UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

VISUAL AUTONOMY – PROTÓTIPO PARA RECONHECIMENTO DE PLACAS DE TRÂNSITO

FERNANDO POFFO

BLUMENAU
2010

2010/1-13

FERNANDO POFFO

VISUAL AUTONOMY – PROTÓTIPO PARA RECONHECIMENTO DE PLACAS DE TRÂNSITO

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Mauro Marcelo Mattos, Dr. - Orientador

BLUMENAU
2010

2010/1-13

VISUAL AUTONOMY – PROTÓTIPO PARA RECONHECIMENTO DE PLACAS DE TRÂNSITO

Por

FERNANDO POFFO

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Mauro Marcelo Mattos, Dr. – Orientador, FURB
Membro:	Prof. Paulo César Rodacki Gomes, Dr. – FURB
Membro:	Prof. Roberto Heinzle, M.Eng. – FURB

Blumenau, 05 de julho de 2010

AGRADECIMENTOS

À minha família, que sempre esteve presente.

Aos meus colegas, pelos empurrões e cobranças.

Às pessoas que disponibilizam, na *internet*, exemplos, documentação e outros materiais que servem como fonte de conhecimento.

Ao meu orientador, Mauro Marcelo Mattos, por ter acreditado na conclusão deste trabalho.

A mente que se abre a uma nova idéia jamais voltará ao seu tamanho original.

Albert Einstein

RESUMO

Este trabalho descreve um protótipo de uma ferramenta de detecção e reconhecimento de placas de trânsito, com possibilidade de ser integrada em um sistema de apoio ao motorista instalado em um computador de bordo veicular. Para a detecção, são implementadas técnicas de processamento de imagens, tal como a transformada de Hough utilizada para localizar o formato circular de um tipo de placa de trânsito. Na etapa do reconhecimento, submete-se a imagem a uma rede neural *multilayer perceptron* treinada utilizando algoritmo *backpropagation*.

Palavras-chave: Visão computacional. Reconhecimento de objetos. Segmentação de imagem. Segurança no trânsito.

ABSTRACT

This work describes a software tool for detection and recognition of road signs. We used image processing techniques, such as the Hough transform for identifying possible road sign images and a multilayer neural network trained with backpropagation algorithm. We verified that the noise and the absence of bright control introduced in the images captured by webcam impacts the precision of the recognition process.

Key-words: Computer vision. Object recognition. Image segmentation. Traffic safety.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

ADAS – *Advanced Driver Assistance Systems*

API – *Application Programming Interface*

BP – *Back Propagation*

CONTRAN – COnselho Nacional de TRÂNsito

EA – Enterprise Architect

GM – General Motors

GPS – *Global Positioning System*

JAI – Java Advanced Imaging

JMF – Java Media Framework

MLP – *MultiLayer Perceptron*

NHD – *Non-Homogeneous Detector*

PRF – Polícia Rodoviária Federal

RF – Requisito Funcional

RGB – *Red, Green, Blue*

RNA – Rede Neural Artificial

RNF – Requisito Não Funcional

UML – *Unified Modeling Language*

LISTA DE SÍMBOLOS

η - etá

SUMÁRIO

1 INTRODUÇÃO

Sistemas de assistência à navegação automobilística têm recebido cada vez mais a atenção de empresas e instituições. Segundo Centro de Informações do Galileo (2005, p. 2), espera-se que metade dos veículos operando na Europa tenha, até 2020, um sistema avançado de assistência ao motorista.

> Até 2010 haverá mais de 670 milhões de automóveis, 33 milhões de ônibus e caminhões e 200 milhões de veículos comerciais leves no mundo. Receptores de navegação por satélite são hoje normalmente instalados em novos automóveis como uma ferramenta essencial no fornecimento de novos serviços para pessoas em movimento: pagamento eletrônico, informação do trânsito em tempo real, chamadas de emergência, guia de rotas, gerenciamento de frotas e sistemas avançados de assistência ao motorista. (CENTRO DE INFORMAÇÕES DO GALILEO, 2005, p. 1).

Cunha, Jacques e Cybis (2006, p. 11) afirmam que a sinalização vertical apresentada em três itens, “presença de placas indicativas de velocidade máxima”, “presença de placas de sinalização de trânsito” e “legibilidade das placas de trânsito”, revelaram não exercer influência na velocidade praticada pelos motoristas entrevistados em sua pesquisa. Segundo Cunha, Jacques e Cybis (2006, p. 12), dentre as características que mais fortemente influenciam os motoristas entrevistados em sua pesquisa a reduzir a velocidade dos seus veículos estão: irregularidades no pavimento; cruzamento com outras vias; presença de curvas; presença de pontos de parada de ônibus e fiscalização eletrônica na via.

Alguns sistemas de assistência ao motorista analisam o ambiente em que haverá a locomoção do veículo através da aquisição e processamento de imagens a fim de auxiliar ou automatizar a navegação do veículo. Dependendo do tipo de informação que se espera extrair da imagem, tais como placas de sinalização, técnicas diferentes podem ser aplicadas. Analisando aspectos das imagens tais como iluminação, cor e forma, pode-se detectar e reconhecer placas de trânsito.

Neste trabalho propõe-se desenvolver uma ferramenta de detecção e reconhecimento de placas de trânsito, que forneça instruções em tempo real para sistemas de apoio ao motorista.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver uma ferramenta que detecte e reconheça placas de trânsito em uma imagem.

Os objetivos específicos do trabalho são:

a) disponibilizar um módulo de análise de imagem de via de trânsito em busca de placas de trânsito;
b) disponibilizar um módulo de destaque e identificação de placas de trânsito detectadas.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está estruturado em quatro capítulos. O segundo capítulo contém a fundamentação teórica necessária para o entendimento do trabalho. Nele são discutidos tópicos relacionados a sistemas de apoio ao motorista, visão computacional, detecção e reconhecimento de placas de trânsito. Também serão discutidas as tecnologias Java Advanced Imaging (JAI), Java Media Framework (JMF) e o *framework* de rede neural Neuroph. Por fim, apresentam-se quatro trabalhos correlatos.

O terceiro capítulo trata do desenvolvimento da ferramenta, onde são explanados os principais requisitos do problema trabalhado, a especificação contendo diagramas de caso de uso, classe e seqüência. Também são feitos comentários sobre a implementação abrangendo as técnicas e ferramentas utilizadas, operacionalidade e por fim são comentados os resultados e discussão.

O quarto capítulo refere-se às conclusões do presente trabalho e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os assuntos e técnicas utilizadas para o desenvolvimento e entendimento deste trabalho. Na seção 2.1 são expostas informações sobre sistemas de apoio ao motorista. Na seção 2.2 são explanados conceitos a respeito de visão computacional e são destacados aspectos referentes à detecção e reconhecimento de placas de trânsito. Na seção 2.3 é abordado a rede neural artificial utilizada. Na seção 2.4 são sintetizadas as tecnologias utilizadas. Por fim, na seção 2.5, são descritos trabalhos correlatos.

2.1 SISTEMAS DE APOIO AO MOTORISTA

Segundo Centurion (2009), através de estatísticas da Polícia Rodoviária Federal (PRF), quase 30% dos acidentes nas estradas ocorrem entre 17h e 20h, período de transição entre dia e noite. Centurion (2009) afirma que algumas das razões são: a dificuldade natural do ser humano para enxergar em situações de pouca luz, iluminação pública deficiente e o desrespeito às leis de trânsito. “Em condições de pouca luminosidade, a capacidade de enxergar é reduzida em até 30%. [...] A perda de noção de distância e profundidade está entre os riscos para os condutores de veículos.” (CENTURION, 2009).

O desenvolvimento de sistemas de apoio ao motorista, para auxiliar o condutor e informá-lo sobre as condições da via, pode contribuir significativamente para a diminuição do número de acidentes. Estes sistemas devem responder em tempo real para serem viáveis e algumas das principais tecnologias empregadas para garantir este requisito são: posicionamento global, radares, *scanners*, sensores e a visão computacional

Os *Advanced Driver Assistance Systems*[1] (ADAS) podem ser divididos de acordo com sua funcionalidade em:

a) navegação por posicionamento global: utiliza sistema de mapas e rotas sincronizadas com informações geográficas recebidas do *Global Positioning System* (GPS) em tempo real, fornecendo informações diversas relacionadas com o trajeto entre origem e destino selecionados pelo motorista;

b) assistência para estacionamento do veículo: utiliza sensores ultra-sônicos e/ou imagens com linhas de referência sobrepostas no painel, para indicar a proximidade de obstáculos ao condutor, conforme a Figura 1. Sistemas atuais controlam ativamente o volante, restando ao motorista controlar o avanço do veículo;

Fonte: Jung et al. (2005, p. 1370).

Figura 1 – Assistência para estacionamento do veículo - ADAS

c) iluminação por faróis adaptativos: utilizam sensores para regular a intensidade da iluminação dos faróis de acordo com as condições externas de luminosidade e clima, e a direção dos faróis acompanhando as curvas e intensificando a iluminação delas;

d) controle de velocidade adaptativo: podem ser diversas combinações de tecnologias como (i) utilizando um sistema de detecção e reconhecimento de placas de trânsito para identificar placas de limite de velocidade; (ii) utilizando comunicação com sistemas de bordo de outros veículos nas proximidades para obtenção de informações de GPS e velocidade e, (iii) utilizando informações de GPS para detectar proximidade de curvas, cruzamentos ou outras características de risco na aproximação do veículo;

e) detecção de colisão: podem utilizar visão computacional, radares ou sensores para detectar pedestres e outros objetos no caminho ou em ponto cego, acionando os freios em caso de colisão eminente;

[1] É o termo genérico em inglês, que significa Sistemas Avançados de Assistência ao Motorista, mais comumente utilizado que equivale ao termo genérico em português: sistemas de apoio ao motorista.

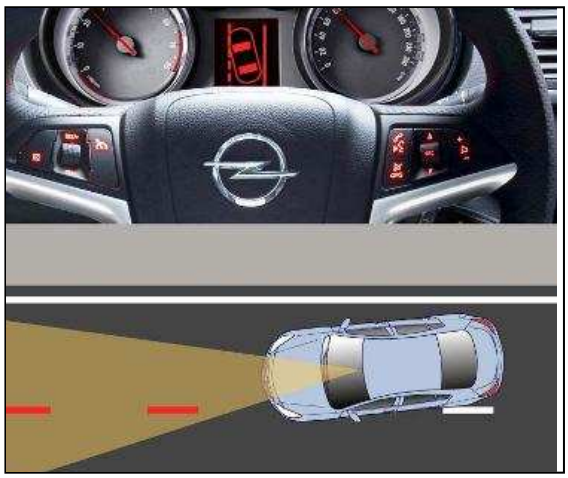
f) visão noturna: utilizam sistema de visão infravermelha para destacar pedestres e obstáculos na escuridão onde a iluminação dos faróis do veículo não é suficiente, conforme a Figura 2. A visão noturna pode ser complementada com a visão de calor, destacando ainda mais pedestres e animais;

Fonte: Ulf (2010).

Figura 2 – Visão noturna - ADAS

g) detecção de sonolência do condutor: utilizam algoritmos de visão computacional especializados em analisar o rosto do motorista a fim de detectar a sonolência, percebida pelo tempo em que as pálpebras permanecem fechadas além do normal;

h) detecção de saída de pista: podem utilizar visão computacional ou sensores para detectarem as bordas da pista e alertar sobre uma saída de pista não sinalizada pelo motorista através do acionamento do pisca-seta. Um sistema de detecção e reconhecimento de placas de trânsito para identificar placas de proibição de ultrapassagem pode complementar o sistema de apoio ao motorista e avisar sobre uma mudança de pista ilegal. A Figura 3 ilustra a detecção de saída de pista.



Fonte: Kempf (2008).

Figura 3 – Detecção de saída de pista - ADAS

2.2 VISÃO COMPUTACIONAL

Wangenheim (1998) define visão computacional como o conjunto de métodos e técnicas através dos quais sistemas computacionais podem ser capazes de interpretar imagens, ou seja, capazes de transformar um conjunto de dados digitais representando uma imagem em uma estrutura de dados descrevendo a semântica deste conjunto em um contexto qualquer.

A visão computacional se inicia pela aquisição da imagem, normalmente através de câmeras. A maioria dos sistemas utiliza visão monocular, ou seja, apenas uma câmera está instalada, podendo ainda ser monocromática ou colorida.

A etapa de pré-processamento encarrega-se de aplicar filtros para melhorar a imagem e transformá-la para a etapa seguinte. Alguns exemplos de transformações:

a) redução de número de cores, por exemplo, para paleta RGB de três bits (oito cores);
b) conversão para escala de cinza;
c) escala, rotação ou deslocamento de pixels;
d) remoção de ruídos;
e) ajuste de brilho ou contraste.

Na etapa de segmentação, são extraídas informações das imagens, tais como regiões que atendem determinadas características necessárias para a etapa de reconhecimento. Exemplos de técnicas de segmentação:

a) algoritmos genéticos;
b) limiarização;
c) detecção de bordas;
d) crescimento de regiões.

Por fim, na etapa de reconhecimento, segundo Kubiça (1999), as regiões segmentadas são classificadas, segundo suas características, através de uma das abordagens básicas: estatística, estrutural ou neural.

As principais aplicações da visão computacional em ADAS são: detecção de saída de pista, detecção de obstáculos, navegação visual e detecção e reconhecimento de placas de trânsito.

2.2.1 Detecção e reconhecimento de placas de trânsito

As placas de trânsito, em geral, possuem cor, geometria e tamanhos específicos. No Brasil, a cor predominante das placas de advertência é a amarela e das placas de regulamentação é a vermelha, definidas pelo COnselho Nacional de TRÂNsito (CONTRAN).

Segundo Jung et al. (2005), a informação cromática tem papel fundamental no contexto da detecção das placas de trânsito. Desta maneira assume-se que a maioria dos algoritmos desenvolvidos em nível mundial explora fortemente a detecção de placas pela cor, porém há informações de sistemas que utilizam imagens monocromáticas e exploram fortemente a detecção de placas pela geometria.

Sistemas de apoio ao motorista, com foco na detecção e reconhecimento de placas, são compostos por pelo menos uma câmera de vídeo (visão monocular), para a aquisição de imagens, e um sistema de *software* embarcado, ambos integrados ao computador de bordo do veículo. Os locais preferidos para a instalação da câmera são atrás do pára-brisa ou atrás da grade frontal do veículo.

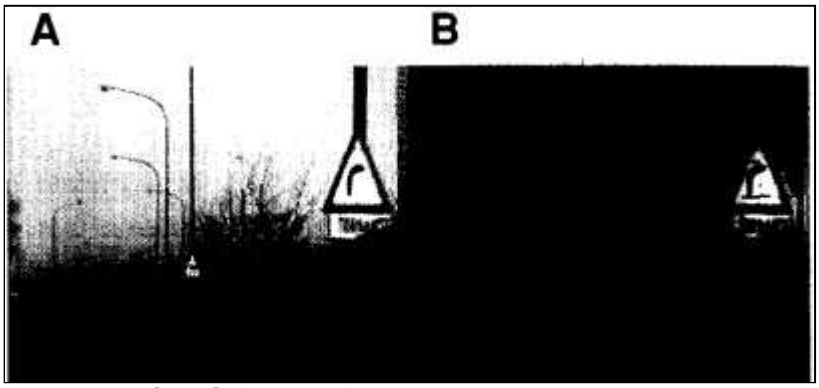
Após a captura da imagem, a mesma pode ser processada por *hardware* dedicado ao processamento de imagens ou por algoritmos implementados no *software*, a fim de detectar e reconhecer as placas de trânsito que seguem o padrão desejado. Segundo Jung et al. (2005), diversos pesquisadores no mundo inteiro vêm propondo soluções diferentes para a tarefa de detecção e reconhecimento de placas de trânsito.

Na solução proposta por Escalera, Armingol e Mata (2003), para a etapa de detecção, os autores optaram por analisar a cor e o formato, conforme a Figura 4. Segundo Escalera, Armingol e Mata (2003, p. 4), o uso da análise de cor é fundamental porque os sinais de trânsito são projetados pensando em usar cores de modo a refletir a mensagem do sinal. Dessa forma, as cores escolhidas destacam-se do ambiente. Na análise da cor são identificados matiz e saturação a fim de encontrarem áreas candidatas para a análise do formato, esta feita através de algoritmos genéticos. Na etapa de reconhecimento são utilizadas redes neurais para classificar as placas previamente detectadas.

Fonte: Escalera, Armingol e Mata (2003, p. 4).

Figura 4 – Utilização das informações de cor por Escalera, Armingol e Mata (2003)

Piccioli et al. (1996) propõe, para a etapa de detecção, que a busca das placas nas imagens seja limitada a uma região conhecida, tal como o lado direito da via, ou a uma região onde ocorrerem as cores pré-determinadas, quando estas estiverem presentes na imagem, conforme a Figura 5. Segundo Piccioli et al. (1996, p. 2), as informações sobre cor fornecem a maneira mais rápida para reduzir a procura de sinais de trânsito a uma pequena região da imagem, e são usadas sempre que possível. O uso de cores para segmentação de imagem corresponde ao modo como os sinais de trânsito são geralmente vistos por motoristas.



Fonte: Piccioli et al. (1996, p. 3).

Figura 5 – Utilização das informações de cor por Piccioli et al. (1996)

A detecção das placas candidatas é efetuada pela análise de borda nas regiões selecionadas. Um filtro de Kalman é utilizado para rastrear as placas detectadas com a finalidade de tornar a solução mais robusta. A etapa de reconhecimento é realizada pela comparação das placas detectadas com um banco de dados.

2.2.2 Quantização de cores

Segundo Chang (2008, p. 20), quantização de cores é um processo que reduz o número de cores distintas em uma imagem, normalmente através da minimização da profundidade da cor.

Existem diversos algoritmos que procuram preservar a similaridade das cores, entre a imagem original e a imagem quantizada, através de cálculos em abordagens diversas. A Figura 6 demonstra um exemplo de quantização uniforme, conforme o Quadro 1, onde a Figura 6(a) utiliza paleta de 16,7 milhões de cores e a Figura 6(b) utiliza paleta quantizada (Figura 7) para 8 cores, onde cada canal está representado por um bit e “0 ou 1” é o mesmo que “0 ou 255”.

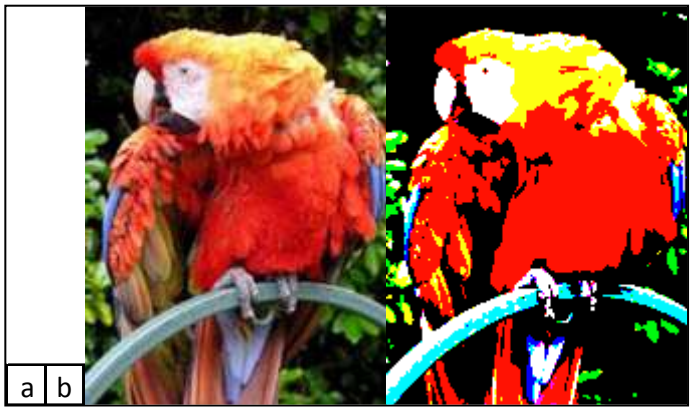


Figura 6 – Exemplo de quantização uniforme

```
Para todo pixel da Figura 1(a)
      Para cada canal de cor RGB
            Se a(x,y) < n
                  então b(x,y) = 0
                  senão b(x,y) = 255
onde: x e y são as coordenadas do pixel
      n é o nível de quantização
```

Quadro 1 – Algoritmo de quantização uniforme

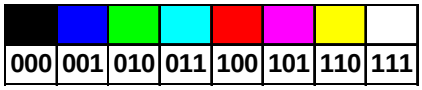


Figura 7 – Paleta RGB quantizada

2.2.3 Transformada circular de Hough

A transformada de Hough, segundo Pedrini e Schwartz (2008, p. 174-182), é uma técnica de segmentação para detecção de um conjunto de pontos em uma imagem digital que

pertençam a uma forma geométrica parametrizada, assim como linhas, círculos, elipses e outras. Mesmo em imagens muito ruidosas ou que possuem formas geométricas com regiões descontinuas, a transformada de Hough ainda permite detectar estas formas com sucesso.

Para simplificar o cálculo da transformada de Hough, a imagem pode ser convertida de colorida para escala de cinza e então submetida a uma técnica de detecção de borda, através do operador de Sobel, que calcula a informação de gradiente, conforme Pedrini e Schwartz (2008, p. 179).

Conforme Pedrini e Schwartz (2008, p. 174-182), o círculo pode ser formulado em equações paramétricas tais como no Quadro 2.

1) coordenadas cartesianas	$(x-a)^2+(y-b)^2=r^2$
2) coordenadas polares	$x = a + r\cos\Theta$
	$y = b + r\cos\Theta$
3) coordenadas polares resolvidas para os parâmetros do círculo	$a = x - r\cos\Theta$
	$b = y - r\cos\Theta$
4) união das equações 2 e 3	$b = a\tan\Theta - x\tan\Theta + y$

Fonte: adaptado de Pedrini e Schwartz (2008, p. 178-179).

Quadro 2 – Equações

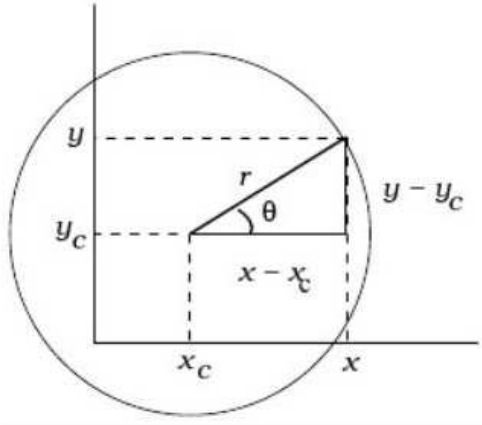
Segundo Pistori, Pistori e Costa (2005, p. 2-5), a transformada circular de Hough pode ser calculada tomando a informação do gradiente a partir de uma imagem em preto e branco e acumulando cada ponto não nulo a partir da imagem gradiente em todo ponto que está a um determinado raio de distância dele, conforme o Quadro 3.

Algoritmo 1 Criação do espaço de Hough

entrada: Matriz I, $n \times m$, representando a imagem binarizada.

saída: Matriz H, com o mesmo tamanho da imagem, representando o espaço de Hough.

```
1:  para x = 0 até n faça
2:    para y = 0 até m faça
3:      se I(x, y) = 255 então
4:        para θ = 0 até 2 * π faça
5:          x_c = x − r * cos(θ)
6:          y_c = y − r * sen(θ)
7:          H(x_c, y_c) = H(x_c, y_c) + 1
8:        fim para
9:      fim se
10:   fim para
11: fim para
```

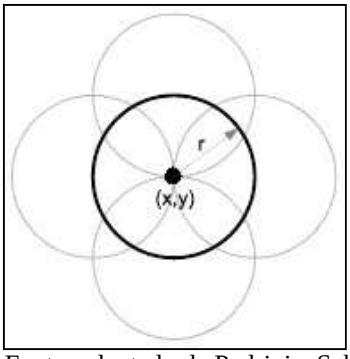


Fonte: adaptado de Pistori, Pistori e Costa (2005, p. 2-3).

Quadro 3 – Algoritmo da transformada circular de Hough

Dessa forma, todos os pontos de borda que se encontram ao longo do contorno de um círculo de um determinado raio, conforme a Figura 8, contribuem para a transformação no centro do círculo, e assim, os picos na imagem transformada correspondem aos centros de

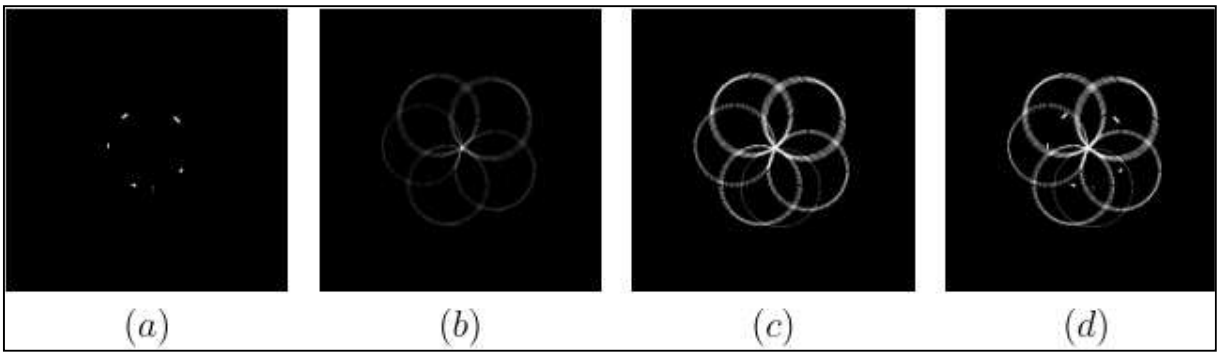
características circulares de um determinado tamanho na imagem original.



Fonte: adaptado de Pedrini e Schwartz (2008, p. 179).

Figura 8 – Operação de transformação circular de Hough

Depois que um pico é detectado e um possível círculo encontrado em um determinado ponto, pontos próximos (em uma distância dentro da metade do raio original) são excluídos como possíveis centros de círculo para evitar a detecção da mesma característica circular repetidamente. A Figura 9 exemplifica essa transformação, onde a Figura 9(a) é a imagem original contendo um círculo altamente corrompido, a Figura 9(b) seu respectivo espaço de Hough, a Figura 9(c) é o mesmo espaço de Hough da Figura 9(b) com ajuste de contraste para facilitar a visualização e na Figura 9(d) a imagem composta pela adição da imagem original da Figura 9(a) com o espaço de Hough da Figura 9(c).



Fonte: Pistori, Pistori e Costa (2005, p. 3).

Figura 9 – Exemplo de transformação circular de Hough

2.3 REDE NEURAL ARTIFICIAL

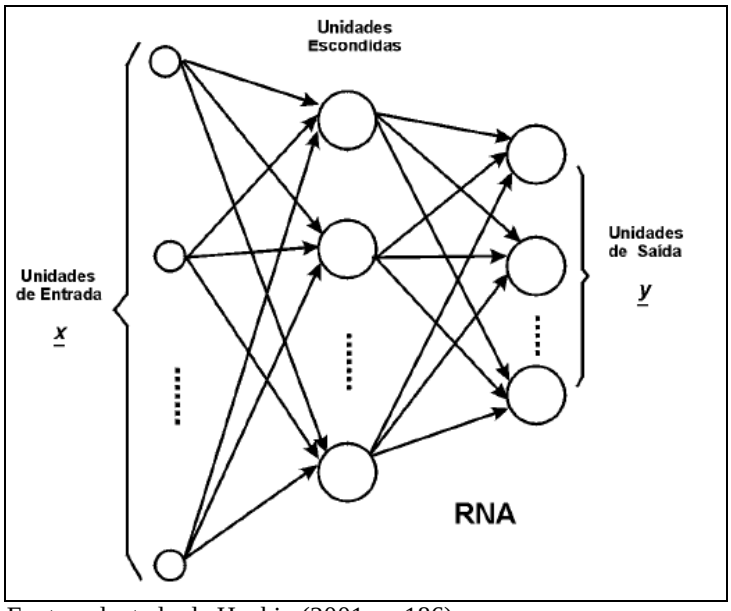
Esta seção apresenta a definição de Rede Neural Artificial (RNA) e, em particular, o modelo de rede neural *MultiLayer Perceptron* (MLP) e o algoritmo de aprendizado *BackPropagation* (BP).

2.3.1 Definição

Conforme Azevedo, Brasil e Oliveira (2000, p. 3), as redes neurais artificiais têm inspiração nos neurônios biológicos e nos sistemas nervosos. Segundo Abelém, Pacheco e Vellasco (1995, p. 109), redes neurais artificiais são sistemas compostos de diversos neurônios artificiais ligados de maneira apropriada para obter comportamentos complexos, que são determinados pela estrutura das ligações entre os neurônios e pelos pesos das conexões. Conforme Haykin (2001, p. 75), estes pesos, por sua vez, são ajustados para armazenar o conhecimento desejado através de um algoritmo de aprendizado, que consiste de um conjunto de regras bem definidas para, segundo Martinelli (1999, p. 13), resolver um problema de aprendizagem. Os algoritmos diferem entre si basicamente pela forma de ajuste dos pesos.

2.3.2 Modelo de rede neural MLP

Uma rede neural artificial do tipo MLP é constituída por um conjunto de nodos de entrada, os quais formam a camada de entrada na rede, que projeta os sinais de entrada sobre uma segunda camada oculta. Os sinais de saída dessa segunda camada podem ser enviados para outras camadas ocultas subseqüentes e assim sucessivamente até encontrarem a camada de saída, ou enviados diretamente para a camada de saída. Com exceção da camada de entrada, todas as outras camadas são constituídas por neurônios e, portanto, apresentam capacidade computacional (Figura 10). Uma rede desse tipo é alimentada adiante (*feedforward*) e pode ser totalmente conectada, quando todos os neurônios de uma camada se comunicam com todos os neurônios da próxima camada ou, parcialmente conectada, onde nem todos os neurônios de uma camada se comunicam com todos os neurônios da próxima camada (HAYKIN, 2001, p. 36).



Fonte: adaptado de Haykin (2001, p. 186).

Figura 10 – Arquitetura de uma rede neural artificial MLP

O número de nós na camada de entrada da rede é determinado pela dimensionalidade do espaço de observação, que é responsável pela geração dos sinais de entrada. O número de neurônios na camada de saída é determinado pela dimensionalidade requerida da resposta desejada. Segundo Castro (1994, p. 2), o projeto de uma rede MLP requer a consideração de três aspectos:

a) determinar o número de camadas ocultas;
b) determinar o número de neurônios em cada uma das camadas ocultas;
c) especificar os pesos sinápticos que interconectam os neurônios nas diferentes camadas da rede.

O primeiro e o segundo aspecto determinam a complexidade da rede MLP e o terceiro envolve a utilização de paradigmas e algoritmos de aprendizado supervisionado, sendo mais comumente usado o de retro propagação do erro, ou BP, segundo Haykin (2001, p. 183). Estudos desenvolvidos indicam que:

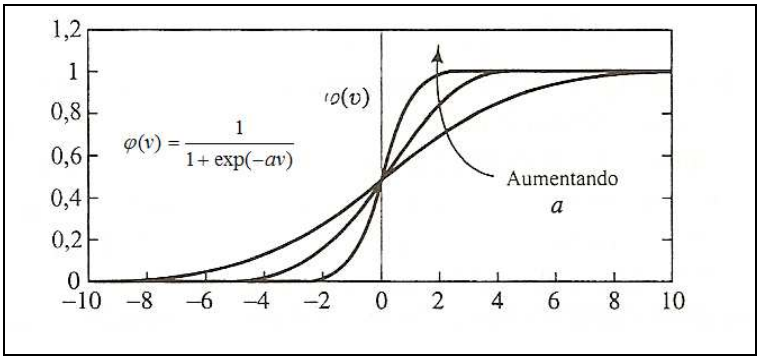
a) uma camada intermediária é suficiente para aproximar qualquer função contínua;
b) duas camadas intermediárias são suficientes para aproximar qualquer função matemática.

Segundo Haykin (1994, p. 178), uma rede MLP apresenta três características distintas:

a) incluir uma função não-linear. Uma comumente usada é a sigmoidal, definida pela função sigmóide logística, conforme a Figura 11, cuja não-linearidade é suave (ou seja, a função é diferenciável em qualquer ponto);
b) conter uma ou mais camadas de neurônios ocultos que não são parte da camada de

entrada ou de saída da rede. Estes neurônios ocultos possibilitam que a rede aprenda tarefas complexas, extraindo progressivamente mais características significativas dos padrões de entrada;

c) exibir um alto grau de conectividade, determinado pelas sinapses da rede. Uma mudança na conectividade da rede requer uma mudança na população de conexões sinápticas.



Fonte: adaptado de Haykin (2001, p. 39).

Figura 11 – Função sigmóide logística e seu gráfico

2.3.3 Algoritmo de aprendizado BP

O algoritmo de aprendizado BP segue o paradigma do aprendizado supervisionado (associativo) e baseia-se na heurística do aprendizado por correção de erro (em que o erro é retro-propagado da camada de saída para as camadas intermediárias da rede neural artificial). Este algoritmo pode ser considerado uma generalização do algoritmo conhecido como regra delta, ou algoritmo dos mínimos quadrados médios (*Least Mean-Square* – LMS), desenvolvido por Widrow e Hoff em 1960.

Segundo Han e Kamber (2000, p. 240), o algoritmo BP aprende iterativamente pelo processamento de um conjunto de exemplos de treinamento, comparando a resposta da rede para cada exemplo com o rótulo de classe atualmente conhecido. Para cada exemplo de treinamento, os pesos são modificados de forma a minimizar o erro entre a saída da rede e a classe atual. Estas modificações são feitas na direção reversa, ou seja, da camada de saída através das camadas ocultas até as primeiras camadas. Daí o nome BP, ou retro propagação.

Conforme Moraes, Nagano e Merlo (2002, p. 4), o erro (e_k) é calculado pela subtração do valor esperado (y_k) do valor obtido (d_k), conforme a equação do Quadro 4(a) e a atualização dos pesos flui de maneira inversa, através da equação do Quadro 4(b), em que a variação do peso da conexão (w) entre os neurônios (j) e (k) é o resultado, (η) é uma constante

positiva que determina a taxa de aprendizagem, (*e*) a taxa de erro e (*x*) o valor de entrada do neurônio subseqüente, todos em determinado tempo (*n*).

a $e_k(n) = d_k(n) - y_k(n)$

b $\Delta w_{kj}(n) = \eta \ e_k(n) x_j(n)$

Quadro 4 – Algoritmo BP

2.4 TECNOLOGIAS UTILIZADAS

Java Advanced Imaging (JAI) é uma interface projetada, com base na tecnologia Java 2D, segundo Sun Microsystems Inc. (1999a), para manipular e processar imagens de maneira fácil, flexível, escalável, extensível e com alto desempenho. Entre as principais funcionalidades, destacam-se:

a) *tiling* (ladrilhar): uma imagem é divida em recortes retangulares que podem ser armazenadas em memória e processadas separadamente;
b) execução retardada: um recorte pode ser processado apenas quando for solicitado pela aplicação;
c) regiões de interesse: partes não retangulares de uma imagem podem ser especificadas como objetos;
d) manipulação de tipos de arquivos: os principais formatos eletrônicos de arquivos podem ser manipulados através de *codecs*.

Java Media Framework (JMF) é um *framework* que, de acordo com Sun Microsystems Inc. (1999b), manipula mídias baseadas em tempo, tais como arquivos ou fluxos de áudio e vídeo, e suporta alguns dos padrões mais comuns do mercado.

Os fluxos de vídeo podem ser obtidos de placas de captura de vídeo, de *webcams* e de transmissões de vídeo por protocolo de rede, entre outras fontes. O JMF permite obter imagens estáticas de um vídeo pela definição de um intervalo de tempo ou através de outros eventos definidos pelo desenvolvedor.

O JMF estende-se com a utilização de *plug-ins* de processamento. Estes *plug-ins*

podem ser desenvolvidos para, por exemplo, aplicar um filtro detecção de borda na imagem.

Neuroph é um *framework* ainda em desenvolvimento que, segundo Sevarac et al. (2008), utilizando apenas tecnologia Java, possui recursos para criar e treinar redes neurais. Podem-se destacar as seguintes características:

a) pequeno número de classes essenciais, que podem ser facilmente reutilizadas;
b) estrutura e lógica fáceis de seguir;
c) suporte a reconhecimento de ótico de caracteres e imagens;
d) Suporte a treze arquiteturas de rede neural.

O pacote disponibilizado para *download* contém:

a) códigos fonte em Java;
b) bibliotecas;
c) documentação;
d) ferramenta gráfica para desenvolvimento de redes neurais, conforme Figura 12;
e) exemplos.

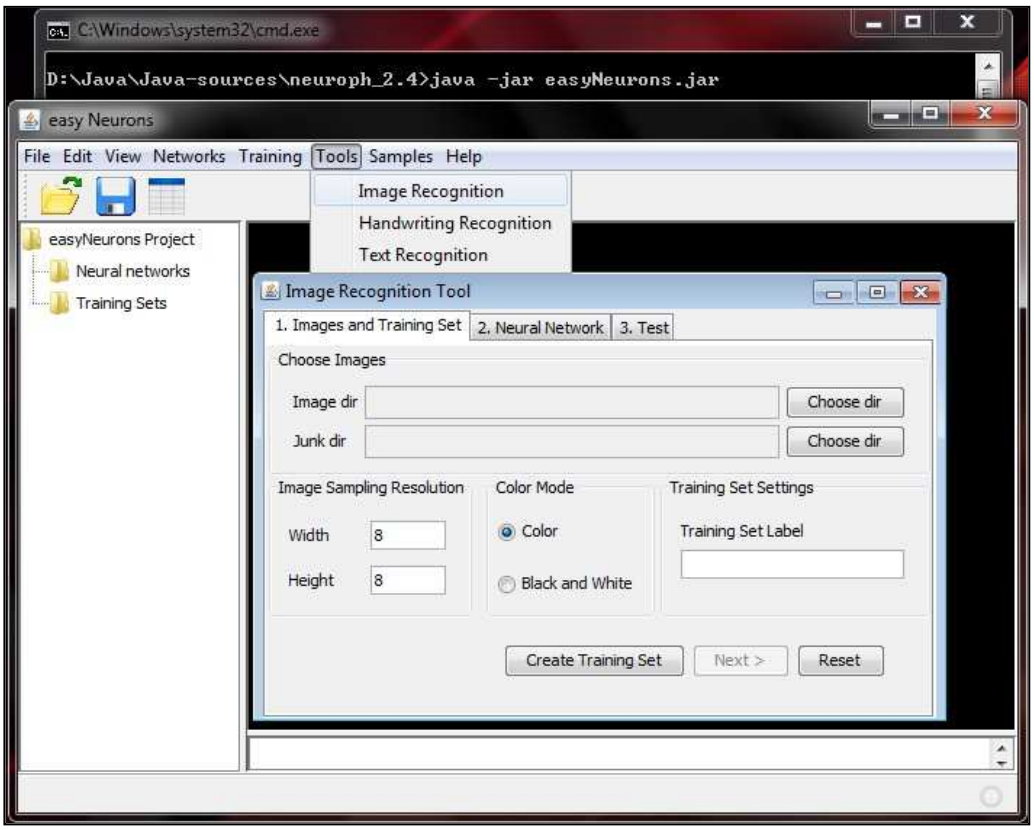


Figura 12 – Ferramenta gráfica para desenvolvimento de redes neurais

2.5 TRABALHOS CORRELATOS

Foram escolhidos quatro trabalhos que envolvem a visão computacional. O primeiro é o Opel Eye (KEMPF, 2008), sistema comercial desenvolvido por uma montadora de veículos. O segundo é um trabalho denominado “Localização e Reconhecimento de Placas de Sinalização Utilizando um Mecanismo de Atenção Visual e Redes Neurais Artificiais” (RODRIGUES, 2002). O terceiro é o “Sistema Óptico para Identificação de Veículos em Estradas” (SANTOS, 2008). Por fim, a “Ferramenta de Detecção da Região da Placa de Veículos” (MEIRELES, 2009).

2.5.1 Opel Eye

Segundo Kempf (2008), o Opel Eye é um sistema de apoio ao motorista, no qual uma das funções é o reconhecimento de placas de trânsito. O sistema estreou como equipamento opcional no lançamento do Insignia, modelo de carro da montadora General Motors (GM) lançado em 2008 na Europa.

O sistema conta com uma câmera com lente grande-angular de alta resolução, localizada junto ao espelho retrovisor central, capaz de captar trinta quadros por segundo e a uma distância de cem metros, em condições ideais de iluminação. A detecção é focada nas placas de proibição de ultrapassagem e limite de velocidade.

Após o quadro capturado ser analisado por dois processadores de sinais e o *software* criado pela GM, um aviso luminoso é exibido na parte central do painel de instrumentos, em frente ao volante do veículo.

2.5.2 Localização e reconhecimento de placas de sinalização utilizando um mecanismo de atenção visual e redes neurais artificiais

Rodrigues (2002, p. 27-94) afirma que o modelo proposto implementa técnicas de processamento de imagens utilizando mecanismo de atenção visual[2] e aprendizado de

[2] Segundo Rodrigues (2002, p. 2), atenção visual é o mecanismo responsável por selecionar as informações mais relevantes das imagens de entrada.

máquina. A arquitetura do modelo proposto possui os seguintes processos:

a) filtragem linear;
b) diferenças centro-vizinhança;
c) soma dos mapas de características;
d) soma dos mapas de conspicuidade;
e) seleção das regiões de interesse;
f) rede neural para reconhecimento da placa de trânsito.

Segundo Rodrigues (2002, p. 88-90), a taxa de acerto na detecção da placa de trânsito atingiu 100% e o reconhecimento atingiu a média, na análise absoluta, em torno de 64%.

2.5.3 Sistema óptico para identificação de veículos em estradas

Santos (2008, p. 6-34) desenvolveu um sistema, na linguagem C++, para contagem de automóveis em vias de tráfego, onde o algoritmo de segmentação adaptativa *Non-Homogeneous Detector* (NHD) analisou a imagem para a definição de elementos em movimento, para então detectar as bordas destes elementos e segmentar a imagem.

Segundo Santos (2008, p. 6-34), a representação da imagem segmentada é dada pela identificação de padrões através da análise das descrições de formas, utilizando descritores de Fourier. Na etapa de interpretação das representações foram utilizadas redes neurais.

2.5.4 Ferramenta de detecção da região da placa de veículos

A ferramenta implementada por Medeiros (2009), na linguagem Java, efetua a localização de uma placa de veículo e utiliza uma biblioteca de reconhecimento de caracteres para extrair e reconhecer os caracteres da placa na região localizada. A imagem é tratada através dos passos a seguir:

a) aplicação de filtro de Gauss para eliminação de ruídos;
b) aplicação de detecção de borda utilizando operador gradiente de Sobel;
c) a posição vertical da placa é identificada;
d) aplicação da operação de fechamento da morfologia matemática;
e) a posição horizontal da placa é identificada;
f) a região segmentada é submetida à biblioteca de reconhecimento de caracteres;

Após os passos acima, a ferramenta informa os caracteres reconhecidos pela biblioteca.

3 DESENVOLVIMENTO DO PROTÓTIPO

Neste capítulo são detalhadas as etapas do desenvolvimento, abordando os principais requisitos, a especificação, a implementação e por fim são listados resultados e discussão.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

A ferramenta deverá:

a) capturar uma imagem contendo um cenário da via à frente do veículo (Requisito Funcional - RF);
b) detectar a região da imagem onde cada placa de trânsito aparece (RF);
c) reconhecer a placa de trânsito de cada região detectada (RF);
d) informar o usuário caso não seja possível reconhecer a placa (RF);
e) informar o usuário para cada placa de trânsito reconhecida (RF);
f) implementar a ferramenta utilizando a tecnologia Java (Requisito Não Funcional - RNF).

3.2 ESPECIFICAÇÃO

A especificação do sistema apresentado utiliza alguns dos diagramas da *Unified Modeling Language* (UML) em conjunto com a ferramenta Enterprise Architect (EA) 7.1 para a elaboração dos diagramas de casos de uso, de classes e de seqüência. Alguns diagramas estão em sua forma resumida para melhor visualização.

3.2.1 Diagrama de casos de uso

A Figura 13 apresenta o diagrama de casos de uso com as principais interações do usuário com o protótipo.

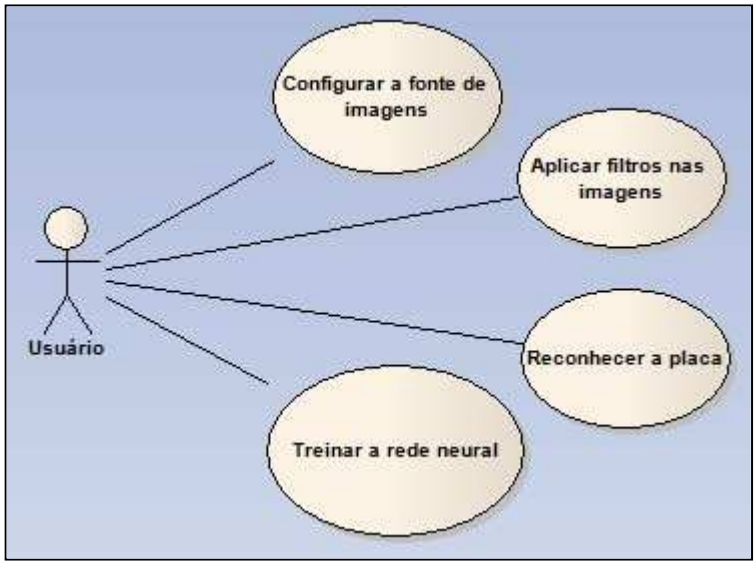


Figura 13 – Diagrama de casos de uso

3.2.1.1 Configurar a fonte de imagens

O caso de uso `Configurar a fonte de imagens` (Quadro 5) descreve a possibilidade do usuário configurar a origem das imagens a ser utilizada. Além do cenário principal, o caso de uso possui um cenário de exceção.

`Configurar a fonte de imagens: permite ao usuário configurar a origem das imagens a ser utilizada.`	
`Pré-condição`	`A fonte de imagens deve estar acessível ao protótipo.`

`Cenário principal`	`1) O usuário configura a fonte de imagens.`
	`2) O usuário executa o protótipo.`
	`3) O protótipo exibe as imagens.`

`Exceção`	`No passo 1, caso não exista nenhum fonte de imagens acessível, o protótipo apresenta uma mensagem de erro que não foi encontrado nenhuma origem.`
`Pós-condição`	`A fonte de imagens é configurada com sucesso.`

Quadro 5 – Caso de uso `Configurar a fonte de imagens`

3.2.1.2 Aplicar filtros nas imagens

O caso de uso `Aplicar filtros nas imagens` (Quadro 6) descreve a possibilidade do usuário definir a utilização de um ou mais filtros nas imagens a serem utilizadas.

Aplicar filtros nas imagens: permite ao usuário definir um ou mais filtros na fila de *codecs.*	
Pré-condição	Uma fonte de imagens estar configurada no protótipo.
Cenário principal	1) O usuário define os filtros de imagens. 2) O usuário executa o protótipo. 3) O protótipo exibe imagens modificadas pelos filtros.
Exceção	No passo 2, caso não exista nenhum filtro de imagens definido, o protótipo exibirá imagens sem alteração.
Pós-condição	Os filtros são aplicados com sucesso.

Quadro 6 – Caso de uso Aplicar filtros nas imagens

3.2.1.3 Treinar a rede neural

O caso de uso Treinar a rede neural (Quadro 7) permite que o usuário treine uma rede neural, podendo utilizar a ferramenta incluída no *framework* Neuroph.

Treinar a rede neural: permite que o usuário treine uma rede neural.	
Pré-condição	Possuir as imagens de placas, modificadas por filtros, para utilizar como conjunto de treinamento da rede neural.
Cenário principal	1) O usuário treina a rede neural. 2) O usuário testa a rede neural. 3) O usuário salva o arquivo da rede neural treinada.
Exceção	No passo 2, caso o teste não satisfaça com taxa de acerto aceitável, o usuário pode treinar novamente a rede, agora com outros parâmetros.
Pós-condição	A rede neural é treinada com sucesso.

Quadro 7 – Caso de uso Treinar a rede neural

3.2.1.4 Reconhecer a placa

O caso de uso Reconhecer a placa (Quadro 8) descreve a possibilidade do usuário reconhecer uma placa utilizando uma rede neural treinada.

Reconhecer a placa: permite que o usuário reconheça uma placa, utilizando uma rede neural treinada.	
Pré-condição	Possuir uma rede neural treinada.
Cenário principal	1) O usuário define as imagens, modificadas por filtros, para serem analisadas pela rede neural. 2) O usuário executa o protótipo. 3) O protótipo exibe o resultado do reconhecimento da placa.
Exceção	No passo 1, caso as imagens não estejam modificadas no mesmo padrão das imagens utilizadas no treinamento da rede neural, o protótipo não reconhecerá a placa.
Pós-condição	A placa é reconhecida com sucesso.

Quadro 8 – Caso de uso Reconhecer a placa

3.2.2 Diagrama de classes

O diagrama de classes apresentado na Figura 14 exibe as principais classes utilizadas no protótipo.

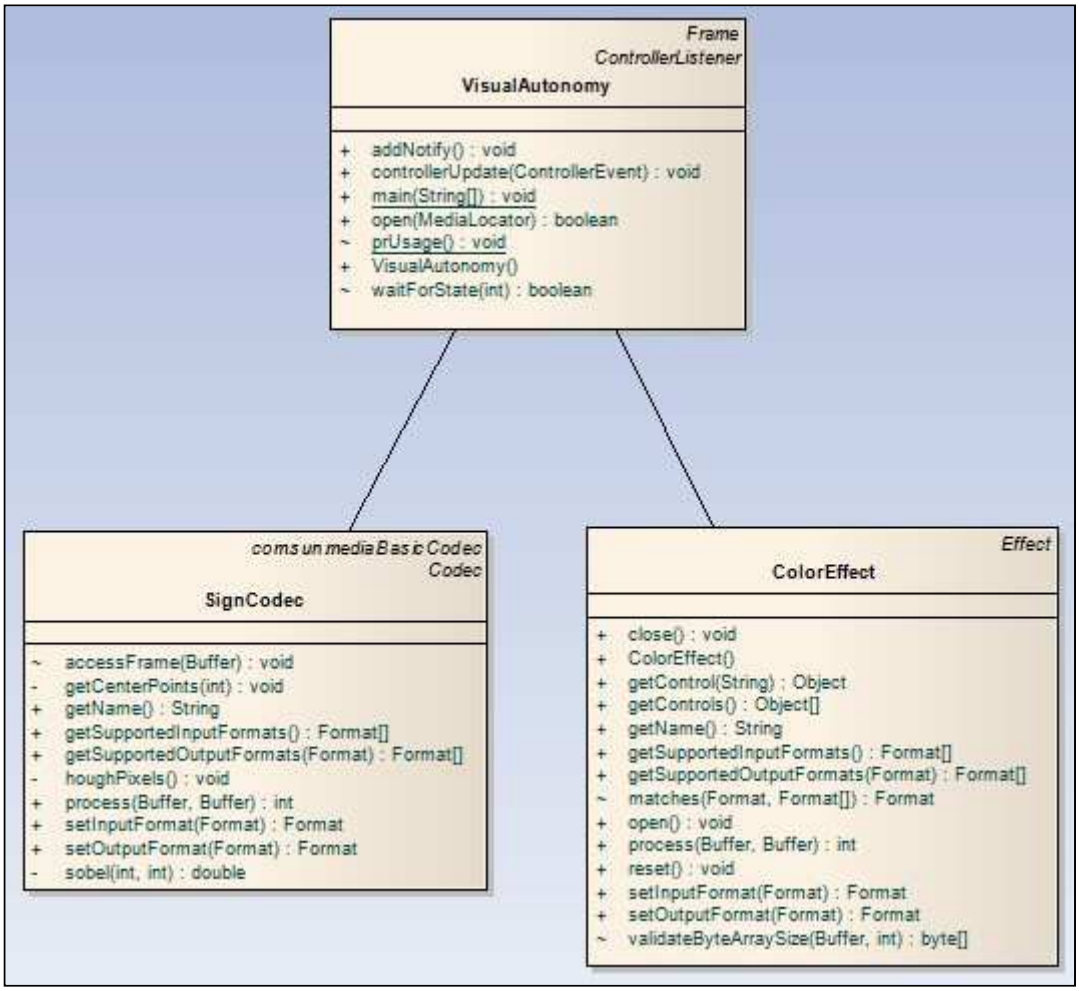


Figura 14 – Diagrama de classes

3.2.2.1 Classe `VisualAutonomy`

A classe `VisualAutonomy` é responsável pela interface gráfica do protótipo, exibindo para o usuário as imagens processadas pelas classes do tipo *codec* e *effect*. Esta classe também é responsável pelo método que carrega a fonte de imagens e pelo método que instancia as classes do tipo *codec* e *effect*.

3.2.2.2 Classe `ColorEffect`

Esta classe processa os efeitos que ajustam e filtram as cores das imagens. O primeiro ajuste feito é a quantização (ver seção 2.2.2), seguido da seleção das cores desejadas. Este processamento é efetuado através da conversão do *frame* em um *array* de *bytes* contendo, separadamente, os três canais de cores RGB. Cada *byte* representa um *pixel* em determinado canal de cor. Este *byte* é então analisado e modificado.

3.2.2.3 Classe `SignCodec`

Através da classe `SignCodec` é realizada a segmentação e reconhecimento da placa de trânsito. A segmentação é efetuada através da transformada circular de Hough (ver seção 2.2.3) e detecta uma possível placa em determinada posição da imagem, onde então o segmento é recortado e submetido ao reconhecimento, através da rede neural, pelo método `recognizeImage` da biblioteca Neuroph.

3.2.3 Diagrama de seqüência

O diagrama de seqüência, conforme a Figura 15, apresenta uma visão passo a passo, de forma resumida para uma melhor visualização, do processo de troca de mensagens entre as classes.

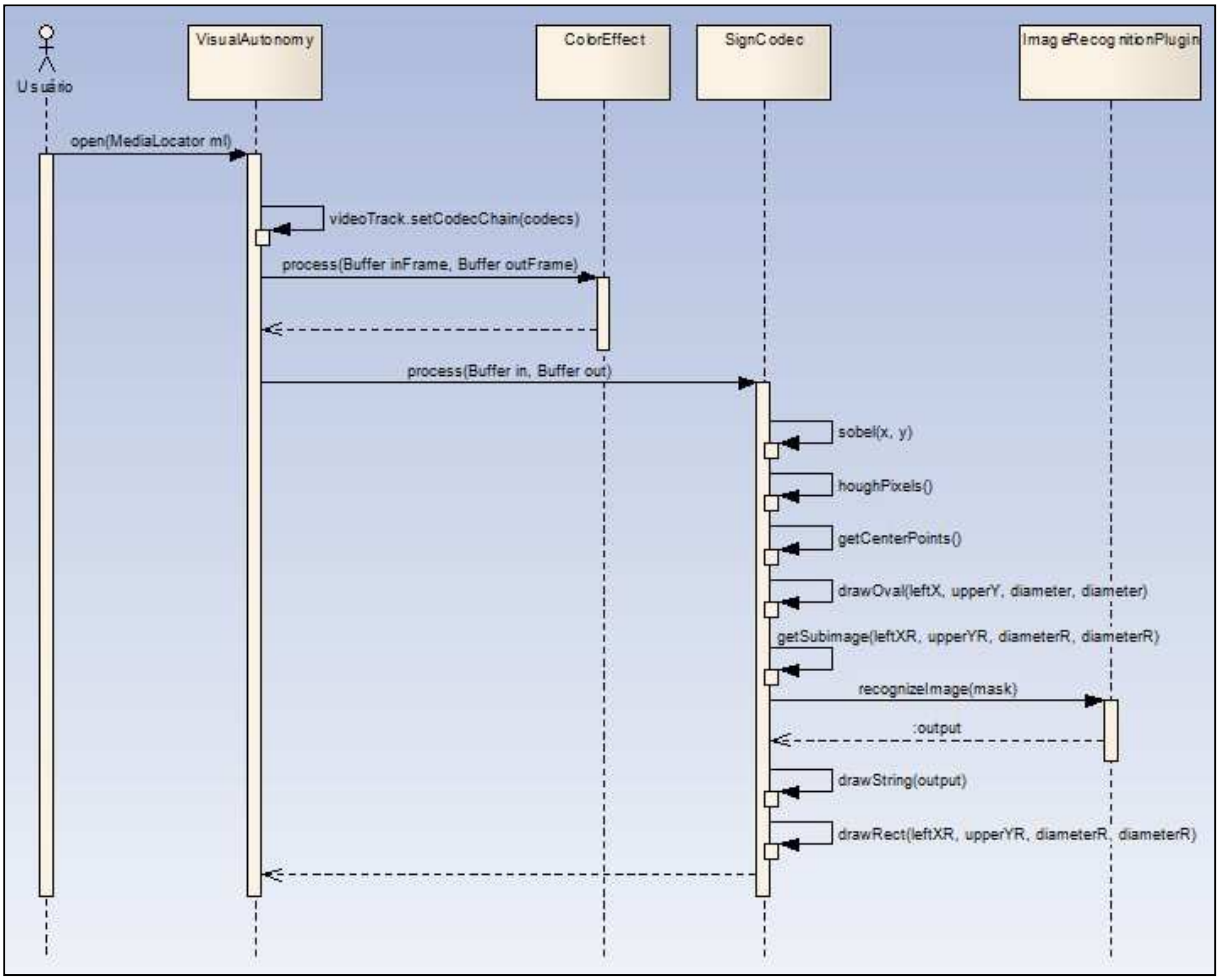


Figura 15 – Diagrama de seqüência

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas no desenvolvimento deste protótipo, serão apresentados trechos do código fonte desenvolvido. Também é demonstrada a operacionalidade da implementação, através de um estudo de caso.

3.3.1 Técnicas e ferramentas utilizadas

Para a implementação da ferramenta foi utilizada a linguagem de programação Java. O ambiente de desenvolvimento escolhido foi o Eclipse IDE. Foi utilizado também, as APIs JMF e JAI, e o *framework* de rede neural para Java Neuroph.

3.3.1.1 Instanciação de classes do tipo *codec* e *effect*

As classes do tipo *codec* e *effect* são carregadas para uma fila (corrente), conforme o Quadro 9, onde o *buffer* de imagens é transferido (a transferência é implícita) de um *codec* para o próximo, como ilustra a Figura 16.

```
125         try {
126             // Instantiate and set the frame access codec to the data flow path.
127             // Instanciar e definir o codec de acesso a frame para o caminho do fluxo de dados.
128             Codec codec[] = {
129                     new ColorEffect(),
130                     new SignCodec()
131             };
132             videoTrack.setCodecChain(codec);
```

Quadro 9 – Instanciação de *codecs*

Figura 16 – Fluxo do *buffer* de imagens do protótipo em execução

3.3.1.2 Pré ajuste das cores do fluxo de imagens

O fluxo de imagens é recebido pelo *codec* através de um *buffer* de *frames* e então separado por canais RGB, com 24 bits (3 canais de 8 bits) de profundidade de cor, como ilustrado no Quadro 10.

```
byte[] indata = (byte[]) inFrame.getData();

for(int index = 0; index < indata.length; index = index + 3)
    int blue =  indata[(int)(index)];
    int green = indata[(int)(index+1)];
    int red =   indata[(int)(index+2)];
```

Quadro 10 – Recebimento do fluxo de imagens

A necessidade do pré ajuste das cores deve-se aos valores recebidos em cada canal não estarem representados em um intervalo prático para a manipulação, como demonstrado na Figura 17.

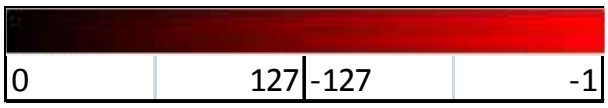


Figura 17 – Intervalo de valores recebido da cor vermelha

O pré-ajuste de cores encarrega-se de somar um valor constante aos valores negativos recebidos, conforme o Quadro 11, resultando em um intervalo mais adequado, conforme a

Figura 18.

```
if (blue < 0)   blue  = blue+255;
if (green < 0)  green = green+255;
if (red < 0)    red   = red+255;
```

Quadro 11 – Pré ajuste de cores

Figura 18 – Intervalo de valores ajustado da cor vermelha

Destaca-se que o pré ajuste não influencia nas cores apresentadas, sendo assim, quando se visualiza uma imagem antes do pré-ajuste e uma imagem depois do pré-ajuste, não se encontram diferenças.

3.3.1.3 Quantização uniforme de cores

Após o pré ajuste de cores, é efetuada a quantização (ver seção 2.2.2) da profundidade de cor de cada canal, conforme o Quadro 12, reduzindo de 256 níveis de cor para 2 níveis de cor em cada canal.

```
int bright = 128;
if (blue < bright)  blue = 0;
    else blue = 255;
if (green < bright) green = 0;
    else green = 255;
if (red < bright)   red = 0;
    else red = 255;
```

Quadro 12 – Quantização uniforme

O resultado da quantização é apresentado na Figura 19(b) em comparação com a imagem original apresentado na Figura 19(a).

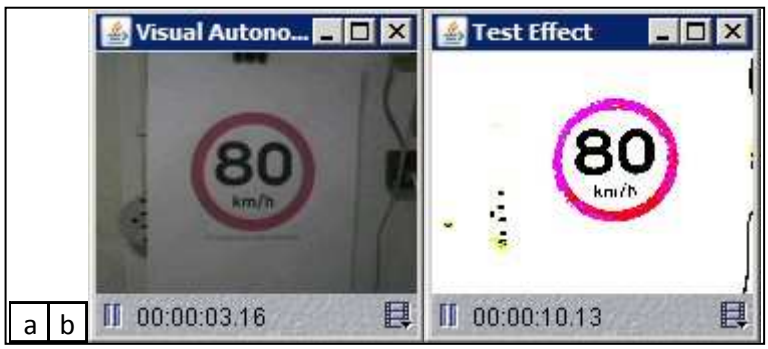


Figura 19 – Resultado da quantização

3.3.1.4 Seleção de cores

Observando o resultado obtido na Figura 19(b), notam-se as cores a serem selecionadas para contemplar a totalidade de detalhes da placa de trânsito. Estas cores (magenta, vermelho e preto) então são convertidas para a cor branca e as demais para a cor preta, conforme o Quadro 13.

```
if (((green == 0) && (red == 255)) ||
        ((blue == 0) && (green == 0) && (red == 0))){
    blue = 255;
    green = 255;
    red = 255;
}
else {
    blue = 0;
    green = 0;
    red = 0;
}
```

Quadro 13 – Seleção de cores

O resultado é apresentado na Figura 20(b).

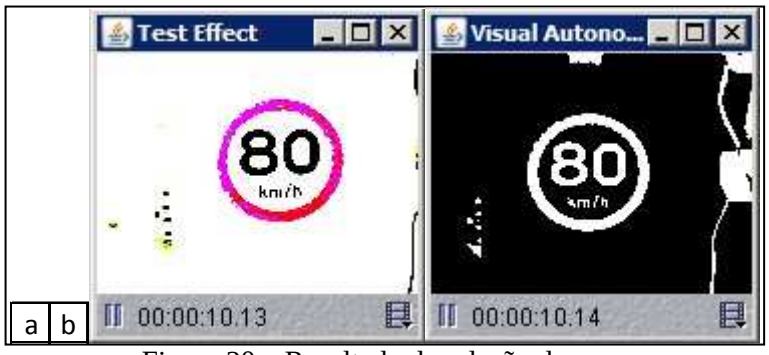


Figura 20 – Resultado da seleção de cores

3.3.1.5 Detecção da placa

A detecção da placa é realizada pelo algoritmo da transformada circular de Hough (ver seção 2.2.3) a partir da Figura 20(b), conforme o Quadro 15. O código fonte contido no Quadro 15, desde a linha 113 até a linha 175, foi adaptado do trabalho de Schulze (2003). A rede neural treinada é carregada a partir de um arquivo `nnet`, conforme o Quadro 14.

Os parâmetros utilizados na transformada de Hough também podem ser vistos no Quadro 14. O raio é determinado arbitrariamente em proporção pela área que a imagem vai

ocupar.

```
35          int[][] imageValues;
36          double[][] houghValues;
37          public int width;
38          public int height;
39          float radiusnet = 43;
40          float radius = 30;
41          float scale = radiusnet/radius;
42          float error = 0.500f;
43          float fps = 15f;
44          int[] pixels;
45          int maxCircles = 1;
46          Point centerPoint[];
47
48          // load trained neural network (specify existing neural network file here)
49          // carga da rede neural treinada (especifique aqui o arquivo de rede neural)
50          NeuralNetwork nnet = NeuralNetwork.load("H:\\Furb\\pn.nnet");
51
52          // get the image recognition plugin from neural network
53          // (image recognition interface for neural network)
54          // obtenha o plugin de reconhecimento de image da rede neural
55          // (interface de reconhecimento de imagem para a rede neural)
56          ImageRecognitionPlugin imageRecognition =
57              (ImageRecognitionPlugin)nnet.getPlugin(ImageRecognitionPlugin.IMG_REC_PLUGIN_NAME);
```

Quadro 14 – Parâmetros para detecção e reconhecimento da placa

```
110 BufferToImage stopBuffer = new BufferToImage((VideoFormat)in.getFormat());
111 Image stopImage = stopBuffer.createImage(in);
112 if(stopImage != null){
113     width = stopImage.getWidth(null);
114     height = stopImage.getHeight(null);
115     int pixels[] = new int[width * height];
116     PixelGrabber pixGrabber = new PixelGrabber(stopImage, 0, 0, width, height, pixels, 0, width);
117     try {
118         pixGrabber.grabPixels();
119     }
120     catch(InterruptedException e) {
121         System.out.println("exception thrown");
122     }
123     pixels = (int[])(pixGrabber.getPixels());//
124     imageValues = new int[width][height];
125     for(int y = 0; y < height; y++) {
126         for(int x = 0; x < width; x++)
127             imageValues[x][y] = pixels[y * width + x] & 0xff;
128     }
129     int numCirclePoints = 0;
130     int min = 1;
131     int maxW = width - min;
132     int maxH = height - min;
133     houghValues = new double[width][height];
134     int numPts = Math.round((float)8 * radius);
135     int circle[][] = new int[2][numPts];
136     for(int i = 0; i < numPts; i++) {
137         double theta = (6.2831853071795862D * (double)i) / (double)numPts;
138         int xx = (int)Math.round((double)radius * Math.cos(theta));
139         int yy = (int)Math.round((double)radius * Math.sin(theta));
140         if((numCirclePoints == 0) | (xx != circle[0][numCirclePoints]) &
141                 (yy != circle[1][numCirclePoints])) {
142             circle[0][numCirclePoints] = xx;
143             circle[1][numCirclePoints] = yy;
144             numCirclePoints++;
145         }
146     }
147
148     for(int y = min; y < maxH; y++) {
149         for(int x = min; x < maxW; x++) {
150             double tempValue = sobel(x, y);
151             if(tempValue == (double)0)
152                 continue;
153             for(int i = 0; i < numCirclePoints; i++) {
154                 int xCoord = x + circle[0][i];
155                 int yCoord = y + circle[1][i];
156                 if((xCoord >= 0) & (xCoord < width) & (yCoord >= 0) & (yCoord < height))
157                     houghValues[xCoord][yCoord] += tempValue;
158             }
159         }
160     }
161     houghPixels();
162     centerPoint = new Point[maxCircles];
163     getCenterPoints(maxCircles);
164
165     Graphics sg = stopImage.getGraphics();
166     sg.setColor(Color.red);
167     int diameter = Math.round((float)2 * radius);
168     //for(int i = 0; i < maxCircles; i++) {
169         int leftX = centerPoint[0].x - Math.round(radius);
170         int upperY = centerPoint[0].y - Math.round(radius);
171         int leftXR = leftX-2;
172         int upperYR = upperY-2;
173         int diameterR = diameter+4;
174         sg.drawOval(leftX, upperY, diameter, diameter);
175     //}
```

Quadro 15 – Detecção da placa

O resultado pode ser visto na Figura 21(b), onde é desenhado um círculo vermelho ao redor da placa detectada.

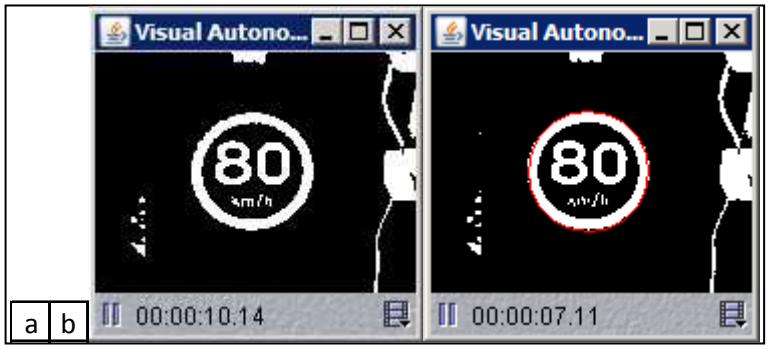


Figura 21 – Placa detectada

3.3.1.6 Reconhecimento da placa

O processo de reconhecimento da placa começa analisando se a placa detectada está contida totalmente dentro da imagem, para então extrair a região onde foi detectada a placa. Esta região é redimensionada para o tamanho das imagens que foram utilizadas no treinamento da rede neural e então submetida ao método `recognizeImage` que é proveniente do *framework* Neuroph, conforme Quadro 16.

```
186     PlanarImage jaiImage = JAI.create("AWTImage", stopImage);
187     BufferedImage bim = jaiImage.getAsBufferedImage();
188     BufferedImage mask = null;
189     if (diameterR+leftXR < width && diameterR+upperYR < height && leftXR > 0 && upperYR > 0){
190         mask = bim.getSubimage(leftXR, upperYR, diameterR, diameterR);
191 
192         AffineTransform af = new AffineTransform();
193         af.scale(scale, scale);
194         AffineTransformOp op = new AffineTransformOp(af, AffineTransformOp.TYPE_BILINEAR);
195         mask = op.filter(mask, null);
196 
197         // image recognition is done here (specify some existing image file)
198         // reconhecimento de imagem é feito aqui (especifique alguma imagem existente)
199         HashMap<String, Double> output = imageRecognition.recognizeImage(mask);
200         sg.setColor(Color.green);
201         sg.drawString(output.toString().substring(3,8),10,10);
202         if (Float.valueOf(output.toString().substring(3,8)).floatValue() > error){
203             sg.setColor(Color.blue);
204             sg.drawRect(leftXR, upperYR, diameterR, diameterR);
205         }
206     }
207     sg.dispose();
208     ImageToBuffer outImageBuffer = new ImageToBuffer();
209     Buffer outBuffer = outImageBuffer.createBuffer(stopImage,fps);
210     in.copy(outBuffer);
211 }
```
Quadro 16 – Reconhecimento da placa

O retorno desse método é então tratado para a extração do valor correspondente a taxa de similaridade entre a imagem submetida e a imagem treinada. Se o valor retornado ultrapassar o valor escolhido como aceitável, desenha-se um retângulo quadrado correspondendo a região submetida à rede neural, conforme a Figura 22(b). O retorno tratado

também é exibido na Figura 22, no canto superior esquerdo, na cor verde.

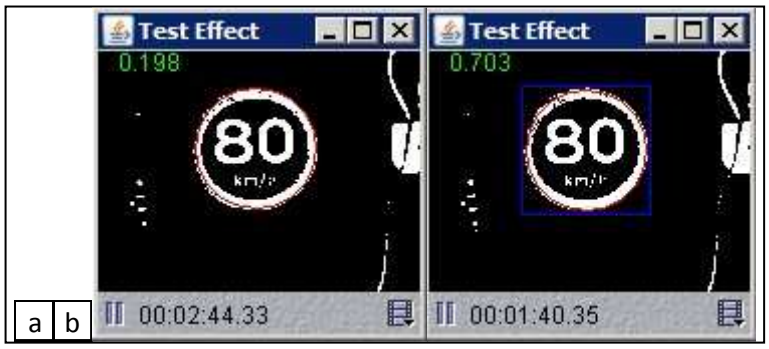


Figura 22 – Placa reconhecida

3.3.2 Operacionalidade da implementação

Esta seção tem como objetivo mostrar a operacionalidade da ferramenta em nível de usuário. Nas próximas seções serão abordadas as principais funcionalidades da ferramenta.

3.3.2.1 Funcionamento do protótipo

Ao se iniciar o protótipo, o primeiro dispositivo de captura de vídeo que o protótipo detectar será carregado automaticamente e aparecerá uma tela, conforme a Figura 23, do protótipo já em funcionamento.

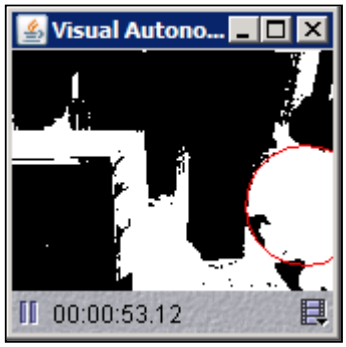


Figura 23 – Tela do protótipo em funcionamento

3.3.2.2 Treinando uma rede neural

Para o treinamento da rede neural, foi utilizada a ferramenta easyNeurons, presente no

framework Neuroph, conforme Figura 24.

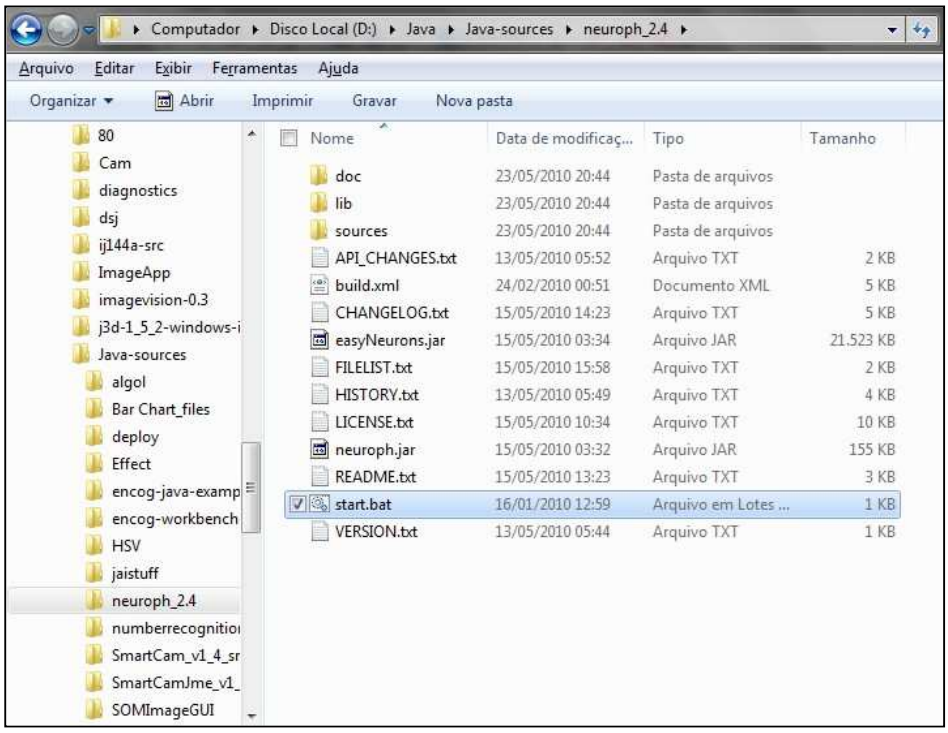


Figura 24 – Ferramenta easyNeurons

Para abrir a ferramenta basta executar o arquivo `start.bat`. Este arquivo irá carregar a ferramenta, desenvolvida em Java, conforme a Figura 25.

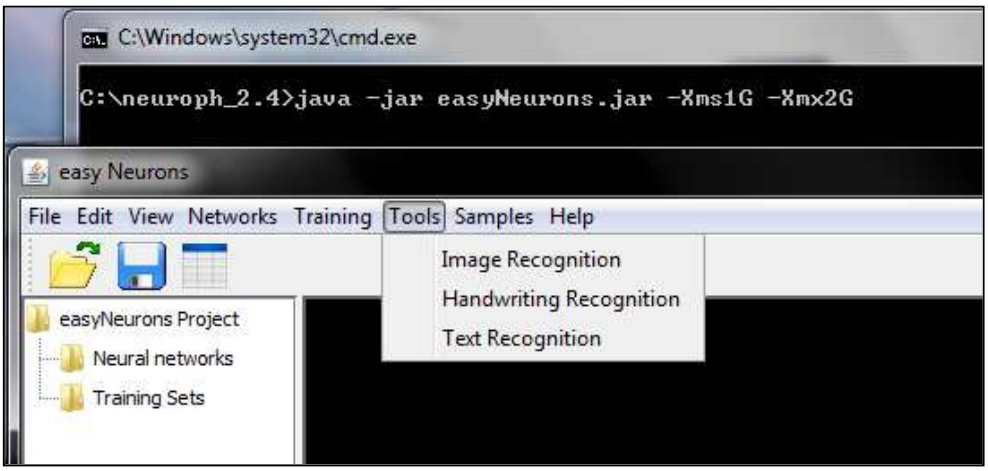


Figura 25 – Tela principal da ferramenta easyNeurons

A tela de treinamento para reconhecimento de imagens, conforme a Figura 26, é acessada através do menu `Tools`, submenu `Image Recognition`, conforme a Figura 25.

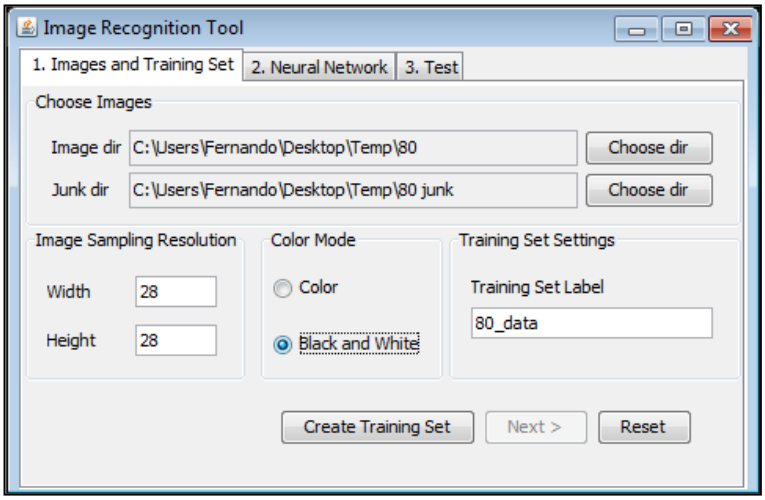


Figura 26 – Tela de entrada de dados para a criação do conjunto de treinamento

O botão `Choose dir`, ao lado do campo `Image dir`, seleciona a pasta onde a imagem da placa, que será reconhecida pela RNA, está armazenada. O botão `Choose dir`, ao lado do campo `Junk dir`, seleciona a pasta onde a imagem da placa, que será ignorada no reconhecimento, está armazenada. Nos campos `Width` e `Height` são digitados, respectivamente, os valores em *pixels*, da largura e altura para redimensionamento automático das imagens presentes nas pastas informadas. Existe ainda a opção de escolher o modo de cor destas imagens.

Após a entrada dos dados, é acionado o botão `Create Training Set` para a criação do conjunto de treinamento baseado nos dados fornecidos e para prosseguir para a tela de criação da rede neural, conforme Figura 27, clica-se no botão `Next`.

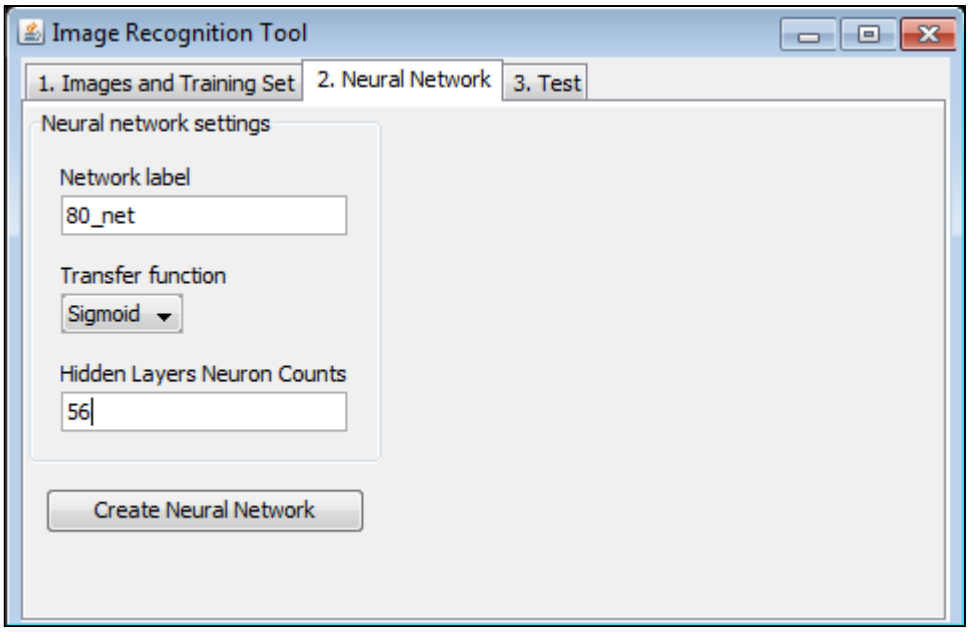


Figura 27 – Tela de entrada de dados para criação da rede neural

A principal informação inserida na tela da Figura 27, é o número de neurônios ocultos.

Para inserir mais de uma camada, podem-se digitar os números de neurônios para cada camada separados com um espaço. O botão `Create Neural Network` cria a rede neural e abre a tela para início do treinamento, conforme Figura 29. A Figura 28 mostra os conjuntos de treinamento e as redes neurais criadas.

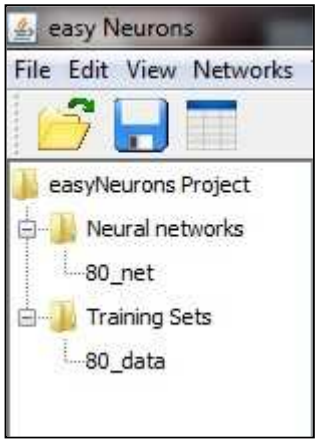


Figura 28 – Estrutura do projeto de rede neural em andamento

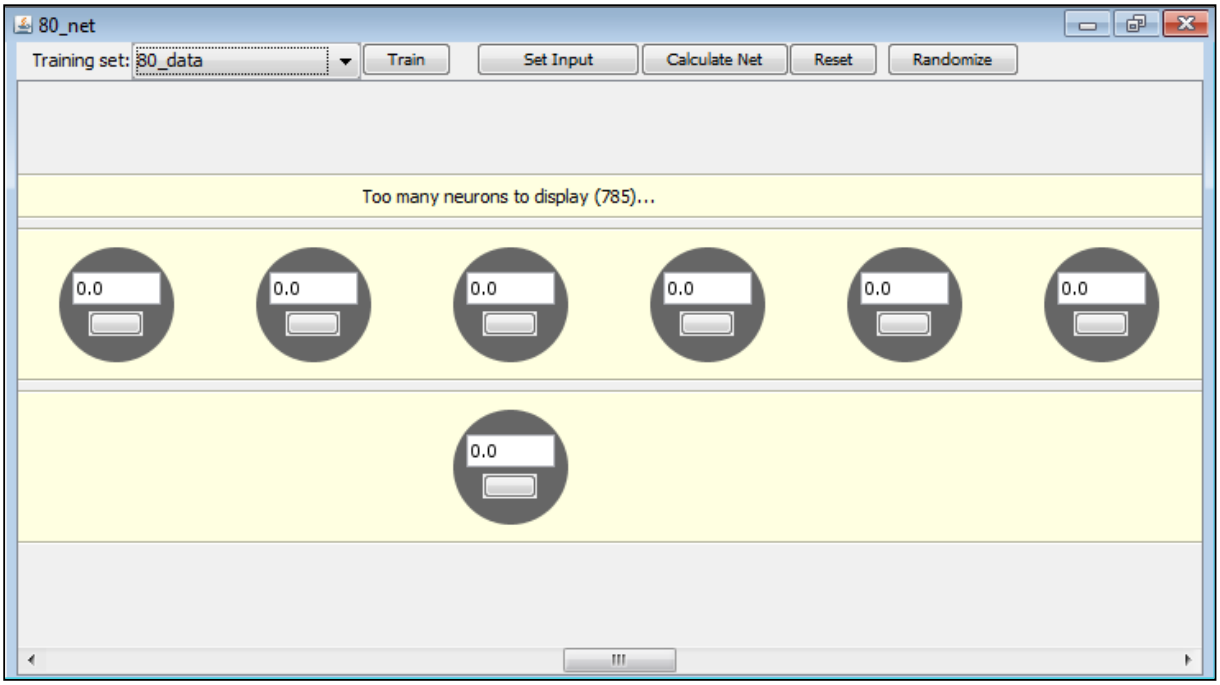


Figura 29 – Tela do diagrama e treinamento da rede neural

A Figura 29 exibe o diagrama da rede neural, com as camadas e quantidade de neurônios em cada camada. A camada de saída, neste caso a camada três, possui quantidade de neurônios conforme a quantidade de imagens existentes na pasta informada no campo `Image dir`, conforme a Figura 26. A camada de entrada, ou seja, a camada um possui a quantidade de neurônios de acordo com o tamanho das imagens redimensionadas, em *pixels*, calculado a partir das informações inseridas na tela de criação do conjunto de treinamento, conforme Figura 26, multiplicando a largura, altura e quantidade de canais de cor RGB, e

adicionando mais um neurônio.

Para iniciar o treinamento clica-se no botão `Train`, que abrirá uma nova tela, conforme a Figura 30, onde há campos para serem digitados os parâmetros de aprendizado.

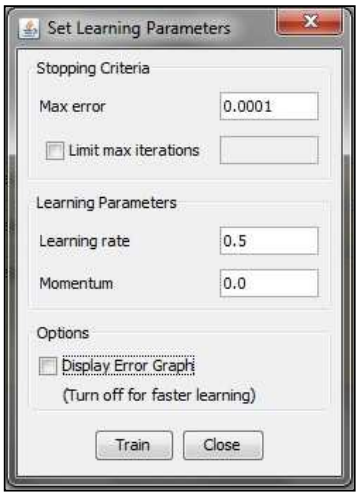


Figura 30 – Tela de parâmetros de aprendizado

Na tela da Figura 30 também há um botão `Train`, que ao ser clicado inicia a execução do treinamento e exibe a tela do treinamento em andamento, conforme a Figura 31(a). Quando o treinamento termina, pode-se fechar a tela, clicando no botão `close`, conforme a Figura 31(b).

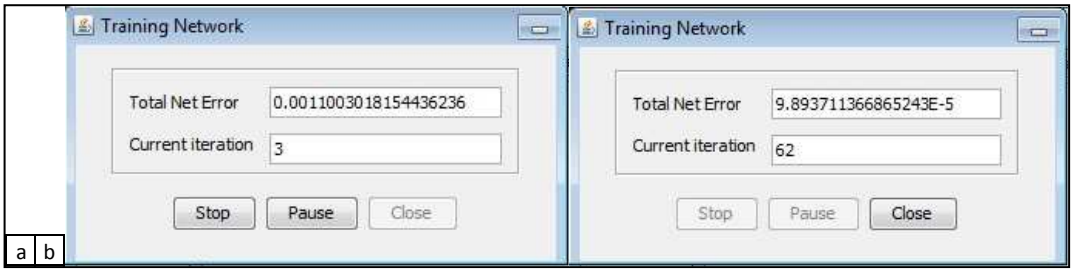


Figura 31 – Tela do andamento do treinamento da rede neural

Para efetuar o teste da rede neural treinada, minimiza-se a tela exibida na Figura 29 e clica-se na aba `Test` e seleciona-se uma imagem para o teste, conforme a Figura 32.

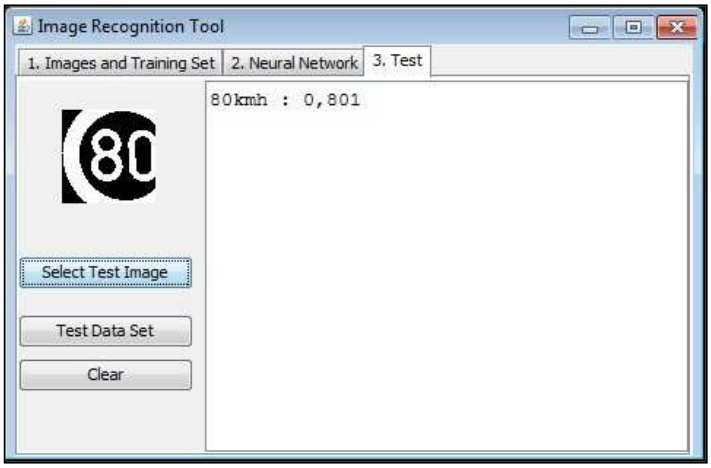


Figura 32 – Tela de teste da rede neural

Caso os testes sejam satisfatórios, a rede neural treinada é gravada através do menu `File`, submenu `Save`, conforme a Figura 25.

3.4 RESULTADOS E DISCUSSÃO

A tarefa de reconhecer uma placa de trânsito foi realizada utilizando apenas a linguagem Java no desenvolvimento do protótipo. O protótipo desenvolvido apresentou resultados satisfatórios, dentro de um ambiente controlado e apesar de algumas limitações, identifica em tempo real uma placa de trânsito.

A classe `VisualAutonomy` foi modificada a partir de exemplos disponíveis no *site* de soluções JMF da Sun Microsystems Inc., onde também há exemplos de efeitos e *codecs* customizados que foram modificados para a criação da classe `ColorEffect` e da classe `SignCodec`. A classe `SignCodec` incorpora os métodos de transformada circular de Hough, reutilizados do trabalho de Schulze (2003) e os métodos reutilizados do código fonte exemplo encontrado no *site* de documentação do *framework* Neuroph.

O protótipo foi configurado para obter as imagens através do *stream* de uma *webcam*, utilizando a configuração padrão da ferramenta JMStudio, incluída no *framework* JMF. A configuração utilizada é de 320x240 *pixels* para o tamanho da imagem e 15 *frames* por segundo para a taxa de transferência.

As imagens obtidas possuem muito ruído, caracterizado por pixels aleatórios com cor incorreta, exigindo a manipulação da imagem, de maneira a minimizar ou ignorar o ruído. A manipulação do ruído impacta no desempenho do protótipo, porém a ausência da manipulação

impacta no reconhecimento da placa pela rede neural.

A quantização das cores permite que o operador de Sobel de detecção de borda, incorporado aos métodos relacionados à transformada de Hough presentes na classe `SignCodec`, gere uma quantidade menor de bordas detectadas e conseqüentemente reduza a ocorrência de falso-positivos de detecção de placas. A quantização pode ser regulada, através de parâmetro, para compensar o brilho do ambiente.

Uma característica não encontrada ao explorar a API JMF, é a capacidade de obter e também definir o controle de auto-brilho de uma *webcam*, o que favoreceria a quantização das características de cor das imagens, já que esse processo pode ser prejudicado pelo brilho. As imagens escuras obtiveram melhores resultados no processo de quantização.

A rede neural foi treinada utilizando um conjunto de parâmetros de treinamento estimados. Então a rede neural foi testada, primeiramente, dentro da ferramenta easyNeurons, que acompanha o *framework* Neuroph, utilizando uma imagem de placa de trânsito. Esta ferramenta possui a limitação de não permitir, por exemplo, que múltiplas placas de proibição de ultrapassagem, com características ligeiramente diferentes, resultem em um único neurônio de saída para esse tipo de placa. Cada imagem, utilizada no conjunto de treinamento, irá gerar um neurônio correspondente na camada de saída.

O consumo de memória da ferramenta easyNeurons afeta o próprio funcionamento. Foi necessário aumentar a quantidade de memória alocada pelo Java para 1 gigabyte, a fim de gravar corretamente o arquivo da rede neural criada. Com a configuração padrão de memória a ferramenta também funciona, porém ocorre erro durante a gravação e o arquivo é gravado corrompido. O corrompimento do arquivo foi detectado pela ocorrência de erro ao tentar carregar o arquivo dentro do protótipo.

Referente aos trabalhos correlatos apresentados, o trabalho de Rodrigues (2002) demonstra uma abordagem possível para a detecção das placas pelo uso do mecanismo de atenção visual, através dos processos de filtragem linear, diferenças centro-vizinhança, soma dos mapas de características e dos mapas de conspicuidade e a seleção das regiões de interesse.

4 CONCLUSÕES

Hoje já são comumente encontradas soluções comerciais de sistemas de apoio ao motorista, algumas já integrando veículos acessíveis a população em geral, tais como sistemas que auxiliam o motorista a estacionar o veículo, deixando de ser um item exclusivo de modelos luxuosos.

A evolução das técnicas e algoritmos, além da evolução e redução de custo do *hardware* utilizado em sistemas de visão computacional, vem favorecendo o aumento do número de projetos desenvolvidos em instituições de ensino, mostrando o interesse que este assunto desperta.

Pode-se notar pelas soluções propostas por Piccioli et al. (1996) e Escalera, Armingol e Mata (2003) e pelo levantamento efetuado por Jung et al. (2005), que técnicas diferentes podem ser combinadas com o objetivo de detectar e reconhecer placas de trânsito. Cada uma das técnicas apresenta vantagens e desvantagens, o que motiva a busca por melhores soluções. Basicamente, a análise de cores e formas e redes neurais tem se mostrado como os métodos empregados com mais sucesso em soluções ao redor do mundo.

As tecnologias JAI, JMF e Neuroph vêm favorecer o requisito de desenvolver a ferramenta somente utilizando a linguagem Java, possibilitando traduzir algoritmos de visão computacional de outros sistemas escritos em linguagem diferente de Java.

O uso da ferramenta Enterprise Architect 7.1 para a elaboração das especificações do trabalho se mostrou ágil e fácil de usar, atendendo todas as necessidades do projeto, não apresentando problemas durante sua execução.

O presente trabalho possibilitou desenvolver uma ferramenta, capaz de detectar e reconhecer placas de trânsito em tempo real, de baixo custo e multiplataforma. Porém ainda é necessário demandar mais tempo em testes e tentativas com outros métodos de visão computacional, a fim de minimizar problemas com a qualidade das imagens processadas pelo protótipo, como também refinar o código fonte da implementação.

4.1 EXTENSÕES

Como sugestão para futuros trabalhos, sugere-se implementar um modo de controlar o brilho de uma imagem, ou utilizar um algoritmo que não sofra influência do brilho.

Sugere-se também a exploração de outras APIs de manipulação de imagens.

Outra melhoria possível é o desenvolvimento de uma ferramenta de treino de rede neural, utilizando a API Neuroph, especializada para as necessidades de reconhecimento de placas, onde as imagens seriam separadas em conjuntos e agrupadas por categorias e então utilizadas no treinamento da rede, sendo que a placa seria identificada pela categoria que pertence.

REFERÊNCIAS BIBLIOGRÁFICAS

ABELÉM, Antônio J. G.; PACHECO, Marco A. C.; VELLASCO, Marley M. B. R. Modelagem de redes neurais artificiais para previsão de séries temporais. In: SIMPÓSIO BRASILEIRO DE REDES NEURAIS, 2., 1995, São Carlos. **Anais...** São Carlos: Universidade Federal de São Carlos, 1995. p. 107-112.

AZEVEDO, Fernando M.; BRASIL, Lourdes M.; OLIVEIRA, Roberto C. L. **Redes neurais com aplicações em controle e em sistemas especialistas**. Florianópolis: Visualbooks, 2000.

CASTRO, Fernando C.; CASTRO, Maria. **Redes neurais artificiais**. Porto Alegre, 2003. Disponível em: <http://www.ee.pucrs.br/~decastro/RNA_hp/RNA.html>. Acesso em: 22 maio 2010.

CENTRO DE INFORMAÇÕES DO GALILEO. **Aplicações GALILEO rodoviário**. São José dos Campos, [2005?]. Disponível em: <http://www.galileoic.org/la/files/Aplicações%20GALILEO%20Rodoviário.pdf>. Acesso em: 11 mar. 2010.

CENTURION, Virgilio. **Minha vida**: a maioria dos acidentes ocorrem no lusco-fusco. [S.l.], 2009. Disponível em: <http://minhavida.uol.com.br/materias/bemestar/A+maioria+dos+acidentes+ocorrem+no+luscofusco.mv>. Acesso em: 30 mar. 2010.

CHANG, Yuchou et al. A robust color image quantization algorithm based on knowledge reuse of k-means clustering ensemble. **Journal of Multimedia**, [S.l.], v. 3, n. 2, p. 20-27, June 2008. Disponível em: <http://www.academypublisher.com/ojs/index.php/jmm/article/download/03022027/1229>. Acesso em: 29 maio 2010.

CUNHA, Kélita R. M. G.; JACQUES, Maria A. P.; CYBIS, Helena B. B. Efeito da percepção dos motoristas sobre as características viário-ambientais nas velocidades praticadas em vias urbanas. In: CONGRESO PANAMERICANO INGENIERIA TRÁNSITO Y TRANSPORTE, 14., 2006, Las Palmas de Gran Canaria. **Anais eletrônicos...** Las Palmas de Gran Canaria: [s.n.], 2006. Não paginado. Disponível em: <http://redpgv.coppe.ufrj.br/arquivos/cunha_jacques_cybis_PANAM2006.pdf>. Acesso em: 11 mar. 2010.

ESCALERA, Arturo de la; ARMINGOL, José M.; MATA, Mario. Traffic sign recognition and analysis for intelligent vehicles**. Image and Vision Computing**, Amsterdam, v. 21, n. 3, p. 247-258, Mar. 2003. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.7754&rep=rep1&type=pdf>. Acesso em: 31 maio 2010.

HAN, Jiawei; KAMBER, Micheline. **Data mining**. Concepts and techniques. São Francisco: Morgan Kaufmann Publishers, 2000.

HAYKIN, Simon. **Neural networks**. A comprehensive Foundation. New Jersey: Prentice Hall, 1994.

_____. **Redes neurais**: princípios e prática. Porto Alegre: Bookman, 2001.

JUNG, Cláudio R. et al. Computação embarcada: projeto e implementação de veículos autônomos inteligentes. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 25., 2005, São Leopoldo. **Anais eletrônicos...** São Leopoldo: UNISINOS, 2005. p. 1385-1395. Disponível em: <http://www.unisinos.br/_diversos/congresso/sbc2005/_dados/anais/pdf/arq0288.pdf>. Acesso em: 18 mar. 2010.

KEMPF, Jean-Philippe. **Opel cars can see**: opel eye camera reads signs, improves safety. Rüsselsheim, 2008. Disponível em: <http://media.gm.com/servlet/GatewayServlet?target=http://image.emerald.gm.com/gmnews/viewmonthlyreleasedetail.do?domain=82&docid=46499>. Acesso em: 01 abr. 2010.

KUBIÇA, Stefano. **Processamento de imagens de documentos - parte V**. Curitiba, [1999?]. Disponível em: <http://www.batebyte.pr.gov.br/modules/conteudo/conteudo.php?conteudo=1435>. Acesso em: 29 maio 2010.

MARTINELLI, Edmar. **Extração de conhecimento de redes neurais artificiais**. 1999. 113 f. Dissertação (Mestrado em Ciências da Computação) – Curso de Pós-graduação em Ciências da Computação, Universidade Federal de São Paulo, São Paulo.

MEDEIROS, Jonathan D. **Ferramenta de detecção da região da placa de veículos**. 2009. 66 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

MORAES, Marcelo B. C.; NAGANO, Marcelo S.; MERLO, Edgard M. Previsão de faturamento no varejo brasileiro utilizando-se de um modelo de redes neurais artificiais. In: ENCONTRO NACIONAL DE ENGENAHRIA DE PRODUÇÃO, 22., 2002, Curitiba. **Anais eletrônicos...** Rio de Janeiro: ABEPRO, 2002. Não paginado. Disponível em: <http://www.abepro.org.br/biblioteca/ENEGEP2002_TR71_1085.pdf>. Acesso em: 29 maio 2010.

PEDRINI, Hélio; SCHWARTZ, William R. **Análise de imagens digitais**: princípios, algoritmos e aplicações. São Paulo: Thomson Learning, 2008.

PICCIOLI, Giulia et al. Robust method for road sign detection and recognition, **Image and Vision Computing**, Amsterdam, v. 14, n. 3, p. 209-223, Apr. 1996. Disponível em: <http://www.inf.unideb.hu/~totht/modszerek/cikkek/robust.pdf>. Acesso em: 31 maio 2010.

PISTORI, Hemerson; PISTORI, Jeferson; COSTA, Eduardo R. Hough-circles: um módulo de detecção de circunferências para o ImageJ. In: WORKSHOP SOFTWARE LIVRE, 6., 2005, Porto Alegre. **Anais eletrônicos...** Porto Alegre: UCDB, 2002. Não paginado. Disponível em: <http://www.gpec.ucdb.br/pistori/publicacoes/pistori_wsl2005.pdf>. Acesso em: 29 maio 2010.

RODRIGUES, Fabrício A. **Localização e reconhecimento de placas de sinalização utilizando um mecanismo de atenção visual e redes neurais artificiais**. 2002. 108 f. Dissertação (Mestrado em Informática) – Curso de Pós-Graduação em Informática, Universidade Federal de Campina Grande, Campina Grande. Disponível em: <http://www.dsc.ufcg.edu.br/~copin/pesquisa/bancodissertacoes/2002/FabricioAugustoRodrigues.pdf>. Acesso em: 31 mar. 2010.

SANTOS, Daniel. **Sistema óptico para identificação de veículos em estradas**. 2008. 65 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SCHULZE, Mark A. **Circular Hough transform**. [S.l.], 2003. Disponível em: <http://www.markschulze.net/java/hough/>. Acesso em: 29 maio 2010.

SEVARAC, Zoran et al. **Java neural network framework Neuroph**. [S.l.], 2008. Disponível em: <http://neuroph.sourceforge.net/features.html>. Acesso em: 29 maio 2010.

SUN MICROSYSTEMS INC. **Programming in Java advanced imaging**. California, 1999a. Disponível em: <http://java.sun.com/products/java-media/jai/forDevelopers/jai1_0_1guide-unc/index.html>. Acesso em: 22 mar. 2010.

_____. **Java media framework API guide**. California, 1999b. Disponível em: <http://java.sun.com/javase/technologies/desktop/media/jmf/2.1.1/guide/JMFTOC.html>. Acesso em: 22 mar. 2010.

ULF, Jon. **Future car tech**. Utah, 2010. Disponível em: <http://myscifi.org/future-car-tech/>. Acesso em: 07 jul. 2010.

WANGENHEIM, Aldo von. **Visão computacional**: seminário introdução à visão computacional. Florianópolis, [1998?]. Disponível em: <http://www.inf.ufsc.br/~visao/>. Acesso em: 11 mar. 2010.