Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Estudo da aplicação do algoritmo Viola-Jones à detecção de pneus com vistas ao reconhecimento de automóveis

Matheus Bezerra Estrela Rodrigues

Campina Grande, Paraíba, Brasil
Matheus Estrela, Fevereiro 2012

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Estudo da aplicação do algoritmo Viola-Jones à detecção de pneus com vistas ao reconhecimento de automóveis

Matheus Bezerra Estrela Rodrigues

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Redes de Computadores e Sistemas Distribuídos

Elmar Uwe Kurt Melcher
(Orientador)

Campina Grande, Paraíba, Brasil
Matheus Estrela, Fevereiro 2012

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

R648e Rodrigues, Matheus Bezerra Estrela
Estudo da aplicação do algoritmo viola-jones à detecção de pneus com vistas ao reconhecimento de automóveis/ Matheus Bezerra Estrela Rodrigues. - Campina Grande, 2012

57f.: il.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina , Centro de Engenharia Elétrica e Informática.
Orientador: Prof. Dr. Elmar Uwe Kurt Melcher
Referências.

1. Reconhecimento de Pneus. 2. Viola-Jones. 3. Visão Computacional. I. Título.

CDU 004.4 (043)

"ESTUDO DA APLICAÇÃO DO ALGORITMO VIOLA-JONES À DETECÇÃO DE PNEUS COM VISTAS AO ECONHECIMENTO DE AUTOMÓVEIS"

MATHEUS BEZERRA ESTRELA RODRIGUES

DISSERTAÇÃO APROVADA EM 29/02/2012

ELMAR UWE KURT MELCHER, Dr.
Orientador(a)

JOSEANA MACÊDO FECHINE RÉGIS DE ARAÚJO, D.Sc
Examinador(a)

MARCOS RICARDO ALCÂNTARA MORAIS, D.Sc
Examinador(a)

CAMPINA GRANDE - PB

Resumo

Impulsionado pelo crescimento no uso de vigilância eletrônica, essa pesquisa introduz o uso de uma técnica que demonstra eficiência no reconhecimento de faces em imagens, alterando o objeto de busca para pneus de veículos, visando o reconhecimento da presença do veículo na cena. A técnica aplicada para o reconhecimento é o algoritmo Viola-Jones. Essa técnica é dividida em dois momentos: o treinamento e a detecção. Na primeira etapa, vários treinamentos são executados, usando aproximadamente 7000 imagens diferentes. Para a etapa final, um detector de faces foi adaptado para reconhecer pneus, utilizando o treinamento da etapa anterior, e sua eficiência em reconhecer os pneus foi comparável à eficiência do detector de faces que usa treinamento de referência da biblioteca em *software* que é referência nesta área, OpenCV. O detector desenvolvido apresentou taxa de reconhecimento de 77%, quando o reconhecimento de faces obteve 80%. A taxa de falsos negativos também foi próxima, apresentando o detector de pneus 2% e o de faces 1%.

Palavras-chave: reconhecimento de pneus, Viola-Jones, visão computacional

Abstract

Motivated by the growing use of electronic surveillance, this research introduces the use of the Viola-Jones algorithm, which is known to be efficient in recognition of human faces in images, changing the object to be recognized to vehicle tires, aiming to detect vehicles in a scene. This approach divides the process in two steps: training and detection. Training was done using around 7000 different images of vehicles. For the detection step, work was done to adapt a face detector to detect vehicles tires. The tire detector was compared to a face detector that used a reference training for faces from OpenCV library. The tire detector showed 77% efficiency, whereas the face detector showed 80%. False negative numbers also showed similar closeness, as 2% for the tire detector and 1% for the reference face detector.

Keywords: tire detection, Viola-Jones, computer vision

"We will call you Cygnus,
the God of Balance you shall be."

Cygnus X-1 Book II: Hemispheres
Rush

A todos da minha família, que sempre me apoiaram e acreditaram em mim.
Eu não seria *nada* sem eles.
Em especial aos meus avós, que tanto lutaram para que seus filhos pudessem estudar, sempre com muito sacrifício e perseverança. Gostaria de homenagear meus pais pela homenagem aos deles: Seu Bezerra, Dona Francisquinha, Zabé e Seu João.
Eu sempre amei e fui muito amado por vocês.

Agradecimentos

Muitas pessoas me apoiaram e acreditaram em mim durante toda esta jornada. Acredito que todos sabem de sua importância, então se esquecer de alguém (eu nunca fui esquecido ...), peço desculpas desde já, não foi por mal.

Antes de falar de quem que seja, agradeço ao pessoal do LAD por acreditar em mim, e condenso este agradecimento nos professores Elmar e Joseana, que mesmo não sendo orientadora me ajudou muito com comentários e dicas sempre construtivas em conjunto com Elmar. Ainda um obrigado especial a Thiago, que também forneceu muita informação sobre o Viola-Jones que sabia de antemão, e pesquisas de códigos de exemplo, e Jorgeluis sempre disponível para dúvidas de Verilog. Muito obrigado a todos !

A todos da Copin, Aninha, Vera e Rebeka, que sempre ajudaram, avisaram, torceram, reclamaram, cobraram e merecem demais este agradecimento. Vocês são 10!

Gostaria também de agradecer ao pessoal do TRE-PB pela cooperação desde o início, e desta forma agradeço a todos que passaram pela Chefia da SEINF neste tempo. Todos foram de grande apoio e muitas das horas que tive para este trabalho consegui graças a vocês. E sempre contei com o apoio, mesmo que em frases de ânimo, dos demais colegas. Obrigado vocês, espero que todos tenham isto que tive.

Aos amigos, que foram pacientes em me esperar chegar atrasado e por vezes nem ir aos encontros marcados, quando estudando. Gostaria de agradecer em especial a André pelas dicas muito úteis durante a pesquisa, a revisão do texto e por vezes companhia em viagens à Campina Grande. E também a Urubatã, por todas as caronas e viagens à Campina quando eu não podia dirigir e o Profeta (e Niedja), que sempre esteve disponível para resolver problemas de ordem burocrática na UFCG, além de sempre deixar sua casa disponível como apoio, e fazer sempre que eu me sentisse em casa.

Meus pais e toda a família. Sempre acreditaram em mim e deram apoio desde os tempos da graduação. Não sei como agradecer a vocês, se eu vivesse 1000 anos eu não conseguiria retribuir nem metade do apoio e sacrifícios que fizeram por mim. Amo vocês todos.

E, finalmente, gostaria de agradecer a minhas companheiras que dividem suas vidas comigo agora. Camila acompanhou todos as fases deste mestrado, sofreu e comemorou comigo todas (nem sempre com sorriso!). Ela foi muitas vezes (dormindo) a

Campina, e sempre que podia (e eu aperreava) estava junto tentando me acalmar quando algo dava errado. Obrigado bichinha ! Te adoro Camila. E, nesta reta final, você e Preta Maria me proporcionaram uma tranquilidade e paz imprescindíveis para poder concluir esta pesquisa. Você sabe que vocês são os motivos de minha tranquilidade e paz nestes meses, e agradeço demais às duas por isso.

Lista de símbolos e abreviaturas

ARHTMW	*Adaptive Hough Transform with Moving Window*
BVM	*Brazil-IP Verification Methodology*
DSP	*Digital Signal Processor*
FPGA	*Field Programmable Gate Array*
HT	*Hough Transform*
IP	*Intellectual Property*
LAD	*Laboratório de Arquiteturas Dedicadas*
LBP	*Local Binary Patterns*
RAM	*Random Access Memory*
RHT	*Randomized Hough Transform*
SMS	*Short Message Service*
SoC	*System on a Chip*
XML	*Extensible Markup Language*

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Fórmulas

1 Considerações Iniciais

1.1 Introdução

Processar vídeo é uma operação complexa e custosa do ponto de vista de recursos computacionais, por conta da grande quantidade de informação nele contida (BENKRID et al., 2001; GUO; ZHANG; ZHANG, 2006; LAM; EN, 1996; RAD; FAEZ; QARAGOZLOU, 2003; SHANG et al., 2009; SHIE; JEN; CHEN, 2006; WU, 2008; XU; OJA; KULTANEN, 1990). Com o avanço da eletrônica e da computação, essa tecnologia se torna cada vez mais disponível e com preço reduzido (SCHWAMBACH, 2009). Disponível, porque as tecnologias de fabricação de circuitos integrados, estão, a cada dia, evoluindo e proporcionando o desenvolvimento de dispositivos menores em tamanho, mais econômicos em relação ao consumo de energia e com desempenho em constante evolução. Com preço reduzido, porque quanto melhor em desempenho, menor em tamanho e menor o consumo de energia, há cada vez mais aplicações possíveis para estes dispositivos embarcados, e, assim, mais unidades serão produzidas, e com a maior escala de produção, torna-se menor o custo por unidade.

Atualmente, a capacidade de processar vídeo está presente em uma gama de dispositivos. Das ilhas de produção de vídeos profissionais presentes em estúdios de filmagem e emissoras de TV, às câmeras de fotografia digital, que muito expandiram esse mercado nos últimos 10 anos, chegando a miniaturização em aparelhos de telefonia móvel celular. Esses últimos, têm apresentado grande crescimento econômico ultimamente, sendo o mercado de telefonia móvel um grande consumidor de novas tecnologias, principalmente no campo da multimídia. Outro mercado que também mostrou grande crescimento foi o de vigilância eletrônica. É comum ver repartições, condomínios e até cruzamentos, nos quais se faz uso desta tecnologia, seja para simples monitoramento, ou para registro em vídeo, caso haja necessidade de posterior auditoria (NETLAN TECNOLOGIA DE SISTEMAS LTDA., 2010; SMART UNION, 2010).

A capacidade de processamento de vídeo em algumas aplicações é utilizada após a aquisição dos dados em um evento que não necessita de transmissão ao vivo. Neste

tipo de aplicação, não há um limite rígido de tempo para que o tratamento da informação aconteça. Em muitos cenários da área de segurança, existe um intervalo de tempo máximo em que o processamento dos dados colhidos deve ser concluído. Esse tipo de processamento é denominado processamento em tempo real (OLIVEIRA, 2009; WIKIPEDIA, 2010; ZIFF DAVIS PUBLISHING HOLDINGS INC, 2010).

O mercado de processamento de vídeo em tempo real também obteve um crescimento notável nos últimos anos, muito impulsionado pelas vendas de produtos eletrônicos como as já citadas câmeras de fotografia digital, câmeras de filmagem domésticas e aparelhos de telefonia móvel, bem como no âmbito da vigilância, dentre outras. Fazendo uso de processamento de vídeo em tempo real, câmeras de fotografia detectam a presença de pessoas ou sorrisos em uma foto, e, assim, disparam a fotografia automaticamente (SONY CORP., 2010). No campo da vigilância, já existe software que detecta se há movimento, e caso não haja, interrompe a gravação, para assim, poupar a mídia de armazenamento (DESKSHARE INCORPORATED, 2010; LAVRSEN, 2010). Porém, muito pode ser feito ainda nesta área, como reconhecer um rosto ou algum outro tipo de objeto, e deste estímulo, disparar um evento (como avisar ao operador ou mandar uma mensagem eletrônica, SMS, etc). Nesse caso, a máquina realizará algo com precisão e velocidade, que pode ser inviável ou até impossível para um ser humano (como, por exemplo, observar 64 câmeras de vídeo procurando algum objeto ou alguém).

Neste contexto de processamento de vídeo em tempo real, mais direcionado ao âmbito da vigilância e/ou monitoramento, estará centrado o trabalho ora desenvolvido. O objetivo desse estudo é propor um modelo de solução para o problema de detectar a presença de veículo em uma imagem, pela detecção de seus pneus na cena.

Esse campo de pesquisa, o reconhecimento de formas em uma imagem, vem sendo estudado há muito tempo (DUDA; HART, 1972; HOUGH, 1959). Mas, apesar de não haver ainda uma solução definitiva para todos os casos, abordagens baseadas na transformada de Hough foram muito pesquisadas (GUO; ZHANG; ZHANG, 2006; LAM; EN, 1996; MCLAUGHLIN; ALDER, 1998; RAD; FAEZ; QARAGOZLOU, 2003; SHANG et al., 2009; SHIE; JEN; CHEN, 2006; TORII; IMIYA, 2007; WU, 2008; XU; OJA; KULTANEN, 1990). Um estudo mais detalhado destes trabalhos será apresentado no Capítulo 2.

No tocante ao reconhecimento de círculos em imagens, existe uma técnica

estudada desde os anos 70, a Transformada de Hough (DUDA; HART, 1972), mas que necessita de muitos recursos de *hardware*, o que poderia torná-la ineficiente e ineficaz para o problema em questão (GUO; ZHANG; ZHANG, 2006; LAM; EN, 1996; SHANG et al., 2009; SHIE; JEN; CHEN, 2006). Por este motivo, algumas alternativas e/ou modificações foram sugeridas para alterá-la, tornando-a mais econômica em relação ao tempo gasto para processamento da informação (desempenho), e à quantidade de recursos necessários para este processamento (GUO; ZHANG; ZHANG, 2006; SHANG et al., 2009; SHIE; JEN; CHEN, 2006; SCHWAMBACH, 2009). Um estudo mais detalhado destes trabalhos será também apresentado no Capítulo 2.

1.2 Objetivos da Pesquisa

O presente trabalho tem como meta desenvolver uma solução capaz de detectar pneus de veículos, a partir de imagens capturadas por câmeras de vigilância comuns (câmeras usadas comercialmente e não dispondo de recurso especial). As imagens podem ter resolução variando de 320 x 240 pixels até 1920 x 1080 pixels.

O detector de pneus terá como entrada a saída de um detector de fundo heterogêneo (OLIVEIRA, 2009), que filtra os pixels pertencentes ao fundo fixo da cena, passando somente pixels de objetos novos na cena.

Este trabalho divide-se em duas etapas bem definidas, apresentadas na Figura 1. A primeira, inicia-se com a coleta de imagens, que contém e que não contém o objeto desejado. Há um tratamento das imagens obtidas, onde são demarcadas as áreas em que os objetos se encontram. No passo final da etapa, o treinamento propriamente dito é executado pela aplicação do algoritmo *AdaBoost* (VIOLA; JONES, 2001), que compõe a técnica de reconhecimento de padrões da solução de Viola-Jones (VIOLA; JONES, 2001). No final do treinamento, um arquivo de descrição dos padrões necessários para a detecção é gerado.

A etapa seguinte executa a detecção, na qual o detector Viola-Jones é implementado. Na execução da etapa de detecção, são usados como entrada a imagem que será examinada e o arquivo gerado na etapa anterior. De posse desses dois dados, o

detector realiza um redimensionamento na imagem de entrada (a imagem e o detector necessitam ter a mesma resolução para esse projeto) e então aplica o algoritmo Viola-Jones, para decidir se a imagem de entrada contém ou não o objeto desejado. Os processos de treinamento e de detecção serão detalhados no Capítulo 3.

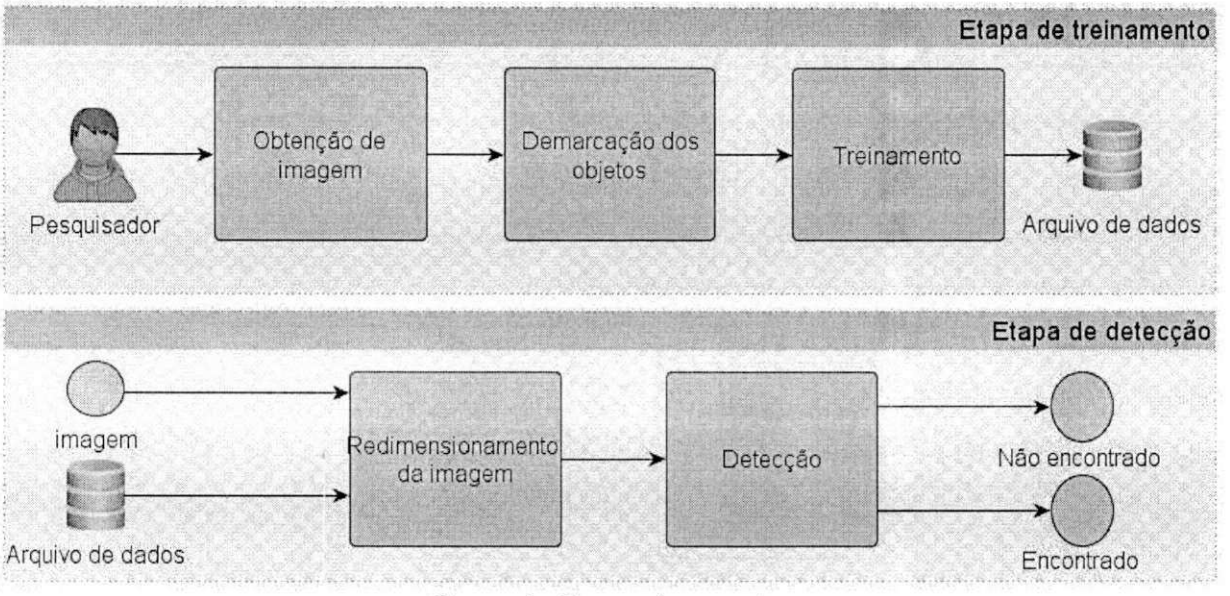

Figura 1: Etapas da pesquisa

1.3 Contribuições do Trabalho

O resultado desta pesquisa visa agregar o poder de reconhecimento de pneus em dispositivos embarcados, sem a necessidade de um computador de mesa. Este módulo poderá ser usado em:

- Sistemas de segurança: a presença de veículo na entrada de um condomínio acionaria o sistema, que reage ao tomar uma decisão automática (pré-configurada) ou alertar algum funcionário;
- Sistemas de monitoramento de tráfego: uma câmera detecta a passagem de carros em sinal fechado, e dispara o sistema de fotografia que registra a placa do infrator. Não há a necessidade, então, de instalação de sensores no asfalto.

Como a solução desenvolvida poderá ser implementada em um SoC (*System on a*

Chip) de uso dedicado, ela poderá ser produzida usando tecnologias de fabricação voltadas ao baixo consumo de energia, ao baixo custo de produção ou ambos. Neste caso, a solução poderá ser integrada em outros sistemas, para prover a funcionalidade proposta.

Outra contribuição do trabalho, é a aplicação do resultado da pesquisa de Viola-Jones usando objetos diferentes de faces, campo no qual a técnica é bem aplicada e apresenta resultados positivos (SCHWAMBACH, 2009; GAO; LU; 2008; CHO *et. al.*, 2009). Não foi encontrada referência a trabalho acadêmico publicado usando esta técnica fora do campo de reconhecimento de faces com fundo heterogêneo.

1.4 Metodologia

Após o estudo sobre o estado da arte de reconhecimento de objetos circulares em imagens, duas soluções foram destacadas para serem usadas como técnicas candidatas a serem utilizadas no detector. Uma, baseada na Transformada de Hough, Adaptive Random Hough Transform using Moving Window (ARHTMW) desenvolvida por GUO *et al*, e a abordagem apresentada por SCHWAMBACH (2009), que aplica o trabalho desenvolvido por Paul Viola e Michael Jones (2001). Este estudo será apresentado no Capítulo 2.

Após análise dos algoritmos pesquisados, houve a constatação de uma constante sobre a Transformada de Hough: alto uso de memória. Mesmo em sua versão mais econômica, a quantidade de possíveis círculos quando salvos em memória, em tempo de execução, supera a necessidade de uma imagem integral da solução de Viola e Jones. Acessos a memória RAM (memória de acesso aleatório, do inglês *Random Access Memory*) são grandes consumidores de energia, mais ainda, em se tratando de memória de grande capacidade, fora do chip de processamento (IAR SYSTEMS, 2012; EIBLMAIER; MAO; WANG, 2009; DINIZ et al., 2007). Como em sistemas computacionais atualmente o maior fator de viabilidade e custo é o consumo de energia (EIBLMAIER; MAO; WANG, 2009; DINIZ et al., 2007), optou-se por não usar algoritmos que trabalham com a Transformada de Hough.

A Transformada de Hough apresenta demanda por processamento em tempo de detecção maior que a solução proposta por Viola-Jones. A solução proposta por Viola e Jones concentra a parte mais dependente de processamento em estágio anterior ao momento de detecção, executando somente comparações numéricas em tempo de detecção. Não há necessidade de salvar mapas para testes posteriores. Na Transformada de Hough, o processo de força bruta acontece no momento de reconhecimento do objeto, demandando mais do processador do dispositivo. O estudo de Schwambach (2009) demonstra esta maior eficiência em dispositivos móveis.

Como fonte de informações e exemplos, foi utilizada a biblioteca OpenCV (OPENCVWIKI, 2012). Aplicativos de exemplos demonstraram o funcionamento do algoritmo, e aplicativos de apoio proporcionaram a produção de treinamentos próprios dedicados ao objeto de estudo desta pesquisa. Duas bases de dados de imagens do objeto foram construídas. Antes de avançar no trabalho, os treinamentos passavam por testes, usando ferramentas da própria OpenCV.

Após verificação do funcionamento do código disponibilizado por Chang (2012[1]), o código foi alterado para reconhecer pneus. Em sua versão original, somente é possível reconhecer faces. Ferramentas de apoio foram desenvolvidas para auxiliar o processo de execução do detector.

Em seguida, um conjunto de 100 imagens de pneus foi analisado, como forma de teste do produto final da pesquisa. Como meio de comparação, outras 100 imagens que não apresentam o objeto desejado foram também testadas.

Para serem usadas como parâmetro de comparação, também 100 imagens de faces foram submetidas ao detector original. E, novamente, as 100 imagens que não continham pneus, nem faces, foram apresentadas a este detector. Os resultados encontram-se expostos no Capítulo 4.

1.5 Estrutura do Trabalho

A estrutura deste documento é detalhada a seguir.

1 O ano da referência não indica o ano da utilização do trabalho.

No capítulo 2 é apresentado um estudo sobre algoritmos de detecção de formas circulares em imagens. No mesmo capítulo, são apresentados trabalhos relacionados ao algoritmo implementado nesse projeto.

No Capítulo 3 é feita a apresentação do modelo proposto nesse trabalho, detalhando-se as funcionalidades de cada parte, suas entradas e saídas. Também é detalhado o algoritmo Viola-Jones, mencionando suas estruturas de dados, fluxo de execução e arquivos gerados e necessários para seu funcionamento. O processo de execução de um treinamento é detalhado neste capítulo.

No Capítulo 4 são apresentados os resultados obtidos, detalhados os treinamentos e resultados das detecções realizadas para avaliar o detector desenvolvido.

No Capítulo 5 são apresentadas as considerações finais sobre o estudo, sua execução, e sugestões para trabalhos futuros.

Há um Apêndice no final do documento em que são apresentados os *scripts* utilizados para os treinamentos. Há, também, outro Apêndice que apresenta a biblioteca OpenCV, mencionada como referência, e as ferramentas nela presentes que foram utilizadas, juntamente com as ferramentas desenvolvidas para esta pesquisa. Também são detalhadas as etapas para a realização de um treinamento. Há também a apresentação do código do detector de Daniel Chang (2012), utilizado neste estudo.

2 Revisão da Literatura

Um algoritmo clássico de busca por figuras geométricas em imagens é a Transformada de Hough (HOUGH, 1959). Quando da sua aparição, em 1959, e posterior patente por Paul Hough em 1962, tratava exclusivamente de linhas em uma imagem. Somente em 1972 que Richard Duda e Peter Hart publicaram uma versão mais genérica, que possibilitaria a identificação de outras formas, como os objetos circulares aqui tratados (DUDA; HART, 1972).

A Transformada de Hough, para identificar círculos em uma imagem, procura por conjuntos de três pontos (x,y,r), em que x e y são o centro deste círculo e r é seu raio, de modo que os círculos definidos por cada conjunto destes se interceptem em um ponto comum. Este ponto comum é o centro do círculo ora procurado.

Richard Duda mostrou que, para um círculo de raio *r* e centro (a,b), as equações 1 e 2 (DUDA; HART, 1972):

$$(x = a + r * \cos(\square)) \quad (1)$$

$$(y = b + r * sen(\square)) \quad (2)$$

o definem completamente, quando o ângulo □ variar de 0 até 360° (estudado em (DUDA; HART, 1972), mais detalhadamente explicado em (RHODY, 2010)).

Assim, para cada ponto analisado, o algoritmo testa se há para algum raio *r* outro conjunto de pontos (z,w,r) cujo círculo formado pelo centro (z,w) e raio *r* tenha um ponto em comum (interseção) com o ponto testado. Caso haja, este ponto é adicionado a uma lista (*array*), cujos círculos também têm este mesmo ponto em comum. Quando finalizada a varredura da imagem trabalhada, a lista é analisada e círculos em potencial que tenham mais conjuntos (x,y,r) que um determinado limiar são considerados círculos reais, sendo apresentados. Este processo está demonstrado na Figura 2.

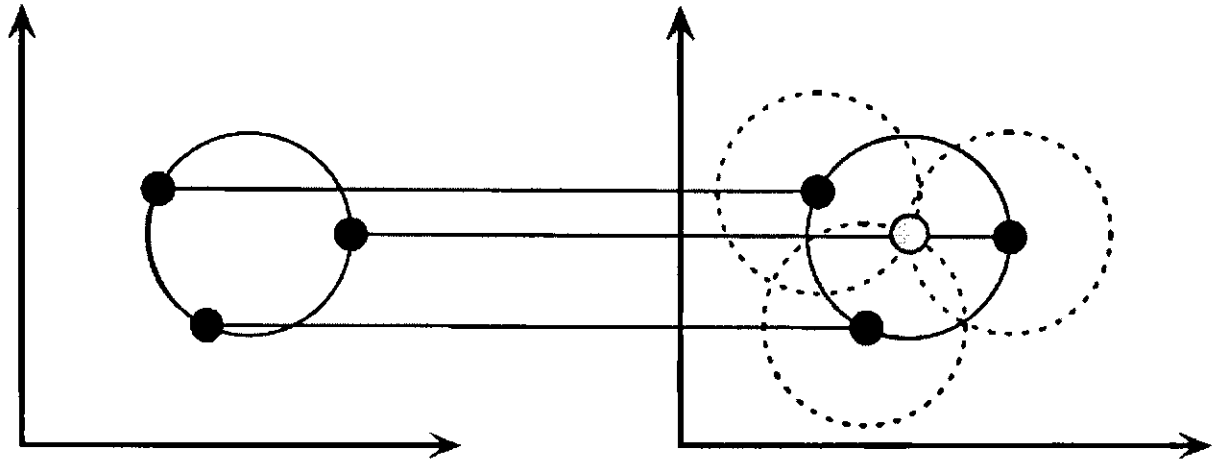

Figura 2: Transformada de Hough
Cada ponto do círculo cria outro que intercepta os círculos criados pelos outros pontos dos originais. Fonte: "Lecture 10: Hough Circle Transform, Harvey Rhody"

Mas, esta transformada, tem suas limitações e problemas (SHIE; JEN; CHEN, 2006). Ela demanda muito em memória e em capacidade de processamento, e tem problemas de escalabilidade (GUO; ZHANG; ZHANG, 2006; LAM; EN, 1996; RAD; FAEZ; QARAGOZLOU, 2003; SHANG et al., 2009; SHIE; JEN; CHEN, 2006; WIKIPEDIA, 2010). Além destes dois problemas, o algoritmo é sensível ao ruído na imagem, perdendo sua precisão no caso em que o ruído esteja demasiadamente presente (GUO; ZHANG; ZHANG, 2006; MCLAUGHLIN; ALDER, 1998). Com o intuito de mitigar estes problemas, sugestões de mudanças na transformada Hough e soluções alternativas foram propostas.

Em Xu *et. al.,* dois pontos são escolhidos de forma aleatória da lista de pontos que resultam da detecção de contornos na imagem a ser processada (*edge detector* – detector de bordas). Esta técnica é chamada de *Random Hough Transform* (RHT) (XU; OJA; KULTANEN, 1990) (XU; OJA, 1993, 2009). Nesta abordagem, para cada objeto encontrado, são eliminados os pontos referentes a este da lista resultante do preprocessamento acima referido. Desta forma, o pico de utilização de memória é alcançado no momento em que o objeto é encontrado. Após este momento, estes valores são retirados da memória e não acumulados como seriam usando HT tradicional. Há grande uso de memória durante toda a execução, mas, seu uso é particionado em fragmentos menores. Desta forma, o processamento é feito usando uma quantidade inferior deste recurso quando comparado ao HT (XU; OJA; KULTANEN, 1990). A

necessidade de processamento também é reduzida, pelo fato de HT necessitar calcular toda a extensão da curva detectada em cada passo, dado que RHT somente trabalha com o ponto encontrado (XU; OJA; KULTANEN, 1990). Apesar de resolver o problema de detectar círculos, este algoritmo não é exclusivo para este fim, RHT pode ser usada também para detecção de outras formas geométricas, do mesmo modo que a HT não é específica para círculos. É também mostrado que RHT necessita de menos operações para obter o mesmo resultado, com dois experimentos mostrando RHT aproximadamente 100 e 20 vezes mais rápido, respectivamente.

GUO *et. al.* (2006) usa o RHT como ponto de partida, em uma abordagem chamada Adaptive Random Hough Transform using Moving Window (ARHTMW) (GUO; ZHANG; ZHANG, 2006). Este método é específico para detecção de círculos e necessita de informação *a priori* do problema, é necessário saber o valor máximo para o raio. A solução proposta neste artigo utiliza uma janela deslizante, cujo tamanho é uma função do raio máximo, e aplica o RHT para cada janela. O autor justifica esta subdivisão pelo fato de o ruído ter menor influência no RHT quando a imagem analisada é menor. Esta solução compartilha as características do RHT, mas por aplicar por várias vezes o mesmo procedimento poderia ser paralelizável via *hardware*, teoricamente, sem muita complexidade. Os testes mostravam um ganho de desempenho de 5 vezes em média em relação ao RHT puro, chegando a 7 vezes no pior caso para os dois algoritmos.

Porém, nem todas as soluções são baseadas em HT, como SHANG *et al* (SHANG et al., 2009). Neste trabalho, o círculo é encontrado achando-se um triângulo retângulo inscrito no círculo procurado. Um vez achado o triângulo, o centro do círculo é o ponto que corresponde a metade da sua hipotenusa, e o raio é metade do tamanho da hipotenusa. Os passos para a detecção do(s) triângulo(s) são simples, mas há um tratamento mais complexo após este passo, para confirmar o círculo encontrado. O artigo também propõe uma estrutura de dados para armazenamento temporário dos dados, em muito semelhante a do RHT, apesar de mais complexa. Há a possibilidade de fazer parte dos passos em paralelo, apesar de ser menos simples que aplicar tal técnica ao RHT. Também como RHT, há pré-processamento da imagem e transformação desta em pontos de contorno. Em experimentos feitos em um Pentium 1.73GHz e imagens de 256 x 256 pixels (com e sem ruídos), esta solução é, em média, 2,5 vezes mais rápida que RHT.

Rad *et al* também propõe uma solução não baseada em HT que demonstra ser mais eficiente que a HT clássica e a versão melhorada (RAD; FAEZ; QARAGOZLOU, 2003). Ele usa pares de vetores do gradiente do contraste de cor que há entre os pixels do círculo e dos que não fazem parte do círculo. Por vezes, usa preprocessamento para tratar a imagem, transformando-a em uma imagem binária em preto e branco com base nos limiares dos objetos. Nos testes feitos, usa-se como base de dados imagens de exames médicos, e o próprio autor indica que pode haver a necessidade de ajustes, de acordo com o tipo de imagem a ser usada. Usando um Pentium IV 1.8GHz sua solução foi, em média, 650 vezes mais rápida que HT em imagens de 256 x 256 pixels. Este ganho passou para aproximadamente 1000 vezes, quando as imagens eram de 512 x 512 pixels. Este algoritmo tem outro ponto positivo de reconhecer elipses (demonstrado no exemplo ao final do texto do artigo), mas necessita de passos de pré-processamento que podem ser diferentes, variando com alguns aspectos da imagem a ser processada. É sugerida uma técnica de armazenamento de dados similar a das soluções com HT.

Outro método não baseado em HT para reconhecimento de formas em imagens é o UpWrite, de McLaughlin e Alder (MCLAUGHLIN; ALDER, 1998). Esta solução, que precisa de um preprocessamento por só tratar imagens em tons de cinza, segmenta a imagem em pedaços menores, dando um valor característico para cada segmento. Em um segundo passo, agrupa estes segmentos usando como parâmetro seus valores e para finalizar, computa o momento Zernike de formas geométricas para saber de qual delas se trata a imagem em questão. No próprio artigo, já é declarado que nem sempre este método é melhor que alguma derivação do HT, superando a Transformada de Hough no caso de objetos mais complexos, como elipses. Outra vantagem deste método é sua versatilidade em reconhecer tanto linhas, círculos e elipses em uma única execução. Os baseados em HT precisam de uma execução para cada tipo de forma geométrica, e este, baseado em UpWrite, pode ser usado para detectar mais de uma forma geométrica por execução. Estas duas características não são cruciais para o estudo em questão.

Outra abordagem que pode ser aplicada ao reconhecimento de pneus em imagens, foi desenvolvida por Paul Viola e Michael Jones (VIOLA; JONES, 2001), que consiste na execução de um treinamento, antes da detecção, usando o algoritmo *AdaBoost*, e usa os valores resultantes do treinamento para comparações e cálculos simples (as quatro operações fundamentais) no momento da detecção. Com esta forma de detecção, o

dispositivo utilizado para a detecção não necessita ter grande capacidade de processamento. Apesar de muito usado para reconhecimento de faces, este algoritmo pode ser usado para outros tipos de objetos. Alguns estudos sobre o algoritmo, relevantes para o trabalho em questão pelo contexto, são apresentados a seguir, e o detalhamento do algoritmo estará apresentado no Capítulo 3.

Jensen (2008) fez um estudo do algoritmo seguido de implementação. Neste estudo foi utilizado majoritariamente o Matlab, havendo partes desenvolvidas em C. O autor desenvolve seus próprios treinamentos, etapa que demanda mais tempo que ele previra. A maior quantidade de tempo gasto nessa etapa implica em ter somente 16 estágios implementados no projeto. Por este motivo, ele não faz comparação de desempenho, mas somente compara taxa de reconhecimento. Apesar de não ter esta funcionalidade, faz referência à sugestão de Viola e Jones, de executar ao menos três classificadores em paralelo, e somente indicar existência de objeto se ao menos dois destes retornarem positivo. Ainda fez um estudo sobre o comportamento das características do primeiro estágio, segundo ele em número de duas, e usa esta informação para escolher a base de dados a ser usada em seu estudo. Somente com algumas referências, Jensen (2008) não apresenta detalhes da abordagem feita em seu trabalho para o tratamento de escalas – segundo Viola e Jones, o detector pode ser usado em janelas maiores que a de treinamento, necessitando ajuste no modo como os dados da janela são lidos (mais detalhes no capítulo 3). Neste trabalho, a escala é tratada pelo detector e não na imagem diretamente.

SCHWAMBACH (2009) trabalha a implementação do Viola e Jones em um processador já existente, estendendo sua funcionalidade com instruções específicas. O autor apresenta ganhos da ordem de 167 vezes, em que, no pior caso, a imagem leva 250 milissegundos para ser processada. Para alcançar este ganho, foram aplicadas otimizações na execução do algoritmo, visando reduzir o número de características processadas e evitar cálculos com dados que são custosos para o processador em questão. Foram implementadas otimizações de divisões e no cálculo de raiz quadrada, máscara de tonalidade de pele e detecção em mais de uma execução – em seu caso, duas. Em uma segunda etapa da pesquisa, foram desenvolvidas otimizações na execução do algoritmo Viola-Jones para o processador em questão, por meio de novas instruções a serem implementadas. O tratamento de escalas também foi abordado,

redimensionando as imagens e não o detector. Segundo o autor, esta forma de tratamento era a mais indicada para o contexto do projeto.

Cho *et. al.* (2009) propuseram uma arquitetura em FPGA para detectar faces, usando a solução proposta por Viola e Jones. Sua implementação abrange desde a captura até a apresentação da imagem final em dispositivo de saída. Apresenta, como pontos fortes, a completude do trabalho (que define todo a arquitetura do detector, seus blocos e conexões) e a eficiência do processador de imagens integrais. Por limites de recursos e visando melhor desempenho do sistema final, é escolhido tratar escalas redimensionando as imagens e não o detector. Por consequência de sua completude, é um projeto complexo e composto por diversas partes – e subpartes – que precisam estar sincronizadas entre si. Este trabalho é a base do trabalho de Daniel Chang (CHANG, 2012), que por sua vez foi tomado como fonte para este projeto de pesquisa. Novamente, não é detalhado o tratamento de escalas do projeto, somente especifica que tem um módulo para tratar este aspecto e é usada escala nas imagens e não no detector.

Também citado por Cho *et. al.* (2009), Gao e Lu (2008) desenvolveram uma solução híbrida, usando um Computador Pessoal (PC) e um *Digital Signal* Processor (DSP) implementado no FPGA. Neste projeto, padronizou-se o número de características por estágio em 16. O estudo foi motivado pelo uso de solução de hardware dedicado (e reconfigurável) para acelerar as partes do processamento que em software demandariam mais da CPU. Segundo os autores, que dividiram este processamento em três partes (pré-processamento, classificador e pós-processamento), a parte correspondente à detecção (executada no classificador) consome 95% do tempo total e é nesta que é mais eficiente o uso do processamento dedicado e especializado. No trabalho, houve também a implementação de um classificador totalmente paralelo, em que todos os 16 estágios são executados simultaneamente. Esta abordagem proporcionou mais ganho em desempenho, por ser implementada em *hardware* em que todos os testes podem executar em paralelo. Mas, para o *software* não há ganho, pelo fato dos primeiros estágios descartarem, sem grande esforço, a maior parte das áreas sem o objeto desejado, chegando a marca de 90% (GAO; LU; 2008). O tratamento de escala não é detalhado, e é mais um exemplo de uso do escala nas imagens, não usando desta capacidade do detector.

Apesar de muitos artigos abordarem Viola-Jones e detecção de faces, a exemplo dos citados acima, nem todas as fontes/referências destas pesquisas servem para o estudo em questão. É conhecido, que este algoritmo, não se aplica somente para detectar faces (VIOLA; JONES, 2001), como se vê neste estudo. Conforme afirmam os autores (traduzido do original):

> "Este artigo apresenta novos algoritmos e ideias para construir um *framework* para detecção rápida e robusta de objetos. Este arcabouço é demonstrado na, e em parte motivado por, tarefa de detectar faces." (VIOLA; JONES, 2001)

Ainda, no mesmo trabalho (traduzido do original):

> "Nós apresentamos uma abordagem para detecção de objetos que minimiza o tempo de processamento enquanto alcança alta precisão na detecção. A abordagem foi usada para construir um sistema detector de faces que é aproximadamente 15 vezes mais rápido que a abordagem anterior." (VIOLA; JONES, 2001)

Há registros de trabalhos, onde houve a tentativa do uso desse algoritmo em objetos diferentes de faces, em citações constantes da lista de discussão da biblioteca OpenCV (ALEXANDRENARTEN, 2010; RBOUGHA, 2009). Há o trabalho de graduação de Evans, Bennett e Sombra (2012[2]), que trabalhava com detecção de folhas de árvores. No treinamento, foram usadas 200 imagens positivas (contendo uma folha) e 7869 imagens negativas (não contendo folha). As imagens positivas, representavam, cada uma, uma única folha em cima de um fundo branco. Na detecção, foram usados dois conjuntos de imagens: Um conjunto com imagens de folhas únicas em fundo branco, e outro conjunto com folhas em fundos heterogêneos. Foram obtidos resultados bons para o primeiro conjunto, mas, os resultados para imagens com fundos heterogêneos foram considerados ruins pelos autores do trabalho.

2.1 Considerações Gerais

Várias soluções para o reconhecimento de objetos circulares em uma imagem foram apresentadas nestes anos, desde o início do estudo em processamento de imagens. Em uma sucinta análise sobre o modo como essas soluções poderiam ser

2 A referência não identifica a data quando este trabalho foi feito nem quando ele foi publicado.

utilizadas no estudo em questão, é observada uma divisão em dois grupos. No primeiro, que usa processamento baseado na intensidade dos pixels, se destacam os formados pela Transformada de Hough e todas as suas derivações. O segundo grupo, consiste somente na solução de Viola e Jones, que trabalha características apresentadas pela diferença de contraste na imagem. A solução adotada baseia-se no algoritmo do segundo grupo, que apresenta melhor desempenho para o ambiente do contexto da pesquisa (SCHWAMBACH, 2009; CHO *et. al.*, 2009; GAO; LU, 2008).

3 Descrição do Modelo

Na Figura 3 é apresentado o diagrama em blocos do sistema desenvolvido.

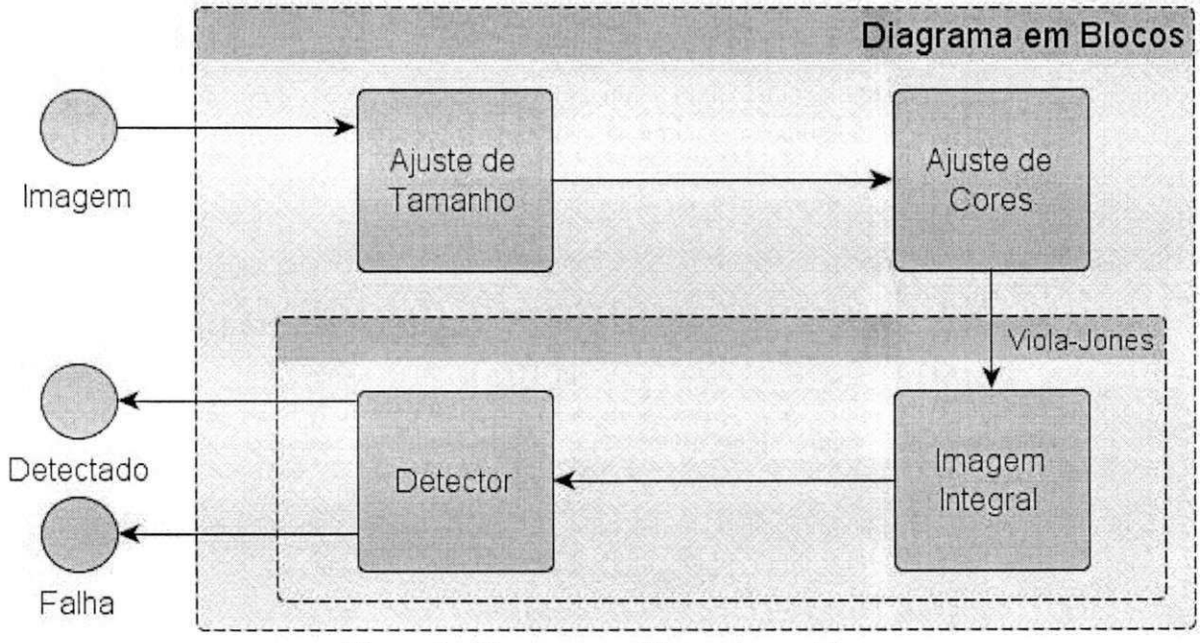

Figura 3: Diagrama em blocos do modelo

O modelo sugerido é composto por 4 blocos: Ajuste de tamanho, Ajuste de Cores, Imagem Integral e Detector. Detalhes de cada bloco serão apresentados a seguir.

O bloco de ajuste de tamanho atua no redimensionamento da imagem de entrada, para que a imagem seja da mesma dimensão esperada pelo detector. A imagem é esperada com as mesmas dimensões da janela de treinamento usada para o classificador usado no projeto. O detector de pneus utilizou janela de 56 x 56 pixels. O ajuste de tamanho da imagem de entrada é necessário pelo fato de não haver implementação do tratamento de escalas. A implementação do tratamento de escalas do algoritmo não era parte do escopo deste projeto, e, não afetou a execução da detecção.

Recebendo a imagem na dimensão esperada, o bloco de Ajuste de Cores transforma a imagem para o espaço de cores YUV. Para o restante do processamento,

somente é usado o componente de luminância, Y, da imagem. Como as características *Haar* não atuam sobre diferenças na tonalidade, esta componente é suficiente para o processamento e as demais não contribuiriam com informações, somente aumentariam a complexidade do processamento. Comportamento similar é visto nos trabalhos de Schwambach (2009), Jensen (2008) e na biblioteca OpenCV (OPENCVWIKI, 2012). Neste bloco, ocorre a extração desta informação e descarte das demais. A saída para o bloco a seguir, consiste em uma matriz de mesma dimensão da imagem redimensionada, em que cada pixel corresponde ao valor do componente Y da imagem redimensionada.

Os dois blocos restantes fazem parte da técnica de reconhecimento de padrões chamada Viola-Jones (VIOLA; JONES, 2001). O bloco Imagem Integral recebe a saída do bloco de Ajuste de Cores, e computa a imagem integral correspondente. Essa informação é repassada para o detector, que executa o reconhecimento de padrões definidos pela técnica Viola-Jones (VIOLA; JONES, 2001) e então retorna a falha ou o sucesso na detecção do objeto.

Pela maior complexidade do subsistema que processa o algoritmo Viola-Jones (VIOLA; JONES, 2001), o detalhamento destes blocos será apresentado em mais detalhes.

3.1 O Algoritmo Viola-Jones

O algoritmo Viola-Jones foi desenvolvido pelos pesquisadores Paul Viola e Michael Jones em 2001 (VIOLA; JONES, 2001). A técnica de reconhecimento de padrões, desenvolvida pelos dois pesquisadores, consiste em encontrar pequenos pedaços da imagem, que apresentem diferença de contraste constante (ou de pequena variação), e que possam ser usados em tempo de detecção, de forma ordenada e bem definida, para reconhecer o objeto. Estes pedaços, denominados pelos autores de características, são encontrados durante o processo de treinamento, através do uso da força bruta. A etapa de treinamento é o momento da técnica que demanda mais recursos computacionais, deixando o momento de detecção mais simples. Mais detalhes sobre esta técnica são apresentados neste Capítulo.

3.1.1 Cacarterísticas Haar – (Haar-like features)

O primeiro conceito a ser introduzido é de características *Haar.* Essa parte do trabalho de Viola e Jones foi baseada no resultado da pesquisa de Papageorgiou, Oren e Poggio (1998). No estudo, Papageorgiou *et. al.* sugerem não usar diretamente a intensidade dos pixels em uma imagem (VIOLA; JONES, 2001). No lugar da intensidade, eles identificam diferenças de contraste na imagem, descrevendo assim o aspecto que as características aqui definidas identificam (SCHWAMBACH, 2009). Para extrair estas características, as imagens são convertidas (se necessário) para tons de cinza. Essa conversão, pode ser vista em diversos trabalhos (SCHWAMBACH, 2009; JENSEN, 2008), bem como na biblioteca OpenCV (OPENCVWIKI, 2012). Segundo os autores da pesquisa de 2001, trabalhar com estas características é mais eficiente, por no mínimo, dois motivos: Um sistema baseado em características é mais rápido que um baseado em pixels, e, ainda, estas podem codificar dados que são difíceis de treinar usando uma quantidade finita de dados no treinamento (VIOLA; JONES, 2001). Segundo os autores, não há como aprender características usando, somente, a informação de intensidade do pixel. Desta forma, a definição de características *Haar* é direcionada a aspectos qualitativos, e não somente quantitativos.

Tomando faces por exemplo, poderia ser usada a característica de que a tonalidade dos olhos, em imagens em tons de cinza, é mais escura que a tonalidade das bochechas. O teste, então, poderia usar dois retângulos, um posicionado sobre os olhos e o outro cobrindo as bochechas, passando pelo nariz. Outro teste, poderia levar em conta que a área do nariz é mais clara que dos olhos. Outro tipo de teste poderia usar a área destes retângulos. Como se daria este teste no domínio dos pixels? Aqui não importa o exato valor de cada pixel, mas a diferença relativa entre eles ou grupos deles. A Figura 4 apresenta uma demonstração destas áreas em uma face.

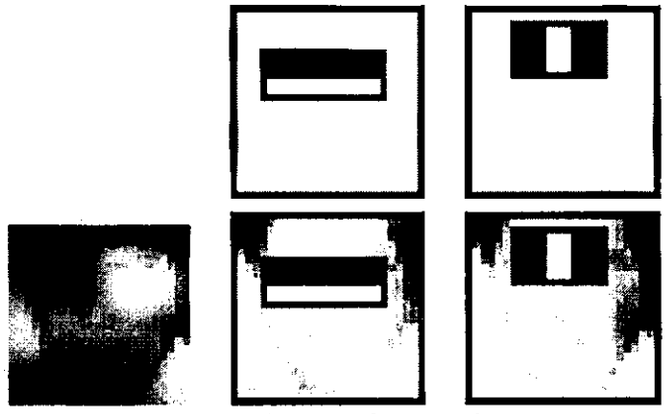

Figura 4: Aplicação de Haar features

Para usá-las, Viola e Jones desenvolveram um subconjunto de características a serem utilizadas no seu algoritmo. Foram assim definidas as características do tipo dois retângulos, em que seu valor final é a diferença da soma dos pixels internos de cada retângulo (como é visto na Figura 5). Há também o tipo com três retângulos: a soma dos pixels dos retângulos externos é diminuída do valor da soma do retângulo interno. E, por fim, o tipo de quatro retângulos: em que o valor final é dado pela diferença dos retângulos diagonais. Outros exemplos são apresentados na Figura 5. Outros pesquisadores expandiram este conjunto de características *Haar*, a exemplo do trabalho de Lienhart e Maydt (2002), que adicionaram características com inclinação de 45º graus.

Estas características não são aplicadas diretamente na imagem como um todo. Elas são processadas e aplicadas em imagens do mesmo tamanho das janelas usadas para a detecção, de tamanho definido em tempo de treinamento. Os treinamentos que acompanham a biblioteca usada neste trabalho, utilizam janelas de 20 x 20 e 24 x 24 pixels. Segundo o autor, essas características independem da escala da imagem, e para detectar objetos maiores que a janela de treinamento, sugere-se escalar o detector em 1.25 vezes por iteração, passando o detector por várias iterações até que não caiba mais na imagem.

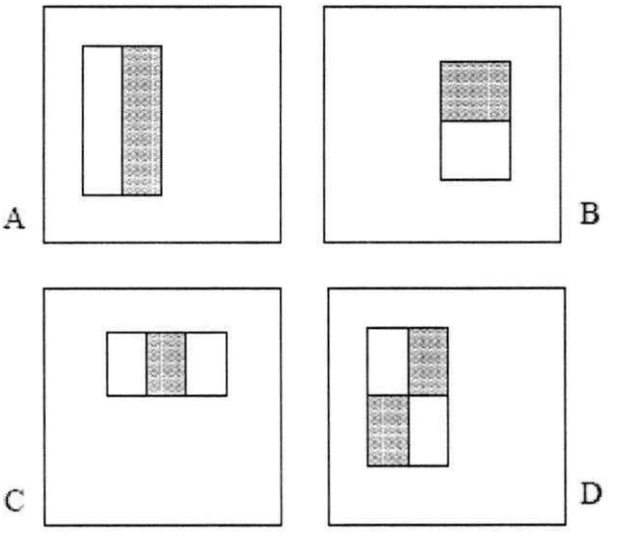

Figura 5: Exemplo de característica Haar

Exemplo de features de formas retangulares. A moldura exterior é a representação da janela usada para treinamento. A soma dos pixels das áreas brancas são subtraídas das suas correspondentes em cinza. Temos em A e B características de dois retângulos, de três em C e quatro em D.

3.1.2 Imagem Integral

Juntamente com o algoritmo proposto por Viola e Jones, foi também introduzida uma nova estrutura de dados. Esta estrutura, foi desenvolvida para permitir que os cálculos necessários aos algoritmos sejam feitos em tempo fixo, independente do tamanho da imagem. Esta contribuição serviu como base e possibilitou as demais contribuições da pesquisa referenciada (VIOLA; JONES, 2001).

A estrutura de dados, consiste no somatório de todos os pixels que estejam acima e posicionados à esquerda do ponto onde se deseja o valor. Formalmente, temos que a imagem integral é dada pela Equação 3 e representada graficamente na Figura 6.

$$ii(x,y) = \sum_{x'<x,\, y'<y} i(x',y') \qquad (3)$$

ii(x,y) sendo a imagem integral, e i(x,y) a imagem digitalizada.

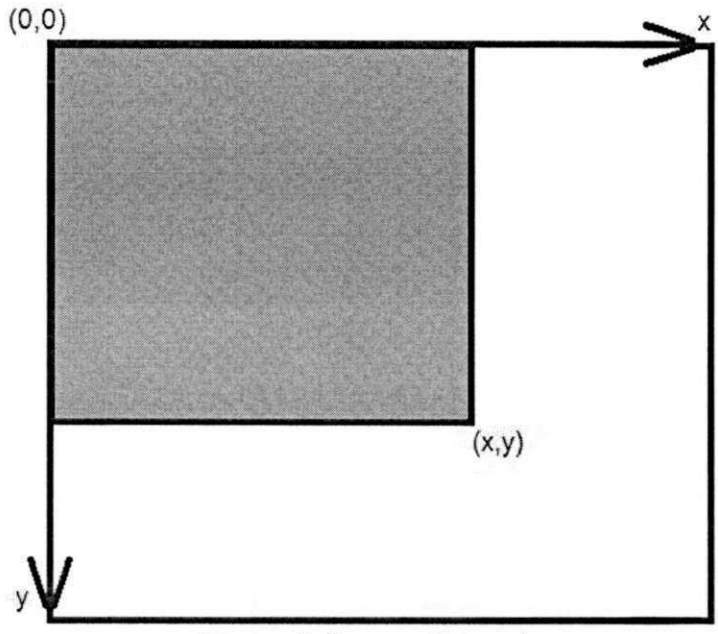

Figura 6: Imagem integral

O valor da imagem integral no ponto (x,y), denotado por ii(x,y), é a soma de todos os pixels acima e à esquerda do ponto (x,y). (Fonte: Viola; Jones, 2001)

A Figura 7 demonstra o efeito do cálculo da imagem integral em uma figura de 3 x 3 pixels. Grifo no fato que o último pixel será o valor da soma de todos os demais.

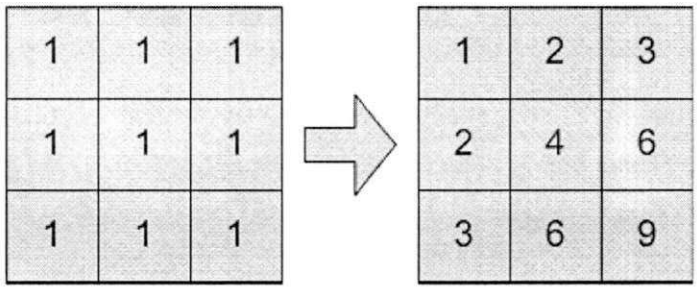

Figura 7: Cálculo de imagem integral
Exemplo de cálculo de imagem integral em matriz 3x3 (Fonte: JENSEN (2009))

Fazendo uso da imagem integral, se torna possível calcular o somatório dos pixels de qualquer forma retangular em tempo constante, fazendo-se a leitura somente de 4 pontos. Usando a Figura 8 como exemplo, para se obter a soma de todos os pixels do retângulo D, usando uma imagem integral, somente se necessita saber dos valores desta nos pontos *a*, *b*, *c* e *d*. A soma em D é dada pelo valor de *d*+*a*-(*b*+*c*). É necessário somar *a*, já que *b* e *c* incluem *a* (o que faz *a* ser subtraído duas vezes).

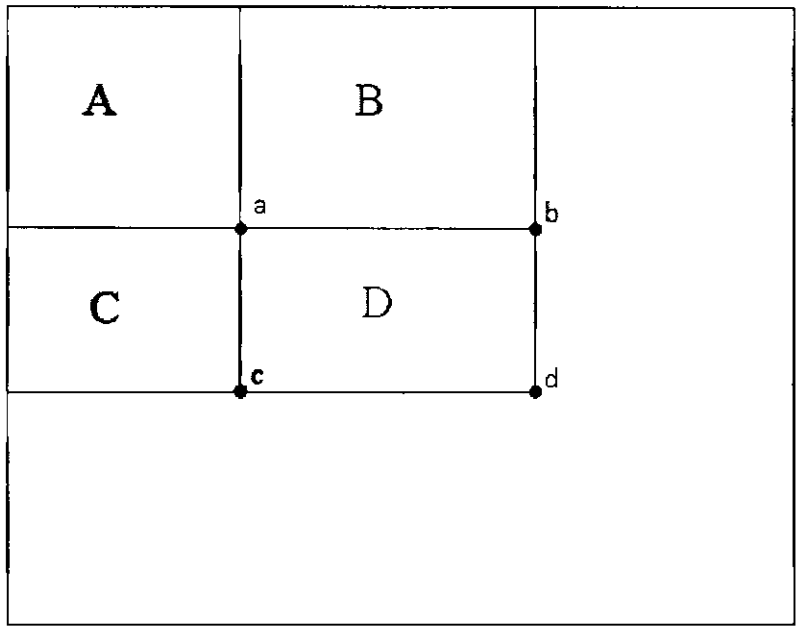

Figura 8: Exemplo uso da Imagem integral
Para obter a soma dos pixels em D, é necessário somente ler a imagem integral em 4 pontos: a, b, c, d. No ponto a, o valor é A. No ponto b, A+B. Em c, A+C e em d A+B+C+D. O valor em D é dada por (d+a)-(b+c).

3.1.3 Cascatas e Classificadores – (Haar Cascades e Haar Classifiers)

O produto final do processo de aprendizagem, desenvolvido por estes pesquisadores, é chamado de uma Cascata *Haar*, e esta cascata é formada por estágios, formados por classificadores fortes, que por sua vez contêm um conjunto de classificadores fracos. A Figura 9 apresenta graficamente estas relações.

Detalhando-os por partes:

Um classificador forte é formado por classificadores fracos e um valor de limiar. Um classificador fraco é formado de uma ou mais características, e seus respectivos pesos. Os classificadores fracos são assim chamados por não haver garantias de que sua classificação seja ótima (a taxa de sucesso pode variar de pouco mais de 50% até 100%). Os pesos e o limiar acima são resultados de um processo de treinamento. No treinamento, é empregado o algoritmo *AdaBoost* (VIOLA; JONES, 2001), o qual escolhe as características que melhor identificam o objeto em questão (seleciona quais as características que são comuns entre as imagens, e quais não são com outras partes que não contêm as características desejadas).

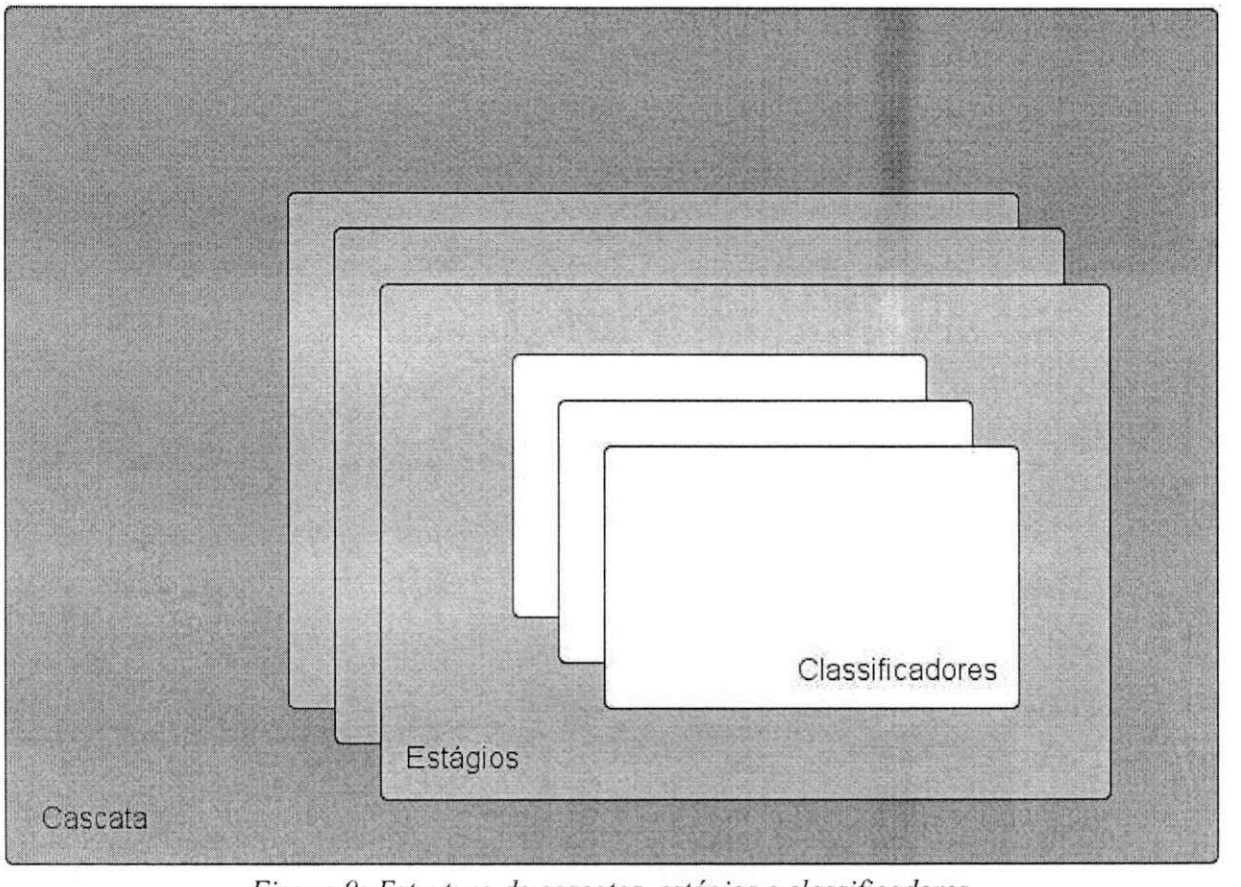

Figura 9: Estrutura de cascatas, estágios e classificadores

Este processo de treinamento consiste em sucessivas iterações, e, ao final de cada uma, os pesos de cada classificador fraco são recalculados, penalizando por falsa detecção e reforçando por sucesso (este processo também é conhecido por *boosting* (SCHWAMBACH, 2009)). A condição de parada do treinamento é atingida ao alcançar um determinado limite de tempo, quantidade de iterações ou por convergência dos pesos calculados (SCHWAMBACH, 2009). É um trabalho de força bruta. O teste de todas as possíveis características é a parte mais custosa em relação ao tempo de todo processo (JENSEN, 2008).

Um estágio, por sua vez, é formado por uma coleção ordenada de classificadores fortes, que são testados sequencialmente, em tempo de detecção, e descartam o processamento dos demais estágios, caso uma janela não obedeça ao seu limiar definido no treinamento. A saída de um estágio é a entrada do posterior, que usa de classificadores mais complexos. Esta sequência é demonstrada na Figura 10.

E, por fim, uma Cascata é a concatenação, em uma sequência bem definida e ordenada, de todos os estágios. É desenvolvida de forma que a grande parte dos resultados negativos seja obtida nos primeiros estágios – que são projetados para utilizar testes mais simples, e assim, descartar áreas que não contenham o objeto tão cedo quanto possível, reduzindo a necessidade de processamento mais complexo. Estes estágios buscam ocorrências de "*não*-objetos", que são mais fáceis de detectar que objetos (JENSEN, 2008). A configuração dos estágios iniciais tem como objetivo direto a maior eficiência do detector.

A cascata de estágios aqui definida é também referenciada pelos autores como Cascatas de Atenção (*Attentional Cascades*), por direcionarem mais "atenção" (recursos computacionais) às áreas da imagem mais propícias a conter o objeto desejado.

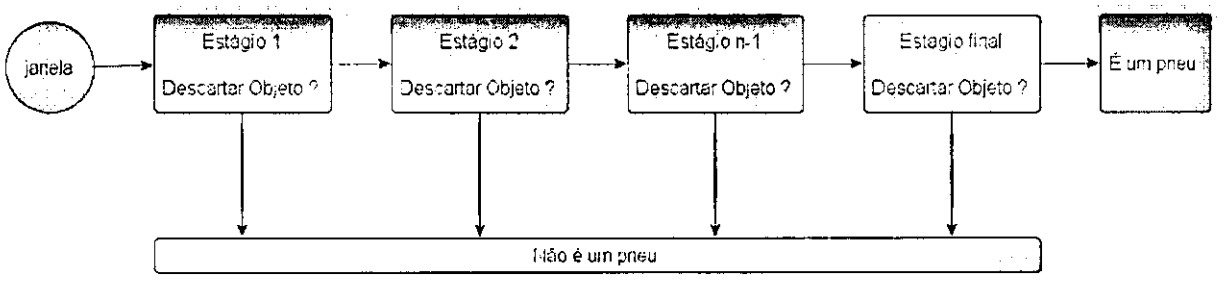

Figura 10: Cascata de estágios

A execução do detector aqui descrito, é feita pela seleção de um tamanho de janela, que então desliza na imagem e analisa cada subimagem delimitada pela janela deslizante. Para cada uma delas, executa toda a cascata – esta execução pode ser interrompida, caso um estágio rejeite a janela atual. O passo, em pixels, definido para determinar a próxima janela é importante. Ele pode aumentar o desempenho do classificador, quando aumentado (gerando menos janelas, e, assim, menos testes). Mas, como efeito negativo, pode saltar objetos se definido demasiadamente grande. É recomendada uma análise do objeto em função da imagem a ser processada.

3.2 Treinamento

Para o treinamento, foram obtidas 5.063 imagens contendo pneus, e 2.032 onde não há o objeto de interesse. Todas as imagens foram obtidas por buscas em sítios da internet, disponíveis para qualquer pessoa, sem restrição de participação em grupo, organização ou registro prévio para o acesso. Para posterior referência, as imagens onde há o objeto serão denominadas imagens *positivas*, e as onde não há, serão referenciadas como *negativas*.

Segundo a documentação consultada, recomenda-se uso de um mínimo de 5.000 imagens positivas e 3.000 negativas para o treinamento (SEO, 2012; LIENHART; KURANOV; PISAREVSKY, 2003; JENSEN, 2008). Levando-se em conta que a média da quantidade de objetos em cada uma das 5.063 imagens é maior ou igual a 2, pode ser tomado o dobro do número acima, como total de imagens positivas (cada aparição do objeto nas imagens é contado como uma imagem). E, como as partes não demarcadas são usadas como imagens negativas, não há de ser prejudicada a eficácia do treinamento, por não ser satisfeita a quantidade de imagens negativas. Como referência, a saída do comando de treinamento do OpenCV, *opencv_traincascade*, tem como valor padrão o número 1.000 para a quantidade de imagens negativas. Neste projeto, as 5.063 imagens são distintas e não há utilização de técnicas para replicar um mesmo objeto de interesse em imagens onde antes este não existia, forçando, assim, a criação de outra imagem positiva sem a necessidade de encontrá-la usando a *internet*, como descrito por Seo (2012).

A sequencia de passos do treinamento é:

1. Coleta de imagens
2. Demarcação dos objetos na imagens
3. Recorte e redimensionamento dos recortes
4. Execução do treinamento

A fase de coleta de imagens foi descrita acima, resultando na base de dados de imagens usada. Os demais passos são executados usando ferramentas da biblioteca OpenCV (OPENCVWIKI, 2012), e são apresentados com mais detalhes no Apêndice B.

A base de dados de imagens foi usada de duas formas. Inicialmente, a demarcação dos objetos ocorreu nas imagens em seu tamanho original. Esta base de dados foi denominada de base Normal. E, diante do fato da menor resolução a ser trabalhada no detector ser 320 x 240 pixels, e também ser a resolução do projeto de Oliveira (2009), que será usada como entrada nesse detector, foi utilizada uma base de dados usando um subconjunto das imagens originais. Para essa seleção, imagens com ângulos de visão mais distantes do observador posicionado em frente ao pneu foram retiradas, e as restantes foram redimensionadas para ter o eixo de comprimento igual a 320 pixels. Essa base foi denominada de Reduzida.

4 Apresentação e Análise dos Resultados

Como a pesquisa foi dividida em duas grandes etapas, os resultados serão apresentados por etapa.

4.1 Treinamento

Foram treinadas várias cascatas, começando com a fase de estudos e testes da ferramenta, chegando então às cascatas para uso no projeto, que variaram o tamanho da janela de 20 x 20 a 128 x 128 pixels e o número de estágios de 4 e de 22. Por serem, os treinamentos, inicialmente limitados a um único processador, só houve êxito nos que faziam uso de janelas no intervalo de 20 x 20 a 56 x 56 pixels. Tentativas com janelas maiores não foram bem sucedidas, todas por motivo de tempo de treinamento, e, algumas, por falta de memória. O pior caso ocupou mais de um mês ininterrupto de CPU, alcançando o segundo após algumas paradas e retomadas dos cálculos. Mas, este processo foi abortado antes de chegar aos seus resultados.

A elevada quantidade de tempo para o processo de aprendizagem, desencadeou uma pesquisa com objetivo de descobrir se este tempo era realmente esperado. Na Tabela 2, são apresentados os tempos de treinamento para os tamanhos de janelas mais importantes para o estudo. Necessário mencionar que os três últimos treinamentos da tabela abaixo necessitavam mais que 6GB de memória RAM.

Janela	Tempo	Base de dados	MinHitRate
20x20	1,5 dias	Normal	0.95
20x20	1,5 dias	Normal	0.99
20x20	1,5 dias	Reduzida	0.95
20x20	1,5 dias	Reduzida	0.99
56x56	4 dias	Normal	0.95
60x60	5 dias	Normal	0.95
80x80	Erro	Normal	0.95
80x80	12 dias	Reduzida	0.95

Tabela 1: Tempo de treinamento usando traincascade

Tentativas de treinamentos com janelas maiores que 80 x 80 pixels aconteceram, mas sem sucesso. Todas aconteceram antes da expansão da memória RAM, e fracassaram logo nos primeiros passos por falta deste recurso. As tentativas usaram janelas de 96 x 96 e 128 x 128 pixels. Para fins de comparação, na Figura 11, é apresentada uma imagem de veículo na resolução menor a ser tratada pelo detector, 320 x 240 pixels.

Figura 11: Exemplo de imagem de carro em 320 x 240 pixels

4.2 Detecção

Para testar a taxa de reconhecimento do detector, foram colhidas 100 imagens distintas de pneus para serem analisadas. Nenhuma delas foi usada como base de dados para o treinamento da cascata. Ainda para o teste, outras 100 imagens foram colhidas mas sem apresentar o objeto em questão. Para estas imagens, usou-se o treinamento de pneus de janela 56 x 56 pixels e de *minHitRate* 0.95. O treinamento para faces utilizado, foi o que acompanha o OpenCV, e, também, o código de Daniel Chang (2012). Este treinamento acompanha a biblioteca e não há informação sobre os procedimentos para repeti-lo, como não é conhecida a base de dados usada, nem os parâmetros usados no aplicativo de treinamento. Somente é sabido o tamanho de sua janela, 20 x 20 pixels.

Como base de comparação para os testes acima, mais 100 imagens contendo faces foram colhidas, e foram apresentadas ao detector original disponível por Daniel Chang (2012). Para o teste com imagens sem faces, as mesmas 100 imagens negativas do teste acima foram utilizadas. Para realizar os testes, foram utilizados recortes de imagens, retirados da imagem em seu tamanho natural, e em formato de um quadrado. Posteriormente, estes recortes foram redimensionados para o tamanho da janela de teste do seu respectivo treinamento.

Os resultados obtidos estão apresentados na Tabela 2.

Teste	Imagens	Detecções	Taxa efetiva
Pneus imagens positivas	100	77	77%
Pneus imagens negativas	100	2	2%
Faces imagens positivas	100	80	80%
Faces imagens negativas	100	1	1%

Tabela 2: Taxa de reconhecimento

Durante os testes, constatou-se, pela observação dos números dos estágios onde

acontecia erro, que dez estágios seria um número ótimo para este classificador reconhecer um pneu, entre os valores de 10, 15 e 22 estágios, também testados. Para todos os testes, o numero de estágios no treinamento era de 22.

No tocante à maturidade do treinamento, a cascata utilizada para faces foi desenvolvida pela equipe da da biblioteca OpenCV, e, é utilizada como referência. Conforme informado, somente é sabido a dimensão da janela, e nada mais. Por outro lado, o treinamento utilizado para pneus foi desenvolvido para esta pesquisa e nunca antes foi testado ou avaliado.

Ainda sobre a maturidade do treinamento, após testes com o enquadramento das imagens, observou-se uma alteração nos resultados do detector. Este comportamento é explicado pelo fato de que um maior número de testes do enquadramento do pneu aumenta a probabilidade de ser escolhido exatamente o recorte correspondente ao pneu esperado pelo detector. E, quanto mais próximo estiver, melhor o resultado. Com este resultado, pode-se afirmar que a implementação de varredura da imagem tende a melhorar o desempenho da detecção, já que, com a varredura, todos os recortes possíveis serão testados. Este comportamento deve ser esperado também para faces, mas em menor grau. Como pneus de carros variam em dimensões, proporções e vizinhança mais que um rosto, é mais simples enquadrar um rosto que um pneu em carros diferentes. Acrescente-se a isto, o fato de uma face ser bem delimitada por um retângulo, sem informações adicionais a seu redor. Em um pneu, se obtido de imagem de carro comum, há sempre pedaços do para-lama na seleção (aliado ao fato de que a distância entre o pneu e o paralama nunca é igual). Observa-se um exemplo deste comportamento nas Figuras 12 e 13, para faces e pneus, respectivamente.

Figura 12: Enquadramento de faces

Figura 13: Enquadramento de pneus

4.3 Base de Dados

Os testes realizados utilizaram imagens colhidas da *Internet*. Cada imagem de pneu apresentava apenas um, centralizado na imagem. Exemplos das imagens utilizadas no experimento cujos resultados foram apresentados na seção anterior podem ser vistos na Figura 14. A Figura 15 apresenta exemplos destas imagens em tamanho maior, antes do redimensionamento.

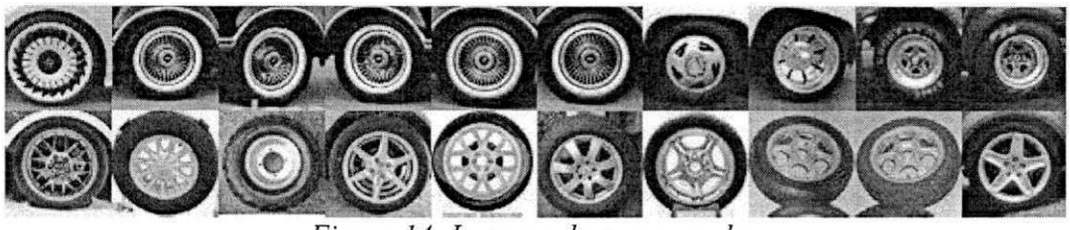

Figura 14: Imagens de pneus usadas

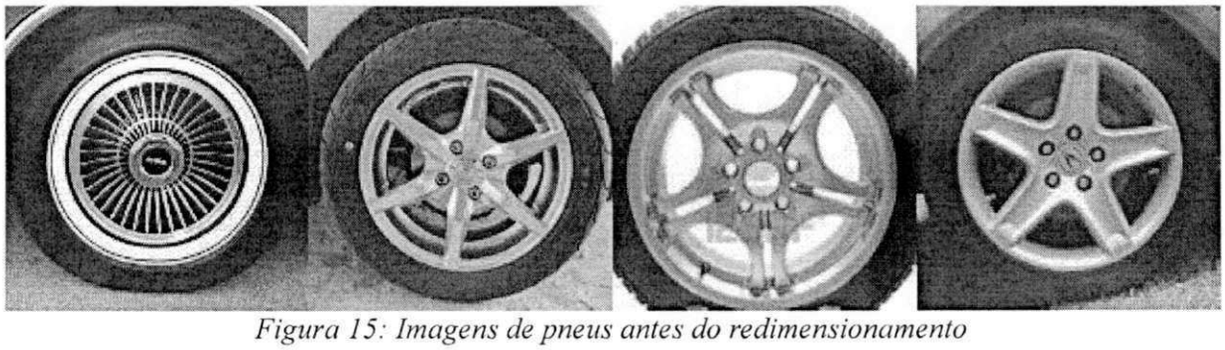

Figura 15: Imagens de pneus antes do redimensionamento

Há variação de ângulo de visão nas imagens, algo presente, também, na base de dados de treinamento.

Para o teste de reconhecimento de faces, o processo de seleção e tratamento das imagens foi o mesmo utilizado com pneus. As Figuras 16 e 17 apresentam exemplos das imagens utilizadas no teste com faces.

Figura 16: Imagens de faces usadas

Como o treinamento usado para faces utilizava janela de 20 x 20 pixels, as imagens de faces eram redimensionadas para esta resolução. No treinamento para pneus, usou-se janela de 56 x 56 pixels.

Figura 17: Imagens de faces antes do redimensionamento

4.4 Considerações Gerais

O treinamento demanda muito tempo e muito memória física (6 GB). Este problema de tempo de treinamento não foi visto pela primeira vez neste tipo de trabalho, Jensen (2008) alegou o mesmo problema de escopo por motivo de tempo dedicado ao treinamento.

Apesar de software *Open Source* ser conhecido como bem documentado, de farta e completa documentação, este aspecto da biblioteca OpenCV não acompanha os demais projetos *Open Source*. Excluindo um livro para cada grande versão deste software (*major version*) – material não disponível, toda a documentação oficial da ferramenta de treinamento se resume a uma página no projeto *Willow Garage* (WILLOW GARAGE, 2012). As demais fontes são de contribuição espontânea de usuários da biblioteca e/ou pesquisadores que dela fizeram uso. Há uma lista, hospedada no *Yahoo.com* (OPENCV ...2012) sobre este software, gerida por usuários, mas frequentada por desenvolvedores – do próprio OpenCV e de outras empresas – que também funciona como fonte de informações.

Como o software *facedetect* foi tomado como referência, imagens somente iriam para a fase de testes com o código em linguagem de descrição de *hardware*, quando o teste em software era bem sucedido. Apesar de treinamentos com janelas maiores apresentarem taxas de reconhecimento altas, o mesmo comportamento não se observava quando o tamanho do objeto se aproximava do tamanho da janela de treinamento.

Outro aspecto em destaque, o fato de não haver trabalhos usando o algoritmo Viola-Jones para reconhecimento de pneus, não há como comparar os resultados obtidos

com outros estudos usando o mesmo objeto, motivo pelo qual a comparação se deu usando a solução com faces. Sobre a quantidade de estágios do detector, não há número definido de estágios para cada objeto. Este estudo obteve melhor eficiência usando 10 estágios, em detrimento de configuração utilizando-se de 15 ou 22 estágios. Como referência, a biblioteca OpenCV (WILLOW GARAGE, 2012) disponibiliza 4 treinamentos para faces, onde o número de estágios varia de 20 a 45.

5 Considerações Finais

O crescimento dos investimentos em segurança e monitoramento de tráfego amplia o campo de aplicação de soluções que assistem ao operador humano nesta tarefa. Nos últimos anos, verificou-se o crescimento do número de empresas de segurança privada. Houve, também, aumento no uso de câmeras de vídeo para monitorar cruzamentos e vias públicas, com vistas ao aperfeiçoamento dos equipamentos utilizados para monitoramento de tráfego e vigilância, foi estudado um modelo de detector com capacidade de reconhecer a presença de veículos em imagens.

O trabalho teve como objetivo central a modelagem do uso do algoritmo Viola-Jones para reconhecimento de pneus, utilizando-se as características deste algoritmo, demonstradas eficientes para o reconhecimento de faces, mas que também podem ser utilizadas com objetos diferente de faces. Outros algoritmos foram estudados, mas nenhum apresentava eficiência para uso em sistemas embarcados, que dispõe de recursos computacionais limitados.

Apesar da implementação completa da solução não está contemplada no escopo do estudo (mais especificamente o tratamento de escalas e a varredura de toda a imagem de entrada), foi implementada parte suficiente para demonstrar que é possível o uso deste algoritmo com pneus, pela análise do desempenho do detector final apresentado, conforme apresentado no Capítulo 4.

No decorrer da pesquisa bibliográfica, não foi encontrado uso do algoritmo Viola-Jones para reconhecimento de outros objetos além de faces, em imagens com fundo heterogêneo.

Uma atenção especial deve ser dada ao processo de treinamento, que pode vir a usar mais tempo que o previsto e é capaz de se tornar a etapa mais demorada de todo o projeto.

5.1 Sugestão de trabalhos futuros

Uma alternativa às características tipo *Haar*, que podem ser experimentadas, são

as características do tipo LBP – *Local Binary Patterns* (LIAO et al., 2007). Este tipo de característica usa valores inteiros, diferente das do tipo *Haar*, e, segundo a documentação da própria biblioteca, o desempenho da fase de treinamento e de detecção é notoriamente melhor (WILLOW GARAGE, 2012).

Outro campo que pode ser explorado é a aplicação de classificadores em paralelo. Abordagem sugerida pelos autores Viola e Jones, Schwambach (2009) demonstrou que é possível aperfeiçoar o detector pela implementação de quantidade de classificadores em número ímpar, e atribuir o sucesso na detecção a uma combinação de resultados dos classificadores.

Trabalhos futuros visando ganho em desempenho podem explorar a implementação de mais blocos em *hardware*, possibilitando, assim, comparações com sistemas puramente desenvolvidos em *software*. Este campo de estudo pode usar o trabalho de Schwambach (2009), em que otimizações foram implementas e obtiveram resultados positivos.

O tratamento de escalas pode ser estudado como extensão deste, que já é capaz de receber imagens em dimensões arbitrárias. A varredura da imagem de entrada é outra função que pode ser integrada ao código, completando-o.

Referências Bibliográficas

ALEXANDRENARTEN. **Wrist Detection!** Disponível em: <http://tech.groups.yahoo.com/group/OpenCV/message/71232>. Acesso em: 19 maio 2010.

ALTERA CORPORATION. **FPGA.** Disponível em: <http://www.altera.com/products/fpga.html>. Acesso em: 15 abr. 2010.

BENKRID, K.; CROOKES, D.; SMITH, J.; BENKRID, A. **High Level Programming for FPGA Based Image and Video Processing Using Hardware Skeletons**. In: ANNUAL IEEE SYMPOSIUM ON FIELD-PROGRAMMABLE CUSTOM COMPUTING MACHINES, 9., 2001, Rohnert Park. Proceedings of the Nineth Annual IEEE Symposium on Field-Programmable Custom Computing Machines. Rohnert Park: Ieee, 2001. p. 219 – 226.

CHANG, Daniel. **Face Detection (Haar Classifiers)**. Disponível em: <http://ercbench.ece.wisc.edu/index.php?option=com_content&view=article&id=39:face-detection&catid=4:image&Itemid=7>. Acesso em: 05 jan. 2012.

CHO, Junguk et al. **FPGA-Based Face Detection System**. In: SIGDA INTERNATIONAL SYMPOSIUM ON FIELD PROGRAMMABLE GATE ARRAYS, 2009., 2009, Monterey. Proceedings... . Nova Iorque: Acm, 2009. p. 1 – 9.

CMUCAM PROJECT (Org.). **Viola Jones Face Detector**. Disponível em: <http://cmucam.org/wiki/viola-jones>. Acesso em: 11 jan. 2012.

DESKSHARE INCORPORATED (New York). **WebCam Monitor 5.24.** Disponível em: <http://www.deskshare.com/lang/po/wcm.aspx>. Acesso em: 20 fev. 2010.

DUDA, Richard O.; HART, Peter E.. **USE OF THE HOUGH TRASFORMTION TO DETECT LINES AND CURVES IN PICTURES**. Communications Of The Acm, Nova Iorque, p. 11-15. jan. 1972.

EVANS, Bonnie; BENNETT, Billy; SOMBRA, Ines. **The Leaf Genie**. Disponível em: <http://theleafgenie.wordpress.com/>. Acesso em: 15 jan. 2012.

FON, Saimon. **Java face detection**. Disponível em: <http://code.google.com/p/jviolajones/>. Acesso em: 11 jan. 2012.

GAO, Changjian; LU, Shih-lien. **NOVEL FPGA BASED HAAR CLASSIFIER FACE DETECTION ALGORITHM**. In: INTERNATIONAL CONFERENCE ON FIELD PROGRAMMABLE LOGIC AND APPLICATIONS, 2008., 2008, Heidelberg. Proceedings... . Heidelberg: Ieee, 2008. p. 373 - 378.

GUO, Si-yu; ZHANG, Xu-fang; ZHANG, Fan. ADAPTIVE RANDOMIZED HOUGH TRANSFORM FOR CIRCLE DETECTION USING MOVING WINDOW. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING AND CYBERNETICS, 5., 2006, Dalian. **Proceedings of the Fifth International Conference on Machine Learning and Cybernetics.** Dalian: IEEE, 2006. p. 3880 – 3885.

IMAGEMAGICK STUDIO LLC. **Imagemagick**. Disponível em: <http://www.imagemagick.org>. Acesso em: 10 jan. 2012.

HOUGH, P.v.c.. **Machine Analysis of Bubble Chamber Pictures.** In: INT. CONF. HIGH ENERGY ACCELERATORS AND INSTRUMENTATION, 11., 1959, Genebra. Proc. Int. Conf. High Energy Accelerators and Instrumentation. Genebra: Ieee, 1959. v. 73, p. 1 – 2.

JENSEN, Ole Helvig. **Implementing the Viola-Jones Face Detection Algorithm**. 2008. 36 f. Dissertação (Mestrado) - Departamento de Department Of Informatics And Mathematical Modeling, Image Analysis And Computer Graphics, Technical University Of Denmark, Kongens Lyngby, 2008.

KROON, Dirk-jan. **Viola Jones Object Detection**. Disponível em: <http://www.mathworks.com/matlabcentral/fileexchange/29437-viola-jones-object-detection>. Acesso em: 11 jan. 2012.

LAM, W.c.y.; EN, S .y. Yu. **Efficient technique for circle detection using hypothesis filtering and Hough transform**. Vision, Image And Signal Processing, Londres, p. 292-300. out. 1996.

LAVRSEN, Kenneth Jahn. **Motion, a software motion detector.** Disponível em: <http://www.lavrsen.dk/twiki/bin/view/Motion/WebHome>. Acesso em: 20 fev. 2010.

LIAO, Shengcai et al. **Learning Multi-scale Block Local Binary Patterns for Face Recognition**. In: INTERNATIONAL CONFERENCE ON BIOMETRICS, 2., 2007, Seul. Proceedings... . Seul: Icb, 2007. p. 828 – 837.

LIENHART, Rainer; KURANOV, Alexander; PISAREVSKY, Vadim. **Empirical Analysis of**

Detection Cascades of Boosted Classifiers for Rapid Object Detection. In: PATTERN RECOGNITION SYMPOSIUM, 25., 2003, Madgeburg. Procedings. Madgeburg: Ieee, 2003. p. 297 – 304.

LIENHART, Rainer; MAYDT, Jochen. **An Extended Set of Haar-like Features for Rapid Object Detection.** In: ICIP, 9., 2002, Rochester. Proceedings... . Rochester: Ieee, 2002. v. 1, p. 900 – 903.

MCLAUGHLIN, Robert A.; ALDER, Michael D.. **The Hough Transform Versus the UpWrite**. Ieee Transactions On Pattern Analysis And Machine Intelligence, Washington, p. 396-400. abr. 1998.

NETLAN TECNOLOGIA DE SISTEMAS LTDA (Santa Catarina). **Sistema de Video Monitoramento de Ambientes.** Disponível em: <http://www.netlan.net/svma/>. Acesso em: 20 fev. 2010.

OLIVEIRA, Helder Fernando de Araújo. **Reformulação, baseada em OVM, da metodologia de verificação funcional VeriSC**. 2010. 110f. Dissertação (Mestrado) – Ufcg, Campina Grande, 2010.

OLIVEIRA, Jozias Parente De. **MÉTODO PARA EXTRAÇÃO DE OBJETOS DE UMA IMAGEM DE REFERÊNCIA ESTÁTICA COM ESTIMATIVA DAS VARIAÇÕES DE ILUMINAÇÃO.** 2009. 167 f. Tese (Doutorado) - Ufpa, Belém, 2009.

OPENCV · **Open Source Computer Vision Library Comm** Disponível em: <http://tech.groups.yahoo.com/group/OpenCV/>. Acesso em: 10 jan. 2012.

OPENCVWIKI (Org.). **OpenCV Wiki**. Disponível em: <http://opencv.willowgarage.com/wiki/>. Acesso em: 05 jan. 2012.

PAPAGEORGIOU, Constantine; OREN, Michael; POGGIO, Tomaso. **A General Framework for Object Detection**. In: INTERNATIONAL CONFERENCE ON COMPUTER VISION, 6., 1998, Bombay. Procedings. Bombay: Narosa Publishing House, 1998. v. 1, p. 555 - 562.

RAD, Ali Ajdari; FAEZ, Karim; QARAGOZLOU, Navid. **Fast Circle Detection Using Gradient Pair Vectors.** In: VIITH DIGITAL IMAGE COMPUTING: TECHNIQUES AND APPLICATIONS, 7., 2003, Sydney. Proceedings of the VIIth Digital Image Computing: Techniques and Applications. Sydney: Ieee, 2003. p. 879 – 887.

RBOUGHA. **Haar training for different gesture forms**. Disponível em: <http://tech.groups.yahoo.com/group/OpenCV/message/62829>. Acesso em: 01 maio 2009.

RHODY, Harvey. **Lecture 10: Hough Circle Transform.** Disponível em: <http://www.cis.rit.edu/class/simg782/lectures/lecture_10/lec782_05_10.pdf>. Acesso em: 10 fev. 2010.

SCHWAMBACH, Vítor. **Optimization of a Face Detection Algorithm for Real-time Mobile Phone Applications.** 2009. 54 f. Dissertação (Mestrado) - Ufpe, Recife, 2009.

SEDCOLE, N.P.; CHEUNG, P.Y.K.; CONSTANTINIDES, G.A.; LUK, W. **A Reconfigurable Platform for Real-Time Embedded Video Image Processing**. In: INTERNATIONAL CONFERENCE ON FIELD-PROGRAMMABLE LOGIC AND APPLICATIONS, 13., 2003, Lisboa. Proceedings of the 13 International Conference on Field-Programmable Logic and Applications. Lisboa: Ieee, 2003. p. 606 - 615.

SEO, Naotoshi. **Tutorial: OpenCV haartraining (Rapid Object Detection With A Cascade of Boosted Classifiers Based on Haar-like Features)**. Disponível em: <http://note.sonots.com/SciSoftware/haartraining.html>. Acesso em: 05 jan. 2012.

SHANG, Fei.; LIU, Jinwei.; ZHANG, Xiao, TIAN, Di. An improved circle detection method based on right triangles inscribed in a circle. In: WORLD CONGRESS ON COMPUTER SCIENCE AND INFORMATION ENGINEERING, 2009., 2009, Los Angeles. **Proceedings of the World Congress on Computer Science and Information Engineering 2009.** Los Angeles: Ieee, 2009. p. 382 - 387.

SHIE, Mon Chau; JEN, Jau Ruen; CHEN, Charlie. A Circular Hough Transform Hardware for Industrial Circle Detection Applications. In: IEEE CONFERENCE ON INDUSTRIAL ELETRONICS AND APPLICATIONS, 1., 2006, Singapore. **Proceedings of the 1st IEEE Conference on Industrial Eletronics and Applications.** Singapore: IEEE, 2006. p. 1 – 6.

SILVA, Karina Rocha Gomes da. **Uma Metodologia de Verificação Funcional para Circuitos Digitais**. 2007. 121 f. Tese (Doutorado) - Ufcg, Campina Grande, 2007.

SMART UNION (São Paulo). **CFTV - DVR - VÍDEO MONITORAMENTO DE IMAGEM DIGITAL.** Disponível em: <http://www.smartunion.com.br/cftv_dvr_video_server_monitoramento_digital_cctv.asp>. Acesso em: 20 fev. 2010.

SONY CORP.. **New digital cameras with smile shutter**. Disponível em: <http://www.sony.co.uk/article/id/1189437949577>. Acesso em: 18 abr. 2010.

TORII, Akihiko; IMIYA, Atsushi. The randomized-Hough transform-based method for great-circle detection on sphere. **Pattern Recognition Letters,** Nova Iorque, p. 1186-1192. jul. 2007.

VIOLA, Paul; JONES, Michael. **Robust Real-time Object Detection**. In: INTERNATIONAL WORKSHOP ON STATISTICAL AND COMPUTATIONAL THEORIES OF VISION – MODELING, LEARNING, COMPUTING, AND SAMPLING, 2., 2001, Vancouver. Proceedings... . Vancouver: Ijcv, 2001. p. 1 – 25.

WIKIPEDIA. **Video Processing.** Disponível em: <http://en.wikipedia.org/wiki/Video_processing>. Acesso em: 20 out. 2009.

WIKIPEDIA. **Real-time computing.** Disponível em: <http://en.wikipedia.org/wiki/Real-time_computing>. Acesso em: 16 abr. 2010.

WIKIPEDIA. **Real-time operating system.** Disponível em: <http://en.wikipedia.org/wiki/Real-time_operating_system>. Acesso em: 16 abr. 2010.

WIKIPEDIA. **Field-programmable gate array.** Disponível em: <http://en.wikipedia.org/wiki/Field-programmable_gate_array>. Acesso em: 18 abr. 2010.

WIKIPEDIA. **Hough transform.** Disponível em: <http://en.wikipedia.org/wiki/Hough_transform>. Acesso em: 18 abr. 2010.

WILLOW GARAGE. **Cascade Classifier Training**. Disponível em: <http://opencv.itseez.com/trunk/doc/user_guide/ug_traincascade.html>. Acesso em: 05 jan. 2012.

WIKIPEDIA (Eua). **OpenCV**. Disponível em: <http://en.wikipedia.org/wiki/OpenCV>. Acesso em: 05 jan. 2012.

WU, Jianping. **Robust Real-time Ellipse Detection By Direct Least-Square-Fitting**. In: INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE AND SOFTWARE ENGINEERING, 1., 2008, Wuhan. Proceedings of the First International Conference on Computer Science and Software Engineering. Wuhan: Ieee, 2008. p. 923 – 927.

XU, Lei; OJA, Erkki; KULTANEN, Pekka. 1.**A new curve detection method: Randomized Hough Transform (RHT)**. Pattern Recognition Letters, North Holland, p. 331-338. jan. 1990.

XU, Lei; OJA, Erkki. **Randomized Hough Transform.** Coruña: Igi Global, 2009. 1777 p.

XU, Lei; OJA, Erkki. **Randomized Hough Transform (RHT): Basic Mechanisms, Algorithms, and Computacional Complexities.** Cvgip: Image Understanding, Eselvier, p. 131-154. mar. 1993.

ZIFF DAVIS PUBLISHING HOLDINGS INC (Estados Unidos). **Definition of: real time**. Disponível em: <http://www.pcmag.com/encyclopedia_term/0,2542,t=real+time&i=50259,00.asp>. Acesso em: 06 maio 2010.

Apêndice A – Scripts de treinamento

Apresentação dos *scripts* e códigos gerados para treinamento

run.sh

```
#!/bin/sh

CLASS=class63_95-norm
W=63
H=63
MEM=2048
STAGES=22
MINHITRATE=0.95

echo "==> Treinamento $CLASS"
echo "==> Classifier: $CLASS ${W}x${H} MEM: $MEM Stages: $STAGES MINHITRATE:
$MINHITRATE"

echo "=== > Detalhes em $CLASS.log"

darkside/0-organiza-txts-sep.txt > /dev/null 2> $CLASS-err.log
echo "=== > Criando lista negativa"
darkside/1-cria-lista-negativas.sh > /dev/null 2>> $CLASS-err.log
echo "=== > Criando samples"
darkside/2-cria_samples-sep.sh $W $H $CLASS > /dev/null 2>> $CLASS-err.log
echo "=== > Iniciando treinamento"
darkside/3-neo-treina-sep.sh $STAGES $W $H $MEM $CLASS $MINHITRATE >
$CLASS.log 2>> $CLASS-err.log
#darkside/3-neo-treina-sep.sh $STAGES $W $H $MEM $CLASS $MINHITRATE
```

0-organiza-txts-sep.txt

```
#!/bin/sh

# Apos o tratamento de cada imagem positiva, eh necessario juntar todos os arquivos txt
em um so
# Simples cat dos txt's em um somente

NUM="0000"
```

```
DIRSAIDA="imagens-positivas-separadas"

if [ $# -lt 2 ]; then

  echo "Uso: 0-junta-txts.sh DIR SAIDA"
  echo "Assumindo o padrao"

  DIR=positivas/
#  SAIDA=$DIRSAIDA/$NUM.txt

else

  # diretorio onde estao os txt
  DIR=$1

  # arquivo de saida
  SAIDA=$2

fi

rm -rf $DIRSAIDA

mkdir $DIRSAIDA

LISTA="0100 0200 0300 0400 0500 0600 0700 0800 0900 1000 1100 1200 1300 1400
1500 1600 1700 1800 1900 2000 2100 2200 2300 2400 2500 2600 2700 2800 2900 3000
3100 3200 3300 3400 3500 3600 3700 3800 3900 4000 4100 4200 4300 4400 4500 4600
4700 4800 4900 5000 5100"
#LISTA="0100"

for arquivo in $LISTA
do
  while read img
    do
      echo $img > $DIRSAIDA/$NUM.tmp
      sed -e "s'^'positivas/'g" $DIRSAIDA/$NUM.tmp > $DIRSAIDA/$NUM.txt

      rm -f $DIRSAIDA/$NUM.tmp

      NUM=`expr $NUM + 1`
      if [ $NUM -lt 10 ]; then
        NUM="000$NUM"
      elif [ $NUM -lt 100 ]; then
        NUM="00$NUM"
      elif [ $NUM -lt 1000 ]; then
        NUM="0$NUM"
      fi
    done < positivas/$arquivo.txt
```

```
done

ln -s ../positivas/ $DIRSAIDA/positivas

#echo "numero de imagens positivas:" `wc -l $SAIDA`
```

1-cria-lista-negativas.sh

```
#!/bin/sh

# Simplesmente cria um txt com a lista das imagens negativas

if [ $# -lt 2 ]; then

  echo "Uso: 2-cria-lista-negativas.sh DIR SAIDA"
   echo "Usando padrao"

  DIR=negativas/
   SAIDA=todas-negativas.txt

else

  DIR=$1
   SAIDA=$2

fi

find $DIR | grep -i jpg > $SAIDA

echo "numero de negativas: " `wc -l $SAIDA`
```

2-cria_samples-sep.sh

```
#!/bin/sh

W=56
H=56

# Teste

darkside/createtrainsamples.pl

  W=$1
```

```
H=$2
CLASSIFIER=$3
POSITIVASSEPARADAS=imagens-positivas-separadas
VECDIR=samples-sep
VECDAT=$CLASSIFIER-$W-$H-sep.dat
VECFILE=$CLASSIFIER-$W-$H-sep.vec
PERL=darkside/createtrainsamples.pl

# Script para criar o arquivos .vec necessario para o opencv-haartraining
#
# necessita da lista das imagens positivas (com a descricao das areas onde ha o objeto
desejado)
# resulta no arquivo .vec
#
# Exemplo:
# ./opencv-createsamples -info /home/group03/haarimages/sign1positive/output.txt
# -vec /home/group03/haarimages/sign1positive/positives.vec -w 24 -h 24
# Usando -w 20 e -h 20, segundo lido em
http://note.sonots.com/SciSoftware/haartraining.html

# apagar $VECDIR
rm -rf $VECDIR

for img in `ls ${POSITIVASSEPARADAS}/`
do
echo  opencv_createsamples -info ${POSITIVASSEPARADAS}/$img -vec $VECDIR/
$img.vec -w $W -h $H -num 1
  opencv_createsamples -info ${POSITIVASSEPARADAS}/$img -vec $VECDIR/$img.vec
-w $W -h $H -num 1
done

rm -f $VECDIR/positivas.vec

# limpando os vecs
cd $VECDIR
  find . -size 1 -exec rm -f {} \;
cd -
# juntando todos num so
find $VECDIR/ -name '*.vec' > $VECDAT

ferramentas/mergevec/mergevec $VECDAT $VECFILE -w $W -h $H
```

3-neo-treina-sep.sh

```
#!/bin/sh
```

```
H=56
W=56
MEM=2048

 echo "Uso: 3-treina.sh TREINAMENTO VEC NEGATIVAS NSTAGES"
 echo "Assimundo padrao"

 CLASSIFIER=$5
 MINHITRATE=$6
 H=$2
 W=$3
 VEC=$CLASSIFIER-$W-$H-sep.vec
 NEGATIVAS=todas-negativas.txt
 NPOS=`ls -1 imagens-positivas-separadas | wc -l`
 NPOS=`expr $NPOS - 1`
 NNEG=`wc -l $NEGATIVAS`

 NSTAGES=$1
 TREINAMENTO=$CLASSIFIER-$NSTAGES-sep
 MEM=$4

# Treinamento propriamente dito
#
# fonte: http://cgi.cse.unsw.edu.au/~cs4411/wiki/index.php?title=OpenCV_Guide
#       http://note.sonots.com/SciSoftware/haartraining.html#Tutorial
# opencv-haartraining -data /home/group03/trainout -vec
/home/group03/haarimages/sign1positive/positives.vec \
#  -bg /home/group03/sign1_negative/negatives.txt -npos 1000 -nneg 1000 -nstages 20

# Chamando o opencv-haartraining com os arquivos gerados nos passos passados

echo opencv_traincascade.2.3.1 -data $TREINAMENTO -vec $VEC -bg $NEGATIVAS -w
$W -h $H -numPos $NPOS -numNeg $NNEG -baseFormatSave -featureType HAAR -bt
GAB -maxDepth 1 -mode ALL -numStages $NSTAGES -minHitRate $MINHITRATE
opencv_traincascade.2.3.1 -data $TREINAMENTO -vec $VEC -bg $NEGATIVAS -w $W -h
$H -numPos $NPOS -numNeg $NNEG -precalcValBufSize 1024 -precalcIdxBufSize 1024
-baseFormatSave -featureType HAAR -bt GAB -maxDepth 1 -mode ALL -numStages
$NSTAGES -minHitRate $MINHITRATE
```

Apêndice B – Suporte ferramental

São apresentados aqui, com mais detalhes, as ferramentas utilizadas durante a pesquisa.

B.1 Biblioteca OpenCV

A biblioteca OpenCV (OPENCVWIKI, 2012) é o resultado do trabalho, inicialmente, da Intel e atualmente é desenvolvida pelo *Willow Garage* (WIKIPEDIA, 2012). Ela é aplicada em diversas áreas de processamento de imagens e vídeos, dentre elas a de detecção de objetos tratada neste trabalho.

Esta ferramenta implementa o algoritmo alvo desta pesquisa, e foi utilizada para estudo do algoritmo desenvolvido por Viola e Jones. Também foi utilizada no momento de treinar o classificador para detectar pneus. Algumas ferramentas desta biblioteca foram utilizadas em diferentes momentos do trabalho. A versão utilizada foi a 2.3.1.

Inicialmente fez-se uso do aplicativo de teste, *facedetect,* para teste do classificador de faces que acompanha o OpenCV, usando diversas imagens para este fim. Apesar do nome (que significa detector de faces em inglês), a ferramenta pode ser utilizada com qualquer cascata de estágios e não somente para detectar faces (OPENCVWIKI, 2012). Ela recebe o treinamento desejado como parâmetro, em conjunto com a imagem a ser analisada.

O passo posterior é demarcar as áreas das imagens em que há o objeto desejado. Neste passo, é necessário editar cada imagem separadamente, marcando cada aparição do objeto em questão. A saída deste passo é um arquivo contendo o caminho para cada imagem, seguido da quantidade de objetos presentes, e por fim, as coordenadas do retângulo que delimita cada um. Um exemplo do conteúdo de um arquivo deste pode ser visto a seguir.

```
0100/_MG_0552.jpg 1 295 596 90 148
0100/_MG_0555.jpg 3 1123 374 109 156 1265 133 73 112 750 513 36 67
```

0100/_MG_0556.jpg 4 350 587 161 253 619 589 89 85 1082 623 169 100 1353 646 324 471
0100/_MG_0559.jpg 2 353 559 214 215 1044 632 117 158

A demarcação dos objetos nas imagens foi executada usando a ferramenta *ObjectMarker* (SEO, 2012). Esta ferramenta não faz parte da biblioteca OpenCV.

O próximo passo utiliza a ferramenta *createsamples*, em que cada arquivo listado no passo anterior é lido, e cada pedaço descrito acima como um objeto é recortado e salvo noutro arquivo do tipo *vec*. A execução deste aplicativo necessita da informação do tamanho da janela a ser utilizada pelo algoritmo.

Inicialmente, seria passada para esta ferramenta uma lista contendo todos os arquivos e localização de objetos. Mas, este expediente não gerava os resultados esperados. A ferramenta *createsamples* não executava todos os comandos conforme o descrito em seu manual. E, consequentemente, o arquivo final resultante não seria válido. Para obter o resultado necessário deste aplicativo, foi necessário executar diretamente uma chamada para cada arquivo de entrada. Esta tarefa foi automatizada em *script*. É gerado, então, um arquivo *.vec* para cada imagem positiva, mas na fase de treinamento é necessário ter somente um único arquivo com toda a informação. Mediante o uso da ferramenta *mergevec*, todos estes arquivos são condensados em um único. Esta ferramenta também não acompanha a biblioteca, e está disponível abertamente na *Internet* (SEO, 2012).

Antes iniciar o treinamento propriamente dito, é necessária uma lista com as imagens negativas a serem utilizadas, com seus respectivos caminho de acesso.

De posse de todos os pré-requisitos, é possível iniciar o processo de treinamento do classificador. Há uma gama de parâmetros que modificam o treinamento e seu processo, destacando o número de estágios, a resolução da janela (que necessariamente deve ser igual ao valor dos passos anteriores) e a taxa de acerto do algoritmo em cada estágio (*minHitRate).* O manual do aplicativo de treinamento lista todas, mas não há explicação detalhada das opções.

A biblioteca OpenCV disponibiliza duas ferramentas para treinamento: *haartraining* e *traincascade*. Não há muita documentação sobre estas ferramentas, como manuais (*man*) quando instaladas em máquinas com Sistemas Operacionais da família Unix, e

somente no histórico da lista oficial do OpenCV no *Yahoo! Groups* (OPENCV, 2012) foi encontrada a diferença básica entre as duas ferramentas. A primeira, é a ferramenta clássica e desenvolvida juntamente com a biblioteca para o treinamento. Ela é limitada, usa só um processador, independentemente de quantos processadores a máquina de treinamento tenha à disposição. A segunda, é mais recente. Apareceu com a versão 2.0 da biblioteca e apesar de ter basicamente as mesmas funcionalidades, é compatível com dois tipos de características de treinamento: *Haar* e *LBP* (LIAO *et. al.*, 2007), e pode fazer uso de processamento paralelo, se compilada com este suporte (em conjunto com biblioteca específica para este fim). Esta capacidade a torna mais eficiente em processadores atuais, que comumente apresentam mais de uma unidade de processamento. Nos primeiros treinamentos, usou-se a mais antiga, mas ao final deste trabalho, houve a troca para a segunda.

Todo o processo de treinamento, desde a criação das listas de imagens positivas e negativas, passando pela geração dos *.vec*, seu tratamento e o treinamento usando todas estas informações, foi automatizado via *scripts* para padronização e praticidade de repetição de cada tarefa. Os *scripts* estão disponíveis no Apêndice A e na página *web* do Laboratório de Arquiteturas Dedicadas (LAD) da UFCG (http://lad.dsc.ufcg.edu.br/~matheus/mestrado).

B.2 Detector de referência

No estudo, utilizou-se como base o detector de faces, implementado em linguagem de descrição de *hardware*, disponibilizado por Daniel Chang (2012). O código foi desenvolvido para o reconhecimento exclusivamente de faces, e somente tem a capacidade de reconhecê-las se receber, diretamente, a imagem integral (os dados são obtidos da leitura de um arquivo texto com os valores) de tamanho 20 x 20 pixels. Não há tratamento de captura e varredura de imagem, nem há geração de imagem integral, e nada fora a leitura da imagem integral, que é carregada de um arquivo de texto no aplicativo de teste (*testbench*) que acompanha o código. Este código somente tem implementado a execução do detector. O detector utiliza os dados carregados da imagem integral, executa a cascata de estágios e repete este procedimento até ser interrompido.

Não há controle de detecção, e, sempre, todos os estágios são percorridos em todas as execuções do detector. O código é uma parte do projeto de Cho *et. al.* (2009), em que um sistema completo – da entrada da imagem ao monitor de saída – foi desenvolvido. O código do projeto de Cho *et al.* não se encontra disponível.

Além do arquivo do detector, existem outros arquivos que definem a memória do hardware descrito. Estes arquivos carregam os dados gerados no treinamento e salvos na cascata de estágios. Há um arquivo para cada tipo de dado, bem como um para os limiares de cada estágio. As informações das características são divididas em três arquivos de memória, um para cada retângulo. Todos os arquivos são indexados pelo número da característica, e todas as características usadas na cascata são numeradas de zero até o número total de características, subtraindo-as em 1.

As informações de limiares de estágios são codificadas diretamente em linhas de memória, aplicando uma multiplicação pelo valor inteiro 4.096 (2^12) para converter os valores fracionários em um número inteiro. Tratamento semelhante é dado aos demais valores (exceto retângulos das características), havendo somente uma diferença no multiplicador usado para o limiar das características.

Os dados referentes aos retângulos, onde serão lidos os valores da imagem integral, apresentam tratamento diferente. Para cada retângulo que define as características *Haar*, um bloco de memória é definido. Se a característica não usar o terceiro retângulo, este tem todos valores iguais a zero. Cada linha de memória codifica todos os dados de um retângulo. A organização é ilustrada na Tabela 3.

Valor	f_weight	f_height	f_width	f_y	f_x
11100100011100011100011	111	00100	01110	00111	00011
11100100100100001000001	111	00100	10010	00010	00001

Tabela 3: Estrutura da memória - Características

As linhas da memória de dados dos retângulos têm 23 bits de tamanho, e codificam, nesta ordem, o peso da característica, sua altura, comprimento, e os valores das coordenadas y e x. Estes dados têm 3, 5, 5, 5 e 5 bits, respectivamente, no código original. Para os dados destes retângulos, não há multiplicação por número inteiro.

Para informar se houve ou não a detecção do objeto, no caso a face no código original, o detector tem duas saídas de um *bit* cada, que sinalizam com valor lógico '1' indicando a ocorrência do evento, de nomes *Failed* e *Detected*.

Além do *testbench* para testar o código, acompanham ainda alguns arquivos de texto contendo imagens integrais de exemplos de faces e de não-faces para testes. O código foi desenvolvido limitando-se a estas restrições.

Para ser capaz de utilizar treinamentos com janelas maiores que 20 x 20 pixels, foram feitas modificações no código original de Chang (2012). As alterações iniciaram com tamanhos de variáveis. Um estudo foi realizado para analisar as variáveis que necessitavam de faixa maior de valores – em Verilog, a declaração da variável é feita explicitando a quantidade exata de *bits*. Um exemplo das variáveis são as que definem o tamanho da imagem integral, de 5 para 9 *bits*, e a variável de armazenamento dos elementos da imagem integral. Como a última posição da imagem integral é a soma de todos os demais valores, e a quantidade de posições aumentara em mais que o quádruplo (400 em 20 x 20 pixels para 6400 em 80 x 80 pixels), o valor da soma também demandou alteração.

Outra mudança aconteceu na definição da memória utilizada, onde são armazenados os dados do treinamento. Estes dados são armazenados um por cada linha, segundo codificação definida no capítulo anterior, e o tamanho da palavra de memória foi alterado para acomodar a nova faixa de valores nela codificada. Como esta palavra contém informações que necessitaram de mais *bits*, a palavra como um todo também foi aumentada. Houve também alteração quanto ao sinal de alguns valores, que no treinamento original não apresentavam números negativos. O código original foi desenvolvido especificamente para o treinamento *haarcascade_frontalface_alt.xml*, com todas as suas restrições de valores.

Não foi alterado o fato do detector deter informações sobre a cascata de estágios, obrigando assim a modificá-lo em virtude de mudança na cascata. O código modificado do detector, juntamente com todos os códigos deste trabalho estão disponíveis na página web do Laboratório de Arquiteturas Dedicadas (LAD) da UFCG (http://lad.dsc.ufcg.edu.br/~matheus/mestrado).

B.3 *Ferramentas desenvolvidas*

Somente o detector foi executado em linguagem de descrição de *hardware*, os demais passos usavam soluções em *software*. O processo foi automatizado, desde o redimensionamento das imagens à resolução desejada, até a coleta dos dados do detector.

O redimensionamento e a conversão para YUV foram realizados usando o *Imagemagick* (IMAGEMAGICK STUDIO LLC, 2012), ferramenta disponível gratuitamente.

Para retirar a camada desejada do arquivo YUV, foi desenvolvida, em C, uma ferramenta que recebe como entrada o arquivo YUV e o tamanho da janela, e retorna um arquivo somente com a camada Y. Este arquivo é, em seguida, usado como entrada para o aplicativo, também desenvolvido em C, para calcular a matriz correspondente à imagem integral da imagem em questão, já disposta segundo o formato esperado pelo código principal do projeto. Todos são chamados em *script* único.

Outra ferramenta desenvolvida nesta pesquisa, consiste num conversor da cascata de estágios do formato *XML* (arquivo de saída do treinamento do OpenCV) para a representação em Verilog usada pelo código, descrito anteriormente neste texto. Este aplicativo, chamado *XML_parser*, é capaz de ler cascatas de quaisquer tamanhos, com o limite de uma característica por classificador fraco. Como saída, este aplicativo gera os arquivos em Verilog correspondentes às memórias do detector, juntamente com o arquivo principal do detector devidamente integrado às mudanças necessárias para uso do novo treinamento.

Esta ferramenta tem parâmetros de configuração em seu código para gerar o código devidamente adequado ao treinamento em questão, no que diz respeito ao tamanho da janela de treinamento, sem necessitar de futura intervenção por parte do desenvolvedor. Para a geração destes arquivos, há um padrão (*template*) para cada tipo e posição (quando necessário). Estes arquivos possuem as partes não mutáveis dos códigos. O aplicativo escreve no arquivo de saída cada parte na sequencia correta, usando o padrão ou a saída do processamento do *xml*, conforme a parte que está sendo processada. Para tanto, foram criados arquivos que contém cada parte que não muda do

arquivo final, em arquivos identificados, para serem lidos e concatenados segundo a sequencia definida no aplicativo.