Sistema de hardware reconfigurável para navegação visual de veículos autônomos

Maurício Acconcia Dias

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Maurício Acconcia Dias

Sistema de hardware reconfigurável para navegação visual de veículos autônomos

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências - Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Fernando Santos Osório

**USP – São Carlos
Dezembro de 2016**

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

D541s Dias, Maurício Acconcia
 Sistema de hardware reconfigurável para navegação
 visual de veículos autônomos / Maurício Acconcia
 Dias; orientador Fernando Santos Osório. -- São
 Carlos, 2016.
 142 p.

 Tese (Doutorado - Programa de Pós-Graduação em
 Ciências de Computação e Matemática Computacional) --
 Instituto de Ciências Matemáticas e de Computação,
 Universidade de São Paulo, 2016.

 1. Veículos Autônomos. 2. Comitês de Redes Neurais
 Artificiais. 3. Hardware Reconfigurável. 4.
 Processamento de Imagens. 5. FPGA. I. Osório,
 Fernando Santos, orient. II. Título.

Maurício Acconcia Dias

Reconfigurable hardware system
for autonomous vehicles visual navigation

Doctoral dissertation submitted to the Instituto de Ciências Matemáticas e de Computação - ICMC-USP, in partial fulfillment of the requirements for the degree of the Doctorate Program in Computer Science and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Fernando Santos Osório

USP – São Carlos
December 2016

Dedico este trabalho de pesquisa a minha esposa Larissa por estar ao meu lado em literalmente todos os momentos. Obrigado por tudo.

AGRADECIMENTOS

Primeiramente gostaria de agradecer a minha esposa Larissa pelo apoio incondicional nos diversos momentos onde esta tese pareceu ser mais importante do que tudo e, na verdade, foi somente importante como deveria ser. Obrigado por estar ao meu lado, por entender as situações difíceis e por tornar a vida melhor durante este período de 5 anos trazendo tudo de bom que você trouxe para a minha vida. Amo você.

Agradeço também ao prof. e amigo Fernando S. Osório por tudo durante este período. Acredito que nunca sabemos o que realmente significa orientar um aluno até que esta tarefa está em nossas mãos. A responsabilidade é muito grande dado que acabamos por ser um modelo de como o orientado reage as mais diversas situações, e em alguns casos, em todas as situações. Obrigado Osório por ser o melhor modelo de todos e por mostrar que orientar é mais do que ter reuniões e cobrar resultados, por mostrar que orientar é entender as mais diversas situações, entender que o aluno é um ser humano que erra e acerta como qualquer outro, mas que ainda assim temos condições de extrair o melhor de cada um quando vamos pelo caminho certo. Levo suas lições de orientação comigo para toda a vida e acredito que meus alunos vão ter muita sorte por isso.

Agradeço à minha família que esteve sempre junto comigo em todos os momentos, especialmente meu avô que sempre foi um modelo pra mim e hoje não está presente para me ver atingindo mais uma meta. Agradeço também aos meus amigos, a família que escolhemos como dizem, por estarem ao meu lado e me ajudarem muito. Especialmente agradeço ao Daniel e ao Raphael por, a cada dia que passa, me mostrarem o significado real da palavra amizade.

Agradeço, por fim, a Universidade de São Paulo, o Instituto de Ciências Matemáticas e de Computação e o Laboratório de Robótica Móvel por proporcionarem a estrutura e os meios para que este trabalho se concretizasse, agradeço à CAPES pelo financiamento durante as pesquisas que foi primordial para que tudo acontecesse. Agradeço a Fatece, minha nova casa, onde comecei minha jornada como professor universitário, e principalmente ao coordenador, prof. Antonio Marcos e ao diretor acadêmico prof. Marcio Girotti, por darem oportunidade a um professor com experiência nenhuma.

*“O método de postular o que queremos tem muitas vantagens;
as mesmas vantagens do roubo sobre o trabalho honesto.
Vamos deixá-lo para outros e seguir adiante com nossa labuta honesta.”
(Bertrand Russel - Introdução à filosofia matemática)*

*“Se, a princípio, a ideia não é absurda, então não há esperança para ela.”
(Albert Einstein)*

RESUMO

DIAS, M. A.. **Sistema de hardware reconfigurável para navegação visual de veículos autônomos**. 2016. 142 f. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

O número de acidentes veiculares têm aumentado mundialmente e a principal causa associada a estes acidentes é a falha humana. O desenvolvimento de veículos autônomos é uma área que ganhou destaque em vários grupos de pesquisa do mundo, e um dos principais objetivos é proporcionar um meio de evitar estes acidentes. Os sistemas de navegação utilizados nestes veículos precisam ser extremamente confiáveis e robustos o que exige o desenvolvimento de soluções específicas para solucionar o problema. Devido ao baixo custo e a riqueza de informações, um dos sensores mais utilizados para executar navegação autônoma (e nos sistemas de auxílio ao motorista) são as câmeras. Informações sobre o ambiente são extraídas por meio do processamento das imagens obtidas pela câmera, e em seguida são utilizadas pelo sistema de navegação. O objetivo principal desta tese consiste do projeto, implementação, teste e otimização de um comitê de Redes Neurais Artificiais utilizadas em Sistemas de Visão Computacional para Veículos Autônomos (considerando em específico o modelo proposto e desenvolvido no Laboratório de Robótica Móvel (LRM)), em hardware, buscando acelerar seu tempo de execução, para utilização como classificadores de imagens nos veículos autônomos desenvolvidos pelo grupo de pesquisa do LRM. Dentre as contribuições deste trabalho, as principais são: um hardware configurado em um FPGA que executa a propagação do sinal em um comitê de redes neurais artificiais de forma rápida com baixo consumo de energia, comparado a um computador de propósito geral; resultados práticos avaliando precisão, consumo de hardware e temporização da estrutura para a classe de aplicações em questão que utiliza a representação de ponto-fixe; um gerador automático de *look-up tables* utilizadas para substituir o cálculo exato de funções de ativação em redes MLP; um co-projeto de hardware/software que obteve resultados relevantes para implementação do algoritmo de treinamento *Backpropagation* e, considerando todos os resultados, uma estrutura que permite uma grande diversidade de trabalhos futuros de hardware para robótica por implementar um sistema de processamento de imagens em hardware.

Palavras-chave: Navegação Visual Autônoma, Redes Neurais em Hardware, Comitês de Redes Neurais Artificiais, Hardware Reconfigurável, Veículos Autônomos.

ABSTRACT

DIAS, M. A.. **Sistema de hardware reconfigurável para navegação visual de veículos autônomos**. 2016. 142 f. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

The number of vehicular accidents have increased worldwide and the leading associated cause is the human failure. Autonomous vehicles design is gathering attention throughout the world in industry and universities. Several research groups in the world are designing autonomous vehicles or driving assistance systems with the main goal of providing means to avoid these accidents. Autonomous vehicles navigation systems need to be reliable with real-time performance which requires the design of specific solutions to solve the problem. Due to the low cost and high amount of collected information, one of the most used sensors to perform autonomous navigation (and driving assistance systems) are the cameras. Information from the environment is extracted through obtained images and then used by navigation systems. The main goal of this thesis is the design, implementation, testing and optimization of an Artificial Neural Network ensemble used in an autonomous vehicle navigation system (considering the navigation system proposed and designed in Mobile Robotics Lab (LRM)) in hardware, in order to increase its capabilities, to be used as image classifiers for robot visual navigation. The main contributions of this work are: a reconfigurable hardware that performs a fast signal propagation in a neural network ensemble consuming less energy when compared to a general purpose computer, due to the nature of the hardware device; practical results on the tradeoff between precision, hardware consumption and timing for the class of applications in question using the fixed-point representation; a automatic generator of look-up tables widely used in hardware neural networks to replace the exact calculation of activation functions; a hardware/software co-design that achieve significant results for backpropagation training algorithm implementation, and considering all presented results, a structure which allows a considerable number of future works on hardware image processing for robotics applications by implementing a functional image processing hardware system.

Key-words: Autonomous Visual Navigation, Hardware Neural Networks, Neural Network Ensembles, Reconfigurable Hardware, Autonomous Vehicles.

LISTA DE ILUSTRAÇÕES

Figura 1 – Veículos Autônomos LRM	24
Figura 2 – Sistema de Navegação Visual. Adaptado de Shinzato(2010).	27
Figura 3 – SoC Design Gap. Adaptado de Semiconductor Research Corporation.	29
Figura 4 – Dispositivos de Computação. Adaptado de (BOBDA, 2007).	46
Figura 5 – <i>Cyclone II FPGA and Logic Block</i> ((BOBDA, 2007)).	48
Figura 6 – Arquitetura típica de um FPGA ((SKLIAROVA; FERRARI, 2003)).	49
Figura 7 – Fluxo de <i>design</i> em FPGAs. Adaptado de (BOBDA, 2007).	51
Figura 8 – Fluxo atual de um co-projeto.	53
Figura 9 – Estrutura básica de um neurônio artificial do tipo Perceptron ((HAYKIN, 1998)).	57
Figura 10 – Funções de ativação: (a) Função degrau. (b) Função linear com limiares nas extremidades. (c) Função sigmoideal.	58
Figura 11 – Algumas arquiteturas de RNAs. (a) <i>Single-Layer Feedforward totalmente conectada</i> . (b) <i>Multi-Layer FeedForward totalmente conectada</i> . (c) <i>Recorrente sem auto-recorrência</i>	59
Figura 12 – Aprendizado supervisionado de RNAs.	60
Figura 13 – Fluxo de sinal da fase backward para uma parte da rede ((HAYKIN, 1998)).	64
Figura 14 – Fluxo de dados em um comitê de RNAs composto por 3 redes.	65
Figura 15 – Possibilidades para Composição de Comitês.	67
Figura 16 – Representação da arquitetura básica de HNNs.	69
Figura 17 – Estrutura básica de um neurônio em hardware. Adaptado de (SAVICH; MOUSSA; AREIBI, 2012)	70
Figura 18 – Diferença na captura de imagens em um sensor CCD e um sensor CMOS.	82
Figura 19 – Procedimentos iniciais do método.	94
Figura 20 – Funcionamento básico do sistema de classificação de imagens.	95
Figura 21 – Diagrama que representa o sistema de navegação autônoma.	96
Figura 22 – Formas possíveis dos templates: (a) reta, (b) e (c) curva leve, (d) e (e) curva acentuada. Adaptado de (SOUZA <i>et al.</i> , 2011)	97
Figura 23 – Proposta de particionamento hardware/software do sistema para implementação.	98
Figura 24 – Descrição dos componentes do kit DE2-70. (Fonte: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=53&No=226&PartNo=3)	100
Figura 25 – Câmera <i>TRDB – D5M</i> e sua montagem no kit DE2-70.	101
Figura 26 – Diagrama do hardware controlador da câmera.	104

Figura 27 – Simplificação do funcionamento do sistema.	104
Figura 28 – Arquitetura final dos processadores ((DIAS; OSORIO; WOLF, 2012)). . . .	105
Figura 29 – Resultado das redes neurais implementadas.	107
Figura 30 – Diagrama de blocos e conexões do sistema de processamento de imagens gerado pelo Quartus II.	109
Figura 31 – Diagrama de blocos simplificado do gerador de atributos.	111
Figura 32 – Diagrama de blocos e conexões do gerador de atributos gerado usando o Quartus II.	112
Figura 33 – Diagrama de blocos simplificado da rede neural.	113
Figura 34 – Diagrama de blocos e conexões da rede neural artificial gerado usando o Quartus II.	114
Figura 35 – Gerador automático de LUTs	115
Figura 36 – Resultado da primeira fase do algoritmo de geração de LUTs para o sistema de processamento de imagens em hardware.	117
Figura 37 – Número de acessos a cada entrada da tabela escolhida para otimização. . . .	117
Figura 38 – Estudo de precisão para o gerador de atributos.	119
Figura 39 – Estudo de precisão para a rede neural em hardware.	120
Figura 40 – Diagrama de blocos e conexões do comitê de redes neurais gerado pelo Quartus II.	122
Figura 41 – Diagrama de blocos e conexões do classificador neural gerado pelo Quartus II. Módulos do classificador neural em destaque.	124
Figura 42 – Comparações entre resultados de classificação dos classificadores utilizados neste trabalho.	125

LISTA DE TABELAS

Tabela 1 – Comparação de hardwares para implementação de RNAs ((OMONDI; RA-JAPAKSE, 2006)).	69
Tabela 2 – Comparação entre diferentes kits de desenvolvimento de FPGAs.	89
Tabela 3 – Resultados Finais ((DIAS; OSORIO; WOLF, 2012)).	105
Tabela 4 – Comparação entre o sistema de processamento de imagens desenvolvido e outros trabalhos da literatura. (Adaptado de (DIAS; OSÓRIO, 2015)	110
Tabela 5 – Classificação da rede desenvolvida.	113
Tabela 6 – Comparação de tabelas para sistemas robóticos	118
Tabela 7 – Comparação entre a classificação das imagens apresentadas na figura 42. . .	126
Tabela 8 – Comparação entre os sistemas desenvolvidos neste trabalho.	127

SUMÁRIO

1	INTRODUÇÃO	21
1.1	Contexto	22
1.2	Motivação	27
1.2.1	<i>Sistemas embarcados</i>	28
1.2.2	<i>Sistema de navegação visual</i>	30
1.3	Objetivos e Contribuições	31
1.4	Organização dos Capítulos	33
2	SISTEMAS EMBARCADOS	35
2.1	Definições	35
2.1.1	<i>Sistemas de Tempo-Real</i>	37
2.1.2	<i>Sistemas Embarcados Críticos</i>	39
2.1.3	<i>Software</i>	40
2.1.4	<i>Hardware</i>	42
2.1.4.1	<i>Representação numérica</i>	42
2.1.4.2	<i>Computação Reconfigurável</i>	46
2.1.4.3	<i>FPGAs</i>	48
2.1.4.4	<i>Soft-Processors</i>	49
2.1.5	<i>Fluxo de Design em FPGAs</i>	50
2.1.5.1	<i>Co-Projeto de Hardware/Software</i>	52
2.2	Considerações Finais	54
3	REDES NEURAIS ARTIFICIAIS	55
3.1	Definição	55
3.2	Modelos e Arquiteturas	57
3.3	Aprendizado	59
3.3.1	<i>Aprendizado Supervisionado</i>	60
3.3.2	<i>Aprendizado Não-Supervisionado</i>	60
3.4	Perceptron e MLPs	61
3.5	Comitês de RNAs	65
3.5.1	<i>Definição e Considerações</i>	65
3.5.2	<i>Esquemas de Votação</i>	66
3.6	Aceleração e Paralelismo	68

3.7	Redes Neurais em Hardware	68
3.7.1	<i>Métricas</i>	73
3.7.2	<i>Implementação de Redes MLP em Hardware</i>	74
3.8	Considerações Finais	79
4	TECNOLOGIAS E DISPOSITIVOS	81
4.1	Sensores de Imagem	81
4.2	FPGA x GPU	84
4.3	Desenvolvimento com FPGAs	87
4.4	Hardware para veículos autônomos	89
4.5	Considerações finais	92
5	MÉTODO E FERRAMENTAS	93
5.1	Sistema de Navegação em Software	93
5.2	Método	97
5.3	Ferramentas	99
6	RESULTADOS	103
6.1	Captura da Imagem	103
6.2	Processamento da Imagem Capturada	104
6.2.1	<i>Treinamento em Hardware</i>	104
6.2.2	<i>Implementação em Ponto-Fixo</i>	106
6.2.3	<i>Classificador Neural</i>	108
6.2.3.1	<i>Construção do sistema de visão computacional</i>	108
6.2.3.2	<i>Classificador Neural em Hardware</i>	110
6.3	Sistema de Hardware para Classificação	118
6.4	Considerações Finais	127
7	CONCLUSÃO	129
7.1	Publicações	131
7.2	Trabalhos Futuros	131
	REFERÊNCIAS	133

INTRODUÇÃO

O cenário atual apresenta um esforço contínuo e conjunto, por parte de várias empresas e universidades de todo o mundo, para atingir o objetivo de desenvolver veículos autônomos ou sistemas de direção assistida. Parte deste esforço é baseada no crescente número de acidentes de trânsito que ocorrem em todo o mundo onde a principal causa é a falha humana. A possibilidade de desenvolver esta tecnologia também criou um novo mercado para montadoras que pretendem explorá-lo o mais rapidamente possível.

Devido à produção em larga escala de veículos, os sistemas desenvolvidos devem ser extremamente confiáveis e tolerante à falhas, robustos e de baixo custo. Estes requisitos de projeto podem ser atendidos de várias formas por vários tipos de dispositivos. Dentre os sensores mais utilizados as câmeras estão presentes na maioria dos sistemas de navegação autônoma devido à riqueza de informações apresentadas pelas imagens e pelo baixo custo do dispositivo. Após a aquisição da imagem é possível extrair informações utilizando um classificador de forma que o sistema possa interpretar a imagem, detectar os elementos e obstáculos da cena, e executar a navegação.

Considerando este sistema de navegação baseado no uso de imagens, e que deve classificar as imagens obtidas em tempo-real, este trabalho implementa, em hardware reconfigurável, um classificador utilizando redes neurais artificiais. O hardware deste trabalho é baseado em um sistema utilizado para classificação de imagens de navegação autônoma desenvolvido para os veículos e robôs do Laboratório de Robótica Móvel (LRM¹) da Universidade de São Paulo (USP). O laboratório possui ampla experiência nas áreas de desenvolvimento de hardware e robótica, tendo desenvolvido diversos trabalhos na área de veículos autônomos.

A inclusão do hardware proposto em sistemas de navegação tem como objetivo acelerar a classificação de imagens permitindo que o veículo navegue de forma mais confiável em baixa velocidade e também possibilitando o aumento da velocidade máxima de navegação. Além deste objetivo, o hardware proposto neste trabalho também é uma alternativa aos diversos dispositivos

recentemente disponibilizados por empresas de desenvolvimento de hardware no mercado que, apesar de serem plataformas aparentemente robustas e com alto poder de processamento, são soluções fechadas e pouco adaptáveis.

1.1 Contexto

Uma estatística do ano de 2012 mostrava que 3.500 pessoas morrem por dia em acidentes relacionados ao trânsito, e 50 milhões são feridas todo ano nas estradas do mundo todo ((KELLY, 2012)), sendo que este número aumenta com o passar dos anos. As estatísticas locais de cada país, juntamente com os dados apresentados, demonstram que existe uma demanda muito alta por sistemas que auxiliem o motorista na tarefa de dirigir, e até mesmo tornem a tarefa autônoma visando uma diminuição significativa no número e na frequência dos acidentes. Este problema possui escala mundial e as possíveis soluções, que se tornaram temas relevantes e recorrentes de pesquisas em grupos de universidades no mundo, são os sistemas de auxílio ao motorista (*Advance Driver Assistance Systems* - ADAS) e os veículos autônomos. As pesquisas sobre veículos autônomos se mostram mais interessantes pois as soluções encontradas para o problema como um todo podem se tornar, caso analisadas separadamente, sistemas de auxílio ao motorista.

Desde o começo dos estudos com veículos autônomos, onde alguns grupos foram pioneiros no desenvolvimento como o Navlab que é responsável pelo desenvolvimento do projeto ALVINN (*Autonomous Land Vehicle In a Neural Network*) ((POMERLEAU, 1989)) e também a van Mercedes automatizada que contou com a participação do pesquisador Ernst Dickmans (o projeto VaMoRs²) ambos nos anos 80, muitos progressos foram atingidos e muitos sistemas e padrões foram testados. Existem vários projetos de veículos autônomos sendo desenvolvidos pelo mundo, dentre eles é possível destacar:

- *Google Driverless Car*³ - veículo autônomo desenvolvido pela empresa Google que possui grande visibilidade e já foi testado em diversas situações diferentes incluindo o trânsito em ruas de diversas cidades dos Estados Unidos. É possível dizer que o projeto obteve sucesso até o momento, porém os veículos do projeto causaram vários acidentes^{4,5}.
- *e-Instein*⁶ - veículo autônomo desenvolvido pelo laboratório *AutoNOMOS Labs* no *Institut für Informatik* de Berlim. Trata-se de um veículo elétrico dotado de vários sensores como câmeras, RADARs e LIDARs. Todo o software de controle do veículo fica instalado no porta-malas.

¹ <http://www.lrm.icmc.usp.br/>

² <http://www.dyna-vision.de/>

³ <https://www.google.com/selfdrivingcar/paint/>

⁴ <http://www.vision-systems.com/articles/2016/03/google-self-driving-car-collides-with-bus-in-minor-accident.html>

⁵ <http://super.abril.com.br/tecnologia/carros-autonomos-do-google-deram-problema-341-vezes-enquanto-andavam-nas-ruas>

⁶ <http://autonomos.inf.fu-berlin.de/vehicles/e-instein/>

- LUTZ Pods⁷ - veículos autônomos desenvolvidos pelo *Mobile Robotics Group* do Departamento de Engenharia da Universidade de Oxford, sendo que o primeiro veículo foi entregue em maio de 2016. Todo o controle dos veículos foi desenvolvido pelo grupo, e os mesmos serão utilizados no centro da cidade de Oxford para avaliar sua viabilidade tanto do ponto de vista tecnológico quanto social.
- Deeva⁸ - veículo autônomo desenvolvido pela empresa VisLab com sede na Itália. O veículo autônomo é o projeto mais recente desenvolvido pela empresa no segmento, e possui mais de vinte câmeras e 4 sensores laser sem modificações aparentes no exterior do veículo.
- Veículos de Montadoras - várias montadoras de veículos por todo o mundo desenvolvem projetos de veículos autônomos e sistemas de direção assistida como por exemplo a BMW⁹, Mercedes¹⁰ e o consórcio Renault-Nissan¹¹.
- Veículos autônomos do LRM - o Laboratório de Robótica Móvel (LRM-ICMC/USP) desenvolveu, até o momento, três grandes projetos de desenvolvimento de veículos autônomos (figura 1). O projeto CaRINA (Carro Robótico Inteligente para Navegação Autônoma) visa o desenvolvimento de veículos autônomos inteligentes capazes de navegar em ambientes urbanos sem a necessidade de um condutor humano. Dois veículos foram modificados para acomodar o sistema de direção autônoma sendo o primeiro um veículo elétrico, e o segundo um Fiat Palio Weekend. O terceiro veículo foi desenvolvido em parceria com a empresa Scania, sendo o primeiro caminhão autônomo da América Latina. O protótipo utilizado nos testes experimentais é um G360, que pesa 9 ton. Nos três veículos foram feitas modificações elétricas e mecânicas para a implantação do sistema de direção autônoma e sensores de alta precisão. Diversos testes foram realizados com os veículos e, no caso do Palio Weekend e do Caminhão, testes em ambientes urbanos. A página dos projetos¹⁴ possui diversos links para vídeos e reportagens sobre os veículos.

Um veículo autônomo pode ser considerado também uma junção de sistemas de direção assistida e, por este motivo, vários sistemas são desenvolvidos separadamente e depois são agregados em um sistema de direção autônoma completo. Considerando os diferentes sistemas e níveis de automação possíveis para um veículo duas agências americanas (*Society of Automotive Engineers*¹² e *National Highway Traffic Safety Administration*¹³) propuseram duas classificações distintas que poderiam ser unificadas, por exemplo, em uma classificação mais abrangente:

⁷ <https://ts.catapult.org.uk/pods>

⁸ <http://vislab.it/automotive/>

⁹ <http://www.bmwblog.com/2015/12/31/bmw-to-show-autonomous-concept-in-2016/>

¹⁰ <https://www.mercedes-benz.com/en/mercedes-benz/next/automation/>

¹¹ <http://blog.alliance-renault-nissan.com/category/tag/autonomous-cars>

¹² <http://www.sae.org/>

¹³ <http://www.nhtsa.gov/>

¹⁴ <http://www.lrm.icmc.usp.br/web/index.php?n=Port.Pesquisa>

Figura 1 – Veículos Autônomos LRM

- Nível 0 - O motorista controla todas as funcionalidades do veículo todo o tempo.
- Nível 1 - O sistema de direção assistida não controla o veículo, porém pode avisar o motorista.
- Nível 2 - Sistemas como controle de estabilidade e freio automático podem tomar o controle em determinadas situações.
- Nível 3 - Sistemas onde o motorista deve ficar atento e tomar o controle do veículo quando necessário. Exemplos são os sistemas de piloto automático, sistema de estacionamento, sistemas que mantêm o veículo em determinada faixa.
- Nível 4 - Sistemas descritos no nível 3, porém podem ser utilizados em conjunto (dois ou mais sistemas simultaneamente).
- Nível 5 - Sistemas onde o motorista é obrigado a tomar medidas de correção em caso de falha do sistema. Neste caso os sistemas executam aceleração, frenagem e esterçamento.
- Nível 6 - Considerando algumas situações específicas, como por exemplo a direção em rodovias, o motorista pode acionar o sistema de direção autônoma e tirar sua atenção totalmente do veículo.
- Nível 7 - O sistema autônomo pode ser ligado em situações que não são consideradas críticas e o motorista pode desviar totalmente sua atenção. As exceções seriam condições de chuvas torrenciais e neve por exemplo.
- Nível 8 - O sistema é totalmente autônomo sendo que só existem passageiros que definem os locais onde desejam ir.

O desenvolvimento de veículos autônomos enfrenta diferentes tipos de problemas. Devido ao grande número de projetos, vários veículos estão fazendo testes, circulando em vias públicas e, em alguns casos, não se tem considerado todas as medidas de segurança. Caso ocorram acidentes, existem problemas com relação às leis de trânsito. As legislações de trânsito vigentes não

englobam discussões que considerem a responsabilidade em um acidente de trânsito envolvendo veículos sem motoristas. Entretanto algumas mudanças ocorreram como, por exemplo, uma alteração incluída na Convenção de Viena sobre Tráfego Rodoviário (que é um tratado ligado às Nações Unidas que forma a base de regulamentação do tráfego em rodovias) e entrou em vigor em 23 de março de 2016 garante que os sistemas autônomos devem ser controláveis, ou seja, que o motorista pode tomar o controle ou desligá-los a qualquer momento¹⁴. O tratado foi assinado por 72 países.

Os principais desafios técnicos enfrentados pelos veículos autônomos são a fusão de sensores (considerando o processamento dos dados e tomada de decisão) e a capacidade de processamento em tempo-real do sistema. O principal desafio ao se desenvolver um sistema autônomo, como uma solução pronta e comercializável, é tornar o sistema de controle viável financeiramente para implantação em veículos produzidos em larga escala, mantendo a robustez presente no estágio de desenvolvimento. Esta robustez é geralmente atingida utilizando-se uma quantidade considerável de sensores que possuem um custo elevado como pode ser notado em todas as soluções autônomas apresentadas por diversos fabricantes.

A utilização de sensores que possuem baixo custo, de forma inteligente, é uma alternativa interessante neste contexto. A navegação autônoma de veículos possui várias possibilidades de ser realizada utilizando a fusão de sensores como câmeras, LIDARs, RADAR, *encoders* e GPS ((LUETTEL; HIMMELSBACH; WUENSCH, 2012)). Em competições promovidas para veículos autônomos como o DARPA Grand Challenge ((BUEHLER; IAGNEMMA; SINGH, 2007)), DARPA Urban Challenge ((BUEHLER; IAGNEMMA; SINGH, 2009)), *Civilian* e *Military* ELROB¹⁵, que visavam o desenvolvimento de métodos e algoritmos de navegação autônoma para veículos, é possível notar a ampla utilização de vários tipos de sensores. Um ponto comum entre todos os veículos autônomos é a utilização de câmeras, que possuem baixo custo (em comparação com o custo de outros sensores) e proporcionam uma quantidade considerável de informações sobre o ambiente, permitindo aos veículos a navegação autônoma utilizando um sistema de visão computacional.

A navegação visual robótica é uma técnica amplamente desenvolvida e reconhecidamente eficiente como pode ser visto no trabalho de Bonin-Font ((BONIN-FONT; ORTIZ; OLIVER, 2008)). O histórico da navegação visual mostra que existe um ciclo com relação aos sensores e o hardware de controle. Inicialmente o gargalo dos sistemas se concentrava no hardware de processamento de imagens que não comportava a velocidade de aquisição dos sensores. Enquanto o hardware de aquisição das câmeras não apresentava uma evolução significativa, os sistemas embarcados e computadores pessoais que processam as imagens se desenvolveram e atingiram alto poder de processamento. Devido a este fato houve uma necessidade de desenvolvimento de sensores mais adequados ao cenário atual. As câmeras da fabricante Point Grey¹⁶ podem atingir

¹⁴ <https://www.mercedes-benz.com/en/mercedes-benz/next/automation/autonomous-driving-the-legal-framework/>

uma taxa de até 200 fps, criando novamente a necessidade por sistemas de processamento de imagens mais rápidos e robustos.

O processamento de imagens atualmente conta com um grande número de implementações de algoritmos disponíveis para utilização. Um exemplo é a biblioteca de processamento de imagens e visão computacional OpenCV¹⁷ que possui implementações otimizadas para um número considerável de algoritmos utilizados em processamento de imagens. No caso dos veículos autônomos é importante que o sistema possua uma resposta rápida permitindo que o controle do veículo possa ser executado de forma a permitir um processamento em tempo real e a direção segura por parte do veículo. Uma das principais estruturas usadas em sistemas de classificação de imagens são as redes neurais artificiais. A escolha se justifica por suas características que tornam a estrutura uma alternativa viável para a robótica de forma geral ((HORNE; JAMSHIDI; VADIEE, 1990)), e seu rápido tempo de execução após o treinamento tornam a estrutura também indicada para sistemas de navegação robótica ((ZOU *et al.*, 2006)), inclusive atendendo requisitos de tempo-real.

O Laboratório de Robótica Móvel (LRM¹⁸), onde este trabalho foi desenvolvido, possui pesquisadores que atuam em pesquisas de robótica com foco maior em desenvolvimento de hardware e software para veículos autônomos. O Laboratório de Robótica Móvel desenvolve diferentes abordagens de navegação para veículos autônomos, onde para a navegação autônoma baseada no uso de uma câmera monocular, uma das abordagens que obteve bons resultados utiliza imagens classificadas por redes neurais artificiais (SHINZATO; WOLF, 2011). Os classificadores baseados em características de imagens coloridas para veículos autônomos são utilizados desde o início das pesquisas na área obtendo resultados relevantes (JOCHEM; BALUJA, 1993).

A rede neural, que foi usada como base para o hardware desenvolvido neste trabalho, é resultado de técnicas propostas e desenvolvidas junto ao próprio LRM ((SHINZATO; WOLF, 2011)). O diagrama da figura 2 apresenta o funcionamento do sistema proposto por Shinzato e Wolf ((SHINZATO; WOLF, 2011)). Inicialmente um especialista seleciona alguns grandes blocos que são etiquetados com as respectivas classes (aprendizado supervisionado), onde depois são sub-divididos em blocos menores de 10x10 pixels, os quais são classificados como representações de áreas navegáveis e não navegáveis. Em seguida o sistema é treinado e passa a executar a classificação da seguinte maneira: a câmera captura a imagem e então a imagem é dividida em pequenos blocos e os atributos referentes à cada bloco são extraídos. Após obter os atributos desejados um comitê, composto por cinco redes neurais treinadas com os mesmos dados, classifica o bloco e é feita média das classificações (ponderação final). Todos os blocos da imagem são então classificados em zona navegável (segura) e não navegável (não segura) gerando, portanto, o mapa de navegabilidade. O mapa é transferido em seguida para o sistema de controle do carro que irá adequar a navegação ao atual estado percebido do ambiente. Os

¹⁵ <http://www.elrob.org/>

¹⁶ <http://www.ptgrey.com/products/index.asp>

¹⁷ <http://opencv.org/about.html>

atributos de imagem utilizados pelo sistema, considerando os sistema de cores RGB (*Red, Green, Blue*) e HSV (*Hue, Saturation e Value*) são: média B, entropia RGB, entropia V, variância de S, energia de S, entropia HSV, entropia de S, média de R, média de H, energia HSV, média G e entropia HSV. Os resultados das combinações de vários atributos e os resultados que levaram a seleção dos grupos utilizados podem ser encontrados em (SHINZATO, 2010).

Figura 2 – Sistema de Navegação Visual. Adaptado de Shinzato(2010).

Dentro do contexto apresentado acima foi desenvolvido este projeto de pesquisa, focando-se no processamento de imagens totalmente implementado em hardware reconfigurável utilizando redes neurais artificiais, mais particularmente, baseado no modelo proposto por Shinzato (2010). O objetivo principal da implementação do sistema de classificação de imagens em hardware é a diminuição no tempo de execução total permitindo uma navegação mais segura em baixas velocidades (até 60 KM/h), e também a navegação em velocidades mais altas (a partir de 110 KM/h). A seguir serão apresentadas as motivações que mostram a relevância do projeto tanto para o laboratório onde será desenvolvido, quanto para as pesquisas em robótica autônoma de forma geral.

1.2 Motivação

Esta tese engloba conceitos de diversas áreas e serão apresentas a seguir as principais motivações deste trabalho com relação às diferentes áreas de pesquisa.

¹⁸ <http://www.lrm.icmc.usp.br/web/index.php?n=Port.Home>

O desenvolvimento de sistemas de auxílio à direção e veículos autônomos é motivado pelo número considerável de acidentes de trânsito como apresentado anteriormente. A maioria destes acidentes têm sua principal causa associada aos condutores e, caso fosse possível substituir o condutor por um sistema de direção autônomo ou mesmo auxiliar o motorista em casos de perigo, vários destes acidentes poderiam ser evitados. A não existência do condutor também é interessante para veículos militares que poderiam cumprir missões críticas sem colocar em risco a vida de soldados.

Uma característica comum aos sistemas de navegação dos veículos autônomos apresentados na seção anterior são os sistemas baseados em visão. Estes sistemas possuíam limitações com relação à capacidade das câmeras que tornavam o processamento, em alguns casos, mais lento do que o necessário para se atingir os requisitos de tempo-real. As novas opções de câmeras, com alto número de quadros por segundo, possibilitaram o desenvolvimento de sistemas que podem cumprir os requisitos de projeto de tempo real. Porém, estes dispositivos sugerem a utilização de um hardware dedicado, como por exemplo, um sistema embarcado, para que seja possível utilizar toda sua capacidade.

No início do desenvolvimento deste projeto de pesquisa, ano de 2011, não existiam alternativas comerciais de hardware dedicado para o desenvolvimento de veículos autônomos e, durante o período de desenvolvimento deste projeto, várias empresas perceberam a demanda e desenvolveram soluções. Este fato demonstra a importância e serve de motivação para o desenvolvimento de hardware dedicado, o qual possa ser embarcado em veículos, visando à solução de problemas de navegação autônoma.

1.2.1 *Sistemas embarcados*

O desenvolvimento de sistemas embarcados atualmente apresenta algumas mudanças. O mercado de sistemas computacionais está forçando o desenvolvimento de novos sistemas com melhor desempenho e mais funcionalidades em curtos períodos de tempo. Devido às novas funcionalidades, a complexidade dos SoCs (*System-on-Chips*) aumenta juntamente com o número de unidades programáveis contidas nestes chips.

Como previsto por Gordon Moore (([MOORE, 1965](#))), o número máximo de transistores possíveis dentro de um chip aumenta de forma considerável com o passar do tempo e atualmente os SoCs possuem bilhões de transistores. Este número de transistores permite que os dispositivos sejam aplicáveis a várias situações e com um grande número de funcionalidades.

Apesar deste fato, o alto número de transistores no *chip* causa um grande problema que é a impossibilidade dos projetistas de lidar com o crescimento da complexidade lógica destes chips durante o desenvolvimento de sistemas. Este problema é atenuado com as ferramentas de *design* porém não é completamente resolvido. A figura 3 ilustra o fenômeno que é conhecido

¹ <http://www.src.org/#>

Figura 3 – SoC Design Gap. Adaptado de Semiconductor Research Corporation.

como lacuna de projetos de SoCs ou *The SoC design gap*¹ ((HENKEL, 2003)).

O problema apresentado pode ser atenuado de forma mais significativa com novas abordagens e métodos de desenvolvimento utilizando SoCs complexos. A utilização de um determinado SoC em diferentes aplicações diminui o custo de desenvolvimento de vários produtos. O desenvolvimento de SoCs complexos gerou uma demanda crescente por ferramentas e métodos que facilitem o processo de otimização dos sistemas com relação a tempo de desenvolvimento, custo, área e consumo de energia.

O desenvolvimento de SoCs para sistemas embarcados possui dois requisitos importantes e conflitantes: Eficiência (atender requisitos como tempo de execução, área ocupada, consumo de energia) e Programabilidade (permitir modificações em sua estrutura e/ou funcionalidade). Existem vários dispositivos que podem ser utilizados para atender estes requisitos porém cada um possui características que não os atendem de forma conjunta. As principais formas de implementação de sistemas embarcados são: ASICs (*Application Specific Integrated Circuit*), DSPs (*Domain Specific Processors*), FPGAs (*Field Programmable Gate Arrays*) e GPPs (*General Purpose Processors*).

FPGAs combinam de uma forma interessante os dois requisitos importantes apresentados. Esta combinação é possível, pois os FPGAs são um tipo de hardware reconfigurável, que apresenta a eficiência do hardware dedicado e a flexibilidade próxima do software por ser programável. A implementação em FPGAs de estruturas de inteligência computacional é um tema atual na área de desenvolvimento de hardware que atinge bons resultados devido ao fato de que os FPGAs podem não atingir o mesmo consumo de energia, *clock* ou densidade de um chip customizado, porém podem apresentar um *speedup* de várias ordens de grandeza comparado com a execução em software ((HAUCK; DEHON, 2007)). Estas propriedades dos projetos baseados em FPGAs

foi um dos principais fatores motivadores desta tese, onde optou-se pelo desenvolvimento de sistemas embarcados de visão computacional e de inteligência computacional baseados e implementados em FPGAs.

1.2.2 Sistema de navegação visual

O foco deste trabalho está na implementação em hardware de um sistema de navegação visual baseado em redes neurais artificiais. A escolha por colocar estas estruturas em hardware resulta na união das características principais destes sistemas que são (([HAYKIN, 1998](#)), ([MISRA; SAHA, 2010](#))):

- Mapeamento de E/S - O aprendizado supervisionado que ocorre através da apresentação de pares de entrada/saída para a rede e ajuste dos pesos de acordo com o erro da saída da rede em relação à saída desejada. Este método faz com que a rede aprenda através do mapeamento de E/S.
- Adaptabilidade - As RNAs podem ser treinadas novamente para se adaptar a uma mudança nos padrões de entradas.
- Não-Linearidade - Uma rede neural pode ser linear ou não linear de acordo com suas funções de ativação e número de camadas.
- Generalização - Uma rede pode ser treinada e responder corretamente a um conjunto de entradas que não estava nos padrões de treinamento.
- Informação Contextual - Uma rede neural pode aprender informações contidas no contexto dos dados de entrada naturalmente.
- Tolerância à Falhas - Como trata-se de uma estrutura composta por vários neurônios (unidades de processamento individual e massivamente interconectadas com alto grau de paralelismo), a rede implementada em hardware pode ser robusta mesmo quando uma quantidade de neurônios é danificada.
- Velocidade - RNAs implementadas em hardware apresentam *speedup* alto devido ao paralelismo intrínseco da estrutura associado a um custo aceitável.
- Custo - A implementação em hardware de RNAs pode diminuir o número de componentes necessários e também o consumo de energia. Estes dois fatores reduzem o custo de implementação do hardware.

A implementação em hardware do sistema de navegação baseado em visão também possui motivações internas com relação ao desenvolvimento de veículos autônomos. O consumo de energia é um problema constante e, apesar de estar parcialmente resolvido em veículos a

combustão, ainda existe em veículos elétricos e híbridos que estão se tornando uma alternativa ao consumo de derivados de petróleo como combustível. Atualmente os veículos estão com restrições na modificação de seus projetos elétricos para evitar roubo de carga que pode inclusive danificar circuitos já existentes no veículo. Outro problema encontrado pelos sistemas atuais é a taxa de processamento das imagens que, quando processadas por todos os sistemas que compõem uma navegação autônoma, não atingem os requisitos de tempo-real do sistema. Apesar de ainda não haver um consenso sobre os requisitos para navegação autônoma de veículos, é necessário que o sistema opere em uma taxa acima da capacidade do olho humano, 32 fps (também considerado 24 fps em alguns casos), para que possa ser considerado um sistema tempo-real. A questão da velocidade máxima atingível também é importante, pois quanto maior a velocidade maior o número de aplicações em que o sistema poderá ser utilizado.

O sistema funcional implementado em hardware pode ser aplicado a navegação robótica visual em diversos tipos de robôs e, além disso, uma vez que exista uma estrutura de redes neurais artificiais implementada em hardware, é possível utilizá-la para acelerar, como co-processador, sistemas inteligentes baseados em redes neurais. O hardware final pode ser fabricado e, como circuito integrado, ser comercializado de forma individual ou juntamente com sistemas embarcados e plataformas de desenvolvimento de hardware.

Portanto, considerando o que foi apresentado, as principais motivações para este projeto são: (i) a relevância da aplicação alvo escolhida ((SHINZATO; WOLF, 2011)) para o laboratório de pesquisa onde este projeto está inserido; (ii) a necessidade da otimização e aceleração do processamento junto ao sistema de visão para a melhoria do sistema de controle do carro autônomo como um todo; (iii) a utilização de novas tecnologias (como os FPGAs) e de metodologias de projeto atuais (como o co-projeto de hardware/software) para criar hardwares otimizados visando o mesmo tipo de aplicações; (iv) o fato das RNAs poderem ser implementadas em hardware, constituindo-se de uma área de pesquisa ativa e relevante; (v) o desafio de buscar soluções que permitam explorar o processamento paralelo das RNAs e, ao mesmo tempo, buscar soluções simples, compactas, de alta performance e de baixo consumo de energia. As motivações e desafios descritos acima constituem um campo de pesquisas atual e com diversas questões em aberto para novos desenvolvimentos e inovações nas diferentes áreas envolvidas sendo, portanto um tema atual e relevante.

1.3 Objetivos e Contribuições

Inicialmente foram definidos o problema e a hipótese de pesquisa desta tese, que levantam a seguinte questão: é possível implementar em hardware um sistema com uso de FPGAs capaz de reproduzir o modelo neural de classificação de áreas navegáveis/não navegáveis proposto por Shinzato e implementado em software, sem perda significativa da qualidade de resposta, com diminuição significativa do tempo de resposta do sistema e com ampla redução no consumo de

energia?

A partir deste problema foi definido o objetivo principal desta tese como *a implementação e otimização de um comitê de Redes Neurais Artificiais utilizadas em Sistemas de Visão Computacional para Veículos Autônomos (considerando em específico o modelo proposto e desenvolvido por (SHINZATO; WOLF, 2011)), em hardware, buscando propor uma alternativa de hardware específico ao sistema implementado em computadores de propósito geral melhorando o desempenho do sistema como um todo, para utilização em veículos e robôs autônomos* (figura 1).

Para atingir o objetivo principal deste trabalho foram atingidos os objetivos específicos: (i) desenvolvimento do sistema de navegação visual, com precisão numérica reduzida utilizando a representação de ponto-fixado de 12 bits, para testes de tempo de execução, consumo de memória e do efeitos da diminuição da precisão em comparação com o sistema que utilizava ponto-flutuante; (ii) Desenvolvimento de uma arquitetura de hardware que contenha todos os elementos de um sistema de visão computacional utilizado para navegação visual (captura da imagem, processamento, classificação); (iii) Desenvolvimento de uma rede neural em hardware para permitir a implementação do sistema de classificação de imagens para navegação visual proposto por (SHINZATO; WOLF, 2011); (iv) Integração da rede neural desenvolvida em hardware com a arquitetura de hardware; (v) testes, avaliação e ajustes finais da estrutura.

As principais contribuições desta tese de doutorado são:

- Uma estrutura de hardware que implementa um sistema de visão computacional de captura de imagem, processamento e classificação que pode atuar de forma específica como no sistema de navegação visual de veículos autônomos e também de forma geral em trabalhos de robótica. Este sistema apresenta características inovadoras no sentido de apresentar uma estrutura em hardware reconfigurável que pode ser modificada, e também no sentido de que este tipo de tecnologia é pouco utilizada em veículos autônomos.
- O desenvolvimento de um hardware para redes neurais artificiais que utiliza várias características de modelos previamente desenvolvidos e também avança o estado da arte das redes neurais artificiais em hardware por apresentar um método de geração automática de *look-up tables* para implementar funções de ativação em hardware.
- Estudo e propostas de abordagens visando tratar o impacto da utilização de representações numéricas diferentes em redes neurais artificiais utilizadas como classificadores de imagens, que geralmente não é tratado de forma complexa já que a representação utilizada é, na maioria das vezes, o ponto flutuante de precisão dupla.
- Este trabalho contribuiu com diferentes áreas de pesquisa da computação através de artigos publicados em conferências e periódicos apresentando os resultados obtidos.

1.4 Organização dos Capítulos

Esta tese está organizada da seguinte maneira: o capítulo 2 apresenta os sistemas embarcados, suas principais características de hardware e software, sistemas embarcados críticos, os FPGAs como ferramentas de implementação de Sistemas Embarcados e, por fim, o co-projeto de hardware/software; o capítulo 3 apresenta a teoria básica de redes neurais artificiais e as características das implementações de redes neurais em hardware; um capítulo de tecnologias e dispositivos é apresentado em seguida (capítulo 4), pois existem diversas alternativas tecnológicas para o desenvolvimento deste trabalho e é necessário conhecer todas afim de identificar os pontos fortes das tecnologias escolhidas; o capítulo 5 apresenta o método e as ferramentas seguido pelo capítulo 6 onde são apresentados os resultados do trabalho e pelo capítulo 7 que apresenta as conclusões. O documento é finalizado com as referências bibliográficas e as referências dos artigos publicados durante o desenvolvimento desta tese.

SISTEMAS EMBARCADOS

Existem basicamente duas formas de se executar um algoritmo, por hardware ou por software ((HAUCK; DEHON, 2007)). No caso do hardware existem os ASICs (*Application Specific Integrated Circuit*) que são projetados para uma aplicação específica e também os DSPs (*Domain Specific Processors*) que são processadores projetados para um domínio específico sendo mais flexíveis em comparação com os ASICs. Apesar de serem específicos e executarem a tarefa para a qual foram projetados de forma eficiente, estes circuitos não podem ser alterados após sua fabricação. Os microprocessadores, que seriam responsáveis pela execução do algoritmo por software, são mais flexíveis na medida em que o software pode ser alterado e outra tarefa pode ser executada sem mudança no hardware. O problema dos microprocessadores é que para que esta flexibilidade seja atingida o desempenho na execução do algoritmo fica geralmente muito abaixo do desempenho de um ASIC ((COMPTON; HAUCK; COMPTON, 2000)).

Sistemas embarcados estão dominando o mercado atualmente e para obter um bom desempenho devem utilizar da melhor forma possível as implementações de algoritmos em hardware e software separadamente ou de forma conjunta. O sistema deste trabalho possui requisitos que devem ser respeitados visando um bom funcionamento do sistema e principalmente a segurança do usuário e das pessoas presentes no ambiente com o qual o sistema irá interagir. O sistema em questão pode ser classificado como um sistema de tempo real e também um sistema embarcado crítico. Este capítulo apresenta os principais conceitos e definições relacionados a sistemas embarcados e também suas principais características de hardware e software.

2.1 Definições

Apresentar uma definição precisa de sistemas embarcados não é uma tarefa trivial ((VAHID; GIVARGIS, 2001)). Entretanto existem várias definições disponíveis na literatura que quando analisadas de forma conjunta apresentam as características principais que definem estes sistemas. A seguir são apresentadas algumas definições:

- Um sistema embarcado é aproximadamente qualquer sistema de computação que não seja um *desktop* ((VAHID; GIVARGIS, 2001)).
- Sistemas embarcados são sistemas dedicados a realizar uma tarefa específica ((BERGER, 2001)).
- Sistemas embarcados são sistemas orientados à aplicação e desenvolvidos observando-se restrições relacionadas ao tempo relativo de suas ações ((GUPTA, 1995)).
- Um sistema embarcado é uma combinação de hardware e software, e talvez alguma parte mecânica adicional, desenvolvido para desempenhar uma função específica ((VALDER-RAMA *et al.*, 2000)).
- Um sistema cuja principal função não é computacional, porém é controlado por um computador embarcado no mesmo ((WILMSHURST, 2006)).

Analisando as definições apresentadas, pode-se definir um sistema embarcado como: um sistema de computação desenvolvido para um propósito específico segundo algumas restrições sendo as principais: tempo (projeto e execução), custo e área ocupada. Existem algumas características que diferenciam os sistemas embarcados e devem ser levadas em consideração durante seu projeto ((BERGER, 2001)):

- Sistemas embarcados possuem o suporte de uma grande variedade de processadores e de arquiteturas de processadores.
- Sensibilidade com relação ao custo do sistema.
- Sistemas embarcados possuem restrições de tempo de execução. Os sistemas podem possuir restrições de tempo normais ou críticas.
- Ao utilizar um sistema operacional em um sistema embarcado este deve preferencialmente ser um RTOS (*Real Time Operating System*).
- As falhas de software em um sistema embarcado são mais graves em comparação às falhas de um software em um *desktop*.
- Sistemas embarcados possuem restrições de consumo de energia.
- Os sistemas embarcados devem conseguir operar em situações extremas.
- Recursos disponíveis em um sistema embarcado são consideravelmente menores em comparação a outros sistemas, portanto usualmente busca-se explorar estes recursos ao máximo.

Ao desenvolver um sistema embarcado, todas as características previamente apresentadas devem ser levadas em conta devido a vários fatores como: o número de unidades a serem produzidas, que se relaciona diretamente com o custo do projeto; o tempo de vida esperado para o sistema que deve ser atingido com plena funcionalidade do mesmo e também a confiabilidade do sistema que no caso deve ser tolerante a falhas por estar atuando em situações críticas (([VALDERRAMA et al., 2000](#))).

O fato de sistemas embarcados possuírem objetivos de utilização definidos faz com que o projeto deste tipo de sistema seja uma experiência diferente. Ao desenvolver sistemas computacionais para atingir certos objetivos, percebe-se que um sistema (genérico ou específico) não pode ser utilizado para várias aplicações e atingir de forma ótima estes objetivos (([WOLF, 2006](#))). O projetista de sistemas embarcados deve construir uma implementação que cumpra a funcionalidade desejada, porém o maior desafio é construir esta implementação de forma a simultaneamente otimizar várias métricas de projeto como custo, tempo, desempenho, consumo de energia, flexibilidade, *time-to-market*, segurança, manutenibilidade e corretude (([VAHID; GIVARGIS, 2001](#))). O hardware reconfigurável atende à maioria dos requisitos apontados de forma satisfatória, se tornado uma alternativa viável para o desenvolvimento de sistemas embarcados. Os dispositivos reconfiguráveis que possuem atualmente uma ampla utilização por parte dos desenvolvedores de hardware reconfigurável são os FPGAs.

2.1.1 *Sistemas de Tempo-Real*

A definição de sistemas de tempo-real depende da definição de sistema. Um sistema é um mapeamento de um grupo de entradas em um grupo de saídas. O tempo necessário para o mapeamento de uma entrada para uma saída é chamado de tempo de resposta do sistema. Um sistema de tempo-real pode ser definido então como um sistema que deve satisfazer requisitos de tempo explícitos ou terá consequências graves, incluindo uma falha (([LAPLANTE, 2004](#))). Existe também a possibilidade de se definir um sistema de tempo-real por suas características básicas (([WILLIAMS, 2006](#))):

- Sistemas de tempo-real devem computar e prover resultados corretos dentro de um intervalo de tempo especificado.
- Sistemas de tempo-real estão diretamente envolvidos com eventos e estes eventos devem ser tratados através de interrupção.
- As linguagens de programação indicadas para programar sistemas de tempo-real são C, ASM, ADA e Modula-2.
- Em sua maioria sistemas de tempo-real são constituídos por hardware dedicado em sua totalidade ou parcialmente configurando, portanto, sistemas embarcados.

- Considerando os sistemas que possuem uma parte em software, o software possui as seguintes características básicas: uma interface de entrada/saída de dados volátil, capacidade de operar multitarefas em tempo-real, escalonamento em tempo-real e imprevisibilidade (devido ao fato de serem orientados a eventos).

Uma falha em um sistema pode ser definida como o não cumprimento de um *deadline* (entendido aqui como uma meta temporal) estipulado para o sistema, de forma satisfatória. Existem basicamente três tipos de sistemas de tempo real ((LAPLANTE, 2004)):

- *Soft real-time*: é um sistema no qual o desempenho pode ser degradado, porém o sistema não será destruído por falhar em atingir um *deadline*.
- *Firm real-time*: é um sistema no qual não cumprir poucos *deadlines* não levará a uma falha total, porém não cumprir mais que estes poucos *deadlines* pode levar à falhas completas e catastróficas.
- *Hard real-time*: é um sistema onde o simples descumprimento de um *deadline* levará à falhas completas e catastróficas.

O desenvolvimento de sistemas de tempo-real demanda atenção para vários problemas, dentre eles ((LAPLANTE, 2004)):

- Seleção de hardware e software e análise do *tradeoff* entre custo e desempenho. Esta escolha deve levar em conta características de sistemas distribuídos, paralelismo e sincronização.
- Especificação e projeto de sistemas de tempo-real e a correta representação de seu comportamento temporal.
- Maximizar a tolerância a falhas do sistema e a confiabilidade através de um bom projeto.
- O projeto e execução de testes.
- Medir e prever o tempo de resposta tentando reduzi-lo ao máximo.

Os sistemas de tempo-real podem possuir em sua composição hardware e software e sua implementação pode envolver ou não o uso de uma CPU. Independentemente de possuir uma CPU os sistemas de hardware precisam utilizar memórias e as características da memória utilizada são importantes para o desenvolvimento de sistemas de tempo-real. A principal neste caso seria o tempo de acesso que é definido como o intervalo entre uma requisição de dados e quando o dado estará disponível para utilização. Outras características relevantes são o consumo de energia, a densidade e o custo.

Sistemas de tempo-real são em sua maioria sistemas reativos (que reagem a um comando que provém de uma interação com o ambiente) ou sistemas embarcados. O sistema pode ser também um sistema de tempo-real controlado por um sistema embarcado. Nestas situações podemos unir as definições e definir, portanto, os sistemas embarcados críticos.

2.1.2 Sistemas Embarcados Críticos

Sistemas computacionais estão sendo utilizados, de maneira cada vez mais frequente, em circunstâncias que uma falha pode ter sérias consequências. Dentre estes sistemas, os sistemas embarcados são os sistemas escolhidos para este tipo de funções, ou seja, aplicações de tempo-real mais complexas. Estas aplicações demandam geralmente alta capacidade computacional, tolerância à falhas, consumo reduzido de energia e comportamentos precisos com relação ao tempo de execução ((SANTOS; SANTOS; OROZCO, 2009)).

Considerando estas características é possível definir os sistemas embarcados críticos (ou *Safety Critical Systems*) como sistemas embarcados que podem causar danos ou mortes, consideradas consequências inaceitáveis, em caso de falhas ou erros ((KALINSKY, 2005),(RUSHBY et al., 1994)). Os sistemas embarcados críticos não possuem um *safe state*. O serviço deve ser prestado continuamente e com alto nível de adequação aos requisitos onde parar não é uma opção. Uma opção para o desenvolvimento de sistemas críticos em FPGA é sua possibilidade de certificação que é tratada em mais detalhes por Bate e Conmy. ((BATE; CONMY, 2009)).

Os requisitos de um sistema embarcado crítico se confundem com os requisitos de sistemas *hard real-time*, porém nem sempre os sistemas *hard real-time* estão envolvidos em atividades que podem causar danos ou mortes. Algumas características podem ser apontadas para sistemas críticos como ((RUSHBY et al., 1994)):

- O sistema crítico não pode falhar. O comportamento do sistema deve sempre estar de acordo com os requisitos, especificações e expectativas. A continuidade do serviço também deve ser garantida com o correto tratamento das falhas.
- O projeto destes sistemas deve utilizar uma abordagem de engenharia de segurança. Devem ser identificadas potenciais falhas do sistema durante o projeto, suas possíveis causas e como evitá-las ou contorná-las. Depois de identificadas as falhas os danos que cada falha pode causar também deve ser previsto e, geralmente, após esta previsão, as falhas são classificadas e o projeto então adaptado para incluir as possíveis soluções.
- Em caso de sistemas que possuam informações importantes, o sistema deve proteger as informações através de um monitor de acesso. Este monitor de acesso irá controlar, quem, quando e como as informações podem ser acessadas. O sistema também deve se proteger contra alterações externas não desejadas em seu funcionamento.
- Sistemas embarcados críticos em sua grande maioria possuem requisitos *hard real-time*.

- Os requisitos de consumo de energia podem forçar os sistemas críticos, que não possuem autogeração de energia, a incorporar otimizações agressivas para diminuir o consumo de energia. Um exemplo de falha em acessos à memória, verificado por Narayanan e Xie (([NARAYANAN; XIE, 2006](#))), mostra que a queda de tensão de 5V para 4V em uma memória SRAM de 4Mb aumenta a incidência de falhas de acesso aos dados de 57 para 658 respectivamente.

O desenvolvimento de hardware e software para sistemas embarcados, considerando as informações apresentadas neste capítulo, são tarefas que envolvem dispositivos e métodos diferenciados.

2.1.3 Software

Embora o principal objetivo deste trabalho seja o desenvolvimento de uma estrutura de hardware, algumas considerações sobre o software embarcado são relevantes tanto para o entendimento dos sistemas como um todo quanto para a análise da importância de se acelerar partes deste sistema (utilizando hardware) possibilitando maior poder de processamento para o software.

Software embarcado é uma realidade para um crescente número de pessoas nas últimas décadas. Utilizando o exemplo automotivo, veículos fabricados atualmente possuem entre 20 e 70 unidades de controle eletrônicas, mais de 100 milhões de instruções de código totalizando mais de 1 GB de software em um veículo de categoria superior. O custo de desenvolvimento em veículos é determinado inicialmente com base no software embarcado que pode apresentar um alto custo e complexidade resultando, conseqüentemente, em alta possibilidade de falhas. Enquanto as falhas mecânicas decresceram ao longo dos anos, defeitos causados por sistemas eletrônicos aumentaram consideravelmente ([EBERT; JONES, 2009](#)).

O grande problema no desenvolvimento de sistemas de software embarcado é que o acesso aos dados necessários para o desenvolvimento não é trivial. Seriam necessários dados sobre os defeitos ocorridos durante a execução do software, dados industriais para *benchmarking* como por exemplo os defeitos esperados, ou seja, o máximo de informações possíveis de sistemas já implementados. Os problemas com estes dados são a obtenção e também a escalabilidade com relação às novas tecnologias e produtos.

Considerar a interação entre diversos componentes de hardware e software, sendo que algumas são esperadas e previsíveis e outras não, se mostra como um dos desafios no desenvolvimento de software embarcado. Dentre as características do desenvolvimento de software embarcado é possível citar: o desenvolvimento está totalmente ligado ao conhecimento que se possui de hardware pois é o software que irá controlar diversos dispositivos e interagir com os mesmos, fato que não ocorre da mesma maneira no desenvolvimento de software não-embarcado; a complexidade no desenvolvimento de software embarcado está em fazer com que o software

consiga atingir os resultados esperados considerando requisitos de custo (de estrutura e de hardware), tempo de execução (requisitos de tempo real), tolerância a falhas (que podem resultar em mortes dependendo de onde o sistema será embarcado), e corretude (realizar a operação de forma correta), sendo que todos estes requisitos não refletem em um grande número de linhas de código (métrica utilizada em desenvolvimento de software) e sim na qualidade das linhas escritas; o desenvolvimento de software considera a utilização de dados e o próprio algoritmo, no caso de sistemas embarcados a questão é mais ampla utilizando estes dados e algoritmos para o controle e gerenciamento de hardware; o desenvolvimento de software embarcado deve ocorrer em placas e hardware dedicado na maioria das vezes, sendo que o desenvolvimento de software pode ocorrer em computadores de propósito geral; e, por fim, o teste e verificação que em sistemas embarcados necessitam de medições de sinais em hardware, analisadores lógicos, dentre outras ferramentas que dificilmente serão utilizadas no caso de um software comum.

A programação otimizada também é um problema grave. Muitos desenvolvedores de software não necessariamente se preocupam realmente com o consumo de memória, muito menos de energia, de seus sistemas. A situação encontrada em sistemas embarcados envolve a produção em massa destes sistemas o que na maioria das vezes acarreta em uma diminuição de custos. Custos são reduzidos utilizando processadores cuja capacidade de processamento não será equivalente à capacidade de um processador de propósito geral (sendo, na maioria das vezes, utilizados microcontroladores), e também utilizando quantidades reduzidas de memória o que faz com que o recurso deva ser utilizado de forma consciente e otimizada (fato que não é trabalhado em diversas áreas de desenvolvimento de software considerando que a memória disponível em computadores pessoais atualmente não é mais um fator limitante). Um bom código embarcado executa suas funções utilizando pouca memória (ou seja, considerando práticas de utilização consciente) e realizando apenas o processamento necessário (o que demanda processadores mais simples e baratos, e também diminuindo o consumo de energia).

O software embarcado é desenvolvido de maneiras diferentes por cada equipe de desenvolvimento devido, principalmente, às suas características específicas. Existe uma dificuldade eminente de se gerar regras e padrões para desenvolvimento de software embarcado, porém algumas tentativas apresentam resultados interessantes, ainda que não adotadas como padrão em todos os casos. A mais comum é a utilização de *design patterns*, onde são criados padrões a serem utilizados com experiências de soluções de problemas de softwares embarcados. Estes padrões auxiliam no desenvolvimento mas não são a solução dos problemas porque raramente se aplica os padrões de forma direta.

No caso dos veículos autônomos, foco deste projeto, existe um conjunto de documentos elaborados pelo consórcio MISRA (*Motor industry Software Reliability Association*) de empresas que desenvolvem software automotivo. Um dos documentos elaborados contém uma versão da linguagem C, apresentada em forma de regras de utilização, chamado de MISRA C²⁰. O objetivo principal é padronizar o desenvolvimento de código para veículos autônomos que utilizam, neste

caso, a linguagem C facilitando a reusabilidade, manutenibilidade e verificação de código. Existe também a versão para C++ e análise de segurança onde as regras são mais detalhadas na parte de integridade e análise de segurança, juntamente com novas regras de gerenciamento de segurança e ciclo de vida baseado em segurança.

Assim como o desenvolvimento de software, o hardware de sistemas embarcados possui características diferenciadas e empresas estão se concentrando em desenvolver placas e ambientes de desenvolvimento para facilitar o projeto e a implementação de sistemas embarcados. A seção seguinte apresenta o hardware de sistemas embarcados.

2.1.4 Hardware

O hardware de sistemas embarcados apresentam características diferentes estando, na maior parte do tempo, em comunicação com outras partes do sistema implementadas tanto em hardware quanto em software. Em casos de sistemas críticos, além das complicações comuns a todos os sistemas embarcados, o hardware deve respeitar os requisitos de segurança e tempo de execução para o funcionamento do sistema e este fato torna o desenvolvimento mais complexo. Algumas questões podem auxiliar no desenvolvimento do hardware e são apresentadas a seguir.

2.1.4.1 Representação numérica

Os números para o hardware de sistemas de computação sejam eles embarcados ou computadores pessoais, são representados por coleções de bits. O seu significado pode ser interpretado de várias maneiras. Considerando uma palavra binária de $N = 8$ bits:

$$b_7b_6b_5b_4b_3b_2b_1b_0 \quad (2.1)$$

é possível interpretar seus bits como um número inteiro simplesmente convertendo as bases numéricas, no caso o cálculo para a conversão do conjunto de bits para um inteiro sem sinal (*unsigned integer*) na base 10 é descrito pela equação 2.2.

$$x = \sum_{i=0}^{N-1} 2^i * b_i \quad (2.2)$$

Utilizando esta representação, o resultado da operação de adição de dois números resulta em uma palavra de $N + 1$ bits, e a operação de multiplicação em uma palavra de até $2 * N$ bits.

Os sistemas computacionais utilizam uma notação para representar os números, chamada de complemento de 2. Esta notação permite que uma palavra de N bits possa representar inteiros

²⁰ <http://www.misra-c.com/Activities/MISRAC/tabid/160/Default.aspx>

no intervalo entre -2^{N-1} e $2^{N-1} - 1$. A conversão para a base 10 de uma palavra de N bits, agora interpretada segundo a notação do complemento de 2, é descrita pela equação 2.3.

$$x = -b_{N-1} * 2^{N-1} + \sum_{i=0}^{N-2} 2^i * b_i \quad (2.3)$$

O primeiro bit na representação do complemento de 2 é o bit de sinal. Para que seja invertido um número (alterado o seu sinal) nesta representação, por exemplo, é necessário que se faça uma inversão de todos os bits e depois seja adicionado 1 ao resultado.

A representação em complemento de 2 é muito útil e facilita as operações aritméticas em hardware, já que a adição e a subtração nesta notação podem ser feitas pela simples soma bit a bit. Porém para que sejam representados números reais ou fracionários em hardware, é preciso que se escolha qual será a melhor representação para um determinado problema e quais serão os padrões envolvidos na representação escolhida. Os números fracionários, que possuem casas decimais (com um "." ou ","), podem ser representados basicamente de duas formas diferentes: utilizando o ponto fixado em uma posição, definindo portanto quantos bits serão utilizados na parte inteira do número e quantos serão utilizados na parte fracionária do número (ponto-fixado); ou então de forma que o local deste ponto possa ser modificado se adequando à precisão necessária (ponto-flutuante).

A representação numérica em ponto-flutuante foi uma padronização adotada para a representação de números reais. Em 1976 um grupo de trabalho, conhecido como IEEE 754 e liderado pelo professor William Kahan, foi responsável por especificar como representar números em precisão simples e dupla, padronizar os arredondamentos de sistema e padronizar situações como divisão por zero e operações envolvendo infinito. Várias empresas participaram do desenvolvimento, dentre elas Apple, Zilog, DEC, Intel, HP, Motorola e National Semiconductors ((OVERTON, 2001)). O resultado do trabalho deste grupo foi publicado e ficou conhecido como ANSI-IEEE Std 754-1985 ((IEEE, 1985)).

Considerando o padrão IEEE 754, fica estabelecido que um número real representado por 32 bits é dividido em: 1 bit para sinal, 8 bits para o expoente da base e 23 bits para a mantissa. Em 64 bits, o expoente tem 11 bits e a mantissa 52.

$\pm$	$e_7 \ e_6 \ .. \ e_0$	$d_{22} \ d_{21} \ .. \ d_0$
-------	------------------------	------------------------------

Os 8 ou 11 bits do expoente devem conter também a informação sobre o sinal do expoente. Os bits do expoente são armazenados da seguinte maneira:

$$e = s - 2^{N-1} \quad (2.4)$$

Esta operação evita o teste sobre o bit para saber se o número é positivo ou negativo. Para que seja possível recuperar o expoente e , subtrai-se de s (sinal) o valor 2^{N-1} . O zero é

representado por todos os bits em 0 e o infinito por todos os bits da mantissa em 0, todos os bits do expoente em 1 e o bit de sinal correspondente ao infinito positivo ou negativo. Neste mesmo caso, se a mantissa possuir algum bit diferente de 0 tem-se então a representação de NaN (*Not a Number*). A mantissa possui portanto 23 bits, sendo que como a base do sistema é 2, o primeiro bit é sempre 1 devido à normalização dos números. Portanto existe um valor implícito, que é armazenado sempre com o valor 1, que permite a representação de uma mantissa de 24 bits. Por exemplo, o número $(0.008)_{10}$ seria representado por $1.0 * 2^{-3}$ e será armazenado como:

0	01111100	0000 .. 0
---	----------	-----------

O expoente -3 é representado por $(127)_{10} + (-3)_{10} = (01111100)_2$, e a mantissa irá ficar toda em 0 já que o único bit 1 já está representado pelo valor implícito.

As operações de adição e subtração requerem que o expoente seja o mesmo para que as mantissas possam ser adicionadas. Para que os números possam ser somados, a mantissa do número de menor expoente deve ser deslocada para a direita. Este deslocamento deve ser de um número de casas decimais equivalente à diferença entre os dois expoentes. Depois de feito o deslocamento as mantissas são somadas e o resultado é normalizado.

A propriedade do elemento neutro na adição de números reais não se verifica na adição em ponto flutuante, pois cada número representado no sistema admite um conjunto de elementos neutros, ou seja, que não influem no resultado final devido ao número de dígitos considerados pelo arredondamento. A propriedade associativa da adição também não se verifica em aritmética de ponto flutuante. Mais detalhes e exemplos podem ser encontrados em (OVERTON, 2001).

A multiplicação em ponto flutuante é feita da seguinte maneira: somam-se os expoentes e depois multiplica-se as mantissas. O resultado apresentado é verificado para tratamento de *overflow* ou *underflow* e depois arredondado. O *overflow* e o *underflow* ocorrem quando o resultado de alguma operação aritmética necessita de um número de bits maior do que o número disponível na arquitetura. O sinal da operação é modificado ao final do cálculo caso seja necessário.

Devido à organização dos bits da representação de ponto-flutuante, as operações básicas demandam esforços consideravelmente maiores em comparação com as operações entre números inteiros. O efeito da implementação destas operações em hardware, como por exemplo, em FPGAs, é um aumento significativo no tempo de execução devido a necessidade de operações de *pack* e *unpack* para a execução dos cálculos (quando o hardware não possui suporte à operações em ponto-flutuante - FPU), ou na área de *chip* necessária para acomodar os elementos de hardware (quando o hardware apresenta suporte à estas operações, apresentando uma unidade de ponto flutuante). Análises sobre o desempenho e consumo de área do *soft-processor* Nios II, da fabricante de FPGAs Altera, em situações de execução de algoritmos com operações em ponto-flutuante por versões do processador que incluíam ou não suporte as operações de operações de

ponto-flutuante podem ser encontradas nos trabalhos de Dias et. Al. ((DIAS; SALES; OSORIO, 2010a), (DIAS; SALES; OSORIO, 2010b), (DIAS; OSORIO, 2011), (DIAS; OSORIO; WOLF, 2012), (DIAS; OSORIO, 2012)).

Utilizar a aritmética de ponto-flutuante permite que sejam feitos cálculos com alto grau de precisão. Porém, quando a alta precisão dos cálculos não é um requisito de sistema que possui uma alta prioridade, existem alternativas de representação que podem acelerar o tempo de execução do sistema utilizando uma quantidade menor de recursos e consequentemente uma precisão menor. Uma das alternativas ao ponto-flutuante é a representação em ponto-fixo, e no caso extremo, o uso de uma representação com números inteiros que em geral seria a representação com o melhor desempenho.

Já a representação em ponto-fixo é uma alternativa para representar números reais ou fracionários em aplicações onde a precisão não é o único requisito de alta prioridade.

A ideia básica da representação de ponto-fixo é dividir o conjunto de bits em duas partes fixas, um que irá representar a parte inteira do número e a outra que irá representar a parte fracionária do número. Um exemplo para palavras de 8 bits pode ser visto abaixo.

2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}	2^{-4}
8	4	2	1	.	1/2	1/4	1/8	1/16

Dentre as notações existentes para ponto-fixo as mais utilizadas são: a notação $M.N$ e a notação $Q.N$. A notação $M.N$ representa os bits da palavra dividindo-os entre os M bits que representam a parte inteira e os N bits que representam a parte fracionária. A notação $Q.N$, também conhecida como $Q-Format$, representa números em ponto-fixo de forma normalizada, ou seja, entre o intervalo $[-1, 1]$ para notação complemento de 2, e entre o intervalo $[0, 1]$ quando utiliza-se a notação padrão. Neste caso o N representa o número de bits da parte fracionária da palavra desconsiderando-se o bit utilizado na parte inteira.

Um sistema que utiliza palavras de 8 bits em representação $Q.7$, utilizando também o complemento de 2, considera o intervalo entre $\frac{(-2)^{N-1}}{2^{N-1}} = -1$ e $\frac{2^{-(N-1)}}{2^{N-1}} = 1 - 2^{N-1}$. O cálculo para interpretar a palavra utilizando esta notação é apresentado a seguir:

$$x = \frac{-(b_{N-1}2^{N-1}) + b_{N-2}2^{N-2} + (...) + b_12 + b_0}{2^{N-1}} = -(b_{N-1}) + \sum_{i=0}^{N-2} (2^{i-(N-1)b_i}) \epsilon[-1, 1 - 2^{N-1}] \quad (2.5)$$

As operações de soma e subtração são feitas somando-se bit a bit os dois números. Estas operações podem resultar em *overflow* ou *underflow*. Existem várias maneiras de se evitar que esta impossibilidade de representação ocorra, sendo que algumas opções são: (i) caso seja possível prever o intervalo de valores utilizados para a aplicação o número de bits pode ser calculado previamente; (ii) é possível truncar os valores que extrapolarem a capacidade da

representação; ou (iii) multiplica-se os números por um fator constante para diminuí-los de forma que as operações nunca excedam os valores máximos e mínimos da representação.

A multiplicação entre duas palavras em $Q.N$ resulta em uma palavra que deve ser representada em $Q.2N$. O resultado da multiplicação também pode gerar mais que um bit para a parte inteira do número. Este segundo bit é chamado de bit de sinal estendido. A operação pode ser feita multiplicando-se os números sem considerar os bits de sinal. Depois de obtido o resultado devem ser adicionados 0's ou 1's, de acordo com o sinal, e o ponto deve ser colocado para que seja obtido um número em notação $Q.2N$. O resultado final é obtido removendo-se o bit de sinal estendido. Para que a multiplicação sempre seja bem sucedida é necessário representar os números das somas intermediárias no formato em que terão após a multiplicação, ou seja, em $Q.2N$ por uma questão de precisão.

Analisando as duas representações apresentadas, é possível notar que as operações com números que utilizam ponto-fixa são basicamente operações entre números inteiros. Este fato permite que o hardware responsável por implementar as operações ocupe uma área menor, e que o tempo de execução seja reduzido. A precisão também pode ser adaptada de acordo com as necessidades da aplicação, melhorando a maneira como os recursos são utilizados. Todas as vantagens apresentadas fazem do ponto-fixa uma opção interessante para vários tipos de aplicações.

2.1.4.2 Computação Reconfigurável

A computação reconfigurável se encontra em um nível entre o hardware e o software para a execução de um algoritmo como apresentado na figura 4, combinando-as e possibilitando assim eliminar grande parte das desvantagens associadas a cada uma das formas de desenvolvimento.

Figura 4 – Dispositivos de Computação. Adaptado de (BOBDA, 2007).

Este tipo de computação baseia-se em dispositivos lógicos reprogramáveis que podem atingir desempenho elevado e fornecer flexibilidade de programação simultaneamente. Sendo assim, a computação reconfigurável é uma forma de programação que utiliza dispositivos reconfiguráveis, configurados de acordo com descrições de hardware, para desenvolvimento, implementação e testes de circuitos eletrônicos ((BOBDA, 2007)). Desde seu início, conceitualmente no final dos anos 50 ((ESTRIN, 2002), (BOBDA, 2007)), a computação reconfigurável tem sido cada vez mais utilizada devido as suas características básicas ((SKLIAROVA; FERRARI, 2003)):

- **Flexibilidade:** a flexibilidade da computação reconfigurável é consequência do modo pelo qual o hardware é desenvolvido. Geralmente são utilizadas linguagens de descrição de hardware para descrever o hardware desejado. Em seguida o hardware é sintetizado e depois configura um dispositivo reconfigurável, como por exemplo, um FPGA. Desta forma uma modificação no hardware desenvolvido é simplesmente uma mudança na descrição deste hardware, que pode ser novamente sintetizado e reconfigurado. Este fato torna o desenvolvimento mais barato e mais rápido em comparação ao desenvolvimento com ASIC's.
- **Custo:** o custo de desenvolvimento de hardware utilizando computação reconfigurável é menor pois não é necessário construir um *chip* para cada protótipo e os testes são feitos antes mesmo do hardware configurar o FPGA. Sendo assim, o hardware final possui um número muito menor de erros necessitando ser produzido, na maioria das vezes, uma única vez. Além do custo de desenvolvimento, o custo de uma plataforma para desenvolvimento com esta tecnologia é baixo se comparado com outros métodos de desenvolvimento de hardware.
- **Eficiência:** considerando as características anteriores, o hardware desenvolvido utilizando computação reconfigurável pode atingir alto nível de eficiência devido a possibilidade maior de otimizações se o tempo de desenvolvimento disponível for o mesmo tempo de desenvolvimento utilizando outros métodos. A ocupação de área do hardware também é otimizada no sentido que ao longo do desenvolvimento as próprias IDEs possuem programas que calculam a melhor utilização de área possível para um determinado hardware dado um dispositivo reconfigurável.

Existem vários dispositivos que podem ser utilizados para implementar a computação reconfigurável como por exemplo as PLAs (*Programmable Logic Arrays*), PALs (*Programmable Array Logics*), CPLDs (*Complex Programmable Logic Devices*) e FPGAs (*Field Programmable Gate Arrays*). Os principais dispositivos utilizados em desenvolvimentos baseados em computação reconfigurável são os FPGAs devido ao fato de possuírem tantos recursos quanto um ASIC juntamente com a flexibilidade inerente à dispositivos reprogramáveis atingida em grande parte

pela existência das linguagens de descrição de hardware ((SKLIAROVA; FERRARI, 2003)). Os FPGAs estão atualmente presentes em vários dispositivos e também em sistemas embarcados.

2.1.4.3 FPGAs

O desenvolvimento de dispositivos reconfiguráveis resultou na criação da computação reconfigurável. O hardware reconfigurável ideal seria um hardware cujas interconexões entre os milhões de transistores que compõem o chip pudessem ser reconfiguradas para que computassem uma nova função qualquer de uma aplicação, sem restrições de uso de componentes e de conectividade entre estes componentes. Como a implementação de um hardware desta forma ainda não é possível, existem formas simples de implementação que se aproximam de forma considerável do modelo ideal e são utilizadas pelos FPGAs ((YALAMANCHILI, 2001)).

Os FPGAs consistem em um conjunto de células lógicas programáveis (*Logic Blocks*) organizadas em um dispositivo de forma a modelar um vetor de recursos de hardware programável. O dispositivo foi denominado *Field Programmable Gate Array* por conter conjuntos de lógicas programáveis em campo, ou seja, o *chip* é programado por seu usuário final após ser produzido. De acordo com a organização destes elementos no *chip* (posicionamento de blocos lógicos e paradigma de interconexão), que é diferente de acordo com o fabricante, os FPGAs podem ser classificados em quatro grupos: *symmetrical array*, *row-based*, *hierarchy-based* e *sea of gates*.

Figura 5 – Cyclone II FPGA and Logic Block ((BOBDA, 2007)).

Um dos dispositivos que será utilizado no desenvolvimento deste projeto de pesquisa, Altera Cyclone II (figura 5), pertence ao grupo das *hierarchy-based* FPGA's. Neste tipo de organização as células são dispostas de forma hierárquica sendo que os elementos de menor granularidade estão na parte mais baixa da hierarquia e são agrupados para formar os elementos de nível superior. Desta forma cada elemento do nível i consiste em um grupo de elementos do nível $i-1$ ((BOBDA, 2007)). A figura 6 mostra como seria esta organização de forma genérica.

Figura 6 – Arquitetura típica de um FPGA ((SKLIAROVA; FERRARI, 2003)).

Os FPGA's da fabricante Altera possuem dois níveis básicos de hierarquia, o mais baixo que são as células lógicas e o mais alto que são os vetores de blocos lógicos (*Logic Array Blocks* ou LABs). O FPGA Cyclone II possui dezesseis elementos lógicos por LAB e as conexões de sinais que saem e que chegam nos pinos do dispositivo por uma estrutura de roteamento são chamadas de *MultiTrack*. O *MultiTrack* consiste em uma série de rápidos e contínuos canais interligados em linhas e colunas que ocupam todo o espaço do dispositivo. Os blocos lógicos nos FPGAs estão rodeados por blocos de entrada/saída (Input/Output ou I/O). Estes blocos de I/O são programáveis permitindo desta forma que o desenvolvimento interno configure os pinos como entrada, saída ou entrada e saída. Cada pino configura um *Tri-State Buffer* que possui um bloco para entrada, um bloco para a saída e um bloco que ativa a saída onde estes pinos também estão ligados ao barramento *MultiTrack*. Os FPGAs, por serem dispositivos complexos, necessitam de um fluxo de *design* para serem utilizados.

2.1.4.4 Soft-Processors

Um processador, para ser considerado um *soft-processor*, precisa ser inteiramente configurável. O contexto dos *soft-processors* é diferente comparado ao contexto dos processadores normais por estarem configurados em um hardware reconfigurável. Este fato impacta diretamente em seu desenvolvimento e em suas características. A utilização deste tipo de processadores foi uma alternativa ao crescimento da utilização de microprocessadores no desenvolvimento de sistemas embarcados que era possível apenas com a adição de *hard-processors* (os processadores comuns) nas plataformas de desenvolvimento.

Existem vantagens e desvantagens na utilização de *soft-processors* e *hard-processors* no desenvolvimento de sistemas embarcados, porém a utilização de *soft-processors* tem sido mais vantajosa devido a vários motivos. Alguns dos motivos apontados em (YIANNACOURAS; ROSE; STEFFAN, 2005) são listados a seguir:

- Geralmente o número de *hard-processors* disponíveis em uma plataforma de desenvolvimento com FPGAs não irá ser o correto para atender aos requisitos dos projetos.
- A posição das conexões entre os *hard-processors* e o FPGA torna mais difícil a etapa de *Place and Route* durante o desenvolvimento do sistema.
- Utilizando um chip genérico de FPGA, um projetista pode implementar o número de *soft-processors* que for necessário, e possível de acordo com o hardware, para o sistema. As ferramentas de desenvolvimento serão responsáveis por fazer o *Place and Route* dos processadores.
- Estes processadores, por serem implementados em lógica configurável, podem ser modificados com relação a sua complexidade de implementação para atingir exatamente os requisitos do sistema.

Os fabricantes de FPGAs disponibilizam, de forma gratuita ou comercialmente, *soft-processors* prontos para serem configurados em seus dispositivos juntamente com ferramentas de desenvolvimento para que estes processadores sejam programados. Alguns exemplos de *soft-processors* disponibilizados por fabricantes de FPGAs são: Nios II (Altera) e MicroBlaze (Xilinx). Existem também *soft-processors* que não são desenvolvidos por fabricantes de FPGAs, como por exemplo o LEON, onde estes *soft-processors* possuem a vantagem de poderem ser utilizados em qualquer FPGA com pequenas modificações ((TONG; ANDERSON; KHALID, 2006)). Atualmente os *soft-processors*, por apresentarem um baixa performance em comparação com *hard-processors* e com o próprio hardware que geralmente é desenvolvido e integrado ao processador, estão sendo utilizados como *glue logic* para os componentes desenvolvidos em hardware e também para teste de unidades de sistemas em desenvolvimento.

O surgimento da computação reconfigurável e dos FPGAs proporcionou uma mudança no paradigma de desenvolvimento de sistemas embarcados, que passaram a ser desenvolvidos em sua maioria como co-projetos de hardware/software.

2.1.5 Fluxo de Design em FPGAs

O design em dispositivos reconfiguráveis como as FPGAs possui um fluxo que é apresentado na figura 7.

Inicialmente a descrição do hardware é feita através de um esquemático, descritor de máquinas de estados finitos ou de linguagens de descrição de hardware. As linguagens de descrição de hardware (HDLs: *Hardware Description Languages*) são linguagens utilizadas para descrição formal de circuitos eletrônicos e lógica digital. A linguagem permite descrever o modo de operação do sistema, o *design* e a organização. As linguagens de descrição de hardware são similares às linguagens de programação concorrentes devido ao paralelismo intrínseco do hardware possuindo notações explícitas para expressar concorrência ((GHOSH, 1999)).

Figura 7 – Fluxo de *design* em FPGAs. Adaptado de (BOBDA, 2007).

Este tipo de linguagem é utilizado para descrever circuitos e com a utilização de simuladores é possível analisar o hardware antes de implementá-lo fisicamente. Após a descrição do hardware, um programa chamado *Synthesizer* atribui operações lógicas de hardware a partir das declarações da linguagem e produz um *netlist* equivalente, de primitivas genéricas de hardware, para implementar o comportamento descrito. Estes programas costumam ignorar as restrições de tempo e utilizar um *clock* para controlar o sistema.

Existem várias linguagens de descrição de hardware, por exemplo: AHDL, Handel-C, C-to-Verilog, SystemC, PALASM, Verilog, VHDL, dentre outras. A linguagem escolhida para desenvolvimento deste projeto de pesquisa foi o VHDL. Conforme detalhado a seguir, dentre outros fatores, a escolha por VHDL acontece basicamente por ser uma linguagem que possui um amplo histórico de utilização para desenvolvimento de hardware, é estruturada e modular, possui vários códigos de dispositivos disponíveis que podem ser reutilizados e suas bibliotecas estão em sua fase madura de desenvolvimento apresentando poucos erros e muito conteúdo ((PEDRONI, 2004),(PELLERIN; TAYLOR, 1997)).

VHDL (*VHSIC Hardware Description Language*; *VHSIC: Very High Speed Integrated Circuit*) é uma das linguagens de descrição de hardware mais utilizadas no meio acadêmico. Foi inicialmente desenvolvida pelo Departamento de Defesa dos EUA para documentar o comportamento dos ASICs. A sintaxe de VHDL é muito parecida com a sintaxe da linguagem de programação ADA. A utilização de linguagens de descrição de hardware, como VHDL, auxilia no desenvolvimento devido a suas características principais ((YALAMANCHILI, 2001)):

- Interoperabilidade: VHDL possui um conjunto de construções que podem ser aplicadas em múltiplos níveis de abstração. Este fato expande significativamente o escopo de aplicações da linguagem e promove um modelo padronizado e portátil de sistemas eletrônicos.
- Independência de Tecnologia: as descrições de design feitas não estão atreladas a nenhuma metodologia específica de desenvolvimento ou tecnologia alvo. VHDL consegue descrever hardware em nível RTL, conjunto de instruções e *switching transistor*.
- Reutilização de *Design*: várias bibliotecas desenvolvidas em VHDL são compartilhadas entre desenvolvedores.

- Prototipação de Hardware/Software: no caso de HDLs a descrição de hardware pode ser simulada e um software pode ser testado nesta simulação podendo desta forma ser avaliado.

As linguagens de descrição de hardware estão no núcleo das metodologias de desenvolvimento dos sistemas digitais modernos. Este tipo de tecnologia é uma das formas de descrever o hardware que irá configurar as FPGAs.

Após a descrição do hardware, o segundo passo do fluxo de desenvolvimento é a simulação funcional. O objetivo desta simulação é verificar se para uma dada entrada de dados o sistema apresenta a saída desejada. Esta simulação é feita através de softwares e geralmente seu resultado é mostrado em uma interface gráfica. Estes passos descritos anteriormente preparam o sistema para a síntese lógica.

Durante a síntese lógica o sistema é descrito como um conjunto de equações booleanas. As equações booleanas são então mapeadas para funções disponíveis na biblioteca de funções do dispositivo alvo. No caso do FPGA em questão, as funções são mapeadas para tabelas chamadas LUTs (*Look-up Tables*) que são os módulos básicos que irão implementar os operadores booleanos. O resultado da fase de síntese é o chamado *netlist* que descreve os módulos utilizados para implementar as funções e também suas interconexões. Para o *netlist* gerado os operadores precisam ser alocados no dispositivo escolhido e conectados através de um roteamento. Estes passos (*Place and Route*) são geralmente feitos automaticamente por ferramentas disponibilizadas pelos fabricantes dos FPGAs. A configuração gera um código chamado *bitstream* que possui todas as informações deste processo codificadas e prontas para configurarem o dispositivo alvo.

2.1.5.1 Co-Projeto de Hardware/Software

Co-projeto de Hardware/Software (*Hardware/Software Co-Design*) é um paradigma de projetos cujo objetivo consiste no projeto de sistemas que satisfaçam as restrições encontrando um particionamento otimizado e balanceado entre componentes de hardware e software do sistema. Devido à heterogeneidade de componentes, hardware e software devem ser projetados concorrentemente (([VALDERRAMA et al., 2000](#))). Utilizar esta técnica significa unir todos os objetivos do sistema explorando a interação entre os componentes de hardware e software durante o desenvolvimento (([HIDALGO; LANCHARES, 1997](#))).

No contexto de sistemas embarcados, é importante que o projeto seja otimizado para que o produto final consiga atingir as restrições de forma satisfatória. Neste caso a utilização do co-projeto é indicada pois seu método explora as interações entre hardware e software com foco na otimização do sistema. Quando as duas partes do sistema são desenvolvidas de forma conjunta o sistema irá conter apenas as características necessárias para seu funcionamento de forma correta (([MICHELI; GUPTA, 1997](#))). A grande diferença apresentada por este método é a tentativa de otimização de várias partes diferentes do sistema ao mesmo tempo fazendo com que

os projetistas utilizem várias ferramentas exaustivamente durante o desenvolvimento ((WOLF, 2006)).

Figura 8 – Fluxo atual de um co-projeto.

A figura 8 mostra como é um fluxo de um co-projeto de hardware/software. Inicialmente os requisitos de um produto são enviados para uma equipe de desenvolvimento e são analisados. A análise consiste em verificar como podem ser implementadas as funcionalidades contidas nos requisitos e propor as implementações possíveis. A fase de projeto inicia-se com o particionamento do sistema, que é uma das fases mais importantes porque as decisões de projeto nesta fase irão impactar todas as outras fases.

O particionamento hardware/software é a fase do co-projeto onde são decididas quais partes do sistema serão implementadas em hardware e quais partes do sistema serão implementadas em software. Esta decisão geralmente considera métricas como desempenho, consumo de energia, área ocupada em *chip* e custo. Esta fase depende da fase de análise de requisitos, para que os módulos sejam identificados, e impacta diretamente na fase de co-síntese sendo de grande importância para o projeto em sua totalidade. Caso uma decisão seja feita de forma errônea o projeto seguramente irá retornar à fase de particionamento após a verificação ou após a validação. O problema do particionamento remete a um grupo de módulos conectados que deverão ser agrupados de forma a satisfazer os requisitos de sistema e também otimizar o máximo possível de variáveis de projeto.

A fase de particionamento do sistema deve ser feita de forma que a funcionalidade do hardware seja determinada antes da fase de síntese ((KAPLAN; KASTNER, 2000)), mesmo que o diagrama de desenvolvimento apresente ciclos. Anteriormente, o particionamento era feito pelos próprios projetistas utilizando sua experiência em projetos. Com o aumento da complexidade dos sistemas embarcados atuais e do hardware disponível para implementá-los, este processo de particionamento levaria um tempo que não é compatível com o *time-to-market* atual de desenvolvimento. A solução para este problema é utilizar alguma das ferramentas disponíveis no mercado ou então desenvolver o sistema de particionamento. Por se tratar de um problema

NP-Difícil ((ARATO *et al.*, 2003)), as ferramentas de particionamento geralmente utilizam heurísticas (como Algoritmos Genéticos e Redes Neurais Artificiais ((DIAS; LACERDA, 2009)) para resolver o problema já que se trata de um espaço de projeto de tamanho exponencial.

Feito o particionamento, as partes que serão implementadas já estão agrupadas em partes de hardware e partes de software. Após o desenvolvimento estas partes deverão ser integradas para que possam ser feitos os testes e avaliações sobre o sistema.

A fase de co-síntese compreende a síntese do hardware para um dispositivo físico alvo e também a compilação do software gerando os executáveis para a arquitetura alvo juntamente com um escalonador que permita a interação correta dos módulos do sistema. A síntese de interface gera os softwares de sincronização dos módulos de hardware e de software desenvolvidos.

Após a fase de co-síntese ocorre a integração dos módulos juntamente com a co-simulação. No desenvolvimento de sistemas embarcados vários módulos diferentes são desenvolvidos sendo necessários vários simuladores trabalhando simultaneamente para que o sistema seja simulado de forma correta. Isto significa que muitas vezes é preciso um sistema de validação heterogêneo onde ferramentas diferentes são utilizadas no processo de validação, isto é, uma co-simulação. Após a co-simulação o sistema é testado e avaliado até que atenda de forma satisfatória os requisitos e possa ser utilizado.

O co-projeto de hardware/software é um paradigma que utiliza de forma melhor as capacidades do hardware reconfigurável e também guia de forma mais rápida o projeto para um estado onde os requisitos serão satisfeitos.

2.2 Considerações Finais

Este capítulo apresentou os sistemas embarcados e suas características juntamente com a computação reconfigurável e seu principal dispositivo, o FPGA. O fluxo de design em FPGAs foi apresentado e em seguida os *soft-processors*, que foram uma consequência natural da existência da computação reconfigurável. O co-projeto de hardware/software, que é a técnica mais utilizada atualmente para o desenvolvimento de sistemas embarcados e que teve o apoio dos FPGAs como principal ferramenta de prototipação rápida, também foi apresentado. O capítulo seguinte apresenta as Redes Neurais Artificiais, estruturas utilizadas para várias aplicações de classificação como, por exemplo, o sistema de visão computacional para navegação autônoma foco da implementação em hardware deste trabalho.

REDES NEURAIS ARTIFICIAIS

Este capítulo apresenta as Redes Neurais Artificiais (RNAs), a estrutura básica dos neurônios artificiais e as formas como podem ser organizados formando as redes neurais. Considerando que as redes são estruturas que necessitam de um aprendizado, também são apresentadas aqui as formas de aprendizado de RNAs, bem como as possibilidades de exploração de paralelismo e implementação em hardware das RNAs. Em seguida são apresentados o perceptron e as redes MLPs. A última seção do capítulo apresenta o conceito de comitês de RNAs e suas principais características. O assunto tratado neste capítulo apresenta uma visão mais resumida desta extensa área de pesquisas, onde uma abordagem mais ampla pode ser encontrada em (([HAYKIN, 1998](#))).

3.1 Definição

Segundo Haykin (([HAYKIN, 1998](#))) uma rede neural é um processador paralelo distribuído composto por unidades de processamento simples, que possuem uma propensão natural à armazenar conhecimento experimental e torná-lo disponível para utilização. Estas estruturas são inspiradas pelo comportamento do cérebro humano principalmente em dois aspectos: o primeiro é a aquisição de conhecimento que ocorre pelo contato com o ambiente através de um aprendizado, e o segundo é o armazenamento do conhecimento que ocorre nas conexões sinápticas inter-neurônios.

As RNAs podem ser implementadas de várias formas sendo que sua implementação de forma paralela tende a obter melhores resultados. Uma das alternativas de se implementar paralelamente as redes neurais é utilizando estruturas de hardware onde o paralelismo inerente da estrutura pode ser amplamente explorado. As implementações em hardware de RNAs serão tratadas no item 3.6 deste capítulo.

As redes neurais, portanto, podem ser utilizadas como componentes de sistemas maiores para executar vários tipos de tarefas sendo que alguns benefícios de sua utilização são (([HAYKIN,](#)

1998)):

- **Generalização:** esta característica é inerente das redes neurais e seu resultado depende de como a rede foi treinada. A generalização é a capacidade da rede em responder corretamente à estímulos não apresentados previamente, ou seja, durante a fase de treinamento. O treinamento quando executado com padrões de entrada que representam bem o problema tratado, e por um número correto de épocas, pode fazer com que a rede neural treinada apresente uma taxa muito alta de acertos para padrões desconhecidos.
- **Não linearidade:** as redes neurais podem ser lineares ou não lineares dependendo da forma como os neurônios são conectados. Caso o resultado do modo de conexão dos neurônios leve a não-linearidade das respostas, esta propriedade estará por toda a rede de forma distribuída.
- **Mapeamento Entrada-Saída:** a rede neural aprende com os exemplos apresentados à ela na fase de treinamento construindo um mapeamento entrada-saída para o problema em questão.
- **Adaptabilidade:** as redes neurais possuem a capacidade de adaptar seus pesos sinápticos de acordo com mudanças no ambiente. A rede pode ser retreinada facilmente caso as mudanças no ambiente sejam pequenas, ou até ser projetada para se adaptar em tempo-real às mudanças.
- **Tempo de Resposta:** após o treinamento de uma rede neural, a resposta à um estímulo apresentado nas entradas da rede treinada é consideravelmente rápida.
- **Tolerância à Falhas:** a rede neural, por ser uma estrutura paralela e com um número alto de unidades de processamento, que inclusive podem apresentar uma certa redundância, pode ser considerada tolerante à falhas. Esta característica é baseada no fato de que caso um neurônio seja perdido a rede não perde totalmente sua capacidade de generalização. Em alguns casos, pode ser necessário que vários neurônios sejam perdidos até que a rede pare de funcionar completamente. A forma como a rede é treinada, e também a estrutura específica de cada rede neural, podem contribuir de forma positiva ou negativa nesta questão.

As RNAs podem ser aplicadas, entre outras possibilidades, à basicamente quatro tipos de classes de problemas ((BRAGA A.; CARVALHO A. P. L.; LUDERMIR, 2007)): problemas de classificação (reconhecimento de caracteres ou imagens, análise de risco de crédito, detecção de fraude), problemas de categorização (mineração de dados, agrupamentos de dados, análise de sequências de DNA), problemas de previsão (previsão de tempo, previsão financeira, modelagem de sistemas dinâmicos) e problemas de otimização (minimização ou maximização de funções de

custo). Dentre os tipos existentes de redes neurais, existe pelo menos um tipo específico que é indicado para a solução de cada uma das classes de problemas citadas.

Para se entender melhor as RNAs é necessário analisar suas estruturas básicas (neurônios) e como elas podem ser organizadas formando as redes neurais.

3.2 Modelos e Arquiteturas

As RNAs são compostas por unidades básicas de processamento, os neurônios artificiais. Podem existir diferentes modelos de neurônios e de redes neurais, porém nesta tese será dado destaque aos modelos baseados no Perceptron. Inspirado pelo estudo do funcionamento do neurônio biológico, o neurônio artificial possui uma estrutura básica que é apresentada na figura 9.

Figura 9 – Estrutura básica de um neurônio artificial do tipo Perceptron ((HAYKIN, 1998)).

O neurônio da figura 9 que pode ser definido como neurônio k , possui um conjunto de entradas de tamanho n ($X_1..X_n$), um *bias* b e um conjunto de pesos sinápticos também de tamanho n ($w_1..w_n$). A saída v é descrita pela equação 3.1.

$$v_k = \left(\sum_{j=1}^n w_{kj} x_j \right) + b \quad (3.1)$$

A saída do neurônio y , que é o resultado da função de ativação ϕ tendo como entrada a soma ponderada v , é descrita pela equação 3.2.

$$y_k = \phi(v_k) \quad (3.2)$$

O procedimento para tornar a saída do neurônio uma saída não linear é escolher uma função de ativação não linear. A figura 10 apresenta as funções de ativação utilizadas com mais frequência na implementação de RNAs onde a função (c), por exemplo, é uma opção de função

não-linear. Os intervalos utilizados usualmente variam sendo entre $[0, 1]$ ou entre $[-1, 1]$. As funções (a), (b) e (c) são descritas respectivamente pelas equações 3.3, 3.4 e 3.5.

Figura 10 – Funções de ativação: (a) Função degrau. (b) Função linear com limiares nas extremidades. (c) Função sigmoidal.

- A função degrau:

$$\varphi(v) = \begin{cases} 1 & (v \geq 0) \\ 0 & (v < 0) \end{cases} \quad (3.3)$$

- Função linear:

$$\varphi(v) = \begin{cases} 1 & (v \geq 0.5) \\ v & (0.5 > v \geq -0.5) \\ 0 & (v < -0.5) \end{cases} \quad (3.4)$$

- Função sigmoidal com parâmetro de inclinação ajustável a :

$$\varphi(v) = \frac{1}{1 + \exp(-av)} \quad (3.5)$$

Os neurônios são organizados em redes neurais porque, independente das funções de ativação escolhida, sua capacidade de processamento é limitada e uma rede neural é capaz resolver problemas mais complexos. Existem algumas indicações de como dimensionar uma rede neural para um determinado problema, porém a arquitetura da rede que irá solucionar de forma satisfatória um problema é definida, na maioria das vezes, por experimentação.

A classificação das arquiteturas de redes pode ser feita com base em alguns parâmetros:

- Sentido de propagação: quando os sinais se propagam em um só sentido, ou seja, da entrada para a saída da rede a rede é chamada de *feedforward*, e quando os sinais retornam à rede a rede é chamada de recorrente (Figura 11c).
- Número de camadas: a rede pode ter uma camada apenas (*single-layer*, figura 11a) ou então várias camadas (*multi-layer* figura 11b). Quando os neurônios estão todos interligados com todos, a rede é chamada auto-associativa.
- Grau de conectividade: a rede pode ser totalmente conectada, quando cada neurônio da camada anterior é ligado com todos da camada seguinte e pode ser parcialmente conectada, quando um neurônio da camada anterior é conectado com alguns neurônios da camada seguinte. No caso das redes recorrentes as conexões recorrentes podem ser apenas para os outros neurônios e então a rede é chamada de rede recorrente sem auto-recorrência, ou as conexões recorrentes podem ser para o próprio neurônio e então a rede é chamada de rede recorrente com auto-recorrência.

Figura 11 – Algumas arquiteturas de RNAs. (a) *Single-Layer Feedforward totalmente conectada*. (b) *Multi-Layer FeedForward totalmente conectada*. (c) *Recorrente sem auto-recorrência*.

A arquitetura da rede influencia principalmente a capacidade da rede de aprender e de generalizar os dados de entrada de um determinado problema, e também no algoritmo de treinamento que deve ser utilizado. Após o projeto da rede neural, a próxima fase é a fase de treinamento onde a rede aprende sobre os dados de entrada através da modificação dos seus pesos sinápticos. Esta modificação é feita por um algoritmo de treinamento que se baseia em um dos paradigmas de aprendizado. Existem dois paradigmas principais de aprendizado, supervisionado e não-supervisionado, que serão apresentados a seguir.

3.3 Aprendizado

O aprendizado em RNAs ocorre através da mudança dos valores dos pesos sinápticos dos neurônios da rede. Esta mudança pode ocorrer de várias formas possíveis considerando

informações apenas das entradas da rede ou também das saídas obtidas e desejadas. Estas possibilidades deram origem à divisão do aprendizado de RNAs em dois paradigmas: supervisionado e não-supervisionado.

3.3.1 *Aprendizado Supervisionado*

O aprendizado supervisionado ocorre quando existe a presença de um supervisor ou "professor", ou seja, os algoritmos que adotam este paradigma utilizam informações para calcular e corrigir o erro das redes baseadas nas saídas obtidas após o estímulo da rede pelos padrões de entrada. Este tipo de aprendizado se aplica em situações onde se deseja obter um mapeamento entre padrões de entrada e saída ((BRAGA A.; CARVALHO A. P. L.; LUDERMIR, 2007)). A figura 12 ilustra a situação onde o supervisor estimula as entradas e calcula o erro entre as saídas obtidas e desejadas. Este erro é utilizado então para corrigir os pesos sinápticos da rede. Para que seja possível o cálculo do erro é necessário que sejam apresentados às redes pares entrada-saída onde, para cada entrada a rede, será apresentada a saída desejada permitindo assim o cálculo do erro.

Figura 12 – Aprendizado supervisionado de RNAs.

Os exemplos mais comuns de aprendizado supervisionado são a regra delta e o algoritmo *back-propagation* que serão abordados na seção seguinte. Existe um caso particular de aprendizado supervisionado que é conhecido como aprendizado por reforço. Neste caso o supervisor é chamado de crítico e tem o papel de reforçar as escolhas corretas da rede neural maximizando seu efeito, não necessariamente através da estimação e correção direta do erro.

3.3.2 *Aprendizado Não-Supervisionado*

Ao contrário do aprendizado supervisionado, o aprendizado não-supervisionado apresenta apenas as entradas para a rede neural e não os pares entrada-saída. Não existe também neste caso o supervisor. Para que o aprendizado ocorra, os dados de entrada são apresentados à rede continuamente e a existência de regularidades nos dados faz com que seja possível o aprendizado ((BRAGA A.; CARVALHO A. P. L.; LUDERMIR, 2007)).

Dois exemplos de aprendizado não-supervisionado são:

- Aprendizado hebbiano: a variação do peso depende apenas de uma taxa de aprendizado η , do valor da entrada no tempo $t = x_i(t)$ e do valor da saída no tempo $t = y_i(t)$. Sendo assim a equação de ajuste dos pesos é:

$$\Delta w_{ij} = \eta y_i(t) x_i(t) \quad (3.6)$$

- Aprendizado por competição: este aprendizado consiste numa competição entre as saídas que, dado um padrão de entrada, vão disputar entre si para serem ativadas. A unidade vencedora tem seus pesos atualizados. As unidades de saída são geralmente ligadas entre si por conexões inibitórias ou negativas.

Este trabalho de pesquisa utiliza redes *multi-layer perceptron* (MLPs) e um algoritmo de aprendizado supervisionado baseado no algoritmo *back-propagation*. Estes conceitos serão apresentados a seguir.

3.4 Perceptron e MLPs

O modelo do neurônio *perceptron* foi proposto por Rosenblatt em 1958 e foi o primeiro modelo que apresentou o aprendizado com um professor ou supervisor. O *perceptron* nada mais é que modelo básico do neurônio artificial (figura 9) utilizando a função de ativação degrau (figura 10a, equação 3.3). Para o *perceptron* foi apresentado um teorema de convergência ((ROSENBLATT, 1958)) que garante que, para uma taxa de aprendizado constante e para dados de entrada de um problema que seja linearmente separável, o algoritmo de treinamento sempre converge. A atualização de pesos do *perceptron* ocorre como demonstra a equação 3.7 (também conhecida como regra delta).

$$w_i(n+1) = w_i(n) + \eta [d_i(n) - y_i(n)] x_i(n) \quad (3.7)$$

Na equação 3.7 o valor do peso $w_i(n+1)$ será atualizado pelo peso atual $w_i(n)$ somado com a multiplicação da taxa de aprendizado η , pelo erro representado pela diferença entre a saída obtida e a saída desejada do neurônio $[d_i(n) - y_i(n)]$, e pela entrada correspondente $x_i(n)$. O tempo de convergência do algoritmo pode ser controlado pela taxa de aprendizado η onde se for adotado um valor alto demais para este parâmetro haverá instabilidade no algoritmo e se for um valor baixo demais irá causar uma demora excessiva no tempo de convergência.

O perceptron se mostrou muito útil para solução de problemas linearmente separáveis, porém a maioria dos problemas do mundo real, onde é necessário que se aplique uma RNA para sua solução, não são linearmente separáveis. O trabalho de Minsky e Papert ((MINKSY; PAPERT, 1969)), ao salientar e comprovar o fato, ocasionou um abandono das pesquisas em RNAs pela maioria dos pesquisadores da área até 1980, quando foi apresentado o algoritmo de treinamento para a rede *Multi-Layer Perceptron* (MLP).

A rede MLP é constituída de mais de um neurônio perceptron, organizados em duas ou mais camadas. Para o caso de uma camada só (camada de saída), tem-se o que é classificado como *single-layer perceptron* e o algoritmo de treinamento é descrito pela equação 3.7, executada para cada neurônio. Neste caso a rede consegue resolver um grupo de problemas que ainda sejam linearmente separáveis.

Para que a rede consiga resolver problemas que não sejam linearmente separáveis é necessário adicionar camadas intermediárias entre as entradas e a camada de saída. Um problema que dificultou a criação de uma regra de aprendizado para este tipo de redes ocorria quando se adiciona uma ou mais camadas intermediárias e não existia o valor esperado para a saída destes neurônios, impossibilitando desta maneira o cálculo do erro pela equação $[d_i(n) - y_i(n)]$.

As camadas intermediárias da rede possuem um papel de transformar sucessivamente o problema representado pelos dados de entrada em uma representação tratável para a camada de saída da rede. Estas transformações atuam nos espaços de representação de cada camada. Segundo Braga ((BRAGA A.; CARVALHO A. P. L.; LUDERMIR, 2007)), a função da primeira camada intermediária é traçar as retas (os chamados "hiperplanos" de separação das classes) para formação da superfície do espaço de entrada, a segunda camada intermediária promove a combinação destas retas formando regiões convexas, e a camada de saída forma regiões combinando as regiões formadas na camada anterior. Geralmente a maioria dos problemas podem ser resolvidos com uma camada intermediária e no máximo duas ((CYBENCO, 1988)), sendo que aumentar indiscriminadamente o número de camadas pode prejudicar o tempo de convergência do algoritmo de treinamento.

Considerando o papel e a importância de cada camada em uma MLP, é necessário que exista um algoritmo que consiga ajustar os pesos de todas elas para que a rede aprenda sobre os dados de entrada de problemas complexos. Este algoritmo foi proposto por Rumelhart e foi aplicado com sucesso ((RUMELHART; MCCLELLAND, 1986), (MCCLELLAND; RUMELHART, 1986)) no treinamento de MLPs. O algoritmo, chamado *back-propagation*, possui duas fases: a fase *forward* que é quando o sinal é propagado pela rede gerando as saídas, e em seguida é iniciada a fase *backward* que utiliza as informações da saída obtida para o ajuste de pesos.

O princípio do algoritmo *back-propagation* é estimar o erro das camadas intermediárias pela estimativa do efeito que estas causam no erro da camada de saída. Esta informação pode ser obtida utilizando o método do gradiente descendente. A utilização do gradiente descendente obriga que a função de ativação seja contínua e diferenciável, como por exemplo, a função sigmoideal (figura 10c, equação 3.5). Utilizando este procedimento o erro da saída da rede é calculado e é retroalimentado para as camadas intermediárias, possibilitando o ajuste dos pesos proporcionalmente aos valores das conexões entre camadas ((BRAGA A.; CARVALHO A. P. L.; LUDERMIR, 2007)). A dedução das equações de ajuste podem ser encontradas em (HAYKIN, 1998) e (BRAGA A.; CARVALHO A. P. L.; LUDERMIR, 2007), e estas equações serão descritas aqui para referência com relação a sua alta complexidade computacional.

Primeiramente é necessário definir a equação do erro. O erro inicial será calculado como era feito para a rede *single-layer perceptron*, e é definido pela equação 3.8.

$$e_j(n) = [d_j(n) - y_j(n)] \quad (3.8)$$

Após definir o erro é necessário definir a equação de ajuste dos pesos. Neste caso o ajuste de pesos é feito por duas equações semelhantes. A estrutura básica é apresentada pela equação 3.9 onde a variação do peso que chega no neurônio j vindo do neurônio i , $\Delta w_{ij}(n)$ é igual a multiplicação da taxa de aprendizado η , pelo gradiente $\delta_j(n)$ e pela saída do neurônio anterior $y_i(n)$.

$$\Delta w_{ij}(n) = \eta \delta_j(n) y_i(n) \quad (3.9)$$

A principal mudança na equação da camada de saída para as camadas intermediárias é o cálculo do gradiente $\delta_j(n)$. Para a camada de saída o cálculo do gradiente, representado pela equação 3.10, é o produto do erro $e_j(n)$ (equação 3.8) pela derivada da função de ativação recebendo como entrada o resultado da equação 3.1, $\phi'(v_j(n))$.

$$\delta_j(n) = e_j(n) \phi'(v_j(n)) \quad (3.10)$$

O gradiente da atualização das camadas intermediárias, considerando que o neurônio em questão é o neurônio j e o neurônio posterior é o neurônio k , é representado pela equação 3.11.

$$\delta_j(n) = \phi'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \quad (3.11)$$

Nesta equação o gradiente é o produto da derivada da função de ativação recebendo como entrada o resultado da equação 3.1, $\phi'(v_j(n))$, pelo somatório, para todos os neurônios da camada posterior, $\sum_k$, do resultado da multiplicação do gradiente do neurônio posterior, $\delta_k(n)$ pelo peso entre o neurônio atual j e o neurônio posterior k , $w_{kj}(n)$. O fluxo de sinais do cálculo de um neurônio na fase backward pode ser visto na figura 13.

O algoritmo *back-propagation* possui um tempo de convergência alto, e para redes complexas com várias camadas este tempo pode tornar-se um problema. Outro problema do algoritmo é a atualização dos pesos que é feita utilizando-se o valor da derivada parcial da função energia em relação ao peso. O problema da utilização direta do valor desta derivada parcial é que mesmo adaptando-se a taxa de aprendizado η durante o treinamento, o efeito pode ser afetado drasticamente pelo comportamento imprevisível da própria derivada.

Para resolver este problema foi desenvolvido o algoritmo *Resilient-back propagation* (Rprop) por Riedmiller e Braun ((RIEDMILLER; BRAUN, 1993)). O algoritmo faz a atualização de pesos e da taxa em que os pesos são atualizados baseando-se apenas no sinal da derivada e

Figura 13 – Fluxo de sinal da fase backward para uma parte da rede ((HAYKIN, 1998)).

não em seu valor numérico. Para que seja possível proceder desta maneira, foi introduzido o valor de atualização individual de cada peso Δ_{ij} . As equações do cálculo do Δ_{ij} são descritas a seguir:

$$\Delta_{ij} = \begin{cases} \eta^+ * \Delta_{ij}^{(t-1)}, & \text{se } \frac{\delta E^{(t-1)}}{\delta w_{ij}} * \frac{\delta E^{(t)}}{\delta w_{ij}} > 0 \\ \eta^- * \Delta_{ij}^{(t-1)}, & \text{se } \frac{\delta E^{(t-1)}}{\delta w_{ij}} * \frac{\delta E^{(t)}}{\delta w_{ij}} < 0 \\ \Delta_{ij}^{(t-1)}, & \text{caso contrário} \end{cases} \quad (3.12)$$

Sempre que a derivada parcial mudar de sinal, o que significa que passou de um mínimo local o valor de atualização Δ_{ij} é decrescido pelo fator η^- . Caso o sinal seja o mesmo o valor de atualização é incrementado consideravelmente com objetivo de acelerar a convergência. Depois de calculado cada valor de atualização, o ajuste dos pesos ocorre de uma forma simples:

$$\Delta w_{ij}^{(t)} = \begin{cases} -\Delta_{ij}^{(t)}, & \text{se } \frac{\delta E^{(t)}}{\delta w_{ij}} > 0 \\ +\Delta_{ij}^{(t-1)}, & \text{se } \frac{\delta E^{(t)}}{\delta w_{ij}} < 0 \\ 0, & \text{caso contrário} \end{cases} \quad (3.13)$$

Existe apenas uma exceção: caso a derivada parcial mude de sinal o ajuste anterior de peso é revertido,

$$\Delta w_{ij}^{(t)} = -\Delta_{ij}^{(t-1)}, \text{ se } \frac{\delta E^{(t-1)}}{\delta w_{ij}} * \frac{\delta E^{(t)}}{\delta w_{ij}} < 0 \quad (3.14)$$

para que não haja uma dupla-punição neste peso, considerando que o sinal irá mudar novamente na iteração seguinte, $\frac{\delta E^{(t-1)}}{\delta w_{ij}} = 0$.

As variáveis envolvidas neste algoritmo são: o valor inicial da atualização de pesos Δ_0 , Δ_{min} e Δ_{max} que são os limites dos valores para atualização de pesos e os fatores de incremento η^+ e decremento η^- . É importante salientar que não existe taxa de aprendizado para este algoritmo.

O Algoritmo RPROP pode ser aplicado no lugar do *backpropagation*, sem alterações na rede neural ou na forma de tratar o problema, e por outro lado, este é um algoritmo de aprendizado bem mais rápido e mais fácil de ser utilizado (menos dependente do ajuste de parâmetros) do que o *backpropagation*, e em função disto tem sido muito usado no lugar deste.

3.5 Comitês de RNAs

Visando obter uma classificação mais correta por parte das RNAs foram criados os comitês de redes neurais artificiais (do inglês *Neural Network Ensembles*, NNEs). Os principais objetivos de unir RNAs em um comitê são: diminuir o erro de generalização e melhorar o desempenho das redes. Os comitês classificam os dados por diferentes redes e depois, por um esquema de votação, decidem qual o resultado final do comitê. Esta técnica é baseada em técnicas de sistemas tolerantes a falhas ((HANSEN; SALAMON, 1990)).

3.5.1 Definição e Considerações

Comitês de RNAs são um paradigma de aprendizado onde uma coleção de membros finita de redes neurais artificiais é treinada para solucionar o mesmo problema ((ZHOU; WU; TANG, 2002)). Os comitês foram criados pelo fato de que a seleção dos pesos das RNAs é um problema de otimização com um grande número de mínimos locais. A inicialização com valores aleatórios dos pesos de uma rede, e o algoritmo de treinamento que irá ajustar estes pesos sendo dependente de seu estado inicial, tende a diferenciar os erros de cada rede e desta forma as redes irão cometer erros em grupos diferentes do conjunto de entrada. A decisão tomada por um comitê tem uma probabilidade menor de estar errada do que a decisão de uma única rede. A figura 14 ilustra o fluxo de dados em um comitê de RNAs.

Figura 14 – Fluxo de dados em um comitê de RNAs composto por 3 redes.

Para formar um comitê de redes neurais existem várias técnicas. As redes podem possuir a mesma arquitetura (*homogeneous*), arquiteturas diferentes (*heterogeneous*) ou tipos diferentes

(*mixed*) e serem treinadas com conjuntos de dados iguais (Data Agreement) ou diferentes. A heterogeneidade das redes pode trazer benefícios ao comitê ((KROGH; VEDELSBY, 1995)). A escolha das redes que irão compor o comitê pode ser feita treinando-se várias redes e dentre as redes treinadas as melhores são escolhidas para compor o comitê ((PERRONE; COOPER, 1993)), técnica chamada de *Overproducing and Choosing* ((GIACINTO, 2001)). Segundo Johansson ((JOHANSSON; LOFSTROM; NIKLASSON, 2005)), esta técnica pode gerar comitês com maior acurácia. O tempo de treinamento neste caso pode ser uma questão que torne inviável a técnica. Escolher apenas as melhores redes também apresenta o problema de se descartar redes que podem possuir informações importantes sobre o conjunto de dados e irão demonstrar o fato durante a execução do comitê. A solução para este problema é selecionar as redes reduzindo ao máximo o número de falhas conjuntas ((SHARKEY, 1996)).

Outra forma de compor um comitê seria treinando apenas o número correto de redes e utilizando todas elas, técnica chamada de *Direct Strategy* ((GIACINTO, 2001)). Algoritmos de inteligência computacional também podem ser utilizados para criar comitês de redes neurais, como por exemplo, os algoritmos evolutivos ((KROGH; VEDELSBY, 1995),(TIAN *et al.*, 2012)).

A combinação de RNAs é útil apenas quando as redes discordam com relação aos dados de entrada ((KROGH; VEDELSBY, 1995)). O comitê se aproxima do ideal quando as RNAs são totalmente conectadas e discordam o máximo possível sobre as entradas. Pode-se definir então um bom comitê como um grupo de RNAs que não sejam totalmente corretas, e sim que cometam erros não-correlacionados (*Indirect Strategy* em diferentes partes do espaço de busca ((KROGH; VEDELSBY, 1995)). Uma alternativa para gerar redes que errem em partes diferentes do espaço de busca é utilizar a técnica de *cross-validation* particionando os dados que cada rede irá receber no treinamento ((TUMER; GHOSH, 1996)).

Giacinto ((GIACINTO, 2001)) aplicou os comitês de RNAs para classificação de imagens. O trabalho comparou várias mudanças de parâmetros durante a criação do comitê e as mudanças que apresentaram melhores resultados foram: a mudança nos tipos de redes criando assim comitês heterogêneos, e o treinamento com conjuntos de dados de treinamento diferentes. Outra estratégia proposta também foi a separação das redes treinadas em grupos e depois as que cometem erros de forma mais independente são selecionadas. Outros exemplos de aplicação de comitês de RNAs são descritos por Zhao ((ZHAO; GAO; YANG, 2005)).

3.5.2 Esquemas de Votação

A votação em comitês de RNAs pode ser feita de várias maneiras, inclusive probabilisticamente. Algumas das alternativas para os esquemas de votação são ((BATTITI; COLLA, 1994)):

- *Accuracy by Agreement*: quanto maior o número de RNAs que concordam com um

Figura 15 – Possibilidades para Composição de Comitês.

determinado resultado, este resultado tem seu grau de acerto elevado. Neste esquema de votação as redes tendem a produzir erros não-correlacionados, e sua utilização é justificada quando existem duas ou mais classes de confusão.

- *Union Scheme*: um resultado só é aceito quando todas as redes do comitê votam no mesmo resultado. Este esquema é arriscado, pois as classes de dados devem estar muito bem representadas pelos dados de treinamento para que o resultado seja satisfatório.
- *Majority Rule*: o resultado é aceito quando em um comitê com N redes, pelo menos $N - M$ redes concordam. O parâmetro M pode ser ajustado de acordo com o problema. Caso $M \leq \frac{N}{2}$ o dado é aceito só se a maioria das redes concordarem, e este esquema também é conhecido como *Majority Voting*. Este esquema pode gerar empates ou pequenas diferenças de votos, um problema que pode ser solucionado alterando-se o parâmetro M .
- *Averaging*: o resultado final (ou vetor de resultados finais) é composto pela média dos valores de saída das redes. Os dados que não possuírem classe definida ao final do cálculo da média podem ser classificados em uma classe padrão. Este esquema de votação suaviza os efeitos da inicialização aleatória das redes do comitê e, em alguns casos, pode aumentar a confiabilidade do resultado.
- *Output Values*: utilizar os próprios valores de saída das redes é um esquema válido porém devem existir regras de aceitação como por exemplo: deve existir um limiar para que um

valor seja considerado de determinada classe, e a diferença entre dois valores ativos deve também possuir um limiar para ser considerada.

Outras técnicas de votação mais elaboradas como *Supra Bayesian* e *Stack Generalization* podem ser encontradas no trabalho de Sharkey ((SHARKEY, 1996)).

A escolha de um esquema de votação depende especificamente dos requisitos da aplicação e da dependência entre as redes. Krogh and Vedelsby ((KROGH; VEDELSBY, 1995)) sugerem que os resultados devem ser combinados apenas por *majority voting* ou *averaging*. Comitês para tarefas de classificação apresentam melhor resultado com técnicas de votação baseadas em *Majority Voting* ((ZHOU; WU; TANG, 2002)). Um resumo das possibilidades de composição de comitês pode ser visto na figura 15.

3.6 Aceleração e Paralelismo

O cálculo que é executado pelas RNAs possui um elevado nível de paralelismo que induz implementações e otimizações que explorem de uma forma mais inteligente esta característica, em comparação com a execução sequencial de um processador.

Existem várias maneiras de se executar as operações das RNAs de forma paralela. A mais direta seria programar a rede de forma a executar cada cálculo em *threads* ou processos através de programação concorrente. Desta forma o sistema poderia ser executado de forma mais eficiente tanto em processadores que possuem vários núcleos como em sistemas distribuídos.

Considerando a possibilidade de modificação do hardware, ou utilização de tipos diferentes de hardware, existe a possibilidade de se implementar as redes em GPUs (*Graphics Processing Units*) buscando um melhor aproveitamento de operações de ponto flutuante, FPGAs para que o hardware possa ser desenvolvido para ser embarcado utilizando diferentes precisões e com um consumo de energia menor, DSPs visando acelerar os cálculos podendo ser utilizados juntamente com os FPGAs, ou ASICs e ASIPs onde a fabricação do hardware apresenta uma velocidade de execução maior e um consumo de energia menor em comparação com os FPGAs comprometendo a flexibilidade. A tabela 1 apresenta uma comparação entre os dispositivos para implementação de RNAs em hardware, onde + + + seria totalmente favorável e — — desfavorável.

3.7 Redes Neurais em Hardware

O paralelismo das RNAs sempre despertaram o interesse de pesquisadores e empresas para implementações em hardware ((MISRA; SAHA, 2010)). Existem várias opções de *chips* comerciais que implementam redes de diferentes modelos e arquiteturas, e também uma ampla quantidade de trabalhos de pesquisa nesta área. O interesse em implementações em hardware

Tabela 1 – Comparação de hardwares para implementação de RNAs ((OMONDI; RAJAPAKSE, 2006)).

	ASIC Analógico	ASIC Digital	FPGA	Baseadas em Processador	Computadores Paralelos
Velocidade	+++	++	+	-	+
Área	+++	++	+	-	-
Custo	-	-	++	++	-
Tempo de Design	-	-	++	+++	+
Confiabilidade	-	+	++	++	++

para redes neurais é originado do fato de que o paralelismo somente pode ser totalmente atingido quando as redes são implementadas em hardware ((LIU; LIANG, 2005)). O fato da estrutura das RNAs serem altamente paralelas e regulares é uma vantagem, pois permite a reutilização do hardware desenvolvido ((SIMOES; UEBEL; BARONE, 1996)).

A estrutura básica de uma implementação de HNN (*Hardware Neural Network*) pode ser vista na figura 16 ((AYBAY, 1996)). As entradas do sistema são os pesos e os dados de entrada a serem propagados pela rede, sendo que os pesos são armazenados no bloco de pesos e as entradas enviadas para o bloco de ativação. O bloco de ativação é responsável por executar os cálculos de cada neurônio e enviar a saída para o bloco da função de ativação. O resultado do cálculo da função de ativação é enviado para o bloco de estado da rede. Caso seja um resultado de saída da rede, é enviado como saída, e caso seja um resultado intermediário volta para o bloco de ativação. O bloco de ativação possui um sinal de controle para garantir a correta execução da propagação.

Figura 16 – Representação da arquitetura básica de HNNs.

A figura 17 apresenta a estrutura básica de um perceptron em hardware. Este neurônio possui três estágios: (i) multiplicação das entradas pelos pesos, (ii) acumulação e (iii) cálculo do valor de saída. Neste caso a função de ativação é implementada por uma LUT.

Figura 17 – Estrutura básica de um neurônio em hardware. Adaptado de (SAVICH; MOUSSA; AREIBI, 2012)

As redes neurais implementadas em hardware podem ser classificadas de várias formas possíveis ((AYBAY, 1996)):

- Quanto ao tipo de hardware:
 - Neurochip - quando existe apenas um *chip* que implementa a rede.
 - Neurocomputador - quando a rede é formada por vários *chips* em cascata.
 - Computador de propósito geral.
 - Sistema embarcado multiprocessado ou monoprocessado com aceleração em hardware.
- Quanto aos neurônios:
 - Número de neurônios.
 - Forma de implementação.
 - Digitais, analógicos ou híbridos.
 - Precisão.
- Quanto aos pesos:
 - Armazenamento dos pesos na estrutura ou armazenamento remoto de pesos.
 - Número de sinapses.
 - Digitais, analógicos ou híbridos.
- Ativação dos neurônios:

- Probabilística ou determinística.
- Função de ativação:
 - *On-chip* ou *Off-chip*.
 - Digital, analógica ou híbrida.
 - Cálculo em tempo de execução, *look-up table* ou *threshold*.
- Fluxo de informação:
 - *Systolic Array*,
 - Fluxo Direto,
 - *Broadcast*,
 - *Pipelining*,
 - Multiplexação.
- Entrada e saída de dados:
 - Digitais, analógicas ou híbridas.
- Treinamento:
 - *Online* ou *Offline*.

Existem algumas características que são comuns à maioria das implementações de HNNs. Muitas operações de multiplicação são executadas durante a propagação do sinal pela rede, o que acarreta em um alto consumo de área de circuito para acomodar o hardware que executa as operações ((DAALEN; JEAUVONS; SHAW-TAYLOR, 1993)). O número de multiplicadores necessário para a execução da rede cresce de forma quadrática em relação ao número de neurônios, e uma das formas de se reduzir este crescimento é a reutilização de multiplicadores ((BADE; HUTCHINGS, 1994)). Este é provavelmente o principal problema na implementação de RNAs em hardware ((SZABO; HORVATH, 2004)). Quando a implementação em hardware utiliza FPGAs, é possível acelerar o cálculo com estruturas chamadas multiplicadores-acumuladores (MACs). Estas estruturas consomem uma quantidade menor de área de chip devido à arquitetura dos FPGAs ((NEDJAH *et al.*, 2009)).

Uma das causas das dificuldades de implementação em hardware de RNAs é a alta complexidade do controle da estrutura que contém um alto número de multiplicadores. Para que esta estrutura seja considerada totalmente paralela, todos os produtos que ocorrem nos neurônios da mesma camada devem ser executados simultaneamente ((SZABO *et al.*, 2000)).

Outra questão importante na implementação das HNNs é a função de ativação. A implementação do cálculo da função em hardware é uma tarefa de alta complexidade ((SZABO;

HORVATH, 2004)). A principal alternativa para a implementação de HNNs é utilizar LUTs para implementar a função de ativação, procedimento que diminui o tempo de execução. Monmasson et. Al. ((MONMASSON *et al.*, 2011)) considera que a utilização de LUTs para implementar a função de ativação pode diminuir a área do hardware em até 70% em comparação com a implementação lógica do cálculo, sendo que as LUTs podem ser otimizadas e adaptadas a cada problema ((MEHER, 2010)).

A implementação em circuitos analógicos consome um número menor de componentes e o tempo de execução também é menor, porém o resultado é menos preciso. O hardware digital é preciso, porém não possui a mesma eficiência ((MISRA; SAHA, 2010)). O consumo de energia do hardware, independente do tipo escolhido, nem sempre é possível de ser adequadamente mensurado. Alguns *chips* possuem hardware para medição do consumo pré-implementado, porém a maioria não possui o que aumenta a complexidade da tarefa.

Ponto-flutuante, em HNNs, não é factível para implementação em FPGAs ((NICHOLS; MOUSSA; AREIBI, 2002)), ou seja, os custos associados à sua implementação não justificam sua utilização. Existe um aumento de área de aproximadamente seis vezes para utilização de ponto-flutuante, e o tempo de execução é ainda maior se comparado com a mesma implementação utilizando ponto-fixa.

A representação numérica utilizada pode aumentar ou diminuir o número de bits utilizados e este fato influencia na área final do hardware, e também no tempo de execução ((NEDJAH; MOURELLE, 2007)). No caso das HNNs este fato pode influenciar a escolha por uma representação numérica que necessite de número menor de bits, como por exemplo, o ponto-fixa.

O treinamento em HNNs, quando implementado em hardware, pode ser executado de forma mais rápida. Porém a implementação em hardware do treinamento exige uma área de hardware consideravelmente maior para cada neurônio incluindo estruturas que, após o treinamento, não serão utilizadas durante a execução. Além disso, o *clock* do hardware pode ficar prejudicado e sua eficiência também. Como o hardware para treinamento aumenta a área, a probabilidade de ocorrer uma falha no sistema também aumenta ((CHUN; MCNAMEE, 1990)).

O projeto de HNNs também pode ser feito com base em reusabilidade, sendo que para atingir este objetivo algumas características são necessárias ((IZEBOUDJEN *et al.*, 2007)): o projeto deve permitir sua reconfiguração para que possa atingir os requisitos de várias aplicações diferentes e deve utilizar uma plataforma padrão. O hardware deve ser sintetizável em *Register Transfer Level* (RTL) e o projeto deve permitir de forma robusta a utilização de *scripts*. Neste contexto os FPGAs são as plataformas mais adequadas porque permitem que o usuário modifique a estrutura do hardware para executar diferentes tipos de aplicação e, ao mesmo tempo, preservam o paralelismo das redes neurais ((YOUSSEF; MOHAMMED; NASAR, 2012)).

Quando um sistema utiliza uma HNN, existem várias alternativas de implementação. Para que seja possível escolher qual a melhor alternativa para um determinado sistema foram

desenvolvidas algumas métricas, encontradas na literatura, e que serão apresentadas a seguir.

3.7.1 Métricas

Existem várias métricas disponíveis para a comparação entre as HNNs ((KEULEN *et al.*, 1994)), porém a maioria dos trabalhos da área utilizam apenas um pequeno grupo para avaliar as implementações. As métricas mais comuns para avaliação de HNNs serão apresentadas de acordo com característica que avaliam. Para avaliar a velocidade de processamento são utilizadas as seguintes métricas:

- *Connections per Second (CPS)* - é o número de operações executadas por um circuito multiplicador-acumulador por segundo durante a fase de execução.
- *Connection Updates per Second (CUPS)* - quando o treinamento da rede é implementado em hardware, esta medida corresponde ao número de pesos que são atualizados por segundo durante a execução do treinamento.
- *Connections per Second per Weight (CPSPW)* - CPS/número de pesos.
- *Connection Updates per Second per Weight (CUPS)* - CUPS/número de pesos
- *Connection Primitives per Second (CPPS)* - o cálculo é feito utilizando-se a equação: $CPPS = b_w * b_i * CPS$, onde b_w é o número de bits por peso da rede e b_i é o número de bits por entrada da rede.

O tamanho das redes também pode ser comparado pelas seguintes métricas:

- *Number of Neurons per Chip (N_n)* - é o número de neurônios que podem ser implementados no hardware.
- *Number of Synapses per Chip (N_s)* - o número total máximo de multiplicadores que estão disponíveis no hardware.
- *Number of Weights per Chip (N_w)* - o número total de pesos que podem ser implementados no hardware.

Custos relacionados ao hardware são classificados da seguinte maneira:

- *Effective Synapse Area (ESA)* - obtido através da divisão da área total do hardware pelo número de sinapses ou pesos.
- *Power Dissipation (PD)* - máxima dissipação de energia do hardware à uma certa frequência de *clock*.

- *Energy per Connection* (EPC) - PD/CPS em uma determinada frequência de *clock*.

Segundo van Keulen existem algumas características necessárias aos critérios de comparação entre HNN ((KEULEN *et al.*, 1994)): o critério deve ser geral, possuir um alto nível conceitual, deve ser independente do problema, deve ter um alto poder de distinção e, o mais importante, os valores utilizados para o cálculo de um critério devem ser possíveis de serem obtidos.

3.7.2 Implementação de Redes MLP em Hardware

Alguns trabalhos apresentam o estado-da-arte em desenvolvimento de HHNs como, por exemplo, o trabalho de Misra ((MISRA; SAHA, 2010)). Estes trabalhos abordam a implementação em hardware de vários tipos de redes em diversos tipos de hardware. A seguir são analisadas as características específicas de trabalhos relevantes que implementam MLPs em hardware reconfigurável.

Bade e Hutchings ((BADE; HUTCHINGS, 1994)) desenvolveram o hardware para uma rede MLP, *feed-forward*, utilizando a linguagem VHDL. A implementação neste caso não possui treinamento em hardware e atingiu 2.2 GCPS utilizando 100 elementos lógicos do FPGA Xilinx 4000. LUTs foram utilizadas para gerar os pesos da rede, neste caso o diferencial do trabalho. Este fato aumentou consideravelmente o número de padrões avaliados por segundo. O hardware foi simulado e também houve uma execução real com um número reduzido de neurônios em comparação com a simulação.

Outro resultado de implementação real pode ser encontrado no trabalho de Botros e Abdul Aziz ((BOTROS; ABDUL-AZIZ, 1994)). Neste trabalho a função de ativação é calculada em hardware com algumas restrições, o que gerou um erro de saída da rede da ordem de 10^{-2} . A implementação em hardware atingiu 4MCPS e ocupou, com 11 neurônios, 2 FPGAs Xilinx XC3042. Esta implementação utilizou ponto-fixe e uma estrutura paralela heterogênea.

Existe a possibilidade de se implementar uma rede neural em hardware utilizando uma memória RAM ou redes neurais do tipo RAM ((SIMOES; UEBEL; BARONE, 1996)). A memória é construída como uma tabela verdade e armazena o mapeamento de todas as possíveis entradas e suas respectivas saídas. No trabalho de Simões et. Al., a rede atingiu aproximadamente 161MCPS com alta precisão. O dispositivo utilizado é o *chip* Altera Max II Plus e os dados obtidos são de uma implementação real. Uma das vantagens da implementação é que, como a rede não é realmente implementada, qualquer arquitetura e qualquer tipo de rede podem ser utilizados. A principal desvantagem é que o tamanho necessário para armazenar uma rede maior pode inviabilizar a implementação. Alguns pesquisadores da área não consideram esta solução como uma HNN.

Szabo et. Al. propõe um modelo que visa ser implementado tanto em FPGAs quanto em ASICs ((SZABO *et al.*, 2000)). O modelo não foi implementado ou passou por fase de

testes, porém o trabalho apresenta uma possível alta precisão devido à implementação utilizando aritmética *Bit-Serial* distribuída e multiplicadores matriz-vetor. Os autores propõem a utilização de LUTs para as funções de ativação, porém a forma como são calculados os valores não é detalhada.

A implementação real de Coric et. Al. ((CORIC; LATINOVIC; PAVASOVIC, 2000)) propõe a comparação de diferentes tipos de neurônios utilizando o FPGA Xilinx XC4000. A função de ativação foi implementada em uma LUT e o projeto ocupou para 5 neurônios: 196 blocos lógicos, 145 flip-flops e aproximadamente 5000 portas lógicas. O *clock* final do projeto atingiu 29MHz, classificando um padrão a cada 211ns.

Krips et. Al. apresenta a implementação de um sistema de *hand-tracking* em tempo-real ((KRIPS; LAMMERT; KUMMERT, 2002)). O design foi feito inicialmente em MATLAB e em seguida comparado à execução real do hardware, desenvolvido em linguagem VHDL e sintetizado para o FPGA Xilinx Virtex XCV100. A função de ativação foi implementada em uma LUT, onde a tabela foi dividida em 15 níveis. A implementação de 7 neurônios ocupou aproximadamente 34% do dispositivo e atingiu o tempo de 71ns para uma propagação de sinal, atingindo alta precisão.

O trabalho de Yun et. Al. apresenta uma HNN expansível e reconfigurável ((YUN *et al.*, 2002)). A arquitetura utiliza representação em ponto-fixa com 24 bits e a função de ativação é implementada por uma LUT. O projeto foi desenvolvido em linguagem Verilog e simulado para o dispositivo Xilinx Virtex 2 6000. O hardware apresenta alta precisão e 60 neurônios ocupam 18712 slices. Nenhum teste de tempo de execução foi efetuado.

Representação de ponto binário (que utiliza a mesma ideia do ponto decimal, porém para a base 2) e um modelo estocástico são utilizados no trabalho de Nedja e Mourelle ((NEDJAH; MOURELLE, 2003)). Neste trabalho a função de ativação é implementada utilizando um circuito comparador simples. A simulação do hardware descrito em linguagem VHDL foi feita para o FPGA Xilinx Virtex-E. O resultado do trabalho demonstra que quando se utiliza o ponto binário é possível otimizar o tempo de execução.

Ferrer et Al. implementa, em linguagem VHDL uma HNN parametrizável. O FPGA utilizado é um Altera ARC-PCI EPF10K50, e a representação numérica escolhida é o ponto-fixa por suas características que facilitam a parametrização. A função de ativação é uma função *piecewise-linear* onde as diferenças de inclinação são potências de 2 permitindo, portanto, uma implementação em hardware de forma mais simples. O hardware apresenta alta precisão e atingiu 16,67 MHz com 8 multiplicadores de 8 bits e 4×10^5 CPS. Esta estrutura ocupou 95% do FPGA para 19 neurônios.

A simulação de uma HNN, descrita em VHDL, para o FPGA Xilinx Virtex 2 6000 é apresentada por Domingos Et. Al. ((DOMINGOS; SILVA; NETO, 2005)). Nesta simulação a rede é implementada com representação de ponto-fixa de 16 bits atingindo 121MHz de *clock*. A

função de ativação foi mapeada para uma LUT, e juntamente com o conjunto de MACs utilizados, proporcionam alta precisão a uma taxa de 21.4MCPS. Todo o hardware, com um total de 12 neurônios, ocupa 1248 *slices*.

Uma alternativa interessante para o consumo de elementos lógicos é apresentada por Canas et. Al. ((CANAS *et al.*, 2006)). O projeto permite que a rede possua diferentes números de bits para as camadas da rede, diminuindo a área do hardware, e utiliza representação de ponto-fixa em complemento de 2. A descrição do hardware é feita em VHDL tendo como alvo o FPGA Xilinx Virtex-E 2000. LUTs são utilizadas para a função de ativação sendo o erro da saída da rede de 10^{-3} e a velocidade de propagação de uma entrada 13.710ms.

Os FPGAs possuem um número limitado de elementos lógicos e, para redes que possuem muitos neurônios, é necessário diminuir ainda mais o consumo de hardware. O trabalho de Himavathi et. Al. propõe a implementação apenas da maior camada da rede em hardware, e a utilização de multiplexadores para permitir que a estrutura calcule os resultados de todas as camadas da rede ((HIMAVATHI; ANITHA; MUTHURAMALINGAM, 2007)). Uma rede real, utilizando esta técnica e configurada em um FPGA Xilinx XCV400hq240, conseguiu uma redução de 70% da área consumida de hardware em comparação com a implementação do hardware com todas as camadas da rede. O *overhead* no tempo de execução foi 25%, consumindo 317 ciclos de *clock* a 73MHz.

A multiplexação de camadas também é utilizada como princípio básico das TNNs (*Tiny Neural Networks*) ((MORENO *et al.*, 2008)). Estas redes possuem uma unidade de aprendizado básica como elemento central. Esta unidade pode ser modificada de forma a acomodar um ou vários neurônios e os barramentos são responsáveis por conectar as estruturas de forma correta. O hardware é apenas simulado para o FPGA Altera Cyclone II, onde 33 neurônios ocupam pouco mais de 30% do dispositivo com *clock* de 47.21MHz. O treinamento *on-chip* é implementado em um processador Power PC separado.

Além das soluções totalmente implementadas em hardware, existe a possibilidade de se implementar as HNN como co-projetos de hardware/software. O trabalho de Petko e Karpiel ((PETKO; KARPIEL, 2007)) apresenta uma implementação que é constituída de um *soft-processor* Nios II, instruções customizadas e hardware dedicado que pode utilizar ponto-fixa ou ponto-flutuante. O projeto, descrito em linguagem VHDL, ocupou aproximadamente 50% dos recursos do FPGA Altera EP1S30 e, com um *clock* de 55MHz, classifica padrões a cada 25us. A implementação da função de ativação não foi mencionada e o erro é calculado especificamente para o problema tratado. Braga et. Al. ((BRAGA *et al.*, 2010)) também propõe um co-projeto utilizando o *soft-processor* MicroBlaze da fabricante Xilinx nos FPGAs Spartan-3E e Virtex 5. Todo o projeto foi desenvolvido utilizando a linguagem VHDL e apenas simulado. O erro estimado é de 10^{-3} e várias formas de implementação das funções de ativação foram testadas como LUTs, *piecewise-linear* e combinatorial. Os projetos apenas em hardware ocuparam uma área menor do FPGA, porém atingiram erros maiores. O melhor projeto totalmente em hardware

apresentou melhores índices de performance, consumo de energia e utilização de recursos em comparação com o melhor co-projeto que utilizou o *soft-processor*. A ordem de grandeza do erro não justifica a utilização do co-projeto, utilizando *soft-processors*, neste tipo de aplicação.

MACs são estruturas comuns em HNNs devido ao alto número de multiplicações executadas na propagação do sinal pela rede. O trabalho de Nedjah et. Al. ((NEDJAH *et al.*, 2009)) propõe uma estrutura para HNNs baseadas em MACs, porém não há resultados de implementações ou simulações. A função de ativação é aproximada por uma função quadrática polinomial utilizando o método de mínimos quadrados.

As HNNs são também utilizadas para aplicações em robótica, como por exemplo, no trabalho de Fuente et. Al. ((FUENTE *et al.*, 2010)). Este trabalho implementa um robô móvel servo-visual que utiliza uma rede neural em hardware desenvolvida em MATLAB. A rede foi simulada, após a geração do hardware em linguagem VHDL, para o FPGA Xilinx Spartan-3A. Este trabalho utiliza a representação de 32 bits em ponto-fixa e implementa a função de ativação utilizando uma LUT em memória RAM. O projeto com 52 neurônios e *clock* de 100MHz ocupou aproximadamente 30% do dispositivo permitindo uma classificação a cada 0.52us.

Atualmente a pesquisa em HNNs busca estruturas que apresentem bom desempenho e níveis de parametrização. Considerando redes que apresentam o treinamento *on-chip*, dois trabalhos apresentam resultados de simulação. Wu et. Al. apresenta uma estrutura de hardware para o processamento do treinamento, que aceita palavras de controle de até 31 bits ((WU; ZHU; LIN, 2011)). A estrutura utiliza compartilhamento de recursos e a unidade de controle sincroniza a execução da rede, para que seja possível reduzir o tamanho do hardware. A função de ativação foi aproximada por equações lineares, porém o erro da rede não foi calculado. A estrutura, desenvolvida em linguagem Verilog para o FPGA Xilinx Virtex-4, possui *clock* de 37MHz e propaga o sinal de uma entrada em 0.25s. Os 21 neurônios implementados ocuparam 2560 *slices*. O trabalho de Gomperts et. Al. propõe um hardware parametrizável desenvolvido em linguagem VHDL para o FPGA Xilinx Virtex-5 ((GOMPERTS; UKIL; ZURFLUH, 2011)). Um framework é configurado e gera o código da rede de acordo com as necessidades da aplicação. A arquitetura apresenta alta precisão e implementa as funções de ativação em LUTs. A velocidade de execução alcançada pela arquitetura é de 536MCPS ou 106MCUPS para os problemas analisados.

O trabalho de Santos ((SANTOS *et al.*, 2011)) apresenta um co-projeto de hardware/-software no FPGA Cyclone II da fabricante Altera. A rede é implementada no *soft-processor* Nios II incluindo uma instrução customizada para a LUT que armazena a função de ativação. Os resultados da implementação real possuem alta precisão e um tempo de propagação de 0.007s para cada padrão. O co-projeto ocupou 33% do dispositivo e, por implementar a rede em software, apresenta alta escalabilidade.

O framework proposto por Skoda et. Al. ((SKODA *et al.*, 2011)) implementa uma biblioteca em linguagem VHDL com as unidades básicas de uma rede neural implementadas, um gerador de redes também desenvolvido em VHDL e o gerador de configurações para a rede. O

sistema foi desenvolvido e simulado para o FPGA Xilinx Virtex-5. A função de ativação é uma LUT fornecida pelo usuário, portanto a precisão da estrutura irá depender da LUT fornecida. Na simulação executada a rede obteve um tempo de propagação de 6.4us em um *clock* de 85MHz.

Hariprasath e Prabakar apresentam uma proposta diferente de implementação em hardware, onde o objetivo é conseguir colocar o maior número de neurônios por *chip* independente do tempo de execução (([HARIPRASATH; PRABAKAR, 2012](#))). Os resultados apresentados são da simulação do hardware, desenvolvido utilizando linguagem VHDL, para o FPGA Altera Stratix III. Três tipos de representação de ponto-fixa foram testadas variando apenas o número de bits da parte inteira e da parte fracionária. As mudanças não representam um aumento ou diminuição de forma linear no consumo de recursos. A implementação da função de ativação utilizou aproximação linear que gerou um erro máximo absoluto de 6.59% em comparação com o valor da função sigmoide real. Uma rede com 2 entradas, 2 neurônios na camada escondida e 1 neurônio de saída classificou um padrão em 34ns e ocupou 132 LUTs e 148 registradores do FPGA.

A implementação real de HNNs mais recente a ser analisada é o trabalho de Joost e Salomon (([JOOST; SALOMON, 2012](#))). Este trabalho apresenta um neurônio diferente, TiCO (*Time Coded Output Neuron*), baseado em implementações em hardware analógico. O trabalho utiliza uma aproximação linear da função sigmoide que mantém uma alta precisão. O treinamento é feito em hardware atingindo, no melhor caso, 1.19TCUPS. O projeto foi desenvolvido em linguagem VHDL e possui portabilidade entre FPGAs das fabricantes Xilinx e Altera. O hardware, para 64 blocos básicos que formam os neurônios TiCO, ocupou 212 elementos lógicos.

Ao analisar os trabalhos apresentados é possível perceber que algumas alternativas de projeto se tornaram constantes sugerindo um caminho a ser seguido durante a implementação de HNNs. A grande maioria dos trabalhos utiliza LUTs para implementar a função de ativação, o que diminui o tempo para propagação do sinal pela rede e também ocasiona um erro que, na maioria dos casos, é aceitável. Grande parte dos trabalhos utiliza FPGAs da fabricante Xilinx, sendo que os mais recentes já demonstram interesse maior por FPGAs da fabricante Altera. Uma pequena quantidade de trabalhos apresenta o treinamento da rede em hardware. Outro grande problema de grande parte das implementações apresentadas é que o hardware é apenas simulado e, por mais robusta que seja a simulação, não representa os problemas encontrados na implementação real do hardware no FPGA. A linguagem de descrição de hardware mais utilizada para desenvolvimento é o VHDL e a representação numérica que apresentou melhores resultados é o ponto-fixa. A quantidade baixa de neurônios por chip da maioria das implementações também é um problema.

Apesar das análises que foram feitas é importante ressaltar que os trabalhos não utilizam uma métrica padrão tornando difícil a comparação direta entre resultados. As métricas utilizadas são muito diferentes e as redes não possuem a mesma arquitetura. As funções de ativação

implementadas em LUTs são raramente descritas em detalhes o que dificulta o entendimento dos trabalhos. A adoção de um padrão de testes para as implementações de HNNs é imprescindível.

3.8 Considerações Finais

Este capítulo apresentou os conceitos básicos relacionados às RNAs como sua estrutura e os paradigmas de aprendizado. As redes *Multi-Layer Perceptron* foram apresentadas em seguida juntamente com seu algoritmo de treinamento *back-propagation* e uma variação, o Rprop. Em seguida foram apresentadas as principais alternativas de implementação para as redes de forma a melhorar o tempo de execução explorando o paralelismo inerente das RNAs. Um das formas mais utilizadas de se implementar em hardware as RNAs é utilizando a computação reconfigurável, paradigma que foi apresentado anteriormente. Considerando a natureza de implementação de hardware para veículos autônomos deste trabalho, o capítulo seguinte apresenta um estudo sobre importantes ferramentas de hardware desenvolvidas para veículos autônomos, apresenta também algumas placas de desenvolvimento de FPGA e um estudo comparativo entre sensores de câmeras CCD e CMOS, e também entre FPGAs e GPUs.

TECNOLOGIAS E DISPOSITIVOS

O desenvolvimento de sistemas embarcados envolve um estudo das possibilidades de utilização de hardware e sensores. Diversos tipos de câmeras foram desenvolvidas nos últimos anos visando obter melhor qualidade de imagem e também aumento na taxa de captura do sensor. Atualmente os sensores de imagem utilizados em câmeras digitais são ou CCD (*charged-coupled device*) ou CMOS (*complementary metal-oxide semiconductor*) e uma comparação entre eles é interessante para entender suas principais características. No caso do hardware para sistemas embarcados, a aceleração de tempo de execução está sendo proposta com base, principalmente, em duas tecnologias: os FPGAs (*field programmable gate arrays*) e as GPUs (*graphics processing units*). Este capítulo, portanto, inicia-se pela comparação entre os sensores de imagem e uma análise de algumas câmeras disponíveis no mercado. Em seguida é apresentado um comparativo entre FPGAs e GPUs, uma análise de placas de desenvolvimento de hardware que contém FPGA e, por fim, uma análise de dispositivos de hardware desenvolvidos para aplicação direta em veículos autônomos.

4.1 Sensores de Imagem

O mercado de sensores de imagens está concentrado em dois tipos básicos: o CCD e o CMOS. Estes sensores são duas tecnologias diferentes para captação de imagens em formato digital. Cada tecnologia possui seus pontos fortes em diferentes situações. Ambas convertem luz em carga elétrica e depois em sinais eletrônicos. No caso de sensores CCD a carga de cada pixel é transferida para um número limitado de nós de saída (normalmente apenas um) para ser convertido em tensão, armazenado, e então ser enviado *off-chip* como um sinal analógico. No caso do CMOS, cada pixel possui seu conversor carga-voltagem e o sensor normalmente inclui amplificadores, correção de ruído e circuitos de digitalização sendo que a saída é digital. A conversão feita separadamente em cada pixel gera menor grau de uniformidade, porém o processo é massivamente paralelo permitindo a transmissão de dados em alta velocidade. A

diferença na aquisição dos sensores é mostrada na Figura 18²¹.

Figura 18 – Diferença na captura de imagens em um sensor CCD e um sensor CMOS.

Aplicações de visão computacional necessitam de otimizações em dois parâmetros básicos: nível de ruído e velocidade de captura. As duas tecnologias de captura de imagens são diferentes na forma em que a imagem é capturada e convertida em sinais analógicos e digitais. Devido a sua construção, os sensores CMOS possuem um barramento de dados massivamente paralelo. Quando o sinal que está sendo processado encontra o gargalo dos sistemas, que normalmente é a interface entre a parte da captura e o circuito externo ao sensor, os dados do sensor CMOS já estão em formato digital. Em contrapartida os sensores CCD de alta velocidade possuem canais de saída rápidos, porém incomparáveis a estrutura paralela de saída dos sensores CMOS. O ruído na captura é maior em sensores CMOS, porém é possível incluir filtros de ruído no circuito de captura que diminuem problema. A natureza do circuito também faz com que o consumo de energia de sensores CCD seja maior. Alguns casos são exceções, como por exemplo, câmeras de infra-vermelho onde os CCDs são projetados para serem muito sensíveis no NIR (*near infrared*) sendo muito melhores que os sensores CMOS.

Considerando o problema do custo em sistemas embarcados, é necessário analisar esta característica dos sensores. Os sensores CMOS possuem a característica de serem mais caros (individualmente, em comparação com CCDs) porque utilizam máscaras de chip mais caras e, consequentemente, qualquer tipo de modificação no sensor afim de obter melhores resultados em determinada aplicação irá favorecer os CCDs. Porém, como os sensores são normalmente produzidos em larga escala os sensores CMOS apresentam um menor custo quando produzidos em grande quantidade. A dificuldade de fabricação de circuitos integrados fez com que por um tempo considerável os CCDs dominassem o mercado com menor custo, cenário que mudou por volta dos anos 90.

Considerando as tecnologias apresentadas para os sensores de imagem é interessante também analisar a capacidade atual das câmeras disponíveis no mercado. A capacidade de

²¹ Imagem adaptada de http://meroli.web.cern.ch/meroli/lecture_spatialResultion/clip_image002.jpg

aquisição de imagem e transferência de dados aumentou consideravelmente nos últimos anos. Segundo Suzuki (SUZUKI, 2010) a ideia principal dos sensores CCD é a diminuição do tamanho do pixel e dos sensores CMOS é obter imagens melhores do que as obtidas pelo olho humano. O desenvolvimento do sensor CMOS fez com que se tornasse a melhor escolha por fatores como o fato em 2009 o sensor CMOS atingiu o tamanho de pixels de $1.4\mu m$ sendo melhores portanto que o sensor CCD, e também que sensores CMOS podem operar em tensões baixas e alta velocidade de captura (SUZUKI, 2010). Ainda segundo Suzuki melhorias em sensores CMOS visam aprimorar suas capacidades buscando solução para problemas de brilho, velocidade e distância de objetos através de taxas de leitura de 1000 *frames/seg* com resolução de quadro de 4Kx2K e buscando também a criação de imagens 3D.

Embora as pretensões sejam audaciosas, os sensores atuais possuem características que permitem uma aquisição rápida de dados com qualidade suficiente para o processamento das imagens. A fabricante Point Grey possui experiência na fabricação de câmeras para diversas aplicações industriais, científicas, médicas, automotivas e de segurança. A empresa apresenta um documento com comparativos entre diversos sensores e várias características com gráficos e tabelas chamado *Camera Sensor Review*²² em sua edição de 2016.

Neste relatório é possível encontrar alguns números interessantes como por exemplo: câmeras com resolução de até 4240x2824 pixels CCD e 2592x1944 pixels CMOS, captura de 120 frames/seg CCD e 162 frames/seg CMOS, tamanho de sensores de até 1/4". Segundo o relatório, o sensor com melhor eficiência em transformar a luz captada em sinal elétrico é um CMOS cuje eficiência atinge 80%. O *throughput* das câmeras podem atingir até 380.6 MB/s para sensores CMOS e 149.9 MB/s para sensores CCD ambos com interface USB 3.0. A taxa de resolução máxima em relação a velocidade máxima de captura também é dominada por sensores CMOS onde o melhor caso é cerca de 30% maior que o melhor caso de câmeras CCD.

Neste trabalho era necessário que a câmera fosse rápida e, principalmente, apresentasse uma interface amigável para integração com a placa de desenvolvimento utilizada. A câmera escolhida possui um sensor CMOS e interface de 40 pinos adaptada para a placa de desenvolvimento utilizada neste trabalho, custo acessível para um sistema embarcado e, também, um número considerável de trabalhos que utilizaram o mesmo sensor com sucesso em diversas áreas. Esta câmera, modelo *TRDB_D5M*²³, possui um número de pixels ativos de 2592x1944 de tamanho $2.2\mu m$. As imagens podem ser capturadas em modo RGB e sua taxa máxima de transferência é de 96MB/s a 96MHz. Em modo VGA pode atingir até 70 fps sendo que em resolução máxima atinge 15 fps. Considerando as especificações apresentadas anteriormente a capacidade da câmera escolhida não é a melhor disponível, porém permite o desenvolvimento do trabalho sem maiores problemas. A voltagem de trabalho é de 3.3V e seu custo é aproximadamente de \$80, valores razoáveis em comparação com os modelos da Point Gray que trabalham em tensão

²² <https://www.ptgrey.com/support/downloads/10625>

²³ <http://www.terasic.com.tw/cgi-bin/page/archive.pl?No=281>

mínima pela porta USB de 5V e possuem custo médio (com especificações semelhantes para o sensor) de \$500.

O objetivo principal desta análise é apresentar os tipos de sensores ópticos e quais as vantagens e desvantagens da utilização de cada sensor. A escolha da câmera *TRDB_D5M* está, portanto, justificada por ser uma tecnologia com vários benefícios interessantes para sistemas embarcados como descrito e também por ser facilmente integrável ao kit de desenvolvimento de FPGA utilizado neste trabalho. Como existe uma constante comparação entre tecnologias para processamento em alto desempenho, e muita informação é omitida ou utilizada de forma errônea, a seção seguinte aborda as vantagens e desvantagens das tecnologias mais utilizadas para processamento de imagens de alto desempenho: FPGAs e GPUs.

4.2 FPGA x GPU

Aplicações que utilizam imagens como dados de entrada possuem atualmente muitas maneiras de serem executadas e também otimizadas. Além dos FPGAs, outra plataforma atingiu resultados expressivos para otimização deste tipo de aplicações, principalmente a partir do momento em que sua possibilidade de programação tornou-se evidente, são as GPUs (do inglês, *Graphic Processing Units*).

GPUs são um tipo especial de processador, geralmente composto por vários núcleos, dedicado a operações de processamento gráfico presentes em várias plataformas como, por exemplo, videogames e computadores pessoais. Estes dispositivos, cujo melhor exemplo de utilização são as placas gráficas, implementam operações primitivas de processamento de imagens em hardware e permitem que estas operações sejam executadas paralelamente. Existe uma maior dificuldade quando se utiliza uma GPU para executar uma tarefa que não apresenta relação com imagens devido à sua estrutura de hardware ((JESSEN, 2009)). Para classificar aplicações que são desenvolvidas desta forma, ou seja, mapeadas e otimizadas para utilizar de forma inteligente a estrutura de uma GPU, utiliza-se o termo GPGPU (do inglês, *General Purpose computing on Graphic Processing Units*).

A partir da definição de GPU é possível classificar, pelo nível de paralelismo, três estruturas de hardware já mencionadas até aqui: (i) CPU: apresenta um nível de paralelismo utilizando blocos maiores de hardware como, por exemplo, processadores multicore; (ii) GPU: possui um nível de paralelismo extremamente alto porém em uma estrutura rígida e com características bem definidas e (iii) FPGA: possui um nível de paralelismo igualmente alto juntamente com a vantagem de ser reconfigurável ((JESSEN, 2009)).

Os desenvolvedores de aplicações para GPUs, e também os desenvolvedores de hardware em FPGAs, demonstram um grande interesse em comparar os resultados destas duas plataformas. Para analisar se é possível esta comparação, e como deve ser feita, serão analisados a seguir alguns trabalhos que apresentam comparações entre estes dispositivos para aplicações específicas.

Cope et. Al. ((COPE *et al.*, 2005)) em um dos primeiros trabalhos que apresentaram uma comparação entre as duas plataformas questionou se as GPUs tornaram os FPGAs redundantes para processamento de vídeo. Os resultados do artigo não somente apontaram que a resposta para esta pergunta é não, como também algumas características importantes dos FPGAs e das GPUs: (i) é possível implementar nos FPGAs barramentos que não necessitam de ciclos com instruções de busca e decodificação; (ii) a memória é usada de forma menos eficiente em GPUs; (iii) a memória on-chip pode ser utilizada de forma mais eficiente nos FPGAs e este fato supera várias limitações das GPUs tornando os FPGAs atrativos e (iv) GPUs não proporcionam um *throughput* adequado para aplicações com um número alto de acessos à memória.

No trabalho de Che et. Al. ((CHE *et al.*, 2008)) as principais características apontadas da GPU foram o baixo custo, o grau de paralelismo, número alto de núcleos e uma arquitetura fixa. Os FPGAs, segundo os autores, apresentam alta flexibilidade que permite que os programadores construam uma estrutura de comunicação entre os módulos que atue de forma mais otimizada e, também, explorem o *tradeoff* entre recursos e desempenho desenvolvendo o nível de paralelismo apropriado para a aplicação. O desempenho nos FPGAs também é maior pois o hardware desenvolvido não precisa ser dependente da arquitetura de *Von Neumann*, podendo atingir uma eficiência maior com relação ao consumo de energia. Além disto, existem atualmente várias opções de *IP Cores*, ou seja, blocos de hardware pré desenvolvidos que executam tarefas e podem ser incluídos no projeto de hardware. Os FPGAs apresentaram um resultado equivalente ou melhor que a GPU em vários casos, sendo que os autores concluem que o hardware ainda poderia ser aperfeiçoado.

O sincronismo nas duas plataformas, segundo os autores, possui também características diferentes. A GPU possui um alto custo associado ao sincronismo, pois as melhores funções são executadas em blocos locais sendo que o sincronismo global só pode ser feito após a inicialização de um novo núcleo de tarefas a serem processadas. No caso dos FPGAs vários tipos diferentes de barreiras podem ser implementados, assim como várias fontes de *clock* diferentes com frequências diferentes podem ser utilizadas.

Outro tópico importante abordado por (CHE *et al.*, 2008) é o custo de comunicação. No caso da GPU o tempo de transferência dos dados para a memória aumenta linearmente com o tamanho dos dados a serem transferidos o que influi diretamente no desempenho geral. No caso dos FPGAs a comunicação pode ser implementada de várias formas sendo que o único custo que é fixo é o de configurar o dispositivo. Esta operação é feita apenas uma vez antes da execução em dispositivos que não suportam reconfiguração em tempo de execução.

Fu ((FU, 2010)) propõe que algumas questões devem ser levadas em conta ao se desenvolver para GPUs e FPGAs. No caso da GPU deve se considerar se existe realmente um paralelismo muito alto, se existe um padrão regular de operações e de acessos à memória que serão executados e também se a memória disponível é suficiente. Os FPGAs apresentam um benefício muito grande quando se utiliza uma precisão menor, porém apresenta o desafio de se

desenvolver um hardware.

Savich et. Al. ((SAVICH; MOUSSA; AREIBI, 2012)) acredita que os FPGAs são melhores, quando comparados à outras técnicas como Clearspeed e CUDA (GPU), pois já foram amplamente utilizados em uma variedade considerável de aplicações e seu custo não é elevado.

Cope et. Al. ((COPE *et al.*, 2010)) apresenta a comparação entre GPU e FPGA para algumas classes de algoritmos. A análise do trabalho mostra que os FPGAs possuem algumas vantagens como a execução de vários estágios de pipeline concorrentemente e o controle total do conteúdo e da organização da memória. Cope considera que uma implementação de hardware em FPGA é um *tradeoff* entre o paralelismo em nível de iterações e o paralelismo em nível de operações buscando a melhor adequação ao algoritmo, respeitando requisitos de área. Outra característica considerada inerente dos FPGAs é o fato de se utilizar uma precisão menor para atingir um *speedup* maior. O número de acessos à memória da aplicação a ser otimizada foi um fator apontado como limitante para o desempenho das GPUs.

O trabalho de Fowers et. Al. ((FOWERS *et al.*, 2012)) conclui que os FPGAs são os dispositivos mais rápidos para aplicações que utilizam janelas deslizantes, e também os únicos que possibilitam atingir uma execução em tempo-real para aplicações ITL (do inglês, *Information Theoretic Learning*). Os resultados do trabalho também apontam que os FPGAs utilizam uma quantidade de energia consideravelmente menor do que outros dispositivos em várias situações sendo, portanto, segundo os autores, considerados os únicos dispositivos que apresentam uma real possibilidade de serem embarcados.

Outros exemplos de bom desempenho dos FPGAs são os trabalhos: (LUK, 2009), (KALAROT; MORRIS, 2010), (UREÑA *et al.*, 2012).

Alguns dos trabalhos apresentados mostram que o ciclo de desenvolvimento para a GPU é menor devido ao seu modelo de programação amplamente conhecido. Os trabalhos também demonstram que os FPGAs se mostram mais eficientes em ambientes de tempo-real onde os tempos de latência devem ser minimizados. A utilização de aplicações em ponto-flutuante possuem uma maior chance de obterem bons resultados quando implementadas em GPUs.

Ao comparar as duas plataformas, alguns trabalhos como (JONES *et al.*, 2010) e (PEIREIRA *et al.*, 2011) utilizam métricas específicas, plataformas específicas e apresentam conclusões gerais sobre as plataformas. Estas conclusões são questionáveis e é possível salientar que, por exemplo, a plataforma utilizada que é composta por FPGAs organizados de forma bem específica não apresentou bons resultados e não os FPGAs de forma geral. Portanto algumas conclusões devem ser levadas em consideração apenas para situações semelhantes.

Após a análise apresentada é possível concluir que os FPGAs possuem vantagens com relação as GPUs, e principalmente no escopo deste trabalho que considera sistemas embarcados e estudo do *tradeoff* entre precisão e *speedup*. Apesar deste fato, é importante salientar que esta comparação na maioria das vezes não se mostra justa pois o hardware configurado em um FPGA

é desenvolvido pelos autores dos trabalhos enquanto o hardware das GPUs é desenvolvido por empresas que possuem um amplo time de desenvolvimento e, portanto, o que é comparado com o hardware configurado nos FPGAs é o software desenvolvido para a GPU. Outro problema da comparação é o escopo para o qual as GPUs foram projetadas que pode não ser o escopo da aplicação em questão, prejudicando desta forma a comparação com o hardware específico configurado em um FPGA.

Caso seja necessária a comparação, algumas métricas possíveis são: área, consumo de energia, *speedup* e *throughput* ((COPE *et al.*, 2005)). A escolha entre as plataformas é uma questão também difícil onde conclui-se que, considerando a análise apresentada acima, deve ser feita baseada nas características inerentes de cada dispositivo e da aplicação alvo, e não na comparação direta entre ambos.

4.3 Desenvolvimento com FPGAs

O desenvolvimento de hardware utilizando FPGAs geralmente envolve a utilização de uma placa de desenvolvimento. As placas de desenvolvimento apresentam todo o circuito já soldado incluindo os controladores, as memórias e todo o hardware que é necessário para a utilização do FPGA juntamente com todas as interfaces disponíveis de entrada e saída de dados. O FPGA como chip individual é utilizado, portanto, somente como componente do circuito final.

Em relação à escolha de um fabricante, a questão é um pouco mais complexa. A especificação dos FPGAs de determinado nível fabricados em diferentes empresas é semelhante. Normalmente a escolha é facilitada por dois fatores: custo das ferramentas de desenvolvimento e disponibilidade de informações sobre os chips, seus problemas, suas atualizações e sua utilização. Os FPGAs da fabricante Altera possuem grande utilização em ambientes de pesquisa e universidades pois apresenta ferramentas de desenvolvimento funcionais em versões gratuitas bem como uma documentação consideravelmente completa sobre seus produtos.

A identificação dos chips de FPGAs são feitas por famílias. As famílias irão apresentar chips com características básicas semelhantes e diferenças no número de registradores, pinos de entrada e saída, blocos de DSPs, números de multiplicadores em hardware, dentre outros. Geralmente estas características são maiores em alguns chips e menores em outros para que possam ser utilizados em aplicações que possuam a demanda de uma característica específica. Alguns FPGAs também possuem processadores integrados. Soluções utilizando *soft-processors* não são suficientemente robustas com relação a tempo de execução para a maioria das aplicações que envolvem FPGAs. Devido à demanda de execução de software e hardware concorrentemente as famílias de FPGAs, nos últimos anos, começaram a englobar processadores inteiros implementados em hardware dentro de seu chip. Estes dispositivos são chamados de SoCs (Systems-on-chip).

Considerando a fabricante Altera existem 4 tipos básicos de famílias: Stratix, Arria,

Cyclone e Max 10. A família Stratix é considerada de alto desempenho por possuir um grande número de blocos programáveis, grande quantidade de memória, suportar velocidade de barramento alta, hardware com suporte a cálculos de ponto-flutuante e controle de consumo de energia no dispositivo. Esta família é a ideal para o desenvolvimento de sistemas de tempo real e diversas aplicações de processamento de sinais. O grande problema é o altíssimo custo que praticamente inviabiliza sua utilização somente para pesquisa e desenvolvimento em países onde é necessária sua importação. Um exemplo de kit de desenvolvimento Stratix V GX que não é o melhor FPGA disponível, já que a família Stratix se encontra na versão 10, possui um custo de compra no site²² de 25 mil dólares. É importante ressaltar que caso o preço deste kit fosse acessível ele seria o mais indicado para o desenvolvimento de qualquer sistema de tempo real.

A segunda opção de produtos da Altera é a família Arria. Apresentando um número reduzido de elementos lógicos em comparação com a família Stratix e também uma largura de banda mediana é uma opção mais barata para aplicações que necessitam de dispositivos de alta capacidade. As vantagens neste caso são a facilidade de integração entre os dispositivos e baixo consumo de energia em comparação com outros FPGAs. Esta família também apresenta opções com processadores integrados (SoCs). Considerando suas especificações seria interessante sua utilização, porém o custo novamente se torna proibitivo sendo que uma placa de desenvolvimento que possui um FPGA da família Arria de última geração custa em torno de 5 mil dólares. Este custo para uma placa de desenvolvimento também é limitante com relação à sua utilização.

Considerando a necessidade de vários dispositivos de entrada e saída, um número médio de elementos programáveis e um custo acessível a melhor opção é a família Cyclone. Com seus melhores kits a um custo máximo por volta de 2 mil dólares é a família que melhor atende a todos os requisitos de projeto. Seus kits possuem várias interfaces de entrada e saída de dados, muitos pesquisadores desenvolvem hardware para estes dispositivos sendo que a variedade de projetos em diversas áreas auxilia no desenvolvimento dos projetos. O programa acadêmico da empresa Altera também apresenta preços promocionais para kits da família DE2, o que viabiliza ainda mais sua utilização já que o preço final gira em torno de 300 dólares. Um hardware desenvolvido utilizando esta tecnologia poderia ser incluído no custo de um veículo sem aumento significativo de seu preço. Este é um diferencial para a escolha da plataforma de desenvolvimento.

A família MAX é a família de menor custo da fabricante Altera possuindo um número reduzido de blocos programáveis e interfaces de entrada e saída de dados. Indicada apenas para aplicações mais simples, o custo de seus kits de desenvolvimento fica em torno de 200 dólares. Esta placa não atende aos requisitos de entrada e saída de dados e também de unidades lógicas programáveis para acomodar todo o hardware deste projeto. A tabela 2 apresenta as principais características dos kits de desenvolvimento citados. Na tabela é possível notar a diferença de custo entre os kits lembrando que o custo do kit DE2-70 já está incluindo uma câmera e um display touchscreen. Ao analisar os elementos dos FPGAs de cada kit é notável

²² https://www.buyaltera.com/Search?keywords=stratix+kit&_ga=1.171914336.1336754980.1466376984&pNum=2

Tabela 2 – Comparação entre diferentes kits de desenvolvimento de FPGAs.

Placa	FPGA	Elem. Lógicos	Memoria Embarcada (MB)	Multiplicadores 18 bits	I/O para Usuários	Preço
100G	Stratix V GX	622	51	512	840	\$24,995
Arria V	Arria V	504	24	2312	704	\$3,995
DE2i-150	Cyclone IV GX	149	6	360	475	\$ 700
MAX10	MAX 10	49	38	144	360	\$ 200
DE2-70 Camera+Display	Cyclone II	68	1	150	622	\$ 874

a grande quantidade de pinos de entrada e saída do kit DE2-70 em comparação com os outros kits de menor custo, sendo que o número de elementos lógicos é baixo porém suficiente. O kit DE2-70 foi lançado no mercado a mais tempo do que todos os outros kits, porém ainda assim apresenta uma configuração interessante.

Veículos autônomos estão sendo desenvolvidos em diversos centros de pesquisa pelo mundo. Este cenário criou uma demanda por dispositivos de hardware dedicados para executar os sistemas de direção autônomos. O desenvolvimento destes sistemas em hardware reconfigurável é uma alternativa que esteve presente desde o início do desenvolvimento dos veículos autônomos, porém as empresas de hardware perceberam a falta de dispositivos dedicados no mercado e iniciaram sua produção. Atualmente existem dispositivos próprios para este fim e alguns serão apresentados a seguir.

4.4 Hardware para veículos autônomos

O desenvolvimento de sistemas tanto de direção autônomos quanto de auxílio ao motorista envolve uma complexa fusão de sensores e processamento de uma quantidade grande de dados. Estes dados são obtidos em tempo-real e processados para que seja possível a tomada de decisão por parte do sistema. Algoritmos utilizados nestes sistemas demandam alto poder de processamento. A ideia de se acelerar parte do sistema com hardware dedicado, ou até mesmo construir o sistema como um todo em uma plataforma de hardware, supre requisitos importantes do sistema como:

- Custo final do sistema - deve ser reduzido ao máximo sem comprometer as funcionalidades, pois o sistema deverá ser embarcado em veículos de produção em massa. Um alto custo do sistema irá acarretar em altos custos de produção e de produto final, o que pode inviabilizar os projetos
- Consumo de energia - o consumo de energia de um sistema de hardware dedicado é menor do que o consumo de um computador de propósito geral que normalmente compõe o hardware dos sistemas de direção autônoma
- Tamanho - o sistema não pode ocupar grandes espaços do veículo, o que acontece nos sistemas de direção autônoma atuais empregados em pesquisas (que ocupam boa parte ou

quase todo o porta-malas dos veículos)

- Eficiência - os sistemas atuais acabam falhando em determinadas situações por não estarem executando em tempo-real. O hardware dedicado pode ser capaz de atingir os requisitos de tempo real e auxiliar o sistema na tomada de decisões em tempos mais adequados para as diferentes aplicações automotivas.

O início do desenvolvimento de veículos autônomos considerou a utilização de computadores pessoais disponíveis no mercado para a execução dos algoritmos. Vários problemas podem ser encontrados com computadores pessoais utilizados para este fim, dentre eles: o hardware consumia espaço dos veículos (geralmente tomando todo o porta-malas); a alimentação do sistema é feita pelo sistema de bateria do próprio veículo, o que pode ocasionar problemas já que o consumo de energia é considerável para um computador pessoal e os sistemas elétricos de veículos (principalmente os atuais) são calculados com pouca margem para modificações; o número de conexões exigido pela grande quantidade de sensores, bem como seu correto gerenciamento, demandam um sistema robusto de entrada e saída de dados dificilmente encontrado em computadores pessoais; a confiabilidade de um computador pessoal para ser embarcado em um veículo é questionável pois existe muita movimentação, vibração, constantes mudanças de temperatura ambiente e o hardware comum não está preparado para este tipo de situação.

A grande demanda por dispositivos de hardware que atendam às especificações e requisitos para serem embarcados em veículos autônomos motivou empresas a apresentarem soluções específicas para esta finalidade, e algumas delas serão apresentadas a seguir.

A empresa NXP disponibiliza um produto chamado *BlueBox: Autonomous Driving Platform*²³. A plataforma apresenta um sistema dedicado a fusão de sensores e também um sistema embarcado para processamento de dados. Segundo o fabricante o produto atinge até 90000 DMIPS com apenas 40W. O sistema utilizado para fusão de sensores é composto por 4 ARM Cortex-A53 de 1GHz e um processador ARM Cortex-M4 dedicado ao controle de entrada e saída de dados. Como interfaces de entrada e saída de dados possui VIU, SDHC, UART, HDMI, FlexRay, FlexCAN, SIPI, LIN, Giroscópio + Bússola Digital, JTAG e Trace port. Já o sistema para processamento de dados é composto por 8 ARM Cortex-A57 de até 2GHz. Apresenta visualmente 8 entradas ethernet, 2 seriais e 2 usb 3.0. O produto está em pré-produção sendo que sua documentação disponível é de maio de 2016.

A empresa Nvidia está investindo significativa em sua tecnologia de GPUs. Foi criada recente uma divisão automotiva com objetivo de desenvolver soluções para serem embarcadas em veículos autônomos. A primeira movimentação neste mercado ocorreu com a placa de desenvolvimento Nvidia Jetson²⁴, porém atualmente a solução que está sendo considerada para o setor automotivo é a plataforma Drive PX²⁵. Segundo a fabricante a plataforma combina *deep learning*, fusão de sensores e *surround vision*. Com relação à fusão de sensores, a placa supostamente pode combinar dados de 12 câmeras, LIDARs, RADARs e sensores de ultrassom.

O foco do processamento utilizando esta plataforma está nas *Deep Neural Networks*, e um framework para trabalhar com o desenvolvimento utilizando esta ferramenta computacional está sendo desenvolvido. O hardware é composto por dois processadores Nvidia Tegra X1 que atingem, combinados, 2,3 Teraflops. A questão do consumo de energia do produto ainda não foi tratada com profundidade, provavelmente pelo fato de o produto ainda não estar disponível no mercado. Ainda não existem muitas informações porque o lançamento está previsto para o final do terceiro trimestre de 2016.

Uma alternativa versátil está sendo desenvolvida pela empresa ASI. Segundo a fabricante, o produto NAV²⁶ é um sistema que permite o veículo ser controlado tanto de forma manual quanto robótica. O objetivo do produto é ser facilmente instalável em qualquer veículo convertendo-o em um veículo autônomo. Devido às diferenças entre veículos, módulos hidráulicos e mecânicos foram desenvolvidos permitindo o acoplamento. O sistema é composto basicamente por dois tipos de módulos, a unidade de controle do veículo e a ponte. A unidade de controle do veículo é composta por um processador i.MX6Q ARM quad-core de 1.2GHz e os conectores de entrada e saída de dados são 4 *Deutsch DT15-12 Key*, 1 *Deutsch DRC22-50P*, 1 *M12 (8) Pin X Coded* e 1 *M12 (4) Pin D Coded*. Os dois últimos são responsáveis pela conexão gigabit ethernet, o conector *DRC22* permite a conexão de (4) CAN, (4) RS232, (2) Analog Inputs, OTG USB, (2) Quadrature Inputs, and (4) Digital Inputs e os conectores *DT15* são responsáveis pelo controle de energia dos sistemas. A ponte que teoricamente conecta a unidade de controle ao veículo não possui descrição. Vários sensores e atuadores são disponibilizados para conexão com a plataforma, dentre eles uma *SPOTPOINT* que é um encapsulamento de sensores como gps, antenas de rádio e câmeras, um botão de parada de emergência, um anel de esterçamento para ser conectado ao volante e uma unidade chamada de *FORECAST* que em sua essência é um LIDAR. O interessante deste produto é que, apesar da pouca documentação e especificações, são citadas algumas características de tolerância à falhas como por exemplo continuar funcionando com variações de tensão até 4,5V (teoricamente o circuito deve operar na tensão de 12V do veículo) e temperatura operacional de -40°C a 85°C .

A empresa *National Instruments* possui uma plataforma chamada *PXI*²⁷. O padrão desenvolvido para barramentos e comunicação combina características elétricas do barramento PCI com a possibilidade de construção modular do CompactPCI e Eurocard, além de incluir barramentos de sincronização e recursos controlados por software. A plataforma possui alto desempenho e baixo custo estando disponível no mercado desde 1998. Estas plataformas funcionam com sistemas de tempo real e são projetadas para operar em amplas faixas de temperatura e ambientes sujeitos a fortes choques e vibrações. O hardware das controladoras é baseado em processadores intel i7 e Xeon possuindo alto poder de processamento. Estas soluções são mais

²³ <http://tinyurl.com/ju57n2t>

²⁴ <http://www.nvidia.com/object/jetson-automotive-development-platform.html>

²⁵ <http://www.nvidia.com/object/drive-px.html>

²⁶ <http://www.autonomoussolutions.com/products/nav/>

²⁷ <http://www.ni.com/white-paper/7740/pt/>

próximas dos computadores pessoais, porém com desempenho melhor apresentando largura de banda de transferência de dados pelo barramento PXI de até 24GB/s.

4.5 Considerações finais

Este capítulo apresentou o estado da arte dos sensores de imagem, uma comparação entre as duas principais tecnologias de hardware para melhorar o desempenho de sistemas, a situação atual das opções para desenvolvimento com FPGAs e, por fim, a questão dos sistemas de hardware desenvolvidos com objetivo de suprir a demanda para o desenvolvimento de veículos autônomos. É interessante notar que de todos os sistemas de hardware apresentados o único que está realmente disponível para a venda e que já foi amplamente utilizado e testado é fabricado pela *National Instruments* e assim possui diversos problemas como o gerenciamento por sistema operacional Windows que não permite modificação, consumo de energia deve ser analisado por se tratar de uma plataforma baseada em computadores pessoais e também seu tamanho que não apresenta melhoras em comparação com soluções já utilizadas atualmente. Este fato demonstra a importância e relevância do tema para o cenário atual do desenvolvimento de veículos autônomos e sistemas de auxílio ao motorista.

O objetivo deste trabalho, considerando as informações apresentadas, é de grande importância no cenário atual de desenvolvimento de veículos autônomos pois apresenta uma alternativa aos sistemas que estão sendo propostos por empresas no mercado e ainda estão sendo fabricados e estarão disponíveis futuramente para serem testados. A escolha pela placa de desenvolvimento DE2-70 cujas características foram comparadas com as placas mais avançadas disponíveis mostrou-se adequada por não estar em desvantagem com placas de custo médio e também por possuir um histórico de desenvolvimento de projetos muito grande comparado com qualquer outra placa da mesma fabricante nas áreas de pesquisa deste trabalho. Além disto, o projeto desenvolvido baseado nesta placa (DE2-70) apresenta características e requisitos que podem ser facilmente adaptados para outras placas e projetos baseados em FPGAs. A câmera escolhida pode não ser a melhor disponível no mercado, porém apresenta todas as características desejáveis para permitir o desenvolvimento do sistema em questão. Comparando FPGAs e GPUs, após a análise apresentada, é possível concluir que a escolha pelos FPGAs possui diversas vantagens que justificam a escolha desta abordagem em detrimento de soluções alternativas que utilizem e sejam baseadas em GPUs. O capítulo seguinte apresenta o sistema desenvolvido, os resultados obtidos e também a análise destes resultados.

MÉTODO E FERRAMENTAS

Esta capítulo apresenta o método utilizado para o desenvolvimento deste trabalho e também as ferramentas escolhidas. Considerando que este trabalho utiliza como base o sistema de software para navegação de veículos autônomos proposto no trabalho de Shinzato (([SHINZATO; WOLF, 2011](#))), o sistema será descrito também neste capítulo bem como os pontos a serem abordados pela otimização em hardware.

5.1 Sistema de Navegação em Software

O sistema proposto por Shinzato (([SHINZATO; WOLF, 2011](#))) será detalhado a seguir. Os veículos autônomos desenvolvidos junto ao LRM-ICMC/USP permitem que sejam adotadas diferentes estratégias para executar a navegação autônoma, considerando diferentes abordagens e conjuntos de sensores. Entre as diferentes abordagens desenvolvidas junto ao LRM, uma delas teve bastante destaque devido ao seu baixo custo e resultados ([FERNANDES *et al.*, 2014](#)) ([SOUZA *et al.*, 2013](#)), sendo baseada no uso de um sistema de visão monocular. Uma vez processadas pelas redes neurais artificiais, as imagens resultantes com a detecção das zonas navegáveis/não navegáveis são então manipuladas de diferentes maneiras para o tratamento dos dados (representados no mapa de navegabilidade). Estas etapas posteriores é que irão então gerar os comandos finais de atuação no veículo. Portanto, o processamento e classificação dos elementos da imagem constituem a primeira etapa do processo de navegação autônoma como um todo, e é esta etapa que foi estudada e aperfeiçoada neste trabalho.

Utilizando uma câmera monocular de vídeo as imagens são capturadas (no sistema original em uma taxa máxima de 30 frames por segundo) onde o sistema recebe as imagens coloridas em tamanho $M \times N$ pixels (no sistema original: 320 x 240 pixels). Após receber as imagens, o sistema divide-as em vários blocos de tamanho $K \times K$ pixels (no sistema original: 10 x 10) (figura 19). Estes blocos contêm grupos de pixels de onde serão extraídas as informações das imagens que serão utilizadas como entradas do classificador, no caso um comitê de RNAs.

A estratégia de divisão da imagem em blocos de pixels é uma forma de tornar mais rápido o processo de classificação.

É importante destacar que estes blocos correspondem a uma janela deslizante sem sobreposição (*overlap*) que vai se deslocando e processando cada bloco, extraindo as informações e repassando para a RNA realizar a classificação de cada um deles. Portanto, considerando a independência do processamento dos dados de cada bloco em relação aos demais, este é um dos pontos que pode ser amplamente explorado a fim de executar em paralelo o processamento de diferentes blocos. Além disto, o tamanho e a sobreposição na varredura dos bloco também poderá ser um importante parâmetro a ser considerado, pois no trabalho anterior a solução adotada foi adequada, porém tendo sido imposta pela própria aplicação em termos de necessidades de tempo final de resposta do sistema.

Figura 19 – Procedimentos iniciais do método.

As informações extraídas da imagem são os atributos dos pixels como, por exemplo: as componentes de cor (RGB,HSV), entropia, energia, luminância e também são feitos cálculos de médias e medianas destes valores. Os conjuntos de atributos que possuem alta representatividade são selecionados dos blocos de pixels para que sejam utilizados na classificação das regiões da imagem. A escolha dos atributos é baseada no trabalho de Shinzato e Wolf (([SHINZATO; WOLF, 2011](#))). Estes conjuntos de dados são os dados de entrada do classificador. Fica claro que o processo de extração de atributos da imagem é uma etapa que pode ser configurada de diferentes maneiras e que possui também um alto impacto sobre o desempenho final do sistema.

O classificador é um comitê de RNAs, composto atualmente por cinco redes do tipo MLP. As arquiteturas das redes variam de acordo com o número de entradas, porém existe apenas uma camada intermediária usualmente com cinco neurônios e um neurônio de saída. As redes utilizam função de ativação sigmoide e possuem orientação *Feed-Forward*.

No comitê de RNAs, cada rede recebe um subconjunto distinto dos atributos extraídos de um bloco da imagem. Isto é o que confere a rede a capacidade de realizar classificações com alto grau de acertos e adequadas, mesmo sob a influência de fortes variações da luminosidade ambiente, inclusive com a presença de zonas encobertas pela sombra de algum elemento do ambiente. Uma questão que não foi abordada de forma mais aprofundada neste modelo foi a elaboração de um comitê que utiliza as mesmas características nas diferentes redes, e também a utilização da melhor rede, ou seja, a rede que acerta o maior número de vezes em relação a votação.

Constata-se aqui mais dois pontos onde pode ser amplamente explorado o paralelismo existente no sistema, primeiramente intra-rede, ou seja, na propagação interna dos sinais de ativação dos neurônios da RNA que podem ser tratados como unidades de processamento paralelo de dados, e depois, inter-redes, pois o comitê de RNAs é composto por múltiplas RNAs que trabalham todas em paralelo. Outro ponto a ser explorado é o fato de que usualmente as entradas da rede são obtidas a partir de valores numéricos inteiros e restritos a intervalos bem definidos (pixels da imagem), assim como as saídas desejadas da rede neural que representam a classificação binária de cada bloco, como sendo uma área navegável (segura) ou não navegável (não segura). O funcionamento básico do sistema é exemplificado na figura 20.

Figura 20 – Funcionamento básico do sistema de classificação de imagens.

As redes são treinadas com o algoritmo Rprop, que recebe os conjuntos de atributos como entrada e classificam cada bloco de pixels como uma região navegável ou não-navegável. Esta classificação é feita, portanto, para todos os blocos da imagem. O treinamento das redes do comitê é baseado em uma classificação prévia dos blocos da imagem, realizada pelo usuário de forma interativa, marcando as regiões navegáveis (p.ex. asfalto, estrada de terra, via) e regiões não navegáveis (p.ex. obstáculos, grama, áreas fora da pista), que definem as duas classes da saída da RNA. Assim é fornecida uma base de dados previamente classificados para o algoritmo de aprendizado da RNA do tipo supervisionado. Utilizando esta informação, várias redes são treinadas, e são escolhidas as redes que classificam o maior número de padrões corretamente. Este processo é feito para cada uma das 5 redes que compõem o classificador.

As redes do comitê então votam sobre a condição (classificação) do bloco que está sendo processado e o resultado das redes é combinado utilizando a média. Para resultados maiores que 0.7 a rede classifica o bloco como navegável, resultados menores que 0.3 são considerados não-navegáveis e resultados intermediários são considerados como erro de classificação. Estes

valores foram definidos por Shinzato. A combinação do resultado de classificação das redes do comitê para cada bloco de pixels forma um mapa de navegabilidade que pode ser utilizado de várias formas diferentes para a navegação autônoma. Um diagrama simplificado do sistema, onde se encontram destacadas as etapas que foram otimizadas por este trabalho de pesquisa e também a fase de treinamento offline do comitê, pode ser visto na figura 21. É importante destacar que neste trabalho a implementação em hardware é relativa ao uso/ativação da rede neural após seu treinamento ("em produção"), onde o treino da RNA ocorre de modo off-line, sendo implementado em software. Um estudo mais detalhado e discussão sobre esta opção de projeto é apresentado na seção 5.4.1.

Figura 21 – Diagrama que representa o sistema de navegação autônoma.

Um exemplo de utilização do mapa de navegabilidade para a navegação autônoma de veículos pode ser encontrada no trabalho de Souza, et. al. ((SOUZA *et al.*, 2011)). Este trabalho utiliza templates que identificam o formato, a posição e a orientação da região navegável, permitindo assim determinar diretamente o grau de esterçamento da direção do veículo (baseado em um algoritmo reativo para o controle da navegação autônoma, que foi adotado em alguns experimentos). A figura 22 apresenta 5 formatos diferentes de templates e a superposição destes (em vermelho) sobre a imagem classificada pela RNA (em tons de cinza). Os tons mais claros representam zonas navegáveis e seguras, e os tons mais escuros representam zonas não

navegáveis e, portanto, que devem ser evitadas pelo veículo. O sistema identifica qual template possui o melhor casamento com a zona navegável do mapa de navegabilidade, e depois de identificada esta situação, o sistema controla o esterçamento e velocidade do carro para continuar a navegação. Este sistema de navegação por imagens foi testado com o veículo autônomo CaRINA I, desenvolvido no LRM, onde resultados da navegação e do desvio de obstáculos usando este sistema de visão podem ser vistos no canal de vídeo do LRM^{28 29}.

Figura 22 – Formas possíveis dos templates: (a) reta, (b) e (c) curva leve, (d) e (e) curva acentuada. Adaptado de (SOUZA *et al.*, 2011)

O sistema de hardware deste trabalho buscou acelerar a composição do mapa de navegabilidade permitindo uma navegação mais segura e em velocidades mais elevadas. Considerando o exemplo acima, caso seja possível classificar um número maior de imagens no mesmo período de tempo, poderemos diminuir o tempo de resposta do sistema e aumentar sua capacidade e velocidade de reação. Portanto, o sistema poderá analisar melhor a situação do veículo proporcionando em baixas velocidades uma navegação mais robusta e viabilizar a navegação em velocidades mais elevadas.

5.2 Método

A implementação em hardware de um sistema já desenvolvido concentra-se em soluções para cada um dos componentes deste sistema que apresentem melhorias no sistema final e também que não apresentem nenhum tipo de comprometimento do tempo de execução e dos resultados alcançados (em termos da qualidade das respostas) pelo sistema em software. Uma das divisões possíveis para o sistema de navegação proposto por Shinzato é apresentada a seguir:

- Fase de treinamento

Captura da imagem: o sistema proposto por Shinzato utiliza a captação das imagens através do sistema operacional ROS (*Robot Operating System*), onde a câmera é conectada ao sistema por um nó e toda a captura é tratada por software.

Classificação por especialista: a primeira imagem capturada é disponibilizada para que seja classificada por um especialista em regiões navegáveis e não navegáveis. Este passo é feito de forma interativa apenas na inicialização do sistema.

²⁸ <https://www.youtube.com/watch?v=hD2azq7nA40>

²⁹ <https://www.youtube.com/watch?v=aJoDrLYS57Y>

Pré-processamento da imagem: a imagem é então dividida em blocos e ocorre a geração dos atributos de cada bloco que irão compor o conjunto de dados de treinamento do sistema. O processamento de imagens é realizado utilizando a biblioteca de software OpenCV.

Treinamento do sistema: é executado o algoritmo de treinamento das redes do classificador com o conjunto de dados gerado.

- Fase de execução:

Captura da imagem: assim como na fase de treinamento, a captação das imagens é feita através do sistema operacional ROS (*Robot Operating System*), onde a câmera é conectada ao sistema por um nó e toda a captura é tratada por software.

Pré-processamento da imagem: as imagens capturadas são divididas em blocos e ocorre a geração dos atributos de cada bloco que serão então apresentados ao classificador neural para serem classificados.

Classificação: os blocos, a partir de seus atributos, são então classificados como áreas navegáveis ou não-navegáveis da imagem. A classificação é realizada por um comitê de redes neurais implementado com o uso da biblioteca de software FANN³⁰. Os blocos classificados irão compor o mapa de navegabilidade que será utilizado para a navegação autônoma.

Figura 23 – Proposta de particionamento hardware/software do sistema para implementação.

³⁰ <https://www.youtube.com/watch?v=aJoDrLYS57Y>

Portanto é possível dividir o problema em diferentes partes que devem ser implementadas em hardware e sincronizadas. A primeira parte é a parte de captura da imagem onde uma câmera deverá ser conectada ao hardware e possibilitar a captura da imagem de duas formas diferentes sendo a primeira que retorne uma imagem para a classificação pelo especialista das áreas navegáveis e não-navegáveis, e a segunda para o funcionamento do sistema com a captura ocorrendo de forma contínua.

A fase de classificação pelo especialista deve permitir que um usuário possa selecionar na imagem as regiões navegáveis e não-navegáveis. Em seguida o banco de dados de treinamento é gerado com base nos atributos dos blocos classificados. O treinamento do sistema de classificação é uma parte importante do sistema. Como o classificador é composto por cinco redes neurais os algoritmos de treinamento deverão ser executados utilizando os bancos de dados gerados. Algoritmos de treinamento de redes neurais artificiais possuem alta complexidade e podem se tornar um problema ao ser implementados em hardware. Nesta tese optou-se por treinar as redes neurais em uma implementação em software, para depois poder usar estas redes neurais treinadas previamente (off-line), podendo então implementar e embarcar em hardware estas redes treinadas, realizando a classificação da área navegável em tempo real junto ao sistema robótico embarcado.

O pré-processamento das imagens, que seria outra parte do sistema (com requisitos de tempo-real), é responsável por considerar a divisão da imagem em blocos de pixels e extrair os atributos de cada bloco de pixels. Após extraídos os atributos são classificados pelo comitê de redes neurais e são novamente agrupados para gerar o mapa de navegabilidade. O classificador compõe a última parte do sistema a ser implementada. Considerando as características principais do sistema e a alta complexidade de algumas partes o particionamento proposto para este trabalho é apresentado pela figura 23.

Utilizando o particionamento proposto o sistema foi desenvolvido. A seguir serão apresentadas as ferramentas utilizadas para o desenvolvimento.

5.3 Ferramentas

O hardware reconfigurável utilizado para o desenvolvimento do sistema foi a placa DE2-70 (Figura 24) que possui o FPGA Cyclone II, da fabricante Altera, apresentada no capítulo 4. A escolha por este dispositivo para o desenvolvimento deste trabalho foi baseada em suas capacidades, na disponibilidade de uma câmera com interface desenvolvida com base nos slots de expansão da placa, pela confiabilidade da placa baseada no amplo desenvolvimento de projetos utilizando a mesma, pelo seu baixo custo (menos de USD\$100), baixo consumo de energia que é estimado em $1.7mA$ com $2.5V$ equivalendo a aproximadamente $4,25W^{31}$, e também pela disponibilidade do dispositivo no laboratório de pesquisa onde o trabalho foi desenvolvido.

³¹ <https://www.altera.com/products/fpga/cyclone-series/cyclone-ii/features/cy2-power-compare.html>

Figura 24 – Descrição dos componentes do kit DE2-70. (Fonte: <https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=53&No=226&PartNo=3>)

A importância de dispor do dispositivo no laboratório de pesquisa é a possibilidade de realizar testes reais com o hardware para avaliar o desempenho do sistema. Mesmo com bons simuladores as repostas dos testes podem apresentar diferenças. Este kit tem a vantagem de possuir dispositivos fabricados com intuito de facilitar a integração.

Como apresentado no capítulo 4 os dois tipos de sensores ópticos, CCD e CMOS, possuem vantagens e a escolha pelo sensor CMOS por suas características mostra-se mais interessante. A câmera escolhida para este trabalho, *TRDB – D5M*, possui alta taxa de transferência (até 70 quadros por segundo em resolução VGA), desempenho aprimorado para situações de pouca luz, procedimentos para pular linhas e colunas e diminuir o tamanho da imagem sem reduzir o campo de visão, dentre outras características que podem ser utilizadas em sistemas de navegação visual para otimização. Dentre suas características, uma que também contribui para sua escolha é o fato de sua interface ser projetada para ser amigável com relação aos slots de expansão de alguns kits de FPGA incluindo o kit DE2-70. Uma imagem frontal da câmera e de um kit utilizando a câmera podem ser vistas na Figura 25.

Este dispositivo é programado por uma linguagem de descrição de hardware e a linguagem escolhida para este trabalho foi a linguagem VHDL que possui características favoráveis ao desenvolvimento do trabalho como apresentado no capítulo 2. A representação numérica

Figura 25 – Câmera *TRDB – D5M* e sua montagem no kit DE2-70.

escolhida foi o ponto-fixo por apresentar estruturas de hardware mais simples em comparação ao ponto flutuante resultando em um tempo de execução menor como apresentado no capítulo 2.

Os testes foram realizados nas estradas que passam ao redor da cidade de São Carlos no estado de São Paulo. O sistema foi embarcado em um veículo controlado por um motorista habilitado e realizou a classificação das imagens desconectado do sistema de navegação autônoma. Utilizando o método e as ferramentas descritas neste capítulo o sistema de hardware foi implementado e testado em diferentes cenários apresentados no próximo capítulo.

RESULTADOS

O resultado deste trabalho é uma estrutura de hardware que executa a captura da imagem, faz o processamento de imagens baseados em redes neurais, como proposto em software no trabalho de Shinzato ((SHINZATO; WOLF, 2011)), e resulta em um mapa de navegabilidade para veículos autônomos. O sistema é apresentado da forma que foi desenvolvido neste capítulo, ou seja, de forma modular para que seja possível substituir os módulos sem que o sistema deva ser todo projetado novamente. Serão apresentados neste capítulo cada um dos blocos de hardware desenvolvidos e, em seguida, os resultados do sistema final.

6.1 Captura da Imagem

O sistema de hardware inicia o processamento de dados com a captura da imagem. Considerando a interface amigável da câmera *TRDB – D5M* o sistema de captura de imagens foi desenvolvido. O *netlist* do hardware é apresentado na Figura 26. O hardware mínimo de controle da câmera envolve a correta configuração dos pinos segundo o manual da Câmera que no caso do diagrama são 29 sendo que os outros 11 não são utilizados. O módulo de controle das memórias *SDRAM* com a respectiva taxa de *clock* controlada por um *Phase-Locked Loop* ou PLL para controle de frequência que no caso é de 160MHz.

Na configuração escolhida a câmera captura imagens em resolução VGA para obter maior taxa de captura operando próxima dos 70MHz necessitando também de uma PLL. O bloco de captura recebe o vetor de 12 pixels da câmera, a configuração para os valores de linha válida, quadro válido, início e termino da captura. Este bloco organiza os dados e repassa-os para o bloco que converte a imagem em RGB, além de incrementar a contagem de frames mostrada no display 7 segmentos do kit DE2-70. Os dados convertidos são enviados para o controlador *SDRAM* e então o quadro é armazenado em memória para utilização do classificador.

Considerando as implementações anteriores e suas limitações, optou-se por desenvolver um co-projeto de hardware/software utilizando o *soft-processor* Nios II para implementar o algoritmo *backpropagation*. O processador possui três versões diferentes e nenhuma delas apresenta o suporte à operações de ponto-flutuante, que são responsáveis por grande parte do tempo de execução do algoritmo. Foram configurados dez processadores diferentes baseados nas combinações entre as versões disponíveis do processador Nios II, e as possibilidades de inclusão de aceleradores de hardware como instruções customizadas. A arquitetura final dos processadores é apresentada pela figura 28.

Figura 28 – Arquitetura final dos processadores ((DIAS; OSORIO; WOLF, 2012)).

Tabela 3 – Resultados Finais ((DIAS; OSORIO; WOLF, 2012)).

Trabalho	Neuronios por Chip	CUPS	Camadas	Clock(MHz)
(DIAS; OSORIO; WOLF, 2012)	>100.000	1250	>7	100
(ISHII <i>et al.</i> , 1992)	10	-	3	-
(CLOUTIER; SIMARD, 1994)	100	-	-	-
(ELDREDGE; HUTCHINGS, 1994)	92	1800000	3	40
(GIRONÉS <i>et al.</i> , 2005)	-	77840	-	17
(MYERS; VINCENT; ORREY, 1993)	4	80000	>7	20
(SILVA; MOURELLE; NEDJAH, 2012)	99	-	7	30

Os resultados do melhor processador configurado, que incluía além da unidade de ponto flutuante suporte à memória SRAM e uma PLL para aumentar a frequência do *clock*, foram comparados com o estado-da-arte em treinamento de RNAs em hardware. Os resultados da comparação presentes na tabela 3 demonstram que para se obter escalabilidade é preciso sacrificar o tempo de execução e este trabalho, por se tratar de um comitê de redes neurais que possui requisitos de tempo-real e compõe um sistema embarcado crítico, deve priorizar o tempo de execução e também a qualidade do resultado. Para que estes dois requisitos sejam atendidos a melhor alternativa atualmente é o treinamento *offline*. Em resumo, o treinamento em hardware foi considerado inadequado pelas seguintes razões:

- O treinamento de uma RNA é um processo iterativo, que realiza um refinamento progressivo dos pesos (ajustes), e portanto não possui um "tempo fixo" para ser realizado, podendo ser bastante longo comprometendo requisitos de tempo-real do sistema.
- Uma vez treinada a RNA esta é colocada em produção, sendo que em um sistema de tempo real precisa apenas da RNA treinada para funcionar (o treinamento pode ser feito separadamente).
- Como descrito, o algoritmo de treinamento requer todo um complexo hardware adicional para diversas tarefas como: cálculo de erro, propagação de erro, ajuste de pesos, avaliação do treinamento e verificação do critério de parada. Este hardware de alta complexidade e custo seria utilizado apenas em uma pequena parte de todo o processo de classificação de imagens o que não justifica seu desenvolvimento.
- Caso adicionado à estrutura, o hardware que implementa o treinamento exige um grande acréscimo de circuitos ao FPGA (blocos lógicos), consumindo boa parte dos recursos disponíveis que poderiam ser utilizados para outras tarefas.
- Ainda que fosse implementado em hardware o treinamento teria que abrir mão de várias técnicas de simples implementação em software como validação cruzada, estimativa das melhores redes, diferentes critérios de parada.

6.2.2 Implementação em Ponto-Fixo

O classificador composto por um comitê de redes neurais implementado no trabalho de Shinzato ((SHINZATO; WOLF, 2011)) sempre utilizou a representação de ponto-flutuante de precisão simples e foi implementado utilizando a biblioteca FANN³⁰. Portanto, os testes com a implementação em ponto-fixe eram necessários para averiguar se a rede poderia obter os mesmos resultados caso a mudança de representação fosse necessária. Os testes foram feitos utilizando-se uma nova implementação do sistema e um grupo de dados diferente do grupo utilizado no artigo (já que na época dos testes os dados não foram disponibilizados). O trabalho desenvolvido ((DIAS; OSORIO, 2012)) utilizou a biblioteca do trabalho de Shinzato para gerar as redes em ponto-fixe. O classificador proposto Shinzato ((SHINZATO; WOLF, 2011)) é um comitê de redes neurais que utiliza a média dos valores de saída de cada rede e este não é o esquema de votação mais indicado para problemas de classificação (GIACINTO, 2001). Portanto foram testados outros esquemas de votação e os resultados de classificação da rede foram comparados.

Os primeiros resultados obtidos neste trabalho foram relativos à arquitetura das redes neurais do comitê. Diferentes funções de ativação foram testadas nas camadas escondidas, porém a função sigmoideal simétrica foi a que apresentou melhores resultados. O número de camadas escondidas também foi aumentado o que acarretou em um aumento no tempo de treinamento

³⁰ <http://leenissen.dk/fann/wp/>

e uma piora no erro quadrático da rede. Como os parâmetros de todos os elementos da rede não foram disponibilizados por Shinzato ((SHINZATO; WOLF, 2011)), o banco de dados gerado precisou de uma rede de 12 neurônios na camada escondida para poder atingir um erro quadrático de 0.023. Os esquemas de votação também foram testados com comitês de redes de cinco neurônios na camada escondida.

No caso de uma rede com 12 neurônios na camada escondida o número de acertos da rede como relação ao conjunto de validação ficou em 60%, o comitê votando com a média atingiu 45% de acertos e o comitê votando pelo método da maioria atingiu 65%. Após encontrar a melhor configuração o comitê foi então implementado em ponto-fixo. O número total de pesos do comitê é de 270 que, utilizando precisão de ponto-flutuante de precisão simples, necessitam de 8640 bits. Na biblioteca de ponto-fixo os mesmos números são armazenados em duas partes inteiras uma de 14 bits e outra de 8 bits. Somente pela conversão direta a mesma rede iria economizar 10 bits por peso armazenado.

Figura 29 – Resultado das redes neurais implementadas.

Os resultados dos primeiros testes com o ponto-fixo foram ruins devido a falta de normalização da entrada. Os valores de entrada devem ser normalizados entre -1 e 1 para que as redes tenham um melhor desempenho. Após a normalização o comitê de redes de ponto-fixo utilizando a votação pela maioria atingiu uma taxa de acertos de 93% que fica com diferença de apenas 3% para as mesmas redes implementadas em ponto-flutuante com precisão simples considerando o mesmo conjunto de dados.

Apesar do resultado de classificação apresentado no trabalho de Dias e Osório ((DIAS;

OSORIO, 2012)) não ter atingido um alto número de acertos como o esperado, testes feitos após a elaboração do trabalho demonstraram que utilizando-se um conjunto de treinamento de tamanho mais adequado é possível atingir uma taxa de acerto de classificação de padrões em torno de 93% em ponto-fixo, resultado que assemelha-se às taxas de acerto do trabalho de Shinzato, et. Al. ((SHINZATO; WOLF, 2011)) para a representação em ponto-flutuante. Este resultado confirma o estudo apresentado pelos desenvolvedores da biblioteca em seu relatório técnico³¹, onde é afirmado que as redes de ponto-fixo possuem o mesmo desempenho das redes de ponto-flutuante com precisão simples.

6.2.3 Classificador Neural

O classificador neural proposto no trabalho de Shinzato ((SHINZATO; WOLF, 2011)) é um comitê de redes neurais que possui cinco redes neurais com baixo número de neurônios que, ao serem implementadas utilizando cálculos de ponto fixo e a *look-up table* como função de ativação em hardware, resultaram em um design que pode ser utilizado no dispositivo escolhido. A seguir será apresentado o sistema de visão computacional desenvolvido bem como as estruturas de hardware de cada rede neural, o gerador de atributos e em seguida o gerador automático de LUTs desenvolvido para solucionar o problema da função de ativação não-linear utilizada neste trabalho.

6.2.3.1 Construção do sistema de visão computacional

Antes de iniciar a implementação do hardware da rede neural seria necessário verificar se, com um classificador mais simples, seria possível atingir uma taxa de pelo menos 30 quadros por segundo (que é aproximadamente a taxa de aquisição/distinção de imagens realizada pelo sistema de visão humano). Utilizando o hardware de captura de imagem desenvolvido, uma estrutura que permite a operação de convolução foi desenvolvida permitindo assim a implementação de vários filtros de imagem que proporcionam uma alternativa simples de implementação para o teste da estrutura.

A estrutura basicamente recebe os pixels capturados da câmera pelo hardware implementado anteriormente, realiza a operação utilizando os filtros de tamanho 3x3 utilizando as estruturas de multiplicadores-acumuladores e somadores paralelos. No caso específico dos testes realizados, o filtro escolhido foi o filtro de Sobel. Este filtro possui dois componentes básicos, um vertical (equação 5.1) e um horizontal (equação 5.2) que devem ser aplicados em conjunto e então utilizados para o cálculo do valor final do pixel (equação 5.6).

$$Vertical = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad (6.1)$$

³¹ <http://fann.sourceforge.net/report/>

$$Horizontal = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \quad (6.2)$$

$$Janela = \begin{pmatrix} p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 \\ p_7 & p_8 & p_9 \end{pmatrix} \quad (6.3)$$

$$X = (p_1 + (p_2 + p_2) + p_3 - p_7 - (p_8 + p_8) - p_9) \quad (6.4)$$

$$Y = (p_3 + (p_6 + p_6) + p_9 - p_1 - (p_4 + p_4) - p_7) \quad (6.5)$$

$$p_5 = \sqrt{X^2 + Y^2} \quad (6.6)$$

O hardware foi desenvolvido para executar estas operações. O diagrama do hardware é apresentado pela figura 30. Neste componente de hardware as entradas são o *clock*, os pixels necessários para o cálculo e, também, o valor de *threshold* para que o valor do pixel não ultrapasse o valor máximo da representação que é 255. A saída deste bloco é o vetor com os pixels processados.

Figura 30 – Diagrama de blocos e conexões do sistema de processamento de imagens gerado pelo Quartus II.

O hardware foi então sincronizado e o resultado foi direcionado para o display LCD que também é parte do kit multimídia da placa DE2-70. Inclusive o sinal de saída que acompanha o vetor de pixels modificado, e pode ser visto no diagrama da figura 30 é um sinal de controle do

Tabela 4 – Comparação entre o sistema de processamento de imagens desenvolvido e outros trabalhos da literatura. (Adaptado de (DIAS; OSÓRIO, 2015))

Trabalho	Clock MHz	FPS	Quadro	Hardware	Consumo de Hardware
(GHOLAMHOSSEINI H.; SHUYING, 2008)	-	-	64x64	Co-Design	High
(TOMASI <i>et al.</i> , 2010)	43	28	512x512	Co-Design	High
(HIRAIWA J.; AMANO, 2013)	30	-	1280x720	Hardware	Low
Sistema Desenvolvido	33	43	800x480	Hardware	Low

display. Considerando portanto o sistema de aquisição de imagens, o sistema de processamento que neste caso é o filtro de Sobel e a imagem ao final da execução tem-se, então, um fluxo de processamento de imagens. A tabela 4 apresenta uma comparação entre o resultado obtido pelo sistema de processamento de imagens e outros trabalhos que apresentam sistemas similares. Nesta tabela a informação do clock é apresentada em MHz e caso o sistema possua componentes em hardware e componentes em software é indicado como *co-design*. O consumo de hardware neste caso seria a porcentagem de consumo de elementos lógicos. Considera-se como um alto consumo de hardware, um valor acima de 80%. É importante ressaltar que existem diversos hardwares disponíveis para download na internet, e inclusive exemplos da fabricante, onde são implementados sistemas semelhantes.

6.2.3.2 Classificador Neural em Hardware

O sistema de geração do mapa de navegabilidade é composto por um gerador de atributos da imagem que envia para o classificador neural os valores dos atributos. Como investigado por Shinzato ((SHINZATO; WOLF, 2011)) os atributos que obtêm os melhores resultados são a média e a entropia, dentre outros possíveis atributos que podem ser usados na classificação. A câmera utilizada neste trabalho disponibiliza as imagens com canais canais R, G_1 , G_2 e B, sendo assim, os atributos que foram utilizados são a média de cada um dos canais e a entropia RGB (no caso com o componente G_2). Como proposto também no trabalho de Shinzato este sistema deve agrupar os pixels em janelas para que o processamento seja mais rápido. O hardware desenvolvido para o sistema de processamento de imagens anterior dispunha de uma janela de 3x3 pixels e atingiu um número de quadros por segundo que alcançou um nível de desempenho adequado aos propósitos deste trabalho, fazendo, portanto, que este tamanho fosse mantido.

Então para que fosse possível gerar o mapa de navegabilidade foram desenvolvidos dois módulos: o gerador de atributos e a rede neural propriamente dita. Para fins de sincronismo estes módulos serão controlados pelos mesmos PLLs utilizados tanto para a captura das imagens quanto para a execução do filtro de imagens. A ideia foi desenvolver o cálculo utilizando estruturas massivamente paralelas para que o caminho crítico suportasse o mesmo clock das outras estruturas permitindo assim o sincronismo. Os cálculos de média e entropia são descritos

pelas equações abaixo.

$$p(i)_c = C_c / (N * M) \quad (6.7)$$

$$M_c = p(i)_c \quad (6.8)$$

$$E_c = -p(i)_c * \log_2(p(i)_c) \quad (6.9)$$

A equação 5.7 mostra o cálculo da probabilidade de ocorrência de um determinado pixel ($p(i)$) que é constituída pelo número de pixels que possuem determinada cor (C) dividido pelo número de pixels da imagem ($N * M$). Este valor é utilizado então para o descritor de média (equação 5.8) e entropia (equação 5.9). O único problema para implementação destes descritores em hardware é o calculo do log cuja solução é a utilização de uma *look-up table* com valores pré calculados. Este procedimento tornou o cálculo mais rápido sem prejudicar o fluxo de processamento de imagens.

Figura 31 – Diagrama de blocos simplificado do gerador de atributos.

A figura 31 apresenta o diagrama de blocos do gerador de atributos. Inicialmente os pixels são enviados do conversor *RAW - RGB* para o gerador que recebe de entrada além dos vetores de pixels o *clock* do sistema. Cada vetor leva os pixels para um contador que retorna o valor $p(i)$ e depois executa a divisão pelo número de pixels do bloco, que neste caso são 9, e retornam o resultado de 12 bits em ponto fixo. Estes valores são a entrada para o bloco que calcula a entropia utilizando a *look-up table* para o logaritmo e retorna o valor da soma dos quatro valores de entropia (um de cada canal). Estas cinco saídas são utilizadas como entrada da rede neural implementada em hardware. É importante lembrar que os cálculos de ponto-fixo utilizados foram feitos por bibliotecas já testadas e desenvolvidas em linguagem

Verilog, e utilizadas em diversos projetos³¹. Os valores de saída são normalizados para serem corretamente utilizados pela rede neural. A figura 32 apresenta o diagrama de blocos e conexões do hardware gerado pelo Quartus II. Os dados de síntese da estrutura sem a integração com o sistema de processamento de imagens são: utilização de 180 elementos lógicos sendo 133 funções combinacionais e 119 registradores dedicados (valores que apresentam menos que 1% de consumo do hardware da placa) e o caminho crítico apresentou clock de 102.97 MHz. Estes dados de síntese foram extraídos do *compilation report* do software Quartus II considerando um projeto para o dispositivo deste trabalho.

Figura 32 – Diagrama de blocos e conexões do gerador de atributos gerado usando o Quartus II.

A rede neural foi projetada em hardware para explorar o paralelismo da melhor forma possível. A rede é uma MLP composta por cinco entradas, uma camada escondida de cinco neurônios e uma saída. Os cinco neurônios da camada escondida de cada uma das redes possuem função de ativação sigmoideal simétrica e o neurônio de saída possui função de ativação normalizada para 1 - navegável e 0 - não navegável. Os pesos de cada rede neural foram inseridos na estrutura após o treinamento offline. Toda a estrutura da rede neural implementada em hardware foi feita em ponto-fixado com precisão de 12 bits. A figura 33 apresenta um diagrama de blocos simplificado da rede neural.

A estrutura desenvolvida para a rede possui um bloco que controla os pesos e as entradas enviando para cada neurônio artificial os valores necessários para o cálculo. Os neurônios são de dois tipos básicos: (i) o neurônio escondido que recebe os valores dos pesos e também de cada entrada correspondente e realiza a operação da soma ponderada. Após o cálculo, o valor de saída é utilizado como endereço da LUT de ativação e o valor é encaminhado como entrada para o neurônio de saída; (ii) e também existe o neurônio de saída que recebe além das entradas e pesos os sinais de controle dos blocos anteriores e, após o cálculo e a saída da função de ativação *threshold*, já disponibiliza na saída os vetores de pixels com os valores modificados

³¹ http://opencores.org/project,verilog_fixed_point_math_library

Figura 33 – Diagrama de blocos simplificado da rede neural.

para 255 navegável e 0 não-navegável. Os dados de síntese da rede neural sem integração com o sistema de processamento de imagens são: utilização de 228 elementos lógicos sendo 180 funções combinacionais e 150 registradores dedicados (valores que apresentam menos que 1% de consumo do hardware da placa) e o caminho crítico apresentou clock de 312.70 MHz. Estes dados de síntese foram extraídos do *compilation report* do software Quartus II considerando um projeto para o dispositivo deste trabalho.

A rede neural desenvolvida pode ser classificada segundo as regras mostradas na seção 3.6.1 deste documento de acordo com as informações disponíveis. As medidas relativas ao consumo de energia não puderam ser calculadas pois não foram realizadas medições relativas ao consumo de energia. O resultado da classificação é apresentado na tabela 5. A área efetiva de sinapses foi calculada com relação ao número de elementos lógicos por se tratar de um hardware reconfigurável.

Tabela 5 – Classificação da rede desenvolvida.

Métrica	Símbolo	Rede deste trabalho
Neurônios por Chip	N_n	6
Sinapses por Chip	N_s	30
Pesos por Chip	N_w	30
Área Efetiva de Sinapses	ESA	7.6
Conexões por Segundo	CPS	44.6×10^6
CPS por peso	CPSW	14.8×10^6
Primitivas de CPS	CPPS	3.131×10^9

A função de ativação da rede neural implementada no trabalho de Shinzato apresenta um neurônio de saída linear, ou seja, os valores são ajustados entre totalmente não-navegável 0 e totalmente navegável 255 linearmente. Este trabalho apresenta uma função *threshold* onde

os valores são binários, ou seja, ou o bloco é navegável ou não é. Embora exista a diferença de implementação nas funções, é possível modificar o hardware para que realize a função de ativação linear de diferentes maneiras, dentre elas: a inclusão de um conjunto de comparadores paralelos (ao invés de apenas um, como é o caso da função *threshold*) o que tornaria a estrutura rápida porém exigiria um barramento maior; ou mesmo a inclusão de uma LUT cuja influência em quantidade de hardware e tempo de execução dependeria de seu projeto. Utilizando a saída linear é possível indicar o grau de certeza da rede em relação a classificação dos pixels adicionando informações ao mapa de navegabilidade. Caso necessário, seria possível inclusive adicionar a saída linear ao hardware e obter dois resultados diferentes. A possibilidade de se realizar modificações nas estruturas desenvolvidas de forma simples, sem a necessidade de se reprojeter toda a estrutura, é uma característica importante deste trabalho.

Figura 34 – Diagrama de blocos e conexões da rede neural artificial gerado usando o Quartus II.

A implementação de uma rede neural em hardware é um desafio, porém a maioria dos problemas ocorrem no projeto do hardware que calcula a função de ativação quando esta função é não-linear. Este trabalho utiliza funções de ativação não-linear sigmoidal simétrica na saída dos neurônios da camada escondida e, de acordo com trabalhos de redes neurais em hardware, calcular o valor real da função em hardware consome recursos e principalmente tempo de execução. A solução mais comum é a utilização de uma *look-up table* com valores de

Figura 35 – Gerador automático de LUTs

aproximação, porém os trabalhos que utilizam este tipo de abordagem não descrevem a tabela, não descrevem seu processo de desenvolvimento e tratam o assunto como óbvio. A solução para o problema, neste trabalho, foi o desenvolvimento de um gerador automático de tabelas. O diagrama *netlist* gerado pelo Quartus II pode ser visto na figura 34. Cada neurônio da camada escondida possui sua LUT que foi gerada pelo gerador automático de LUTs desenvolvido.

O objetivo principal do gerador automático de tabelas é otimizar o resultado de uma *look-up table* utilizada como função de ativação de uma rede neural artificial. Diversos problemas que utilizam RNAs como classificadores possuem requisitos de tempo-real demandando um nível alto de otimização no tempo de execução sem uma grande perda de precisão. O problema das LUTs é que o número de entradas, ou seja, a quantidade de valores que são usados para discretizar a função não-linear, influencia diretamente no tamanho final da tabela (o que pode ser um problema para a implementação em hardware) e também no tempo de execução.

O funcionamento do gerador automático de LUTs é detalhado na Figura 35. A primeira fase do gerador automático gera diferentes tabelas com diferentes números de entradas e passos (intervalos entre valores) e testa estas tabelas com relação ao tamanho e a precisão final das respostas da rede. Caso as tabelas geradas automaticamente a partir dos parâmetros de entrada não atinjam o tamanho e erro desejados outro passo de otimização é iniciado. Os testes são feitos passando os dados do banco de dados de teste por duas redes, uma que utiliza o cálculo original da função e outra que utiliza as tabelas geradas. Caso o erro de resposta entre a tabela e a função

esteja muito alto para uma tabela de tamanho aceitável é necessário otimizar ainda mais a tabela. A segunda fase de otimização utiliza a melhor tabela encontrada na fase anterior como função de ativação da rede neural e identifica quais as entradas da tabela são mais requisitadas e quais são menos requisitadas passando todo o banco de dados pela rede que utiliza a tabela como função de ativação. As entradas mais requisitadas são então expandidas e as entradas menos requisitadas são comprimidas em intervalos maiores de outras entradas. Este procedimento permite que a precisão de resposta da tabela seja melhorada sem comprometer o tamanho da tabela. A figura 35 apresenta o funcionamento do sistema.

O sistema de geração automática apresenta como parâmetros de entrada: (i) o erro máximo de saída que o usuário deseja obter utilizando a tabela; (ii) o número máximo de entradas da tabela; (iii) o número mínimo de entradas da tabela; (iv) o intervalo máximo entre duas entradas da tabela; (v) número máximo de bits da precisão de cada entrada; (vi) número mínimo de bits da precisão de cada entrada e (vii) o valor do passo de precisão para a geração das tabelas. Como o gerador considera redes MLP, todas as características da rede neural a ser utilizada devem ser apresentadas juntamente com o número máximo de passos de iteração para a análise das tabelas geradas e também para a segunda fase da geração automática. O banco de dados de treinamento também deve ser disponibilizado. Como resultado após as duas fases o método apresenta uma biblioteca em linguagem C contendo a tabela gerada e também um arquivo em VHDL contendo o hardware para utilização da tabela. O sistema foi desenvolvido e testado em duas situações de avaliação clássicas (rede neural para o aprendizado do problema XOR com um espaço de entradas com valores contínuos e rede da função $\sin(x)$) e também em dois sistemas robóticos complexos sendo um de navegação topológica e outro de auto-localização usando a força do sinal WiFi apresentando resultados satisfatórios. Maiores informações sobre cada um dos testes e detalhamento sobre os resultados destes sistemas podem ser encontradas no trabalho de Dias, Sales e Osório (DIAS; SALES; OSORIO, 2016).

A rede deste trabalho de pesquisa utiliza a função de ativação sigmoide simétrica nos 5 neurônios escondidos de cada rede que compõe o comitê. A geração das tabelas foi feita utilizando o sistema desenvolvido e foram executados os dois passos para garantir que o sistema atingisse o melhor nível de otimização possível. Na primeira etapa foram geradas aproximadamente 60 tabelas com diferentes configurações de acordo os parâmetros de entrada de erro máximo de 0.02, tamanho máximo de 6Kb, intervalo máximo entre duas entradas da tabela de 1.0, número máximo de precisão de bits 12, número mínimo também em 12 (para ficar de acordo com a arquitetura do sistema), e o valor do passo de precisão 0,1 que em concordância com os outros parâmetros resultou no número de aproximadamente 60 tabelas. Os resultados, mostrados graficamente na figura 36, sugerem que já existem 4 tabelas que respeitam os requisitos de tamanho e erro, porém como o objetivo é a otimização, a melhor tabela foi escolhida para ser otimizada.

O mapeamento dos acessos foi feito e o gráfico que indica a frequência de acesso de cada

Figura 36 – Resultado da primeira fase do algoritmo de geração de LUTs para o sistema de processamento de imagens em hardware.

Figura 37 – Número de acessos a cada entrada da tabela escolhida para otimização.

entrada é apresentado na figura 37. É possível identificar alguns pontos onde o segundo passo de otimização pode melhorar o resultado da rede. As entradas escolhidas para serem agrupadas a outras entradas próximas foram simplesmente as que apresentavam valores baixos de frequência, escolhidas de forma randômica já que não existe nenhuma entrada que apresenta frequência muito abaixo das outras. O tamanho da tabela após o segundo passo permanece o mesmo de aproximadamente 3,5Kb, mas o erro foi diminuído de 0.0175 para 0.0069 o que melhorou o desempenho da rede. Os resultados para este trabalho são comparados com os resultados dos outros sistemas robóticos para a mesma função de ativação na tabela 6 e indicam que este sistema obteve um bom nível de otimização.

O gerador automático de LUTs desenvolvido para este trabalho solucionou o problema

Tabela 6 – Comparação de tabelas para sistemas robóticos

	Sistema de navegação topológica	Sistema de localização WIFI	Hardware deste trabalho
Tipo de Rede	MLP	MLP	MLP
Arquitetura	8-20-20-2	36-10-7	5-5-1
Tamanho Final LUT	163,458 Kbytes	2,38 Kbytes	3,5 Kbytes
Erro Final LUT	0.0985	1.8067	0.0069
Tempo de Geração	6h	5h30	3h
Comportamento da Frequência	Concentrado	Esparso	Esparso

das LUTs que foram utilizadas no sistema de processamento de imagens em hardware. Este era o objetivo principal quando a ferramenta foi desenvolvida, porém o sistema apresenta uma solução para o problema da geração de tabelas de um modo mais geral podendo ser aplicado a diversos tipos de problemas que necessitem de otimizações, como por exemplo os sistemas robóticos que utilizam RNAs para diferentes situações. Da forma como foi desenvolvido o sistema é capaz de gerar LUTs para qualquer sistema que utilize uma rede neural MLP desde que a arquitetura e os pesos da rede estejam disponíveis, bem como o conjunto de treinamento. O sistema pode ainda ser modificado e gerar tabelas para qualquer rede neural, bastando que o tipo da rede esteja implementado no sistema.

Como apresentado anteriormente o classificador neural deste trabalho é composto pelo gerador de atributos da imagem e pela rede neural implementada em hardware. Após o desenvolvimento das estruturas, elas foram conectadas e sincronizadas ao sistema de aquisição e processamento de imagens, desenvolvido inicialmente apenas com o filtro de imagens, e agora sendo substituído pela RNA que gera o mapa de navegabilidade. A seção seguinte apresenta o sistema final e os resultados.

6.3 Sistema de Hardware para Classificação

As partes do sistema e seu desenvolvimento foram apresentadas nas seções anteriores e todas as informações obtidas durante o desenvolvimento foram empregadas no desenvolvimento do sistema final de hardware. O hardware final foi, portanto, desenvolvido apresentando uma alternativa em hardware ao sistema desenvolvido por Shinzato ((SHINZATO; WOLF, 2011)) que é totalmente implementado em software.

Após o desenvolvimento das estruturas necessárias para a classificação da imagem em hardware alguns testes foram feitos com relação às variáveis possíveis de serem modificadas neste trabalho. As possibilidades são a mudança do número de bits e o seu impacto no consumo de hardware, bem como no caminho crítico de cada componente e também o tamanho da janela considerada para os cálculos. O número de bits impacta na precisão do sistema como um todo que, considerando os resultados apresentados em seções anteriores, deve ser de pelo menos 12

bits. Portanto foram testados os impactos de estruturas de 32 bits até 12 bits. A implementação de ponto fixo utilizou a notação $Q.N$ descrita no capítulo 2 onde a representação é de forma normalizada e, para este trabalho, $Q = 12$.

Figura 38 – Estudo de precisão para o gerador de atributos.

O estudo de precisão foi feito para avaliar quais os impactos de se utilizar precisões maiores no tamanho do hardware e no clock do caminho crítico. É importante lembrar que o caminho crítico depende da solução da etapa de *place and route* da síntese do circuito e que os resultados serão sempre diferentes já que os algoritmos são heurísticos. Entretanto, considerando um alto número de sínteses e testes para cada hardware foi feita uma média desprezando-se *outliers* e o desvio padrão se manteve em aproximadamente 10 MHz. O número de elementos lógicos também é uma média, porém seu desvio padrão seria de 10 a 15 elementos pois este resultado pouco se altera nas otimizações. Outra questão importante é que existem níveis de otimizações para síntese do Quartus II que foram mantidos na configuração padrão para que a divergência de resultados fosse minimizada (no sentido que os algoritmos de otimização iriam produzir resultados com desvio padrão consideravelmente alto).

É possível notar em ambos os gráficos (figuras 38 e 39) que há um aumento considerável no número de elementos lógicos e também uma diminuição considerável do clock devido a um aumento do caminho crítico. Quanto menor o número de bits mais rápida a estrutura e menor o custo (em termos de elementos lógicos) da implementação. Os algoritmos de otimização de *Place and Routing*, que posicionam as estruturas de hardware no dispositivo e configuram os barramentos entre os dispositivos, são heurísticos. Devido a este fato, os resultados para as estruturas mais complexas podem atingir valores de *clock* máximo maiores por uma diminuição do caminho crítico da estrutura resultante da otimização do algoritmo. Este fato justifica a

Figura 39 – Estudo de precisão para a rede neural em hardware.

necessidade de se fazer os cálculos estatísticos nos resultados e não exclui a possibilidade de que, em determinada síntese, caminhos críticos mais curtos sejam encontrados.

O sistema de hardware implementado possui, portanto, 5 redes neurais formando um comitê que possui votação por maioria que é o método mais indicado para o sistema em questão (GIACINTO, 2001). Uma das principais motivações para a criação de comitês de redes neurais foi o fato de que a seleção dos pesos das RNAs é um problema de otimização com um grande número de mínimos locais (ZHOU; WU; TANG, 2002). O comitê gerado pode ser classificado como um comitê homogêneo (considerando que as redes possuem a mesma arquitetura) com *data agreement*, que significa que o comitê utiliza os mesmos dados para treinamento de todas as redes. A princípio pode parecer que a utilização dos mesmos dados para treinamento das redes faz com que não haja diversidade suficiente, porém o conjunto de pesos gerados para cada rede é diferente promovendo uma análise diferente das entradas. Shinzato define 3 grupos de atributos em seu trabalho sendo os que apresentaram melhor resultado e utiliza redes treinadas com estes grupos, porém não há informação de testes realizados com comitês de *data agreement* ou mesmo dados comparativos indicando que apenas comitês heterogêneos podem atingir resultados satisfatórios.

A imagem então após captada pela câmera passa pelo gerador de atributos que gera os valores para a rede neural. Em seguida os valores são classificados por cada rede e a saída que está presente na maioria das redes é escolhida como saída do sistema para o bloco.

³² ftp://ftp.altera.com/up/pub/Altera_Material/11.0/University_Program_IP_Cores/Memory/SD_Card_Interface_for_SoPC_Builder.pdf

³³ https://www.usna.edu/EE/ec463/notes/09_EC463_SD_Card_student.pdf

Os pesos armazenados em hardware para o comitê de redes neurais foram obtidos com treinamento offline como definido anteriormente e indicado na figura 21. Para que os pesos representem o conhecimento em relação ao conjunto de dados que será classificado no hardware é necessário criar um banco de testes baseado nas imagens que serão captadas no FPGA. A imagem é armazenada em bits que representam a intensidade dos pixels na memória SDRAM configurada para armazenar a imagem. Existem projetos de hardware disponíveis em diversos sites e fóruns da internet da utilização do cartão SD do kit DE2-70. A solução encontrada para armazenar os dados da SDRAM no cartão utiliza uma SPI *Serial Peripheral Interface* implementada em um bloco de hardware³². Após salvar a imagem no cartão SD, os valores foram carregados em um arquivo texto³³ e armazenados em uma matriz em openCV que depois foi convertida para *bitmap*. O *bitmap* então era mostrado na tela e a seleção dos blocos navegáveis e não-navegáveis era feita visualmente. A entrada para o treinamento *offline* foi um conjunto de pixels de uma imagem totalmente classificados, ou seja, 307.200 pares entrada-saída de dados para que fosse feito o *10-fold cross-validation*. Como esperado pela natureza da imagem de uma via, o número de locais não-navegáveis era maior que o de locais navegáveis gerando uma divisão de, aproximadamente, 35% dos pares de regiões navegáveis e 65% dos pares de regiões não-navegáveis.

O algoritmo de treinamento utilizado foi o algoritmo Rprop da biblioteca FANN³⁴. O treinamento foi configurado para obter os dados de um arquivo utilizando o tipo de treinamento *train_on_file* com número máximo de épocas 30.000, e o erro desejado de 0.001. Estes parâmetros devem ser configurados na chamada da função de treinamento, porém dentro da função é possível utilizar apenas um como critério de parada que, no caso da rede, foi o erro desejado. Este algoritmo possui os parâmetros δ_{zero} , δ_{min} , δ_{max} , *increase factor*, *decrease factor* que foram escolhidos, respectivamente, 0.1, 0.5, 5, 1 e 50. Os treinamentos foram interrompidos quando o erro quadrático das redes atingia um valor abaixo de 0.001. O método de teste e validação utilizado foi o *10-fold cross validation*. As redes escolhidas para o hardware foram redes de índices de acerto, em relação ao conjunto de teste, de 87%, 90%, 95%, 85% e 93% e MSE de 0.00083, 0.00099, 0.00094, 0.00078, e 0.00088. As redes em software também foram escolhidas de forma que as diferenças entre modelos fossem minimizadas, ou seja, as redes treinadas que obtiveram os valores de erro mais próximos aos erros obtidos para a rede em hardware forem escolhidas.

As mesmas redes escolhidas para a inclusão no hardware foram utilizadas pelo sistema de geração automática de LUTs para gerar a tabela que foi utilizada pelos neurônios das camadas escondidas. Para cada rede uma tabela diferente foi criada e o erro do resultado para cada tabela foi 0.00851, 0.00673, 0.00994, 0.00720, e 0.00907. A frequência de acessos às entradas não foi concentrada em alguns valores de modo que a otimização foi restrita. O número de entradas de cada tabela foi de aproximadamente 300 sendo de tamanho menor que 1Kb. Este valor só foi

³⁴ <http://leenissen.dk/fann/wp/>

possível porque as entradas da tabela representavam em sua maioria médias de valores de 0 a 255.

O comitê de redes neurais, cujo diagrama pode ser visto na figura 40, apresenta as cinco redes neurais implementadas e suas saídas alimentando um sistema de votação. Como os vetores de saída de cada rede são todos iguais, para que a cor do bloco seja preta ou branca, apenas um deles é necessário para fazer a comparação e gerar os vetores de resultado. O bloco sai pronto do sistema de votação para compor a imagem que será exibida no display. A compilação do comitê com otimizações atingiu um clock máximo de 304.46 MHz em seu caminho crítico tendo ocupado 940 elementos lógicos o que não representa nem 1% dos elementos do FPGA.

Figura 40 – Diagrama de blocos e conexões do comitê de redes neurais gerado pelo Quartus II.

Utilizando o hardware do comitê de redes neurais, juntamente com o gerador de atributos, foi possível compor o hardware do sistema de classificação utilizando o sistema de processamento de imagens criado anteriormente (figura 41). O hardware final ocupou em sua totalidade aproximadamente 10% de todos os elementos lógicos do FPGA o equivalente a 6468. Como existem diferentes tipos de *clock* no sistema e foi utilizado para as estruturas um dos *clocks* de uma PLL que teve mudança em seu caminho crítico atingindo 104.35 MHz. Após a sincronização os requisitos de tempo mínimos do sistema tiveram que ser alterados e vários ajustes com buffers

e modificações no sistema foram realizadas. A estrutura é a mesma, porém o hardware necessita de mais ciclos de clock para fazer os cálculos referentes à rede.

A sincronização da estrutura e os ajustes para o funcionamento do sistema são tarefas complexas que demandaram muitos testes e muitos ajustes finos na estrutura. Os experimentos com o hardware são complexos pois o FPGA é alimentado com uma fonte e o programa precisa ser baixado toda vez que o FPGA é iniciado. A montagem da estrutura para execução do sistema em um veículo foi então realizada e foi possível avaliar a qualidade da classificação. Os testes foram executados com imagens da mesma cena com a seguinte configuração: um vídeo foi feito do percurso e simultaneamente o FPGA, configurado com o hardware deste trabalho, foi classificando a imagem estando posicionado ao lado da câmera que capturou as imagens. Desta forma o vídeo pôde ser utilizado como vídeo para as redes classificadoras e os testes de software. Para a geração dos pesos do comitê em hardware as redes foram treinadas com o conjunto de treinamento baseado nos valores de imagens capturadas pelo hardware. No caso do software as redes utilizaram os vídeos gerados pela câmera que estava ao lado do FPGA no veículo.

Os testes foram realizados com diversas imagens originadas do vídeo e um conjunto de treinamento foi organizado. A comparação das imagens (figura 42) apresentadas é composta por cinco colunas de imagens. A coluna inicial é a imagem da câmera captada, a segunda coluna representa a classificação pelo comitê de ponto-flutuante seguida pela coluna da pior rede em ponto-fixa e, por fim, a quarta coluna apresenta a classificação do comitê em ponto-fixa seguida pela classificação do hardware. A imagem da classificação do hardware teve as cores modificadas, para preto e branco, devido ao fato de que o display LCD deixa as imagens azuladas e difíceis de serem visualizadas.

A figura 42 apresenta a classificação para casos complexos e casos mais simples. A primeira linha apresenta a classificação logo onde iniciou-se o processamento sendo, portanto, o local onde há menor variação de brilho, contraste e cores com relação à imagem utilizada para treinamento. Os problemas começam a ocorrer com regiões de sombra (por ser um final de tarde) onde o classificador apresenta uma classificação com blocos esparsos. A terceira linha, apesar da incidência de luz indireta conservou as características da pista que pôde ser identificada. Na quarta linha o sol foi de encontro aos sensores ópticos fazendo com que apenas parte da imagem fosse classificada corretamente. A quinta linha é outro exemplo de local onde a parte da rodovia conserva suas características apesar das mudanças no restante da imagem. A sexta linha apresenta uma imagem onde o sol muda a cor da pista que fica mais laranja e confunde consideravelmente o classificador. Por fim a última linha apresenta o retorno ao ponto de partida onde a classificação foi melhor. Apesar das diferenças o classificador de hardware se mostrou robusto em diversas situações acompanhando o resultado dos classificadores de software. As imagens utilizadas para os testes da figura 42 foram armazenadas no cartão SD, bem como as imagens classificadas.

Utilizando estes dados foi possível comparar os blocos classificados pelas redes e gerar

Figura 41 – Diagrama de blocos e conexões do classificador neural gerado pelo Quartus II. Módulos do classificador neural em destaque.

Figura 42 – Comparações entre resultados de classificação dos classificadores utilizados neste trabalho.

os dados de classificação da tabela 7. Os valores da tabela representam a porcentagem de acerto considerando como *benchmark* a classificação realizada pela rede implementada em software com ponto-flutuante. Por exemplo, a pior rede de ponto-fixo acertou 78,25% em relação à rede de ponto-flutuante que acertou 98,79% no mesmo caso. A ideia de considerar a rede de ponto-flutuante como referência é baseada no fato de que o sistema de hardware deveria ter resultados melhores ou ao menos semelhantes à rede de ponto-flutuante escolhida como base para este trabalho.

Após a realização dos testes com a estrutura foi possível verificar que o sistema está executando a uma taxa de 25fps. O sistema para executar em tempo real para esta aplicação deveria ter atingido ao menos 30fps, porém este resultado mostrou um avanço considerável

Tabela 7 – Comparação entre a classificação das imagens apresentadas na figura 42.

Linha	Pior rede Ponto-Fixo	Comitê Ponto-Fixo	Hardware deste trabalho
Linha 1	78,25%	97,15%	95,98%
Linha 2	68,63%	90,87%	88,01%
Linha 3	82,44%	94,56%	92,33%
Linha 4	65,07%	92,19%	89,98%
Linha 5	79,81%	91,88%	92,03%
Linha 6	67,22%	87,34%	83,03%
Linha 7	80,17%	96,32%	95,44%

em comparação com o classificador implementado em software pois a imagem que atingia no máximo 30fps no sistema proposto por Shinzato era de tamanho 320x240 enquanto a imagem trabalhada no sistema de hardware é de tamanho VGA (640x480). Além disso, o sistema está trabalhando com blocos de 9 pixels (3x3) enquanto o sistema anterior trabalhava com blocos de 10x10 pixels resultando em uma menor "granularidade" da classificação dos pixels da imagem.. Testes demonstraram que na mesma condição o sistema de software reproduzido para fins de teste neste trabalho apresenta taxas de 15fps a 20fps. Dois outros tamanhos de janela foram testados no sistema de hardware 5x5 e 10x10. Com o número maior de pixels analisados a velocidade do sistema aumentou, porém a qualidade das classificações também piorou. Nos blocos de 5x5 o sistema conseguiu uma resposta de 32fps enquanto nas janelas 10x10 foram atingidos 45fps. Os gargalos do sistema, portanto, podem ser identificados como a aquisição da imagem que é limitada pelo clock máximo do fabricante e, também, os cálculos em ponto-fixos que atingiram o resultado fixados em 12 bits. O aumento do número de bits pode resultar na diminuição do tempo de resposta, sendo necessário o desenvolvimento de uma estrutura com maior nível de paralelismo para acomodar um hardware mais preciso. Em relação a tolerância a falhas, o sistema ficou ligado em funcionamento por uma semana e as imagens continuaram a ser classificadas. A tabela 8 apresenta um resumo dos resultados dos 6 classificadores desenvolvidos para este trabalho com relação a plataforma de desenvolvimento, a precisão do sistema relativa a representação escolhida, o número de quadros por segundo, o maior erro encontrado durante o desenvolvimento e o erro médio relativo a todos os testes realizados com cada estrutura.

O kit escolhido para o desenvolvimento, De2-70, se mostrou robusto e permitiu que o sistema fosse desenvolvido como esperado. A questão de ser um dispositivo de capacidade média se refletiu nos resultados do número de quadros por segundo, resultado de um clock disponível limitante. As PLLs tornam possível o aumento do clock, porém funcionam com base no clock disponível pela placa que neste caso é de 50 MHz. Sobre o consumo de energia, uma comparação do consumo médio de um processador Intel i3 ou i5 que fica em média entre 70W e 100W em regime de trabalho³⁶ e do consumo do FPGA embarcado no kit DE2-70 que é em média de 4W a 6W mostra que a utilização do FPGA apresenta uma redução significativa do consumo de energia, ainda que o consumo do sistema deste trabalho não tenha sido aferido.

Tabela 8 – Comparação entre os sistemas desenvolvidos neste trabalho.

Sistema	Plataforma	Precisão	FPS	Erro Max	Erro Medio	Tam. Imagem
Rede 1 - Comitê	Software	Float	30	5%	3.7%	320x240
Rede 2 - RNA	Software	12-bit P. Fixo	30	12%	6.4%	320x240
Rede 3 - Comitê	Software	12-bit P. Fixo	30	8%	5.3%	320x240
Rede 4 - Comitê Janela 3x3	Hardware	12-bit P. Fixo	25	12%	6.9%	640x480
Rede 5 - Comitê Janela 5x5	Hardware	12-bit P. Fixo	32	16%	10.3%	640x480
Rede 6 - Comitê Janela 10x10	Hardware	12-bit P. Fixo	45	21%	17.5%	640x480

A utilização de um FPGA *high-end*, como por exemplo a linha Stratix da fabricante Altera, tornaria possível implementações mais rápidas, pois existe um número maior de hardwares especializados disponíveis (como, por exemplo, multiplicadores) e também a possibilidade de utilização de *clocks* reais mais rápidos (por exemplo, o kit de desenvolvimento Stratix V GX³⁵ que possui o preço de 6.995 dólares apresenta uma melhora em todos os requisitos em comparação com o kit De2-70, inclusive osciladores de 50, 100, 125 e 148 MHz). Portanto com um dispositivo melhor, melhores resultados seriam obtidos com relação ao número de quadros por segundo e consumo de hardware. No caso dos dispositivos Stratix é possível também realizar medições de energia, permitindo assim a quantificação e, consequentemente, a otimização do consumo. Outra vantagem de um FPGA melhor seria a possibilidade de replicar a estrutura e ter a análise de blocos feita paralelamente (com, por exemplo, a implementação de 10 comitês paralelos realizando a classificação simultaneamente).

6.4 Considerações Finais

Este capítulo apresentou os resultados deste trabalho de pesquisa e também uma análise de cada etapa do desenvolvimento da estrutura de hardware. A questão do treinamento em hardware foi revista porém o resultado não difere de pesquisas anteriores, o ponto-fixe foi explorado de forma consistente mostrando que é possível sua utilização. A implementação em hardware do gerador de atributos e da rede neural foram apresentadas com seus respectivos desafios e limitações. O sistema desenvolvido neste trabalho também foi apresentado e analisado. O capítulo seguinte apresenta as conclusões deste trabalho de pesquisa.

³⁵ https://www.altera.com/products/boards_and_kits/dev-kits/altera/kit-sv-gx-host.tablet.html

³⁶ http://www.xbitlabs.com/articles/cpu/display/core-i5-4670s-4670t_7.html

CONCLUSÃO

O presente trabalho apresentou o desenvolvimento de um hardware para classificação de imagens que utiliza um comitê de redes neurais artificiais implementadas utilizando representação de ponto-fixo em hardware reconfigurável. O hardware desenvolvido apresentou uma alternativa viável ao sistema previamente desenvolvido totalmente implementado em software por Shinzato.

Durante o desenvolvimento um co-projeto de hardware/software foi desenvolvido como alternativa para a implementação do algoritmo backpropagation em hardware. Os resultados obtidos demonstraram que a estrutura é mais eficiente em quase todos os quesitos presentes nos resultados de trabalhos relacionados, porém mesmo assim não é viável utilizar o treinamento em hardware. A solução para este problema talvez envolva a utilização de uma estrutura baseada em co-projeto com um *hard-processor* fazendo o processamento e não um soft-processor como o processador Nios II.

O estudo sobre a implementação de redes neurais utilizando a representação de ponto fixo demonstrou a viabilidade de se construir redes neurais utilizando esta representação. Os resultados iniciais não demonstraram a robustez esperada, porém com estudos posteriores foram encontradas soluções para a melhoria dos resultados alcançados, como por exemplo a utilização da técnica de normalização das entradas e os ajustes dos parâmetros do algoritmo de treinamento após a execução de testes. Durante o desenvolvimento da rede neural em hardware que iria formar o comitê de classificação estudos foram realizados visando melhorar a qualidade dos resultados, o que incluiu melhorias em relação a votação na obtenção da resposta do comitê de RNAs, onde por fim, os problemas da diferença de qualidade nos resultados em relação aos trabalhos originais foram solucionados. A utilização de estruturas de ponto-fixo permitiu que todo o trabalho fosse implementado no mesmo FPGA consumindo um número aceitável de elementos lógicos. Considerando que o sistema pode ser produzido para ser embarcado em um veículo, a diminuição de elementos lógicos se torna um objetivo importante relacionado ao custo final do sistema.

Outro avanço no estado-da-arte do desenvolvimento de redes neurais artificiais em hardware foi o desenvolvimento do gerador automático de *look-up tables*. As tabelas são amplamente utilizadas como função de ativação, porém muito pouco estudadas e detalhadas nas implementações de RNAs em Hardware (HNNs). O método proposto leva em conta o problema e a própria rede neural que irá utilizar a tabela fazendo que com o resultado seja especializado, obtendo assim um menor erro de resposta da rede neural em hardware.

O hardware desenvolvido integra o cenário de desenvolvimento de hardware para veículos autônomos, uma área em expansão nos últimos anos. Considerando que este projeto de pesquisa iniciou-se há cinco anos, o resultado é relevante tanto como sistema quanto como alternativa já que de acordo com a pesquisa realizada nenhum dos veículos autônomos em destaque atualmente utiliza uma plataforma de hardware reconfigurável.

As melhorias com relação ao sistema implementado em software por Shinzato são: o tamanho da imagem analisada que passa de 320x240 para 640x480 podendo ainda ser ampliada já que a câmera do sistema captura imagens de até 5MP; o número de pixels do bloco analisado diminui de 100 para 9 resultando em uma qualidade de classificação melhor para a imagem, principalmente com relação aos detalhes; o consumo de energia não foi medido pela dificuldade de se medir o consumo do FPGA que está integrado em um kit de desenvolvimento e não possui uma estrutura interna para esta operação, porém a comparação do sistema atual desenvolvido em um chip e o sistema anterior que era um software executado em um computador pessoal indica uma redução no consumo; o sistema por se tratar apenas de um kit de desenvolvimento com uma câmera ocupa um espaço físico menor podendo ser mais facilmente embarcado em um veículo.

Em resumo as principais contribuições deste trabalho são:

- O desenvolvimento de um co-projeto de hardware/software que executa o treinamento da rede neural em hardware melhorando os resultados apresentados em trabalhos anteriores e que confirma a necessidade do treinamento *offline* para sistemas de tempo-real.
- O estudo sobre implementação de redes neurais artificiais utilizando a representação numérica de ponto-fixado, assunto que ainda apresenta espaço para análises e otimizações.
- O projeto e desenvolvimento de uma estrutura de hardware que realiza processamento da imagens baseado em um classificador neural e apresenta potencial para atingir requisitos de tempo real considerando-se os 30 fps baseados na visão humana (ainda que estes requisitos não estejam totalmente definidos para veículos autônomos).
- O desenvolvimento de um gerador automáticos de LUTs baseado em redes MLP e a otimização pela expansão de partes da tabela mais utilizadas e agrupamento de partes menos utilizadas. Este trabalho apresentou avanços no estado-da-arte de redes neurais artificiais em hardware por ser uma ferramenta que resolve um problema comum das implementações de forma automática.

7.1 Publicações

Os artigos publicados com os resultados deste trabalho de pesquisa estão listados a seguir:

DIAS, M. A. ; OSORIO, F. S. . Fixed-Point Neural Network Ensembles for Visual Navigation. In: 2012 Brazilian Robotics Symposium and IEEE Latin American Robotics Symposium, 2012, Fortaleza - CE. Proceedings of SBR/LARS 2012, 2012. v. 1. p. 307-3012.

DIAS, M. A. ; OSORIO, F. S. . RNAs em Hardware para Navegação de Veículos Autônomos Baseada em Visão. In: I WTDR - Workshop de Teses e Dissertações de Robótica, 2012, Fortaleza CE. SBR/LARS 2012, 2012. p. 1-10.

DIAS, M. A. ; OSORIO, F. S. ; WOLF, D. F. . Harwdware/Software Co-design Implementation of On-Chip Backpropagation. In: Brazilian Symposium on Neural Networks, 2012, Curitiba - PR. Proceedings of SBRN 2012 - Bracis 2012, 2012. p. 107-112.

DIAS, M. A.; SALES, D.O., OSÓRIO, F. S. Automatic Generation of LUTs for Hardware Neural Networks. In: 2014 Brazilian Conference on Intelligent Systems (BRACIS), 2014, Sao Paulo. p. 115-120, 2014

DIAS, M. A.; OSÓRIO, F. S. Reconfigurable hardware architecture for vision-based driving systems. In: 2015 12th Latin American Robotics Symposium and 2015 3rd Brazilian Symposium on Robotics (LARS-SBR), 2015. p. 97–102.

DIAS, M. A.; SALES, D.O., OSÓRIO, F. S. Automatic generation of luts for hardware neural networks. Neurocomputing, Elsevier, Amsterdam, v. 180, p. 108–120, 2016.

7.2 Trabalhos Futuros

Uma das principais características deste trabalho de pesquisa é proporcionar o desenvolvimento de diversos sistemas baseados em seu resultado. O trabalho futuro mais imediato seria o desenvolvimento da integração do sistema com o sistema de navegação autônoma dos veículos do LRM para um teste real. Esta tarefa possui um elevado grau de dificuldade já que seria necessário desenvolver um driver para a comunicação do sistema do FPGA com o sistema, que atualmente utiliza o ROS (*Robot Operating System*). Após o desenvolvimento é possível também retirar o design do kit de desenvolvimento e criar um hardware embarcado somente com o FPGA, a câmera e as interfaces necessárias resultando em um sistema ainda menor.

Possibilidade de novos designs de RNAs em hardware, pois atualmente vários estudos apresentam a utilização de redes com Deep Learning que utilizam modelos diferenciados, como

as redes convolucionais. Novamente, estas redes podem ser treinadas offline (procedimento utilizado neste trabalho e também em empresas que desenvolvem veículos autônomos atualmente como a Google) e depois os pesos podem ser utilizados na estrutura de hardware. A utilização de FPGAs melhores, e interconectados, pode permitir a implementação de vários processadores em paralelo para otimizar a estrutura e a análise de mais de um quadro da imagem simultaneamente, tornando o sistema multi-core e aumentando consideravelmente a taxa de quadros por segundo.

O sistema de classificação em hardware, resultado deste trabalho, possibilita diversas otimizações e modificações. Dentre elas é possível destacar: o sistema pode ser melhorado com as otimizações de síntese e sincronizações necessárias para torná-las viáveis, o hardware pode ser modificado para possibilitar a utilização de imagens maiores e outros classificadores já que o sistema de processamento de imagens está desenvolvido, o sistema pode ser integrado a diversos tipos de robôs *indoor* e *outdoor* para navegação. Em resumo, este sistema funcional é um ponto de partida para qualquer sistema que implemente um classificador baseado em imagens.

REFERÊNCIAS

ARATO, P.; S.JUHASZ; MANN, A. Z.; ORBAN, A.; PAAP, D. Hardware-software partitioning in embedded system design. **Proceedings of the IEEE International Symposium on Intelligent Signal Processing**, 2003. Citado na página 54.

AYBAY, I. Classification of neural network hardware. **Neural Network World**, v. 6, n. 1, p. 11–29, 1996. Citado 2 vezes nas páginas 69 e 70.

BADE, S.; HUTCHINGS, B. Fpga-based stochastic neural networks-implementation. In: **FPGAs for Custom Computing Machines, 1994. Proceedings. IEEE Workshop on**. [S.l.: s.n.], 1994. p. 189–198. Citado 2 vezes nas páginas 71 e 74.

BATE, I.; CONMY, P. Certification of fpgas - current issues and possible solutions. In: **Safety-Critical Systems: Problems, Process and Practice**. [S.l.]: Springer London, 2009. p. 149–165. Citado na página 39.

BATTITI, R.; COLLA, A. M. Democracy in neural nets: Voting schemes for classification. **Neural Networks**, v. 7, n. 4, p. 691–707, 1994. Citado na página 66.

BERGER, S. A. **Embedded Systems Design: An Introduction to Processes, Tools and Techniques**. Berkley, CA, USA: CMP Books, 2001. ISBN 1578200733. Citado na página 36.

BOBDA, C. **Introduction to Reconfigurable Computing: Architectures, Algorithms, and Applications**. [S.l.]: Springer Publishing Company, Incorporated, 2007. ISBN 1402060882, 9781402060885. Citado 5 vezes nas páginas 15, 46, 47, 48 e 51.

BONIN-FONT, F.; ORTIZ, A.; OLIVER, G. Visual navigation for mobile robots: A survey. **J. Intell. Robotics Syst.**, Kluwer Academic Publishers, Hingham, MA, USA, v. 53, n. 3, p. 263–296, 2008. Citado na página 25.

BOTROS, N.; ABDUL-AZIZ, M. Hardware implementation of an artificial neural network using field programmable gate arrays (fpga's). **Industrial Electronics, IEEE Transactions on**, v. 41, n. 6, p. 665–667, dec 1994. Citado na página 74.

BRAGA, A.; LLANOS, C.; GOHRINGER, D.; OBIE, J.; BECKER, J.; HUBNER, M. Performance, accuracy, power consumption and resource utilization analysis for hardware / software realized artificial neural networks. In: **Bio-Inspired Computing: Theories and Applications (BIC-TA), 2010 IEEE Fifth International Conference on**. [S.l.: s.n.], 2010. p. 1629–1636. Citado na página 76.

BRAGA A.; CARVALHO A. P. L.; LUDERMIR, T. **Redes Neurais Artificiais: Teoria e Aplicações**. [S.l.]: LTC, 2007. Citado 3 vezes nas páginas 56, 60 e 62.

BUEHLER, M.; IAGNEMMA, K.; SINGH, S. **The 2005 DARPA Grand Challenge: The Great Robot Race**. 1. ed. [S.l.]: Springer, 2007. Citado na página 25.

_____. **The DARPA Urban Challenge: Autonomous Vehicles in City Traffic**. 1. ed. [S.l.]: Springer Publishing Company, Incorporated, 2009. Citado na página 25.

CANAS, A.; ORTIGOSA, E.; ROS, E.; ORTIGOSA, P. Fpga implementation of a fully and partially connected mlp. In: OMONDI, A.; RAJAPAKSE, J. (Ed.). **FPGA Implementations of Neural Networks**. [S.l.]: Springer US, 2006. p. 271–296. Citado na página 76.

CHE, S.; LI, J.; SHEAFFER, J.; SKADRON, K.; LACH, J. Accelerating compute-intensive applications with gpus and fpgas. In: **Application Specific Processors, 2008. SASP 2008. Symposium on**. [S.l.: s.n.], 2008. p. 101–107. Citado na página 85.

CHUN, R.; MCNAMEE, L. Immunization of neural networks against hardware faults. In: **Circuits and Systems, 1990., IEEE International Symposium on**. [S.l.: s.n.], 1990. p. 714–718 vol.1. Citado na página 72.

CLOUTIER, J.; SIMARD, P. Y. Hardware implementation of the backpropagation without multiplication. In: **In Proceedings of the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems**. [S.l.: s.n.], 1994. p. 46–55. Citado na página 105.

COMPTON, K.; HAUCK, S.; COMPTON, K. An introduction to reconfigurable computing. **IEEE Computer**, 2000. Citado na página 35.

COPE, B.; CHEUNG, P.; LUK, W.; WITT, S. Have gpus made fpgas redundant in the field of video processing? In: **Field-Programmable Technology, 2005. Proceedings. 2005 IEEE International Conference on**. [S.l.: s.n.], 2005. p. 111–118. Citado 2 vezes nas páginas 85 e 87.

COPE, B.; CHEUNG, P.; LUK, W.; HOWES, L. Performance comparison of graphics processors to reconfigurable logic: A case study. **Computers, IEEE Transactions on**, v. 59, n. 4, p. 433–448, abril 2010. Citado na página 86.

CORIC, S.; LATINOVIC, I.; PAVASOVIC, A. A neural network fpga implementation. In: **Neural Network Applications in Electrical Engineering, 2000. NEUREL 2000. Proceedings of the 5th Seminar on**. [S.l.: s.n.], 2000. p. 117–120. Citado na página 75.

CYBENCO, G. **Continuous valued neural networks with two hidden layers are sufficient**. 1988. Citado na página 62.

DAALEN, M. van; JEAUVONS, P.; SHAW-TAYLOR, J. A stochastic neural architecture that exploits dynamically reconfigurable fpgas. In: **FPGAs for Custom Computing Machines, 1993. Proceedings. IEEE Workshop on**. [S.l.: s.n.], 1993. p. 202–211. Citado na página 71.

DIAS, M.; LACERDA, W. Hardware/software co-design using artificial neural network and evolutionary computing. In: **Programmable Logic, 2009. SPL. 5th Southern Conference on**. [S.l.: s.n.], 2009. p. 153–157. Citado na página 54.

DIAS, M.; OSORIO, F. Fixed-point neural network ensembles for visual navigation. In: **Robotics Symposium and Latin American Robotics Symposium (SBR-LARS), 2012 Brazilian**. [S.l.: s.n.], 2012. p. 307–312. Citado 3 vezes nas páginas 45, 106 e 108.

DIAS, M.; OSORIO, F. S.; WOLF, D. Hardware/software co-design implementation of on-chip backpropagation. In: **Neural Networks (SBRN), 2012 Brazilian Symposium on**. [S.l.: s.n.], 2012. p. 107–112. Citado 4 vezes nas páginas 16, 17, 45 e 105.

DIAS, M. A.; OSORIO, F. S. Hardware/software co-design for image cross-correlation. In: **Proceedings of The First International Conference on Integrated Computer Technology (INTECH), Communications in Computer and Information Science**. [S.l.]: Berlin Heidelberg : Springer-Verlag, 2011. v. 165, p. 161–175. Citado na página [45](#).

DIAS, M. A.; OSÓRIO, F. S. Reconfigurable hardware architecture for vision-based driving systems. In: **2015 12th Latin American Robotics Symposium and 2015 3rd Brazilian Symposium on Robotics (LARS-SBR)**. [S.l.: s.n.], 2015. p. 97–102. Citado 2 vezes nas páginas [17](#) e [110](#).

DIAS, M. A.; SALES, D. O.; OSORIO, F. S. Hw/sw co-design architecture for evolutionary robotics. In: **Proceedings of The 7th Latin American Robotics Symposium (LARS)**. [S.l.: s.n.], 2010. p. 43 – 48. Citado na página [45](#).

_____. A profile-based method for hardware/software co-design applied in evolutionary robotics using reconfigurable computing. In: **IEEE Electronics Robotics and Automotive Mechanics Conference (CERMA)**. [S.l.: s.n.], 2010. p. 463–468. Citado na página [45](#).

_____. Automatic generation of luts for hardware neural networks. **Neurocomputing**, Elsevier, Amsterdam, v. 180, p. 108–120, 2016. Citado na página [116](#).

DOMINGOS, P.; SILVA, F.; NETO, H. An efficient and scalable architecture for neural networks with backpropagation learning. In: **Field Programmable Logic and Applications, 2005. International Conference on**. [S.l.: s.n.], 2005. p. 89 – 94. Citado na página [75](#).

EBERT, C.; JONES, C. Embedded software: Facts, figures, and future. **Computer**, v. 42, n. 4, p. 42–52, April 2009. Citado na página [40](#).

ELDREDGE, J. G.; HUTCHINGS, B. L. Rrann: A hardware implementation of the backpropagation algorithm using reconfigurable fpgas. In: **In IEEE World Conference on Computational Intelligence**. [S.l.: s.n.], 1994. p. 2097–2102. Citado na página [105](#).

ESTRIN, G. Reconfigurable computer origins: the ucla fixed-plus-variable (f+v) structure computer. **Annals of the History of Computing, IEEE**, v. 24, n. 4, p. 3–9, Oct-Dec 2002. Citado na página [47](#).

FERNANDES, L. C.; SOUZA, J. R.; PESSIN, G.; SHINZATO, P. Y.; SALES, D.; MENDES, C.; PRADO, M.; KLASER, R.; MAGALHÃES, A. C.; HATA, A.; PIGATTO, D.; BRANCO, K. C.; JR., V. G.; OSORIO, F. S.; WOLF, D. F. Carina intelligent robotic car: Architectural design and applications. **Journal of Systems Architecture**, v. 60, n. 4, p. 372 – 392, 2014. Citado na página [93](#).

FOWERS, J.; BROWN, G.; COOKE, P.; STITT, G. A performance and energy comparison of fpga, gpu, and multicore for sliding-window applications. In: **Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays**. New York, NY, USA: ACM, 2012. (FPGA '12), p. 47–56. Citado na página [86](#).

FU, H. Accelerating scientific computing through gpu and fpga. In: . [S.l.]: Presented at Stanford Center for Computational Earth and Environmental Science, 2010. p. 52. Citado na página [85](#).

FUENTE, M. de la; ECHANOBÉ, J.; CAMPO, I. del; SUSPERREGUI, L.; MAURTUA, I. Hardware implementation of a neural-network recognition module for visual servoing in a mobile robot. In: **Database and Expert Systems Applications (DEXA), 2010 Workshop on**. [S.l.: s.n.], 2010. p. 226 –232. Citado na página 77.

GHOLAMHOSSEINI H.; SHUYING, H. A high speed vision system for robots using fpga technology. In: **Mechatronics and Machine Vision in Practice, 2008. M2VIP 2008. 15th International Conference on. IEEE**. [S.l.: s.n.], 2008. p. 81–84. Citado na página 110.

GHOSH, S. K. **Hardware Description Languages: Concepts and Principles**. [S.l.]: Wiley-IEEE Press, 1999. ISBN 0780347447. Citado na página 50.

GIACINTO, G. Design of effective neural network ensembles for image classification purposes. **Image and Vision Computing**, v. 19, n. 9-10, p. 699–707, 2001. Citado 3 vezes nas páginas 66, 106 e 120.

GIRONÉS, R. G.; PALERO, R. C.; BOLUDA, J. C.; CORTÉS, A. S. Fpga implementation of a pipelined on-line backpropagation. **J. VLSI Signal Process. Syst.**, Kluwer Academic Publishers, Hingham, MA, USA, v. 40, n. 2, p. 189–213, jun. 2005. ISSN 0922-5773. Disponível em: <<http://dx.doi.org/10.1007/s11265-005-4961-3>>. Citado na página 105.

GOMPERTS, A.; UKIL, A.; ZURFLUH, F. Development and implementation of parameterized fpga-based general purpose neural networks for online applications. **Industrial Informatics, IEEE Transactions on**, v. 7, n. 1, p. 78 –89, feb. 2011. Citado na página 77.

GUPTA, R. K. **Co-Synthesis of Hardware and Software for Digital Embedded Systems**. Norwell, MA, USA: Kluwer Academic Publishers, 1995. Foreword By-Micheli, Giovanni De. ISBN 0792396138. Citado na página 36.

HANSEN, L. K.; SALAMON, P. Neural network ensembles. **IEEE Trans. Pattern Anal. Mach. Intell.**, IEEE Computer Society, Washington, DC, USA, v. 12, n. 10, p. 993–1001, 1990. ISSN 0162-8828. Citado na página 65.

HARIPRASATH, S.; PRABAKAR, T. Fpga implementation of multilayer feed forward neural network architecture using vhdl. In: **Computing, Communication and Applications (ICCCA), 2012 International Conference on**. [S.l.: s.n.], 2012. p. 1 –6. Citado na página 78.

HAUCK, S.; DEHON, A. **Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation (Systems on Silicon)**. 30 Cororate Drive, Suite 400, Burlington, MA: Morgan Kaufmann, 2007. ISBN 0123705223. Citado 2 vezes nas páginas 29 e 35.

HAYKIN, S. **Neural Networks: A Comprehensive Foundation**. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998. Citado 7 vezes nas páginas 15, 30, 55, 56, 57, 62 e 64.

HENKEL, J. Closing the soc design gap. **Computer**, v. 36, n. 9, p. 119–121, 2003. Citado na página 29.

HIDALGO, J. I.; LANCHARES, J. Functional partitioning for hardware-software codesign using genetic algorithms. In: **Proceedings of the 23rd EUROMICRO conference**. [S.l.: s.n.], 1997. p. 631–638. Citado na página 52.

HIMAVATHI, S.; ANITHA, D.; MUTHURAMALINGAM, A. Feedforward neural network implementation in fpga using layer multiplexing for effective resource utilization. **Neural Networks, IEEE Transactions on**, v. 18, n. 3, p. 880 –888, may 2007. Citado na página 76.

HIRAIWA J.; AMANO, H. An fpga implementation of reconfigurable real-time vision architecture. In: **Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on**. [S.l.: s.n.], 2013. p. 150–155. Citado na página 110.

HORNE, B.; JAMSHIDI, M.; VADIEE, N. Neural networks in robotics: A survey. **Journal of Intelligent and Robotic Systems**, Kluwer Academic Publishers, v. 3, p. 51–66, 1990. Citado na página 26.

IEEE. **IEEE standard for binary floating point arithmetic, ANSI IEEE std 754 1985**. 1985. 9 - 25 p. Disponível em: <http://grouper.ieee.org/groups/754/>. Citado na página 43.

ISHII, H.; SHIBATA, T.; KOSAKA, H.; OHMI, T. Hardware-backpropagation learning of neuron mos neural networks. In: **Electron Devices Meeting, 1992. IEDM '92. Technical Digest., International**. [S.l.: s.n.], 1992. p. 435 –438. ISSN 0163-1918. Citado na página 105.

IZEBOUDJEN, N.; FARAH, A.; BESSALAH, H.; BOURIDENE, A.; CHIKHI, N. Towards a platform for fpga implementation of the mlp based back propagation algorithm. In: **Proceedings of the 9th international work conference on Artificial neural networks**. [S.l.]: Springer-Verlag, 2007. (IWANN'07), p. 497–505. Citado na página 72.

JESSEN, J. B. **Comparing FPGA and GPU performance for both sparse and dense stereo reconstruction**. [S.l.], 2009. 60 p. Citado na página 84.

JOCHER, T.; BALUJA, S. **Massively Parallel, Adaptive, Color Image Processing for Autonomous Road Following**. Pittsburgh, PA: [s.n.], 1993. Citado na página 26.

JOHANSSON, U.; LOFSTROM, T.; NIKLASSON, L. Obtaining accurate neural network ensembles. In: **(CIMCA-IAWTIC'06) - Volume 02**. Washington, DC, USA: IEEE Computer Society, 2005. (CIMCA '05), p. 103–108. Citado na página 66.

JONES, D.; POWELL, A.; BOUGANIS, C.-S.; CHEUNG, P. Gpu versus fpga for high productivity computing. In: **Field Programmable Logic and Applications (FPL), 2010 International Conference on**. [S.l.: s.n.], 2010. p. 119 –124. Citado na página 86.

JOOST, R.; SALOMON, R. Time coding output neurons in digital artificial neural networks. In: **Neural Networks (IJCNN), The 2012 International Joint Conference on**. [S.l.: s.n.], 2012. p. 1 –8. Citado na página 78.

KALAROT, R.; MORRIS, J. Comparison of fpga and gpu implementations of real-time stereo vision. In: **Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on**. [S.l.: s.n.], 2010. p. 9 –15. Citado na página 86.

KALINSKY, D. **Architecture of safety-critical systems**. 2005. 1 - 9 p. Disponível em: <http://tinyurl.com/bgvdg8k>. Citado na página 39.

KAPLAN, A.; KASTNER, R. A survey of hardware/software system partitioning. In: **UCLA Computer Science Department and UCSB Electrical and Computer Engineering Department**. [S.l.: s.n.], 2000. Citado na página 53.

KELLY, A. **Traffic accidents are biggest killer of young people worldwide, report says**. 2012. Disponível em: <http://tinyurl.com/d2fu2m9>. Citado na página 22.

KEULEN, E. van; COLAK, S.; WITHAGEN, H.; HEGT, H. Neural network hardware performance criteria. In: **Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on**. [S.l.: s.n.], 1994. v. 3, p. 1885–1888 vol.3. Citado 2 vezes nas páginas 73 e 74.

KRIPS, M.; LAMMERT, T.; KUMMERT, A. Fpga implementation of a neural network for a real-time hand tracking system. In: **Electronic Design, Test and Applications, 2002. Proceedings. The First IEEE International Workshop on**. [S.l.: s.n.], 2002. p. 313–317. Citado na página 75.

KROGH, A.; VEDELSBY, J. Neural network ensembles, cross validation, and active learning. In: **Advances in Neural Information Processing Systems**. [S.l.]: MIT Press, 1995. p. 231–238. Citado 2 vezes nas páginas 66 e 68.

LAPLANTE, P. A. **Real-time systems design and analysis - An engineers handbook**. 3. ed. [S.l.]: IEEE Press - Wiley, 2004. Citado 2 vezes nas páginas 37 e 38.

LIU, J.; LIANG, D. A survey of fpga-based hardware implementation of anns. In: **Neural Networks and Brain, 2005. ICNN B '05. International Conference on**. [S.l.: s.n.], 2005. v. 2, p. 915–918. Citado na página 69.

LUETTEL, T.; HIMMELSBACH, M.; WUENSCH, H.-J. Autonomous ground vehicles - concepts and a path to the future. **Proceedings of the IEEE**, v. 100, n. Centennial-Issue, p. 1831–1839, 2012. Citado na página 25.

LUK, W. Accelerating neural models with fpgas and gpus. In: . [S.l.]: Presented at BIECS09, Imperial College, London, 2009. p. 56. Citado na página 86.

MCCLELLAND, J. L.; RUMELHART, D. E. **Parallel Distributed Processing - Explorations in the Microstructure of Cognition**. [S.l.]: A Bradford Book - The MIT Press, 1986. v. 2: Psychological and Biological Models. Citado na página 62.

MEHER, P. An optimized lookup-table for the evaluation of sigmoid function for artificial neural networks. In: **VLSI System on Chip Conference (VLSI-SoC), 2010 18th IEEE/IFIP**. [S.l.: s.n.], 2010. p. 91–95. Citado na página 72.

MICHELI, G. de; GUPTA, R. K. Hardware/software co-design. In: **Proceedings of IEEE**, vol. 85. [S.l.: s.n.], 1997. p. 349–365. Citado na página 52.

MINKSY, M.; PAPERT, S. **Perceptrons: An introduction to computational geometry**. [S.l.]: The MIT Press, Massachusetts, 1969. Citado na página 61.

MISRA, J.; SAHA, I. Artificial neural networks in hardware: A survey of two decades of progress. **Neurocomputing**, v. 74, n. 1, p. 239–255, 2010. Citado 4 vezes nas páginas 30, 68, 72 e 74.

MONMASSON, E.; IDKHAJINE, L.; CIRSTEAN, M.; BAHRI, I.; TISAN, A.; NAOUAR, M. Fpgas in industrial control applications. **Industrial Informatics, IEEE Transactions on**, v. 7, n. 2, p. 224–243, may 2011. Citado na página 72.

MOORE, G. E. Cramming more components onto integrated circuits. **Electronics**, v. 38, n. 8, 1965. Citado na página 28.

- MORENO, F.; ALARCON, J.; SALVADOR, R.; RIESGO, T. Fpga implementation of an image recognition system based on tiny neural networks and on-line reconfiguration. In: **Industrial Electronics, 2008. IECON 2008. 34th Annual Conference of IEEE**. [S.l.: s.n.], 2008. p. 2445–2452. Citado na página 76.
- MYERS, D.; VINCENT, J.; ORREY, D. Hannibal: A vlsi building block for neural networks with on-chip backpropagation learning. **Neurocomputing**, v. 5, n. 1, p. 25 – 37, 1993. Citado na página 105.
- NARAYANAN, V.; XIE, Y. Reliability concerns in embedded system designs. **Computer**, v. 39, n. 1, p. 118 – 120, jan. 2006. Citado na página 40.
- NEDJAH, N.; MOURELLE, L. Fpga-based hardware architecture for neural networks: binary radix vs. stochastic. In: **Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings. 16th Symposium on**. [S.l.: s.n.], 2003. p. 111 – 116. Citado na página 75.
- NEDJAH, N.; MOURELLE, L. de M. Reconfigurable hardware for neural networks: binary versus stochastic. **Neural Comput. Appl.**, Springer-Verlag, v. 16, n. 3, p. 249–255, 2007. Citado na página 72.
- NEDJAH, N.; SILVA, R. M. da; MOURELLE, L. M.; SILVA, M. V. C. da. Dynamic mac-based architecture of artificial neural networks suitable for hardware implementation on fpgas. **Neurocomput.**, Elsevier Science Publishers B. V., v. 72, n. 10-12, p. 2171–2179, 2009. Citado 2 vezes nas páginas 71 e 77.
- NICHOLS, K. R.; MOUSSA, M. A.; AREIBI, S. M. Feasibility of floating-point arithmetic in fpga based artificial neural networks. In: **In CAINE**. [S.l.: s.n.], 2002. p. 8–13. Citado na página 72.
- OMONDI, A.; RAJAPAKSE, J. **FPGA Implementations of Neural Networks**. [S.l.]: Springer, 2006. Citado 2 vezes nas páginas 17 e 69.
- OVERTON, M. L. **Numerical Computing with IEEE Floating Point Arithmetic**. 1. ed. [S.l.]: Soc for Industrial & Applied Math, 2001. Citado 2 vezes nas páginas 43 e 44.
- PEDRONI, V. A. **Circuit Design with VHDL**. Cambridge, MA, USA: MIT Press, 2004. ISBN 0262162245. Citado na página 51.
- PELLERIN, D.; TAYLOR, D. **VHDL made easy!** Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1997. ISBN 0-13-650763-8. Citado na página 51.
- PEREIRA, K.; ATHANAS, P.; LIN, H.; FENG, W. Spectral method characterization on fpga and gpu accelerators. In: **Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on**. [S.l.: s.n.], 2011. p. 487 –492. Citado na página 86.
- PERRONE, M. P.; COOPER, L. N. When networks disagree: Ensemble methods for hybrid neural networks. In: . [S.l.]: Chapman and Hall, 1993. p. 126–142. Citado na página 66.
- PETKO, M.; KARPIEL, G. Implementation of control algorithms in field programmable gate arrays. In: **Advanced intelligent mechatronics, 2007 IEEE/ASME international conference on**. [S.l.: s.n.], 2007. p. 1 –6. Citado na página 76.
- POMERLEAU, D. A. **ALVINN, an autonomous land vehicle in a neural network**. [S.l.], 1989. Citado na página 22.

RIEDMILLER, M.; BRAUN, H. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In: **Proceedings of the IEEE International Conference on Neural Networks (ICNN)**. [S.l.: s.n.], 1993. p. 586–591. Citado na página 63.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. **Psychol**, n. 65, p. 386–408, 1958. Citado na página 61.

RUMELHART, D. E.; MCCLELLAND, J. L. **Parallel Distributed Processing - Explorations in the Microstructure of Cognition**. [S.l.]: A Bradford Book - The MIT Press, 1986. v. 1 - Foundations. Citado na página 62.

RUSHBY, J.; SHA, S. L.; LEHOCZKY, J. P.; SOLUTIONS, R. R. **Critical System Properties: Survey and Taxonomy**. 1994. 1 - 31 p. Citado na página 39.

SANTOS, P.; OUELLET-POULIN, D.; SHAPIRO, D.; BOLIC, M. Artificial neural network acceleration on fpga using custom instruction. In: **Electrical and Computer Engineering (CCECE), 2011 24th Canadian Conference on**. [S.l.: s.n.], 2011. p. 000450–000455. Citado na página 77.

SANTOS, R. M.; SANTOS, J.; OROZCO, J. D. Power saving and fault-tolerance in real-time critical embedded systems. **Journal of Systems Architecture**, v. 55, n. 2, p. 90 – 101, 2009. Citado na página 39.

SAVICH, A.; MOUSSA, M.; AREIBI, S. A scalable pipelined architecture for real-time computation of mlp-bp neural networks. **Microprocessors and Microsystems**, v. 36, n. 2, p. 138 – 150, 2012. Citado 3 vezes nas páginas 15, 70 e 86.

SHARKEY, A. J. C. On combining artificial neural nets. **Connection Science**, v. 8, p. 299–313, 1996. Citado 2 vezes nas páginas 66 e 68.

SHINZATO, P. Y. **Sistema de identificação de superfícies navegáveis baseado em visão computacional e redes neurais artificiais**. Dissertação (Mestrado) — Instituto de Ciências Matemáticas e de Computação - USP, Sao Carlos - SP, 2010. Citado na página 27.

SHINZATO, P. Y.; WOLF, D. F. A road following approach using artificial neural networks combinations. **J. Intell. Robotics Syst.**, Kluwer Academic Publishers, Hingham, MA, USA, v. 62, n. 3-4, p. 527–546, 2011. Citado 11 vezes nas páginas 26, 31, 32, 93, 94, 103, 106, 107, 108, 110 e 118.

SILVA, R. M. da; MOURELLE, L. de M.; NEDJAH, N. Compact yet efficient hardware implementation of artificial neural networks with customized topology. **Expert Systems with Applications**, v. 39, n. 10, p. 9191 – 9206, 2012. Citado na página 105.

SIMOES, E. do V.; UEBEL, L. F.; BARONE, D. A. C. Hardware implementation of ram neural networks. **Pattern Recognition Letters**, v. 17, n. 4, p. 421 – 429, 1996. Citado 2 vezes nas páginas 69 e 74.

SKLIAROVA, I.; FERRARI, A. B. Introdução à computação reconfigurável. **Revista do DETUA**, v. 2, n. 6, September 2003. Citado 4 vezes nas páginas 15, 47, 48 e 49.

SKODA, P.; LIPIC, T.; SRP, A.; ROGINA, B.; SKALA, K.; VAJDA, F. Implementation framework for artificial neural networks on fpga. In: **MIPRO, 2011 Proceedings of the 34th International Convention**. [S.l.: s.n.], 2011. p. 274 – 278. Citado na página 77.

SOUZA, J. R.; PESSIN, G.; SHINZATO, P. Y.; OSORIO, F. S.; WOLF, D. F. Vision-based waypoint following using templates and artificial neural networks. **Neurocomputing**, v. 107, p. 77 – 86, 2013. Timely Neural Networks Applications in Engineering Selected Papers from the 12th {EANN} International Conference, 2011. Citado na página 93.

SOUZA, J. R.; SALES, D. O.; SHINZATO, P. Y.; OSORIO, F. S.; WOLF, D. F. Template-based autonomous navigation and obstacle avoidance in urban environments. **SIGAPP Appl. Comput. Rev.**, ACM, New York, NY, USA, v. 11, n. 4, p. 49–59, 2011. Citado 3 vezes nas páginas 15, 96 e 97.

SUZUKI, T. Discussing image sensor evolution and the future of imaging at the isscc 2010 plenary session. **Side View**, v. 60, p. 1–4, may 2010. Citado na página 83.

SZABO, T.; ANTONI, L.; HORVATH, G.; FEHER, B. A full-parallel digital implementation for pre-trained nns. In: **Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on**. [S.l.: s.n.], 2000. v. 2, p. 49 –54 vol.2. Citado 2 vezes nas páginas 71 e 74.

SZABO, T.; HORVATH, G. An efficient hardware implementation of feed-forward neural networks. **Applied Intelligence**, Kluwer Academic Publishers, v. 21, p. 143–158, 2004. Citado 2 vezes nas páginas 71 e 72.

TIAN, J.; LI, M.; CHEN, F.; KOU, J. Coevolutionary learning of neural network ensemble for complex classification tasks. **Pattern Recogn.**, Elsevier Science Inc., New York, NY, USA, v. 45, n. 4, p. 1373–1385, 2012. Citado na página 66.

TOMASI, M.; VANEGAS, M.; BARRANCO, F.; DIAZ, J. A novel architecture for a massively parallel low level vision processing engine on chip. In: **Industrial Electronics (ISIE), 2010 IEEE International Symposium on**. [S.l.: s.n.], 2010. p. 3033–3039. Citado na página 110.

TONG, J.; ANDERSON, I.; KHALID, M. Soft-core processors for embedded systems. In: **Microelectronics, 2006. ICM '06. International Conference on**. [S.l.: s.n.], 2006. p. 170 –173. Citado na página 50.

TUMER, K.; GHOSH, J. **Error Correlation And Error Reduction In Ensemble Classifiers**. 1996. Citado na página 66.

UREÑA, R.; MARTINEZ-CAÑADA, P.; GÓMEZ-LÓPEZ, J. M.; MORILLAS, C. A.; PELAYO, F. J. Real-time tone mapping on gpu and fpga. **EURASIP J. Image and Video Processing**, p. 1–15, 2012. Citado na página 86.

VAHID, F.; GIVARGIS, T. **Embedded System Design: A Unified Hardware/Software Introduction**. New York, NY, USA: John Wiley & Sons, Inc., 2001. ISBN 0471386782. Citado 3 vezes nas páginas 35, 36 e 37.

VALDERRAMA, C. A.; LIMA, M. E. de; CAVALCANTE, S.; BARROS, E. **Hardware/-Software co-design: projetando hardware e software concorrentemente**. São Paulo, SP, Brasil: Escola de Computação, 2000. ISBN 0000000000. Citado 3 vezes nas páginas 36, 37 e 52.

WILLIAMS, R. **Real-Time Systems Development**. 1. ed. [S.l.]: Butterworth-Heinemann, 2006. Citado na página 37.

WILMSHURST, T. **Designing Embedded Systems with PIC Microcontrollers: Principles and Applications**. [S.l.]: Newnes, 2006. ISBN 0750667559. Citado na página 36.

WOLF, W. **High-Performance Embedded Computing: Architectures, Applications, and Methodologies**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006. ISBN 012369485X. Citado 2 vezes nas páginas 37 e 53.

WU, G.-D.; ZHU, Z.-W.; LIN, B.-W. Reconfigurable back propagation based neural network architecture. In: **Integrated Circuits (ISIC), 2011 13th International Symposium on**. [S.l.: s.n.], 2011. p. 67 –70. Citado na página 77.

YALAMANCHILI, S. **Introductory VHDL: From Simulation to Synthesis**. [S.l.]: Prentice Hall Inc., 2001. ISBN 130809829. Citado 2 vezes nas páginas 48 e 51.

YIANNACOURAS, P.; ROSE, J.; STEFFAN, J. G. The microarchitecture of fpga-based soft processors. In: **CASES '05: Proceedings of the 2005 int. conf. on Compilers, arch. and synthesis for emb. sys**. New York, NY, USA: ACM, 2005. p. 202–212. ISBN 1-59593-149-X. Citado na página 49.

YOUSSEF, A.; MOHAMMED, K.; NASAR, A. A reconfigurable, generic and programmable feed forward neural network implementation in fpga. In: **Computer Modelling and Simulation (UKSim), 2012 UKSim 14th International Conference on**. [S.l.: s.n.], 2012. p. 9 –13. Citado na página 72.

YUN, S. B.; KIM, Y. J.; DONG, S. S.; LEE, C. H. Hardware implementation of neural network with expansible and reconfigurable architecture. In: **Neural Information Processing, 2002. ICONIP '02. Proceedings of the 9th International Conference on**. [S.l.: s.n.], 2002. v. 2, p. 970 – 975 vol.2. Citado na página 75.

ZHAO, Y.; GAO, J.; YANG, X. A survey of neural network ensemble. In: **Presented at International Conference on Neural Networks and Brain**. Beijing, China: [s.n.], 2005. Citado na página 66.

ZHOU, Z.-H.; WU, J.; TANG, W. **Ensembling Neural Networks: Many Could Be Better Than All**. 2002. Citado 3 vezes nas páginas 65, 68 e 120.

ZOU, A.-M.; HOU, Z.-G.; FU, S.-Y.; TAN, M. Neural networks for mobile robot navigation: A survey. In: WANG, J.; YI, Z.; ZURADA, J.; LU, B.-L.; YIN, H. (Ed.). **Advances in Neural Networks - ISNN 2006**. [S.l.]: Springer Berlin Heidelberg, 2006. (Lecture Notes in Computer Science, v. 3972), p. 1218–1226. Citado na página 26.