Arquitetura multi-core reconfigurável para detecção de pedestres baseada em visão

José Arnaldo Mascagni de Holanda

José Arnaldo Mascagni de Holanda

Arquitetura multi-core reconfigurável para detecção de pedestres baseada em visão

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. *EXEMPLAR DE DEFESA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Eduardo Marques

USP – São Carlos
Janeiro de 2017

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

H634a

Holanda, José Arnaldo Mascagni de
Arquitetura multi-core reconfigurável para detecção de pedestres baseada em visão / José Arnaldo Mascagni de Holanda; orientador Eduardo Marques. - São Carlos - SP, 2017.
146 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, 2017.

1. Detecção de Pedestres. 2. Arquiteturas Multi-core. 3. Computação Reconfigurável. 4. FPGA. I. Marques, Eduardo, orient. II. Título.

José Arnaldo Mascagni de Holanda

Multi-core reconfigurable architecture for vision-based pedestrian detection

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. *EXAMINATION BOARD PRESENTATION COPY*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Eduardo Marques

USP – São Carlos
January 2017

Dedico este trabalho aos meus queridos pais,
José Dantas e Vilza, pelo suporte e incentivo
em cada etapa da minha vida.

Dedico também à minha amada esposa, Michele,
pelo carinho, companheirismo, compreensão
e por toda a ajuda na realização deste trabalho.

AGRADECIMENTOS

À Deus, por Sua bondade, fidelidade, misericórdia, graça e amor.

Ao meu orientador, Eduardo Marques, pelas direções no trabalho, pelos conselhos preciosos, pela paciência em ensinar e pela amizade.

Ao meu supervisor no doutorado sanduíche, João Manuel Paiva Cardoso, pela contribuição preciosa neste trabalho, por compartilhar seu tempo e conhecimento, e por me receber tão bem em seu laboratório.

Aos colegas de laboratório do LCR-USP e do SPECS-FEUP, pelo conhecimento compartilhado e pelos momentos de descontração.

Aos amigos e familiares que, mesmo em momentos difíceis, tornam a vida mais leve e alegre.

Ao Instituto Federal de São Paulo, pelo apoio financeiro e por permitir dedicar-me ao doutorado, assim como aos colegas de trabalho, por todo o suporte.

À CAPES, pelo apoio financeiro na forma de Bolsa de Doutorado Sanduíche (PDSE).

“Bem-aventurado o homem que acha sabedoria,
e o homem que adquire conhecimento;
Porque é melhor a sua mercadoria
do que artigos de prata, e maior
o seu lucro que o ouro mais fino.”
(Provérbios 3:13,14)

RESUMO

HOLANDA, J. A. M. DE. **Arquitetura multi-core reconfigurável para detecção de pedestres baseada em visão**. 2017. 146 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2017.

Dentre as diversas tecnologias de Assistência Avançada ao Condutor (ADAS) que têm sido adiconadas aos automóveis modernos estão os sistemas de detecção de pedestres. Tais sistemas utilizam sensores, como radares, lasers e câmeras de vídeo para captar informações do ambiente e evitar a colisão com pessoas no contexto do trânsito. Câmeras de vídeo têm se apresentado como um ótima opção para esses sitemas, devido ao relativo baixo custo e à riqueza de informações que capturam do ambiente. Muitas técnicas para detecção de pedetres baseadas em visão têm surgido nos últimos anos, tendo como característica a necessidade de um grande poder computacional para que se possa realizar o processamento das imagens em tempo real, de forma robusta, confiável e com baixa taxa de erros. Além disso, é necessário que sistemas que implementem essas técnicas tenham baixo consumo de energia, para que possam funcionar em um ambiente embarcado, como os automóveis. Uma tendência desses sistemas é o processamento de imagens de múltiplas câmeras presentes no veículo, de forma que o sistema consiga perceber potenciais perigos de colisão ao redor do veículo. Neste contexto, este trabalho aborda o coprojeto de hardware e software de uma arquitetura para detecção de pedestres, considerando a presença de quatro câmeras em um veículo (uma frontal, uma traseira e duas laterais). Com este propósito, utiliza-se a flexibilidade dos dispositivos FPGA para a exploração do espaço de projeto e a construção de uma arquitetura que forneça o desempenho necessário, o consumo de energia em níveis adequados e que também permita a adaptação a novos cenários e a evolução das técnicas de detecção de pedestres por meio da programabilidade. O desenvolvimento da arquitetura baseou-se em dois algoritmos amplamente utilizados para detecção de pedestres, que são o *Histogram of Oriented Gradients* (HOG) e o *Integral Channel Features* (ICF). Ambos introduzem técnicas que servem como base para os algoritmos de detecção modernos. A arquitetura implementada permitiu a exploração de diferentes tipos de paralelismo das aplicações por meio do uso de múltiplos processadores *softcore*, bem como a aceleração de funções críticas por meio de implementações em hardware. Também foi demonstrada sua viabilidade no atendimento a um sistema contendo quatro câmeras de vídeo.

Palavras-chave: Detecção de Pedestres, Arquiteturas Multi-core, Computação Reconfigurável, FPGA.

ABSTRACT

HOLANDA, J. A. M. DE. **Multi-core reconfigurable architecture for vision-based pedestrian detection**. 2017. 146 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2017.

Among the several Advanced Driver Assistance (ADAS) technologies that have been added to modern vehicles are pedestrian detection systems. Those systems use sensors, such as radars, lasers, and video cameras to capture information from the environment and avoid collision with people in the context of traffic. Video cameras have come out as a great option for such systems because of the relatively low cost and all of information they are able to capture from the environment. Many techniques for vison-based pedestrian detection have appeared in the last years, having as characteristic the necessity of a great computational power so that image can be processed in real time, in a robust and reliable way, and with low error rate. In addition, systems that implement these techniques require low power consumption, so they can operate in an embedded environment such as automobiles. One trend of these systems is the processing of images from multiple cameras mounted in vehicles, so that the system can detect potential collision hazards around the vehicle. In this context, this work addresses the hardware and software codesign of an architecture for pedestrian detection, considering the presence of four cameras in a vehicle (one in the front, one in the rear and two in the sides). For this purpose, the flexibility of FPGA devices is used for design space exploration the construction of an architecture that provides the necessary performance, energy consumption at appropriate levels and also allows adaptation to new scenarios and evolution of pedestrian detection techniques through programmability. The development of the architecture was based on two algorithms widely used for pedestrian detection, which are Histogram of Oriented Gradients (HOG) and Integral Channel Features (ICF). Both introduce techniques that serve as the basis for modern detection algorithms. The implemented architecture allowed the exploration of different types of parallelism through the use of multiple softcore processors, as well as the acceleration of critical functions through implementations in hardware. It has also been demonstrated its feasibility in attending to a system containing four video cameras.

Keywords: Pedestrian Detection, Multi-core architecture, Reconfigurable Computing, FPGA.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADAS	*Advanced Driver Assistance Systems*
ASIC	*Application Specific Integrated Circuit*
ASIP	*Application Specific Processors*
CIE	*Commission Internationale de l'Éclairage*
CNN	*Convolutional Neural Networks*
DNIT	Departamento Nacional de Infraestrutura de Transportes
DPM	*Deformable Part Models*
DSP	*Digital Signal Processor*
DSS	*Driver Support System*
ECU	Electronic Control Units
FPDW	*Fastest Pedestrian Detector in the West*
FPGA	*Field-programmable Gate Array*
GOPS	GIGA operações por segundo
GPROF	GNU *Profiler*
GPS	*Global Positioning System*
GPU	*Graphics Processing Units*
HDR	*High Dynamic Range*
HOG	*Histogram of Oriented Gradients*
ICF	*Integral Channel Features*
IP	*Intelectual Property*
ITS	*Intelligent Transport Systems*
IV	*Intelligent Vehicle*
IVIS	*In-Vehicle Information Systems*
IVSS	*Intelligent Vehicle Safety Systems*
LBP	*Local Binary Patterns*
LIDAR	*Light Detection And Ranging*
LUT	*Look-up Tables*
MPC	*Multithreaded Processing Cluster*
MPSoC	Multi-Processor System-on-Chip
PLL	Phase Locked Loops
PMA	*Programmable Macro Array*

PPS	*Pedestrian Protection Systems*
RADAR	*Radio Detecting and Ranging*
ROI	*Regions of Interest*
SDRAM	*Synchronous Dynamic Random Access Memory*
SIFT	*Scale Invariant Feature Transform*
SoC	*System-on-a-Chip*
SONAR	*Sound Navigation and Ranging*
SSRAM	*Synchronous Static Random Access Memory*
TCDM	*Tightly-Coupled Data Memories*
ULA	Unidade Lógica e Aritmética
VMP	*Vector Microcode Processors*
VRU	*Vulnerable Road Users*

SUMÁRIO

CAPÍTULO

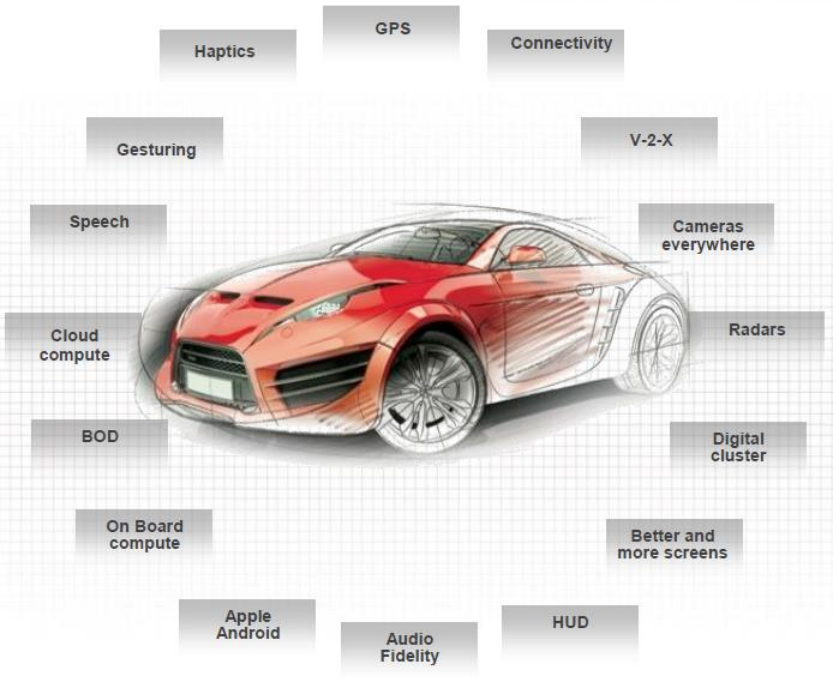
1

INTRODUÇÃO

Desde a invenção dos primeiros veículos movidos a motor até os dias atuais, a indústria automobilística têm agregado em seus produtos um grande número de inovações. Além da notável evolução mecânica e estrutural dos automóveis ao longo dos anos, uma das áreas que mais tem evoluído neste setor é a da eletrônica embarcada, a qual é a atual responsável por cerca de 22% das receitas do mercado global de sistemas embarcados (Global Market Insights, 2016). Essa grande relevância tem sido impulsionada pela produção de veículos híbridos ou elétricos, bem como a adoção de novas tecnologias que visam tornar o veículo mais inteligente e seguro. Segundo Osadja (2008), atualmente, cerca de 80% das inovações no projeto de um veículo estão relacionadas aos sistemas eletrônicos embarcados. Este número expressivo indica uma presença cada vez maior dos circuitos eletrônicos controlando diversas funcionalidades dentro de um automóvel. Estima-se que a quantidade de eletrônica embarcada em um veículo tradicional (a combustão) no estado-da-arte atinja 40% de seu valor, podendo chegar a 75% em um veículo elétrico (SCURO, 2012). A Figura 1 mostra exemplos de tecnologias que já estão ou que têm sido incorporadas aos veículos atuais e que são controladas eletronicamente.

Muitas das tecnologias que têm sido adicionadas em automóveis modernos têm como objetivo fornecer informações relevantes ao condutor sobre as condições operacionais e ambientais ao redor do veículo, de forma a evitar acidentes. Algumas delas, inclusive, podem atuar no controle do veículo, por exemplo, reduzindo a velocidade ao detectar uma distância não segura a um outro veículo à frente. A adoção de tais tecnologias tem a intenção de mitigar o número pessoas mortas ou feridas em acidentes de trânsito. Em todo o mundo, $1,3$ milhões de pessoas morreram no ano de 2004, vítimas de acidentes de trânsito. Calcula-se que em 2030 esse número será de aproximadamente 2 milhões, subindo da nona posição no *ranking* de maiores causas de fatalidades para o quinto lugar (MODY *et al.*, 2016). No Brasil, segundo dados do relatório de números de acidentes por gravidades do Departamento Nacional de Infraestrutura de Transportes (DNIT), ocorreram aproximadamente 189 mil acidentes de trânsito durante o ano de 2011, com cerca de 7 mil vítimas fatais (DNIT, 2011). Motociclistas e passageiros dos veículos estão entre

Figura 1 – Funcionalidades controladas por sistemas embarcados em um automóvel.



Fonte: Mody *et al.* (2016).

as maiores vítimas, com cerca de 44 mil e 28 mil acidentes, respectivamente. Contudo, acidentes envolvendo pedestres costumam ser mais letais, levando as vítimas ao óbito em 26% dos casos.

Tecnologias automotivas desse tipo são classificadas como *Advanced Driver Assistance Systems* (ADAS), ou, em português, Sistemas de Assistência ao Condutor. Tais sistemas utilizam vários sensores para perceber o ambiente interno e ao redor do veículo, como por exemplo, *Radio Detecting and Ranging* (RADAR), *Light Detection And Ranging* (LIDAR), *Sound Navigation and Ranging* (SONAR), dispositivos de visão noturna e câmeras de vídeo, entre outros. As informações dos sensores podem ser combinadas, de modo que se aproveite as vantagens de cada um para monitorar situações tanto próximas quanto distantes ao veículo. Câmeras de vídeo fornecem dados com grande riqueza de detalhes e são utilizadas em grande parte das implementações de ADAS (MODY *et al.*, 2016). Exemplos de ADAS incluem:

- Auxílio ao estacionamento do veículo;
- Controle de velocidade cruzeiro adaptativo;
- Sistemas de pré-colisão;
- Detecção de saída de pista;
- Assistência à mudança de pista;
- Monitoramento de pontos cegos;

- Monitoramento de sonolência do motorista;
- Visão Noturna;
- Detecção de obstáculos;
- Detecção de pedestres; e
- Detecção da sinalização de trânsito.

Há diversos desafios a serem superados no projeto, implementação, implantação e operação de ADAS. É esperado que esses sistemas sejam capazes de capturar precisamente os dados de entrada, que sejam rápidos em processá-los, que consigam predizer o contexto com precisão e que disparem uma reação em tempo real (ZHAO, 2015). É preciso também que, como sistemas embarcados em um automóvel, tenham baixo consumo de energia. Além disso, devem ser robustos, confiáveis e devem apresentar taxas de erro baixas.

ADAS baseados em visão e que, portanto, têm como entrada imagens provindas de câmeras de vídeo, necessitam de um grande poder computacional, pois é necessário que realizem um processamento complexo, em tempo real, sobre uma imensa quantidade de dados (KURODA; KYO, 2007). O trabalho realizado por Nishiwaki *et al.* (2006) mostrou que sistemas para detecção de objetos no trânsito, como pedestres, placas e obstáculos podem necessitar de centenas de GIGA operações por segundo (GOPS). É, portanto, necessário, que plataformas de hardware embarcadas possuam recursos de processamento para serem capazes de atender a demanda computacional dessas aplicações.

A evolução dos sistemas embarcados nos últimos anos tem permitido a construção de sistemas de hardware cada vez mais complexos e maiores. Tais sistemas são formados por vários processadores heterogêneos, blocos *Intelectual Property* (IP), memórias *on-chip* e periféricos, sendo também conhecidos como *System-on-a-Chip* (SoC). A possibilidade de se ter configurações mais complexas de hardware, contendo vário núcleos de processamento (*multi-core*) *on-chip* têm beneficiado a implementação de sistemas de detecção de pedestres em arquiteturas embarcadas, permitindo alcançar um desempenho até maior do que um processador de propósito geral, mantendo o consumo de energia em patamares adequados para um sistema embarcado automotivo. Assim, a pesquisa e o desenvolvimento de sistemas embarcados mostra-se um tópico bastante relevante para atender às crescentes demandas de processamento introduzidas pelo aperfeiçoamento das técnicas de detecção de pedestres, ao mesmo tempo reduzindo o consumo de energia nesses sistemas.

1.1 Objetivo

O objetivo principal desta pesquisa é dominar as técnicas de desenvolvimento de um sistema *multi-core on-chip* baseado em hardware reconfigurável, voltado para aplicações de

detecção de pedestres a partir de imagens provindas de quatro câmeras de vídeo. Propõe-se, assim, uma arquitetura de hardware embarcada em dispositivos *Field-programmable Gate Array* (FPGA) para o processamento de algoritmos de detecção de pedestres considerados eficientes pela comunidade científica, os quais tem sido utilizados como base para o desenvolvimento de outros algoritmos. Mais especificamente, utilizam-se os algoritmos *Histogram of Oriented Gradients* (HOG) e *Integral Channel Features* (ICF) como técnicas de detecção.

O processo de domínio dessa tecnologia passa por diversas fases, as quais foram aqui cumpridas: (i) formação de um *background* tanto das técnicas para detecção de pedestres quanto das arquiteturas de hardware utilizadas neste contexto; (ii) elaboração de uma nova proposta de forma a suprir carências ou deficiências identificadas em arquiteturas existentes e incrementar qualidades, buscando equilibrar vantagens e desvantagens do sistema proposto; (iii) projetar e implementar sistema *multi-core* para detecção de pedestres; (iv) validar a arquitetura desenvolvida em dispositivos FPGA; e (v) analisar o comportamento e o desempenho do sistema desenvolvido.

1.2 Justificativa

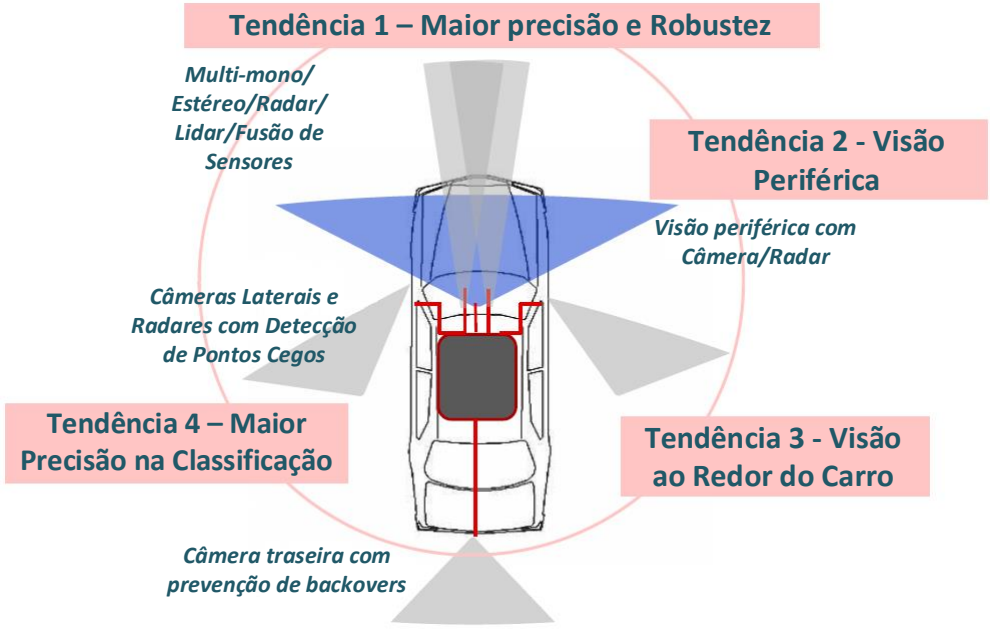
O desenvolvimento de algoritmos para detecção de pedestres tem sido objeto de pesquisa há quase duas décadas, durante a qual muitas técnicas foram propostas. Uma boa revisão das técnicas e dos métodos já criados para este fim encontra-se em Gandhi e Trivedi (2006), Gavrila (2001), Bertozzi *et al.* (2002), Bu e Chan (2005) e Enzweiler e Gavrila (2009), e mais recentemente em Dollar *et al.* (2012), Geronimo *et al.* (2010) e Benenson *et al.* (2014). Contudo, apesar do grande número de publicações em torno deste tema, produções recentes ainda mostram melhoras significativas na precisão da detecção, o que significa que o ponto de saturação ainda não foi atingido (ZHANG *et al.*, 2016).

Os dois algoritmos escolhidos como objeto de estudo neste trabalho são importantes representantes do universo de algoritmos de detecção de pedestres existentes na literatura. No caso do algoritmo HOG, centenas de técnicas foram e ainda têm sido desenvolvidas tendo como base as ideias propostas por Dalal e Triggs (2005). Este algoritmo, que é uma das técnicas pioneiras na detecção de pedestres, serve, ainda hoje como base de comparação para algoritmos mais recentes. Benenson *et al.* (2014) apresentam um *ranking* com 44 métodos de detecção de objetos utilizando o *benchmark* de detecção de pedestres da Caltech (DOLLAR *et al.*, 2009b). Dentre os métodos com menor taxa de erros, pelo menos 75% deles utilizam, de alguma forma, as *features* HOG para detectar objetos em imagens. Além disso, entre os dez métodos melhores colocados no *ranking*, 6 deles utilizam HOG. Estes dados mostram a popularidade, a efetividade e a representatividade deste algoritmo. O segundo algoritmo (ICF) é parte de uma geração mais recente de algoritmos para detecção de pedestres (DOLLAR *et al.*, 2009a) e popularizou a ideia do uso *features* de primeira ordem e segunda ordem com o uso de canais. Este algoritmo é raiz de uma família de algoritmos bem sucedidos (ZHANG *et al.*, 2016) e que também estão no *ranking*

estabelecido por Benenson *et al.* (2014), tendo inclusive influenciado algoritmos de detecção de famílias diferentes. É, portanto, um representante significativo dos detectores de pedestres modernos.

Este trabalho considera como plataforma automotiva um veículo com quatro câmeras de vídeo dispostas conforme ilustrado na Figura 2. A utilização de quatro câmeras permite uma visão bastante abrangente dos arredores do veículo, fornecendo informações importantes sobre possíveis situações de perigo. Esta configuração tem se mostrado uma tendência na implementação de ADAS na indústria com o objetivo de fornecer um nível maior de segurança no trânsito e de viabilizar a existência de veículos autônomos (MODY *et al.*, 2016; MOBILEYE, 2015; NVIDIA, 2016).

Figura 2 – Disposição e uso das quatro câmeras em um veículo.



Fonte: Adaptada de Mody *et al.* (2016).

Como mostrado na Figura 2, o uso da câmera frontal no veículo visa fornecer uma maior precisão e robustez na detecção de diversos tipos de objetos, sendo combinada com outros tipos de sensores para uma maior confiabilidade na prevenção de colisões frontais. As câmeras laterais têm o objetivo de fornecer visão periférica ao condutor, permitindo a detecção de objetos nos pontos cegos dos automóveis. Esta solução é bastante conveniente em países como o Brasil, onde há um grande número de motociclistas em circulação e pedestres realizando travessias em situações de trânsito.

A câmera traseira tem sido bastante utilizada atualmente no auxílio ao estacionamento em marcha à ré. Entretanto, é necessário que este sensor seja utilizado de forma mais ativa, detectando objetos (principalmente pessoas) e emitindo avisos expressos ao condutor sobre a

possibilidade de acidentes. Estatísticas mostram que, nos Estados Unidos, todo ano, cerca de 13.000 pessoas são vítimas de atropelamentos na região traseira dos veículos, sendo 232 fatais (Kids and Cars, 2014). Ainda, estima-se que 50 crianças sejam atropeladas por semana devido à colisão com veículos em marcha à ré, pois o condutor não conseguiu enxergá-las.

Embora o enfoque deste trabalho esteja na detecção de pessoas, outros tipos de objetos também podem ser detectados pelas diversas câmeras, como outros veículos, motociclistas, ciclistas, animais e placas de trânsito, entre outros obstáculos. Dessa forma, o objetivo do uso de quatro câmeras é prover um sistema de detecção mais confiável do que sistemas contendo uma única câmera frontal.

Sistemas de detecção de pedestres baseados em visão necessitam de um grande poder computacional devido, principalmente, ao processamento complexo, em tempo real, sobre uma imensa quantidade de dados (KURODA; KYO, 2007). Além disso, por estarem embarcados em automóveis, tais sistemas devem apresentar um consumo de energia bastante reduzido, pois a alimentação é feita por uma bateria. Junta-se a isso a necessidade de soluções flexíveis que permitam adaptar ou evoluir as aplicações nas plataformas embarcadas de forma a atender à novas demandas e possibilitar uma melhor exploração do espaço de projeto. Neste sentido, dispositivos FPGA fornecem a flexibilidade, o desempenho e níveis adequados de consumo de energia pra o projeto e o desenvolvimento de *Systems-on-chip* utilizando múltiplos núcleos de processamento, aceleradores de hardware e arranjos de memória com o objetivo de atender às demandas de uma aplicação de detecção e pedestres.

Nos últimos anos, muitos estudos tiveram como foco implementações embarcadas em FPGA de algoritmos de detecção de pedestres, principalmente o algoritmo HOG, como por exemplo Advani *et al.* (2015). No entanto, a maioria delas é baseada em implementações em hardware fixo, o que dificulta a adaptação do algoritmo a novos cenários e uma melhor exploração do espaço de projeto. Neste trabalho adota-se um linha diferente destas implementações, valorizando-se a programabilidade por meio do uso de processadores *softcore* e, consequentemente a flexibilidade da solução, como em Kelly *et al.* (2014), sem abdicar de aceleradores de hardware para obter um melhor desempenho. Esta linha de solução embarcada tem sido adotada e promovida em sistemas de detecção recentes, tanto na indústria com soluções ASIC (MOBILEYE, 2015; NVIDIA, 2016) quanto na academia com soluções baseadas em FPGA (KELLY *et al.*, 2014) e ASIC (CLEMONS *et al.*, 2011).

1.3 Organização do Documento

O projeto de pesquisa descrito neste documento está dividido em cinco capítulos. Após a introdução realizada no Capítulo 1, o Capítulo 2 abordará os conceitos básicos existentes em áreas relacionadas a essa pesquisa, como computação reconfigurável, coprojeto de hardware e software e sistemas ADAS para detecção de pedestres. Após essa contextualização teórica o

Capítulo 3 apresentará o estado da arte em algoritmos para detecção de pedestres, assim como as arquiteturas de hardware já desenvolvidas para a execução desses algoritmos.

O Capítulo 4 descreve a implementação das arquiteturas de hardware e software desenvolvidas em duas fases. Na primeira fase a arquitetura tem como alvo o algoritmo HOG e, na segunda fase, o algoritmo ICF. São mostradas também as plataformas de hardware utilizadas para as implementações e validação do sistema. O Capítulo 5 mostra, em detalhes, os resultados alcançados e, por fim, o Capítulo 6 apresenta as conclusões deste trabalho, sugerindo também possibilidades de trabalhos futuros para continuidade da pesquisa desenvolvida.

CAPÍTULO

2

FUNDAMENTAÇÃO TEÓRICA

2.1 Computação Reconfigurável

O surgimento da tecnologia de circuitos integrados permitiu a integração de grandes quantidades de transistores dentro de uma mesma pastilha de silício, também chamado de *chip*. Confirmando a tendência de aumento da densidade, prevista por Gordon Moore (MOORE, 1998), chips atuais contém bilhões de transistores que implementam sistemas complexos, formados por diferentes unidades funcionais e núcleos de processamento.

A numerosa quantidade de aplicações que podem ser atendidas por circuitos integrados motivou o desenvolvimento de diversas arquiteturas de processamento, cada uma delas otimizada para um determinado objetivo. Segundo Bobda (2010), essas arquiteturas podem ser categorizadas em três grupos principais, tendo como critério o grau de flexibilidade. São eles:

- **Processadores de Propósito Geral**: Este grupo de arquiteturas está baseado, principalmente, nas contribuições de John Von Neumann em 1945. Esse matemático demonstrou que um computador poderia ter uma estrutura fixa, relativamente simples e que seria capaz de executar qualquer tipo de cálculo sem a necessidade de mudanças no hardware. Tal estrutura é formada por uma memória que armazena os programas e os dados (no caso da arquitetura Harvard, memórias separadas para dados e programa), uma unidade de controle que gerencia os endereços a serem acessados na memória e uma unidade lógica e aritmética, que executa as instruções de um programa. Cada programa consiste em uma série de instruções executadas sequencialmente, uma após a outra. Diversas técnicas desenvolvidas ao longo do tempo possibilitaram um maior desempenho dessas arquiteturas, tais como *pipelining*, predição de desvio, execução especulativa, etc.

- **Processadores de Domínio Específico**: Se a classe de algoritmos a ser executada for conhecida de antemão, então o processador pode ser modificado para melhor atender a

essas aplicações. O objetivo desses processadores é prover recursos para execução eficiente de um conjunto de operações que caraterizam uma classe de algoritmos. Os representantes mais conhecidos desse grupo são os *Digital Signal Processor* (DSP), ou Processadores Digitais de Sinais. Um DSP é um processador especializado que otimiza o cálculo de tarefas repetitivas e numericamente intensivas. Algumas classes de aplicações que se beneficiam e utilizam tais arquiteturas são telecomunicações, multimídia, processamento de imagens, automóveis, radares, etc. Embora Processadores de Domínio Específico incorporem as características inerentes a uma determinada aplicação, essas arquiteturas ainda adotam a abordagem de execução sequencial, proposta por Von Neumann. Esta última característica, apesar de fornecer um certo grau de flexibilidade na programação, acaba por limitar o desempenho.

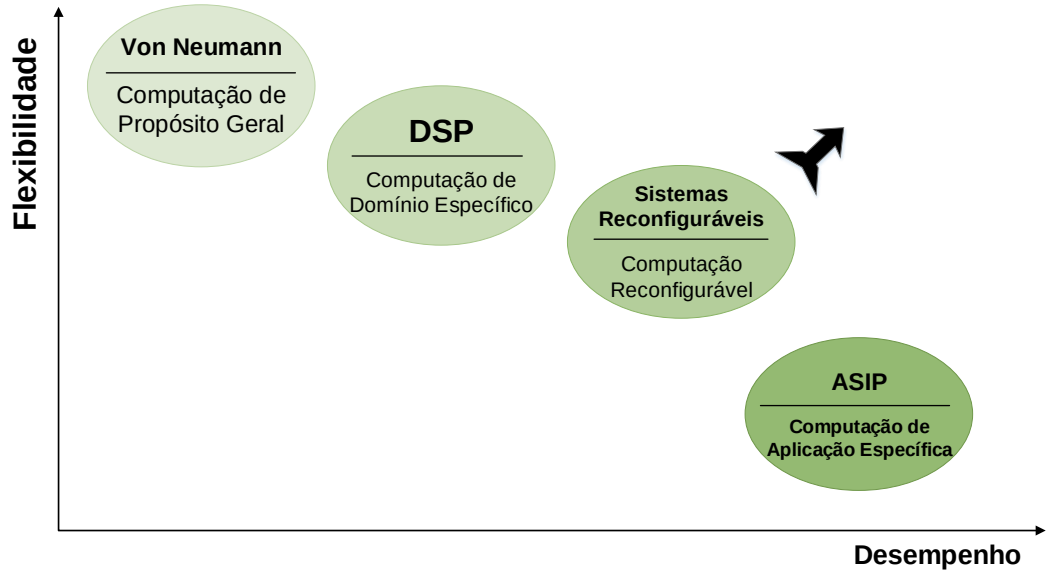
- **Processadores de Aplicação Específica**: Arquiteturas deste tipo baseiam-se na ideia de que, se um processador será utilizado somente para uma aplicação, a qual já é conhecida e é fixa, então a unidade de processamento poderia ser projetada e otimizada para aquela aplicação. Processadores projetados para uma única aplicação são também chamados de *Application Specific Processors* (ASIP), ou Processadores de Aplicação Específica. ASIPs são frequentemente implementados em pastilhas de silício únicas chamadas de *Application Specific Integrated Circuit* (ASIC). Tal processador não pode ser utilizado para outras tarefas além daquela para a qual foi projetado.

Bobda (2010) identifica dois conceitos-chave para a caracterização de processadores: flexibilidade e desempenho. Processadores de propósito geral, baseados na máquina de Von Neumann, apresentam o maior grau de flexibilidade, uma vez que são capazes de executar qualquer tipo de tarefa. Contudo, devido à característica sequencial dessa arquitetura o desempenho fica comprometido. Processadores de Aplicação Específica, por sua vez, fornecem um maior desempenho pois são otimizados para uma determinada aplicação. A flexibilidade deste tipo de arquitetura, contudo, fica prejudicada, pois o conjunto de instruções necessário para executar as aplicações está implementado fisicamente (*hardwired*) dentro do chip.

A Figura 3 apresenta a relação entre flexibilidade e desempenho dos tipos de processadores discutidos. Observa-se que de um lado estão os processadores baseados em Von Neumann e do outro lado estão os ASIPs. Entre os dois tipos encontram-se um grande número de processadores, os quais podem oferecer maiores ou menores flexibilidade e desempenho. A escolha de uma ou outra arquitetura passa primeiro pela definição da aplicação ou do conjunto de aplicações a ser executado. Idealmente, um processador deveria oferecer no mesmo chip, a flexibilidade dos processadores de propósito geral e o desempenho de um ASIP. Tal arquitetura seria capaz de se adaptar à aplicação em tempo real.

Nos anos 1960, Estrin *et al.* (1963) propuseram uma arquitetura de computador formada por um processador e um arranjo de hardware denominado "reconfigurável". Nesta arquitetura, o

Figura 3 – Flexibilidade vs. desempenho de classes de processadores.



Fonte: Adaptada de Bobda (2010).

processador era utilizado para controlar o comportamento do hardware reconfigurável, permitindo que a funcionalidade deste fosse otimizada para uma tarefa específica. Uma vez encerrada a tarefa específica, o hardware poderia ser reutilizado para outras tarefas diferentes. Desde essa época, começou-se a utilizar o termo computação reconfigurável como referência a sistemas que incorporam alguma forma de programabilidade de hardware. Compton e Hauck (2000) definem essa programabilidade como a capacidade de adaptação do hardware à sua utilização mediante o emprego de pontos de controle físico.

Bobda (2010) define o termo Computação Reconfigurável como o estudo da computação utilizando dispositivos reconfiguráveis. Por configuração e reconfiguração deve-se entender que é o processo de mudar a estrutura de um dispositivo em sua inicialização (*start-up time*), ou repectivamente, em tempo de execução. Segundo Hsiung, Santambrogio e Huang (2009), arquiteturas de computação reconfigurável podem ser também descritas como *Hardware-On-Demand*, hardware customizável de propósito geral ou uma abordagem híbrida entre ASIC e Processadores de Propósito Geral.

A estrutura espacial de um dispositivo reconfigurável pode ser modificada de forma a utilizar a melhor abordagem computacional para acelerar uma determinada aplicação. Se houver necessidade de uma nova aplicação ser executada, a estrutura do dispositivo poderá ser modificada novamente para se adaptar a essa nova aplicação. Ao contrário dos computadores baseados na arquitetura de Von Neumann, os quais são programados por um conjunto de instruções executadas sequencialmente, a estrutura dos dispositivos reconfiguráveis pode ser modificada em tempo de compilação ou tempo de execução, bastando carregar um *bitstream* no

dispositivo.

Conforme pode ser visto na Figura 3, sistemas reconfiguráveis fornecem um bom compromisso entre flexibilidade e desempenho, possibilitando a aceleração de aplicações por meio da adaptação do hardware à necessidades computacionais específicas (BONDALAPATI; PRASANNA, 2002). A tecnologia de reconfiguração em dispositivos apresentou uma grande evolução nas últimas décadas. Muita desta evolução é devida à ampla aceitação de dispositivos reconfiguráveis, como os FPGA na construção de sistemas comerciais e científicos.

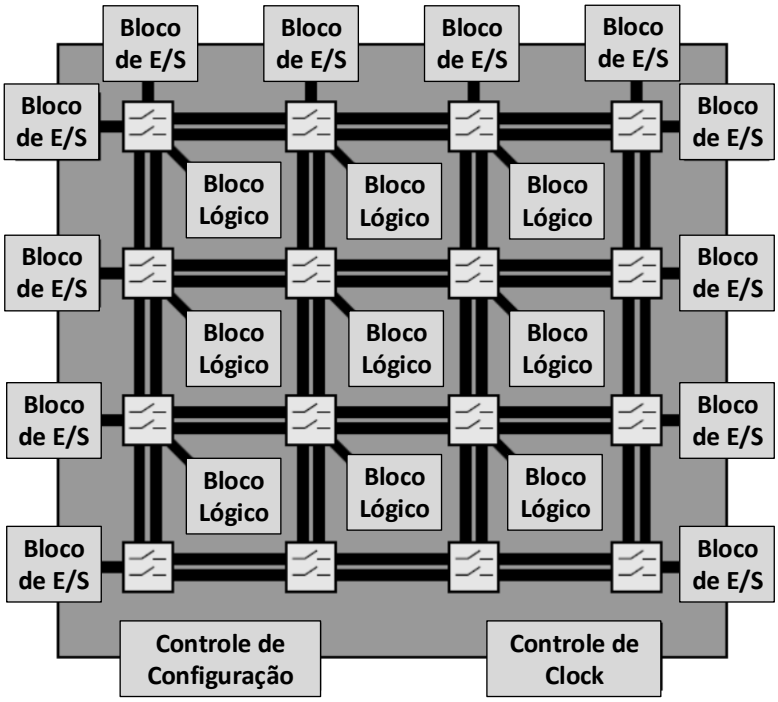
2.1.1 Field-Programmable Gate Arrays

A ideia por trás de um hardware programável é a de um circuito genérico onde a funcionalidade possa ser programada para uma aplicação particular. Computadores convencionais são baseados nesta ideia, na qual a Unidade Lógica e Aritmética (ULA) pode realizar uma de várias operações de cada vez. Assim, um certa aplicação deve ser quebrada em uma sequência de sinais de controle para coordenar as funções da ULA e uma lógica para fornecer os dados de entrada da ULA. A sequência de controle provém de instruções de programa armazenadas na memória do computador. Por outro lado, a lógica programável representa essa funcionalidade como um circuito que pode ser programado para atender aos requisitos de uma aplicação. A distinção principal é que a funcionalidade é implementada como um sistema paralelo, ao invés de sequencial.

Um FPGA é um dispositivo que não implementa a função lógica no método tradicional, por meio portas dedicadas. Cada elemento lógico individual é na verdade uma pequena matriz de memória que é programada diretamente com a função desejada, a partir da tabela verdade (BERGER; BERGER, 2002). Esta matriz de memória está localizada nos blocos lógicos existentes no FPGA. A estrutura geral do FPGA é apresentada na Figura 4, que apresenta três tipos de recursos: (i) os blocos de I/O para conectar os pinos de entrada e saída do dispositivo; (ii) os blocos lógicos dispostos num arranjo bi-dimensional e (iii) um conjunto de chaves de interconexões organizadas como canais de roteamento horizontal e vertical entre as linhas e colunas dos blocos lógicos.

A Figura 4 mostra a estrutura básica e os componentes essenciais de um FPGA genérico. A lógica programável consiste em um conjunto de blocos que são utilizados para implementar a lógica da aplicação. Os blocos lógicos são, geralmente, baseados em uma arquitetura de *Look-up Tables* (LUT), que os permitem implementar qualquer função arbitrária sobre as entradas. Os blocos lógicos estão arranjados em uma estrutura em grade e são interconectados por uma matriz de roteamento programável, que possibilita a interconexão entre os blocos lógicos em configurações arbitrárias. Os blocos de entrada e saída (E/S) realizam a comunicação entre a parte interna do núcleo do FPGA e os dispositivos externos. O roteamento permite que virtualmente qualquer sinal possa ser roteado para qualquer pino de E/S do dispositivo.

Figura 4 – Arquitetura básica de um FPGA.



Fonte: Adaptada de Bailey (2011).

Além disso, a maioria dos FPGAs fornecem alguma forma de sincronização de *clock* para controlar a temporização de um sinal de *clock* relacionado a uma fonte externa. Uma rede de distribuição de *clocks* fornece esses sinais para todas as partes do FPGA. Há ainda uma lógica dedicada ao carregamento da configuração no FPGA. Esta lógica não é incluída diretamente como parte do projeto de um desenvolvedor de hardware, mas é uma carga extra necessária para que os FPGAs sejam programados apropriadamente.

Em aplicações embarcadas, o consumo de potência dos sistemas, em particular daqueles baseados em FPGA, é de grande importância. A potência dissipada por um FPGA pode ser dividida em dois componentes: potência estática e potência dinâmica. A potência dinâmica é aquela necessária para chavear sinais do nível lógico 0 para o nível lógico 1 e vice-versa. Já a potência estática é aquela que é consumida quando não há o chaveamento de sinais e ocorre simplesmente em virtude do dispositivo estar ligado. A maioria dos FPGAs é baseado na tecnologia CMOS, cujos circuitos tipicamente têm uma dissipação de potência estática muito baixa. Qualquer potência significativa somente é dissipada quando as saídas mudam seus estados. Essa potência é dada pela equação

$$P = NCV_{DD}^2 f \tag{2.1}$$

onde N é o número médio de saídas que estão em mudança em cada ciclo de *clock*, C é a

capacitância média em cada saída e f é a frequência do *clock*. O consumo de energia em sistemas embarcados pode ser reduzido tomando-se ações sobre cada um dos itens da Equação 2.1 (BAILEY, 2011). As ações incluem limitar o número de saídas que mudam seu estado em cada ciclo de *clock*, minimizar o *fan-out* de cada porta e o uso de fios longos (diminuindo a capacitância), reduzir a voltagem da fonte de potência (quando possível) e reduzir a frequência de *clock*. Esta última tem o efeito mais significativo na diminuição do consumo de energia, uma vez que é o sinal de *clock* que dissipa a maior parte da potência. Isto porque esse sinal alterna seu estado a cada ciclo de *clock* e sua rede de distribuição estende-se por todo o FPGA, o que lhe confere uma alta capacitância. Outro fator que afeta a potência dissipada por um FPGA é o projeto do usuário. Nele, a eficiência no consumo consiste em minimizar o número de sinais que transicionam a cada ciclo de *clock*. Contudo, muitas das características citadas são gerenciadas automaticamente pelas ferramentas de *place & route*, o que limita o controle do desenvolvedor a algumas configurações de mais alto nível.

2.2 Coprojeto de Hardware e Software

O coprojeto de hardware e software consiste em combinar componentes de hardware e componentes de software de forma que estes cooperem para atingir os objetivos do projeto de um sistema computacional. Uma definição para o termo é dada por Schaumont (2013), na qual "Coprojeto de hardware e software é o particionamento e o projeto de uma aplicação em termos de componentes fixos e flexíveis". Embora 'componente fixo' geralmente se refira a componentes de hardware e 'componente flexível' relacione-se à componentes de software, a definição não é restrita, principalmente quando se consideram dispositivos de hardware que proveem certos níveis de flexibilidade, como os FPGAs.

Desempenho e eficiência energética são, geralmente, os fatores que conduzem as decisões por implementações em hardware ou software. Contudo, Schaumont (2013) aponta alguns *trade-offs* que tornam conflituosa a escolha entre hardware e software. A favor das escolhas para implementações em hardware encontram-se fatores como:

- **Desempenho:** A especialização da arquitetura e o uso de aceleradores levam a uma melhor relação de trabalho feito por ciclo de *clock*.
- **Eficiência Energética:** A importância desse fator se deve à ampla popularização de aparelhos eletrônicos alimentados por bateria. Embora tais dispositivos executem muitas das aplicações utilizadas em ambientes de alto desempenho ou *desktop*, mover parte das implementações em software para hardware fixo pode aumentar a eficiência energética da aplicação.
- **Densidade de Potência:** A dissipação de potência aumenta com a frequência de *clock* e em muitos dispositivos a capacidade de arrefecimento já se encontra no limite dos

custos. O investimento em arquiteturas de hardware paralelas permite um aumento de desempenho com frequências de operação menores, dissipando assim, uma quantidade menor de potência.

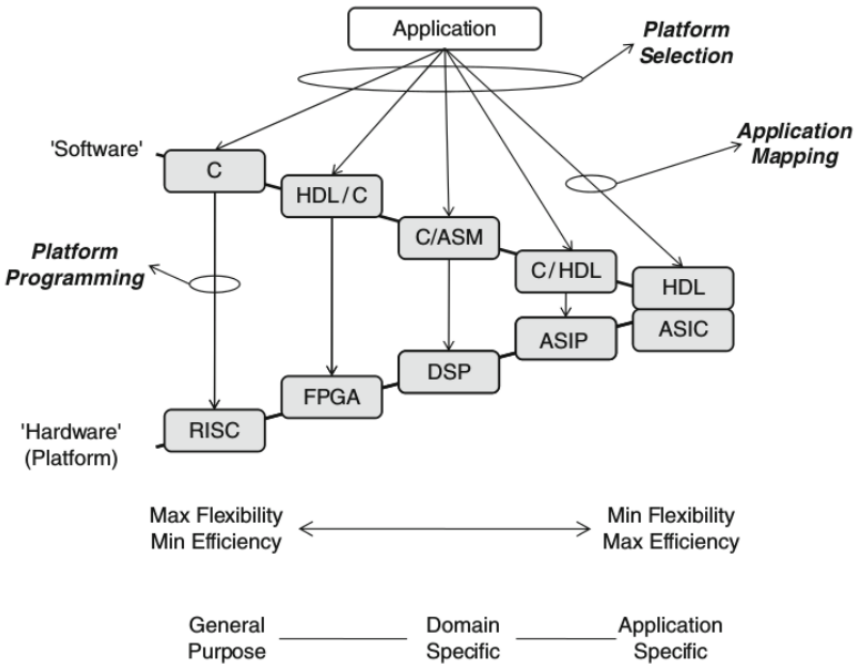
Se, por um lado, os fatores apresentados contribuem para uma escolha pelo hardware, outros fatores são argumentos para um maior uso de software:

- **Complexidade do Projeto:** Sistemas eletrônicos modernos estão a cada dia mais complexos, o que torna a sua implementação em hardware fixo cada vez mais difícil. Ao invés disso, implementações mais flexíveis têm sido adotadas, utilizando processadores programáveis e permitindo lidar mais facilmente com futuras necessidades dos sistemas.

- **Custo de Projeto:** O projeto de novos *chips* é bastante caro e, portanto, empresas buscam fabricar *chips* programáveis de forma a permanecerem um maior tempo no mercado, tendo seu produto reutilizado de diferentes maneiras. Nesse contexto encontram-se também os FPGAs, que permitem o reuso por meio da reprogramação.

- **Cronogramas de Projetos Encolhidos:** A velocidade com que novas tecnologias substituem as antigas é cada vez maior, assim como a sua complexidade. Desta forma, o ciclo de desenvolvimento destas tecnologias é cada vez mais curto, o que acarreta em desvantagem para a implementação de sistemas de hardware.

Figura 5 – Espaço de coprojeto de hardware e software.



Fonte: Schaumont (2013).

Em meio a esses fatores, torna-se complexo encontrar um equilíbrio no contexto de um espaço de projeto. A Figura 5 mostra uma representação das atividades realizadas no espaço de projeto a fim de encontrar uma solução que combine hardware e software. O objetivo de um projeto passa por mapear a aplicação em uma plataforma, podendo isto significar a escrita de um software ou a customização de um hardware. Na Figura 5, as plataformas estão organizadas, da esquerda para a direita, de acordo com sua flexibilidade. As consideradas mais flexíveis são as RISC e FPGAs. Plataformas específicas, como ASIC, são otimizadas para uma só aplicação e encontram-se no outro extremo da comparação de flexibilidade. No meio, encontram-se plataformas de domínio específico, as quais podem executar conjuntos de aplicações de um determinado domínio, como criptografia, processamento de sinal digital, entre outros.

A flexibilidade tratada na Figura 5 significa o quão bem uma plataforma pode ser adaptada a diferentes aplicações, o que pode ser interessante mesmo depois que a plataforma já foi fabricada. Neste caso, os FPGAs se apresentam como uma solução bastante flexível, pois podem ser programadas por linguagens de propósito geral para descrição do hardware ou ainda podem conter instâncias de processadores que executam códigos em C, por exemplo. A eficiência pode estar tanto relacionada ao desempenho absoluto quanto ao uso de energia para implementação dos cálculos.

O desafio do coprojeto de hardware e software é combinar dois paradigmas de projeto bastante diferentes. A Quadro 1 mostra, de forma resumida, algumas diferenças fundamentais entre os projetos de hardware e software.

Quadro 1 – O dualismo do projeto de hardware e de software.

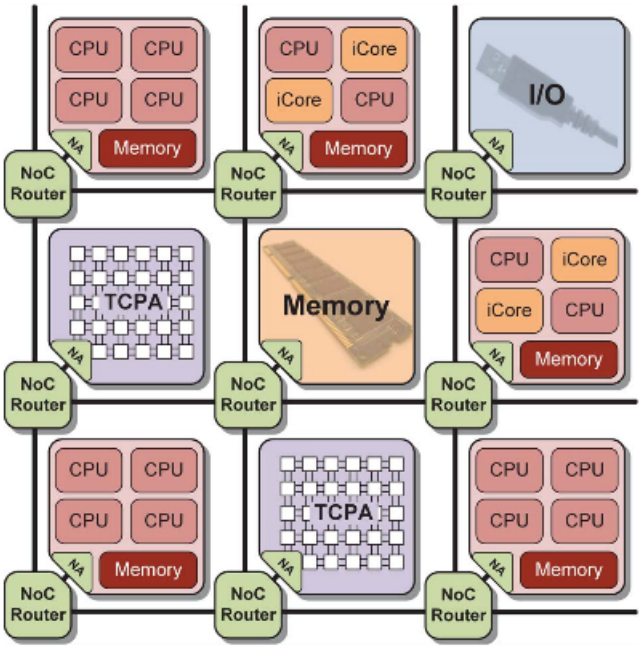
	Hardware	**Software**
Paradigma de projeto	Decomposição no espaço (portas)	Decomposição no tempo (Instruções)
Custo do recurso	Área (número de portas)	Tempo (número de instruções)
Flexibilidade	Deve ser projetada	Implícita
Paralelismo	Implícito	Deve ser projetado
Modelagem	Modelo $\neq$ implementação	Modelo $\sim$ implementação
Reuso	Incomum	Comum

Fonte: Schaumont (2013).

Atualmente, a tecnologia de *Systems-on-a-chip* heterogêneos é uma realidade graças aos avanços da microeletrônica. Em um sistema complexo encontram-se diversos tipos de processadores, DSPs, processadores de propósito específico, aceleradores de hardware customizáveis, blocos analógicos, periféricos e diversos tipos de memória. Esta complexidade reflete-se também no nível de comunicação destes sistemas. Por exemplo, em um automóvel moderno podem ser encontradas de 70 a 90 Electronic Control Units (ECU), provendo as mais diversas funcionalidades. De forma até mais rápida, a complexidade do software em sistemas embarcados

também foi afetada. Em um veículo, estima-se que mais de 100 milhões de linhas de código são executadas. Consequentemente, também crescem as complexidades de verificação e teste de tais sistemas (TEICH, 2012). Tais informações mostram a dificuldade e o desafio da realização de um coprojeto de hardware software nos sistemas atuais. A Figura 6 ilustra um exemplo geral de um Multi-Processor System-on-Chip (MPSoC) heterogêneo, no qual podem haver diferentes níveis de integrações de hardware e software.

Figura 6 – Diagrama de um modelo de *MPSoC* heterogêneo com diferentes CPUs, arranjos de processadores fortemente acoplados (TCPAs) e memórias interconectadas por uma *Network on-Chip*.



Fonte: Teich (2012).

2.3 Sistemas de Auxílio ao Condutor

A indústria automobilística tem despendido grandes esforços no desenvolvimento de tecnologias que promovam o aumento da segurança no trânsito. Apesar de muitos avanços terem sido obtidos nos últimos anos, algumas iniciativas pioneiras não são recentes, como as ocorridas no Japão nos anos 1980. Contudo, as tecnologias disponíveis na época e seus altos custos limitavam a extensão dessas iniciativas. Atualmente, centenas de projetos, provindos tanto do meio industrial quanto acadêmico, têm se concentrado em desenvolver diferentes aspectos de sistemas de segurança para o trânsito. Estes foram iniciados em vários lugares do mundo, como Europa, Estados Unidos da América, Austrália, China e Coréia do Sul (BISHOP, 2005).

Dentro dos vários projetos existentes, diferentes termos têm sido empregados para designar os sistemas que auxiliam na segurança do trânsito. Por vezes, a definição desses termos se sobrepõe, sendo os principais:

- ***Intelligent Transport Systems* (ITS)**: Sistemas de Transportes Inteligentes compreendem tecnologias que estão baseadas em informações provenientes tanto do veículo e quanto do trânsito. Nesses sistemas, as interações veículo-trânsito ou veículo-veículo auxiliam o condutor e/ou o gerenciamento do tráfego. No contexto de ITS, quando a referência é o veículo, duas subdivisões são encontradas: IVIS e ADAS (LINDER *et al.*, 2007).

- ***In-Vehicle Information Systems* (IVIS)**: Sistemas de Informação Internas ao Veículo são sistemas que interagem com o condutor e induzem tarefas que não são diretamente relacionadas com a tarefa de dirigir nos níveis tático e operacional. Essas tarefas adicionais são chamadas de tarefas secundárias e podem interferir nas tarefas primárias. (JOHANSSON *et al.*, 2004).

- ***Advanced Driver Assistance Systems* (ADAS)**: Sistemas Avançados de Assistência ao Condutor referem-se a sistemas que interagem com o condutor do veículo, auxiliando-o nos níveis tático e operacional (JOHANSSON *et al.*, 2004). Tais sistemas procuram reduzir a ocorrência de erros humanos por meio da observação do ambiente de condução do veículo e do aviso de situações potenciais de perigo.

- ***Intelligent Vehicle* (IV)**: Veículos Inteligentes compreendem os sistemas que trabalham com a percepção do ambiente rodoviário. Eles atuam disponibilizando informações ou controlando o veículo para auxiliar o condutor de forma que se obtenha uma operação ótima do veículo. Sistemas IV operam no nível tático do ato de dirigir (acelerar, frear, mudar de direção), em detrimento das decisões estratégicas tal como escolha de rota, as quais devem ser providas por um sistema de navegação *on-board* (BISHOP, 2005).

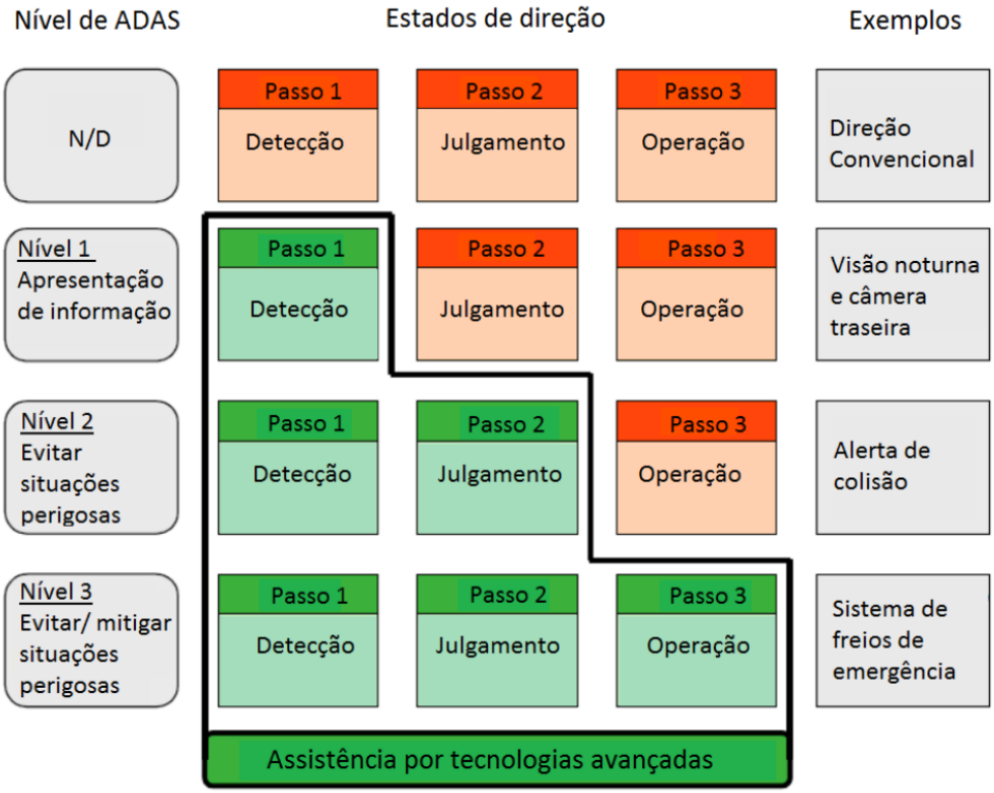
- ***Intelligent Vehicle Safety Systems* (IVSS)**: Sistemas de Segurança Inteligentes para Veículos são sistemas inteligentes que utilizam a tecnologia da informação moderna para evitar colisões e prevenir lesões. Além disso, esses sistemas buscam aperfeiçoar os veículos quanto às suas capacidades de aderência à estrada e resistência ao choque (ABELE *et al.*, 2005).

- ***Pedestrian Protection Systems* (PPS)**: Sistemas de Proteção ao Pedestre buscam detectar a presença de pedestres, estacionários ou em movimento, em uma área específica de interesse ao redor de um veículo em movimento, com o intuito de alertar o condutor, executar ações de acionamento dos freios e ativar airbags externos no caso de uma colisão ser inevitável. (GERONIMO *et al.*, 2010). Sistemas PPS formam um subconjunto de ADAS para a segurança no trânsito.

Este projeto tem como foco os sistemas classificados como ADAS, os quais são também conhecidos como *Driver Support System* (DSS) (AKHLAQ *et al.*, 2012). O propósito de um ADAS é diminuir a carga de trabalho atribuída ao condutor de um veículo, proporcionando um foco maior na pista, facilitando assim a condução e reduzindo o número de acidentes.

Segundo IHRA (2011), sistemas ADAS podem interagir com o condutor em diferentes níveis de assistência e auxiliá-lo em diferentes estados de direção, como ilustrado na Figura 7. No nível N/D não há sistemas ADAS presentes, portanto o condutor é responsável por executar todas as atividades, como detectar elementos do trânsito, julgar possíveis riscos, operar o veículo e etc. No nível um (1), os sistemas atuam fornecendo informações importantes detectadas no ambiente ao redor do veículo por meio de seus sensores.

Figura 7 – Modelo comportamental e nível de assistência ao motorista.

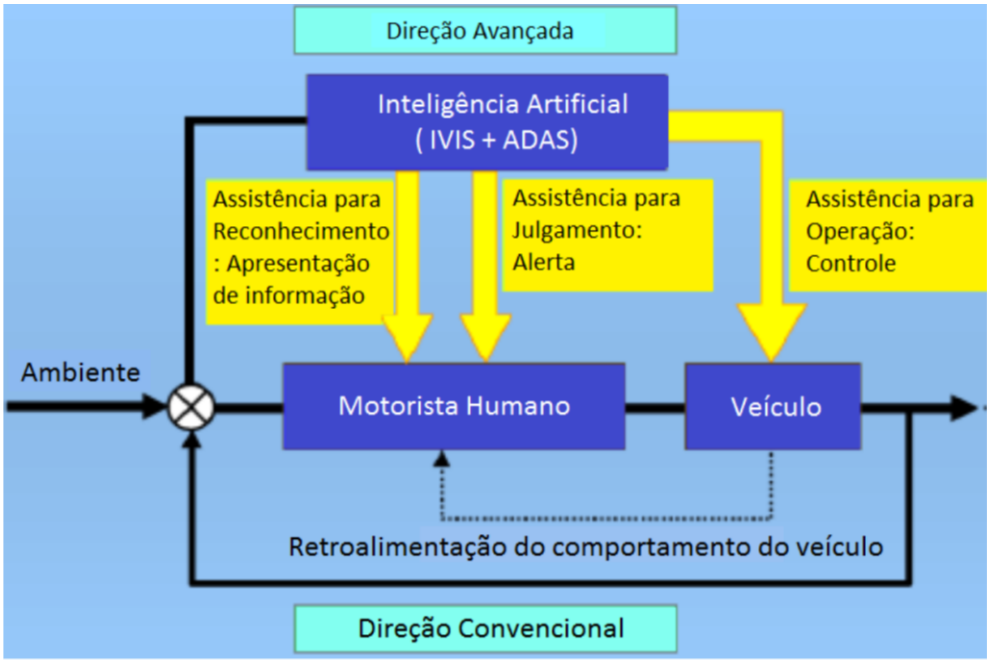


Fonte: Adaptada de IHRA (2011).

No nível dois (2), o sistema ADAS processa as informações provindas dos sensores e apenas auxilia o condutor na avaliação de situações críticas. No nível três (3), o sistema ADAS é capaz de controlar o veículo automaticamente (de forma parcial ou total) a fim de minimizar danos ou escapar de situações de perigo. O diagrama representado na Figura 8 mostra a forma como um sistemas ADAS influencia no padrão de comportamento da condução convencional. A atuação dos sistemas de assistência avançada abrange tanto o comportamento do condutor humano quanto o funcionamento do próprio veículo.

Sistemas ADAS utilizam sensores que captam dados do ambiente ao redor do veículo. Esses dados podem incluir velocidade, distância, posição relativa, direção do movimento, tamanho e tipo dos veículos e objetos presentes na pista (AKHLAQ *et al.*, 2012). Para essa finalidade, diversas tecnologias de sensores estão disponíveis, tais como RADAR, LIDAR, SONAR, *Global*

Figura 8 – Interação de um sistema ADAS com o comportamento habitual de condução em um veículo.



Fonte: Adaptada de Burns (2005).

Positioning System (GPS) e câmeras de vídeo. A Tabela 1 mostra uma comparação entre os sensores mencionados.

Os sensores mencionados fornecem uma grande riqueza de informações, motivando a construção de uma grande variedade de aplicações ADAS. Cada uma delas tem sido aperfeiçoada tanto por meio de pesquisas desenvolvidas no meio industrial quanto no meio acadêmico. Aplicações, como visão noturna, detecção de sonolência do condutor, detecção de pontos cegos e detecção de colisão já são empregadas em carros de luxo. Outras, como assistência ao estacionamento, têm sido incorporadas em veículos de menor custo, o que indica uma crescente popularização dessas tecnologias. A Tabela 2 mostra diversas aplicações de ADAS.

ADAS atuam em várias frentes com o objetivo de alertar o condutor e evitar acidentes. Cada uma dessas frentes considera um aspecto diferente da proteção ao condutor. Por exemplo, sistemas de detecção de sonolência atuam no ambiente interno ao veículo, monitorando sinais do estado de consciência do motorista e procurando mantê-lo alerta. Sistemas de alerta de perigo utilizam a infraestrutura de comunicação local ou até mesmo a comunicação veículo-veículo para transmitir e receber dados sobre o tráfego da região. Outros sistemas buscam detectar veículos e objetos ao redor do veículo a fim de evitar colisões. Dentre os objetos a serem detectados estão os chamados *Vulnerable Road Users* (VRU), traduzido livremente para Usuários Viários Vulneráveis, que englobam ciclistas, motociclistas e pedestres. Sistemas criados para garantir a segurança de pedestres são o foco deste trabalho e serão vistos com maiores detalhes.

Tabela 1 – Tecnologias de sensores para ADAS.

	RADAR	**SONAR**	**LIDAR**	**GPS**	**Câmera**
Tecnologia	Ondas de rádio (300 MHz - 30 GHz)	Ondas de som (infrasom e ultrasom)	Laser ou luz infravermelha	24-32 satélites transmitem a localização como sinais de rádio	Sensores de imagem (CCD ou CMOS)
Medidas	Velocidade, distância e direção	Velocidade, distância, direção e tamanho	Velocidade, distância e formato	Velocidade, direção,posição relativa	Velocidade, distância, posição relativa, direção, tamanho e tipo
Alcance aproximado	150m	2,13m	25km	Disponibilidade Global	25m (depende da câmera)
Métodos /princípios	Análise de deslocamento Dopler, *Time of flight*	Análise de deslocamento Dopler, *Time of flight*	Efeito Dopler, *Time of flight*, etc.	Triangulação	Processamento de imagem, Reconhecimento de padrões, etc.
Prós	Confiabilidade, precisão e capacidade de funcionar em qualquer condição ambiental	Capacidade de funcionar tanto sob a água quanto na superfície	Precisão, baixo custo, reconhecimento de objetos e visão noturna	Disponibilidade global	Baixo custo, alta disponibilidade, diversas aplicações, utilizável como um *add-on*, pode ser combinado com outros sistemas, fácil de manejar
Contras	Alto custo, absorventes, objetos fantasma, incapacidade de distinguir objetos, campo de visão limitado ($\leq$ 16), baixa resolução lateral pode encontrar um alvo incorreto	Absorvente de som e medidas imprecisas de distância durante rajadas de vento, neve, chuva, etc.	Interferência da iluminação dos arredores, inviabilidade para condições de tempo ruins, campo de visão limitado e degradação do desempenho devido a reflexões da neve	Desempenho ruim em áreas urbanas com edificações altas, precisão baixa, disponibilidade seletiva devido ao controle militar norte-americano, custo alto e tempo de inicialização relativamente longo	Dependência das condições do tempo, menor precisão, e tecnologia imatura

Fonte: Adaptada de Akhlaq *et al.* (2012).

Tabela 2 – Aplicações de ADAS

Aplicação	Definição	Sensores
Visão Noturna	Visa melhorar a percepção do condutor em situações de baixa visibilidade.	câmeras de infravermelho
Aviso de Saída de pista	Atua na extrapolação de limites de distância ou tempo para cruzar a pista. Ativa o algoritmo de predição de saída de pista do sistema, acionando sinais acústicos, ópticos ou hápticos para alertar o condutor.	câmeras de vídeo
Aviso de Colisão próxima	Visa detectar outros veículos muito próximos e alertar o condutor da possibilidade de colisão	RADAR, LIDAR e sensores visuais
Aviso de Curvas e Limite de Velocidade	Informa ao condutor sobre limites de velocidades e a velocidade recomendada em curvas. Pode ter o auxílio de mapas digitais, comunicação entre veículos e infraestrutura de trânsito.	Sensores visuais e hápticos
Assistente para Manter Pista	Detecta a pista e alerta o condutor sobre mudanças de trajetória, podendo agir ativamente na direção do veículo.	Câmeras de vídeo
Aviso de perigos locais	Caso ocorram situações de perigo fora do campo de visão do condutor, o sistema gera alertas. Utiliza sistemas de comunicação.	Rádio
Assistente de mudança de pista	Gera alertas ao condutor antes e durante o processo de mudança de pista. Pode haver intervenção do sistema para mudança de trajetória do veículo.	Câmera de vídeo
Aviso de colisão em obstáculos	Alerta o condutor com sinais acústicos e/ou visuais no caso de uma colisão em potencial. Ações de evasão ou frenagem tomadas.	Câmera de vídeo, RADAR, LIDAR
Prevenção de obstáculos	Estende a funcionalidade do aviso de colisão em obstáculos adicionando uma intervenção autônoma sobre veículo.	Câmera de vídeo, RADAR, LIDAR
Controle de cruzeiro adaptativo	Diminui automaticamente a velocidade ao se aproximar outro veículo à frente e acelera quando o tráfego permite.	Câmera de vídeo, RADAR, LIDAR
Reconhecimento de sinais de trânsito	Identifica qualquer sinal de trânsito e avisa o condutor para agir adequadamente.	GPS, radio, Câmera de vídeo
Detecção de ponto Cego	Ajuda a evitar acidentes durante mudança de pista, quando outros veículos estão presentes no ponto cego.	SONAR, RADAR, LIDAR, câmera de vídeo
Reconstrução do ambiente	Identifica todos os objetos vizinhos ao veículo e reporta informações sobre velocidade, distância, direção, tamanho e tipo do objeto.	Câmera, infravermelho, RADAR e LIDAR
Detecção de Pedestres	Identifica humanos caminhando na pista ou próximo a ela, alertando o condutor para evitar a colisão.	SONAR, RADAR, LIDAR, câmera
Detecção de sonolência do condutor	Detecta se o condutor está sonolento ou dormindo e o acorda para evitar acidentes.	Sensores de stress, batimentos cardíacos, pressão sanguínea, temperatura e câmera
Assistência ao Estacionamento	Auxilia na prevenção de colisão ao estacionar o veículo. Podem assumir o comando do volante e estacionarem ativamente o veículo. Outros fornecem uma visão dos arredores e emitem alertas na caso de possíveis colisões.	Câmera, SONAR, RADAR, LIDAR

Fonte: Elaborada pelo autor.

2.3.1 Sistemas de Proteção ao Pedestre

Sistemas ADAS desempenham um papel importante ao melhorar a atenção e desempenho dos condutores, fornecendo informações relevantes quando necessário. ADAS são utilizados para monitorar ativamente o ambiente de condução, produzindo avisos e assumindo o controle em situações de grande perigo.

Um dos tipos particulares de ADAS são os Sistemas de Proteção ao Pedestre ou PPS - do inglês *Pedestrian Protection System*. O objetivo de um PPS é detectar a presença tanto de pessoas estáticas quanto em movimento, em uma área específica de interesse ao redor de um veículo que contém tal sistema (*host*). Essa detecção permite alertar o condutor do veículo, acionar a frenagem e até mesmo abrir *airbags* externos, caso uma colisão seja inevitável. Ainda, caso os arredores do pedestre sejam devidamente captados, seria possível tomar ações evasivas.

2.3.1.1 Motivação para o Desenvolvimento de PPS

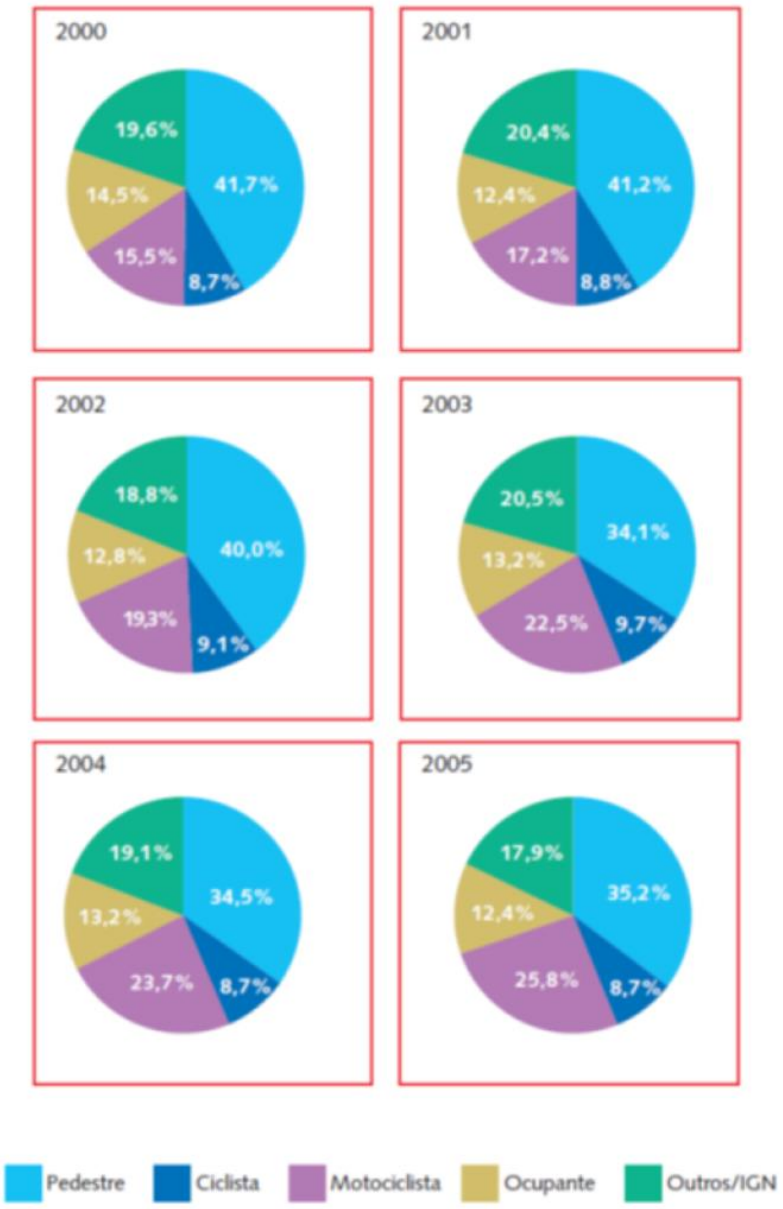
No último século, houve uma enorme popularização dos automóveis, principalmente em países em desenvolvimento como o Brasil, que passou por um grande crescimento econômico. Junto a essa popularização, houve um aumento significativo do número de acidentes de trânsito, que se tornaram uma das principais causas de fatalidades.

Em um estudo feito por Jorge e Koizumi (2008), foram analisados os acidentes ocorridos no Brasil entre os anos de 1998 a 2005 e notou-se uma tendência de crescimento no número de acidentes com vítimas no trânsito urbano. Houve também um crescimento no número de internações de vítimas, o que gera um impacto grande no orçamento dos governos, uma vez que muitos dos atendimentos e cirurgias em hospitais são pagos com dinheiro público. Em 2005, o número de acidentes com vítimas foi de 383.371. Em média, houve 1406 acidentes por dia e 1369 vítimas por dia. O número de mortes por acidentes de trânsito, em 2005, no Brasil, foi de 35.763, o que corresponde à média de 98 mortes por dia. Dentre as vítimas mais afetadas estão os pedestres, conforme pode ser observado nos gráficos da Figura 9.

Em 2005, 35% das internações estiveram relacionadas a acidentes com pedestres. Ainda no mesmo ano, o número de mortes de pedestres em acidentes de trânsito correspondeu a cerca de 30% da quantidade total de vítimas fatais (conforme Figura 10). Nota-se nesse gráfico uma pequena diminuição no número de mortes de pedestres ao longo do tempo. Isso ocorreu devido à contínua adoção de políticas de segurança no trânsito e medidas punitivas, como as multas por excesso de velocidade. Contudo, as estatísticas ainda reforçam a necessidade de criação de mecanismos de proteção ao pedestre.

A fim de construir sistemas de proteção ao pedestre mais efetivos, convém estudar também a forma como esses acidentes acontecem. Gavrila, Marchal e Meinecke (2003) realizaram um estudo dentro do projeto SAVE-U, identificando os principais cenários de acidentes de trânsito ocorridos na Europa e Estados Unidos da América no final da década de 1990 e início

Figura 9 – Internações por lesões decorrentes de acidentes de trânsito por tipo de vítima. Brasil, de 2000 a 2005.



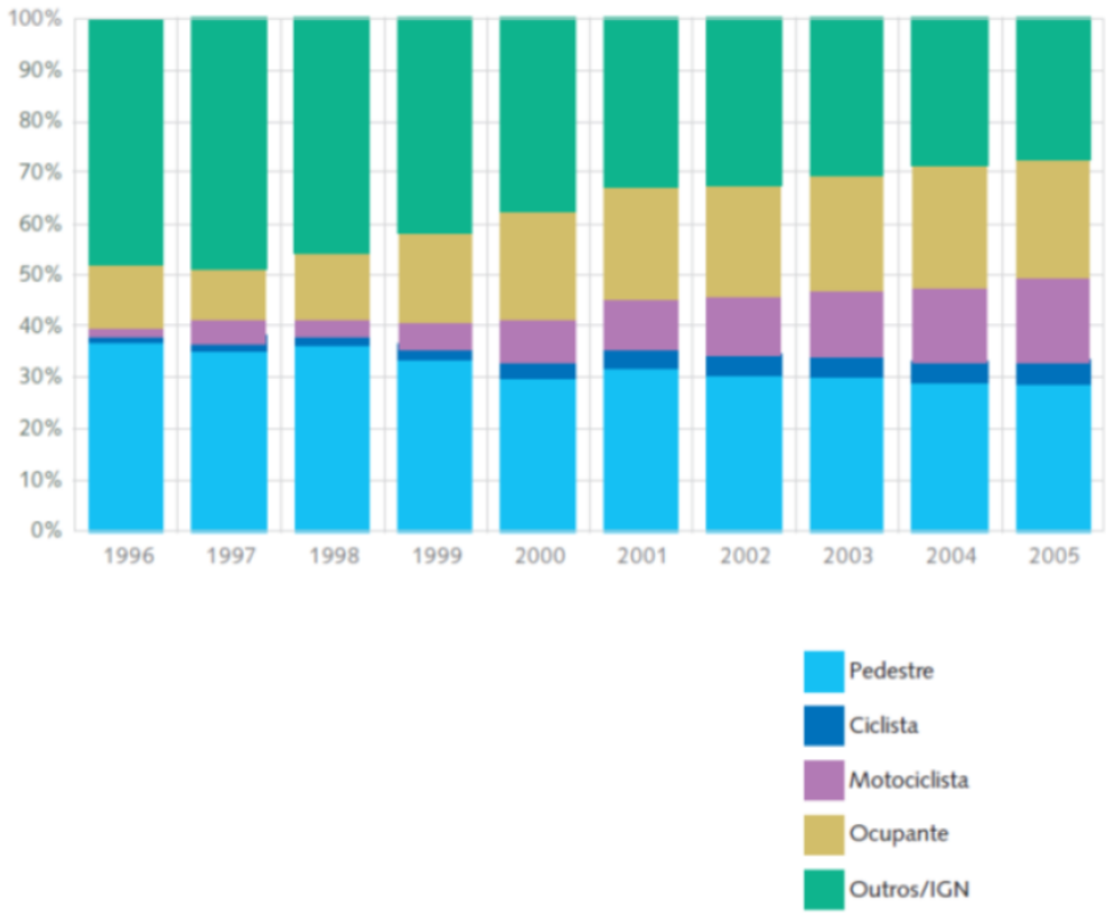
Fonte: Jorge e Koizumi (2008).

da década de 2000. Os dados estatísticos apresentados na sequência são retirados desse estudo.

Para a identificação do cenário, é necessário, primeiramente, determinar em qual ambiente há uma maior incidência de acidentes. A Figura 11 mostra a distribuição dos acidentes de trânsito por tipo de usuário vulnerável e se estes ocorreram na área urbana ou na área rural. Nota-se que a maior parte destes acidentes ocorre em área urbana. Para pedestres, que são o foco deste trabalho, o índice ficou em 93,6%.

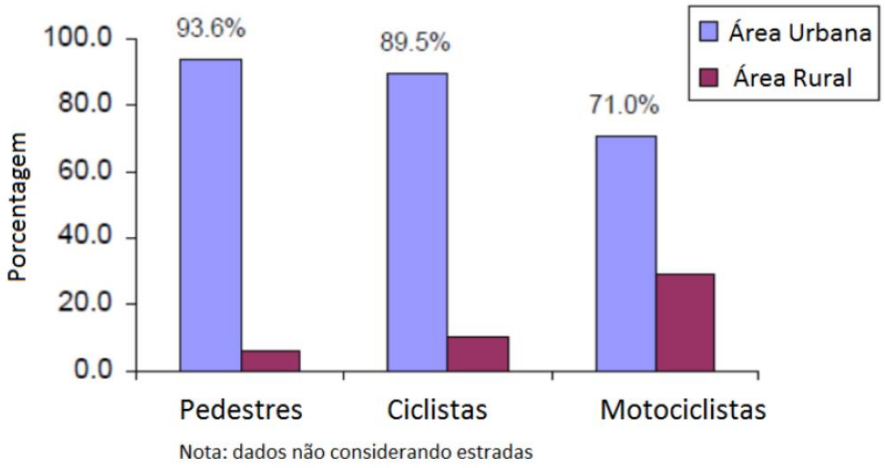
Segundo o estudo desenvolvido por Gavrila, Marchal e Meinecke (2003), em áreas

Figura 10 – Mortes decorrentes de acidentes de trânsito por tipo de vítima. Brasil, de 1996 a 2005.



Fonte: Jorge e Koizumi (2008).

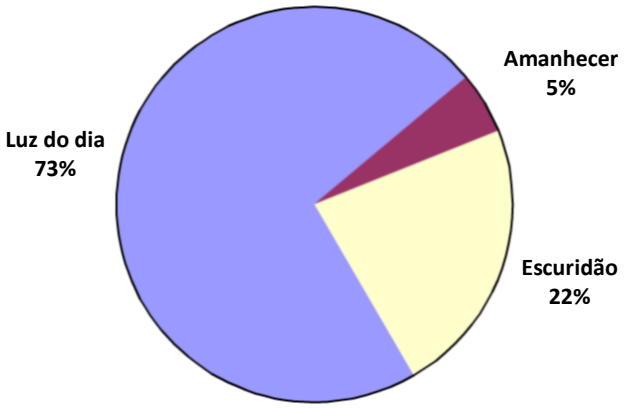
Figura 11 – Distribuição de acidentes de trânsito por área de ocorrência.



Fonte: Adaptada de Gavrila, Marchal e Meinecke (2003).

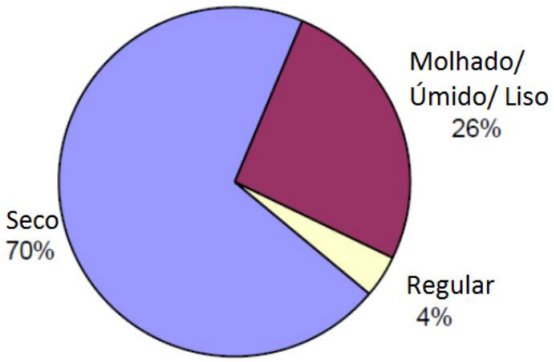
urbanas o maior número de acidentes tende a se concentrar durante o período de luz do dia. O gráfico da Figura 12 mostra que esse número corresponde a 73% da quantidade total de acidentes. Entretanto, o estudo alerta que os acidentes ocorridos durante o período da noite podem ter uma severidade maior do que os que acontecem durante o dia. O estudo também analisa as condições do tempo em que ocorrem os acidentes, conforme pode ser visto na Figura 13. Embora o tempo úmido/molhado/liso tenha uma porcentual significativo do número de acidentes (26%), é no tempo seco em que eles mais acontecem (70%).

Figura 12 – Acidentes de trânsito com injúria em área urbana por condições de iluminação.



Fonte: Adaptada de Gavrila, Marchal e Meinecke (2003).

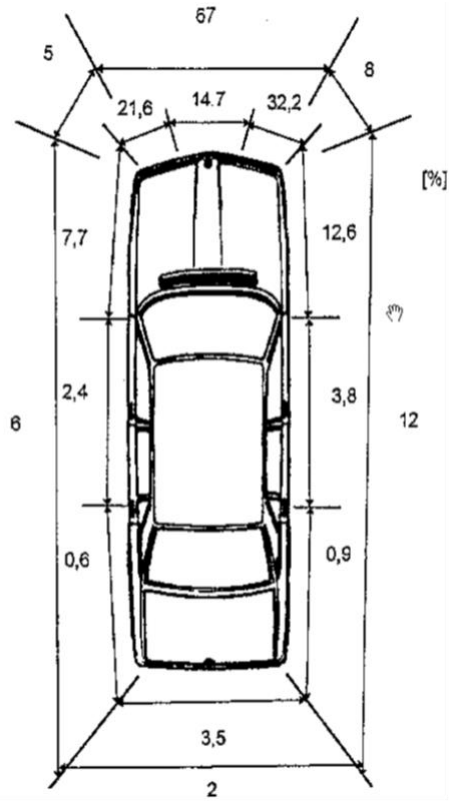
Figura 13 – Acidentes de trânsito em área urbana por condições do tempo.



Fonte: Adaptada de Gavrila, Marchal e Meinecke (2003).

A Figura 14 mostra a incidência dos primeiros contatos em casos de acidentes no trânsito entre pedestres e veículos por zona no veículo. Nota-se que a maior parte dos eventos envolve a parte frontal do veículo. Geronimo *et al.* (2010) confirmam e atualizam essa estatística, indicando que, na União Européia, cerca de 70% dos pedestres vítimas de acidentes com carros foram atingidos pela parte frontal do veículo. Os autores acrescentam ainda que em torno de 90% desses pedestres estavam em movimento.

Figura 14 – Zona de primeiro contato entre pedestre e veículo.



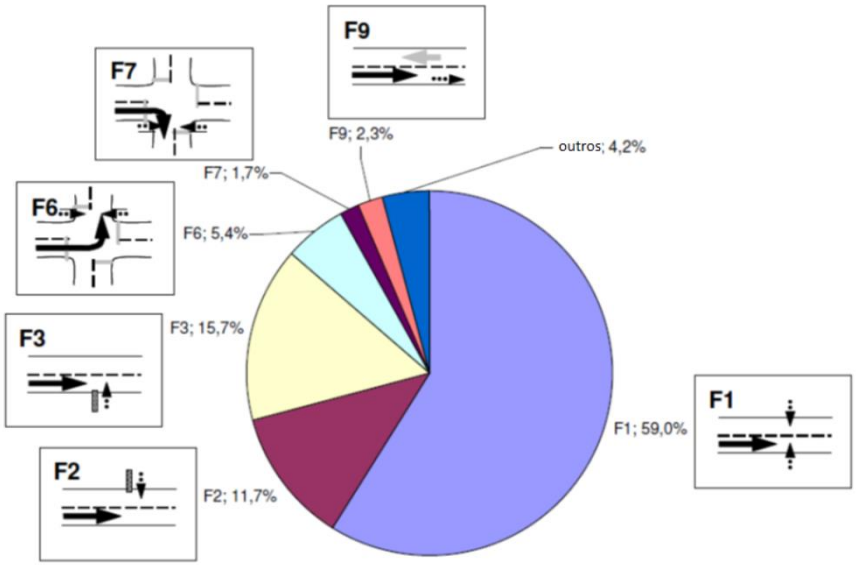
Fonte: Gavrila, Marchal e Meinecke (2003).

Um estudo realizado por Lange (2007) analisou acidentes envolvendo o impacto frontal de veículos com pedestres. Como resultado, o estudo obteve diferentes cenários envolvendo pedestres cujas casualidades tenham sido graves, exigindo vários dias de tratamento em hospital. O resultado dessa pesquisa está ilustrado na Figura 15, na qual pode-se notar que a maior parte (86,4%) dos acidentes de trânsito frontais envolvendo pedestres ocorre durante a travessia de vias retilíneas (cenários F1, F2 e F3).

O estudo feito por Gavrila, Marchal e Meinecke (2003), aponta ainda que a maioria das velocidades de colisão em acidentes envolvendo veículos e pedestres em ambientes urbanos concentra-se substancialmente entre 10 e 50 km/h. De todos os acidentes ocorridos com velocidades de colisão que puderam ser aferidas, as velocidades maiores do que 50 km/h representam menos de 20% do total.

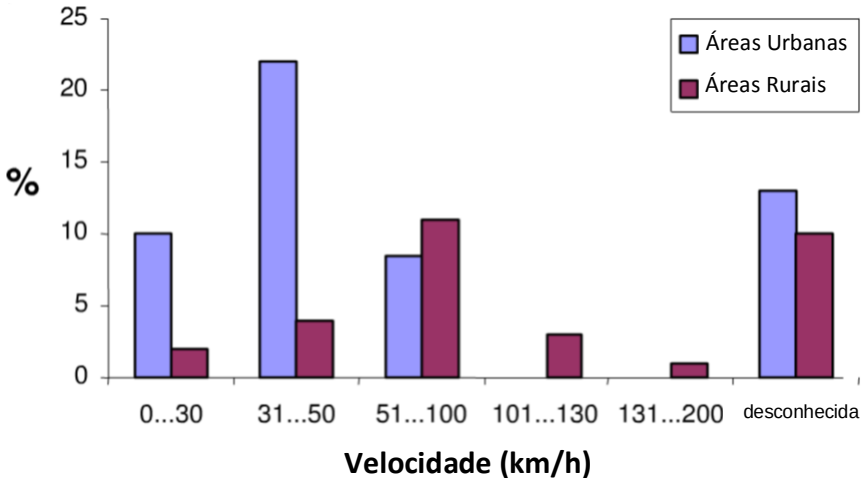
Considerando um veículo em movimento, a distância percorrida antes que o veículo seja parado é dependente da percepção do motorista em executar tal ação e da inércia do veículo. Esses dois fatores são, por sua vez, impactados pela velocidade do veículo. A Figura 17 ilustra a dependência entre a distância percorrida e a velocidade do automóvel.

Figura 15 – Distribuição de acidentes frontais por cenário envolvendo pedestres com casualidades.



Fonte: Adaptada de Lange (2007).

Figura 16 – Distribuição de velocidades de colisão.

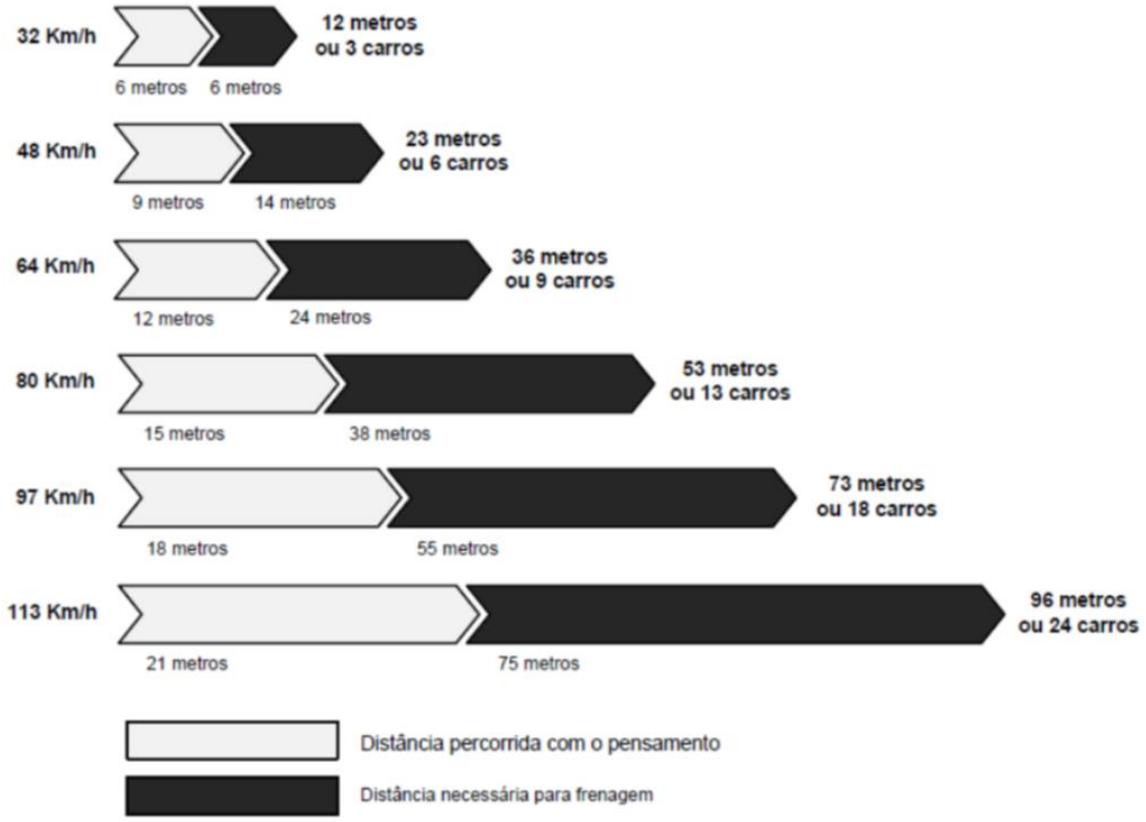


Fonte: Adaptada de Gavrila, Marchal e Meinecke (2003).

2.3.1.2 Desafios na Detecção de Pedestres

Os principais desafios em Sistemas de Proteção ao Pedestre estão relacionados à detecção de pedestres em uma determinada cena. Geronimo *et al.* (2010) apontam alguns pontos que dificultam a tarefa de desenvolver esse tipo de sistemas. O primeiro deles é a dificuldade em determinar a aparência dos pedestres. Pessoas caminham pelas ruas vestindo diferentes roupas, carregando os mais variados objetos e apresentam uma variação considerável de tamanho e de altura. Pedestres podem, ainda, aparecer em diferentes ângulos (e.g., de frente, de costas,

Figura 17 – Distância requerida para frenagem de um veículo.



Fonte: Martinez (2011).

lateralmente).

Outro desafio encontra-se no fato de que pedestres devem ser identificados em cenários urbanos externos, ou seja, devem ser detectados em meio a um cenário carregado de detalhes, sob variadas condições de iluminação e sujeito às condições do tempo. Além disso, pedestres podem estar parcialmente ocluídos por elementos urbanos, como automóveis, placas de publicidade e de trânsito.

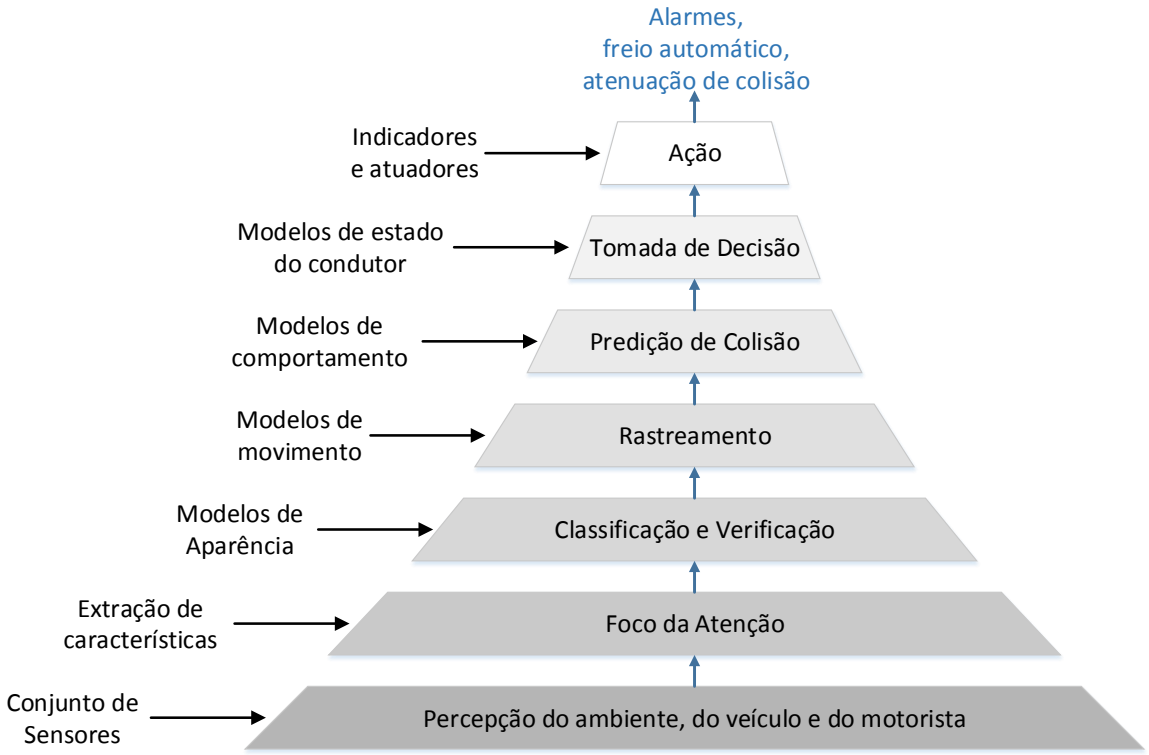
Pedestres devem também ser identificados em cenas altamente dinâmicas, pois tanto o pedestre quanto a câmera estão em movimento. Este fator traz grande complexidade a tarefas como rastreamento e análise de movimento.

Alguns trabalhos presentes na literatura buscam padronizar o perfil de um pedestre, de forma a determinar o padrão a ser reconhecido em uma determinada cena. Um dos exemplos mais recentes desse tipo de esforço está descrito em Dollar *et al.* (2012). Nesse trabalho, os autores criaram o *Caltech Pedestrian Data Set*, uma base de dados para pesquisa de sistemas de detecção de pedestres em ambientes urbanos. A análise desta base de dados permitiu extrair uma série de características sobre a aparência de pedestres, como a proporção entre altura e largura de um pedestre, seu tamanho em pixels na imagem, aspectos de pedestres que sofrem oclusão, etc.

2.3.1.3 Fluxo de Processamento em PPS

Sistemas de proteção ao pedestre em ADAS apresentam o fluxo de informação mostrado na Figura 18. Nela, a cadeia de processamento está organizada como uma pirâmide com a base de tamanho maior, indicando uma grande quantidade de dados de sinais brutos obtidos a partir de sensores. À medida em que se sobe na pirâmide, toda a informação útil é destilada em estágios sucessivos, até que finalmente, uma ação é tomada.

Figura 18 – Fluxo de informação em sistemas de detecção de pedestres.



Fonte: Elaborada pelo autor.

No primeiro estágio da Figura 18 (base da pirâmide), os dados do ambiente ao redor e internos ao veículo são capturados por sensores. Essa percepção do ambiente se caracteriza pela enorme quantidade de dados fornecida pelos sensores presentes no sistema. Os sensores mais frequentemente utilizados em aplicações ADAS foram descritos na Tabela 1. A escolha do sensor irá impactar diretamente na complexidade das soluções das etapas subsequentes. Dentre os sensores que podem ser utilizados, os mais comuns são:

- **Sensores Infravermelhos**: Sensores infravermelhos térmicos captam a radiação emitida pelo corpo humano e, por isso, são bastante efetivos para a detecção de pedestres, principalmente durante a noite. Embora este tipo de sensor tenha um custo relativamente mais alto, ele tem sido bastante utilizado para oferecer a capacidade de visão noturna em veículos modernos. Contudo, esse tipo de imagem é menos efetiva durante o calor

do dia, onde há uma menor diferença de temperatura entre os pedestres e o ambiente. Outro tipo de sensor que pode ser útil no período da noite são os sensores infravermelhos de curto alcance acompanhados por um iluminador. Atualmente, sistemas que utilizam esses sensores são mais baratos do que os que utilizam IR térmicos e têm sido usados em aplicações de vigilância. Sistemas que usam esse tipo de sensor produzem imagens que lembram as imagens produzidas por sensores de luz visível, como câmeras de vídeo. Por esta razão, técnicas de processamento de imagem desenvolvidas para sistemas de luz visível monocromática podem ser modificadas para análises com essas imagens.

- **Radares**: Sensores como radares fornecem uma informação precisa sobre distância, medindo o tempo que leva para que os raios emitidos retornem ao sensor. Radares utilizam energia eletromagnética na região de microondas para medir a distância a objetos. Para localizar objetos na dimensão azimutal, radares com vários feixes são empregados em aplicações de veículos. Sistemas de controle de velocidade de cruzeiro, os quais objetivam manter uma distância para um veículo à frente, já foram introduzidos nos mercados de carro de luxo nos últimos anos.

- **Lasers**: Também fornecem informação precisa sobre a distância. Lasers são constituídos por um transmissor pulsante de laser, um receptor e um espelho que rotaciona em volta de um eixo vertical. O espelho reflete o pulso de laser para direcioná-lo em algum ângulo do azimute no plano horizontal. O sensor então gera um mapa de alcance (distância) de uma seção horizontal da cena chamada de plano de varredura, fornecendo como saída dados de alcance e reflectividade para um número de amostras de ângulos do azimute. Um dos problemas encontrados na utilização desta tecnologia em veículos ocorre devido às características da pista ou rua. Inclinações e curvaturas podem nem sempre manter o plano de varredura paralelo à rua, ocasionando a perda de obstáculos ou detectando a própria rua como obstáculo. Para tentar solucionar este problema, *scanners* de laser multicamadas foram projetados.

- **Câmeras de vídeo**: Sensores de imagem são capazes de capturar uma visão perspectiva de alta resolução, contendo uma grande riqueza de informações. A extração dessas informações envolve uma quantidade substancial de processamento. Contudo essa solução possui um custo relativamente menor quando comparada às soluções com lasers e radares.

Todos os sensores possuem suas vantagens e suas limitações. Para melhor se aproveitar das vantagens e superar as limitações, pode-se usar uma combinação de múltiplos sensores. Desta forma, cada sensor fornece informações complementares às de outro sensor. Por exemplo, câmeras de vídeo podem ser utilizadas juntamente com sensores IR para que um sistema de detecção esteja operante tanto de dia quanto de noite.

Os sensores capturam dados do ambiente e os repassam ao próximo estágio de processamento descrito na Figura 18. O estágio de foco de atenção trabalha com os dados brutos

provenientes da fase de percepção do ambiente, utilizando algoritmos para identificar regiões de interesse (ROI - *Region of interest*). As regiões de interesse são partes da imagem original que possuem uma alta probabilidade de conter pedestres. A determinação das ROIs é, normalmente, baseada em um série de *features* extraídas da cena. Nesta etapa, é importante que haja uma alta taxa de detecção de ROIs, ainda que resultem em falsos alarmes nas fases posteriores.

O conjunto de ROIs é então recebido pelo próximo estágio da pirâmide, chamado de classificação e verificação. Esse estágio aplica algoritmos de decisão sobre cada ROI, a fim de detectar se estas contém ou não pedestres genuínos ou se simplesmente contém falsos alarmes. Os algoritmos de decisão geralmente utilizam modelos de aparência para classificar corretamente os pedestres.

As ROIs contendo informações de pedestres classificados como positivos passam para a etapa de rastreamento. Nessa etapa, uma ROI contendo um pedestre é monitorada durante vários quadros da sequência de imagens. Modelos de comportamento irão auxiliar para que o sistema possa determinar a possível trajetória de um pedestre na cena.

Os dados de trajetória do pedestre são repassados para a etapa de predição de colisão. Essa etapa avalia se a trajetória do veículo poderá coincidir com a trajetória do pedestre, provocando assim uma colisão. Caso essa possibilidade se confirme, um alerta é enviado à próxima etapa da pirâmide, chamada de ação. Nessa etapa, o motorista poderá receber sinais de advertência para que possa tomar uma ação corretiva. Se a colisão for iminente, sistemas automáticos de segurança podem ser acionados para desacelerar o veículo, reduzindo a força do impacto, ou evitando o acidente.

CAPÍTULO

3

ESTADO DA ARTE

Sistemas de Proteção ao Pedestre (PPS) têm se apresentado como uma área de pesquisa bastante ativa. Dentre os diferentes processos relacionados a essa área, a detecção de pedestres apresenta-se como uma tecnologia chave para a melhoria da segurança no trânsito. Neste sentido, muitos trabalhos têm sido desenvolvidos com o objetivo de propor técnicas eficientes para a detecção precisa de pedestres nas mais diversas cenas. Observa-se também que, à medida em que tecnologias de proteção ao pedestre popularizam-se e são empregadas em automóveis modernos, diversas arquiteturas para o processamento de algoritmos de detecção de pedestres têm surgido tanto na academia quanto na indústria, buscando formas de satisfazer os requisitos inerentes aos sistemas embarcados.

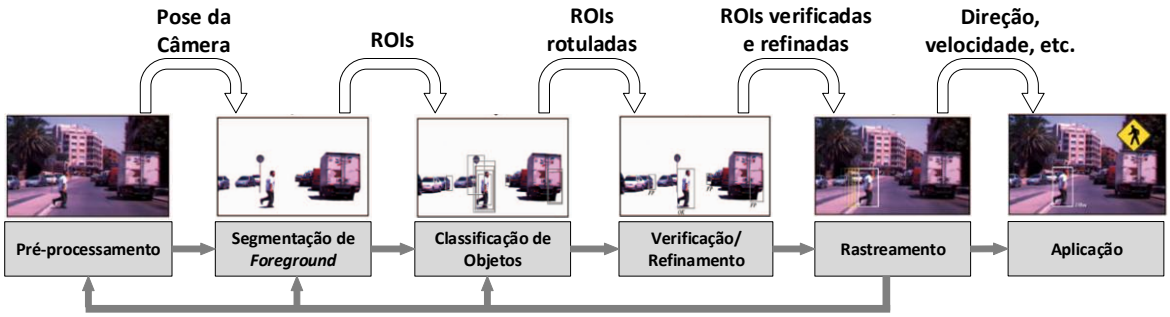
O propósito deste capítulo é mostrar o estado da arte das técnicas para detecção de pedestres e das arquiteturas existentes na literatura que atendam a esse domínio da aplicação. Para isso, inicialmente, é apresenta a estrutura geral de um algoritmo de detecção de pedestre, de forma que se possa compreender os fatores envolvidos no desenvolvimento desses sistemas. Em seguida, são apresentados dois algoritmos que são a base para a maior parte dos detectores de pedestres modernos e que são alvo de estudo e implementação neste trabalho. Por fim, são citados diversos trabalhos que desenvolveram arquiteturas de hardware para a detecção de pedestres e que mostram qual é o estado da arte na área.

3.1 Estrutura Geral dos Algoritmos de Detecção de Pedestres

A estrutura geral de um algoritmo para detecção de pedestres baseada em visão está ilustrada no diagrama da Figura 19 (GERONIMO *et al.*, 2010). O fluxo de processamento mostrado inclui seis módulos organizados em cascata: pré-processamento, segmentação de primeiro plano (ou, no inglês, *foreground*), classificação de objetos, verificação, rastreamento e

aplicação. Embora nem todos os trabalhos presentes na literatura encaixem-se nesses módulos, a maioria dos sistemas pode ser mapeada conceitualmente nesse fluxo, ainda que seja necessário agrupar alguns dos módulos. A divisão em etapas, entretanto, permite uma melhor análise dos diferentes métodos existentes.

Figura 19 – Diagrama da arquitetura de um sistema de detecção de pedestres baseado em visão.



Fonte: Adaptada de Geronimo *et al.* (2010).

As três primeiras etapas do fluxo podem ser consideradas de nível de processamento baixo a médio e favorecem a investigação de implementações eficientes em hardware (KIM; SHIN, 2014). As últimas etapas são mais complexas em lógica de controle e são melhor implementadas em software. Neste trabalho, o foco está nas três primeiras etapas de processamento e no pós-processamento pra tratamento dos resultados de detecção obtidos.

3.1.1 Pré-processamento

O módulo de pré-processamento geralmente tem o objetivo de aplicar melhoramentos nas imagens captadas pelos sensores. Entre os primeiros três estágios, este é o que apresenta funções mais simples, tais como correção da distorção da câmera, retificação e ajustes de ganhos. Outras funções encontradas incluem aquelas para o controle da luminosidade na imagem, como correção gama, ou diminuição de ruídos, como filtros de suavização. Em outras palavras, as funções de pré-processamento visam corrigir problemas que prejudicariam a precisão dos algoritmo dos módulos posteriores no fluxo de detecção, tais como a saturação em imagens devido ao rápido movimento da cena ou ainda à passagem por túneis.

Apesar de não ser abordada neste trabalho, é interessante citar que uma das soluções de pré-processamento que tem ganho destaque é a chamada *High Dynamic Range* (HDR) (MARSI *et al.*, 2007; KNOLL, 2007). Esta técnica está sendo utilizada em Sistemas de Assistência ao Condutor devido ao seu potencial em fornecer um alto contraste mesmo em condições de cenas como as mencionadas anteriormente.

3.1.2 Segmentação de Primeiro Plano

A segmentação do primeiro plano tem o objetivo de extrair *Regions of Interest* (ROI), ou regiões de interesse, de uma imagem, de forma que elas possam servir como entradas para os próximos módulos no fluxo de detecção de pedestres. Embora alguns dos trabalhos presentes na literatura não possuam um módulo específico para segmentação, estas técnicas são importantes para reduzir o número de candidatos. Algumas restrições a respeito do tamanho do pedestre podem ser importantes nesse módulo, tais como relação de aspecto, tamanho e posição, as quais permitirão selecionar ROIs que tenham uma maior probabilidade de conterem pedestres. Alguns trabalhos definem essas restrições, como o de Geronimo *et al.* (2007), que assume pedestres com $1,7m$ de altura e uma proporção de aspecto de $1/2$. O trabalho desenvolvido por Dollar *et al.* (2012) adota uma métrica para o tamanho de pedestres medida em pixels e os dividem em três escalas, de acordo com suas distâncias em relação à câmera. Pedestres próximos possuem acima de 80 pixels de altura, pedestres em uma distância média possuem de 30 a 80 pixels e aqueles distantes, menos de 30 pixels. A proporção do aspecto, neste caso, é de $0,41$.

Uma das primeiras técnicas para a geração de ROIs é a busca exaustiva. Contudo, neste caso, uma grande quantidade de dados será gerada, fazendo com que as etapas posteriores consumam muito tempo de processamento e espaço de armazenamento. A etapa de segmentação deve eliminar o maior número possível de regiões de interesse contendo informação de segundo plano e selecionar eficientemente ROIs que tenham qualquer possibilidade de conter um objeto do primeiro plano.

A técnica mais simples para obter hipóteses inicias de localização dos pedestres é chamada de janela deslizante (DALAL; TRIGGS, 2005; PAPAGEORGIOU; POGGIO, 2000). Essa abordagem é baseada no escaneamento exaustivo da imagem, selecionando os candidatos de acordo com as restrições a respeito do pedestre. Um dos problemas dessa técnica ocorre devido ao enorme número de candidatos, o que torna difícil o alcance de requisitos de tempo real. Alguns trabalhos, como os de Wojek *et al.* (2008) e Zhang, Zelinsky e Samaras (2007), tentam corrigir esse problema. Em Enzweiler e Gavrila (2009), os autores realizam uma combinação de janelas deslizantes com outras técnicas para obter um maior *speed-up*. Outra dificuldade com essa abordagem é que muitas regiões irrelevantes são passadas para os módulos seguintes, o que aumenta o potencial de falsos positivos.

O movimento também é um aspecto importante que tem sido utilizado na detecção de pedestres. No caso de câmeras fixas em uma infra-estrutura, a detecção dos objetos em movimento é feita simplesmente pela subtração do *background* (PICCARDI, 2004). Para câmeras em movimento, há o fator de *ego-motion*, que depende do movimento da câmera e da estrutura da cena. A fim de descobrir quais objetos estão se movendo, é necessária a subtração desse *ego-motion* do fluxo de movimento da cena. Em Zhang *et al.* (2006) a estimativa do ego-motion é feita utilizando-se fluxo óptico esparso baseado em *features* angulares. Em Liu e Fujimura (2004), um procedimento de correspondência estéreo e de detecção de movimento em dois estágios

é desenvolvido para distinguir o movimento de um objeto inconsistente com o *background*. Informações sobre o movimento também podem ser combinadas com informações sobre textura, como em Viola, Jones e Snow (2003). Além da necessidade de se ter movimento na cena para descobrir as regiões de interesse, outros problemas surgem para abordagens baseadas em movimento. Por exemplo, para pedestres movendo-se lateralmente, geralmente é possível separar o *ego-motion* dos movimentos dos pedestres da cena. Contudo, para pedestres movendo-se longitudinalmente, o movimento da imagem é paralelo ao *ego-motion* e portanto difícil de ser separado.

A utilização de visão estéreo também tem sido uma das abordagens utilizadas para a detecção de pedestres em uma cena. Franke e Kutzbach (1996) apresentaram um dos primeiros algoritmos desenvolvidos especificamente para ADAS utilizando visão estéreo, mais tarde estendido em Franke e Joos (2000). Muitos autores (BROGGI *et al.*, 2003; GRUBB *et al.*, 2004) fizeram uso de representações de *v-disparidade* para identificar o chão e objetos verticais. Essas abordagens são baseadas no fato de que um plano no espaço euclideano torna-se uma linha reta no espaço de *v-disparidade*. A análise de mapas de disparidade juntamente a restrições de tamanho de pedestres são utilizadas para extrair candidatos em Zhao e Thorpe (1999), Broggi *et al.* (2000), Soga *et al.* (2005).

Uma outra abordagem que tem sido amplamente utilizada visa detectar *features* na imagem que auxiliem na identificação do pedestre. A intenção das *features* é recuperar regiões com alto conteúdo de informação baseado em descontinuidades locais dos valores de brilho da imagem. Tal característica geralmente ocorre nas fronteiras de um objeto (AGARWAL; AWAN; ROTH, 2004; LEIBE *et al.*, 2007; LOWE, 2004; SEEMANN; FRITZ; SCHIELE, 2007).

A adoção de *features* baseadas em informações do gradiente da imagem proporcionou uma grande contribuição para a precisão dos detectores de pedestres. Inspirados por Lowe (2004) com *Scale Invariant Feature Transform* (SIFT), o trabalho de Dalal e Triggs (2005) popularizou o uso de histogramas de gradientes orientados (HOG), mostrando ganhos significativos sobre outras *features* da época. Desde a sua introdução, muitas variações das *features* HOG foram criadas e praticamente todos os detectores modernos as utilizam de alguma forma, como, por exemplo, os trabalhos de Zhu *et al.* (2006), Wu e Nevatia (2008) e Dollar *et al.* (2009a). *Features* HOG também são utilizadas como base para o modelo baseado em partes, ou, em inglês, *Deformable Part Models* (DPM) desenvolvido por Felzenszwalb *et al.* (2010).

Features baseadas em formato (*shape*) também são utilizadas com frequência. Gavrila e Philomin (1999) e Gavrila (2007) empregaram um hierarquia de *templates* e uma métrica de distância para encontrar a correspondência entre as bordas de imagens e um conjunto de *templates*. Wu e Nevatia (2005) utilizaram *features edgelet* para representar formas. O trabalho desenvolvido por Sabzmeydani e Mori (2007) utilizou descritores de formato aprendidos a partir de gradientes de regiões locais, chamados de *shapelets*. *Features* com granularidades ajustáveis são propostas por Liu *et al.* (2009), utilizando HOG e *edgelets* para calibrar níveis de incerteza.

Isoladamente, *features* HOG proporcionam resultados melhores que as outras. Contudo, observa-se que a adição de novas *features* pode fornecer informações complementares, melhorando a precisão das detecções. O trabalho desenvolvido por Wojek e Schiele (2008) realiza uma combinação de *features* Haar, *shapelets*, contexto de formatos e HOG, a qual provou ser melhor do que uma única *feature* isolada. Outros autores, como Walk *et al.* (2010) adicionam auto-similaridade de cores e *features* de movimento. Wu e Nevatia (2008) combinaram HOG, *edgelet*, e *features* de covariância, enquanto Wang, Han e Yan (2009) combinam HOG e descritores de textura baseados em *Local Binary Patterns* (LBP).

Ainda no contexto de combinação de *features*, Dollar *et al.* (2009a) estenderam o trabalho pioneiro de Viola e Jones (2004) com o cálculo de *features* semelhantes à Haar, obtidas por múltiplos canais de dados de imagem. Os canais utilizados incluem o espaço de cores LUV, escalas de cinza, magnitude de gradiente e histogramas de gradiente. Este trabalho, conhecido como *Integral Channel Features* (ICF) introduziu um novo *framework* para a integração de múltiplas *features* e serviu como base para o desenvolvimento de novos detectores. Seguindo o sucesso desse detector, muitas variações foram propostas e mostraram melhorias significativas na detecção (ZHANG; BAUCKHAGE; CREMERS, 2014; ZHANG; BENENSON; SCHIELE, 2015; NAM; DOLLAR; HAN, 2014; PAISITKRIANGKRAI; SHEN; van den Hengel, 2014; ZHANG *et al.*, 2015; BENENSON *et al.*, 2013). Nesse sentido, pode-se citar trabalho intitulado *Fastest Pedestrian Detector in the West* (FPDW) (DOLLAR; BELONGIE; PERONA, 2010), que estendeu o *framework* do ICF para a detecção mais eficiente em várias escalas da imagem de entrada, demonstrando como *features* calculadas em uma única escala podem ser aproximadas para escalas próximas. Ainda, (ZHANG; BAUCKHAGE; CREMERS, 2014) utiliza um conjunto de *templates* baseados no modelo de um pedestre dividido em três componentes. As *features* são calculadas a partir de vários canais e representam diferentes características entre as partes do corpo humano.

Alguns trabalhos consideram o uso de mais de uma ordem de magnitude de canais (LIM; ZITNICK; DOLLAR, 2013; PAISITKRIANGKRAI; SHEN; van den Hengel, 2013). Segundo (BENENSON *et al.*, 2014), apesar de todas as melhorias introduzidas pelo uso de muitos canais, o melhor desempenho é alcançado com apenas os dez canais introduzidos pelo algoritmo ICF, sendo eles: seis de orientações de gradiente, um de magnitude de gradiente e três de cor LUV.

3.1.3 Classificação de Objetos

O estágio de classificação utiliza técnicas de aprendizado de máquina e reconhecimento de padrões para classificar as regiões de interesse como pedestres ou não pedestres. Geralmente, um classificador funciona em duas fases, sendo a primeira a fase de treino, com o objetivo de construir o classificador, e a segunda a fase de reconhecimento (KIM; SHIN, 2014). Na primeira fase, o classificador é treinado com um conjunto de dados que promovem um ajuste dos seus parâmetros internos para um classificação mais precisa. Na segunda fase, o classificador

é utilizado para tomar uma decisão sobre se um objeto pertence ou não a uma categoria. Os dados de entrada de um classificador para detecção de pedestres geralmente são constituídos por valores brutos de pixels ou *features* extraídas durante a fase de segmentação. Os dados de saída do classificador consistem na classe do objeto avaliado ou em um valor de confiança a respeito da decisão, o qual será chamado de pontuação.

Embora cada tipo de classificador tenha suas especificidades, estes são treinados utilizando-se um número significativo de casos de treino positivos e negativos, de forma a determinar os limites das funções de decisão. Para a detecção de pedestres, os classificadores mais utilizados incluem *support vector machines*, *boosting* adaptativo (AdaBoost) e redes neurais.

Support Vector Machines (SVM), criadas por Vapnik (1979) é um dos métodos mais populares de aprendizado supervisionado utilizado em detecção de pedestres, muito por seu desempenho. A ideia desta técnica é maximizar a margem entre as categorias de objetos a serem classificados de forma a aumentar a capacidade de separação. A maximização da distância das margens contribui para uma maior precisão do classificador. SVMs são amplamente utilizadas em importantes técnicas de detecção de pedestres presentes na literatura, tais como o HOG (DALAL; TRIGGS, 2005), utilizando um núcleo linear, e o Modelo de Partes Deformáveis (FELZENSZWALB *et al.*, 2010), utilizando SVM latente.

O método de classificação AdaBoost, criado por Freund e Schapire (1997), tem como princípio a combinação de um conjunto de classificadores fracos para obtenção de um classificador forte. A ideia é que a combinação de classificadores fracos seja mais eficiente em termos de esforço de treinamento e melhoria da precisão. O foco do AdaBoost está no conjunto de entradas no qual o classificador realizou um decisão errônea, dando um peso mais alto a essas entradas quando um novo classificador é gerado. Com isso, o método busca se recuperar da falha do classificador anterior. Muitos dos detectores de pedestres modernos utilizam o Adaboost como classificador, como o ICF (DOLLAR *et al.*, 2009a) e os detectores desenvolvidos por Zhang, Benenson e Schiele (2015).

Redes Neurais é um outro método de classificação utilizado em detecção de pedestres e que tem a intensão de imitar o mecanismo de aprendizado e computação do cérebro humano. Uma das propriedades desse método é a sua capacidade de separação não linear, assim como sua estrutura altamente paralelizável. Muitas variações de redes neurais já foram inventadas, tal como a *Multi-layer Perceptron* e as *Convolutional Neural Networks* (CNN). As características de uma rede neural são determinadas pela forma como são combinados os seus neurônios, pela seleção das funções de transferência e pelo ajuste dos valores dos pesos. De forma similar ao SVM e ao AdaBoost, os pesos dos neurônios são ajustados em resposta aos dados fornecidos durante o treinamento, de forma que uma rede neural posso reconhecer objetos apropriadamente. O uso de CNNs para detecção de pedestres têm sido o foco de muitos trabalhos que visam construir arquiteturas profundas, como o desenvolvido por Sermanet *et al.* (2013), que extrai *features* diretamente dos valores de pixel de uma imagem e usa uma CNN para classificação.

Na avaliação de várias técnicas de detecção de pedestres realizada por Benenson *et al.* (2014), conclui-se que não há uma técnica específica de classificação que se destaque em relação a outras técnicas, assim como não há evidências empíricas indicando que *kernels* de classificação não lineares obtenham ganhos significativos sobre *kernels* lineares.

3.1.4 Pós-processamento

O pós-processamento é composto pelas etapas de verificação/refinamento e rastreamento, as quais são tipicamente baseadas em algoritmos ricos em estruturas de controle. Na etapa de verificação/refinamento são analisadas as detecções resultantes da classificação dos objetos, os quais são inspecionados para prevenir falsos positivos e até mesmo verdadeiros negativos utilizando um determinado critério. Dentre esses critérios, incluem-se a proporção de largura e altura de um pedestre, bem como a sua altura relativa à câmera.

Geralmente, a classificação em detectores baseados em janelas deslizantes,produz uma grande quantidade de *bounding boxes* ao redor de objetos classificados como pedestres. Isto significa que um mesmo objeto pode ter sido detectado mais de uma vez, por exemplo, ao se processar diferentes escalas da imagem. Trabalhos como o desenvolvido por Dalal e Triggs (2005) utilizam uma função de supressão não máxima para eliminar ou reunir detecções cujas áreas de seus *bounding boxes* se sobreponham mais do que um determinado fator (e.g. 50%).

No caso de rastreamento, o objeto é monitorado durante o tempo para que se possa predizer as suas posições futuras e extrair as regiões de interesse mais facilmente na fase de segmentação. Este tópico, contudo, não é contemplado neste projeto.

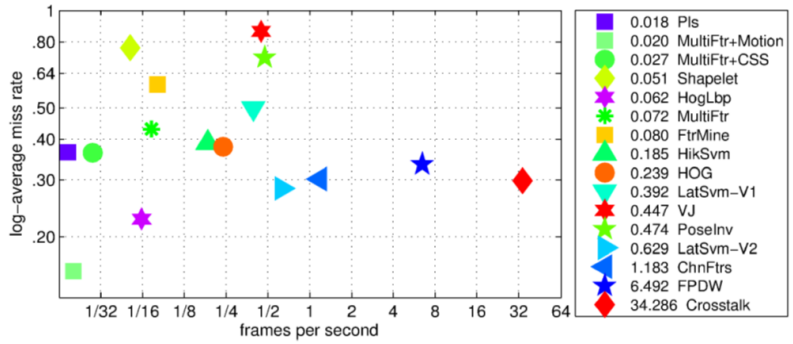
3.1.5 Considerações sobre os Algoritmos para Detecção de Pedestres

A área de detecção de pedestres presenciou, nos últimos quinze anos, a criação de diversos algoritmos. Benenson *et al.* (2014) e Dollar *et al.* (2012) fazem uma extensa revisão dos algoritmos existentes na literatura. Ambos os trabalhos disponibilizam também uma comparação das técnicas já criadas, permitindo assim, verificar o desempenho e a precisão ao processarem imagens de diferentes bases de dados de pedestres.

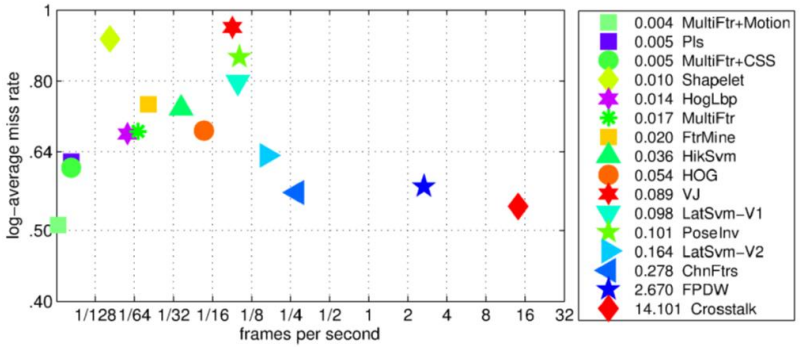
Os gráficos da Figura 20 mostram o estado da arte no desempenho e precisão de diversos algoritmos de detecção de pedestres Dollar *et al.* (2012). Os dados referem-se ao processamento de imagens com resolução 640×480 pixels da base de dados de pedestres da Caltech. Os tempos reportados dos detectores são relativos à execução em um computador moderno, de forma que todos os resultados são comparáveis. As legendas estão ordenadas por velocidade, a qual é reportada em *frames* por segundo (fps). Observa-se que, tanto para pedestres próximos (Figura 20a) quanto mais distantes (Figura 20b), algoritmos baseados em *features* de canais integrais, como o *ChnFtrs* (que é o próprio ICF) e o FPDW estão entre os mais velozes e com maior precisão. O algoritmo HOG, embora tenha sido um dos pioneiros, mantém um equilíbrio

entre velocidade e precisão quando comparado às outras técnicas.

Figura 20 – Gráfico de taxa de erros *versus* tempo de execução para diversos algoritmos no estado da arte para a detecção de pedestres.



(a) Pedestres $\geq$ 100 pixels.



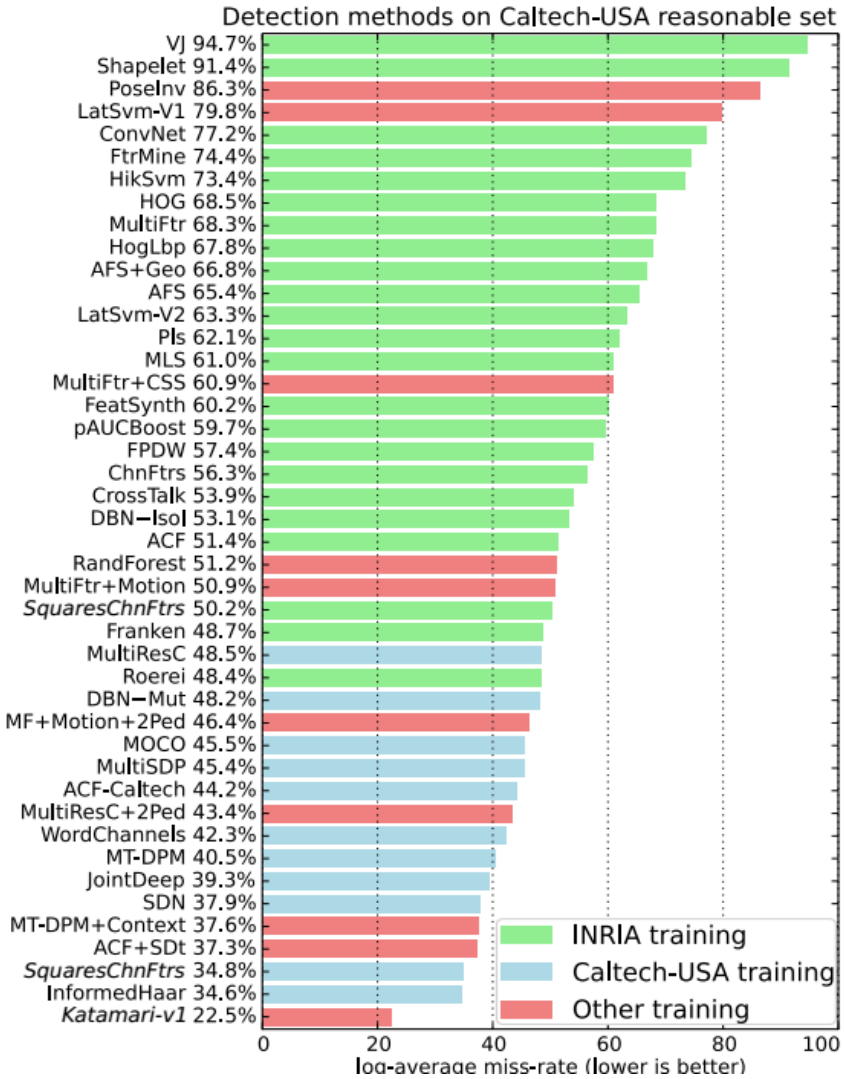
(b) Pedestres $\geq$ 50 pixels.

Fonte: Dollar *et al.* (2012).

Em uma revisão da área mais atual, Benenson *et al.* (2014) realiza a comparação das principais técnicas de detecção de pedestres, utilizando como métrica a taxa de erros ao classificar pedestres nas imagens da base de dados da Caltech. As técnicas estão listadas na Figura 21, sendo que as menos precisas encontram-se no topo da lista. Neste experimento, embora os resultados sejam comparados pela detecção na mesma base de dados, o classificador de cada técnica foi treinado em uma base de dados de pedestres distinta. Nota-se neste gráfico que os algoritmos derivados do trabalho de Dollar *et al.* (2009a) com canais integrais (e.g. *InformedHaar*, *SquaresChnFtrs*, *ACF*, *Roerei*) estão entre aqueles que apresentam taxas de erro mais baixas, o que revela a eficácia desse tipo de solução. É importante também observar que de todas as técnicas listadas, pelo menos 75% delas utilizam, de alguma forma, *features* HOG para detectar pedestres em imagens. Além disso, dentre as dez técnicas com menor taxa de erros, seis utilizam *features* HOG. Estas estatísticas mostram a popularidade e efetividade deste algoritmo.

Uma tendência cada vez maior observada nas técnicas de detecção de pedestres presentes na literatura, é a adoção de múltiplos tipos de *features*. A razão para essa diversificação

Figura 21 – Resultados de detecção sobre a base de dados da Caltech por diversos algoritmos de detecção de pedestres.

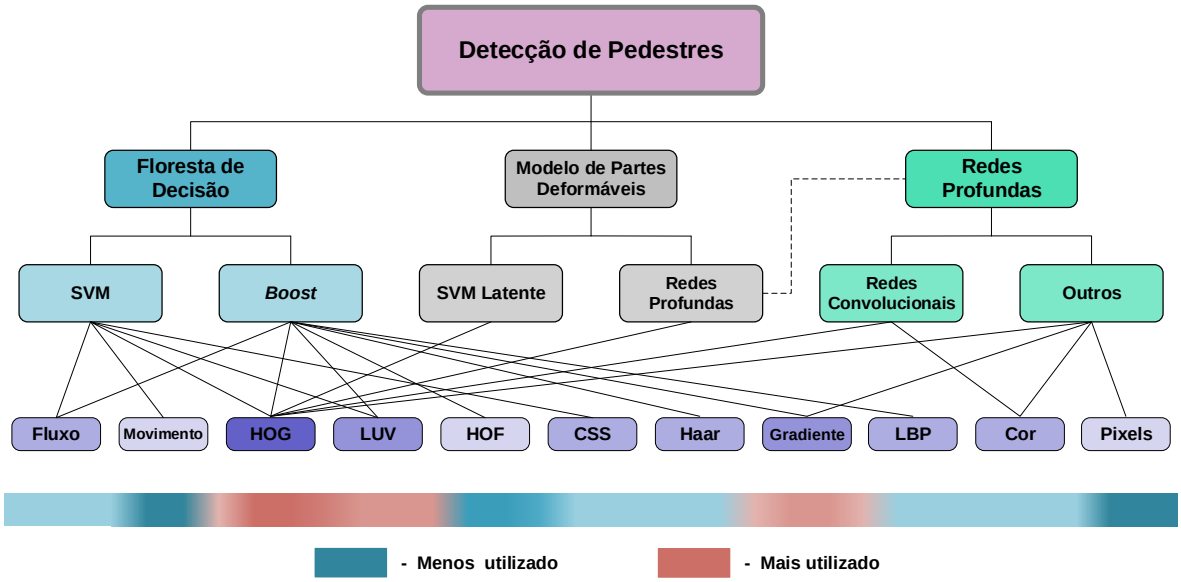


Fonte: Benenson *et al.* (2014).

encontra-se no fato de que, muitas vezes, os resultados proporcionados por diferentes *features* são complementares e contribuem para uma detecção mais precisa. Apesar das técnicas de detecção existentes implementarem as etapas do fluxo de processamento apresentado na Figura 19 de formas diferentes (e.g. classificação por meio de SVM, Adaboost ou redes neurais), é possível discernir um conjunto de *fetures* mais frequentemente empregadas. Para realizar essa identificação, foi feita uma classificação das técnicas de detecção de pedestres baseada nas "famílias de soluções" mencionadas por Benenson *et al.* (2014). O agrupamento das técnicas permite visualizar de forma mais objetiva a interação com os diferente tipos de *features*, conforme pode ser visto na Figura 22.

Os grupos criados por Benenson *et al.* (2014) estão divididos em florestas de decisão, modelos de partes deformáveis e redes profundas. Florestas de decisão são compostas por técnicas que classificam as *features* calculadas utilizando uma ou mais árvores de decisão, que

Figura 22 – Classificação dos algoritmos de detecção de pedestres.



Fonte: Elaborada pelo autor.

podem ser entendidas como um conjunto de classificadores fracos. Incluem-se nesse grupo os algoritmos que utilizam técnicas similares ao AdaBoost e implementações de SVM. O grupo de Modelos de Partes Deformáveis é composto por técnicas que sigam o trabalho de Felzenszwalb *et al.* (2010), definindo regiões de interesse interconectadas na imagem que modelam diferentes partes do corpo humano. Nas regiões de interesse, são extraídas *features* que são submetidas a um classificador SVM Latente ou a uma rede profunda, a qual é geralmente implementada como uma rede convolucional (CNN). O terceiro grupo é formado pelas técnicas que submetem diretamente *features* ou, até mesmo, os pixels da imagem a uma rede profunda, implementada como uma CNN ou uma rede *Multi-layer Perceptron*, por exemplo.

A informação sobre a utilização dos diferentes tipos de *features* por parte dos grupos baseou-se nas revisões de técnicas de detecção de pedestres já citadas. A *feature* mais utilizada por todos os grupos é a HOG e, portanto, pode ser compreendida como sendo a mais descritiva para segmentação de pedestres. Em segundo lugar estão as *features* de gradiente e de cores, incluindo a representação LUV.

O estudo realizado com as técnicas de detecção de pedestres existentes na literatura permite concluir que, neste cenário, duas técnicas se destacam pela popularidade e pela eficácia que proporcionam aos detectores. A primeira delas é o HOG, que tem sido amplamente explorado em diversos trabalhos em diferentes variações. A segunda é o ICF, que estabeleceu um *framework* de detecção baseado em canais e que tem permitido a exploração de algoritmos cada vez mais eficientes e precisos. Na seção seguinte, são descritos os algoritmos dessas duas técnicas de detecção, a fim de que se possa compreender melhor as demandas computacionais envolvidas na implementação e o fluxo de processamento de cada um deles.

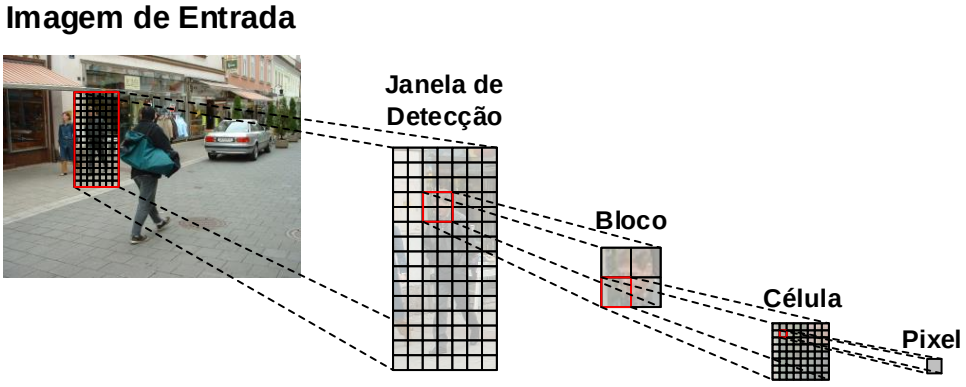
3.2 Descrição dos Algoritmos Utilizados

Os algoritmos HOG e ICF, descritos nesta seção, são a base para diversas das técnicas de detecção de pedestres. A compreensão do funcionamento desses algoritmos, bem como das estruturas de dados utilizadas é essencial para o desenvolvimento do co-projeto de hardware e software descrito no Capítulo 4. Para isso, primeiramente é apresentada uma descrição do processo e da matemática do algoritmo HOG e em seguida, é feito o mesmo para o algoritmo ICF.

3.2.1 Histograma de Gradientes Orientados (HOG)

O algoritmo HOG, originalmente proposto por (DALAL; TRIGGS, 2005), utiliza a abordagem de janelas deslizantes para detecção de objetos baseada em imagens. Para cada janela de detecção, *features* baseadas na distribuição de gradientes de instensidades locais são extraídas e concatenadas para formar descritores HOG, os quais servem como entrada para um classificador SVM. A Figura 23 mostra as estruturas básicas utilizadas pelo algoritmo para o cálculo dos descritores.

Figura 23 – Relação entre as estruturas de janela, bloco, célula para o algoritmo HOG.

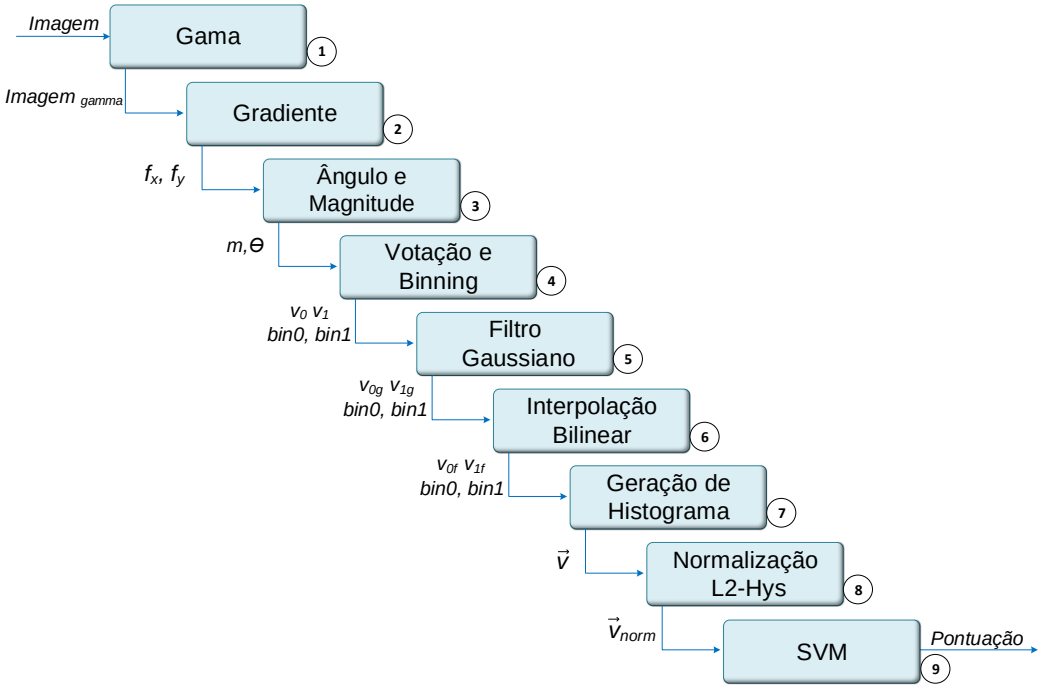


Fonte: Elaborada pelo autor.

As janelas de detecção são divididas em regiões uniformemente espaçadas, chamadas de células, as quais também são agrupadas em blocos sobrepostos. Esta organização permite a aplicação de normalizações e interpolações em regiões de diferentes tamanhos. Blocos reúnem todos os histogramas unidimensionais de direções de gradientes acumulados em suas células, formando assim, histogramas de blocos. Todos os histogramas de blocos dentro de uma janela de detecção são concatenados para formar um descritor HOG, o qual é submetido a um classificador SVM. A sequência de passos na cadeia de detecção está ilustrada na Figura 24.

Na medida em que os pixels da imagem entram na cadeia de detecção baseada em HOG, correção gamma é aplicada a fim de reduzir a influência de variações na intensidade da luz. Para

Figura 24 – Cadeia de extração de *features* e detecção de objetos do algoritmo HOG.



Fonte: Elaborada pelo autor.

calcular o gradiente, a máscara 1-D $[-1\,0\,1]$ é aplicada em amabas as direções (x and y) para cada pixel. A partir dos valores dos gradientes, pode-se encontrar a magnitude m utilizando a norma euclideana e o ângulo θ por meio da função *arcotangente*.

O próximo passo na cadeia de detecção consiste em acumular os valores de orientção nos histogramas de células, que são compostos por 9 *bins* para orientações variando entre 0^o e 360^o. Para reduzir o efeito de *aliasing* das orientações dentro de uma célula, m é dividido em dois votos entre *bins* vizinhos. A votação utiliza o peso α, que é calculado com base na distância de θ à borda do *bin* correspondente e dados por

$$\alpha = \frac{9 \times \theta}{\pi} - 0.5 \tag{3.1}$$

Os dois votos, v_0 e v_1, são definidos pela Equação 3.2. Um filtro Gaussiano é aplicado a todos os votos dentro de um bloco para reduzir a contribuição dos pixels próximos às bordas de um bloco. Os votos v_0 e v_1 são, ainda, interpolados bilinearmente com os votos das células vizinhas dentro do mesmo bloco. Os votos são então acumulados nos histogramas de células correspondentes e concatenados em histogramas de blocos.

$$\begin{cases} v_0 = (1-\alpha) \times m(x,y) \\ v_1 = \alpha \times m(x,y) \end{cases} \tag{3.2}$$

Cada histograma de blocos é normalizado para melhor invariância a mudanças de contraste e iluminação utilizando a norma *L2-hys*

$$\vec{v} = \frac{\vec{v}}{\sqrt{\|\vec{v}\|^2 + \varepsilon^2}} \tag{3.3}$$

a qual é aplicada duas vezes para cada vetor de histograma de bloco $\vec{v}$. Todos os histogramas de blocos dentro de uma janela de detecção são concatenados para forma o vetor descritor. Este vetor é fornecido como entrada à etapa de classificação SVM, a qual realiza um produto escalar com um vetor de classificação já treinado, mais um termo de *bias* ρ. O classificador retorna um valor de confiança que é comparado a um limiar. Se o valor for maior que o limiar, a janela de detecção classificada contém uma instância do objeto e sua informação é armazenada. É possível ainda realizar a detecção em múltiplas escalas ao aplicar a cadeia de detecção várias vezes em diferentes escalas da imagem. Por fim, ao utilizar um algoritmo de supressão não-máxima, é possível eliminar candidatos redundantes encontrados pelo processo de classificação.

3.2.2 Integral Channel Features

O algoritmo *Integral Channel Features*, proposto por Dollar *et al.* (2009a), também baseia-se na abordagem de janelas deslizantes e realiza a extração de *features* por meio de canais e imagens integrais. Um canal consiste em uma função que é aplicada à porções de pixels da imagem de entrada, gerando uma saída que preserva o *layout* da imagem. Assim, dada uma imagem I, um canal C pode ser descrito como

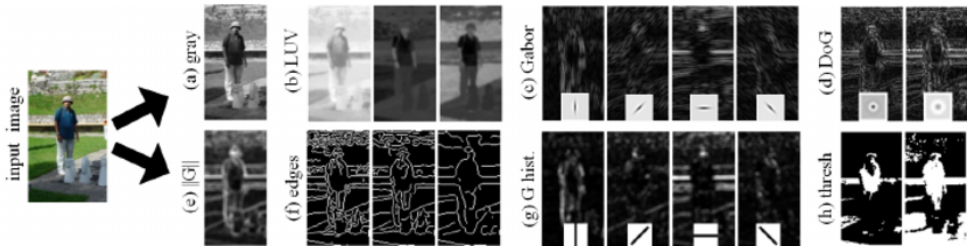
$$C = \Omega(I) \tag{3.4}$$

onde Ω é a função geradora do canal C. A função Ω pode tão simples quanto uma transformação dos pixels para escala de cinza ou mais complexa, como uma transformação não linear. Uma restrição para a escolha de um canal é que este seja invariante à translação, o que permite que a função geradora seja aplicada uma única vez sobre imagem, ao invés de uma vez para cada janela de detecção.

De acordo com Dollar *et al.* (2009a), canais podem gerar *features* de primeira ou segunda ordem. Uma *feature* de primeira ordem consiste na soma dos pixels em uma região retangular em um determinado canal. Já uma *feature* de segunda ordem é definida como qualquer *feature* que possa ser calculada utilizando múltiplas *features* de primeira ordem. A Figura 25 ilustra a aplicação de diferentes tipos de funções geradoras de canais utilizando *features* de primeira e segunda ordens.

A soma de pixels em regiões retangulares pode ser eficientemente realizada por meio do algoritmo de imagens integrais. Esse algoritmo foi introduzido por Crow (1984) e tornado popular

Figura 25 – Canais calculados utilizando diferentes transformações da imagem de entrada. Em seguida, *features* tais como somas locais, histogramas e wavelets Haar são extraídas.



Fonte: Dollar *et al.* (2009a).

pelo trabalho sobre detecção de faces feito por Viola e Jones (2004) como uma representação intermediária da imagem para extração de *features* Haar. Matematicamente, a imagem integral na posição (x,y) contém a soma dos pixels acima e à direita de x e y, conforme a equação:

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x',y') \tag{3.5}$$

onde $ii(x,y)$ e $i(x',y')$ representam os valores da imagem integral e da imagem original, respectivamente. Isto significa que, para uma imagem de entrada com resolução 320×240 pixels, 1.474.560.000 de operações de adição são necessárias. De forma mais geral, devem-se realizar $\frac{1}{4}M^2N^2$ adições para uma imagem de resolução $M \times N$. Contudo, para reduzir a quantidade de operações, utiliza-se o seguinte par de recorrências:

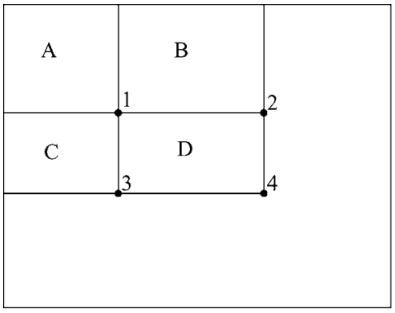
$$s(x,y) = s(x,y-1) + i(x,y) \tag{3.6a}$$
$$ii(x,y) = ii(x-1,y) + s(x,y) \tag{3.6b}$$

onde $s(x,y)$ é a soma cumulativa da linha em x e y. Cabe ressaltar que $s(x,-1) = 0$, $ii(-1,y) = 0$ e que, com essas equações, a imagem integral pode ser calculada percorrendo-se uma só vez a imagem original. O par de recorrências apresentados reduz o número de operações realizadas a $2MN$. Ao utilizar imagens integrais, qualquer soma retangular pode ser calculada por meio de quatro referências à estrutura de dados da imagem. Por exemplo, na Figura 26, a soma dos pixels contidos na região D ($i(D)$) é obtida por meio dos valores da imagem integral nos pontos 1, 2, 3 e 4, conforme a Equação 3.7.

$$\sum i(D) = ii(4) + ii(1) - (ii(2) + ii(3)) \tag{3.7}$$

Vários tipos de canais podem ser utilizados para capturar diferentes aspectos da imagem e, ao mesmo tempo, manter a invariabilidade à translação. Incluem-se nessa lista sistemas de cor (RGB, LUV, HSV, escalas de cinza), filtros lineares, transformações não lineares, transformações

Figura 26 – A soma dos pixels contidos no retângulo D pode ser obtida por meio dos valores da imagem integral em 1, 2, 3 e 4.



Fonte: Viola e Jones (2004).

ponto-a-ponto, histogramas integrais e histogramas de gradientes. Destes, o algoritmo ICF utiliza três tipos de canais: sistema de cor LUV, magnitude de gradiente e histogramas de gradientes.

O sistema de cores LUV, proposto pela *Commission Internationale de l'Éclairage* (CIE), divide a cor em componentes de crominância (UV) e luminância(L). Tal sistema caracteriza-se por sua uniformidade perceptual, o que significa que a distância euclideana entre duas cores modela a percepção humana da diferença entre essas cores (MRAK; GRGIC; KUNT, 2010). Para calcular os valores de luminância e crominância LUV a partir de uma imagem RGB é necessário, primeiramente, realizar a conversão para o espaço de cores XYZ. Isto é feito aplicando-se uma matriz de transformação sobre os pixels da imagem da seguinte forma:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} R \\ G \\ B \end{bmatrix} \begin{bmatrix} 0.430574 & 0.341550 & 0.178325 \\ 0.222015 & 0.706655 & 0.071330 \\ 0.020183 & 0.129553 & 0.939180 \end{bmatrix} \tag{3.8}$$

Em seguida, realiza-se a conversão da imagem em XYZ para Luv, considerando $Y_n = 1,0$ e utilizando as equações

$$L = \begin{cases} 116(\frac{Y}{Y_n})^{\frac{1}{3}} - 16, & \text{se } \frac{Y}{Y_n} > 0,01 \\ 903,3(\frac{Y}{Y_n})^{\frac{1}{3}}, & \text{caso contrário} \end{cases} \tag{3.9}$$

$$U = 13L(U' - U_n'), \text{ onde } U' = \frac{4X}{X + 15Y + 3Z} \text{ e } U_n' = 0,197833 \tag{3.10}$$

$$V = 13L(V' - V_n'), \text{ onde } V' = \frac{9Y}{X + 15Y + 3Z} \text{ e } V_n' = 0,468331 \tag{3.11}$$

Ao final dos cálculos, cada componente de cor L, U e V irá gerar um canal do algoritmo ICF. Outro canal utilizado pelo algoritmo está baseado no valor da magnitude dos gradientes calculados a partir dos pixels da imagem de entrada. O cálculo dos gradientes é feito pela Equação 3.12. Baseado nos valores dos gradientes, calcula-se a magnitude com a Equação 3.13.

$$f_x(x,y) = f(x+1,y) - f(x-1,y) \tag{3.12a}$$
$$f_y(x,y) = f(x,y+1) - f(x,y-1) \tag{3.12b}$$

$$m(x,y) = \sqrt{f_x(x,y)^2 - f_y(x,y)^2} \tag{3.13}$$

Os demais canais gerados pelo algoritmo consistem em histogramas de gradientes, obtidos de forma semelhante ao processo do algoritmo HOG visto na subseção 3.2.1. Contudo, algumas das etapas cumpridas no algoritmo original não são realizadas na geração dos histogramas para o ICF. A aplicação da correção gamma e do filtro gaussiano não são realizadas e a interpolação é feita calculando-se a média dos pixels em uma vizinhança 3×3. Além disso, o número de *bins* é reduzido de nove para seis. Cada posição do histograma relativa a um *bin* gera um canal para o algoritmo ICF.

Os valores resultantes de cada um dos dez canais são transformados em dez imagens integrais, sobre as quais desliza a janela de detecção. O processamento realizado dentro da janela segue as características das *features* de primeira ordem geradas randomicamente. Dollar *et al.* (2009a) avaliam o uso de *features* de segunda ordem também randômicas, extraídas a partir dos dados de múltiplos canais integrais. Estas, contudo, não se mostraram efetivas no processo de detecção de pedestres, sendo preteridas por *features* de primeira ordem mais eficientes. Tais *features* são calculadas por meio da Equação 3.7 em regiões retangulares das janelas de detecção. Por fim, o valor de cada uma das cinco mil *features* são submetidas a um classificador Adaboost, que estabelecerá uma pontuação para a janela de detecção avaliada.

3.3 Arquiteturas para Detecção de Pedestres

Diversos trabalhos têm sido propostos com arquiteturas de hardware para a detecção de pedestres. Como já mencionado anteriormente, o foco de implementação deste trabalho está nos algoritmos HOG e ICF, embora a arquitetura desenvolvida intente permitir adaptações e evoluções destes algoritmos provendo flexibilidade, tanto por meio da programabilidade do software quanto pela configurabilidade do hardware. Assim como ocorre com as diferentes técnicas de detecção, o algoritmo HOG está presente na grande maioria das implementações encontradas na literatura. Faz-se, primeiramente, uma revisão do estado da arte de tais arquiteturas. No melhor do conhecimento do autor deste trabalho, para o algoritmo ICF, não foram encontradas arquiteturas em hardware e, portanto, não são discutidas implementações desse algoritmo. Em seguida são consideradas as arquiteturas para detecção de pedestres que são direcionadas para outros algoritmos ou classes de algoritmos. Nessa revisão, além de arquiteturas desenvolvidas por meio de trabalhos acadêmicos, são apresentadas também algumas soluções desenvolvidas em ambientes industriais para o atendimento de aplicações de proteção ao pedestre e, mais

genericamente, sistemas ADAS. Contudo, para efeito de comparação, são consideradas somente as arquiteturas implementadas na mesma plataforma utilizada neste trabalho, no caso, em FPGA.

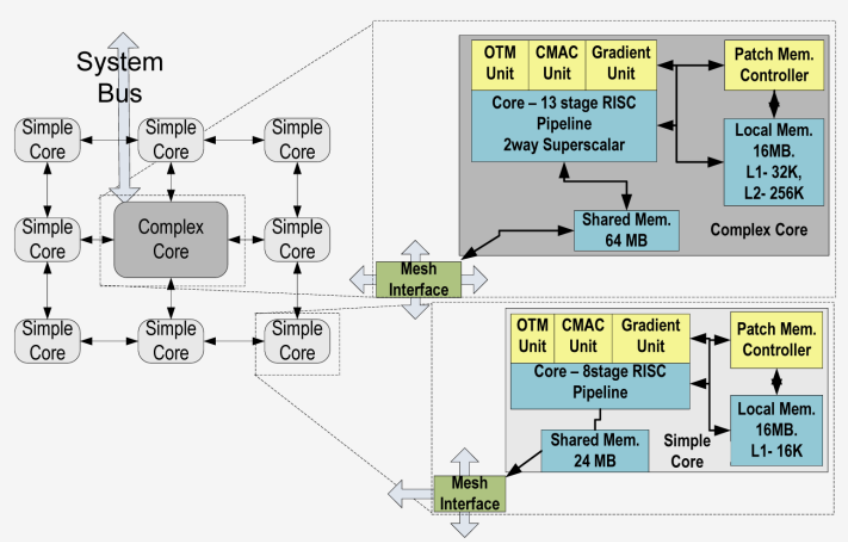
A popularização do uso do algoritmo HOG para detecção de pedestres proporcionou a sua exploração de muitas formas diferentes, tanto em hardware quanto em software. O processamento desse algoritmo é uma tarefa computacionalmente intensiva e requer um grande poder de processamento para alcançar um desempenho em tempo real. Como consequência, muitas arquiteturas de hardware foram propostas para acelerar o tempo de execução utilizando *Graphics Processing Units* (GPU) ou até mesmo abordagens híbridas contendo processadores *desktop*, GPUs e FPGAs. Exemplos de arquitetura híbrida são os trabalhos desenvolvidos por Blair (2014) e Bauer *et al.* (2010). Contudo, a utilização de GPUs e CPUs *desktop* não satisfazem as demandas de sistemas embarcados móveis que requerem baixa potência e consumo de energia reduzido.

Neste trabalho, o foco está em estudos que objetivam a implementação do algoritmo HOG utilizando sistemas embarcados, especialmente FPGAs. A grande maioria das arquiteturas encontradas na literatura utilizam implementações em hardware fixo. Tais arquiteturas geralmente promovem várias simplificações ou otimizações algorítmicas para obter uma taxa maior de *frames* processados por segundo, como o trabalho pioneiro desenvolvido por Kadota *et al.* (2009). Algumas dessas simplificações também foram adotadas e aprimoradas em trabalhos subsequentes. Outros autores buscam acelerar a detecção de pedestres focando em diferentes aspectos algorítmicos. Komorkiewicz, Kluczewski e Gorgon (2012) estenderam a detecção em hardware a outras classes de objetos utilizando blocos de classificação dedicados para cada objeto. Ma, Najjar e Roy-Chowdhury (2015) realizaram uma implementação em ponto fixo promovendo uma análise da quantidade de bits necessária para manter a precisão do algoritmo. Hahnle *et al.* (2013) propõem uma multiplexação do processamento de escalas das imagens de entrada no tempo, possibilitando uma diminuição nos recursos utilizados. Muitos outros autores também realizaram implementações em hardware fixo, tais como Mizuno *et al.* (2012), Lee *et al.* (2013), Hemmati *et al.* (2014) e Advani *et al.* (2015).

Alguns trabalhos introduzem variações do descritor HOG em suas implementações em hardware. Hiromoto e Miyamoto (2009) apresentam a implementação de uma variação chamada de *co-occurrence* HOG (Co-HOG). Este descritor utiliza pares de orientações de gradientes, dos quais deriva uma matriz de co-ocorrência que irá gerar os vetores para a classificação. Martelli *et al.* (2011) adotam uma abordagem para extração de features pelo uso de janela deslizantes, blocos e células, extraindo e combinando features de co-variância. Utilizando a representação integral, são calculados tensores para cada quatro células adjacentes, compondo assim, o tensor dos blocos. Uma matriz de covariância é então calculada para cada bloco e a concatenação das projeções de cada matriz forma o descritor que será submetido a um classificador SVM linear. Tasson *et al.* (2015) implementam o algoritmo de partes deformáveis utilizando *features* HOG. Negi *et al.* (2011) utilizam *features* HOG com padrão binário em conjunto com um classificador AdaBoost.

Apesar da maioria das implementações mencionadas atingirem desempenho de tempo real e oferecerem níveis de consumo de energia adequados para sistemas embarcados, o uso de hardware fixo torna difícil a adaptação do algoritmo a diferentes situações, aplicações ou até mesmo a combinação com outras técnicas, assim como a exploração do espaço de projeto. Uma solução que forneça uma maior flexibilidade e mantenha desempenho e o consumo de energia em níveis apropriados é, desta maneira, desejável.

Figura 27 – Diagrama da arquitetura do EFFEX.

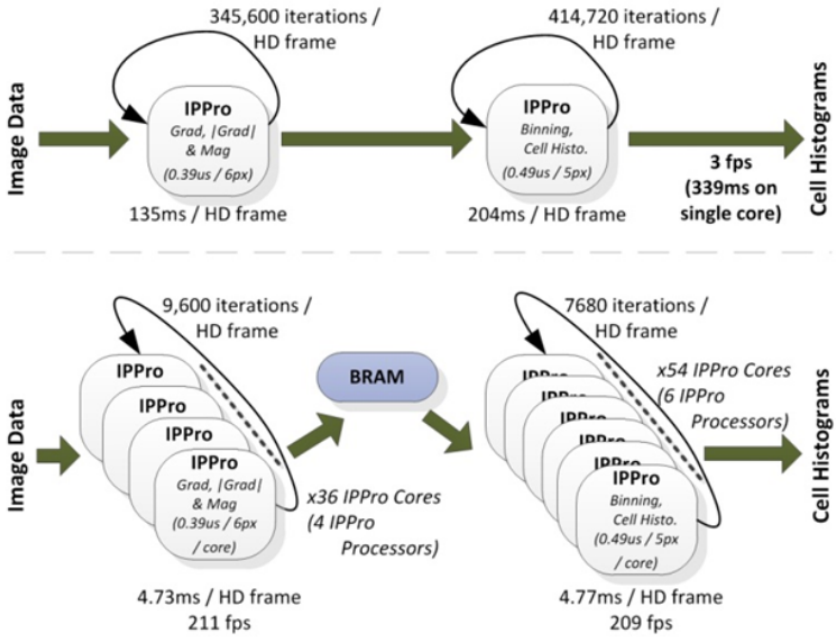


Fonte: Clemons *et al.* (2011).

Algumas arquiteturas oferecem a programabilidade do software para a implementação de sistemas de detecção de pedestres. Embora não implementada em FPGA, cita-se a EFFEX, proposta por Clemons *et al.* (2011), é uma arquitetura *multi-core* heterogênea desenvolvida para aplicações de extração de *features*. O diagrama da arquitetura está ilustrado na Figura 27. A aceleração das aplicações é realizada pelo uso de um núcleo de processamento central mais complexo, destinado a executar tarefas de alto nível, e diversos núcleos periféricos mais simples, os quais contém unidades funcionais especializadas que operam em vizinhanças de pixels. As unidades funcionais implementam operações de comparação de um para muitos, MAC de convolução e gradiente. Por meio de simulação da arquitetura em processadores ARM, aplicações de visão computacional para extração de *features*, como o SIFT, o HOG e o FAST, foram executadas. Os resultados mostram a execução de imagens com resolução 1024×768 pixels a uma taxa de $0,1$ *frames* por segundo com o núcleo a 1 GHz. As execuções do software não incluem as fases de pré-processamento e classificação, assim como também não é detalhado como é feita a programação das unidades de aceleração.

O trabalho desenvolvido por Kelly *et al.* (2014) utiliza um processador projetado especificamente para o processamento de imagens, chamado de IPPro. O núcleo deste processador tem como base a unidade de processamento de sinal Xilinx DSP48E. Estes processadores podem

Figura 28 – Arquitetura do algoritmo HOG para implementações *single*-IPPro (topo) e multi-IPPro (abaixo).



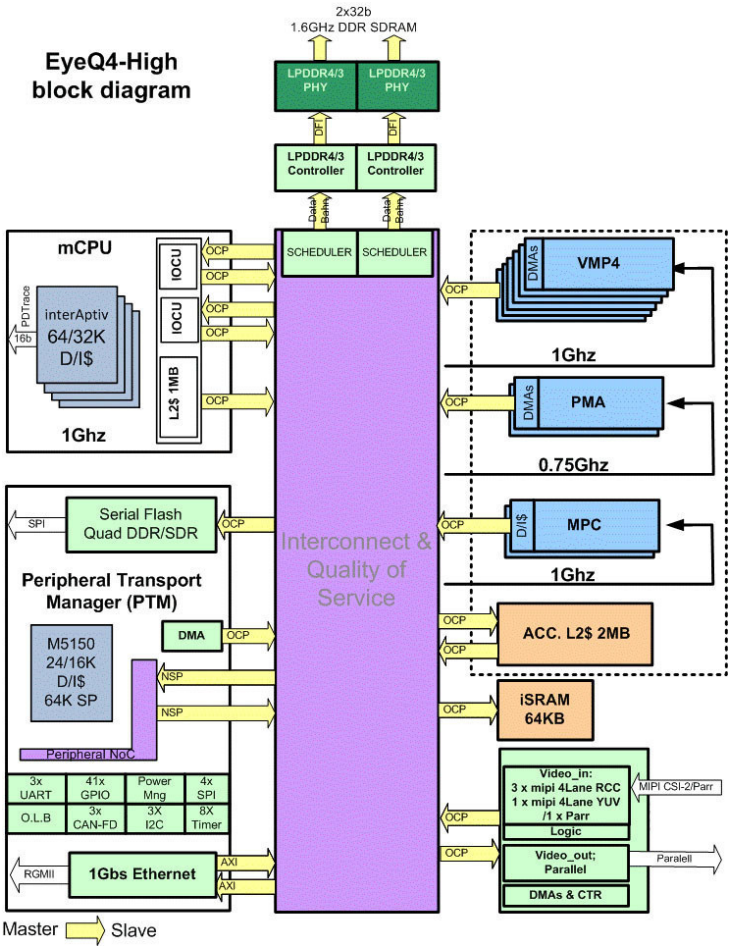
Fonte: Kelly *et al.* (2014).

ser agrupados em arranjos com interconexão via FIFO ou endereçamento direto entre os núcleos. A Figura 28 apresenta dois arranjos desses processadores para o processamento das fases iniciais do algoritmo HOG. No primeiro arranjo, o processamento é divido em dois estágios (processadores), sendo o primeiro responsável pelos cálculos de gradiente, ângulo e magnitude e o segundo estágio pelas operações de *binning* e geração de histogramas de células. A arquitetura processa imagens de entrada com resolução de 1920×1080 pixels. No segundo arranjo, cada estágio é paralelizado, sendo que o primeiro utiliza 36 núcleos IPPro e o segundo 54 núcleos. O arranjo *single-IPPro* funciona a 509 MHz e o multi-IPPro funciona a 404 MHz. Para o sistemas *single*-IPPro, o primeiro estágio é processado em 135ms e o segundo em 204ms. O sistema multi-IPPro atinge um *throughput* máximo de 264 fps, com uma dissipação de potência de $3,6W$. O algoritmo HOG foi traduzido manualmente para instruções do processador devido à falta de ferramentas de compilação, o que torna difícil o seu uso. Além disso, os resultados apresentados não incluem a cadeia de detecção completa do algoritmo HOG.

A abordagem flexível adotada neste trabalho utiliza um processador *softcore* bem estabelecido como base para acelerar os algoritmos HOG e ICF e permite a exploração de diferentes configurações, variações e combinações com outras *features*. Isto representa uma vantagem considerando o conjunto de ferramentas disponíveis para desenvolvedores. A flexibilidade dos dispositivos FPGA possibilita a exploração de diferentes arquiteturas, aceleradores e tipos de paralelismo.

Na indústria, diversas empresas têm concentrado seus esforços no desenvolvimento de

Figura 29 – Arquitetura do EyeQ4 da Mobileye.



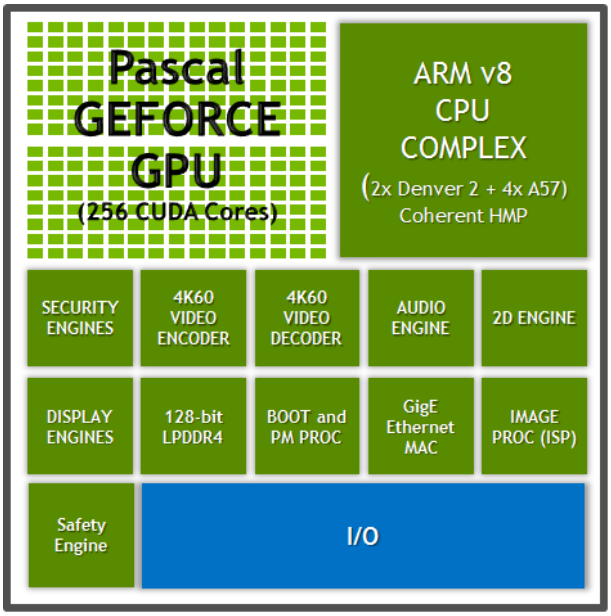
Fonte: Mobileye (2015).

arquiteturas que visam não só atender a detecção de pedestres, mas um conjunto de aplicações ADAS. Um processador que representa o estado da arte é o EyeQ4 da Mobileye (MOBILEYE, 2015), cuja arquitetura é mostrada na Figura 30. A arquitetura contém quatro núcleos de processamento com suporte em hardware a quatro *threads* cada. Esses núcleos estão acoplados a outros seis núcleos chamados de *Vector Microcode Processors* (VMP), legados das gerações anteriores do processador. São utilizados também dois tipos de aceleradores: dois núcleos *Multithreaded Processing Cluster* (MPC) e dois núcleos *Programmable Macro Array* (PMA). O primeiro, substitui uma GPU ou um acelerador OpenCL e o segundo visa acelerar funções específicas. O sistema é capaz de processar $2,5$ *teraflops* com um consumo de aproximadamente $3W$. O EyeQ4 permite também processar informações provindas de oito câmeras simultaneamente a 36 *frames* por segundo.

Uma outra arquitetura atual para sistemas ADAS é a NVIDIA DRIVE PX 2 (NVIDIA, 2016), a qual combina técnicas de aprendizado profundo, fusão de sensores e visão ao redor do veículo para facilitar o desenvolvimento de aplicações de reconhecimento de veículos e placas de trânsito, detecção de pedestres, criação de mapas 3D do ambiente ao redor do veículo e direção

autônoma. A arquitetura possui 12 núcleos de processamento, com quatro núcleos Denver 2.0 e oito núcleos Cortex-A57. Possui também uma GPU Pascal com 256 núcleos CUDA, separadas dos *chips* gráficos discretos. Essa arquitetura suporta oito máquinas virtuais simultâneas, de forma que se possa executar sistemas diferentes ao mesmo tempo.

Figura 30 – Arquitetura do NVIDIA DRIVE PX 2.



Fonte: NVIDIA (2016).

Observa-se dessas arquiteturas a tendência de processamento de vários sensores ao redor do veículo. Além disso, é cada vez mais frequente a adoção de soluções programáveis que se beneficiam de aceleradores em hardware. Nesse sentido, o sistema projetado neste trabalho acompanha o estado da arte das arquiteturas para processamento de pedestres, prevendo um sistema composto por quatro câmeras ao redor de um veículo e fornecendo a flexibilidade do software e dos FPGAs.

Faltou uma conclusão do capitulo.

CAPÍTULO

4

PROJETO E IMPLEMENTAÇÃO

Neste capítulo são descritas as implementações em hardware reconfigurável das arquiteturas para os algoritmos de detecção de pedestres HOG e ICF, descritos na subseção 3.2.1 e na subseção 3.2.2, respectivamente. Tais implementações consistem na realização de um coprojeto de hardware e software e, portanto, devem, primeiramente, submeter os algoritmos em questão à análise para identificação de regiões críticas em termos de desempenho e para extração de diferentes tipos de paralelismo. Deve-se observar que o objetivo dessas implementações é permitir que a solução final preserve a programabilidade do software em estágios importantes do algoritmo, de forma a facilitar a exploração do espaço de projeto e a adaptabilidade à especificidades dos algoritmos em diferentes cenários, especialmente dada a miríade de aplicações existentes e que impõem diferentes requisitos, e que se beneficiam do uso desses algoritmos.

A implementação das arquiteturas ocorreu em duas fases. A primeira fase teve como objetivo compreender e explorar a estrutura geral de um algoritmo clássico de detecção de pedestres, que é o HOG. Como já mencionado anteriormente, este algoritmo vem sendo estudado, adaptado e expandido de diferentes maneiras desde sua implementação original. Além disso, serve como base para o desenvolvimento de algoritmos mais modernos, como é o caso do ICF. Muitas implementações de arquiteturas para o processamento do algoritmo HOG em FPGA têm surgido nos últimos anos, sendo que a grande maioria utiliza hardware fixo, o que impossibilita sua fácil modificação. Neste trabalho, foi desenvolvida uma arquitetura consistindo principalmente em processadores *softcore* de propósito geral auxiliados por unidades aceleradoras que exploram diferentes níveis de especialização e paralelismo.

O intuito da segunda fase é construção de uma arquitetura para o processamento de um algoritmo que representa a base das técnicas do estado da arte para detecção de pedestres, que é o caso do ICF. A implementação dessa arquitetura segue o propósito definido na primeira fase, no sentido de prover a programabilidade do software em estágios importantes do algoritmo. Contudo, nesse caso, explora-se também a implementação em hardware fixo de partes do algoritmo com

o objetivo de atender às demandas computacionais de um sistema de detecção de pedestres contendo quatro câmeras, como proposto no Capítulo 1.

Primeiramente, neste capítulo, são apresentadas as plataformas de hardware embarcadas que foram utilizadas para as implementações, tanto de hardware quanto de software. Em seguida, são descritas as implementações das arquiteturas nas duas fases do trabalho.

4.1 Plataformas de Hardware

Como plataformas dos experimentos descritos neste trabalho, foram utilizadas duas placas de desenvolvimento FPGA. A primeira delas é a Terasic DE2i-150, que contém um FPGA Altera Cyclone IV 4CX150 e um processador Intel Atom N2600 integrados (TERASIC, 2013). Os recursos da placa estão distribuídos entre os dois dispositivos de forma independente e a intercomunicação é feita somente utilizando o barramento PCI *Express Gen 1*.

O FPGA Cyclone IV EP4CGX150DF31 contém 149.760 elementos lógicos, 720 blocos de memória M9K, 6.480 Kbits de memória embarcada, 8 Phase Locked Loops (PLL). Externamente ao FPGA, a placa fornece um oscilador de 50 MHz, 128 MB de *Synchronous Dynamic Random Access Memory* (SDRAM), 4 MB de *Synchronous Static Random Access Memory* (SSRAM), 64 MB de memória Flash. Outros dispositivos de hardware e *interfaces* de conexão podem ser vistos na Figura 31.

O processador Atom N2600 possui dois núcleos de processamento de 64 bits funcionando a 1.6 GHz. Cada núcleo implementa a tecnologia *Hyper-Threading* e pode, portanto, executar 4 *threads* simultaneamente. O processador contém ainda 1MB de memória *cache*, processador gráfico de 400MHz integrado e extensões do conjunto de instruções SSE2, SSE3 e SSSE3. Externamente ao processador a placa fornece 2 GB de memória SDRAM DDR3 SO-DIMM, conexão sem fio e outros recursos demonstrados na Figura 31.

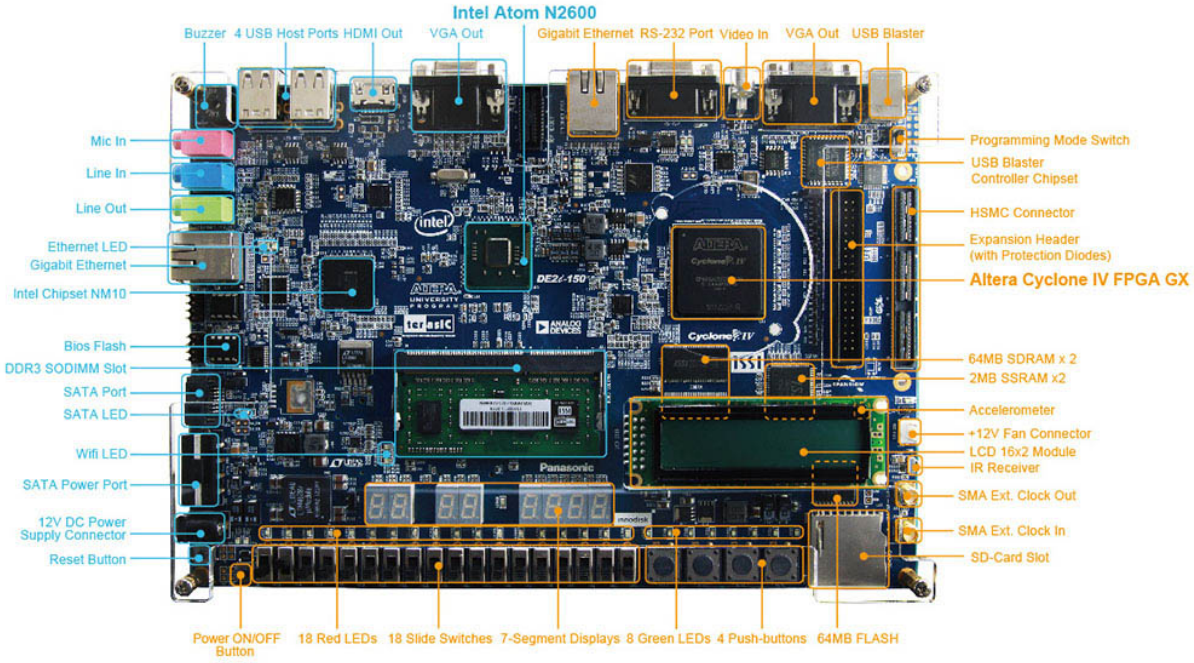
Outra placa de desenvolvimento utilizada é Stratix V GX FPGA, da Altera (ALTERA, 2012), a qual traz diversas características que possibilitam a implementação de sistemas complexos e que exijam maior desempenho, conforme pode ser visto na Figura 32.

A placa contém um FPGA Altera Stratix V, modelo 5SGXEA7K2F4C2N. Este FPGA disponibiliza 622.000 elementos lógicos, 50 Mbits de memória embutida, 28 PLLs, 36 *tranceivers* (12.5 Gbps), 696 pinos de E/S e 512 multiplicadores de 18x18 bits. O núcleo do FPGA consome 900 mV.

Para armazenamento temporário a placa possui 1152 Mbyte de memória SDRAM DDR3 com barramento de dados de 72 bits. Possui ainda 72 Mbyte de RLDRAM com barramento de dados de 18 bits, 4.5 Mbyte de SRAM QDRII e duas memórias *flash* síncronas de 512 Mbyte e barramento de dados de 16 bits.

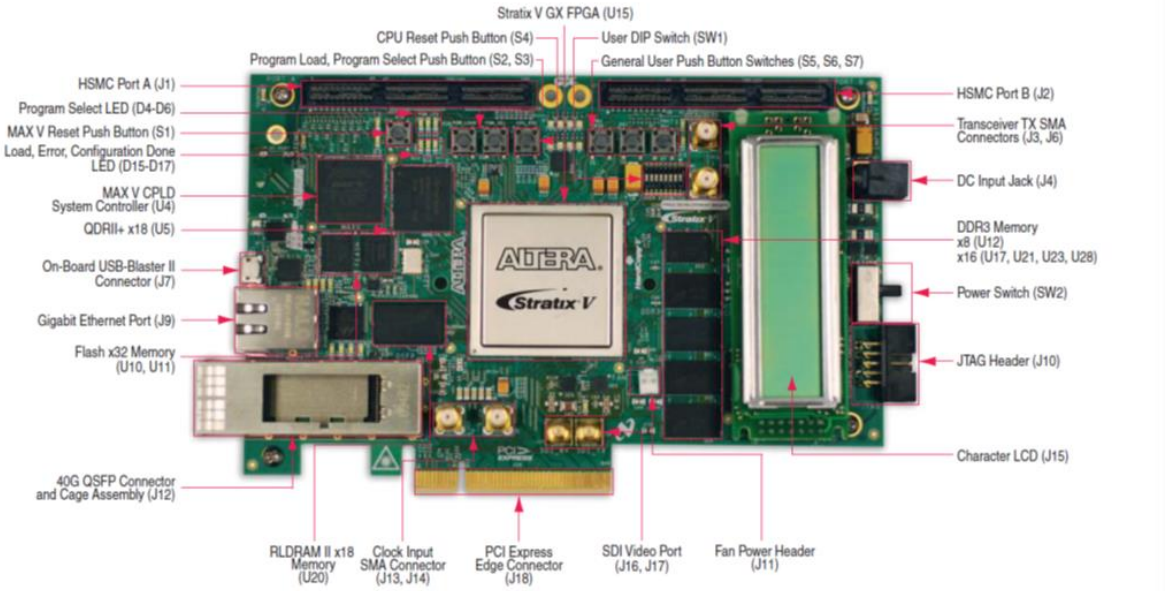
A infraestrutura de comunicação inclui um conector PCI express x8, duas portas HSMC

Figura 31 – Visão geral das características da placa de desenvolvimento DE2i-150 FPGA.



Fonte: Terasic (2013).

Figura 32 – Visão geral das características da placa de desenvolvimento Stratix V GX FPGA.



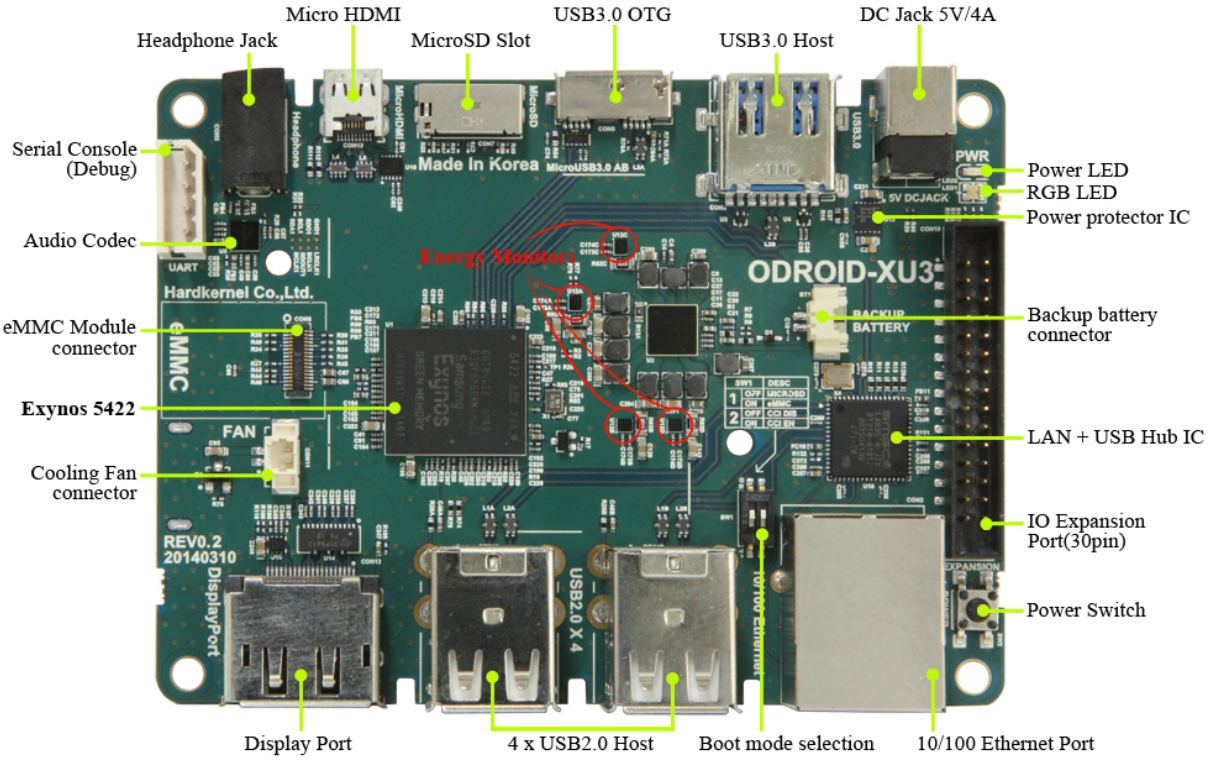
Fonte: Altera (2012).

(*High-Speed Mezzanine Card*), SDI, USB 2.0 e rede Gigabit Ethernet. Há também um conector USB-Blaster II que permite fazer o carregamento de projetos desenvolvidos no FPGA. O conector PCI Express permite conectar a placa diretamente a um *slot* em computador *Desktop*, permitindo uma maior flexibilidade no desenvolvimento de aplicações. O circuito de *clock* da placa possui

osciladores programáveis e permite frequências de 50, 100 e 125 MHz.

Uma terceira plataforma de desenvolvimento embarcada foi utilizada para testes com paralelismo de dados. A placa ODROID XU3 (ODROID, 2015) contém um processador Exynos5 Octa Core (big.LITTLE), sendo que quatro núcleos são de um ARM Cortex-A15 a 2 GHz e os outros quatro são de um ARM Cortex-A7 a 1.3 GHz. A placa fornece 2 GB de memória RAM a 933 MHz e pode ser vista com maiores detalhes na Figura 33.

Figura 33 – Visão geral das características da placa de desenvolvimento ODROID-XU3.



Fonte: Odroid (2015).

4.2 Fase 1 - Implementação do Algoritmo HOG em FPGA

A implementação da arquitetura para o HOG inicia-se com a análise do algoritmo, a qual visa descobrir os gargalos de processamento (ou regiões críticas) do código desenvolvido para encontrar as funções que necessitem ter sua execução acelerada. O processo de aceleração de funções pode incluir tanto otimizações de software quanto a construção de aceleradores em hardware. A análise visa também identificar o potencial de paralelização do algoritmo e quais tipos de paralelismo podem ser explorados para otimização do desempenho. Em seguida, são descritas as implementações de software e de hardware realizadas no coprojeto do algoritmo HOG, mostrando as transformações e especializações necessárias para uma execução mais eficiente.

4.2.1 Análise do Algoritmo

Para a realização da análise do algoritmo foi desenvolvida uma versão de referência do algoritmo HOG, baseada na implementação da biblioteca OpenCV (BRADSKI, 2000). Nela, foram utilizadas somente bibliotecas padrão da linguagem C++, embora tenham sido mantidas algumas otimizações realizadas nas funções do OpenCV, tais como uso de tabelas de consulta para os coeficientes das normalizações e o uso do formato do vetor de classificação SVM. O motivo da não utilização de bibliotecas de processamento de imagem disponíveis é permitir uma maior portabilidade do código para análise em diferentes plataformas e compiladores.

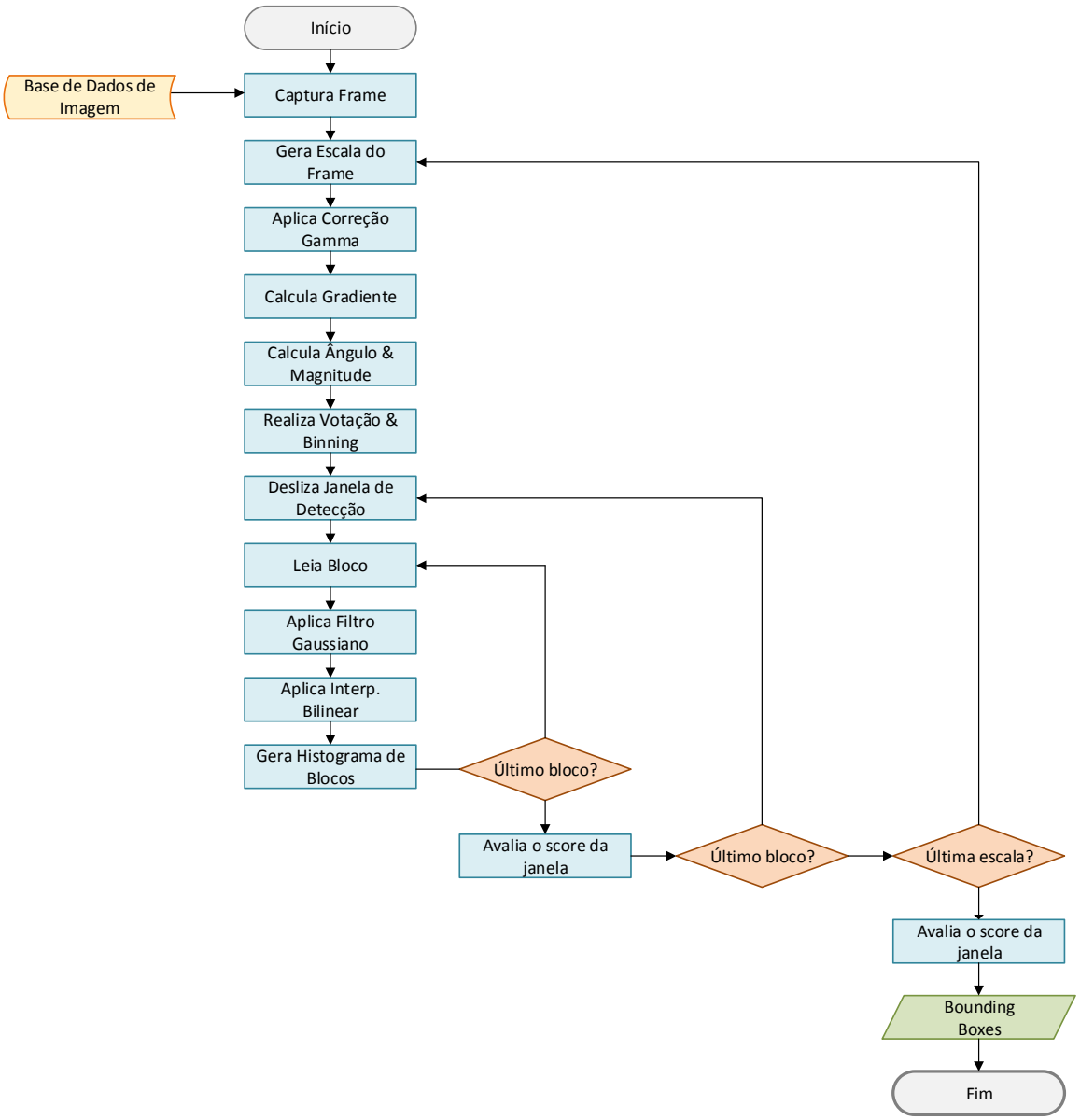
Como descrito na subseção 3.2.1, o algoritmo HOG possui uma série de parâmetros cujos valores influenciam na qualidade da detecção e desempenho do algoritmo. Dependendo do valor dos parâmetros, a quantidade de processamento necessária pode aumentar ou diminuir consideravelmente. Os parâmetros utilizados neste trabalho são, em sua maioria, os mesmos que obtiveram os melhores resultados na implementação original de Dalal e Triggs (2005). O tamanho adotado para a janela de detecção é de 64×128 pixels, com blocos contendo 2×2 células de 8×8 pixels cada. Blocos vizinhos se sobrepõem em 8 pixels tanto verticalmente quanto horizontalmente e, portanto, em uma janela de detecção há 105 blocos. Como cada célula gera um histograma de 9 *bins*, o descritor final de uma janela possui 3.780 valores, que é o mesmo tamanho do vetor de classificação SVM. A janela de detecção se move em ambos os eixos da imagem com um deslocamento de 8 pixels, o que significa processar 3.285 janelas por *frame*. O algoritmo processa imagens de entrada em níveis de cinza com resolução de 640×480 pixels, selecionadas a partir da base de dados de pedestres do INRIA. É utilizada nesta implementação uma detecção em várias escalas da imagem de entrada, de forma a detectar objetos de diferentes tamanhos. O fator de escala utilizado é de 1.05.

A Figura 34 ilustra a sequência de passos do algoritmo HOG implementado. É interessante notar que o algoritmo possui três grandes laços principais. O mais externo tem como objetivo iterar sobre as várias escalas da imagem de entrada. O laço intermediário controla o deslizamento da janela de detecção sobre a imagem. O terceiro laço é responsável por iterar sobre cada bloco contido em uma janela de detecção para realizar o cálculo do descritor. Ao final do algoritmo, uma função de supressão não máxima é aplicada sobre as janelas de detecção classificadas como positivas para a presença do objeto, eliminando assim, detecções redundantes.

4.2.1.1 Perfil de Execução do Algoritmo

A análise da versão de referência do algoritmo HOG permite um melhor entendimento das características da implementação que mais afetam o seu desempenho. Assim, tem-se um direcionamento daquilo que pode ser feito em termos de arquitetura para se obter melhorias na execução do algoritmo. Neste sentido, o perfil de execução do algoritmo foi levantado e estudado, bem como a resolução dos gargalos de processamento e a possibilidade de paralelização de suas tarefas.

Figura 34 – Fluxograma do algoritmo HOG para detecção de pedestres em várias escalas.



Fonte: Elaborada pelo autor.

O perfil de execução permite saber, entre outras informações, quais das funções de um programa são responsáveis por gastar a maior parte de seu tempo. A fim de traçar o perfil de execução do algoritmo HOG, a versão de referência foi executada utilizando a ferramenta GNU *Profiler* (GPROF) (FENLASON; STALLMAN, 1998). Esta ferramenta insere, automaticamente, código instrumentado ao programa sob análise durante o processo de compilação. Ao executar o programa, o GPROF gera um arquivo contendo o relatório do monitoramento dessa execução, o qual exibe informações a respeito do tempo de execução de cada função e um grafo de chamadas de funções.

Primeiramente, o perfil da versão de referência foi extraído pela execução do algoritmo

em duas plataformas computacionais diferentes. Uma delas é um computador *desktop* com processador Intel Core i5-650, com frequência de 3.20 GHz, 4M de memória Cache, 4 GB de memória RAM e 2 núcleos com *hyper-threading*. A outra plataforma é embarcada e contém o processador ARM Cortex-A7 descrito na seção 4.1. O experimento nestas plataformas permite uma rápida exploração e análise inicial do algoritmo. Os dados do experimento baseiam-se no tempo de execução de cem imagens contendo pedestres selecionados da base de dados INRIA. A execução do algoritmo é feita em somente um núcleo em ambos processadores e considera múltiplas escalas da imagem de entrada. O relatório emitido pela ferramenta GPROF indica que os tempos de execução das etapas do algoritmo HOG estão distribuídas conforme informado na Tabela 3.

Tabela 3 – Resultados do *profiling* para a versão de referência do algoritmo HOG executada nos processadores Intel I5 e ARM A7.

Funções	**Intel I5**	**ARM A7**
Processamento de Blocos	85,5%	88,4%
Normalização L2-hys	9,7%	6,5%
SVM	1,5%	1,4%
Gradiente e *Binning*	2,0%	2,2%
Redimensionamento	1,0%	0.6%
Outras	0,3%	0.9%
Tempo por *frame*	5*s*	16*s*

Fonte: Elaborada pelo autor.

Para facilitar o entendimento das informações de *profiling*, algumas funções foram agrupadas de acordo com o seu contexto na implementação da cadeia de detecção. Mais especificamente, o grupo denominado "Gradiente e *Binning*" é composto pelas funções de extração de gradientes, cálculos de magnitude e ângulo, votação ponderada e *binning*. O grupo denominado "Processamento de Blocos" inclui as funções de filtro Gaussiano, interpolação bilinear e geração de histogramas. O grupo "Outras" inclui funções auxiliares que, sozinhas, não demonstraram um influência significativa no tempo de execução. As demais funções listadas na Tabela 3 correspondem diretamente à etapas do algoritmo HOG.

A partir das informações apresentadas pela Tabela 3, percebe-se que, apesar das diferenças entre as plataformas, ambas obtiveram perfis de execução do algoritmo HOG bastante semelhantes. Observa-se que a maior parte do tempo de execução é dominada pelas funções de processamento de blocos, as quais consomem uma parcela de tempo aproximadamente dez vezes maior do que a segunda função com maior custo. Portanto, esta é a região crítica do algoritmo e tem uma maior relevância para ser analisada com o objetivo de melhorar o desempenho.

As funções de processamento de blocos possuem como entrada os votos ponderados calculados a partir dos blocos da janela de detecção e os pesos para as funções de normalização e de interpolação. Ambas as funções caracterizam-se, em sua maior parte, por executarem sequências de operações do tipo multiplica-acumula em ponto flutuante, cálculos de índices e acessos à memória. Como produto final, as funções produzem histogramas de bloco que, por sua vez, são concatenados para formar o descritor da janela de detecção. As funções deste grupo são implementadas dentro do laço mais interno do algoritmo ilustrado no diagrama da Figura 34. Nele, as funções são executadas uma vez para cada bloco dentro de uma janela de detecção. Conforme os parâmetros do HOG considerados nessa implementação, isto significa processar 105 blocos por janela ou 344.925 blocos apenas na primeira escala do *frame*. Ao considerar todas as escalas (ao todo são 28) o número de blocos processados sobe para 2.716.12, aumentando consideravelmente a quantidade de dados processada.

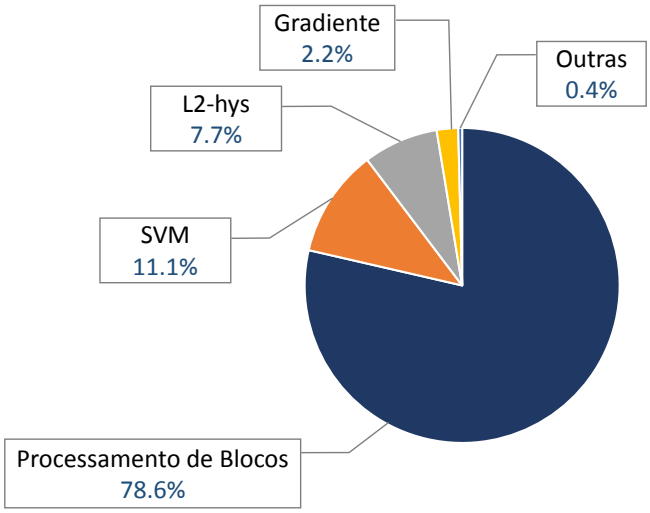
Uma forma encontrada para reduzir o tempo de execução dessas funções do algoritmo é promover o reaproveitamento de dados já processados. Para isso, cabe observar que, ao longo da execução do programa, um mesmo bloco pode pertencer a várias janelas de detecção. É importante também perceber que as operações realizadas pelas funções de interpolação bilinear e filtro Gaussiano atuam sobre os dados de cada bloco separadamente. Dessa forma, uma vez que um determinado bloco gera sempre o mesmo histograma quando processado, este pode ser reaproveitado para compor o descritor de diferentes janelas de detecção. Com esta melhoria, cada bloco da imagem de entrada é processado uma única vez e assim, o número de blocos processados é reduzido para 4.661, considerando a execução de uma escala, e reduzido para 46.108, levando-se em conta 28 escalas da imagem de entrada. Esta redução de duas ordens de magnitude acarreta em uma diminuição considerável do tempo de execução das funções do grupo de Processamento de Blocos, como pode ser visto com maiores detalhes na seção 5.1.

A segunda função na Tabela 3 é responsável pela normalização l2-Hys, e suas operações consistem principalmente em multiplicações, somas e inversa da raiz quadrada em ponto flutuante. A terceira função (SVM) realiza o produto escalar do descritor HOG com o vetor de classificação, o qual se resume a um laço que contém operações do tipo multiplica-acumula também em ponto flutuante. Ambas as funções recebem como entrada os histogramas de blocos gerados pelas funções de processamento de blocos.

Em seguida, realiza-se a execução da versão de referência em um sistema baseado em FPGA contendo um processador *softcore*. O propósito deste experimento é compreender o desempenho da aplicação em um ambiente com recursos mais restritos, quando comparado aos sistemas utilizados no experimento anterior, e analisar seu comportamento ao interagir com os recursos oferecidos pelo processador *softcore*. Para isso, utiliza-se a placa de desenvolvimento FPGA DE2i-150, apresentada na seção 4.1. No FPGA Cyclone IV é instanciado um processador Nios II com configuração de núcleo *fast*, frequência de 150MHz, 4KB de cache de instruções, 2 KB de cache de dados, RAM *on-chip* e acesso a 128 MB de memória externa via controlador

SDRAM para manipulação de estruturas de dados intermediárias. No núcleo de processamento foram habilitadas divisões e multiplicações por hardware. O processador se comunica via barramento Avalon a uma ROM *on-chip*, a qual armazena LUTs, o vetor de classificação SVM e uma imagem de entrada. Uma vez que o algoritmo HOG faz uso intensivo de operações em ponto flutuante, o processador Nios II é também auxiliado pelo hardware de ponto flutuante da Altera (Altera Corporation, 2015). Este componente estende o conjunto de instruções do Nios II com implementações em hardware de instruções de ponto flutuante, permitindo um melhor desempenho quando comparado à emulação de ponto flutuante padrão do processador. Durante os experimentos, verificou-se uma melhoria de 10× utilizando esse hardware ao processar a versão de referência.

Figura 35 – Resultados do *profiling* para a versão de referência do algoritmo HOG executada no processador Nios II considerando uma única escala de imagem.



Fonte: Elaborada pelo autor.

Na análise do tempo de execução da versão de referência no processador Nios II, verificou-se que o processamento de múltiplas escalas da imagem de entrada resulta em um tempo proibitivo de milhares de segundos por *frame*. Assim, para simplificar o levantamento do perfil de execução do algoritmo neste sistema, considera-se somente uma escala da imagem de entrada. Embora esta ação diminua a quantidade de dados processados, o perfil de execução do algoritmo HOG é pouco afetado pois o processamento, em sua maior parte, é proporcional ao tamanho da imagem de entrada e não à quantidade de objetos na cena. Consequentemente, o processamento de diferentes cenas não altera a demanda computacional das funções do algoritmo. A exceção à esta regra é a função que realiza a supressão não máxima de todos os resultados provenientes da detecções positivas de objetos. Contudo, no experimento inicial com os processadores Intel I5 e ARM A7, a contribuição desta função para o tempo de execução total mostrou-se irrelevante.

O gráfico ilustrado na Figura 35 mostra as porcentagens de tempo que cada função ou

grupo de funções do algoritmo HOG gastam ao processar uma imagem de entrada. Pode-se observar que os resultados obtidos nesse perfil seguem o que foi levantado no experimento anterior, com exceção da função de classificação SVM, que têm sua contribuição para o tempo de execução bastante aumentada. Neste caso, o esforço de otimização deve estar concentrado na execução eficiente de multiplicações seguidas de adições em ponto flutuante e na eficiência do acesso à memória onde está armazenado o vetor de classificação. O tempo por *frame* para o processador Nios II é de 94,9 segundos.

4.2.1.2 Exploração de Paralelismo

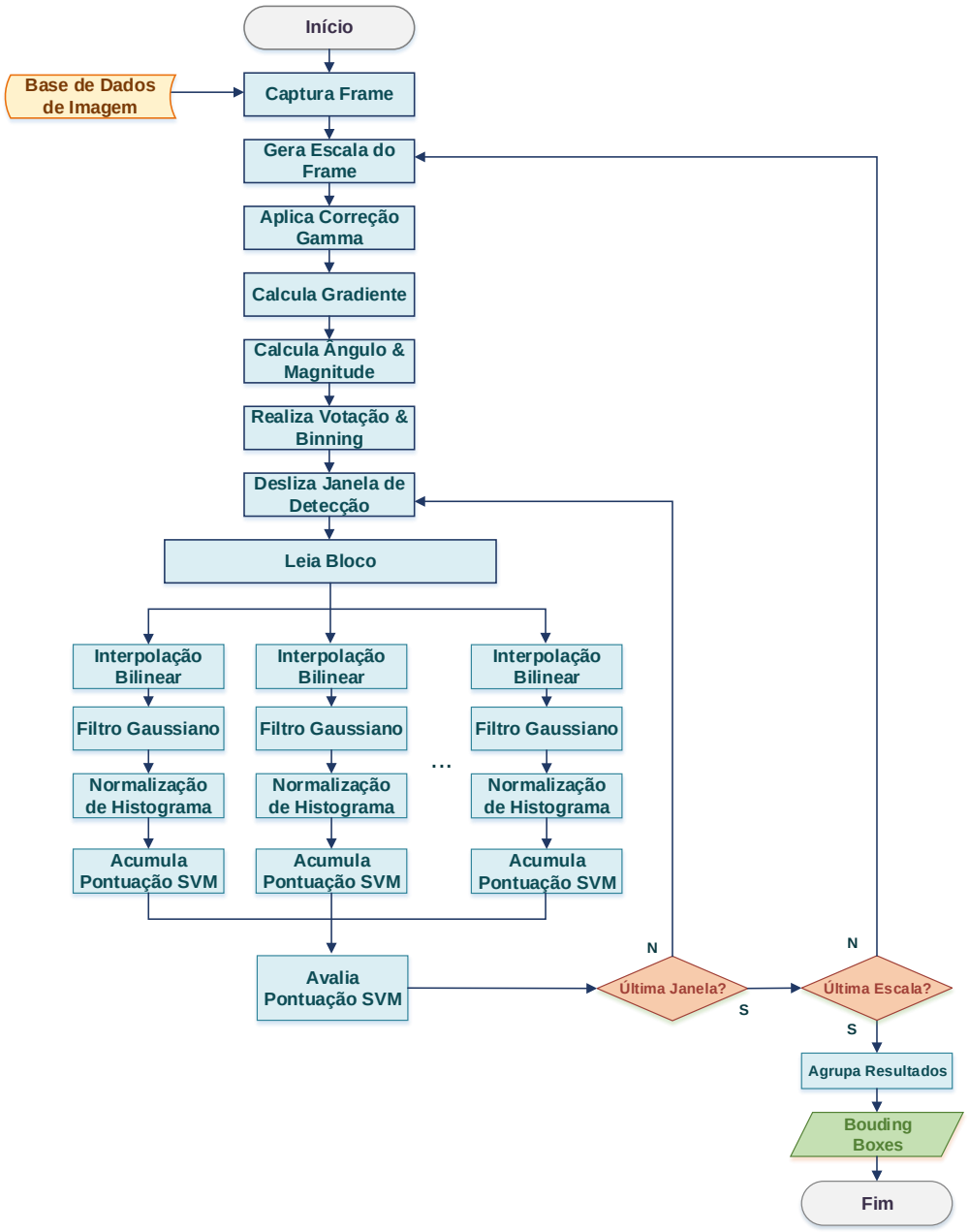
Arquiteturas de computadores atuais contém múltiplos núcleos de processamento *on-chip*, sejam estes homogêneos ou heterogêneos. Para que a aplicação se beneficie da presença de vários desses núcleos, é necessário investigar o paralelismo e a escalabilidade do algoritmo HOG. O desafio, neste caso, consiste em extrair o paralelismo inerente à aplicação e, ao mesmo tempo, reduzir um consequente desequilíbrio de carga. Analisa-se aqui a extração de dois tipos de paralelismo do algoritmo HOG: de dados e de *pipeline*.

Na aplicação desenvolvida foram identificados três granularidades diferentes de dados que podem ser utilizadas para obter fluxos paralelos na cadeia de detecção do HOG. A primeira abordagem, que possui uma granularidade mais fina e atua no nível de blocos, consiste em processar todos os blocos dentro de uma determinada janela de detecção em paralelo (veja Figura 36). Contudo, quanto mais fina a granularidade, mais intensivas são as operações sobre dados compartilhados. Neste algoritmo, isto ocorre, principalmente, devido à sobreposição entre os blocos e à necessidade de reunir os histogramas de blocos gerados para cálculo da etapa final de classificação.

A segunda abordagem é um equilíbrio entre soluções de granularidades fina e grossa. Nela, a paralelização ocorre em nível de janelas de detecção, ou seja, o conjunto de blocos que forma uma janela é processado em paralelo aos os blocos que formam outras janelas, conforme ilustrado na Figura 37. A sobrecarga dessa abordagem ocorre devido ao compartilhamento de blocos entre diferentes janelas, ou à replicação desses blocos, o que gera uma necessidade de armazenamento temporário em memória muito maior. Contudo se comparada à primeira abordagem, a sobrecarga derivada do compartilhamento de dados é menor.

A terceira abordagem consiste na paralelização em nível de escalas e possui a granularidade de dados mais grossa entre as três. Neste caso, cada escala da imagem de entrada é processada paralelamente, evitando assim, o compartilhamento de dados de uma mesma imagem entre diversos núcleos de processamento (veja Figura 38). O lado negativo desta abordagem encontra-se no fato de que é necessário ter um bom balanceamento de carga entre os núcleos, sob o risco de não se ter um bom aproveitamento de todos os recursos computacionais oferecidos pela arquitetura. Isto pode ocorrer pois as imagens de entrada para cada núcleo possuem tamanhos diferentes e, portanto, representam cargas com diferentes pesos.

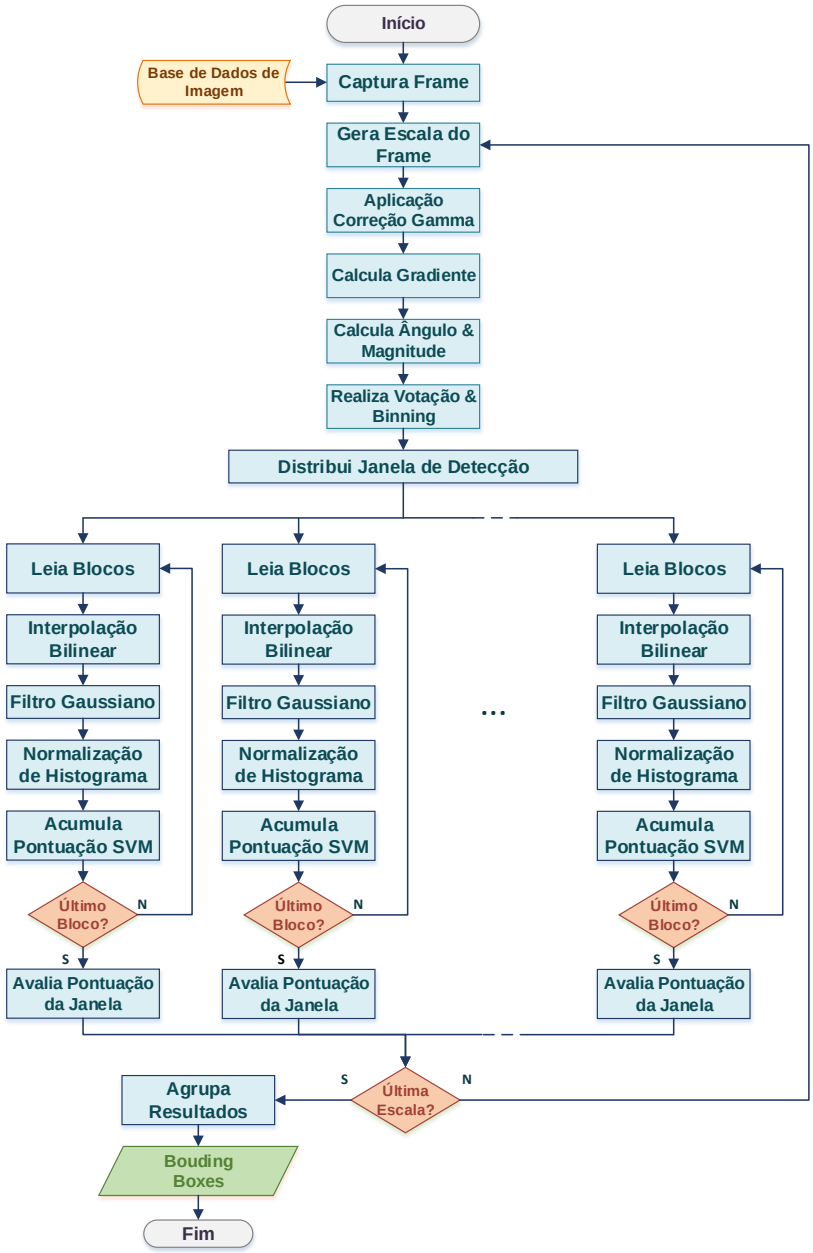
Figura 36 – Paralelismo em nível de blocos para o algoritmo HOG.



Fonte: Elaborada pelo autor.

A Figura 39 mostra um gráfico com o desempenho por *frame* da execução do algoritmo HOG com a exploração de cada um dos tipos de paralelismo. Para isto, foi utilizado um computador *desktop* com um processador Intel Core i5-650, com frequência de 3.20 GHz, 4M de memória Cache, 4 GB de memória RAM e 2 núcleos com *hyper-threading*. No experimento, são criadas diferentes quantidades de *threads*, por meio da biblioteca OpenMP, para avaliar cada tipo de paralelismo. Observa-se que o paralelismo em nível de janelas obteve um melhor *speedup* $(2,6\times)$ com 16 *threads* em execução. Este resultado mostra a importância de se avaliar

Figura 37 – Paralelismo em nível de janelas para o algoritmo HOG.

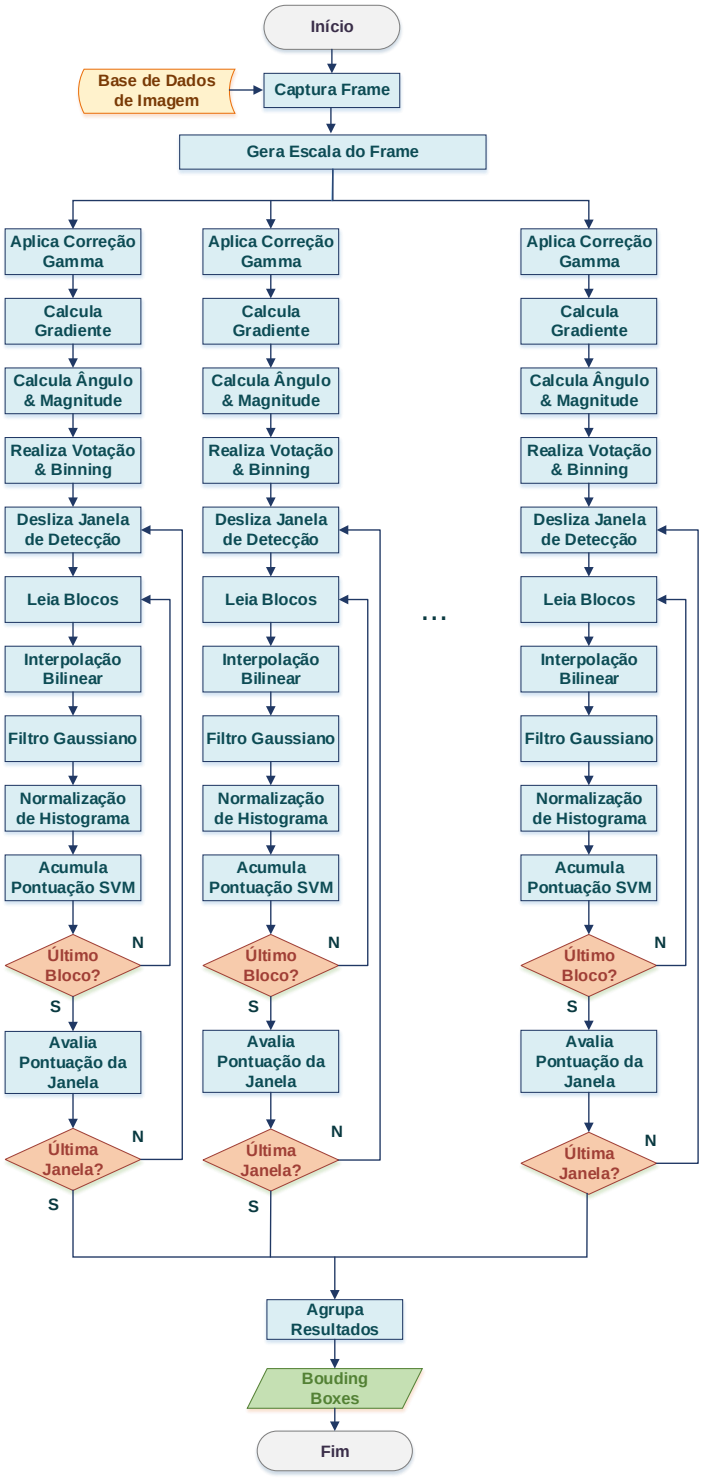


Fonte: Elaborada pelo autor.

os *trade-offs* entre granularidade dos dados, compartilhamento de memória e arquitetura para obtenção do máximo de desempenho por meio de paralelismo.

A cadeia de detecção de objetos do algoritmo HOG caracteriza-se por sua série de estágios consecutivos que devem realizar operações sobre os dados de entrada na ordem em que foram fornecidos. Em um sistema *multi-core* pode-se distribuir esses estágios entre os núcleos de processamento e estabelecer uma forma de intercomunicação desses núcleos de maneira que a sequência de execução dos estágios permaneça a mesma. Com essa estratégia, torna-se possível

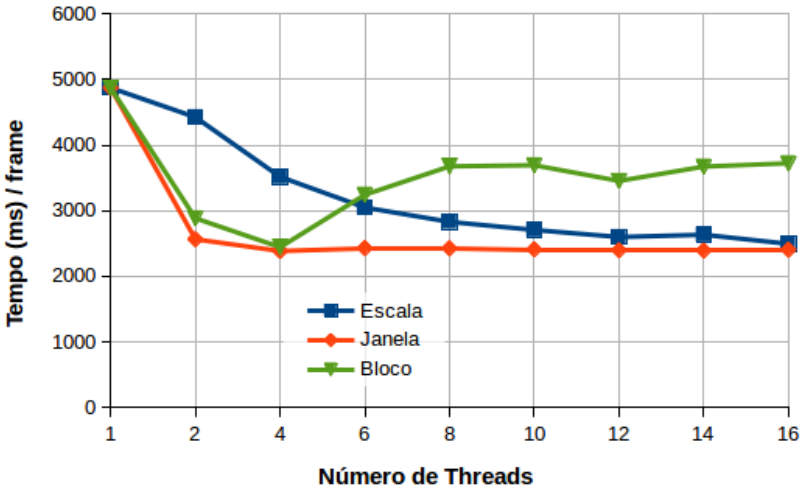
Figura 38 – Paralelismo em nível de escalas para o algoritmo HOG.



Fonte: Elaborada pelo autor.

explorar o paralelismo de *pipeline* do algoritmo, permitindo que os dados de entrada fluam pelos núcleos de processamento, enquanto estes executam paralelamente suas funções.

Figura 39 – Comparação entre os diferentes níveis de paralelismo para o algoritmo HOG.



Fonte: Elaborada pelo autor.

4.2.2 Implementação em FPGA

4.2.2.1 Otimização do Software

A análise do perfil de execução da versão de referência mostrou algumas direções para aumentar o desempenho do algoritmo HOG. Antes de seguir as direções levantadas, algumas modificações menores são realizadas no código de referência para melhoria de desempenho. Primeiramente, adotou-se números em ponto flutuante com precisão simples ao invés de dupla, o que influência $1,2\times$ no aumento de desempenho. Apesar dessa mudança acarretar em perda de precisão na representação dos números, observou-se que a precisão de detecção foi mantida, como em Ma, Najjar e Roy-Chowdhury (2015).

Algumas funções padrão da linguagem C++ também foram substituídas por código mais eficiente. Por exemplo, a função *floorf()* da biblioteca *Math.h* gera aproximadamente quinhentas linhas de código *assembly* do processador Nios II. Contudo, no algoritmo, esta função é usada dentro de um intervalo restrito de valores, de modo que pode-se substituir a operação de divisão existente nessa função por código condicional mais simples, que gera apenas 5 linhas de código *assembly* do Nios II. Além disso, o hardware de ponto flutuante do processador Nios II oferece implementações bastante eficientes de diversas funções, como raiz quadrada e arredondamentos. A utilização dessa unidade de hardware fica transparente ao programador quando está habilitada, ficando a cargo do compilador gerar as instruções que acessam esse hardware automaticamente. O uso de *arrays* de estruturas bem alinhadas para agrupar dados sobre *bins*, votos e pesos também se mostrou vantajoso para aumento de desempenho. Esta modificação resultou em um *speedup* de $1.3\times$ quando comparada a uma implementação com *arrays* homogêneos separados.

Observou-se que o uso de tabelas de consulta (LUTs) para os coeficientes da interpolação bilinear e do filtro Gaussiano resultou em um aumento de desempenho de 4%. O reuso de

resultados intermediários também é necessário para minimizar acessos à memória e economizar recursos de memória do FPGA. Assim como em Kelly *et al.* (2014), o paralelismo temporal foi explorado por meio da concatenação dos cálculos dos gradientes, dos ângulos, das magnitudes e dos votos ponderados.

A partir dos resultados de *profiling* obtidos para a versão de referência, concluiu-se que a região crítica do algoritmo consistia no grupo de funções responsáveis pelo processamento de blocos. A solução apresentada para este gargalo consistia no reuso de dados para reduzir a quantidade de blocos processados por janela de detecção. Para eliminar o processamento redundante de blocos, é utilizado um *buffer* que armazena dados de blocos já processados. Este *buffer* atua como um suporte ao movimento de deslizamento da janela de detecção, permitindo o reuso de blocos antigos e processando somente os novos. Para isto, assume-se que os blocos são processados coluna a coluna, iniciando do bloco do topo à esquerda. Neste caso, as janelas de detecção também deslizam em colunas, começando do topo à esquerda.

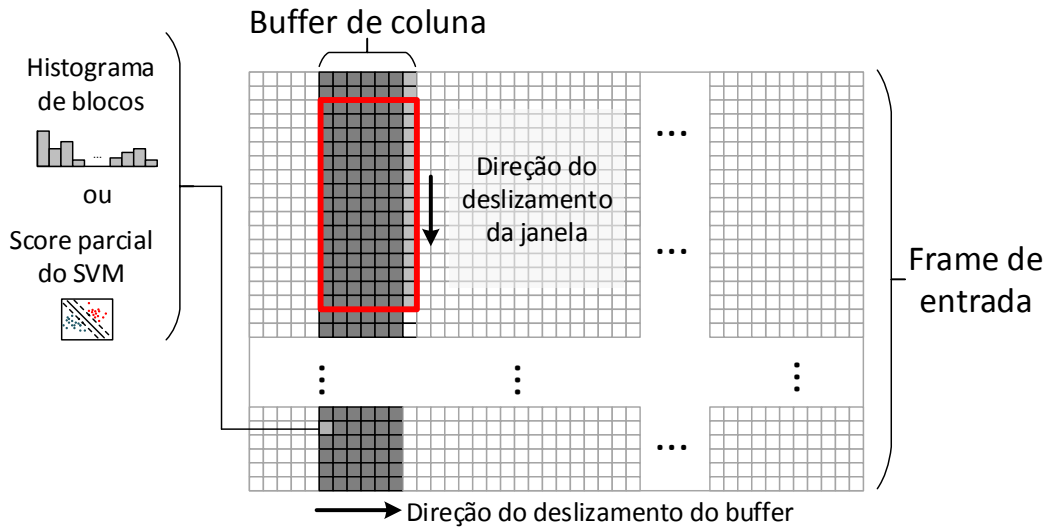
À medida em que a janela desliza, somente um novo bloco necessita ser processado. A primeira janela de detecção em uma coluna é a única que lê 15 novos blocos (que é a altura da janela) da memória. Uma vez que todos os blocos em uma janela de detecção são lidos e processados, o último *score* parcial do classificador SVM é calculado e o algoritmo pode retornar a resposta sobre a detecção de pedestres.

Duas abordagens diferentes são examinadas para a implementação do *buffer*, conforme ilustrado no diagrama da Figura 40. A primeira é similar à implementação em hardware de Ma, Najjar e Roy-Chowdhury (2015) e define cada elemento do *buffer* como um histograma de blocos normalizado de tamanho 1×36. Nesta abordagem, para cada janela de detecção, o algoritmo busca cada histograma de blocos armazenado no *buffer* e calcula o *score* SVM parcial. Quando o último novo bloco para aquela janela é processado, seu histograma é armazenado no *buffer* e o *score* final é calculado e avaliado.

A segunda abordagem está baseada na implementação em hardware de Hahnle *et al.* (2013). Nela, ao invés de armazenar histogramas de blocos, o *buffer* armazena *scores* parciais do classificador SVM para todas as janelas de detecção com blocos já processados. Quando um novo bloco é processado, o *score* parcial de cada janela que contém aquele bloco é atualizado. Se o último bloco de uma janela for processado, seu *score* final é calculado e avaliado.

Para os parâmetros do algoritmo HOG adotados, a primeira abordagem requer 58 KB de memória para armazenar 7×59 histogramas de blocos utilizando ponto flutuante com precisão simples. A segunda abordagem reduz bastante o uso de memória, com um *buffer* de tamanho 1,2 KB. Com respeito ao desempenho, a primeira abordagem é $1,6\times$ mais rápida e será utilizada nas comparações feitas em diante neste trabalho. A referência a versão melhorada do código de referência do algoritmo HOG será pelo nome de *otimizada*. A Tabela 4 mostra o impacto das otimizações no perfil de execução do algoritmo HOG para o código executado no processador Nios II. Observa-se que o uso do *buffer* reduziu em duas ordens de magnitude o tempo gasto

Figura 40 – Diagrama do funcionamento do *buffer* de coluna.



Fonte: Elaborada pelo autor.

com as funções de processamento de blocos, tendo impacto semelhante também nas funções posteriores da cadeia de detecção. Como é adotada uma abordagem flexível baseada em software, outras transformações de código ou simplificações de funções poderiam ser facilmente aplicadas, embora com possível perda de precisão.

Tabela 4 – Comparação dos resultados do *profiling* para as versões de referência e otimizada do algoritmo HOG executadas no processador Nios II.

Funções	**Referência**		**Otimizada**	
	%	**Ciclos**	**%**	**Ciclos**
Processamento de Blocos	78,6	1.1×10^{10}	20,5	1.6×10^{8}
SVM	11,1	1.6×10^{9}	65,0	5.0×10^{8}
Normalização L2-hys	7,7	1.1×10^{9}	1,3	1.0×10^{7}
Gradiente e *Binning*	2,2	3.2×10^{8}	11,2	8.7×10^{7}
Outras	0,4	5.7×10^{7}	2.0	1.5×10^{7}
Todas	100	1.4×10^{10}	100	7.7×10^{8}

Fonte: Elaborada pelo autor.

4.2.2.2 Implementação do Hardware

A partir da versão otimizada do software, iniciou-se a exploração de processamento paralelo para atingir melhores níveis de desempenho, aproveitando-se da flexibilidade do FPGA. Para

isso, as funções do HOG são distribuídas entre arranjos de processadores Nios II organizados em *pipeline*. Arranjos com um, dois, três e quatro estágios de processadores Nios II são implementados. A Figura 42 mostra a arquitetura desses arranjos. Cada processador contém memória cache de dados (D$) e de instruções (I$), hardware de ponto flutuante (FPH2) e memórias *on-chip* locais. De acordo com a posição do processador no *pipeline*, este pode ter um ou mais módulos FIFO utilizados para intercomunicação dos núcleos de processamento. Somente o primeiro processador Nios II do *pipeline* tem acesso à memória externa por meio do módulo controlador da SDRAM, onde são armazenados dados do programa. Além disso, dependendo da função processada, pode-se adicionar memórias de dados fortemente acopladas (*Tightly-Coupled Data Memories* (TCDM)) para armazenar dados usados frequentemente, como tabelas de consulta (LUTs) para filtros, normalizações, interpolações e classificador SVM. Como estas memórias não se encontram conectadas a nenhum barramento, acessos podem ser completados em um único ciclo.

Nesta arquitetura, a intercomunicação dos núcleos é realizada por meio de módulos FIFO. Estes módulos foram implementados como instruções customizadas do processador Nios II, de forma que, ao acessar a FIFO para leitura ou escrita, não é necessário acesso ao barramento de sistema. Foram implementados dois módulos FIFO, um deles com uma interface mestre (FIFOMST) para enviar dados e um com interface escrava (FIFOSVL) para receber dados. Do ponto de vista do software, qualquer acesso à FIFO é realizado por uma chamada *put()* no lado mestre ou *get()* no lado escravo. Cada uma dessas instruções customizadas demora dois ciclos de *clock* para completar, a menos que a FIFO esteja cheia ou vazia, nos casos das interfaces mestre e escravo, respectivamente. Nesta situação as instruções bloqueiam o processador até que haja espaço para um novo *put()* ou dados disponíveis para um *get()*.

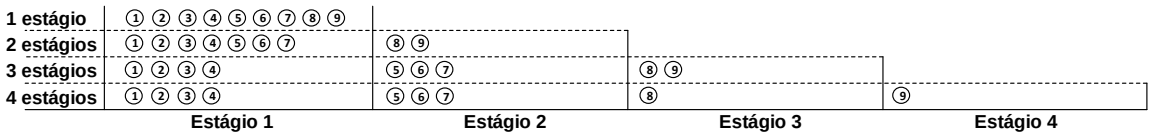
O balanceamento das cargas (funções HOG) entre os processadores da arquitetura utiliza informações colhidas no *profiling* da aplicação. Para o *pipeline* de dois estágios, os melhores resultados são obtidos mapeando-se a normalização L2-hys e o cálculo do *score* SVM no processador (P2). O processador P1 realiza a leitura do *frame*, processa os estágios desde a correção gamma até a interpolação bilinear. O software em P2 implementa um *buffer* de coluna para reuso de dados e obtenção de melhor desempenho.

Para a arquitetura com *pipeline* de três estágios, o primeiro é responsável por ler o *frame* de entrada da memória externa, computar os votos e os *bins* para cada pixel, enviando os resultados ao estágio 2. Este, recebe os dados das células e os armazena em uma versão mais simples do *buffer* de coluna, promovendo o reuso de dados. Esses dados são reunidos em blocos, são aplicados o filtro Gaussiano e a interpolação bilinear. Quando o bloco já foi processado, o histograma correspondente é enviado ao estágio 3, que o normaliza e o armazena no *buffer* de coluna local, realizando então a classificação SVM.

Finalmente, no *pipeline* de 4 estágios, os primeiros dois estágios executam as mesmas funções da arquitetura com 3 estágios. O terceiro estágio é responsável por aplicar a normalização

L2-hys e o quarto estágio realiza a classificação SVM. Ambos os estágios implementam *buffers* de coluna para reuso de dados e obtenção de melhor desempenho. Em todos os casos, os módulos FIFO estão configurados para armazenar 1K palavras de 32 bits. A Figura 41 mostra o melhor mapeamento obtido para as funções do HOG considerando *pipelines* de até quatro estágios. Os números mostrados nessa figura correspondem aos estágios do HOG apresentados anteriormente na Figura 24.

Figura 41 – Mapeamento das funções do algoritmo HOG nas diferentes configurações de *pipeline*.



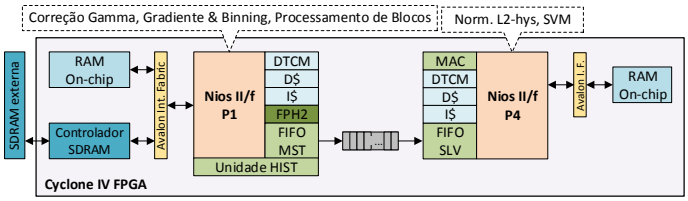
Fonte: Elaborada pelo autor.

A Figura 42 apresenta os arranjos em *pipeline* nas diversas organizações utilizadas, as quais foram projetadas para explorar e avaliar o impacto no desempenho do paralelismo alcançado por *pipelining* em nível de tarefas. A relação dos números com os estágio Esta distribuição precisa ser baseada no tempo de execução de cada tarefa e na conformidade com a transformação das funções para um modelo de execução do tipo *stream-in/stream-out*.

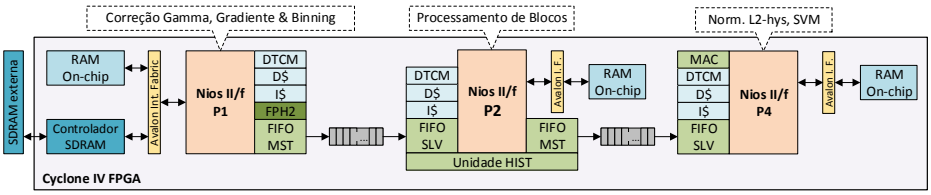
Além do paralelismo, explora-se também a flexibilidade dos processadores *softcore*, adicionando-se unidades de aceleração. Os potenciais candidatos para aceleração por hardware foram identificados pelo *profiling* apresentado na Tabela 3. O foco inicial está sobre as funções de processamento de bloco e classificação SVM, que são os que mais consomem tempo na cadeia de detecção do HOG. É importante notar que essas funções encontram-se nos últimos estágios do *pipeline*, o que leva a *stalls* e prejudica o desempenho. Para reduzir o impacto dessas funções, foram desenvolvidas unidades de aceleração que podem ser instanciadas junto ao *datapath* do processador *softcore* utilizando a interface de instruções customizadas do Nios II (ALTERA, 2005). Tais unidades são adjacentes à Unidade Lógica e Aritmética (ULA), evitando assim, acessos ao barramento do sistema e permitindo que o acesso à unidade pelo software seja feito por meio de macros. Para manter a flexibilidade proposta neste arquitetura, as unidades de aceleração foram desenvolvidas de forma que elas possam trazer benefícios para diferentes variações do algoritmo, ou até mesmo outras aplicações.

A primeira unidade desenvolvida funciona como um coprocessador de operações multiplica-acumula em ponto flutuante. A operação implementada pela denominada Unidade MAC, é largamente usada pela função de classificação SVM, que consiste em um produto escalar entre um vetor de classificação e descritor de uma janela de detecção. O resultado dessa operação é um valor que representa a confiança no veredito da classificação (*score*). Para janelas de detecção de $128x64$ pixels, tanto o descritor quando o vetor de classificação possuem 3.780 elementos,

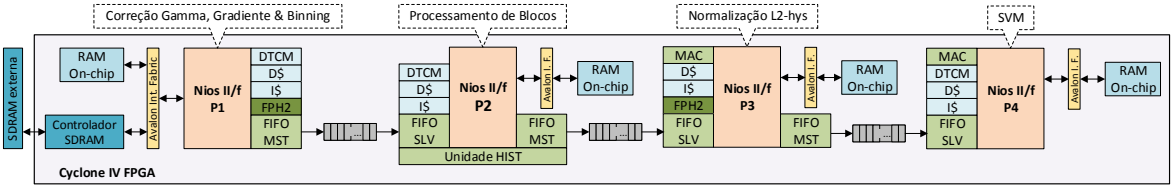
Figura 42 – Arquiteturas para o algoritmo HOG implementadas com diferentes números de estágios (processadores).



(a) Arquitetura com dois estágios.



(b) Arquitetura com três estágios.



(c) Arquitetura com quatro estágios.

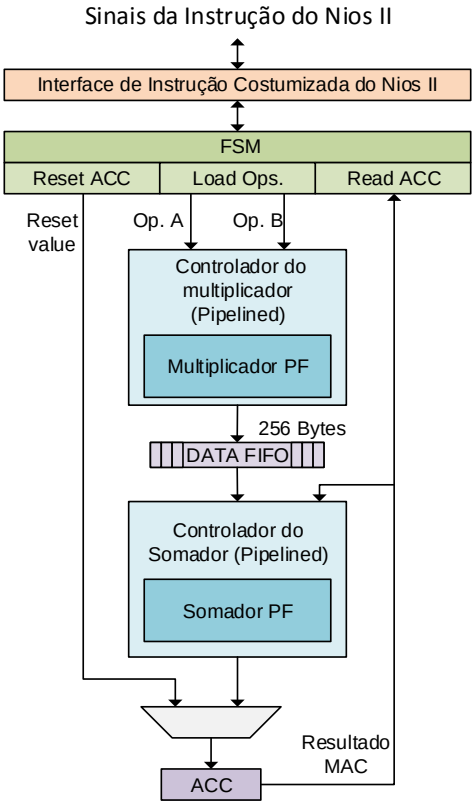
Fonte: Elaborada pelo autor.

que também é o número de operações MAC necessárias por janela. Como 3.285 janelas são processadas por frame, 12.417.300 operações MAC de ponto flutuante são realizadas.

A Figura 43 mostra o diagrama da arquitetura da unidade MAC. Esta unidade implementa três instruções customizadas que atuam sobre um registrador acumulador local (ACC). A primeira instrução (*Reset ACC*) inicializa o acumulador com uma valor fornecido pelo software. A segunda instrução (*Load ops*) carrega na unidade os valores a serem multiplicados e a terceira instrução (*Read ACC*) permite ao software ler o conteúdo do acumulador. Devido à sua implementação em *pipeline*, múltiplas operações MAC podem ser iniciadas sequencialmente. A ativação de cada operação bloqueia o processador por dois ciclos de *clock* e então, este é liberado para processar novos dados. Sem a unidade MAC, uma operação do tipo multiplica-acumula com precisão simples seria realizada em 9 ciclos (no melhor caso) utilizando o hardware de ponto flutuante da Altera. Uma instrução de leitura bloqueia o processador até que todas as operações enfileiradas no *pipeline* da Unidade MAC tenham sido completadas.

A unidade MAC também pode ser útil para outras funções na cadeia de detecção do HOG. Por exemplo, a normalizaçao L2-hys aplica uma norma L2 (Equação 3.3) em um histograma de

Figura 43 – Arquitetura da Unidade MAC.



Fonte: Elaborada pelo autor.

blocos seguida de um corte (*clipping*) e outra aplicação da norma L2. Para um *frame* de tamanho VGA, isto significa calcular 9.322 vezes a norma do vetor $||\vec{v}||$ e sua raiz quadrada. O cálculo da norma consiste em múltiplas operações de multiplica-acumula em ponto flutuante, uma para cada posição de um histograma de blocos.

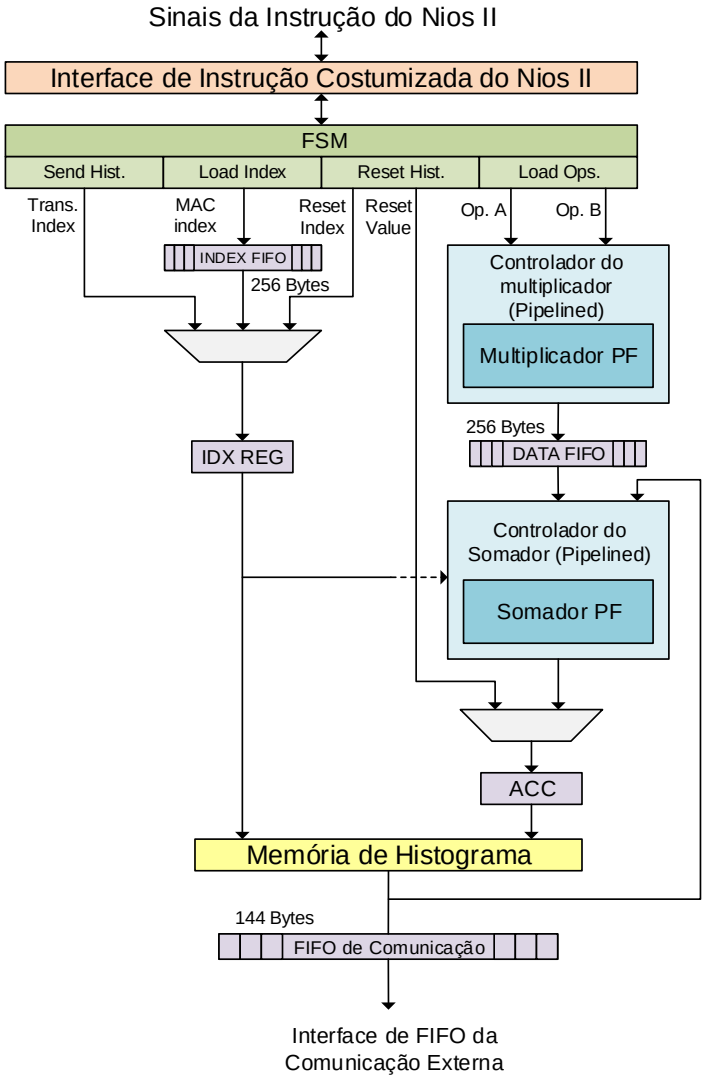
Além da unidade MAC, foi também desenvolvida a unidade de histogramas (HIST) que funciona como um co-processador para operações intensivas em um histograma. Esta unidade foi desenvolvida para suportar eficientemente a aplicação de pesos em cada elemento de um histograma, que é o núcleo das funções que consomem mais tempo no processamento de blocos. As funções de filtro Gaussiano e interpolação bilinear aplicam pesos lidos a partir de tabelas de consulta em seus dados de entrada, que são os votos ponderados reunidos em blocos. Esses votos são acumulados em posições de histogramas indicados pelos *bins* correspondentes, por meio de operações do tipo multiplica-acumula. Para um *frame* com resolução VGA contendo 4.661 blocos, um total de 4.772.864 operações MAC devem ser realizadas.

A Figura 44 ilustra a arquitetura de uma unidade de histograma. Esta unidade implementa cinco instruções customizadas que operam uma memória de histograma interna. A primeira instrução (*Reset Hist*) reinicia o histograma, inicializando-o com um valor especificado pelo software como argumento da instrução. Neste caso, o processador é bloqueado até que o processo de reinicialização tenha sido completado e todos os elementos do histograma tenham sido

devidamente inicializados. As operações MAC são aplicadas ao histograma por meio de duas instruções. Primeiro, o software deve prover a posição do histograma (índice) que será modificado utilizando a instrução *Load index*. Em seguida, deve fornecer os operandos da multiplicação utilizando a instrução *Load ops*. Ambas as instruções são não bloqueantes e demoram dois ciclos de *clock* para completarem.

Os valores armazenados na memória de histograma podem ser lidos por qualquer outro processador presente no *pipeline* que esteja conectado à unidade de histograma. Dessa forma, não é necessário que o processador que contém a unidade de histograma realize a leitura de todos os resultados antes de transmiti-lo a outro processador no arranjo. Para este propósito, foi implementada a instrução *Send Hist*, que envia o conteúdo do histograma para uma FIFO de comunicação. Os valores dessa FIFO podem ser lidos utilizando-se o módulo FIFO com a interface FIFO SLV, descrita anteriormente, a qual também pode ser usada para comunicação

Figura 44 – Arquitetura de uma Unidade de Histograma (Unidade HIST).



Fonte: Elaborada pelo autor.

entre processadores.

4.3 Fase 2 - Implementação do algoritmo ICF em FPGA

A implementação da arquitetura para processamento do algoritmo ICF também é iniciada com a análise do algoritmo. Nela, abordam-se outros aspectos relacionados à exploração do espaço de projeto de hardware e software, como, por exemplo, a memória ocupada durante o tempo de execução. Analisa-se também o potencial de paralelismo do algoritmo e explora-se a implementação em hardware dos diversos canais do algoritmo ICF.

4.3.1 Análise do Algoritmo

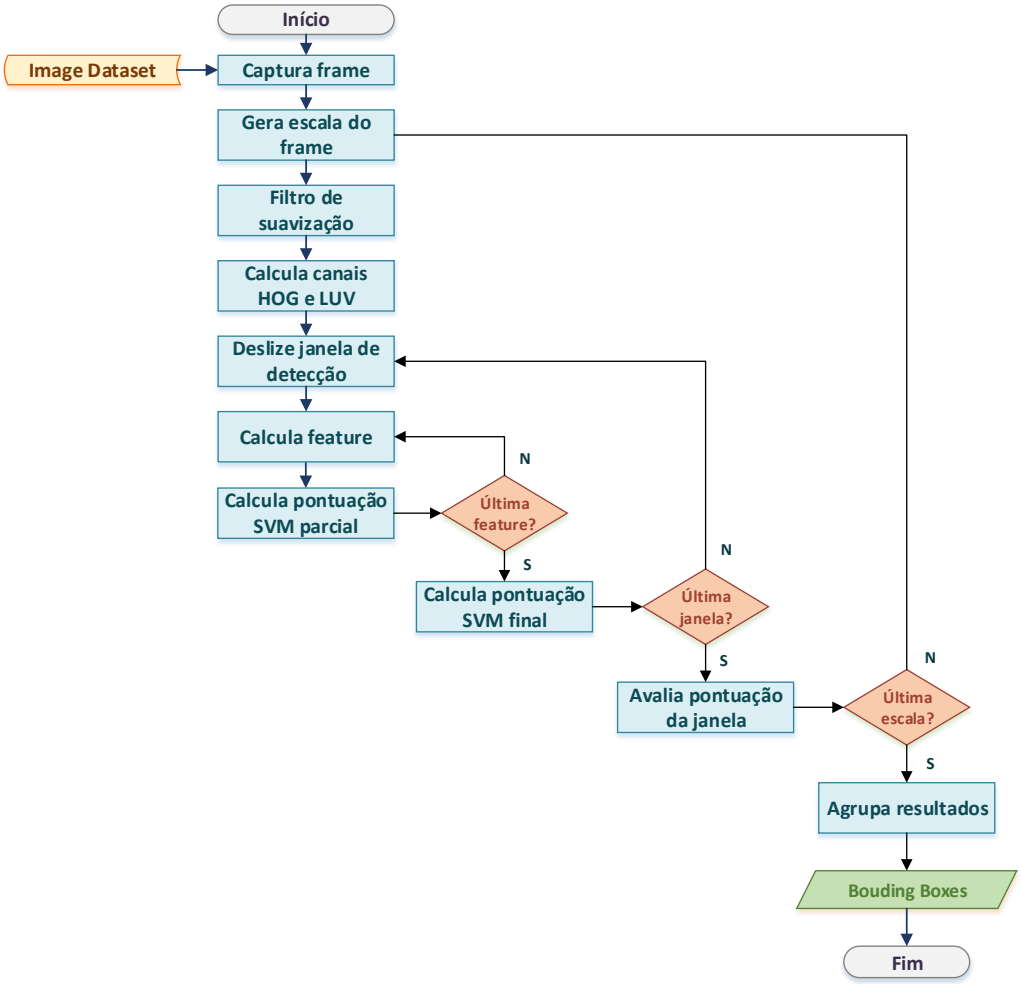
Para realizar a análise do algoritmo, foi desenvolvida uma versão de referência para o ICF, baseada no algoritmo apresentado por Dollar *et al.* (2009a) e na implementação de Benenson *et al.* (2014). Diferentemente do algoritmo original, na versão de referência implementada utiliza-se para classificação o método SVM Linear ao invés do Adaboost, embora o primeiro demande um poder computacional maior, pois é necessário que todas as *features* sejam avaliadas. Contudo, o foco nesta implementação encontra-se nas etapas anteriores à classificação, utilizando o SVM como um exemplo possível de classificador. O diagrama da Figura 45 mostra a sequência de etapas do algoritmo ICF.

Os parâmetros utilizados para o algoritmo ICF seguem, em sua maior parte, o trabalho original de Dollar *et al.* (2009a). Como entrada, são fornecidas imagens com resolução 640×480, coloridas (RGB), obtidas pelo redimensionamento das imagens da base de dados de pedestres do INRIA. Um filtro Gaussiano de suavização com $r = 1$ é aplicado à imagem de entrada. São utilizados dez tipos de canais, sendo três para informação sobre cor no formato LUV, um para magnitude de gradiente e seis para histogramas de gradiente (HOG). Diferentemente do HOG original, aqui utilizam-se somente histogramas com 6 *bins*. As *features* candidatas são geradas randomicamente, tanto o índice do canal quanto a área do seu retângulo. Os retângulos têm largura e altura entre 8 e 16 pixels e ao todo são geradas 5000 *features* uniformemente entre os canais. Utiliza-se um deslocamento da janela de 4 pixels em ambas as direções e um fator de escala de $1,2$ para redimensionamento da imagem de entrada.

4.3.1.1 Perfil de Execução do Algoritmo

A Tabela 5 mostra o perfil de execução com as porcentagens de tempo de cada função do ICF em relação ao tempo de execução total. Para traçar o perfil, o software foi executado em um computador *desktop* com processador AMD FX-6300, contendo seis núcleos de processamento, com 2 MB de cache cada, frequência a 3.5 GHz e 8 GB de memória RAM. O *profiling* foi realizado com a ferramenta GPROF, assim como na análise do algoritmo HOG. A função mais custosa é a que calcula a pontuação da classificação SVM, consumindo $68,3\%$ do tempo total.

Figura 45 – Diagrama da versão de referência do algoritmo ICF.



Fonte: Elaborada pelo autor.

As operações que constituem essa função são do tipo multiplica-acumula e são executadas 5000 vezes por janela, uma para cada *feature*. Esta é uma desvantagem do uso de classificadores SVM, pois necessitam processar todas as *features* para chegar à pontuação final. Técnicas de classificação em cascata, por exemplo, podem utilizar uma sequência de classificadores fracos e, à medida em que os classificadores iniciais rejeitam o objeto, o processo de classificação é abortado. Contudo, neste trabalho, avalia-se o desempenho de um classificador SVM como padrão para os algoritmos.

A segunda função que demanda mais tempo é a responsável pelo cálculo dos histogramas de gradiente. O processo de construção desses histogramas é uma variação do processo original do algoritmo HOG, pois utiliza-se um número menor de *bins* e normalização L1 para os pixels vizinhos somente. Porém, são mantidas operações custosas, como *arcotangente* e raiz quadrada para cálculos de ângulo e magnitude, respectivamente. Na terceira posição da Tabela 5 está a função que calcula *features* integrais. Esta função realiza os cálculos em ponto flutuante descritos

Tabela 5 – Resultados do *profiling* para a versão de referência do algoritmo ICF executada no processador AMD FX 6300.

Funções	**% do tempo total**
Score SVM	68,3%
Histogramas de gradiente	8,5%
Features integrais	7,3%
Imagens integrais	7,2%
Redimensionamento da Imagem	3,6%
Suavização da Imagem	1,2%
LUV	1,2%
Outras	2,7%
Tempo por *frame*	$1,7s$

Fonte: Elaborada pelo autor.

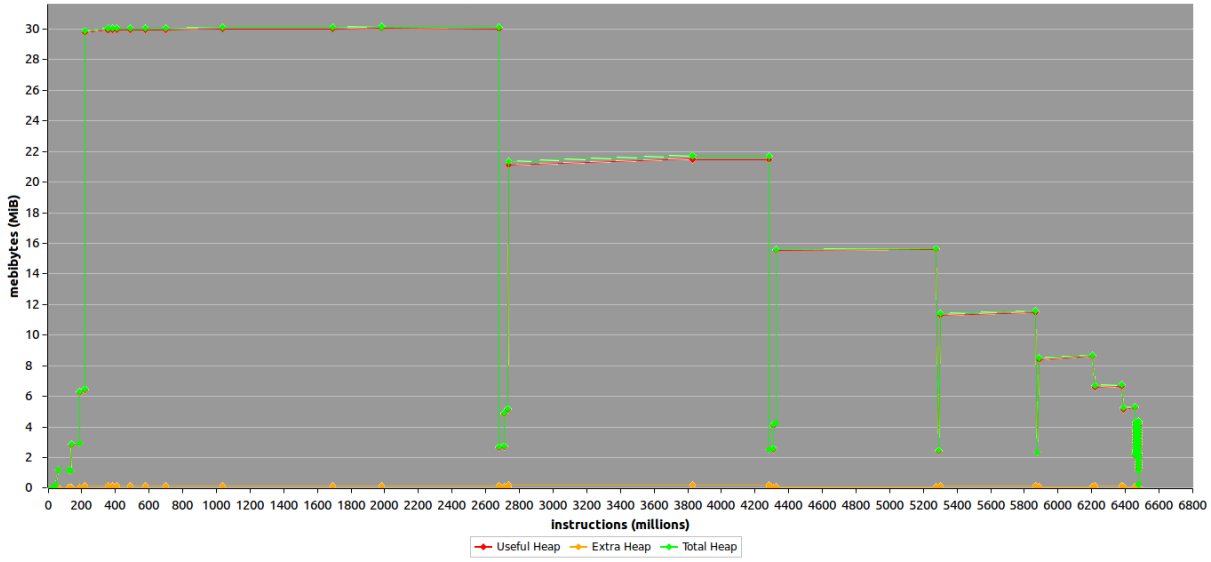
na Equação 3.7, a qual é executada também uma vez para cada *feature*. A função de imagens integrais constrói uma imagem integral para cada canal do algoritmo, conforme descrito na Equação 3.6. Ela é executada para cada elemento provindo de um canal, ou seja, $10 \times 320 \times 240$ vezes para cada imagem de entrada, segundo os parâmetros utilizados.

As três últimas funções são as responsáveis pelo redimensionamento, pela suavização da imagem e pela transformação da imagem de RGB para LUV. Para um fator de escala de $1,2$, o algoritmo redimensiona a imagem de entrada sete vezes, realizando a interpolação de pixels vizinhos para a geração da nova escala de imagem. O filtro de suavização realiza um produto escalar entre um kernel de tamanho 3×3 e a imagem de forma a eliminar ruídos. A função LUV transforma dados em RGB da imagem para o espaço de cor LUV, passando pelo espaço intermediário XYZ. Esta transformação consiste em uma série de operações aritméticas sobre os pixels de entrada, incluindo o cálculo de raiz cúbica.

Além do esforço computacional demandado pelas funções citadas, o algoritmo ICF tem como característica a geração de uma grande quantidade de dados intermediários para processamento. Por exemplo, para cada pixel de uma imagem de entrada são gerados 10 dados derivados em ponto flutuante durante a construção das imagens integrais. Com a finalidade de determinar a ocupação máxima em memória dos dados processados, foi utilizada a ferramenta *Massif* (VALGRIND, 2016), que monitora a variação do tamanho da memória dinâmica (*heap*) durante a execução de um programa. O gráfico da Figura 46 mostra a dinâmica da ocupação de memória durante a execução do algoritmo ICF. Nele, podem-se observar sete picos de ocupação de memória durante o tempo, os quais estão relacionados aos processamento de cada uma das sete escalas da imagem de entrada. O primeiro pico (à esquerda) representa o processamento da

escala maior, no caso um *frame* de tamanho 640×480, que ocupa 30 MB de memória *heap*.

Figura 46 – Relatório da ferramenta Massif, mostrando a utilização da memória *heap* durante a execução da versão de referência do algoritmo ICF.



Fonte: Elaborada pelo autor.

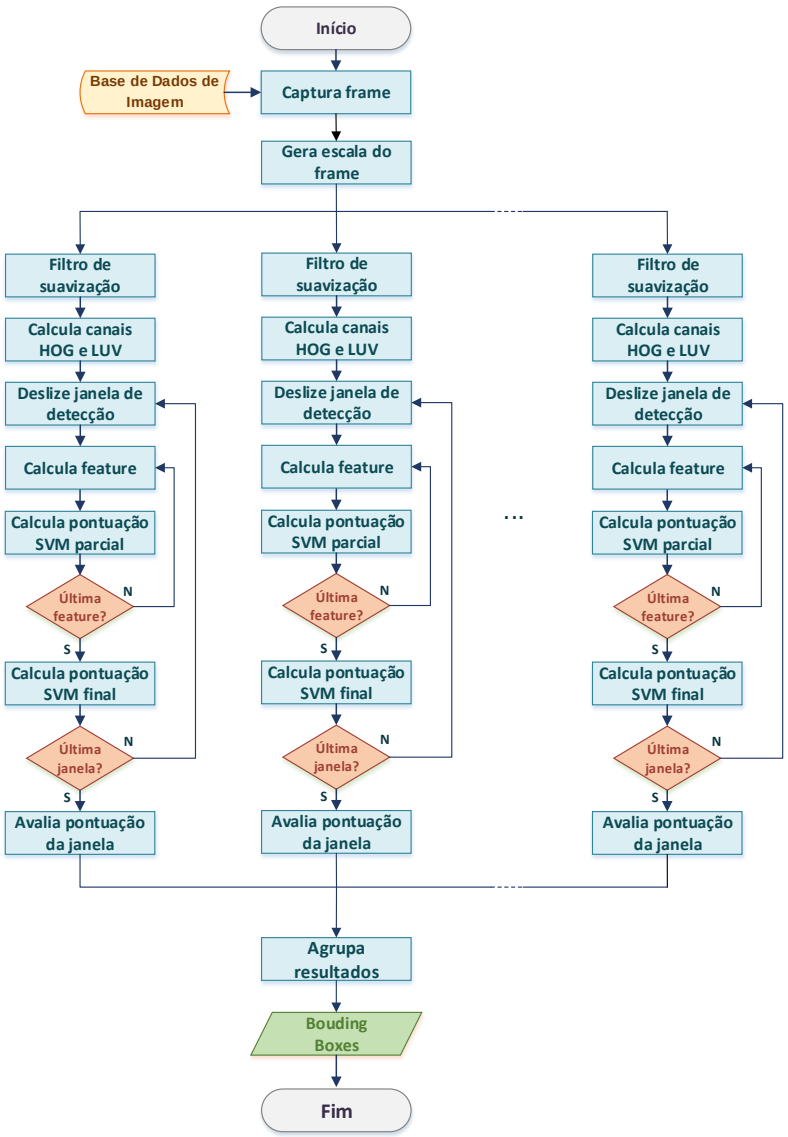
Para efeito de comparação, o algoritmo HOG, ao processar um *frame* de mesmo tamanho, apresenta uma ocupação máxima de 6,5 MB de memória. Se a resolução da imagem de entrada for reduzida para 320×240 pixels, 8 MB de memória são ocupados no máximo. Ao extrapolar o algoritmo para o processamento de 4 imagens simultâneas, a ocupação pode chegar a 24 MB. Estes números são de grande importância para o dimensionamento do hardware implementado em FPGA, visto que a quantidade de memória *on-chip* é bastante limitada e o acesso à memória externa ao FPGA pode representar atrasos no tempo de execução. Por exemplo, o FPGA Stratix V utilizado neste trabalho possui cerca de 50.000 kbits de memória *on-chip*, o que equivale a 6,2 MB de memória disponível, considerando que seja possível alocar 100% destes recursos durante a síntese. Nos experimentos realizados neste trabalho, foi possível alocar no máximo 75% dos blocos de memória disponíveis nesse FPGA.

Outra característica do algoritmo ICF está relacionada ao grande número de acessos realizados à memória. Por meio da ferramenta *Cachegrind* (VALGRIND, 2016) foi possível detectar um maior número de faltas na memória *cache* durante o acesso às imagens integrais para cálculo das *features* de primeira ordem. A sequência de acesso é determinada por uma tabela de consulta com as coordenadas dos retângulos e o número do canal de cada *feature*. Como estas informações têm origem aleatória, os acessos não são realizados em um padrão que favoreça os mecanismos de localidade dos dados da *cache*. No modelo de cache criado pela ferramenta *Cachegrind*, há falta em 28,7% dos acessos de leitura à *cache* no primeiro nível. Apesar deste não se apresentar como um fator crítico para o desempenho, otimizações podem ser obtidas também pela reorganização do acesso aos dados.

4.3.1.2 Exploração de Paralelismo

Semelhantemente à análise de paralelismo feita para o algoritmo HOG, foram encontradas três formas principais de paralelismo de dados no algoritmo ICF, cada uma com uma granularidade diferente. A primeira forma consiste no processamento de diversas escalas do algoritmo em paralelo e tem como característica a granularidade de dados grossa. É importante observar que, neste caso, cada *thread* é responsável por processar uma quantidade de dados não uniforme, visto que o tamanho de cada escala é decrescente. Dessa forma, é interessante que o escalonamento das *threads* seja feito dinamicamente, a fim de aproveitar todo o potencial deste tipo de paralelismo. O diagrama da Figura 47 mostra quais funções do algoritmo ICF são executadas simultaneamente no paralelismo em nível de escala.

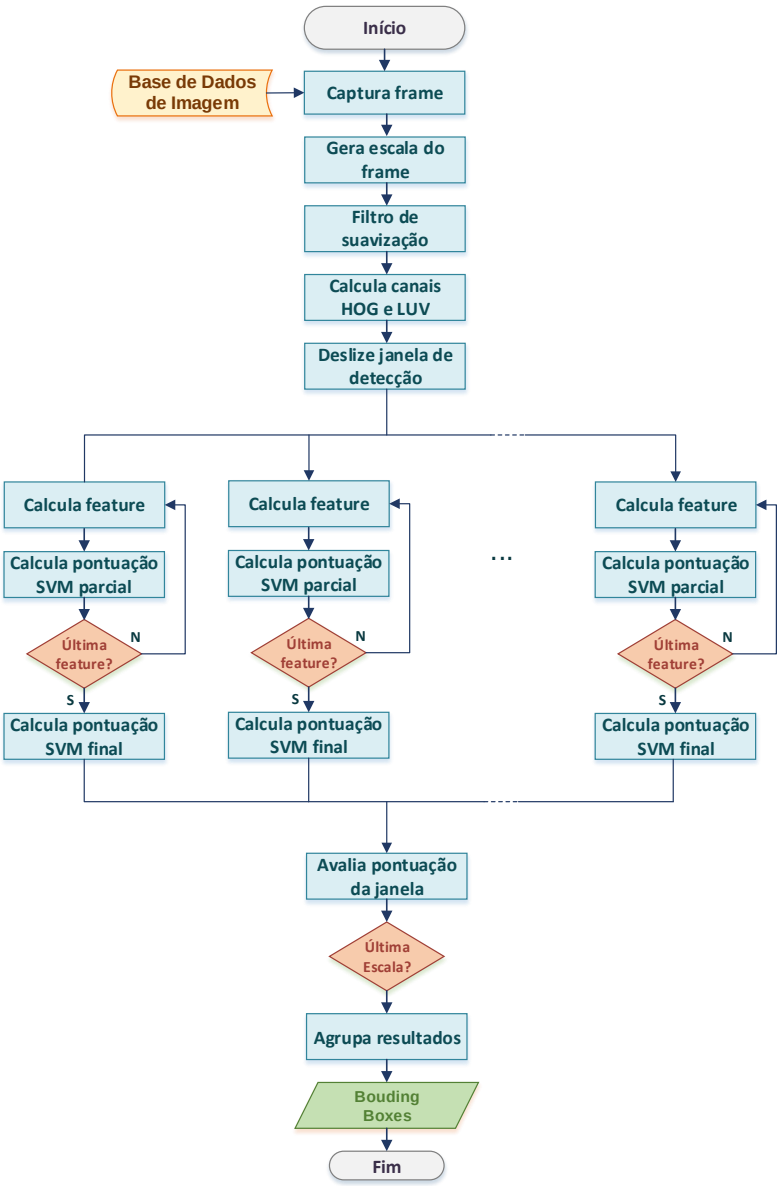
Figura 47 – Diagramas do algoritmo ICF para o paralelismo em nível de escalas.



Fonte: Elaborada pelo autor.

O segundo tipo de paralelismo identificado é em nível de janelas. Nele, cada janela de detecção pode ser processada em paralelo, compartilhando com outras *threads* os dados das outras janelas sobrepostas. Este tipo de paralelismo representa um compromisso entre as granularidades de dados grossa e fina. A Figura 48 mostra o diagrama do algoritmo ICF implementado para execução simultânea das janelas de detecção.

Figura 48 – Diagramas do algoritmo ICF para o paralelismo em nível de janelas.

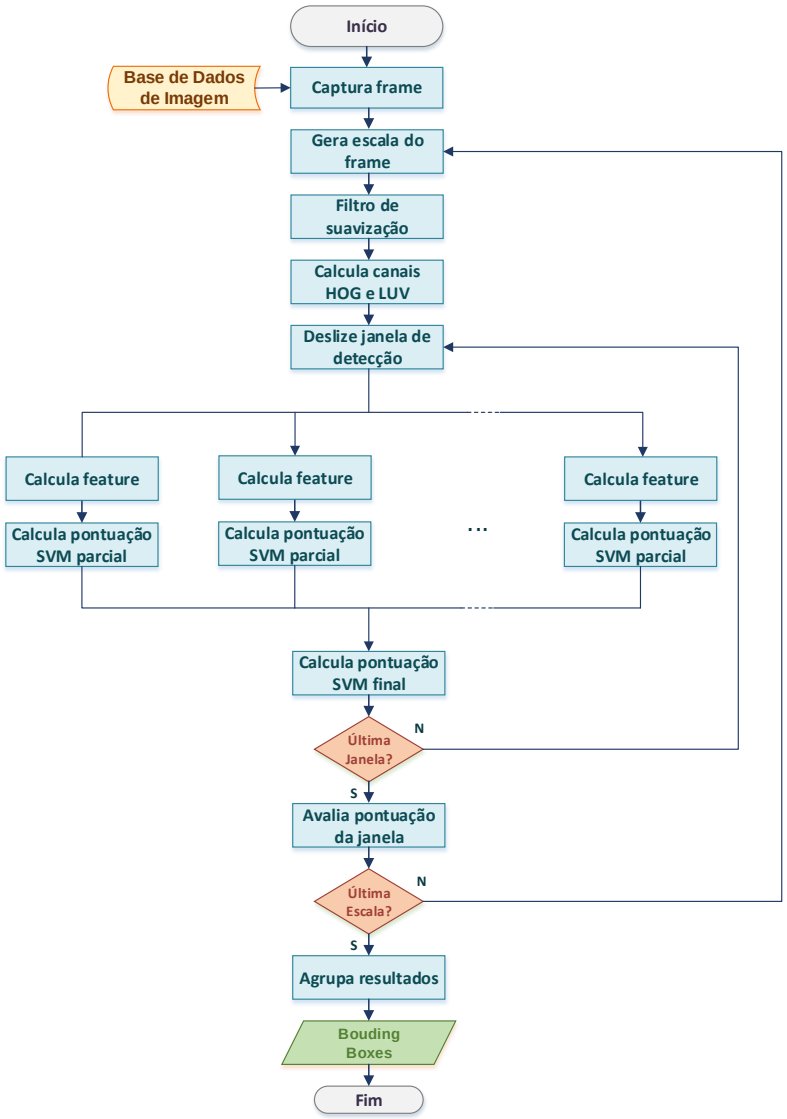


Fonte: Elaborada pelo autor.

O terceiro tipo consiste em executar o cálculo da área do retângulo e a pontuação SVM para uma *feature* em paralelo ao mesmos cálculos para outras *features*, conforme ilustrado no diagrama da Figura 49. Este tipo de paralelismo de granularidade mais fina, ao mesmo tempo que permite a criação de um grande número de *threads*, acaba por atuar em uma pouca quantia

de dados compartilhados, o que acaba degradando seu desempenho. O gráfico da Figura 50 mostra o desempenho por *frame* da execução desses tipos de paralelismo. Estes experimentos foram realizados em um computador *desktop* com processador AMD FX-6300, contendo seis núcleos de processamento. A criação e o gerenciamento das *threads* é feito por meio da biblioteca OpenMP.

Figura 49 – Diagramas do algoritmo ICF para o paralelismo em nível de cálculo de *features*.

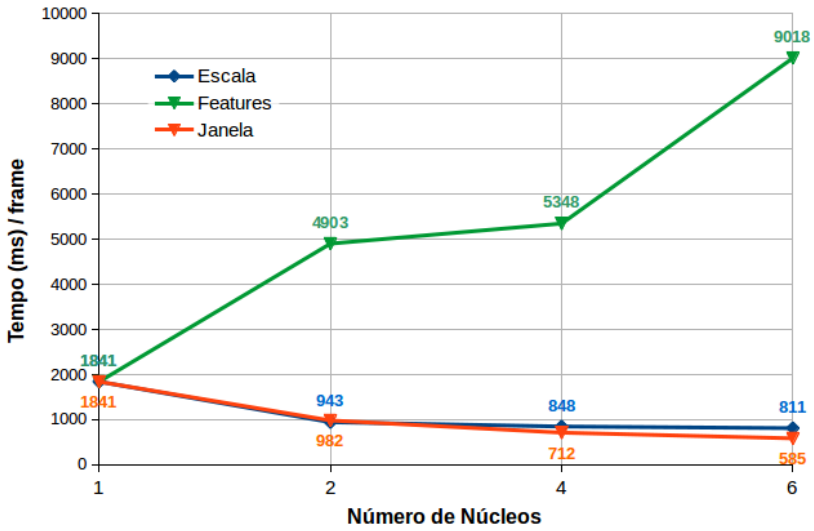


Fonte: Elaborada pelo autor.

Internamente às funções também é possível identificar oportunidades de paralelismo. No cálculo dos histogramas de gradientes, por exemplo, é possível realizar o cálculo do ângulo e da magnitude simultaneamente. Como essa operação é realizada muitas vezes é possível obter uma aceleração local no processamento da função. Como a granularidade dos dados desse paralelismo é bastante fina, a distribuição desse processamento em diversas *threads* pode levar à degradação do desempenho. Contudo, neste projeto, explora-se essa oportunidade na implementação em

hardware, conforme será demonstrado na subseção 4.3.2. Outra oportunidade identificada e também explorada em hardware é o cálculo em paralelo para a geração dos canais LUV, HOG e magnitude.

Figura 50 – Comparação entre os diferentes níveis de paralelismo para o algoritmo ICF.



Fonte: Elaborada pelo autor.

De forma semelhante ao HOG, a cadeia de detecção do ICF também é formada por uma série de estágios consecutivos que podem ser distribuídos entre diversos núcleos de processamento para exploração de paralelismo temporal. Utilizando uma estrutura de comunicação entre processadores como a utilizada na arquitetura para o HOG, constituída por FIFOs, é possível construir uma arquitetura em *stream*, para que os dados fluam pelos estágios da cadeia de detecção da imagem, extraindo o máximo de processamento de cada estágio.

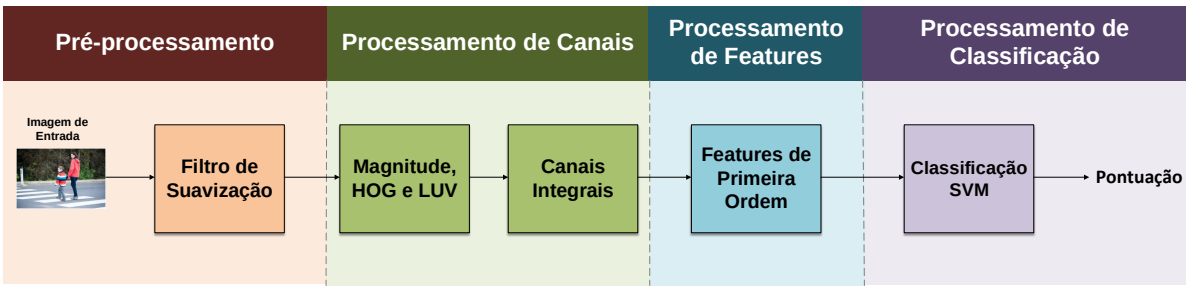
4.3.2 Implementação em FPGA

A análise da versão de referência do algoritmo ICF permitiu identificar os fatores mais importantes para a implementação de uma arquitetura em hardware. A descrição dessa implementação segue o fluxo da cadeia de detecção do algoritmo, apresentada na Figura 51. Cada elemento do diagrama representa um bloco de hardware desenvolvido ou um núcleo de processamento utilizado.

A cadeia de detecção implementada atende ao processamento de uma única escala da imagem de entrada (maior escala). Isto implica que o sistema é capaz de detectar apenas pedestres do mesmo tamanho utilizado no processo de treinamento. Embora essa característica sugira uma limitação da arquitetura desenvolvida, uma solução multi escala pode ser adotada como em Xylon (2014), no qual se insere o núcleo de detecção em um *framework* que fornece diferentes escalas do mesmo *frame*. Ainda, devido à escalabilidade da solução de hardware, é possível

também utilizar múltiplas instâncias da cadeia de detecção em paralelo e fornecer a cada uma um conjunto de escalas da imagem de entrada. Dessa forma, torna-se possível a detecção de pedestres se movendo em distâncias diferentes em relação à câmera. Ressalta-se, contudo, que as implementações e resultados que serão apresentados para a arquitetura desenvolvida referem-se à detecção em única escala.

Figura 51 – Fluxo de detecção do algoritmo ICF. Cada etapa do fluxo é implementada em hardware fixo ou em software em um núcleo de processamento.



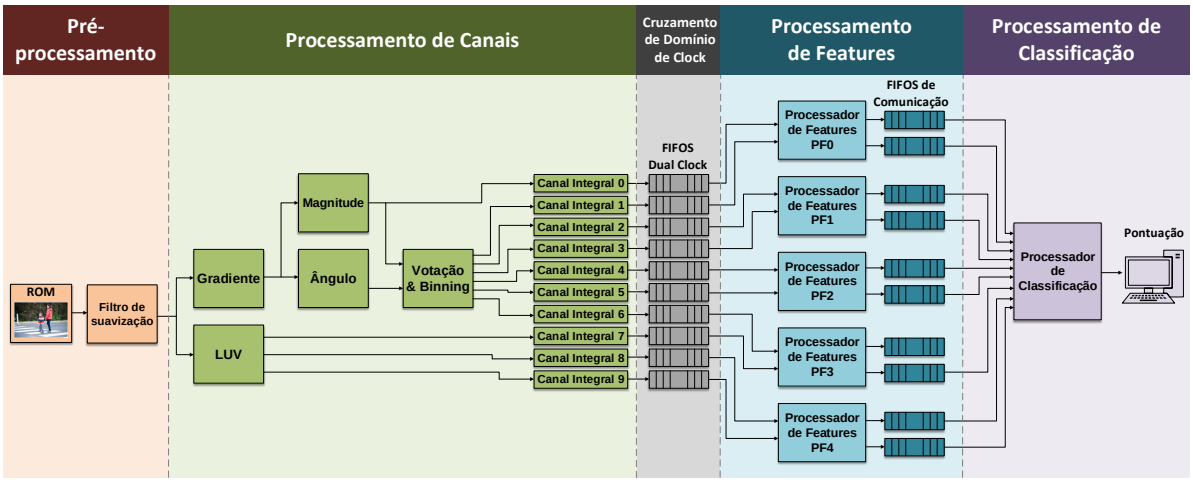
Fonte: Elaborada pelo autor.

4.3.2.1 Visão Geral da Arquitetura

Para a implementação da arquitetura em FPGA, a resolução da imagem de entrada é de 320×240 pixels. O objetivo do uso desta resolução é possibilitar os experimentos de escalabilidade da arquitetura de forma a processar imagens provindas de quatro câmeras. Embora esta resolução tenha sido adotada, é possível processar imagens com resoluções de outros tamanhos, bastando configurar os parâmetros do hardware e adaptar o software executado nos diversos núcleos de processamento. Os dados de imagem lidos pelo sistema em hardware são acessados a partir de uma memória ROM, que armazena os pixels do *frame* de entrada no formato RGB em 24 bits (8 bits por cor). Os pixels são armazenados em ordem de coluna, que também é a ordem adotada pela implementação do algoritmo nos outros estágios. O objetivo da utilização dessa memória é emular a presença de uma câmera de vídeo fornecendo *frames* ao sistema, sem que seja necessária a lógica de integração e a engenharia da conexão de uma câmera com a placa de desenvolvimento.

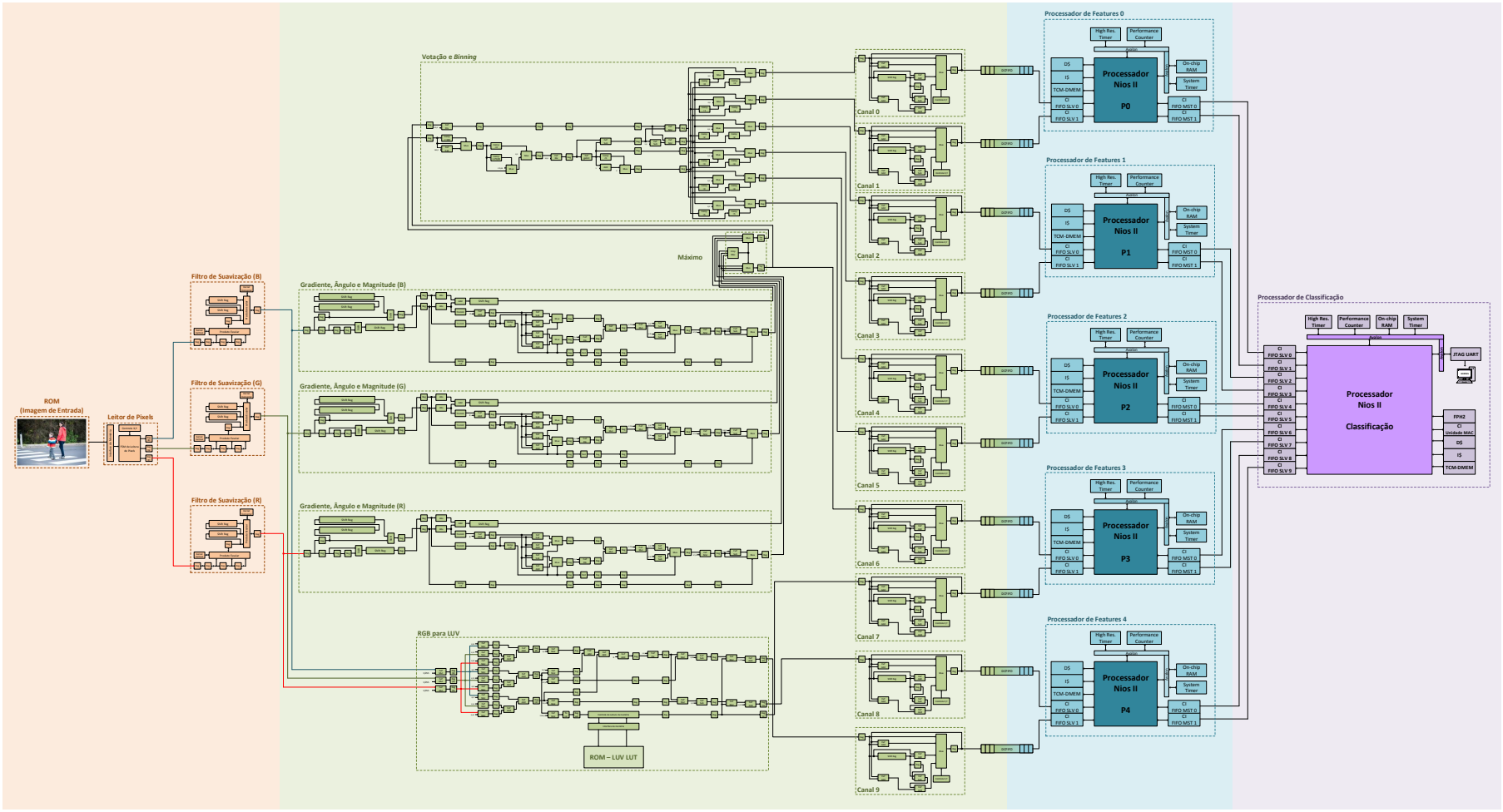
A Figura 53 mostra com detalhes a arquitetura desenvolvida para detecção de pedestres utilizando o algoritmo ICF. Se este trabalho estiver sendo lido em formato digital, recomenda-se utilizar a ferramenta *zoom* do leitor de documentos para contemplar os detalhes de implementação da arquitetura.

Figura 52 – Visão geral dos módulos que constituem a arquitetura para execução do algoritmo ICF.



Fonte: Elaborada pelo autor.

Figura 53 – Arquitetura detalhada para processamento do algoritmo ICF.

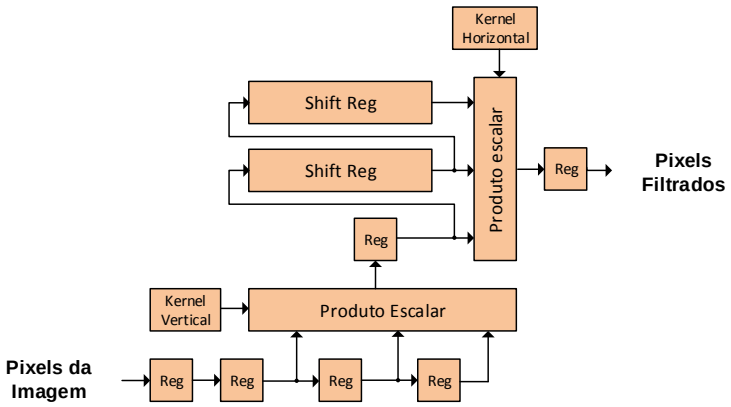


Fonte: Elaborada pelo autor.

4.3.2.2 Pré-processamento

Os pixels da imagem de entrada são lidos por um módulo que implementa o controle dos sinais de interface com a memória ROM por meio de uma máquina de estados. Após a leitura de um pixel, este é destinado ao módulo responsável por aplicar o filtro de suavização para eliminação de ruídos. O diagrama de blocos desse filtro está ilustrado na Figura 54. O filtro Gaussiano utilizado na imagem de entrada é separável e, portanto, a aplicação do filtro é feita em duas operações de produto escalar subsequentes, utilizando os *kernels* $[1\,2\,1]^T$ e $[1\,2\,1]$.

Figura 54 – Diagrama da arquitetura do filtro de suavização.



Fonte: Elaborada pelo autor.

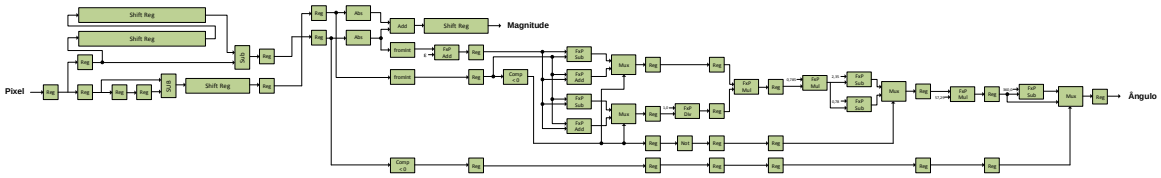
Após a aplicação do primeiro *kernel* os pixels processados são colocados em *shift registers*, que funcionam como *buffers* que armazenam duas colunas da imagem. Quando os *shift registers* estão cheios, isto significa que o *kernel* horizontal do filtro pode ser aplicado, gerando como saída os pixels suavizados. Cada componente de cor (R, G e B) é filtrada por uma instância diferente do filtro paralelamente e todas as operações envolvidas na filtragem envolvem números inteiros.

4.3.2.3 Processamento de Canais

Após o pré-processamento, o algoritmo ICF gera dez canais integrais para o posterior cálculo das *features*. Os seis primeiros canais correspondem aos seis *bins* do histograma de gradientes. O cálculo dos histogramas inicia-se com a extração dos gradientes dos pixels provenientes da etapa de pré-processamento. Para isto, o hardware desenvolvido aplica, simultaneamente, os *kernels* $[-1\,0\,1]^T$ e $[-1\,0\,1]$ nas colunas e nas linhas da imagem. Como os pixels são recebidos em ordem de coluna, o *kernel* vertical é aplicado diretamente aos dados que chegam ao módulo. Para a aplicação do *kernel* horizontal, é necessário realizar o armazenamento temporário de duas colunas, possibilitando assim, o acesso posterior aos valores dos pixels em cada linha. A fim de sincronizar a saída dos resultados, os produtos da aplicação do primeiro *kernel* são armazenados

em um *buffer* até que a aplicação do segundo *kernel* tenha sido completada para o mesmo pixel. Os produtos deste processo são os valores dos gradientes nos eixos x e y.

Figura 55 – Arquitetura do módulo para cálculo dos gradientes, do ângulo e da magnitude.



Fonte: Elaborada pelo autor.

O próximo módulo é responsável por calcular o ângulo e a magnitude baseado nas informações de gradiente. Nesse módulo as operações são realizadas utilizando representação numérica em ponto fixo com as partes inteira e fracionária contendo 16 bits cada uma. O cálculo tradicional do valor da magnitude envolve o uso de raiz quadrada (norma euclideana), contudo, nesta implementação, adota-se a aproximação mostrada na Equação 4.1. Para obtenção do valor do ângulo, utiliza-se a aproximação da função *arcotangente* apresentada por Shima (1999). Os valores de ângulo e de magnitude são computados em paralelo para os três componentes de cor de cada pixel, sendo escolhidos para uso nas fases posteriores o conjunto com maior magnitude. A Figura 55 mostra a arquitetura do módulo desenvolvido para cálculo dos gradientes, do ângulo e da magnitude.

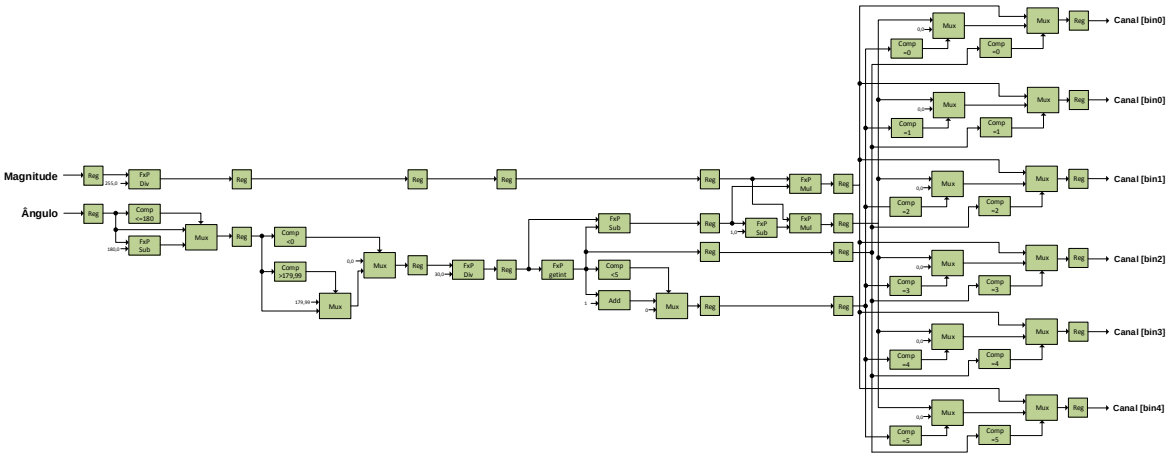
$$|G| = |G_x| + |G_y| \tag{4.1}$$

O resultado do cálculo da magnitude é utilizado para a geração do sétimo canal. Além disso, a magnitude e o valor do ângulo alimentam o módulo que realiza a votação ponderada e o *binning*. A votação ponderada segue a Equação 3.2 e os valores dos votos são distribuídos pelos canais correspondentes aos *bins*, os quais são definidos conforme o valor do ângulo. Vale ressaltar que o valor da magnitude é sempre distribuído entre um *bin* e o próximo *bin* vizinho. O diagrama do hardware utilizado para votação é apresentado na Figura 56.

Os três últimos canais são gerados pela transformação dos componentes R, G e B para o espaço de cores LUV, a qual ocorre pela aplicação das Equações 3.8, 3.9, 3.10, 3.11. No cálculo do componente L a partir do componente Y do espaço XYZ, é utilizada uma operação de raiz cúbica. Na implementação em hardware, essa operação é substituída por uma tabela de consulta, que é armazenada em uma memória ROM dedicada ao módulo lUV. A tabela possui 1064 palavras de 32 bits, totalizando $4,1KB$ de dados. A arquitetura deste canal é apresentada na Figura 57.

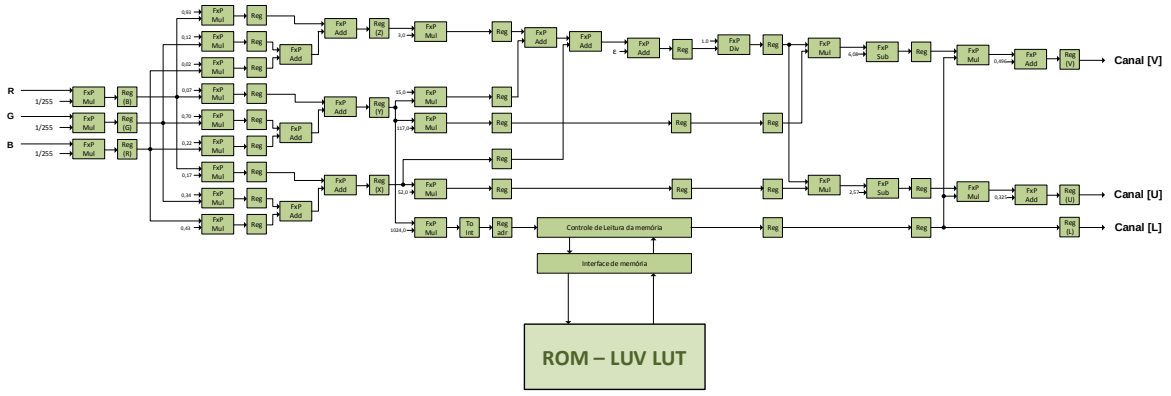
A última etapa do processo de geração de canais, é o cálculo da imagem integral, conforme descrito na subseção 3.2.2. O módulo ilustrado pela Figura 58 é replicado para cada

Figura 56 – Arquitetura do módulo de votação e *binning*.



Fonte: Elaborada pelo autor.

Figura 57 – Arquitetura do módulo do canal LUV.



Fonte: Elaborada pelo autor.

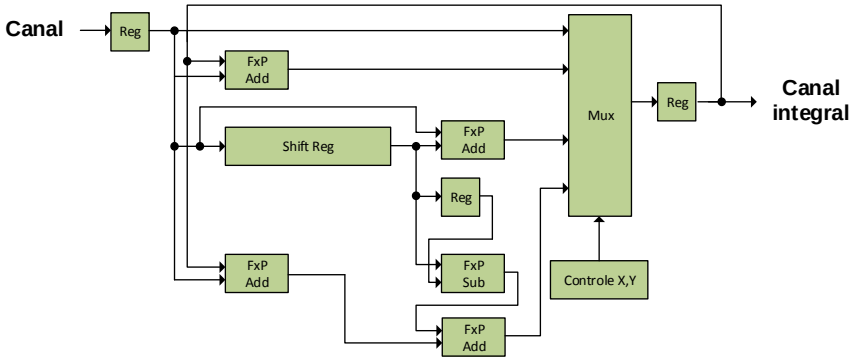
canal gerado e seu resultado é enviado à FIFO para *bufferização* e passagem para o outro domínio de *clock*.

4.3.2.4 Cruzamento de Domínio de Clock

O hardware desenvolvido para o processamento de canais integrais opera a uma frequência de *clock* menor que o sistema de processamento de *features*. Isto porque a frequência máxima deste hardware permitida pela ferramenta de síntese é de 14 MHz. Embora a baixa frequência de operação signifique uma menor taxa de processamento de *frames*, a eficiência da implementação dos módulos em *streaming*, possibilita o processamento de aproximadamente 180 *frames* por segundo. Um efeito colateral do uso de baixas frequências de operação é o baixo consumo de energia, o que é benéfico para a implementação de um sistema embarcado.

Na implementação do sistema, foram criados dois domínios de *clock*, um que contém o

Figura 58 – Arquitetura do módulo para cálculo dos canais integrais.



Fonte: Elaborada pelo autor.

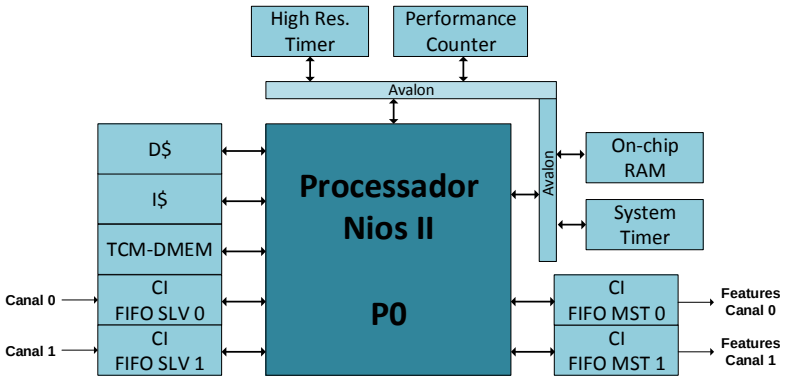
hardware de processamento de canais e um outro para o processamento de *features* e classificação. O primeiro domínio de *clock* opera a uma frequência de 10 Mhz e o segundo domínio opera a 100 Mhz. Para evitar problemas de metaestabilidade ao enviar dados de um domínio para o outro, faz-se necessária a utilização de um mecanismo de sincronização, como FIFOs *dual clock*. Na arquitetura para o ICF, utilizam-se dez FIFOS de 2 KB cada uma, interconectando os módulos de processamento de canais integrais às instruções customizadas de comunicações por FIFO nos processadores de *features*. A interface da instrução customizada é a mesma utilizada no projeto da arquitetura para o algoritmo HOG, o que mostra a possibilidade de reuso dos módulos de hardware desenvolvidos.

4.3.2.5 Processadores de Features

Na etapa de processamento de *features* foram utilizadas cinco instâncias do processador Nios II, no modo *fast*. Cada núcleo de processamento é responsável por realizar o cálculo de *features* de primeira ordem provindas de dois canais integrais. Os dados dos canais são recebidos por meio da instrução customizada FIFOSLV, que se conecta às FIFOs de sincronização com o domínio de *clock* de 10 MHz. Os resultados são enviados à etapa de classificação por meio de duas FIFOs (uma para cada *feature*) utilizando a instrução customizada FIFOMST. A arquitetura do processador de *features* está ilustrada na Figura 59

O processamento das *features* é feito aplicando-se a Equação 3.7. Os cálculos são realizados utilizando representação numérica em ponto fixo, evitando assim a inclusão de hardware para ponto flutuante e aproveitando a velocidade da aritmética inteira. Juntamente com o cálculo das *features*, é realizado o controle do deslizamento da janela de detecção. Para minimizar a ocupação de memória com resultados intermediários e promover o reuso dos dados de janelas já processadas, foi utilizado o *buffer* de coluna de forma semelhante ao algoritmo HOG. Contudo, ao invés de armazenar histogramas de blocos, os *buffers* de coluna armazenam os

Figura 59 – Arquitetura do processador de classificação, baseado no *softcore* Nios II.



Fonte: Elaborada pelo autor.

valores das imagens integrais recebidos dos canais. Como cada Nios II processa a informação de dois canais, dois *buffers* de colunas são implementados por processador. O *buffer* é manipulado como uma estrutura circular, portanto não é necessário realizar deslocamentos de dados, mas somente o controle das colunas que serão reutilizadas.

Para o processamento das *features* cada Nios II é equipado com 4 KB de *cache* de instruções, 2 KB de *cache* de dados, duas instruções customizadas de intercomunicação para recebimento dos dados dos canais, duas instruções customizadas para envio dos resultados para o processador de classificação, e uma memória de dados fortemente acoplada para leitura das coordenadas e tamanhos dos retângulos que representam as *features*. Um instrução customizada de comunicação extra foi adicionada para recebimento de um sinal de sincronismo com o processador de classificação.

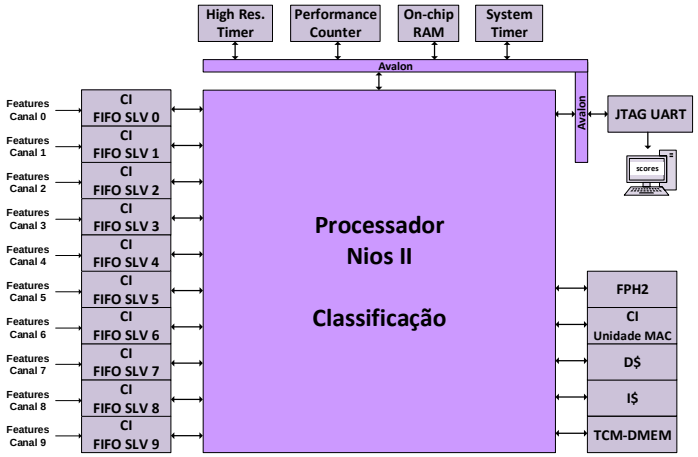
Por meio de comandos *get()*, o software lê os dados provindos dos canais alternadamente, armazenando-os no *buffer* de coluna. Quando são lidos todos os dados de uma janela de detecção, é chamada uma rotina para o cálculo das *features*. À medida em que a janela desliza verticalmente, na maior parte das vezes, é necessário ler somente uma quantidade de dados igual ao deslocamento da janela (4 pixels) para o novo cálculo das *features*. Quando o deslocamento é horizontal, é necessário ler 3 novas colunas inteiras de dados da imagem integral mais o número de pixels para completar a primeira janela de detecção da coluna. O valor das *features* é transmitido ao processador de classificação por meio de um comando *put()* na FIFO correspondente ao canal.

4.3.2.6 Processador de Classificação

A etapa de processamento para classificação por SVM utiliza um único núcleo de processamento, formado por um processador *softcore* Nios II *fast*. Para o recebimento das *features* calculadas no estágio anterior, são utilizadas dez instruções customizadas de comunicação por FIFO (FIFOSLV), representando as *features* calculadas para cada canal integral. O resultado da

classificação é enviado a um computador externo, via JTAG UART, para exibição em tela. O processador Nios II é equipado também memória *cache* de dados de 4 KB e *cache* de instruções de 2 KB, além de uma memória de dados fortemente acoplada para armazenamento de uma tabela com a sequência de utilização das *features* e do vetor de classificação SVM. Ao todo, são armazenadas 6250 palavras de 32 bits, ou 25 KB de memória. A tabela com a sequência de utilização das *features* é importante para que as *features* sejam lidas das FIFOs de entrada na ordem em que as operações com o vetor de classificação são realizadas. Embora a execução fora de ordem seja possível, o acesso sequencial a um conjunto de dados na memória permite ao compilador gerar código mais eficiente. A Figura 60 ilustra a arquitetura do processador de classificação.

Figura 60 – Arquitetura do processador de classificação, baseado no *softcore* Nios II.



Fonte: Elaborada pelo autor.

Conforme detectado na análise do perfil de execução da versão de referência do algoritmo ICF (subseção 4.3.1.1), a função encarregada da classificação SVM ocupa a maior parte do tempo total de execução. Para acelerar a execução dessa função, é adotada a mesma estratégia utilizada com o algoritmo HOG. O processador Nios II desta etapa é incrementado com a unidade MAC, descrita na subseção 4.2.2.2. Esta unidade implementa eficientemente operações de ponto flutuante do tipo multiplica-acumula, que é a base do cálculo da pontuação SVM. Assim, os dados recebidos são convertidos de representação em ponto fixo para ponto flutuante e então são processados pela unidade MAC.

O software no processador de classificação, inicialmente, lê da memória fortemente acoplada o índice do próximo canal a ser lido e realiza a leitura. Se porventura o dado não estiver disponível na FIFO de comunicação correspondente, o processador é bloqueado até que o dado seja enviado pelo processador de *features*. Após a leitura do valor da *feature*, o software lê o peso correspondente no vetor de classificação SVM e os fornece à unidade MAC, que libera o processador imediatamente para buscar outros dados. Quando todas as *features* já

foram submetidas e processadas, o processador lê o acumulador da unidade MAC, que contém a pontuação da janela de detecção.

CAPÍTULO

5

RESULTADOS

Neste capítulo são apresentados os resultados obtidos pelas implementações apresentadas no Capítulo 4. Inicialmente, são mostrados os resultados da primeira fase do projeto, que consistia na construção de uma arquitetura para execução do algoritmo HOG. A segunda fase compreende a construção de uma arquitetura para processar o algoritmo ICF, tendo como objetivo atender um sistema contendo quatro câmeras em um automóvel.

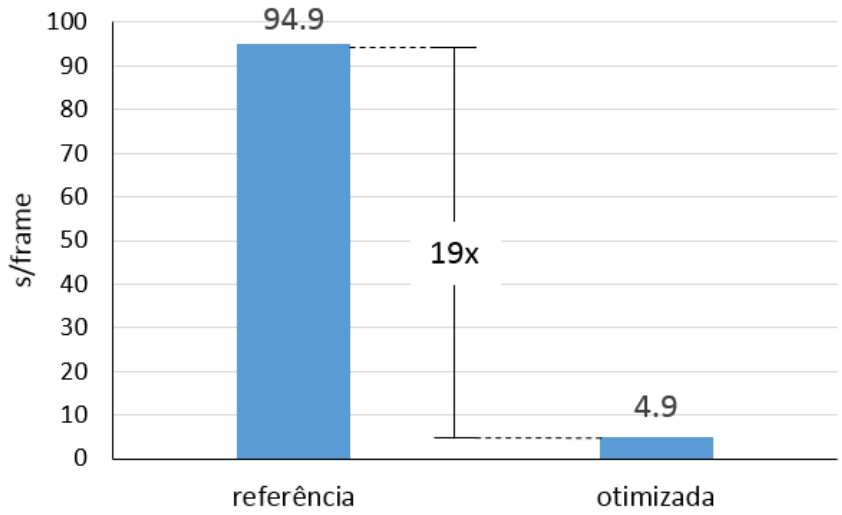
5.1 Resultados da Fase 1

Na fase 1, avalia-se o desempenho alcançado pela arquitetura *multi-core* para execução do algoritmo HOG e como esta implementação pode ser replicada de forma a atender demandas computacionais de um sistema de detecção de pedestres. Primeiramente são mostrados os resultados decorrentes das transformações de código, partindo da versão de referência para a versão otimizada.

O gráfico da Figura 61 mostra o ganho de desempenho obtido por meio das transformações de código e adoção do *buffer* de coluna para o reaproveitamento de dados de blocos já processados. A versão otimizada obteve um *speedup* de 19× comparada à versão de referência e reduzindo a influência das funções de processamento de bloco de 78% para 20%.

O uso de paralelismo temporal também provou ser eficaz na aceleração do algoritmo HOG. O uso de arquiteturas com dois e quatro estágios de *pipeline* permitiram atingir um *speedup* de $2,4\times$ quando comparados à versão otimizada. O *pipeline* com três estágios, contudo, supera os outros arranjos de processadores, atingindo um *speedup* de $2,5\times$. É importante observar que estes dados ainda não consideram o uso dos aceleradores de hardware. A Figura 62 mostra o tempo de execução por *frame* das versão sequenciais do algoritmo comparadas às versões em *pipeline*. Neste experimento, considera-se as versões com dois (*pipeline-2stg*), três (*pipeline-3stg*) e quatro (*pipeline-4stg*) estágios.

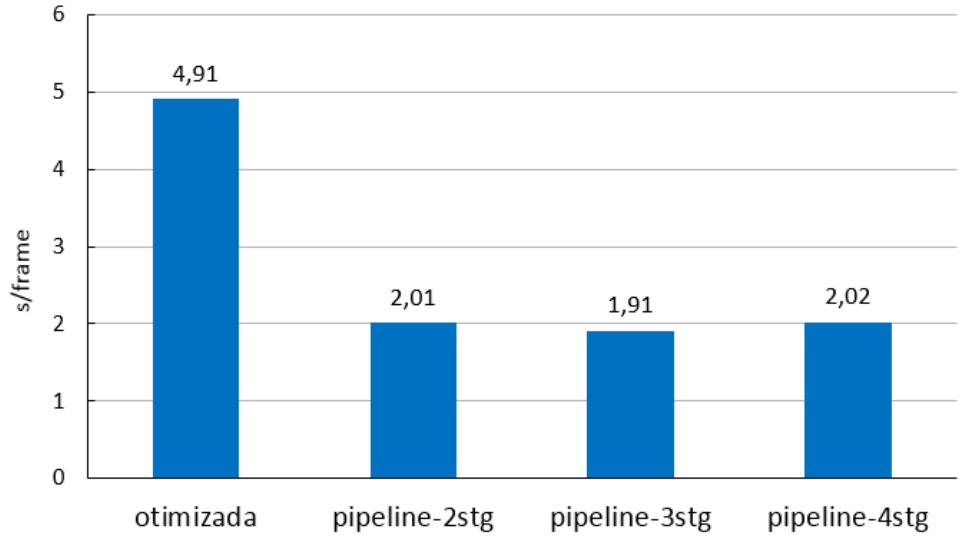
Figura 61 – Comparação dos tempos de execução das versões de referência e otimizada do algoritmo HOG executadas em uma arquitetura de único núcleo.



Fonte: Elaborada pelo autor.

A influência dos aceleradores de hardware no *pipeline* também é avaliada (Figura 63). Para isso, são realizados testes com os aceleradores em diferentes configurações do *pipeline*. O *pipeline* de dois estágios se beneficia mais do uso da Unidade de Histogramas, apresentando uma *speedup* de $1,12\times$. Juntas, as unidades MAC e de Histograma atingem um *speedup* de $1,28\times$ quando comparadas à versão sem aceleradores. Para o *pipeline* de três estágios, as unidades MAC e de Histograma contribuem com *speedups* de $1,21\times$ e $1,1\times$, respectivamente. Juntas, promovem um *speedup* de $1,28\times$ ($1,49$ segundos por *frame*) comparado à configuração de dois

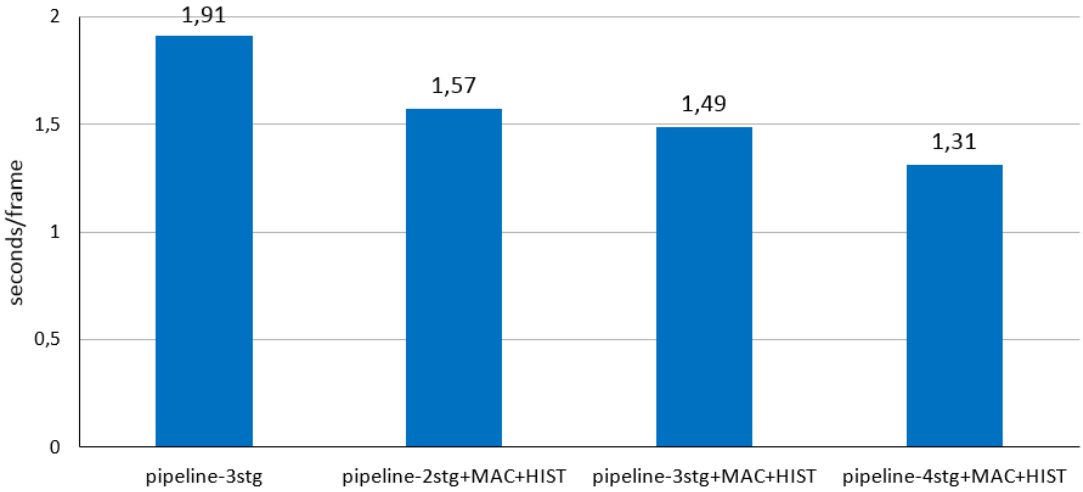
Figura 62 – Comparação dos tempos de execução do algoritmo HOG executado nas arquiteturas com 2, 3 e 4 núcleos de processamento.



Fonte: Elaborada pelo autor.

estágios (1,57 segundos por frame).

Figura 63 – Comparação dos tempos de execução do algoritmo HOG executado nas arquiteturas com 2, 3 e 4 núcleos de processamento e unidades aceleradoras.



Fonte: Elaborada pelo autor.

O *pipeline* de quatro estágios apresenta *speedups* de $1,23\times$ e $1,14\times$ para as unidades de Histograma e MAC, respectivamente. Os resultados obtidos utilizando ambos os aceleradores atingem um *speedup* de $1,54\times$, com o melhor tempo por *frame* de 1,31s. Comparada às versões de núcleo único, *speedups* de $3,74\times$ and $72,4\times$ são obtidos para as versões otimizada e de referência, respectivamente.

A distribuição do algoritmo nos processadores em *pipeline* não somente melhora o desempenho, mas também reduz a complexidade do código. Tal fato tem um impacto positivo na programabilidade do software, pois é possível obter um código mais limpo. A partir de uma análise de métricas do código, verifica-se que as transformações de código que culminaram na versão otimizada aumentaram em 20% a complexidade ciclomática máxima comparada à versão de referência. Esse aumento é devido principalmente à lógica de controle utilizada para manipular o *buffer* de coluna. Contudo, observa-se também que a melhoria no desempenho é bastante significativa.

Por meio da análise de métricas para versão *pipeline-3stg*, observou-se que a complexidade máxima do código é reduzida em 9% quando comparada à versão de referência e 27% com relação à otimizada. Além disso, a utilização de FIFOs para a comunicação entre processadores permite que os núcleos enviem e recebam dados na ordem em que são executados. Isto evita a adição de lógica de controle para a manipulação de índices em laços aninhados. Ainda, pode-se perceber um decréscimo de 18% no número de *statements* em relação à versão otimizada, o que sugere um código mais limpo e uma quantidade menor de memória ocupada pelo programa de cada processador.

A Tabela 7 mostra os recursos do FPGA ocupados pelos sistemas utilizados nos experi-

Tabela 6 – Comparação entre as métricas de qualidade de código de três versões implementadas do algoritmo HOG.

Métricas	**referência**	**otimizada**	***pipeline-3stg***
Linhas de código	993	1059	763
Statements	626	635	520
% de *Statements* de desvio	12,9	14,0	11,5
Complexidade Máxima	23	29	21
Máxima profundidade de bloco	6	8	6
Profundidade média de bloco	1,58	1,98	1,48
Complexidade Média	6,33	6.15	4,33

Fonte: Elaborada pelo autor.

mentos com o algoritmo HOG. Nota-se que as unidades de histograma ocupam três vezes mais espaço no FPGA do que as unidades MAC. Isto se deve à maior presença de hardware de controle e de estruturas de armazenamento para o histograma e para as FIFOs. A adição ao arranjo com 4 estágios de unidades MAC para a classificação SVM e para o cálculo da normalização L2-hys, além das unidades de histograma para o processamento de blocos aumentou em 20% a quantidade de elementos lógicos.

Tabela 7 – Relatório de utilização de recursos do FPGA para os sistemas com um único núcleo, com múltiplos núcleos e para as unidades de aceleração.

Configuration	**Elementos Lógicos**	**Bits de Memória**	**Multiplicadores**	**Registradores**
single-core	7.0K	2.5M	13	2.6K
pipeline-2stg	14.7K	4.6M	26	7.3K
pipeline-3stg	21.6K	5.0M	39	10.7K
pipeline-4stg	28.3K	5.1M	52	14.3K
Histogram unit	5.2K	42.4K	11	–
MAC unit	1.9K	2.0K	11	–

Fonte: Elaborada pelo autor.

5.2 Resultados da Fase 2

Com o objetivo de realizar a avaliação de desempenho da arquitetura *multi-core* para processamento do algoritmo ICF, ilustrada na Figura 53, foi utilizada a placa de desenvolvimento em FPGA Stratix V, mostrada na seção 4.1. Os dados da imagem de entrada, com resolução

320×240 pixels, são lidos a partir de uma memória ROM. As pontuações calculadas pelo SVM para cada janela de detecção são enviadas via módulo JTAG, localizado no processador de classificação, para um terminal de um computador *desktop* ao final da execução a fim de serem verificadas.

Embora se busque avaliar o desempenho geral da arquitetura, considera-se aqui também analisar o desempenho dos vários estágios da arquitetura separadamente, permitindo assim, uma melhor compreensão do potencial e das limitações das implementações realizadas a partir do coprojeto de hardware e de software. Inicialmente, são apresentados os resultados para o hardware de pré-processamento e processamento de canais. Ambos se encontram no mesmo domínio de *clock* de 10 MHz e são implementados como hardware fixo, porém parametrizável. Em seguida são analisados os dois estágios que fornecem programabilidade ao sistema e que se encontram no domínio de *clock* de 200 MHz: processamento de *features* e processamento de classificação.

Além de analisar o desempenho obtido pelo processamento da imagem, são feitas considerações a respeito do consumo de energia do hardware e a quantidade de recursos do FPGA ocupados por cada módulo. O consumo de energia é estimado utilizando-se a ferramenta *Powerplay Power Analyzer* da (Altera, 2016). Esta ferramenta funciona de forma integrada ao Quartus II e possibilita a geração de relatórios de consumo de energia de cada módulo a partir de informações de síntese e de simulação. Os dados de consumo dinâmico apresentados neste trabalho são obtidos por uma estimativa *vectorless*, ou seja, sem as informações de transição de sinal geradas por simulação. Ao invés disso, considera-se uma taxa média do chaveamento dos sinais internos aos módulos de 50%. Por padrão, considera-se um valor de $12,5\%$, que é a taxa de chaveamento de um contador em hardware. Contudo, neste trabalho procura-se fazer uma estimativa de pior caso, o que resultaria em um limite máximo. Apesar de possuir uma menor precisão do que uma estimativa com dados de simulação, os resultados da estimativa *vectorless* apresentam-se de forma satisfatória Kuon e Rose (2010). A informação sobre a utilização dos recursos do FPGA é fornecida pela ferramenta Quartus II de forma precisa, uma vez que a mesma é responsável pelas operações de alocação e roteamente no processo de compilação do hardware.

5.2.1 Pré-processamento e Processamento de Canais

Na arquitetura desenvolvida, as etapas de pré-processamento e geração de canais integrais são implementadas em hardware fixo, seguindo uma abordagem de processamento dos dados em *streaming*. Este tipo de implementação permite um aproveitamento máximo dos circuitos desenvolvidos, pois é possível consumir dados de entrada e gerar dados de saída a cada ciclo de *clock*.

A Tabela 8 apresenta os dados obtidos de desempenho, potência dissipada e ocupação de recursos do FPGA para os módulos de pré-processamento e processamento de canais. O

desempenho é representado pela latência de processamento de um pixel no módulo. A potência considerada é a dinâmica e é reportada em *miliWatts* (mW). Os recursos do FPGA são dados em *Adaptive Logic Module* (ALM), que é o bloco básico de construção dos FPGAs da Altera, quantidade de bits de memória *on-chip* ocupada e número de blocos DSP utilizados.

Tabela 8 – Desempenho, dissipação de potência e ocupação dos recursos do FPGA para os módulos de pré-processamento e processamento de canais.

Módulo	**ALM**	**Bits de Mem.**	**DSP**	**Pot. Dinâmica** (mW)	**Latência** (ciclos)
Filtro de Suavização (inclui ROM)	202	1854720	0	1,7	244
HOG	6055	18030	19	76,4	264
LUV	2199	58752	21	18,9	7
Magnitude	3943	0	15	38,2	4
Canal Integral	160	15296	0	0,5	1
Total	10293	2084462	40	102,7	519

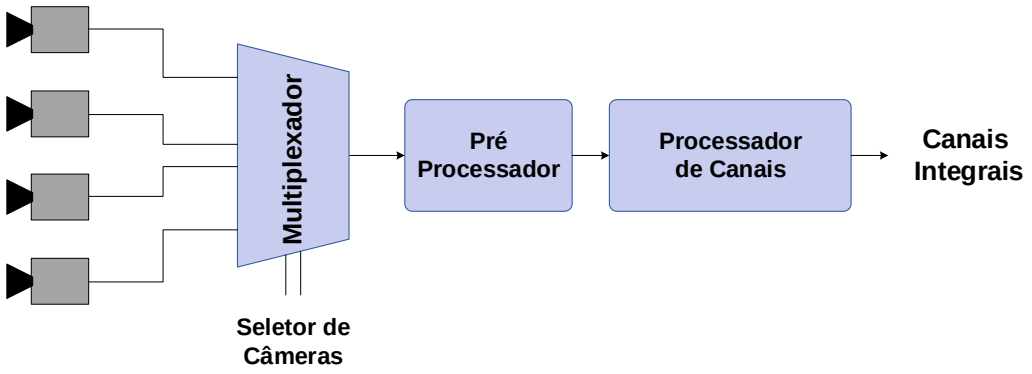
Fonte: Elaborada pelo autor.

O módulo do filtro de suavização apresenta uma grande ocupação de memória, pois, na implementação, além da lógica de filtragem, incluiu-se também a ROM que armazena a imagem de entrada. O maior módulo do sistema de geração de canais é o HOG, tanto em recursos ocupados, quanto em potência dissipada e também o que tem maior impacto na latência total. Nos experimentos, o hardware opera a 10 MHz, embora a frequência máxima possível seja de 15 MHz. Essa limitação decorre dos caminhos críticos gerados pelas operações de divisão em ponto fixo realizadas nos cálculos de arcotangente e LUV, utilizando as bibliotecas de ponto fixo da linguagem Bluespec.

Embora a frequência de operação seja significativamente menor em comparação aos estágios com processadores (200 MHz), a eficiência da implementação desse hardware possibilita o processamento de *frames* com resolução de 320×240 *pixels* a uma taxa de 130 fps. Em um sistema composto por quatro câmeras, isto significaria poder processar as imagens de cada uma a 32 fps, atendendo aos requisitos de sistemas ADAS (MODY, 2016). Neste caso, o hardware gerador de canais poderia ser compartilhado por quatro câmeras, cujas imagens são fornecidas ao sistemas de forma multiplexada, conforme ilustrado na Figura 64.

Quanto à potência dinâmica, o hardware gerador de canais dissipa apenas $102,7$ mW, o que representa 6% do consumo total da arquitetura de hardware do algoritmo ICF, mesmo ocupando 44% do total de ALMs e 16% do total de memória *on-chip*. Seu consumo é também 50% menor do que um processador de *features* utilizado no projeto. Um dos fatores envolvidos no baixo consumo é o uso mais intensivo de blocos DSP (72% do total), que, segundo Kuon e Rose (2010), são mais eficientes em termos de energia.

Figura 64 – Sistema com quatro câmeras multiplexadas utilizando o hardware de processamento de canais.



Fonte: Elaborada pelo autor.

5.2.2 Processamento de Features e Classificação

As etapas de processamento de *features* e classificação são implementadas de forma mais flexível, utilizando processadores *softcore* Nios II para executá-las. A flexibilização desses etapas permitem que o sistema seja adaptado a diferentes cenários e outros tipos de *features* sejam calculadas a partir dos canais integrais, bem como outros tipo de classificadores sejam utilizados. A tabela Tabela 9 mostra o desempenho atingido, os recursos utilizados no FPGA e o consumo de energia para os processadores de *features* e de classificação.

Tabela 9 – Desempenho, dissipação de potência e ocupação dos recursos do FPGA para os processadores de *feature* e de classificação.

Módulo	**ALM**	**Bits de Mem.**	**DSP**	**Pot. Dinâmica** (mW)	**Tempo /** ***frame*** (ms)
Processador de *Features*	1787	1808832	15	207,8	7
Processador de Classificação	3316	1086272	5	244,8	294
Total	12243	10130432	2	1442,3	519

Fonte: Elaborada pelo autor.

O tempo mostrado na última coluna da Tabela 8 refere-se à execução dos estágios do algoritmo nos processadores sem a interação com as estruturas de comunicação (FIFO), ou seja, sem envio ou recebimento de dados pelas FIFOs de comunicação. O propósito dessa medição é compreender o desempenho possível proporcionado por cada processador Nios II. Neste caso, a exploração do paralelismo no estágio de computação das *features* mostrou-se bastante eficiente, com cada processador de *features* processando 2 canais integrais em 7 ms. Já o processador de classificação é capaz de realizar os cálculos das pontuações para um *frame* em 294 ms, sem a interferência de comunicações externas. Este fato mostra uma oportunidade de otimização do último estágio do algoritmo, com a possível paralelização da classificação SVM para obtenção

de melhor desempenho. Verificou-se também que aumentos no tamanho da memória cache para estes processadores resultaram em nenhum ganho.

Percebe-se que o consumo de energia do processador de classificação é maior que o consumo do processador de *features*. Essa diferença no consumo se deve, principalmente, à presença da unidade MAC e do hardware de ponto flutuante, responsáveis por $13,9$mW e $18,7$mW, respectivamente. As instruções customizadas FIFO SLV contribuem pouco para o consumo total, com $0,01$ mW cada uma. Como comparação, as instruções FIFO MST, que servem para envio de dados pela FIFO, consomem $1,6$mW. Esse consumo maior se justifica pois o lado transmissor da instrução customizada implementa a FIFO de comunicação.

5.3 Considerações sobre a Arquitetura para o Algoritmo ICF

Os resultados obtidos para arquitetura de hardware para o algoritmo ICF como um todo estão descritas na Tabela 10. Os dados de ocupação, desempenho e consumo referem-se a uma única instância da cadeia de processamento desenvolvida no FPGA Stratix V. A ocupação desta instância, em termos de ALMs, foi de 10% e, em termos de DSP, foi de 21%. Foram ocupados também 25% da quantidade de bits de memória *on-chip* disponível, o que representa a utilização de somente 3 MB para o processamento de toda a cadeia de detecção. O consumo de energia do hardware desenvolvido (1,6W) encontra-se em níveis adequados para sistemas embarcados automotivos e é comparável ao de outras arquiteturas para detecção de pedestre baseada em processadores *softcore*, como a de Kelly *et al.* (2014).

Tabela 10 – Desempenho, dissipação de potência e ocupação dos recursos do FPGA finais para a arquitetura de processamento do algoritmo ICF.

.

Módulo	**ALM**	**Bits de Mem.**	**DSP**	**Pot. Dinâmica** (mW)	**Tempo /** ***frame*** (ms)
Total - Arquitetura (ICF)	23032	12870254	55	1686	583

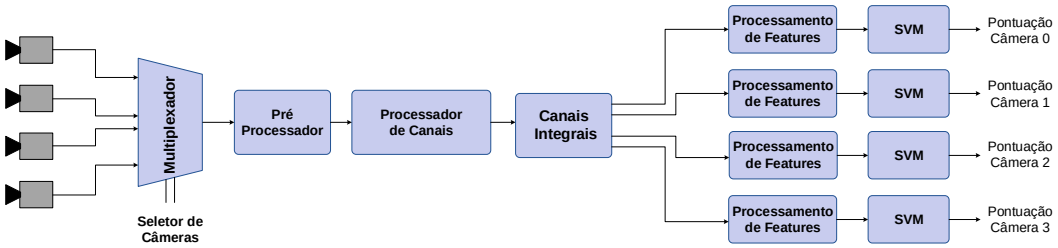
Fonte: Elaborada pelo autor.

Para o processamento de imagens de 4 câmeras de vídeo, é necessário considerar escalabilidade da arquitetura. A forma mais simples de se atender à essa demanda é por meio da replicação de todo o *stream*, de modo que cada câmera no sistema teria o seu fluxo exclusivo de processamento. Contudo, essa solução revela-se bastante custosa em termos de hardware, ainda que factível. Nos experimentos realizados neste trabalho, tentou-se realizar a replicação da arquitetura na placa de desenvolvimento Stratix V, que teoricamente, possui espaço suficiente para as 4 instâncias da arquitetura. Entretanto, apenas duas instanciações foram possíveis devido

à limitações de roteamento dos recursos do FPGA durante o processo de síntese, o que confirma o estudo de (Dini Group, 2016), que aponta apenas 60% dos recursos de um FPGA como utilizáveis. No caso do experimento realizado, o recurso limitante foi a memória *on-chip*.

É necessário, porém, considerar que atualmente existem FPGAs de grande capacidade, como o Stratix-10, da Intel, com aproximadamente o triplo de memória on-chip (aproximadamente 140.000 Kbits) do FPGA utilizado neste trabalho. Analisando-se a possibilidade de uso de um FPGA Stratix X - G(S)X5500 [1] neste projeto, quatro instâncias da arquitetura ocupariam 36% dos bits de memória, 4% dos ALM e 11% dos blocos DSPs, tornando o sistema viável. Contudo, devido à eficiência do hardware implementado, uma outra possibilidade de arquitetura é apresentada na Figura 65, como extensão à idéia apresentada na Figura 64. Nela, utiliza-se o hardware de processamento e geração de canais de forma compartilhada e então paralelizam-se os estágios de processamento de *features* e classifcação SVM para cada câmera. Dessa forma, obtém-se uma arquitetura mais econômica em termos de recursos de hardware.

Figura 65 – Sistema com quatro câmeras, compartilhando o hardware gerador de canais e com os estágios de processamento de *features* e classificação em paralelo.



Fonte: Elaborada pelo autor.

[1] Recursos deste modelo de FPGA disponíveis em <https://www.altera.com/products/fpga/stratix-series/stratix-10/overview.html>

CAPÍTULO

6

CONCLUSÃO

6.1 Aspectos Gerais

O trabalho apresentou uma discussão geral sobre Sistemas Avançados de Assistência ao Condutor (ADAS) e, mais especificamente, sobre sistemas de detecção de objetos baseados em câmeras de vídeo. Os tipos de objetos incluem pedestres, ciclistas, veículos, sinais de trânsito, faixas, obstáculos, entre outros. Dentro da área de pesquisa em Visão Computacional, os algoritmos para detecção de pedestres são considerados uma importante subclasse dos métodos para detecção de objetos, o que pode ser comprovado pela grande quantidade de pesquisas tanto acadêmicas quanto industriais presentes na literatura. Embora seja um assunto bastante explorado, novos algoritmos ou evoluções de algoritmos mais antigos ainda têm sido propostos com o objetivo de aumentar a precisão e melhorar o desempenho. Além disso, a demanda por esses sistemas têm crescido vertiginosamente. O enfoque deste trabalho esteve apenas nos algoritmos de detecção de pedestres. Contudo, não está proibitiva a sua expansão para os outros tipos de objetos já mencionados.

Algoritmos de detecção de pedestres baseados em visão demandam um grande poder computacional. Assim, implementações embarcadas desses algoritmos devem prover meios de atender aos requisitos de desempenho e consumo de energia, os quais são bastante restritivos para sistemas embarcados automotivos. Além desses requisitos, o ambiente de atuação de sistemas de detecção de pedestres é composto por inúmeros cenários que resultam em uma gama imensa de possibilidades de detecção. Para atender a essas demandas, utilizou-se a metodologia de coprojeto de hardware e software, visando maiores escalabilidade e flexibilidade do problema em questão e permitindo também, um melhor tratamento desta problemática.

Dentro desse contexto, a detecção de pedestres torna-se um desafio em termos de arquiteturas de sistemas embarcados, de modo que se possa atender às altas demandas computacionais no processamento de imagens, principalmente devido ao crescente aumento de resolução das

câmeras. Com a arquitetura desenvolvida neste trabalho, procurou-se tratar essas demandas e atendê-las na medida do possível.

O trabalho também faz uma discussão geral sobre os sistemas ADAS e suas possibilidades de implementação com computação reconfigurável, explorando duas fases de implementação. Na primeira fase, foi explorada uma arquitetura *multi-core* organizada na forma de um *pipeline* de processadores *softcore*, porém com desempenho aquém do esperado. Na segunda fase, a parte crítica do software foi transformada em hardware para obtenção de maior desempenho, sem, contudo, prejudicar sua flexibilidade com relação a adaptações algorítmicas em estágios importantes para implementação de outras técnicas de detecção.

Finalmente, cabe ressaltar que a arquitetura desenvolvida não se limita a detectar apenas pedestres, pois, com pequenas mudanças em estágios específicos da arquitetura, é possível a detecção de outros tipos de objetos, como placas de trânsito, automóveis, ciclistas, entre outros, bem como a adaptação a novos cenários ou evolução dos algoritmos.

6.2 Contribuições

A principal contribuição do trabalho está na discussão, no projeto, na implementação e na validação da arquitetura reconfigurável *multi-core* para detecção de pedestres baseada em visão. Neste sentido, optou-se pelos algoritmo HOG e ICF para extração de *features* e SVM para classificação dos objetos. Tais técnicas já são consagradas no meio acadêmico e industrial como representativas do estado da arte de detecção de objetos.

A principal inovação desta pesquisa apresenta-se sob a forma da inclusão e do tratamento simultâneo de detecção de pedestres com quatro câmeras acopladas ao veículo; abordagem esta que não foi encontrada na literatura até a presente data. Todos os trabalhos estudados incluíam uma única câmera frontal para a detecção de pedestres. A introdução da detecção de pedestres na câmera de ré mostra ser de uma utilidade imensa em virtude das péssimas estatísticas de atropelamento por parte dos condutores de veículos no processo de dar marcha à ré, como já mencionado na seção 1.2. O bipe auditivo, já implementado em muitos países, no momento em que o caminhão vai dar marcha à ré, tem se mostrado muitas vezes ineficiente e inoperante, causando acidentes fatais em crianças e adultos. Com a introdução da detecção de pedestres na câmera de ré, pretende-se diminuir drasticamente esse tipo de acidente, denominado *backovers* em inglês[1]. Assim, na iminência de um acidente, o sistema é capaz de emitir um aviso expresso na cabine do motorista, evitando-se a colisão do veículo com o indivíduo. Ainda, cabe destacar que, com as câmeras laterais, os pontos cegos do veículo são monitorados, evitando a colisão de pedestres na parte lateral do veículo. Tal situação pode ocorrer, por exemplo, quando um veículo sai em marcha à ré de uma vaga de estacionamento com pessoas passando ao seu lado. Com pequenas alterações em partes específicas da arquitetura, é possível ainda detectar

[1] Termo *backovers* retirado do website <http://www.kidsandcars.org/how-kids-get-hurt/backovers>

ciclistas e motociclistas posicionados nos pontos cegos, situação frequente no contexto do trânsito brasileiro.

Uma das vantagens no uso da arquitetura desenvolvida provém do fato de que ela fornece programabilidade em estágios importantes do algoritmo ICF. Esta programabilidade torna a solução flexível o suficiente para que sejam aplicadas otimizações, evoluções ou adaptações algorítmicas. Como exemplo, pode-se citar a possibilidade de substituição da técnica de classificação de objetos, bastando substituir o código fonte no processador correspondente. Outro exemplo seria evoluir o ICF para um algoritmo da mesma família, como o descrito por Zhang, Benenson e Schiele (2015) ou por Zhang, Bauckhage e Cremers (2014), bastando, nesse caso, implementar o cálculo de *features* de segunda ordem nos processadores de *features*. Uma segunda vantagem da arquitetura desenvolvida reside em sua escalabilidade, uma vez que é possível replicar estágios ou até mesmo toda a cadeia de processamento a fim de obter maior desempenho ou processar uma maior quantidade de dados.

Da mesma forma, é importante reconhecer as limitações e desvantagens da arquitetura. Uma delas diz respeito à baixa taxa de *frames* processados por segundo, o que representa uma limitação no desempenho do fluxo de processamento da arquitetura. A solução para esta situação pode ocorrer de duas formas. Na primeira, replica-se todo o fluxo de detecção e divide-se o processamento da imagem de entrada entre os vários fluxos, diminuindo assim, a quantidade de dados processada em cada fluxo. A segunda solução passa por identificar o módulo responsável por processar o gargalo computacional do algoritmo e replicá-lo, de forma que se obtenha um processamento paralelo dos dados do estágio em questão. Por exemplo, pode-se aumentar a quantidade de processadores de *features* por canal, no caso em que o cálculo das *features* seja a parte crítica do fluxo de processamento. Outra desvantagem reside no fato de que, nesta implementação, somente uma escala da imagem de entrada é processada. Com isso, a detecção de pedestres em uma imagem fica limitada a objetos de mesmo tamanho, como já explicado na subseção 4.3.2. Isto pode ser resolvido também aproveitando-se a escalabilidade da arquitetura. Neste caso, a solução passa pela replicação de todo o fluxo de detecção, de forma que o processamento das múltiplas escalas seja dividido entre os múltiplos fluxos. Para isto, é necessário adição de um elemento de processamento no início do fluxo capaz de redimensionar a imagem de entrada e fornecer as diferentes escalas ao vários elementos de processamento de canais. Cabe ressaltar que as soluções para as duas desvantagens mencionadas ficam limitadas pela capacidade do dispositivo FPGA, as quais foram analisadas na seção 5.2.

A parte mais crítica e importante para o sucesso da arquitetura projetada foi a construção dos canais utilizados no algoritmo ICF totalmente em hardware. Tal abordagem também é desconhecida na literatura. Ao aproveitar o desempenho oferecido por esse hardware, torna-se possível reprojetar a arquitetura de forma que o hardware de canais atenda a mais de uma câmera de video por meio de uma multiplexação das entradas. Essa possibilidade representa uma vantagem, pois atende-se a uma maior demanda de processamento e diminui-se a ocupação

de portas lógicas dentro do FPGA. Ademais, a construção desse módulo em *stream* reduz a necessidade de armazenamento temporário de dados e, consequentemente, a quantidade de memória *on-chip* utilizada.

A distribuição de parte do processamento do algoritmo ICF e também do HOG entre diversos núcleos de processamento em *pipeline* e a interação com a infraestrutura de comunicação intra-núcleos exigiu uma transformação do software no sentido de delimitar suas funções e explorar mais eficientemente os recursos embarcados disponíveis. Como consequência, observou-se uma diminuição na complexidade e tamanho dos códigos gerados, traduzindo-se em uma melhor clareza da codificação e reuso facilitado.

Devido ao número reduzido de testes com bases de dados de imagens de pedestres, se faz necessária a realização de uma quantidade considerável de análise sobre a taxa de acertos dos algoritmos. Entretanto, os resultados iniciais apresentados nesta pesquisa estimulam sua continuidade, propiciando a aplicação de outras bases de dados de pedestres (como Caltech, Kitti, Daimler) para uma melhor análise do comportamento e da precisão do sistema como um todo.

Finalmente, a aplicação da computação reconfigurável demonstrou ser a única possibilidade viável de implementação dessa arquitetura, que utilizou um hardware de aproximadamente 1 milhão de portas lógicas. Sem o uso da computação reconfigurável seria impossível a realização de tal sistema embarcado, uma vez que o Brasil não possui um infraestrutura para confecção de circuitos semicondutores de tal porte. Assim, com a combinação de diversas áreas do conhecimento, tais como Processamento de Imagens, Aprendizado de Máquinas, Computação Paralela, Compiladores, Computação Reconfigurável e Arquitetura de Computadores, tornou-se possível a realização desta pesquisa, que vem contribuir de maneira significativa para a área de hardware, tão carente no Brasil.

6.3 Trabalhos Futuros

Durante a realização desta pesquisa, muitas propostas para detecção de pedestres foram discutidas. Fica evidente que a expansão da arquitetura para detectar carros, bicicletas, motocicletas e placas de trânsito seria imprescindível para o sucesso total deste trabalho. A realização de tais implementações envolve um trabalho de engenharia considerável, porém de baixa complexidade.

Em uma escala de menor complexidade, cabem ainda os seguintes trabalhos futuros:

- Utilização de representação em ponto flutuante ao invés de ponto fixo no cálculo dos canais

- Integração da arquitetura desenvolvida com câmeras de vídeo. Pelo fato da arquitetura

lidar com informações de imagem em baixo nível, qualquer tipo de câmera poderá ser tratada por este trabalho.

REFERÊNCIAS

ABELE, J.; KRUEGER, S.; BAUM, H.; GEISSLER, T.; GRAWENHOFF, S.; SCHNEIDER, J.; SCHULZ, W. H. **Exploratory Study on the potential socio-economic impact of the introduction of Intelligent Safety Systems in Road Vehicles**. [S.l.], 2005. Citado na página 44.

ADVANI, S.; TANABE, Y.; IRICK, K.; SAMPSON, J.; NARAYANAN, V. A scalable architecture for multi-class visual object detection. In: **25th International Conference on Field Programmable Logic and Applications**. London: IEEE, 2015. p. 1–8. Citado nas páginas 32 e 75.

AGARWAL, S.; AWAN, A.; ROTH, D. Learning to detect objects in images via a sparse, part-based representation. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE Computer Society, Washington, DC, USA, v. 26, n. 11, p. 1475–1490, Novembro 2004. ISSN 0162-8828. Citado na página 62.

AKHLAQ, M.; SHELTAMI, T.; HELGESON, B.; SHAKSHUKI, E. M. Designing an integrated driver assistance system using image sensors. **Journal of Intelligent Manufacturing**, Springer US, v. 23, n. 6, p. 2109–2132, 2012. ISSN 0956-5515. Citado nas páginas 44, 45 e 47.

ALTERA. **Quartus II Handbook Volume 3**. [S.l.], 2005. Citado na página 98.

____. **Stratix V GX FPGA Development Board: Reference Manual**. San Jose, California, 2012. Citado nas páginas 82 e 83.

Altera. **Quartus Prime Standard Edition Handbook**. 2016. Citado na página 125.

Altera Corporation. **Nios II Classic Processor Reference Guide**. San Jose, CA, United States of America: [s.n.], 2015. Citado na página 89.

BAILEY, D. G. **Design for embedded image processing on FPGAs**. [S.l.]: Wiley. com, 2011. Citado nas páginas 39 e 40.

BAUER, S.; KOHLER, S.; DOLL, K.; BRUNSMANN, U. Fpga-gpu architecture for kernel svm pedestrian detection. In: IEEE. **Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on**. [S.l.], 2010. p. 61–68. Citado na página 75.

BENENSON, R.; MATHIAS, M.; TUYTELAARS, T.; GOOL, L. V. Seeking the strongest rigid detector. In: **IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2013. p. 3666–3673. Citado na página 63.

BENENSON, R.; OMRAN, M.; HOSANG, J. H.; SCHIELE, B. Ten years of pedestrian detection, what have we learned? In: **Computer Vision - ECCV 2014 Workshops**. Zurich, Switzerland: [s.n.], 2014. p. 613–627. Citado nas páginas 30, 31, 63, 65, 66, 67 e 102.

BERGER, A.; BERGER, A. **Embedded Systems Design: An Introduction to Processes, Tools, and Techniques**. CMP Books, 2002. (CMP Books). ISBN 9781578200733. Disponível em: <http://books.google.com.br/books?id=sSoPLxAQrtkC>. Citado na página 38.

BERTOZZI, M.; BROGGI, A.; CELLARIO, M.; FASCIOLI, A.; LOMBARDI, P.; PORTA, M. Artificial vision in road vehicles. In: **Proceedings of the IEEE**. [S.l.: s.n.], 2002. v. 90, n. 7, p. 1258–1271. Citado na página 30.

BISHOP, R. **Intelligent vehicle technology and trends**. [S.l.]: Artech House ITS library, 2005. Citado nas páginas 43 e 44.

BLAIR, C. G. **Real-time video scene analysis with heterogeneous processors**. Tese (Doutorado) — University of Glasgow, UK, 2014. Disponível em: <http://theses.gla.ac.uk/5061/>. Citado na página 75.

BOBDA, C. **Introduction to Reconfigurable Computing: Architectures, Algorithms, and Applications**. Springer, 2010. ISBN 9789048175314. Disponível em: <http://books.google.com.br/books?id=r_3KcQAACAAJ>. Citado nas páginas 35, 36 e 37.

BONDALAPATI, K.; PRASANNA, V. K. Reconfigurable computing systems. In: **Proceedings of the IEEE**. [S.l.: s.n.], 2002. p. 1201–1217. Citado na página 38.

BRADSKI, G. The opencv library. **Doctor Dobbs Journal**, M AND T PUBLISHING INC, v. 25, n. 11, p. 120–126, 2000. Citado na página 85.

BROGGI, A.; BERTOZZI, M.; FASCIOLI, A.; SECHI, M. Shape-based pedestrian detection. In: **Intelligent Vehicles Symposium, 2000. IV 2000. Proceedings of the IEEE**. [S.l.: s.n.], 2000. p. 215–220. Citado na página 62.

BROGGI, A.; FASCIOLI, A.; FEDRIGA, I.; TIBALDI, A.; ROSE, M. Stereo-based preprocessing for human shape localization in unstructured environments. In: **Intelligent Vehicles Symposium, 2003. Proceedings. IEEE**. [S.l.: s.n.], 2003. p. 410–415. Citado na página 62.

BU, F.; CHAN, C.-Y. Pedestrian detection in transit bus application: sensing technologies and safety solutions. In: **Proceedings of the IEEE Intelligent Vehicles Symposium**. [S.l.]: IEEE, 2005. p. 100–105. Citado na página 30.

BURNS, P. C. **International Harmonized Research Activities - Intelligent Transport Systems (IHRAITS) Working Group Report**. [S.l.], 2005. Http://www-nrd.nhtsa.dot.gov/pdf/esv/esv19/05-0461-O.pdf (Paper number 05-0461) (Acessador em 23/08/2013). Citado na página 46.

CLEMONS, J.; JONES, A.; PERRICONE, R.; SAVARESE, S.; AUSTIN, T. Effex: an embedded processor for computer vision based feature extraction. In: **Proceedings of the 48th Design Automation Conference**. New York, NY, USA: ACM, 2011. (DAC '11), p. 1020–1025. ISBN 978-1-4503-0636-2. Disponível em: <http://doi.acm.org/10.1145/2024724.2024949>. Citado nas páginas 32 e 76.

COMPTON, K.; HAUCK, S. An introduction to reconfigurable computing. **IEEE Computer**, Apr 2000. Citado na página 37.

CROW, F. C. Summed-area tables for texture mapping. In: **Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques**. New York, NY, USA: ACM, 1984. (SIGGRAPH '84), p. 207–212. ISBN 0-89791-138-5. Citado na página 71.

DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. In: **IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2005. v. 1, p. 886–893. ISSN 1063-6919. Citado nas páginas 30, 61, 62, 64, 65, 69 e 85.

Dini Group. **Dini Group FPGA Selection Guide**. 2016. Dinigroup.com. Disponível em: <http://www.dinigroup.com/product/common/DINI_selection_guide_v431_2.pdf>. Citado na página 129.

DNIT. **Estatsticas de Acidentes**. 2011. Disponível em: <http://www.dnit.gov.br/rodovias/operacoes-rodoviarias/estatisticas-de-acidentes>. Citado na página 27.

DOLLAR, P.; BELONGIE, S.; PERONA, P. The fastest pedestrian detector in the west. In: **Proceedings of the British Machine Vision Conference**. [S.l.]: BMVA Press, 2010. p. 68.1–68.11. ISBN 1-901725-40-5. Doi:10.5244/C.24.68. Citado na página 63.

DOLLAR, P.; TU, Z.; PERONA, P.; BELONGIE, S. Integral channel features. In: **Proceedings of the British Machine Vision Conference**. London, England: BMVA Press, 2009. p. 91.1–91.11. ISBN 1-901725-39-1. Citado nas páginas 30, 62, 63, 64, 66, 71, 72, 74 e 102.

DOLLAR, P.; WOJEK, C.; SCHIELE, B.; PERONA, P. Pedestrian detection: A benchmark. In: **Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on**. [S.l.: s.n.], 2009. p. 304–311. ISSN 1063-6919. Citado na página 30.

____. Pedestrian detection: An evaluation of the state of the art. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 34, n. 4, p. 743–761, 2012. ISSN 0162-8828. Citado nas páginas 30, 55, 61, 65 e 66.

ENZWEILER, M.; GAVRILA, D. Monocular pedestrian detection: Survey and experiments. **Pattern Analysis and Machine Intelligence, IEEE Transactions on**, v. 31, n. 12, p. 2179–2195, 2009. Citado nas páginas 30 e 61.

ESTRIN, G.; BUSSELL, B.; TURN, R.; BIBB, J. Parallel processing in a restructurable computer system. **IEEE Transactions on Electronic Computers**, EC-12, n. 5, p. 747–755, December 1963. Citado na página 36.

FELZENSZWALB, P. F.; GIRSHICK, R. B.; MCALLESTER, D.; RAMANAN, D. Object detection with discriminatively trained part-based models. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 32, n. 9, p. 1627–1645, Sept 2010. Citado nas páginas 62, 64 e 68.

FENLASON, J.; STALLMAN, R. **GNU gprof: The GNU Profiler**. 1998. Disponível em: <ftp://ftp.gnu.org/pub/old-gnu/Manuals/gprof/html_chapter/gprof_toc.html>. Citado na página 86.

FRANKE, U.; JOOS, A. Real-time stereo vision for urban traffic scene understanding. In: **Intelligent Vehicles Symposium, 2000. IV 2000. Proceedings of the IEEE**. [S.l.: s.n.], 2000. p. 273–278. Citado na página 62.

FRANKE, U.; KUTZBACH, I. Fast stereo based object detection for stop and go traffic. In: **Proceedings of intelligent vehicles symposium**. [S.l.: s.n.], 1996. p. 339–344. Citado na página 62.

FREUND, Y.; SCHAPIRE, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. **Journal of Computer and System Sciences**, v. 55, n. 1, p. 119 – 139, 1997. Citado na página 64.

GANDHI, T.; TRIVEDI, M. M. Pedestrian collision avoidance systems: A survey of computer vision based recent studies. In: **Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC'06)**. Toronto, Canada: IEEE, 2006. p. 976–981. Citado na página 30.

GAVRILA, D. A bayesian, exemplar-based approach to hierarchical shape matching. **Pattern Analysis and Machine Intelligence, IEEE Transactions on**, v. 29, n. 8, p. 1408–1421, 2007. ISSN 0162-8828. Citado na página 62.

GAVRILA, D.; PHILOMIN, V. Real-time object detection for ldquo;smart rdquo; vehicles. In: **Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on**. [S.l.: s.n.], 1999. v. 1, p. 87–93 vol.1. Citado na página 62.

GAVRILA, D. M. Sensor-based pedestrian protection. **IEEE Intelligent Systems**, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 16, n. 6, p. 77–81, 2001. Citado na página 30.

GAVRILA, D. M.; MARCHAL, P.; MEINECKE, M. M. **Deliverable 1-A: Vulnerable Road User Scenario Analysis**. [S.l.], 2003. Citado nas páginas 49, 50, 51, 52, 53 e 54.

GERONIMO, D.; LOPEZ, A.; SAPPA, A.; GRAF, T. Survey of pedestrian detection for advanced driver assistance systems. **Pattern Analysis and Machine Intelligence, IEEE Transactions on**, v. 32, n. 7, p. 1239–1258, 2010. Citado nas páginas 30, 44, 52, 54, 59 e 60.

GERONIMO, D.; SAPPA, A.; PEñA, A. L.; PONSA, D. Adaptive image sampling and windows classification for on-board pedestrian detection. In: **Proceedings of the 5th International Conference on Computer Vision Systems (ICVS07)**. Bielefeld (Germany): [s.n.], 2007. Citado na página 61.

Global Market Insights. **Embedded System Market Size By Application (Automotive, Industrial, Consumer Electronics, Telecommunication, Healthcare, Military & Aerospace), By Product (Software, Hardware) Industry Outlook Report, Regional Analysis, Application Development Potential, Price Trends, Competitive Market Share & Forecast, 2016 to 2023**. [S.l.], 2016. Disponível em: <https://www.gminsights.com/industry-analysis/embedded-system-market>. Citado na página 27.

GRUBB, G.; ZELINSKY, A.; NILSSON, L.; RILBE, M. 3d vision sensing for improved pedestrian safety. In: **Intelligent Vehicles Symposium, 2004 IEEE**. [S.l.: s.n.], 2004. p. 19–24. Citado na página 62.

HAHNLE, M.; SAXEN, F.; HISUNG, M.; BRUNSMANN, U.; DOLL, K. Fpga-based real-time pedestrian detection on high-resolution images. In: **IEEE Conference on Computer Vision and Pattern Recognition Workshops**. Portland: IEEE, 2013. p. 629–635. Citado nas páginas 75 e 95.

HEMMATI, M.; BIGLARI-ABHARI, M.; BERBER, S.; NIAR, S. Hog feature extractor hardware accelerator for real-time pedestrian detection. In: **17th Euromicro Conference on Digital System Design**. Verona: IEEE, 2014. p. 543–550. Citado na página 75.

HIROMOTO, M.; MIYAMOTO, R. Hardware architecture for high-accuracy real-time pedestrian detection with cohog features. In: IEEE. **Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on**. [S.l.], 2009. p. 894–899. Citado na página 75.

HSIUNG, P.-A.; SANTAMBROGIO, M. D.; HUANG, C.-H. **Reconfigurable System Design and Verification**. 1st. ed. Boca Raton, FL, USA: CRC Press, Inc., 2009. ISBN 1420062662, 9781420062663. Citado na página 37.

IHRA. **Design Principles for Advanced Driver Assistance Systems: Keeping Drivers In-the-Loop**. [S.l.], 2011. Disponível em http://www.unece.org/fileadmin/DAM/trans/doc/2011/wp29/ITS-19-07e.pdf (Acessado em 23/08/2013). Disponível em: <http://www.unece.org/fileadmin/DAM/trans/doc/2011/wp29/ITS-19-07e.pdf>. Citado na página 45.

JOHANSSON, E.; ENGSTRöM, J.; CHERRI, C.; NODARI, E.; TOFFETTI, A.; SCHINDHELM, R.; GELAU, C. **Review of existing techniques and metrics for IVIS and ADAS assessment**. [S.l.], 2004. Citado na página 44.

JORGE, M. H. P. de M.; KOIZUMI, M. S. Acidentes de trânsito no brasil: um atlas de sua distribuição. **ABRAMET. Associação Brasileira de Medicina de Tráfego, São Paulo**, v. 26, n. 1, p. 52–58, 2008. Citado nas páginas 49, 50 e 51.

KADOTA, R.; SUGANO, H.; HIROMOTO, M.; OCHI, H.; MIYAMOTO, R.; NAKAMURA, Y. Hardware architecture for hog feature extraction. In: **Proceedings of the 2009 Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing**. Washington, DC, USA: IEEE Computer Society, 2009. (IIH-MSP '09), p. 1330–1333. ISBN 978-0-7695-3762-7. Disponível em: <http://dx.doi.org/10.1109/IIH-MSP.2009.216>. Citado na página 75.

KELLY, C.; SIDDIQUI, F.; BARDAK, B.; WOODS, R. Histogram of oriented gradients front end processing: An fpga based processor approach. In: **IEEE Workshop on Signal Processing Systems**. Belfast: IEEE, 2014. p. 1–6. Citado nas páginas 32, 76, 77, 95 e 128.

Kids and Cars. **Backover Fact Sheet**. 2014. Kidsandcars.org. Disponível em: <http://www.kidsandcars.org/files/2015/04/Backover-fact-sheet-FINAL1.pdf>. Citado na página 32.

KIM, J.; SHIN, H. **Algorithm & SoC Design for Automotive Vision Systems: For Smart Safe Driving System**. 1. ed. [S.l.]: Springer Netherlands, 2014. Citado nas páginas 60 e 63.

KNOLL, P. Hdr vision for driver assistance. In: HOEFFLINGER, B. (Ed.). **High-Dynamic-Range (HDR) Vision**. Springer Berlin Heidelberg, 2007, (ADVANCED MICROELECTRONICS, v. 26). p. 123–136. ISBN 978-3-540-44432-9. Disponível em: <http://dx.doi.org/10.1007/978-3-540-44433-6_8>. Citado na página 60.

KOMORKIEWICZ, M.; KLUCZEWSKI, M.; GORGON, M. Floating point hog implementation for real-time multiple object detection. In: **22nd International Conference on Field Programmable Logic and Applications**. Oslo: IEEE, 2012. p. 711–714. Citado na página 75.

KUON, I.; ROSE, J. **Quantifying and Exploring the Gap Between FPGAs and ASICs**. [S.l.]: Springer US, 2010. Citado nas páginas 125 e 126.

KURODA, I.; KYO, S. Media processing lsi architectures for automotives – challenges and future trends. **IEICE TRANS. ELECTRON.**, E90–C, n. 10, p. 1850–1857, October 2007. Citado nas páginas 29 e 32.

LANGE, R. de. **Aprosys SP 1 - detailed draft test scenarios for a specific pre-crash safety system - deliverable 1.3.3 - Revision: Final**. 2007. Nederlands Organsiatie voor toegepast natuurwetenschappelijk onderzoek (TNO). Disponível em http://www.transport-research.info/Upload/Documents/201203/20120313$_1$43540$_7$9385$_A$P − SP13 − 0035$_D$133.pdf(Acessadoem09/2013).Citadonaspginas53e 54.

LEE, S.; SON, H.; CHOI, J.-C.; MIN, K. High-performance hog feature extractor circuit for driver assistance system. In: **Consumer Electronics (ICCE), 2013 IEEE International Conference on**. [S.l.: s.n.], 2013. p. 338–339. ISSN 2158-3994. Citado na página 75.

LEIBE, B.; CORNELIS, N.; CORNELIS, K.; GOOL, L. V. Dynamic 3d scene analysis from a moving vehicle. In: **Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on**. [S.l.: s.n.], 2007. p. 1–8. ISSN 1063-6919. Citado na página 62.

LIM, J. J.; ZITNICK, C. L.; DOLLAR, P. Sketch tokens: A learned mid-level representation for contour and object detection. In: **IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2013. p. 3158–3165. Citado na página 63.

LINDER, A.; KIRCHER, A.; VADEBY, A.; NYGåRDHS, S. **Intelligent Transport Systems (ITS) in passenger cars and methods for assessment of traffic safety impact : a literature review**. [S.l.]: VTI, 2007. 108 p. Citado na página 44.

LIU, X.; FUJIMURA, K. Pedestrian detection using stereo night vision. **Vehicular Technology, IEEE Transactions on**, v. 53, n. 6, p. 1657–1665, 2004. ISSN 0018-9545. Citado na página 61.

LIU, Y.; SHAN, S.; ZHANG, W.; CHEN, X.; GAO, W. Granularity-tunable gradients partition (ggp) descriptors for human detection. In: **Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on**. [S.l.: s.n.], 2009. p. 1255–1262. ISSN 1063-6919. Citado na página 62.

LOWE, D. G. Distinctive image features from scale-invariant keypoints. **International Journal of Computer Vision**, Kluwer Academic Publishers, Hingham, MA, USA, v. 60, n. 2, p. 91–110, nov 2004. ISSN 0920-5691. Disponível em: <http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94>. Citado na página 62.

MA, X.; NAJJAR, W.; ROY-CHOWDHURY, A. Evaluation and acceleration of high-throughput fixed-point object detection on fpgas. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 25, n. 6, p. 1051–1062, June 2015. ISSN 1051-8215. Citado nas páginas 75, 94 e 95.

MARSI, S.; IMPOCO, G.; UKOVICH, A.; CARRATO, S.; RAMPONI, G. Video enhancement and dynamic range control of hdr sequences for automotive applications. **EURASIP Journal on Advances in Signal Processing**, v. 2007, n. 1, p. 080971, 2007. ISSN 1687-6180. Disponível em: <http://asp.eurasipjournals.com/content/2007/1/080971>. Citado na página 60.

MARTELLI, S.; TOSATO, D.; CRISTANI, M.; MURINO, V. Fpga-based pedestrian detection using array of covariance features. In: **Distributed Smart Cameras (ICDSC), 2011 Fifth ACM/IEEE International Conference on**. [S.l.: s.n.], 2011. p. 1–6. Citado na página 75.

MARTINEZ, L. A. **Projeto de um sistema embarcado de predição de colisão e pedestres baseado em computação reconfigurável.** Dissertação (Mestrado) — Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, SP, Brasil, 2011. Citado na página 55.

MIZUNO, K.; TERACHI, Y.; TAKAGI, K.; IZUMI, S.; KAWAGUCHI, H.; YOSHIMOTO, M. Architectural study of hog feature extraction processor for real-time object detection. In: **Signal Processing Systems (SiPS), 2012 IEEE Workshop on**. [S.l.: s.n.], 2012. p. 197–202. ISSN 2162-3562. Citado na página 75.

MOBILEYE. **Mobileye.com**. 2015. Disponível em: <www.mobileye.com>. Citado nas páginas 31, 32 e 78.

MODY, M. **ADAS Front Camera: Demystifying Resolution and Frame-Rate**. 2016. Eetimes.com. Disponível em: <http://www.eetimes.com/author.asp?section_id=36&doc_id=1329109>. Citado na página 126.

MODY, M.; SWAMI, P.; CHITNIS, K.; VISWANATH, P. **Computer Vision for Automotive/ADAS Market: Challenges and Embedded Vision Solutions**. 2016. Tutorial at the 29th IEEE Conference on Computer Vision and Pattern Recognition. Citado nas páginas 27, 28 e 31.

MOORE, G. Cramming more components onto integrated circuits. **Proceedings of the IEEE**, v. 86, n. 1, p. 82–85, 1998. ISSN 0018-9219. Citado na página 35.

MRAK, M.; GRGIC, M.; KUNT, M. **High-Quality Visual Experience: Creation, Processing and Interactivity of High-Resolution and High-Dimensional Video Signals**. [S.l.]: Springer-Verlag Berlin Heidelberg, 2010. (Signals and Communication Technology, 1). Citado na página 73.

NAM, W.; DOLLAR, P.; HAN, J. H. Local decorrelation for improved pedestrian detection. In: **Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems**. Montreal, Quebec, Canada: [s.n.], 2014. p. 424–432. Disponível em: <http://papers.nips.cc/paper/5419-local-decorrelation-for-improved-pedestrian-detection>. Citado na página 63.

NEGI, K.; DOHI, K.; SHIBATA, Y.; OGURI, K. Deep pipelined one-chip fpga implementation of a real-time image-based human detection algorithm. In: IEEE. **Field-Programmable Technology (FPT), 2011 International Conference on**. [S.l.], 2011. p. 1–8. Citado na página 75.

NISHIWAKI, D.; TAKAHASHI, K.; ISHIDERA, E.; KYO, S.; OKAZAKI, S. In-vehicle image recognition technologies for the assistance of safe driving. In: **Proceedings of the Vision Engineering Workshop (ViEW)**. [S.l.]: EK-1, 2006. p. 189–195. Citado na página 29.

NVIDIA. **Get Under the Hood of Parker, Our Newest SOC for Autonomous Vehicles**. 2016. Disponível em: <https://blogs.nvidia.com/blog/2016/08/22/parker-for-self-driving-cars/>. Citado nas páginas 31, 32, 78 e 79.

ODROID. **ODROID-XU3 Wiki**. 2015. Disponível em: <http://odroid.com/dokuwiki/doku.php?id=en:odroid-xu3>. Citado na página 84.

OSADJA, M. **Automotive Market trends - Driving Automotive Semiconductor into a Greener, Safety Oriented Future**. 2008. Freescale technology Forum. Citado na página 27.

PAISITKRIANGKRAI, S.; SHEN, C.; van den Hengel, A. Efficient pedestrian detection by directly optimizing the partial area under the ROC curve. In: **IEEE International Conference on Computer Vision (ICCV'13)**. Sydney, Australia: [s.n.], 2013. Citado na página 63.

____. Strengthening the effectiveness of pedestrian detection with spatially pooled features. In: **Proc. European Conf. Comp. Vis.** [S.l.: s.n.], 2014. Citado na página 63.

PAPAGEORGIOU, C.; POGGIO, T. A trainable system for object detection. **Int. J. Comput. Vision**, Kluwer Academic Publishers, Hingham, MA, USA, v. 38, n. 1, p. 15–33, jun. 2000. ISSN 0920-5691. Disponível em: <http://dx.doi.org/10.1023/A:1008162616689>. Citado na página 61.

PICCARDI, M. Background subtraction techniques: a review. In: **IEEE International Conference on Systems, Man and Cybernetics**. [S.l.: s.n.], 2004. v. 4, p. 3099–3104 vol.4. ISSN 1062-922X. Citado na página 61.

SABZMEYDANI, P.; MORI, G. Detecting pedestrians by learning shapelet features. In: **Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on**. [S.l.: s.n.], 2007. p. 1–8. ISSN 1063-6919. Citado na página 62.

SCHAUMONT, P. R. **A Practical Introduction to Hardware/Software Codesign**. 2nd. ed. [S.l.]: Springer US, 2013. ISBN 1441959998, 9781441959997. Citado nas páginas 40, 41 e 42.

SCURO, G. **Automotive industry: Innovation driven by electronics**. 2012. Embedded Computing Design. Disponível em: <http://embedded-computing.com/articles/automotive-industry-innovation-driven-electronics/>. Citado na página 27.

SEEMANN, E.; FRITZ, M.; SCHIELE, B. Towards robust pedestrian detection in crowded image sequences. In: **Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on**. [S.l.: s.n.], 2007. p. 1–8. ISSN 1063-6919. Citado na página 62.

SERMANET, P.; KAVUKCUOGLU, K.; CHINTALA, S.; LECUN, Y. Pedestrian detection with unsupervised multi-stage feature learning. In: **Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition**. Washington, DC, USA: IEEE Computer Society, 2013. (CVPR '13), p. 3626–3633. Citado na página 64.

SHIMA, J. **DSP Trick: Fixed-pt. Atan2 with Self Normalization**. 1999. Disponível em: <http://dspguru.com/dsp/tricks/fixed-point-atan2-with-self-normalization>. Citado na página 114.

SOGA, M.; KATO, T.; OHTA, M.; NINOMIYA, Y. Pedestrian detection with stereo vision. In: **Data Engineering Workshops, 2005. 21st International Conference on**. [S.l.: s.n.], 2005. p. 1200–1200. Citado na página 62.

TASSON, D.; MONTAGNINI, A.; MARZOTTO, R.; FARENZENA, M.; CRISTANI, M. Fpga-based pedestrian detection under strong distortions. In: **Computer Vision and Pattern Recognition Workshops (CVPRW), 2015 IEEE Conference on**. Boston, MA: IEEE, 2015. p. 65–70. Citado na página 75.

TEICH, J. Hardware/software codesign: The past, the present, and predicting the future. **Proceedings of the IEEE**, v. 100, n. Special Centennial Issue, p. 1411–1430, 2012. ISSN 0018-9219. Citado na página 43.

TERASIC. **De2-i 150 Development kit: FPGA system User manual**. [S.l.], 2013. Citado nas páginas 82 e 83.

VALGRIND. **The Valgrind Quick Start Guide**. 2016. Disponível em: <http://valgrind.org/docs/manual/valgrind_manual.pdf>. Citado nas páginas 104 e 105.

VAPNIK, V. N. Estimation of dependences based on empirical data (in russian). In: . [S.l.: s.n.], 1979. Citado na página 64.

VIOLA, P.; JONES, M. Robust real-time face detection. **International Journal of Computer Vision**, v. 57, p. 137–154, 2004. Citado nas páginas 63, 72 e 73.

VIOLA, P.; JONES, M.; SNOW, D. Detecting pedestrians using patterns of motion and appearance. In: **Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on**. [S.l.: s.n.], 2003. p. 734–741 vol.2. Citado na página 62.

WALK, S.; MAJER, N.; SCHINDLER, K.; SCHIELE, B. New features and insights for pedestrian detection. In: **Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on**. [S.l.: s.n.], 2010. p. 1030–1037. ISSN 1063-6919. Citado na página 63.

WANG, X.; HAN, T.; YAN, S. An hog-lbp human detector with partial occlusion handling. In: **Computer Vision, 2009 IEEE 12th International Conference on**. [S.l.: s.n.], 2009. p. 32–39. ISSN 1550-5499. Citado na página 63.

WOJEK, C.; DORKÓ, G.; SCHULZ, A.; SCHIELE, B. Sliding-windows for rapid object class localization: A parallel technique. In: **DAGM-Symposium**. [S.l.: s.n.], 2008. p. 71–81. Citado na página 61.

WOJEK, C.; SCHIELE, B. A performance evaluation of single and multi-feature people detection. In: RIGOLL, G. (Ed.). **Pattern Recognition: 30th DAGM Symposium Munich, Germany, June 10-13, 2008 Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 82–91. Citado na página 63.

WU, B.; NEVATIA, R. Detection of multiple, partially occluded humans in a single image by bayesian combination of edgelet part detectors. In: **Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on**. [S.l.: s.n.], 2005. v. 1, p. 90–97 Vol. 1. ISSN 1550-5499. Citado na página 62.

____. Optimizing discrimination-efficiency tradeoff in integrating heterogeneous local features for object detection. In: **Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on**. [S.l.: s.n.], 2008. p. 1–8. ISSN 1063-6919. Citado nas páginas 62 e 63.

XYLON. **logiPDET Pedestrian Detector**. 2014. Data Sheet. Disponível em: <https://www.xilinx.com/products/intellectual-property/1-2ppois.html>. Citado na página 109.

ZHANG, S.; BAUCKHAGE, C.; CREMERS, A. B. Informed haar-like features improve pedestrian detection. In: **2014 IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2014. p. 947–954. ISSN 1063-6919. Citado nas páginas 63 e 133.

ZHANG, S.; BAUCKHAGE, C.; KLEIN, D. A.; CREMERS, A. B. Exploring human vision driven features for pedestrian detection. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 25, n. 10, p. 1709–1720, Oct 2015. Citado na página 63.

ZHANG, S.; BENENSON, R.; OMRAN, M.; HOSANG, J.; SCHIELE, B. How far are we from solving pedestrian detection? In: **Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2016. Citado na página 30.

ZHANG, S.; BENENSON, R.; SCHIELE, B. Filtered channel features for pedestrian detection. In: **IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2015. Citado nas páginas 63, 64 e 133.

ZHANG, W.; ZELINSKY, G.; SAMARAS, D. Real-time accurate object detection using multiple resolutions. In: **Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on**. [S.l.: s.n.], 2007. p. 1–8. ISSN 1550-5499. Citado na página 61.

ZHANG, Y.; KISELEWICH, S.; BAUSON, W.; HAMMOUD, R. Robust moving object detection at distance in the visible spectrum and beyond using a moving camera. In: **Computer Vision and Pattern Recognition Workshop, 2006. CVPRW '06. Conference on**. [S.l.: s.n.], 2006. p. 131–131. Citado na página 61.

ZHAO, L.; THORPE, C. Stereo- and neural network-based pedestrian detection. In: **Intelligent Transportation Systems, 1999. Proceedings. 1999 IEEE/IEEJ/JSAI International Conference on**. [S.l.: s.n.], 1999. p. 298–303. Citado na página 62.

ZHAO, M. **Advanced Driver Assistant System: Threats, requirements, security solutions**. [S.l.], 2015. Citado na página 29.

ZHU, Q.; ZHU, Q.; AVIDAN, S.; AVIDAN, S.; YEH, M. chen; YEH, M. chen; CHENG, K. ting; CHENG, K. ting. Fast human detection using a cascade of histograms of oriented gradients. In: **In CVPR06**. [S.l.: s.n.], 2006. p. 1491–1498. Citado na página 62.