., Marques, E., *"Using Embedded Processors in Hardware Models of Artificial Neural Networks"*, aceito para publicação no V SBAI Simpósio Brasileiro de automação Inteligente, Canela, 2001.
- [Yalamanchili, 2001] Yalamanchili, S. *"Introductory VHDL: from simulation to synthesis"*, Prentice Hall, c2001.