UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

FERRAMENTA DE DETECÇÃO DA REGIÃO DA PLACA DE VEÍCULOS

JONATHAN DAMÁSIO MEDEIROS

BLUMENAU
2009

2009/1-11

JONATHAN DAMÁSIO MEDEIROS

FERRAMENTA DE DETECÇÃO DA REGIÃO DA PLACA DE VEÍCULOS

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 Ciências da Computação — Bacharelado.

Prof. Dalton Solano dos Reis, M.Sc. - Orientador

BLUMENAU
2009

2009/1-11

FERRAMENTA DE DETECÇÃO DA REGIÃO DA PLACA DE VEÍCULOS

Por

JONATHAN DAMÁSIO MEDEIROS

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

Presidente:	Prof. Dalton Solano dos Reis, M.Sc. – Orientador, FURB
Membro:	Prof. Paulo César Rodacki Gomes, Dr. – FURB
Membro:	Prof. Mauro Marcelo Mattos, Dr. – FURB

Blumenau, 07 de julho de 2009

Dedico este trabalho a minha família, minha noiva e a todos os amigos que me apoiaram para a realização deste.

AGRADECIMENTOS

A Deus, pela saúde e possibilidade de sempre concluir os meus objetivos.

À minha família e a minha noiva, que sempre me apoiaram e contribuíram para a conclusão da minha graduação.

Ao meu orientador, Dalton Solano dos Reis, pelo comprometimento, atenção e conhecimento fornecido, que foram fundamentais para a conclusão deste trabalho.

O homem é aquilo que sabe.

Francis Bacon

RESUMO

No presente trabalho são descritas técnicas, bem como a especificação e implementação de uma ferramenta para extrair os caracteres da placa de um veículo, através de uma imagem digital. São utilizadas técnicas de processamento de imagens para segmentar e preparar a imagem para ser analisada, enfatizando os algoritmos de gradiente utilizando o filtro de Sobel para detecção de bordas, e a morfologia matemática para obter a localização exata da placa do veículo na imagem. A ferramenta através de uma biblioteca OCR é capaz de reconhecer os caracteres da placa do veículo. O método utilizado para a localização da placa do veículo na imagem apresentou um comportamento dentro do esperado. Já a biblioteca MODI utilizada para o reconhecimento dos caracteres da placa do veículo, apresentou comportamento não adequado no reconhecimento dos caracteres da placa do veículo.

Palavras-chave: Placa veículo. Processamento de imagens. Morfologia matemática. Gradiente. Sobel.

ABSTRACT

This work describes techniques, as well as the specification and implementation of the tool to extract characters of vehicle license plate, by a digital image. It uses techniques of image processing to segment and prepare the image to be analyzed, emphasizing the gradient algorithm using Sobel filter to the border detection, and the mathematical morphology to get the exactly location of license plate. The tool by an OCR library is able to recognize the characters of vehicle license plate. The way used to find the vehicle license plate in the image show a behavior within the expected. Already the library MODI used to recognize the characters of vehicle license plate, show a not appropriate behavior in the characters recognize of vehicle license plate.

Key-words: Vehicle plate. Images processing. Mathematical morphology. Gradient. Sobel.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

ANPR - *Automatic Number Plate Recognition*

API – *Application Programming Interface*

CPU - *Central Processing Unit*

DLL – *Dynamically Linkable Library*

EE – Elemento Estruturante

JAI – *Java Advanced Imaging*

JUDE – *Java UML Modeling Tool*

MODI – *Microsoft Office Document Imaging*

OCR - *Optical Character Recognition*

RGB – *Red, Green and Blue*

SDK - *Software Development Kit*

UML – *Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

Com a necessidade de deslocamento, cada dia mais pessoas utilizam veículos automotores como meio de transporte. De acordo com Pera (2007), estima-se que em 2008 sejam vendidos mais de dois milhões de veículos no mercado interno. Diante do aumento gradativo da quantidade de veículos nas vias e estacionamentos, surge a necessidade de controles de acesso para veículos.

O controle de acesso para veículos pode ser adotado em estacionamentos, condomínios, pedágios, entre outros. Geralmente é realizado pela placa do veículo que é um identificador único. Os caracteres da placa podem ser obtidos de forma manual, quando uma pessoa realiza a extração visual dos caracteres, ou automática quando a extração é realizada por um equipamento e/ou software.

No controle de acesso manual a garantia da eficiência fica dependente das pessoas que realizam a extração dos caracteres das placas e para uma boa eficiência no controle de acesso, é necessária uma ferramenta que consiga obter os caracteres do veículo de forma automatizada. Uma possível solução é a utilização de um software *Optical Character Recognition* (OCR) para que através de uma imagem, consiga extrair os caracteres da placa do veículo. Entretanto, nos testes realizados com a biblioteca JavaOCR (JAVAOCR TEAM, 2008) e com o software TopOCR (TOPSOFT, 2008), constatou-se que o reconhecimento não é realizado com sucesso, mesmo que sejam recortados os caracteres da placa do veículo.

Diante do exposto, propõe-se a construção de uma ferramenta capaz de localizar, extrair e reconhecer os caracteres da placa do veículo de forma confiável. Para o reconhecimento serão realizados os tratamentos necessários para que uma biblioteca OCR consiga interpretar os caracteres.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver uma ferramenta que seja capaz de reconhecer em uma imagem os caracteres da placa de um veículo.

Os objetivos específicos são:

a) analisar uma imagem contendo um veículo e identificar a região da placa;

b) realizar os tratamentos necessários na imagem da região da placa para que uma biblioteca OCR realize o reconhecimento;
c) extrair e apresentar os caracteres da imagem identificada como placa.

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 controle de acesso para veículos, processamento de imagens, aquisição de imagens, segmentação de imagem, detecção de bordas, morfologia matemática, reconhecimento e uma técnica para obtenção da placa do veículo. Por fim, apresentam-se dois trabalhos correlatos.

O terceiro capítulo trata sobre o 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 abordados os assuntos e técnicas utilizadas para o desenvolvimento da ferramenta de reconhecimento de placas de veículos. Na seção 2.1 são apresentadas informações sobre sistemas de controle de acesso para veículos. Na seção 2.2 são apresentados conceitos e técnicas de processamento de imagens. Na seção 2.3 são apresentados os conceitos de morfologia matemática. Na seção 2.4 são apresentadas informações sobre reconhecimento de imagem. Na seção 2.5 é apresentado um artigo sobre reconhecimento de placas de veículos. Por fim, na seção 2.6 são apresentados dois trabalhos correlatos.

2.1 SISTEMAS DE CONTROLE DE ACESSO PARA VEÍCULOS

Conforme Conselho Nacional de Trânsito (2007), “cada veículo será identificado por placas dianteira e traseira, afixadas em primeiro plano e integrante do mesmo. A placa contém sete (7) caracteres alfanuméricos individualizados, sendo o primeiro grupo composto por três (3), resultante do arranjo, com repetição de vinte e seis (26) letras, tomadas três a três. O segundo grupo é composto por quatro (4), resultante do arranjo, com repetição, de dez (10) algarismos, tomados quatro a quatro”.

Seja qual for o tipo de controle utilizado para veículos, em especial controle de acesso, geralmente é utilizada a placa como o identificador único do veículo. Diante disso, surge a necessidade de sistemas de reconhecimento de placas de veículos para automatizar o processo de identificação do veículo.

De acordo com Xu e Zhu (2007, p. 180), nos últimos anos vem aumentando a necessidade de sistemas de reconhecimento de placa de veículos. A localização e segmentação[1] são processos fundamentais no reconhecimento da placa devido às diferentes condições de iluminação e complexos *backgrounds* que uma imagem pode conter. A qualidade do processo de localização da imagem influencia diretamente na velocidade e precisão dos sistemas de reconhecimento.

[1] O processo de segmentação consiste em localizar sub-imagens a partir das bordas dos objetos.

Xu e Zhu (2007, p. 180) afirmam que pesquisadores encontraram muitos métodos de localização de placas. Porém, cada algoritmo possui suas próprias características, como por exemplo, alguns são sensíveis ao brilho e necessitam de muito tempo de processamento, outros não são suficientemente robustos para os diferentes ambientes. Desta forma, muitas pesquisas são realizadas com o intuito de desenvolver um método que seja eficiente em qualquer situação.

2.2 PROCESSAMENTO DE IMAGENS DIGITAIS

De acordo com Camapum (2005), o conhecimento de como reage o sistema visual humano e de algumas técnicas disponíveis para melhor adequar a imagem à aplicação são fatores importantes para desenvolver soluções computacionais automatizáveis. Nesta ótica, o processamento de imagens digitais permite viabilizar um grande número de aplicações, tanto no domínio do aprimoramento de informações para interpretação humana quanto no domínio da análise automática por computador de informações extraídas de uma imagem ou cena.

Segundo Gomes e Velho (2003, p. 3), no processamento de imagens o sistema admite como entrada uma imagem que, após processada, produz outra imagem na saída. Um exemplo clássico desta área é o processamento de imagens enviadas por um satélite com o objetivo de colorizar ou de realçar detalhes.

2.2.1 Aquisição de imagens

O processo de aquisição da imagem consiste em obter uma imagem digital representada por pontos chamados de *pixels*. A aquisição da imagem pode ser adquirida através de uma câmera digital.

Segundo Kubiça, Facon e Letheiler (2008), a estação de captura da imagem deve estar preparada para gerar imagens com a melhor qualidade possível, evitando que a imagem gerada apresente informações indesejáveis, também conhecidas como ruído. Isso pode fazer com que informações importantes sejam distorcidas. Quanto melhor a qualidade da imagem[2],

[2] A qualidade da imagem pode ser atribuída à quantidade de *pixels* e quantidade de cores.

mais eficientes serão os processamentos das fases posteriores.

2.2.2 Segmentação de imagens

Segundo Gonzalez e Woods (2000, p. 295), o primeiro passo em análise de imagens é a segmentação de imagem. A segmentação subdivide uma imagem em suas partes ou objetos constituintes. O nível até o qual essa subdivisão deve ser realizada depende do problema a ser resolvido. Ou seja, a segmentação deve parar quando os objetos de interesse na aplicação tiverem sido isolados. Por exemplo, em aplicações de aquisição autônoma aérea de alvos terrestres, o interesse reside, entre outras coisas, na identificação de veículos em uma estrada. O primeiro passo é a segmentação da estrada na imagem, seguida da segmentação dos elementos constituintes da estrada em objetos que possuam tamanho pertencente a uma faixa de tamanhos correspondentes a veículos em potencial. Não se deve realizar a segmentação abaixo dessa escala, bem como não existe a necessidade de realizar uma segmentação de componentes da imagem que estejam fora dos limites da estrada.

Em geral a segmentação autônoma é uma das tarefas mais difíceis em processamento de imagens. Por essa razão, um cuidado considerável deve ser tomado para melhorar as chances de uma segmentação robusta.

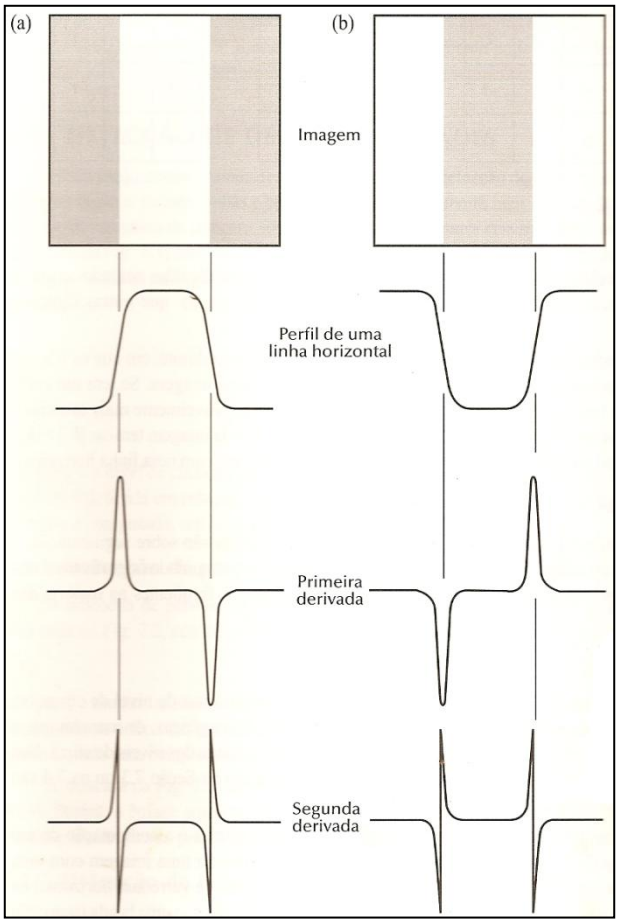
2.2.3 Detecção de bordas

De acordo com Gonzalez e Woods (2000, p. 297), a detecção de pontos e de linhas são elementos de qualquer discussão sobre segmentação de imagens, detecção de bordas é, de longe, a abordagem mais comum para a detecção de descontinuidades significantes nos níveis de cinza. A razão é que pontos e linhas finas isoladas não são ocorrências freqüentes na maioria das aplicações práticas.

Uma borda é o limite entre duas regiões com propriedades relativamente distintas de nível de cinza. Assume-se que as regiões em questão são suficientemente homogêneas, de maneira que a transição entre duas regiões pode ser determinada com base apenas na descontinuidade dos níveis de cinza. Basicamente, a idéia por trás da maioria das técnicas para a detecção de bordas é a computação de um operador local diferencial. A Figura 1(a) mostra uma imagem com uma faixa clara sobre o fundo escuro, o perfil de nível de cinza ao

longo de uma linha de varredura horizontal da imagem e a primeira e a segunda derivadas dessa linha de perfil. A partir do perfil que uma borda (transição do lado escuro para o claro) é modelada como uma mudança suave dos níveis de cinza, em vez de uma mudança abrupta. Esse modelo reflete o fato que bordas em imagens digitais são, geralmente, levemente borradas devido à amostragem.

A Figura 1(a) mostra que a primeira derivada do perfil de níveis de cinza é positiva na primeira borda (à esquerda), negativa na segunda (à direita) e nula nas áreas de nível de cinza constante. A segunda derivada é positiva na parte da transição associada ao lado escuro da borda, negativa na parte da transição associada ao lado claro da borda e nula nas áreas de nível de cinza constante. Dessa forma, a magnitude da primeira derivada pode ser usada na detecção da presença de uma borda em uma imagem, enquanto que o sinal da segunda derivada pode ser usado para determinar se um pixel da borda localiza-se no lado escuro ou no claro da mesma. A segunda derivada possui um cruzamento por zero na posição de cada borda.



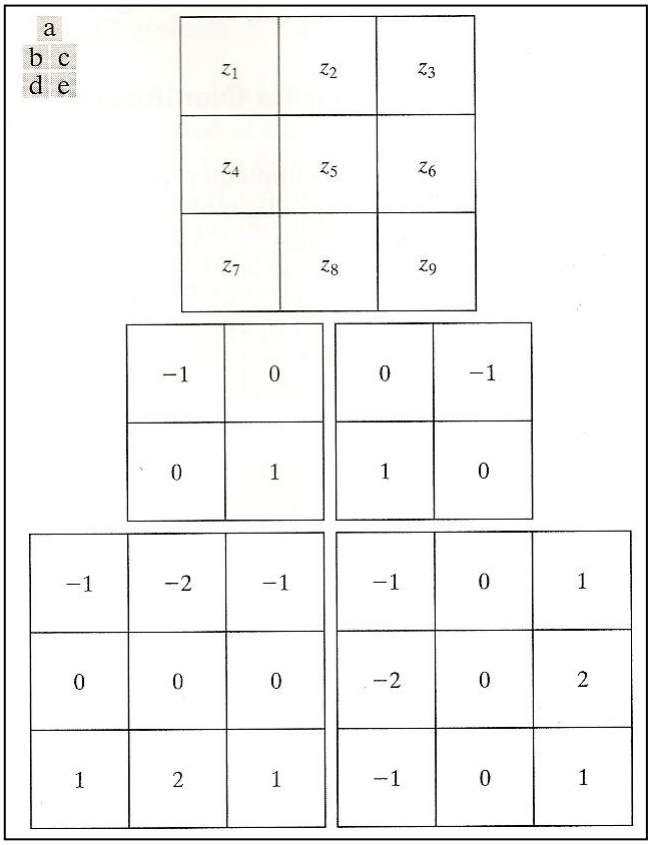
Fonte: adaptado Gonzalez e Woods (2000, p.298).

Figura 1 – Detecção de bordas por operadores de derivação

2.2.4 Gradiente

De acordo com Gonzalez e Woods (2008, p. 168), o gradiente é utilizado freqüentemente nos processos industriais para detecção de defeitos ou o que é mais comum, como uma etapa de pré-processamento em uma inspeção automatizada.

Para realizar a operação de gradiente, é necessário utilizar uma máscara para a aproximação da derivada, as máscaras mais utilizadas na operação do gradiente são isotrópicas[3] em múltiplos de 90º. Utilizando a Figura 2(a) para representar a intensidade dos pontos da imagem em uma região de tamanho 3x3. O centro do ponto, $z5$, representa $f(x,y)$ em uma localização qualquer de (x,y); $z1$ denota $f(x-1,y-1)$; e assim sucessivamente conforme apresentado na Figura 3.

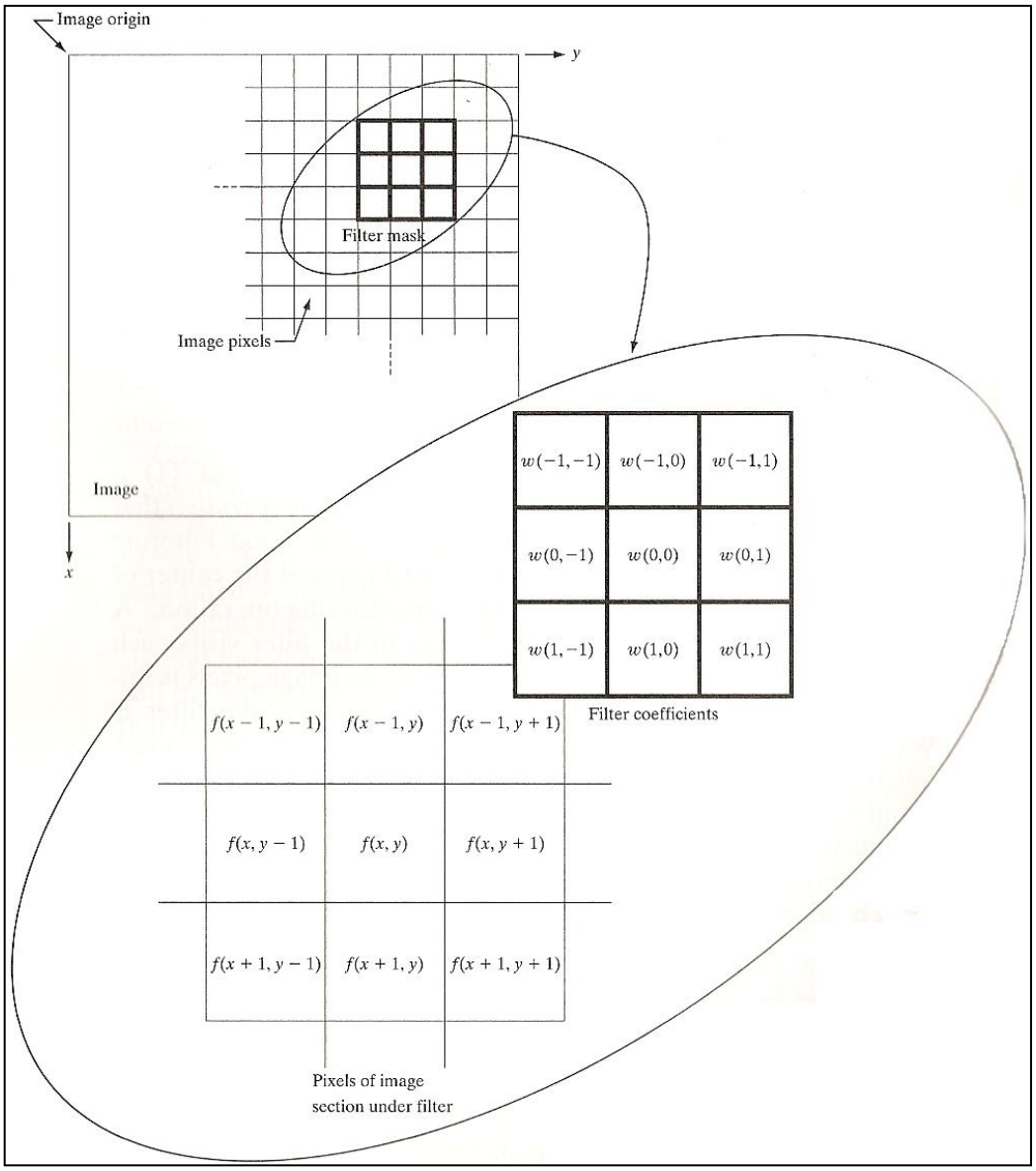


Fonte: Gonzalez e Woods (2008, p.166).

Figura 2 – Exemplo de máscaras de gradiente

As máscaras de tamanho par Figura 2(b) e (c), são inadequadas para a implementação computacional pelo motivo de que não possuem um centro de simetria.

[3] A soma de todos os coeficientes é igual a zero.



Fonte: Gonzalez e Woods (2008, p.146).

Figura 3 – Coordenadas dos coeficientes da máscara

Segundo Gonzalez e Woods (2008, p. 167), as menores máscaras são as de tamanho 3x3. A diferença entre a terceira e a primeira linha da Figura 2, é que a máscara da Figura 2(d) aproxima a derivada parcial na direção x, enquanto que a Figura 2(e) aproxima a derivada na direção y. Calculando as derivadas parciais com estas máscaras, obtém-se a magnitude do gradiente através da seguinte equação: $M(x,y) \approx |(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)| + |(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)|$. As máscaras das Figura 2(d) e (e) são chamadas de operadores de Sobel. O objetivo do valor dois no centro do coeficiente é realizar uma suavização, dando mais importância ao ponto do centro. Nos operadores de Sobel, a soma de todos os coeficientes é igual a zero. Dessa forma, é obtido resposta zero em áreas de constante intensidade, como é esperado de um operador de derivação, conforme mencionado na seção 2.2.3. A Figura 4 possui uma imagem a ser aplicado o gradiente.

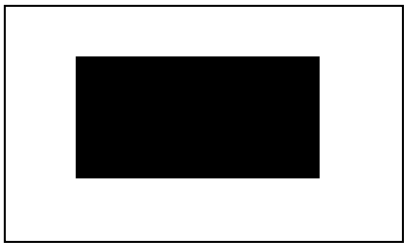


Figura 4 – Imagem original

Após a aplicação do gradiente através da máscara de Sobel, pode-se observar na Figura 5 que as regiões de constante intensidade da Figura 4 não foram destacadas, permanecendo em preto.

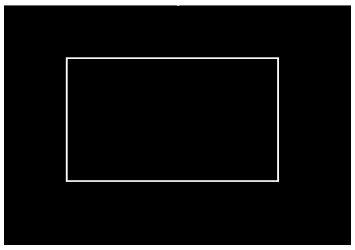
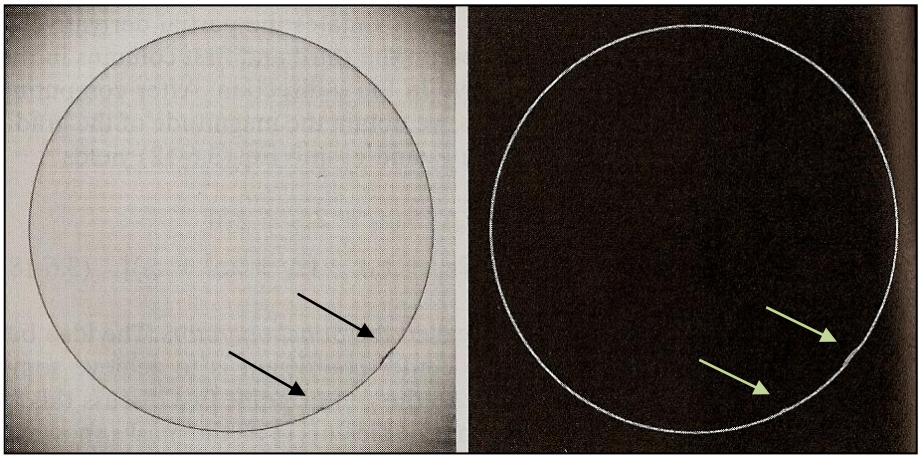


Figura 5 – Bordas destacadas

Uma aplicação prática é apresentada na Figura 6(a) que mostra a imagem óptica de uma lente de contato. A lente possui uma iluminação capaz de destacar imperfeições, como os dois defeitos na borda da lente vistos as quatro e cinco horas (conforme em destaque). A Figura 6(b) mostra o gradiente obtido com as duas máscaras de Sobel (Figura 2(d) e (e)). Os defeitos na borda são facilmente visíveis na imagem, com a vantagem de que o contraste ou pequenas variâncias de tonalidade foram eliminados, simplificando dessa forma consideravelmente a tarefa computacional requerida para automatizar a inspeção.



Fonte: adaptado Gonzalez e Woods (2008, p.168).

Figura 6 – Exemplo de gradiente utilizando máscara de Sobel

2.3 MORFOLOGIA MATEMÁTICA

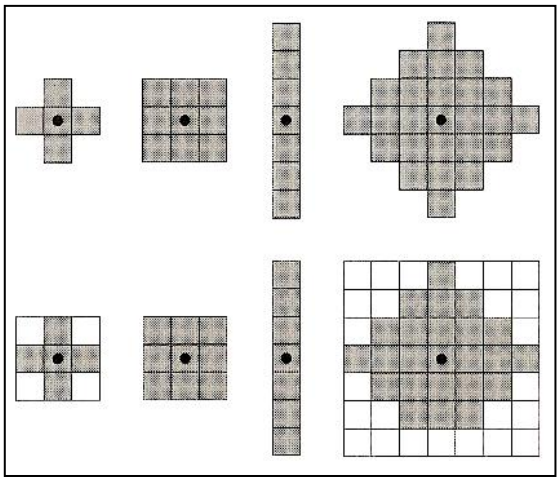
Segundo Gonzalez e Woods (2000, p. 369), a palavra morfologia normalmente denota uma área da biologia que trata a forma e a estrutura de animais e plantas. Utiliza-se a mesma palavra no contexto de morfologia matemática, como sendo uma ferramenta utilizada para a extração de componentes de imagens que sejam úteis na representação e descrição da forma de uma região.

Costa e César Jr. (2001, p. 255) afirmam que a morfologia matemática é muito aplicada como uma ferramenta de pré e pós-processamento de imagem para solução de diversos problemas. A morfologia matemática busca extrair de uma imagem desconhecida a sua geometria e topologia, sendo amplamente utilizada em: decomposição de imagens, reconhecimento de imagens, entre outros.

O principio básico da morfologia matemática é a teoria dos conjuntos, tornando-a uma abordagem unificada e poderosa para numerosos problemas de processamento de imagens. De acordo com Gonzalez e Woods (2000, p. 370), os conjuntos em morfologia matemática representam as formas dos objetos em uma imagem. Por exemplo, o conjunto de todos os *pixels* pretos em uma imagem binária é uma descrição completa dessa imagem. Em imagens binárias, os conjuntos em questão são membros do espaço bidimensional de números inteiros z^2, em que cada elemento do conjunto é um vetor bidimensional cujas coordenadas são as coordenadas (x, y) dos pixels pretos da imagem. Imagens digitais em níveis de cinza podem ser representadas por conjuntos cujos componentes estejam em z^3. Neste caso, dois componentes de cada elemento do conjunto se referem às coordenadas do *pixel*, enquanto o terceiro corresponde ao valor discreto de intensidade. Conjuntos em espaços de maiores dimensões podem conter outros atributos de imagens, como cor e componentes que variem com o tempo.

2.3.1 Erosão e dilatação

Segundo Gonzalez e Woods (2008, p. 630), a erosão e a dilatação são operadores primitivos do processamento morfológico. Para a realização da erosão ou dilatação em uma imagem, é necessário utilizar um Elemento Estruturante (EE), que é um pequeno conjunto ou uma sub-imagem Figura 7.



Fonte: Gonzalez e Woods (2008, p.629).
Figura 7 - Exemplo de elemento estruturante

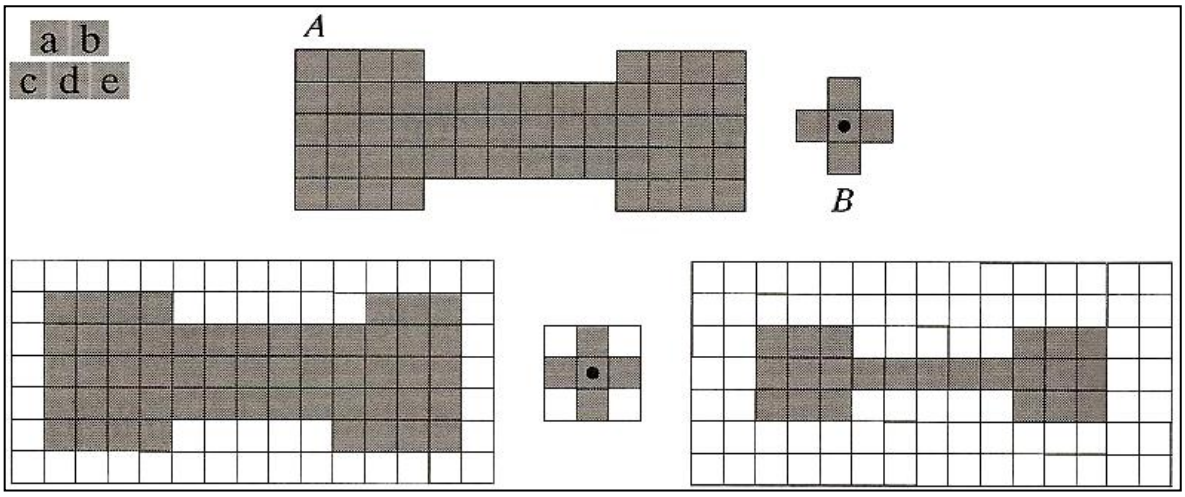
A primeira linha da Figura 7 mostra os variados tipos de EE, onde cada quadrado é parte do elemento estruturante. A origem do EE está indicada com um ponto preto (apesar da origem aparecer no centro do EE, a escolha da origem depende do problema em questão). Quando trabalha-se com imagens, é necessário que o EE seja um *array* retangular. Dessa forma, a segunda linha da Figura 7 apresenta a conversão para um *array* retangular dos EE que não possuem a forma original de um retângulo.

2.3.1.1 Erosão

Considerando A e B como elementos de z^2, a erosão de A por B denotado $A \ominus B$ é definida como: $A \ominus B = \{z|(B)_z \subseteq A\}$. Em outras palavras, a equação indica que a erosão de A por B é o conjunto de todos os pontos z tais que B, traduzido por z, está contido em A.

A Figura 8(a) e (b) apresenta um conjunto simples e um EE de tamanho 3x3 com origem no centro. Conforme mencionado anteriormente, para uma implementação computacional é necessário que os conjuntos A e B (elemento estruturante) sejam convertidos para um *array* retangular através da inclusão de elementos de fundo (*pixels* brancos para formar um retângulo), originando as Figura 8(c) e (d). Um novo conjunto é criado executando o EE (B) sobre o conjunto A. Para isso, é necessário que B visite todos os elementos de A. A cada posição da origem de B, se B está contido em A, marca-se esta posição como um membro do novo conjunto (sombreado), senão marque a posição como um não membro do novo conjunto (em branco). A Figura 8(e) apresenta o resultado desta operação. Observa-se que quando a origem de B está na borda do elemento A, parte do B não está contida em A,

eliminando assim a localização em que B está centrada como um possível membro para o novo conjunto. O resultado final da operação é que a borda do conjunto é corroída (erosão).



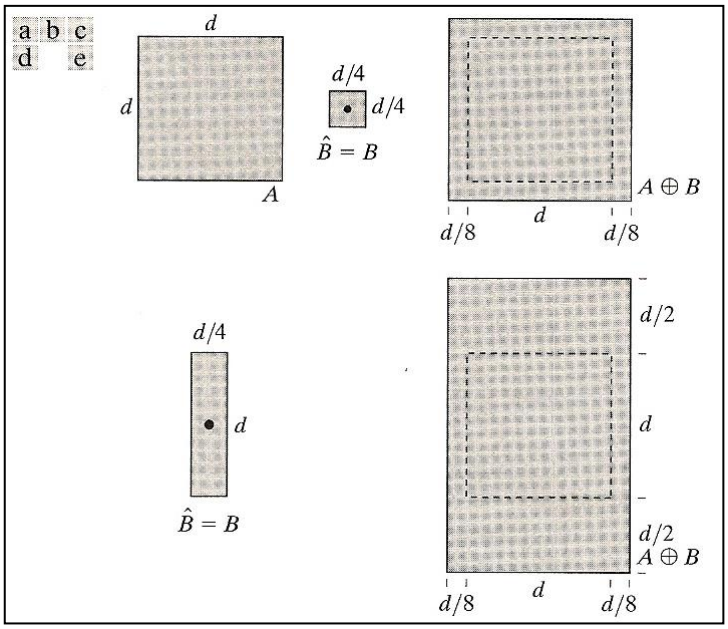
Fonte: Gonzalez e Woods (2008, p.630).

Figura 8 - Operação de erosão

2.3.1.2 Dilatação

Considerando A e B como elementos de z^2, a dilatação de A por B, denotada $A \oplus B$ é definido como: $A \oplus B = \left\{z \middle| \left[(\hat{B})_z \cap A \subseteq A\right]\right\}$, tal que B é o EE, e A é o conjunto (imagem) a ser dilatada.

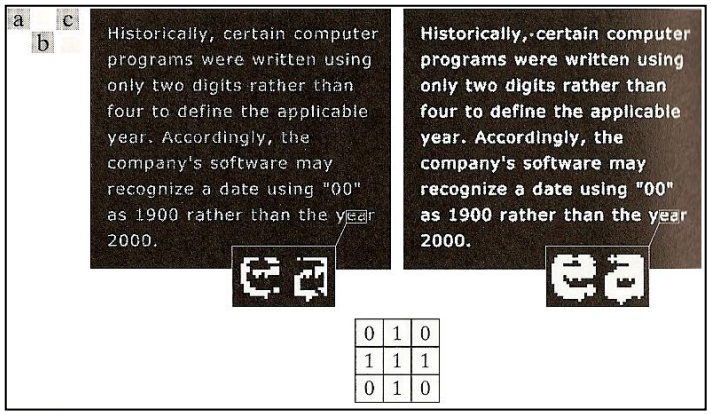
Diferente da erosão, que é o encolhimento ou operação de desbaste, a dilatação aumenta ou engorda os objetos de uma imagem binária. A forma de controle do crescimento ou engorda da imagem é realizado através do elemento estruturante. A Figura 9(a) e (b) apresentam um conjunto e um elemento estruturante. A Figura 9(c) apresenta o conjunto dilatado, o pontilhado delimita o conjunto original. A Figura 9(d) apresenta um elemento estruturante designado para alcançar uma maior dilatação vertical do que horizontal. A Figura 9(e) apresenta a dilatação alcançada com este elemento estruturante.



Fonte: adaptado Gonzalez e Woods (2008, p.634).

Figura 9 - Operação de dilatação

Uma aplicação simples da dilatação é o preenchimento de espaços. A Figura 10(a) apresenta uma imagem com caracteres "quebrados". A Figura 10(b) apresenta o elemento estruturante a ser utilizado para a dilatação, note que os 1s são utilizados para representa um elemento do EE, enquanto os 0s são utilizados para representar o fundo da imagem. A Figura 10(c) apresenta o resultado da dilatação da imagem original com o EE. Observa-se que as lacunas faltantes entre as letras foram preenchidas através do operador morfológico de dilatação.



Fonte: Gonzalez e Woods (2008, p.634).

Figura 10 - Exemplo de dilatação

2.3.2 Abertura e fechamento

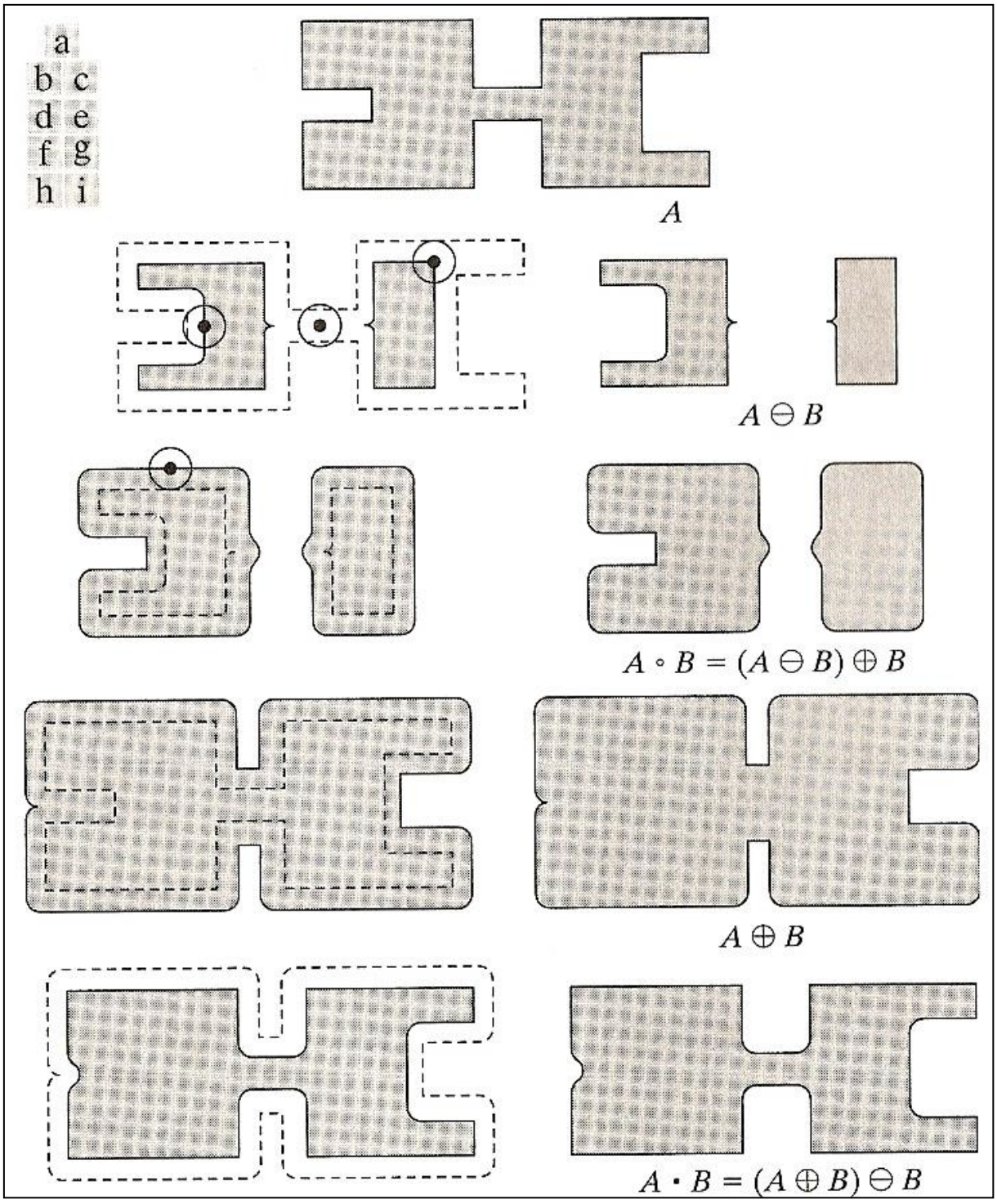
Conforme visto anteriormente, a dilatação expande os componentes da imagem,

enquanto que a erosão os encolhe. Através destes operadores primitivos, é possível realizar as operações de abertura e fechamento. A abertura geralmente suaviza o contorno de um objeto e elimina pequenas ligações. O fechamento também tende a suavizar os contornos da imagem, mas como o oposto da abertura, ele gera fusão em ligações quebradas, elimina pequenos buracos e preenche lacunas faltantes.

A abertura do conjunto A pelo elemento estruturante B, denotado $A \circ B = (A \ominus B) \oplus B$. Deste modo, a abertura de A por B é a erosão de A por B, seguida pela dilatação do resultado por B. Similarmente, o fechamento do conjunto A pelo elemento estruturante B, denotado $A \bullet B = (A \oplus B) \ominus B$, que diz que o fechamento de A por B é simplesmente a dilatação de A por B, seguida pela erosão do resultado de B.

A Figura 11 ilustra a operação de abertura e fechamento. A Figura 11(a) mostra o conjunto A, e a Figura 11(b) mostra as varias posições do elemento estruturante durante a operação de erosão. Quando concluído, esse processo resultou na separação da figura conforme Figura 11(c). A separação ocorreu na ligação "ponte" entre as duas principais seções. A largura da ponte era menor em relação ao diâmetro do elemento estruturante; por causa disso, o elemento estruturante não pode ser completamente contido nesta parte do conjunto, ocasionado a eliminação da ponte. A Figura 11(d) mostra o processo de dilatação do conjunto erodido, e a Figura 11(e) mostra o resultado final da operação de abertura, onde as bordas externas do conjunto foram arredondadas, enquanto que as bordas internas foram mantidas.

Similarmente, da Figura 11(f) até a Figura 11(i), apresenta o resultado de fechamento do conjunto A com o mesmo elemento estruturante. No resultado final, as bordas internas foram arredondadas, enquanto que as bordas externas foram mantidas. Além de que o espaço em branco no lado esquerdo da imagem original foi quase que totalmente preenchido.



Fonte: adaptado Gonzalez e Woods (2008, p.637).

Figura 11 - Exemplo de abertura e fechamento

2.4 RECONHECIMENTO

Loesch e Sari (1996, p. 5) afirmam que o reconhecimento computacional estuda e

descreve sistemas de visão artificial implementados por hardware ou software, geralmente fazendo uso de redes neurais artificiais.

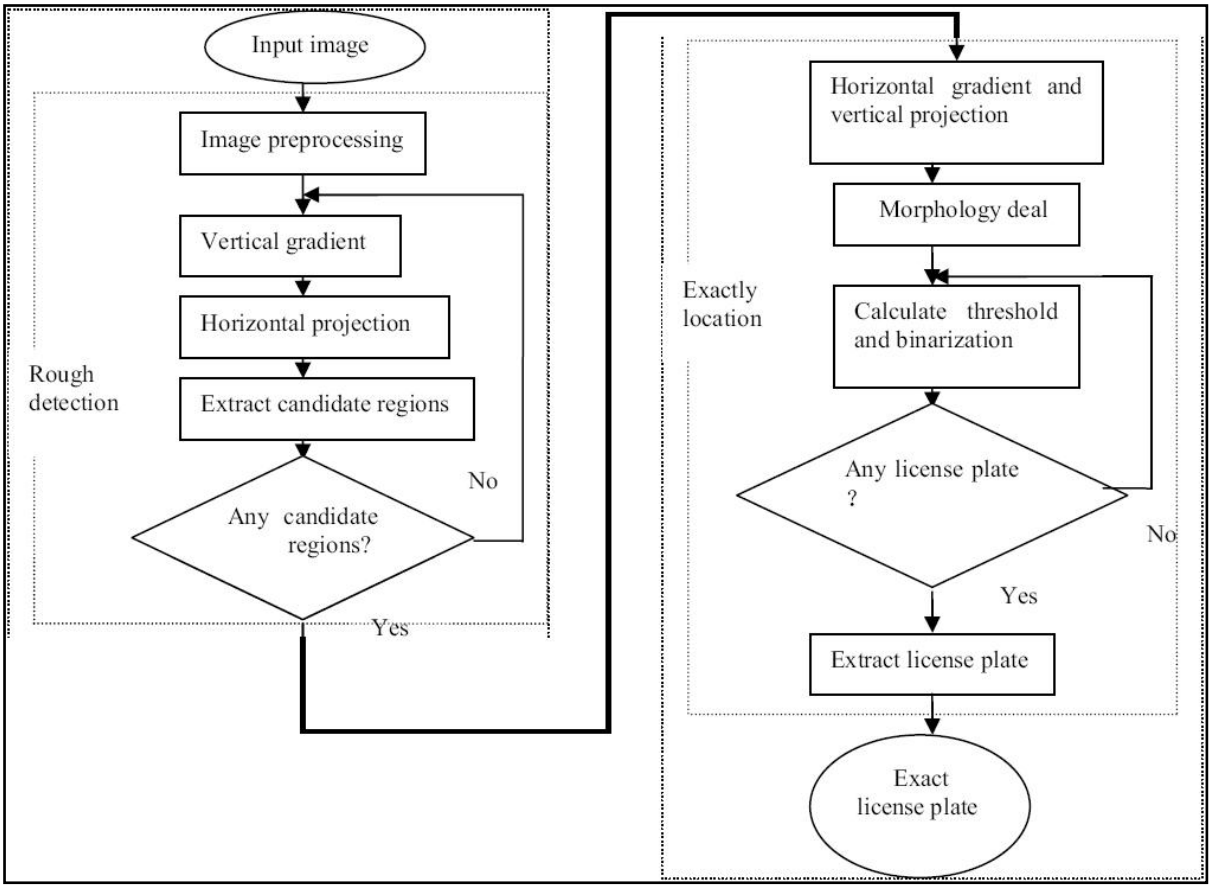
Segundo Alecrim (2004), redes neurais artificiais é um conceito da computação que visa trabalhar no processamento de dados de maneira semelhante ao cérebro humano. O cérebro é tido como um processador altamente complexo que realiza processamentos de maneira paralela, utilizando para isso os neurônios.

Segundo Loesch e Sari (1996, p. 5), o tempo de desenvolvimento de uma rede neural é razoável, e freqüentemente realiza tarefas bem melhor do que outras tecnologias mais convencionais, por exemplo sistemas especialistas. Desta forma, as redes neurais podem ser utilizadas para diversos tipos de reconhecimentos, tais como: caracteres, face, voz, entre outros.

2.5 LOCALIZAÇÃO DA PLACA DO VEÍCULO BASEADO EM HISTOGRAMA E MORFOLOGIA MATEMÁTICA

De acordo com Yang e Ma (2005, p. 89), a localização da placa do veículo pode ser baseada em histograma[4] e morfologia matemática. A maior variação de gradiente na imagem do veículo representa a região da placa e a morfologia matemática é muito eficiente na localização de *pixels* pretos em fundo branco ou *pixels* brancos em fundo preto. Desta forma, a região da placa pode ser obtida analisando o histograma da variação de gradiente e delimitada utilizando a morfologia matemática. O algoritmo proposto por Yang e Ma (2005) é apresentado no fluxograma da Figura 12.

[4] Histograma é utilizado para indicar a distribuição de dados.



Fonte: adaptado Yang e Ma (2005, p. 90).

Figura 12 - Fluxograma do algoritmo

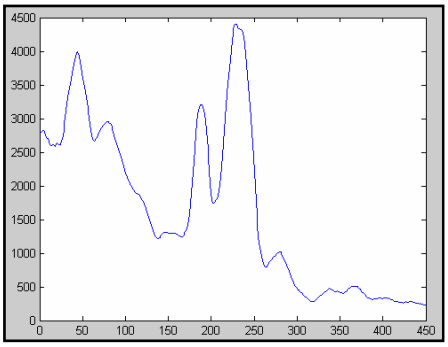
O processo é dividido em dois estágios. O primeiro, denominado detecção aproximada, obtém as regiões candidatas à placa usando detecção do gradiente vertical. No segundo estágio, localização exata, a posição vertical da placa é localizada através da morfologia matemática e a exata região é extraída das regiões candidatas.

2.5.1 Detecção aproximada

Como as imagens são adquiridas de um ambiente real, podendo a placa estar manchada, caracteres apagados, sujos, distâncias diferentes, iluminação, entre outros, é necessário realizar um pré-processamento na imagem antes da detecção da região da placa. As placas possuem duas cores, onde geralmente os caracteres são escuros e o fundo é claro. Partindo deste princípio, as maiores intensidades da variação média de gradiente representam as regiões candidatas a placa em uma imagem.

Segundo Yang e Ma (2005), a região da placa tende a possuir um grande valor de gradiente vertical sobre a projeção horizontal. Entretanto, antes de buscar os maiores valores

da projeção é necessário aplicar o filtro de Gauss para eliminar ruídos na imagem. Após a aplicação do filtro de Gauss, obtém-se o resultado apresentado na Figura 13.



Fonte: Yang e Ma (2005, p. 91).

Figura 13 - Projeção horizontal após filtro de Gauss

As regiões candidatas à placa são obtidas através de dois princípios. O primeiro diz que a placa geralmente está na parte inferior das áreas candidatas da imagem. O segundo diz que os pontos com maior pico e mais próximos entre si, representam uma possível área horizontal da placa. Com estas informações, são copiadas da imagem original as regiões candidatas a placa (Figura 14). Em alguns veículos algumas regiões podem enganar o algoritmo, como por exemplo as janelas. Desta forma, podem existir mais do que uma região candidata.

Fonte: Yang e Ma (2005, p. 91).

Figura 14 - Região candidata obtida da imagem original

Caso sejam apresentadas muitas regiões candidatas ou então nenhuma região candidata à placa, será necessário verificar os parâmetros utilizados nos filtros e algoritmo.

2.5.2 Localização exata

Com o resultado da detecção aproximada, uma ou mais regiões candidatas são obtidas da imagem original. Nas regiões candidatas, novamente são verificadas as maiores intensidades da variação média de gradiente para localizar a projeção vertical da placa.

Para localizar a margem esquerda e direita da região candidata, após a introdução da morfologia matemática, é necessário tornar a imagem binária. Na Figura 15 as regiões brancas representam as regiões candidatas a posição vertical da placa. Para acelerar o reconhecimento e melhorar a precisão é utilizado o tamanho geométrico das placas automotivas, onde as

regiões muito grandes e muito pequenas são descartadas, restando apenas a área da placa.

Fonte: Yang e Ma (2005, p. 92).

Figura 15 - Imagem binária candidata

Com a localização da placa do veículo na imagem, é realizada a extração da mesma. Segundo Yang e Ma (2005, p. 93), nos testes realizados o algoritmo apresentou uma eficiência de 97,78%.

2.6 TRABALHOS CORRELATOS

Dentre os trabalhos pesquisados, os que mais se assemelham com o presente trabalho são: reconhecimento automático de placas de veículos (NAKASHIMA, 2004) e a ferramenta comercial Carmen FreeFlow (2008).

2.6.1 Reconhecimento automático de placas de veículos

Nakashima (2004) afirma que o protótipo construído do sistema de reconhecimento de placas utiliza técnicas de processamento de imagens e aprendizado de máquina. O protótipo tem como principais etapas:

a) conversão da imagem de entrada em níveis de cinza e aplicação de um operador de morfologia matemática;
b) transformação da imagem em uma imagem binária;
c) utilização de um operador de dilatação, fazendo com que os caracteres da placa se tornem um retângulo;
d) localização dos objetos conexos que tenham a razão entre suas dimensões próxima de um caractere;
e) utilização de redes neurais artificiais para a identificação dos caracteres da placa. Segundo Nakashima (2004, p. 16), as taxas de acerto são: localização da placa de 87,2% e reconhecimento dos caracteres de 69,5%.

2.6.2 Carmen FreeFlow – *Automatic Number Plate Recognition* (ANPR) software *package*

A API (Carmen FreeFlow, 2008) pode ser utilizada por qualquer aplicação que tem a necessidade do reconhecimento automático da placa de veículos. A aquisição do produto contempla: Carmen *Engine*[5] e *Software Development Kit* (SDK), contendo tutoriais e código fonte de exemplo em diversas linguagens. As principais características da API são:

a) reconhecimento de placas de qualquer país;
b) trabalha com imagens entre 300 x 200 e 2500 x 2000 *pixels*;
c) tempo de processamento de 100ms em uma *Central Processing Unit* (CPU) 500MHz, para uma imagem de 768 x 288 *pixels*;
d) roda nas plataformas WinNT, Win2k e Linux;
e) eficiência acima de 96% no reconhecimento;
f) custo da API é de cinco mil reais (R$ 5.000,00).

[5] Responsável pelo reconhecimento da placa do veículo.

3 DESENVOLVIMENTO DA FERRAMENTA

Neste capítulo são abordados os requisitos principais do problema, especificação, técnicas utilizadas para implementação da ferramenta, e por fim os resultados e discussões.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O presente trabalho possui como requisitos funcionais o Quadro 1 e como requisitos não funcionais o Quadro 2.

REQUISITOS FUNCIONAIS	CASO DE USO
RF01: Recuperar uma imagem contendo um veículo posicionado frontalmente.	UC01
RF02: Identificar a região da placa do veículo.	UC02, UC03
RF03: Realizar a identificação dos caracteres contidos na placa através de uma biblioteca OCR.	UC02, UC03
RF04: Apresentar para o usuário os caracteres reconhecidos da placa do veículo.	UC04

Quadro 1 – Requisitos funcionais

REQUISITOS NÃO FUNCIONAIS
RNF01: Implementar a ferramenta utilizando a tecnologia Java
RNF02: Utilizar a biblioteca gráfica JAI para manipulação da imagem

Quadro 2 – Requisitos não funcionais

3.2 ESPECIFICAÇÃO

A especificação do presente trabalho foi desenvolvida utilizando a notação UML (UML, 2002) em conjunto com a ferramenta JUDE (JUDE, 2008). São explanados diagramas de casos de uso, classe e seqüência.

3.2.1 Casos de uso

A ferramenta de reconhecimento de placas de veículos possui quatro casos de uso conforme a Figura 16, sendo que seguindo uma seqüência lógica, o primeiro e o segundo casos de uso devem ser obrigatoriamente executados pelo usuário. No terceiro caso de uso, o

usuário visualiza as etapas do processamento enquanto que no quarto o usuário obtém o resultado final.

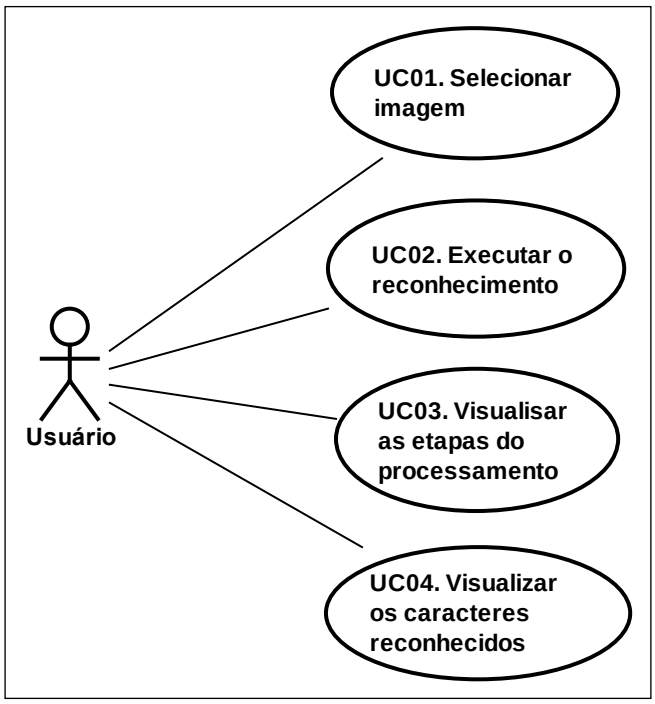


Figura 16 – Diagrama de casos de uso

3.2.1.1 Selecionar imagem

O primeiro caso de uso (Quadro 3), designado `Selecionar imagem`, descreve como o usuário seleciona a imagem a ser reconhecida. Além do cenário principal, o caso de uso possui um cenário de exceção.

UC01 – Selecionar imagem: possibilita ao usuário selecionar a imagem contendo um veículo no centro da mesma.	
Requisitos atendidos	RF01.
Pré-condições	O usuário possuir uma imagem em formato JPG ou BMP.
Cenário principal	1. Usuário solicita acesso a tela de seleção de imagens. 2. A ferramenta apresenta uma tela para que o usuário selecione arquivo (imagem) a ser reconhecida. 3. O usuário escolhe a imagem. 4. O sistema valida o formato do arquivo selecionado.
Exceção	No passo 3, caso o usuário escolha um arquivo diferente de JPG ou BMP, é apresentada uma mensagem informando o usuário sobre o formato inválido da imagem.
Pós-condições	A ferramenta apresenta a imagem selecionada.

Quadro 3 – Caso de uso `UC01`

3.2.1.2 Executar o reconhecimento

O segundo caso de uso (Quadro 4), designado `Executar o reconhecimento`, descreve como o usuário inicia o processamento da imagem a ser reconhecida.

UC02 – Executar o reconhecimento: permite que o usuário inicie o processo de reconhecimento da imagem.	
Requisitos atendidos	RF02, RF03.
Pré-condições	O usuário ter selecionado a imagem a ser reconhecida.
Cenário principal	1. Usuário solicita o inicio do reconhecimento. 2. A ferramenta apresenta uma barra de status informando o progresso do reconhecimento.
Exceção	Caso seja apresentado algum problema na execução, o reconhecimento é abortado e informado erro ao usuário.
Pós-condições	A ferramenta informa da conclusão do reconhecimento.

Quadro 4 – Caso de uso `UC02`

3.2.1.3 Visualizar as etapas do processamento

O terceiro caso de uso (Quadro 5), designado `Visualizar as etapas do processamento`, descreve como o usuário visualiza cada etapa do processamento.

UC03 – Visualizar as etapas do processamento: permite que o usuário visualize todas as etapas do processamento.	
Requisitos atendidos	RF02, RF03.
Pré-condições	O usuário ter executado o reconhecimento da imagem.
Cenário principal	1. O usuário seleciona a etapa que quer visualizar em detalhes. 2. A ferramenta apresenta a etapa selecionada em tamanho ampliado.

Quadro 5 – Caso de uso `UC03`

3.2.1.4 Visualizar os caracteres reconhecidos

O quarto caso de uso (Quadro 6), designado `Visualizar os caracteres reconhecidos`, descreve o resultado final do reconhecimento.

UC04 – Visualizar os caracteres reconhecidos: permite que o usuário visualize o resultado final do reconhecimento.	
Requisitos atendidos	RF04.
Pré-condições	O usuário ter executado o reconhecimento da imagem.
Cenário principal	1. O usuário visualiza os caracteres reconhecidos da placa do veículo.

Quadro 6 – Caso de uso `UC04`

3.2.2 Diagrama de classes

O diagrama de classes apresentado na Figura 17 apresenta uma visão de como as classes e pacotes estão estruturados. Nas seções seguintes são descritas como as classes se relacionam entre si para o funcionamento da ferramenta.

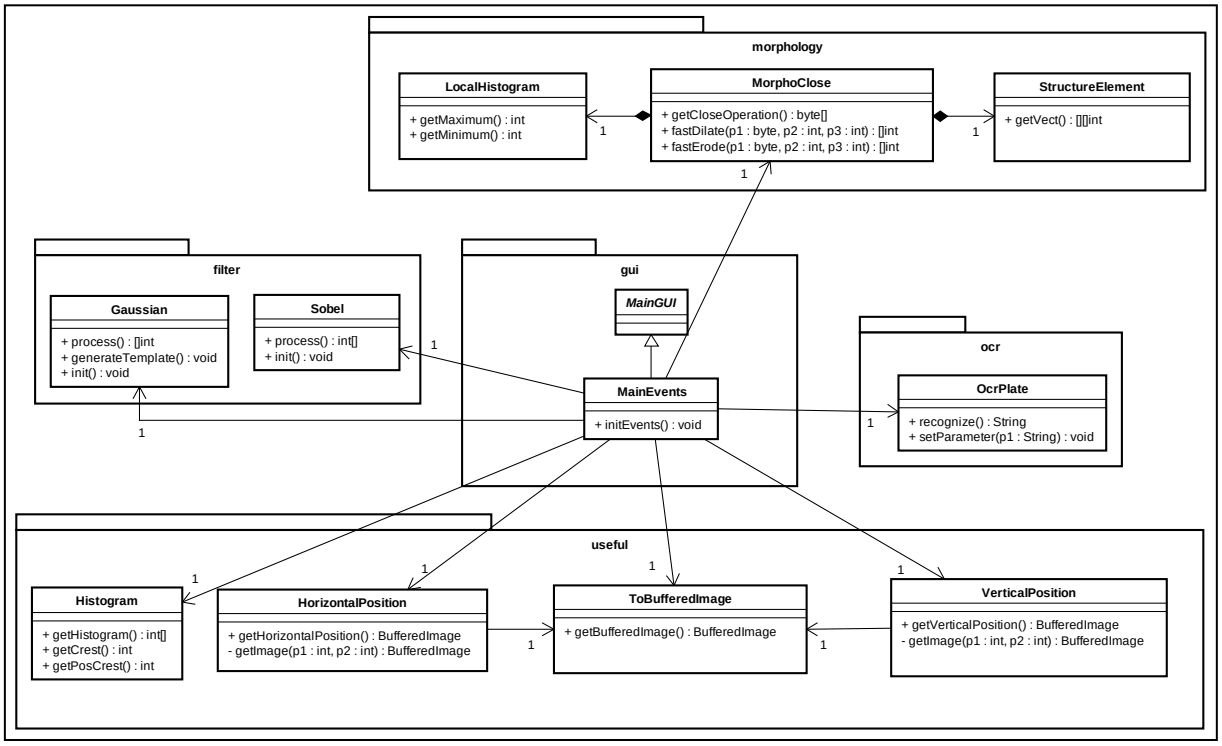


Figura 17 – Diagrama de classes

3.2.2.1 Pacote `gui`

O pacote `gui` é o pacote responsável pela interação com o usuário e pela captura dos eventos da tela. Nele estão contidas as classes `MainGUI` e `MainEvents`.

A classe `MainGUI` é uma classe abstrata que armazena todos os componentes e suas propriedades. A classe `MainGUI` é estendida pela classe `MainEvents` que possui a função de capturar os eventos da tela da ferramenta realizando o gerenciamento entre as demais classes.

3.2.2.2 Pacote `filter`

O pacote `filter` contém as classes `Gaussian` e `Sobel` que são responsáveis pelo pré-

processamento da imagem. A classe `Gaussian` (seção 2.5.1) é responsável pela aplicação do filtro de Gauss, sendo este o primeiro processo a ser aplicado na imagem com o objetivo de diminuir os ruídos. A classe `Sobel` (seção 2.2.4) é utilizada para aplicar o gradiente na imagem fazendo uso da máscara de Sobel. A classe `Sobel` possui um parâmetro onde é informado em qual eixo (`x` ou `y`) deseja-se aplicar o gradiente na imagem.

3.2.2.3 Pacote `useful`

O pacote `useful` é responsável por prestar suporte aos demais pacotes da ferramenta. A classe `ToBufferedImage` converte um *array* de bytes em uma `BufferedImage`, a ser apresentada para o usuário nas guias da ferramenta.

A classe `Histogram` é responsável por mapear as áreas de interesse da imagem. No eixo horizontal, a classe `Histogram` é utilizada como sucessora da classe `Sobel`, sendo utilizada como parâmetro para a classe `HorizontalPosition` para que seja extraída da imagem apenas a região de interesse. No eixo vertical, a classe `Histogram` é utilizada como sucessora da classe `MorphoClose` e como parâmetro para a classe `VerticalPosition`, para que seja extraída da imagem apenas a placa do veículo.

3.2.2.4 Pacote `morphology`

O pacote `morphology` possui as classes `MorphoClose`, `StructureElement` e `LocalHistogram`. O objetivo deste pacote é aplicar na imagem a operação de fechamento da morfologia matemática. As classes `StructureElement` e `LocalHistogram` são utilizadas pela classe `MorphoClose` para que seja realizada a operação de fechamento utilizando para a operação um elemento estruturante com máscara circular de raio 20.

3.2.2.5 Pacote `ocr`

O pacote `ocr` possui a classe `OcrPlate`, que é responsável por realizar o reconhecimento da imagem da placa do veículo. A classe `OcrPlate` utiliza a biblioteca

Microsoft Office Document Imaging (MODI) disponibilizada pelo Microsoft Office para realizar o reconhecimento (MICROSOFT, 2008). A integração entre o Java e a biblioteca é realizada através de uma DLL desenvolvida utilizado o *Borland Delphi Enterprise 7.0*. A DLL recebe o endereço em disco do arquivo contendo a placa do veículo a ser reconhecida, e retorna os caracteres reconhecidos pela biblioteca MODI para a aplicação.

3.2.3 Diagrama de seqüência

O diagrama de seqüência apresenta uma visão passo a passo do processo de troca de mensagens entre as classes. Na Figura 18 é apresentado o diagrama de seqüência do caso de uso UC02.

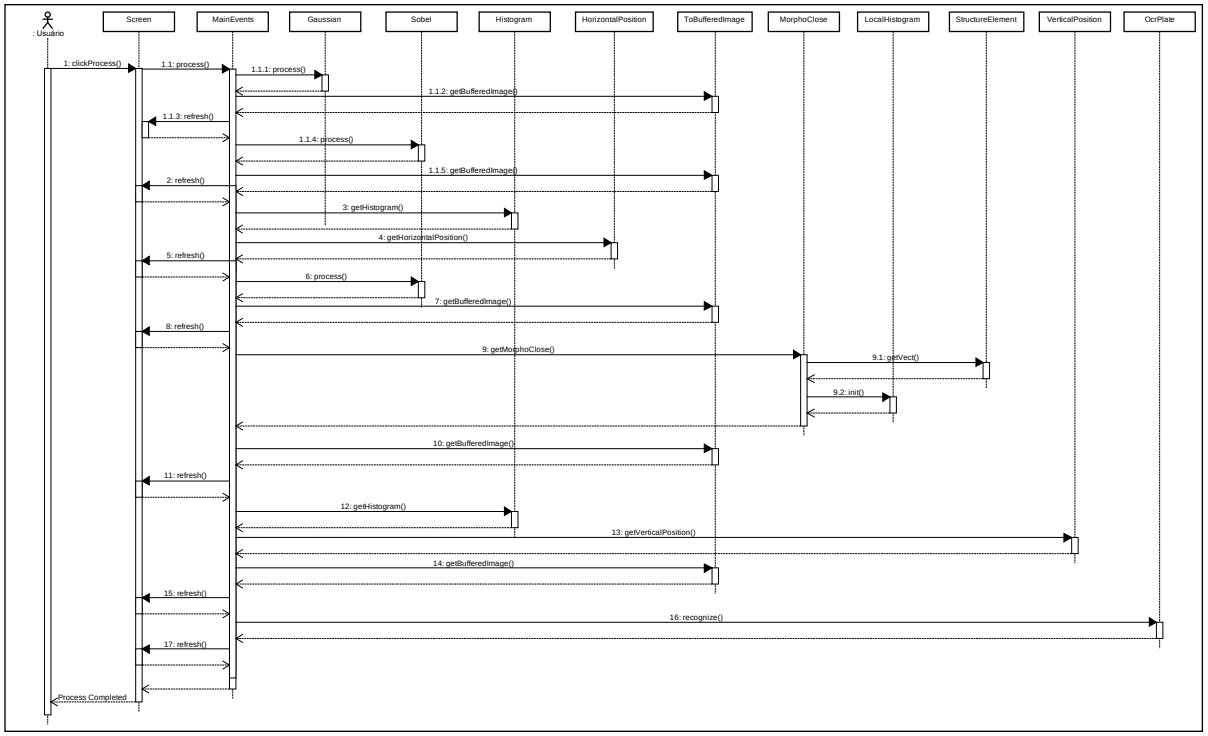


Figura 18 – Diagrama de seqüência

3.3 IMPLEMENTAÇÃO

A seguir são apresentadas as técnicas e ferramentas utilizadas, bem como a operacionalidade da ferramenta.

3.3.1 Técnicas e ferramentas utilizadas

Para a implementação da ferramenta foi utilizada a linguagem de programação Java, contando com a extensão JAI. O ambiente de desenvolvimento escolhido foi o Eclipse (ECLIPSE, 2008). Para o desenvolvimento da interface gráfica utilizou-se o *plugin* Jigloo (JIGLOO, 2004) para o Eclipse.

3.3.1.1 Filtro de Gauss

De acordo com Hutton e Dowling (2005), o operador Gaussiano possui como objetivo "borrar" a imagem digital eliminando ruídos. O Quadro 7 apresenta a inicialização do algoritmo tendo os seguintes parâmetros:

a) `original`: *array* que contém a imagem a ser processada;
b) `sigmaIn`: quantidade da "suavização" no processamento;
c) `tempSize`: tamanho do *template* a ser gerado;
d) `widthIn`: largura da imagem a ser processada;
e) `heightIn`: altura da imagem a ser processada.

```
public void init(int[] original, double sigmaIn, int tempSize, int widthIn, int heightIn) {
    sigma=sigmaIn;
    templateSize=tempSize;
    width=widthIn;
    height=heightIn;
    input = new int[width*height];
    output = new int[width*height];
    template = new float[templateSize*templateSize];
    input=original;
}
```

Quadro 7 - Inicialização do operador de Gauss

Após a inicialização, é necessário gerar o *template* para ser utilizado na imagem a ser processada. O *template* é gerado através da fórmula $\frac{1}{2\pi\sigma^2} exp\left\{-\frac{x^2+y^2}{2\sigma^2}\right\}$ conforme o Quadro 8. O sigma na fórmula representa a quantidade da "suavização" na imagem.

```
public void generateTemplate() {
    float center=(templateSize-1)/2;

    float total=0;

    for(int x = 0; x < templateSize; x++) {
        for(int y = 0; y < templateSize; y++) {
            template[x*templateSize+y] = (float)(1/(float)(2*Math.PI*sigma*sigma))*(float)
                                         Math.exp((float)(-((x-center)*(x-center)+(y-center)*
                                                           (y-center))/(2*sigma*sigma)));
            total+=template[x*templateSize+y];
        }
    }
    for(int x = 0; x < templateSize; x++) {
        for(int y = 0; y < templateSize; y++) {
            template[x*templateSize+y] = template[x*templateSize+y]/total;
        }
    }
}
```

Quadro 8 - Geração do *template*

Com o *template* gerado é realizado o processamento aplicando o *template* na imagem original conforme o Quadro 9. O resultado final é gravado na variável `outputsmaller` que é utilizada nos processos seqüentes na etapa do reconhecimento.

```
float sum;
int outputsmaller[] = new int[(width-(templateSize-1))*(height-(templateSize-1))];

for(int x=(templateSize-1)/2; x<width-(templateSize+1)/2;x++) {
    for(int y=(templateSize-1)/2; y<height-(templateSize+1)/2;y++) {
        sum=0;
        for(int x1=0;x1<templateSize;x1++) {
            for(int y1=0;y1<templateSize;y1++) {
                int x2 = (x-(templateSize-1)/2+x1);
                int y2 = (y-(templateSize-1)/2+y1);
                float value = (input[y2*width+x2] & 0xff) * (template[y1*templateSize+x1]);
                sum += value;
            }
        }
        outputsmaller[(y-(templateSize-1)/2)*(width-(templateSize-1))+
                      (x-(templateSize-1)/2)] = 0xff000000 | ((int)sum << 16 | (int)sum << 8 | (int)sum);
    }
}
```

Quadro 9 - Processamento do filtro de Gauss

3.3.1.2 Aplicação do gradiente no eixo vertical

Conforme mencionado na seção 2.2.4, para a detecção de bordas da imagem pode-se fazer uso do gradiente em conjunto com a máscara de Sobel. Para obter apenas a detecção de borda no eixo vertical foi utilizada uma máscara de Sobel de tamanho 3x3, conforme Quadro 10.

-1	0	1
-2	0	2
-1	0	1

Quadro 10 – Máscara de Sobel utilizada para detecção de bordas no eixo `y`

O Quadro 11 apresenta a inicialização do algoritmo de inicialização do gradiente no eixo vertical. O algoritmo possui os seguintes parâmetros:

a) `original`: *array* contendo a imagem a ser processada;
b) `widthIn`: largura da imagem a ser processada;
c) `heightIn`: altura da imagem a ser processada;
d) `eixo`: corresponde ao eixo em que a operação será realizada, onde no nosso caso é o eixo `y`.

```
public void init(int[] original, int widthIn, int heightIn, char eixo) {
    width=widthIn;
    height=heightIn;
    input = new int[width*height];
    output = new int[width*height];
    input=original;
    this.eixo = eixo;
}
```

Quadro 11 – Inicialização operação de Sobel

Após a inicialização, é realizado o processamento através do método `process()`, detalhado no Quadro 12. As linhas 33 e 34 do Quadro 12 são responsáveis por varrer cada *pixel* da imagem a ser processada. Na linha 36 o algoritmo verifica em qual eixo será realizado o processamento (neste caso no eixo `y`). Nas linhas 37 e 38 é processado cada valor da máscara e aplicada a fórmula mencionada na seção 2.2.4 ($M(x,y) \approx |(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)| + |(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)|$). Na linha 45 apenas são processados os valores correspondentes ao eixo `y`, no caso: $M(x,y) \approx (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$. Nas linhas 59 e 60, para cada pixel processado é realizada a junção entre a detecção de borda do eixo `x` e `y`. Porém conforme já mencionado, neste momento apenas é processada a detecção de borda do eixo `y`. Como o eixo `x` não foi processado, não possui nenhuma borda detectada. Por questões de desempenho a linha 61 foi otimizada para obter o valor absoluto entre a soma da detecção de borda do eixo `x` e `y`, completando então a fórmula $M(x,y) \approx |(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)| + |(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)|$. Antes de finalizar o processamento, das linhas 67 até a linha 72, os valores inteiros são convertidos para *pixels* no formato RGB.

```
25        public int[] process() {
26
27            float[] GY = new float[width*height];
28            float[] GX = new float[width*height];
29            int[] total = new int[width*height];
30            int sum=0;
31            int max=0;
32
33            for(int x=(templateSize-1)/2; x<width-(templateSize+1)/2;x++) {
34                for(int y=(templateSize-1)/2; y<height-(templateSize+1)/2;y++) {
35                    sum=0;
36                    if (eixo == 'y'){
37                        for(int x1=0;x1<templateSize;x1++) {
38                            for(int y1=0;y1<templateSize;y1++) {
39                                int x2 = (x-(templateSize-1)/2+x1);
40                                int y2 = (y-(templateSize-1)/2+y1);
41                                float value = (input[y2*width+x2] & 0xff) * (template[y1*templateSize+x1]);
42                                sum += value;
43                            }
44                        }
45                        GY[y*width+x] = sum;
46                    }else{
47                        for(int x1=0;x1<templateSize;x1++) {
48                            for(int y1=0;y1<templateSize;y1++) {
49                                int x2 = (x-(templateSize-1)/2+x1);
50                                int y2 = (y-(templateSize-1)/2+y1);
51                                float value = (input[y2*width+x2] & 0xff) * (template[x1*templateSize+y1]);
52                                sum += value;
53                            }
54                        }
55                        GX[y*width+x] = sum;
56                    }
57                }
58            }
59            for(int x=0; x<width;x++) {
60                for(int y=0; y<height;y++) {
61                    total[y*width+x]=(int)Math.sqrt(GX[y*width+x]*GX[y*width+x]+GY[y*width+x]*GY[y*width+x]);
62                    if(max<total[y*width+x])
63                        max=total[y*width+x];
64                }
65            }
66            float ratio=(float)max/255;
67            for(int x=0; x<width;x++) {
68                for(int y=0; y<height;y++) {
69                    sum=(int)(total[y*width+x]/ratio);
70                    output[y*width+x] = 0xff000000 | ((int)sum << 16 | (int)sum << 8 | (int)sum);
71                }
72            }
73            return output;
74        }
75    }
```

Quadro 12 – Processamento gradiente eixo vertical utilizando máscara de Sobel

3.3.1.3 Seleção da posição horizontal da imagem

Tendo como retorno o processamento do gradiente através da máscara de Sobel, é realizada a seleção da posição horizontal da imagem. Para isso é analisado o histograma da imagem através da classe `Histogram` possuindo os seguintes parâmetros de inicialização conforme o Quadro 13:

a) `pixel`: corresponde os *pixels* em que se deseja obter o histograma (retorno do processamento do item 3.3.1.2);

b) `w`: largura da imagem;

c) `h`: altura da imagem;

d) `e`: corresponde ao eixo em que deseja-se obter o histograma.

```
public Histogram(int[] pixel, int w, int h, char e) {
    eixo = e;
    arrayPixel = pixel;
    width = w;
    height = h;
    crest = 0;
    media = 0;
    halfImage = h / 2;
    init();
}
```

Quadro 13 – Inicialização do histograma

Após a inicialização, o início do processamento é realizado através do método `init()` no eixo `x` ou no eixo `y`, conforme Quadro 14 linha 23. Neste caso, o processamento é realizado no eixo `y`. Nas linhas 27 e 28 são percorridos todos os *pixels* da imagem passada como parâmetro. Na linha 29 é armazenada a soma dos *pixels* no eixo `x` para que nas linhas 33 a 38 seja identificado na imagem onde está localizada a maior concentração de bordas detectadas na horizontal da imagem. Observa-se que na linha 33 apenas a metade de baixo da imagem é utilizada para identificar a maior concentração da posição horizontal de bordas detectadas.

```
22    protected static void init() {
23        if (eixo == 'y') {
24            histogram = new int[height];
25            double tempMedia = 0;
26
27            for (int y = 0; y < height; y++) {
28                for (int x = 0; x < width; x++) {
29                    histogram[y] += (arrayPixel[y * width + x] & 0xff);
30                }
31                tempMedia += histogram[y];
32                // Obtém o pico apenas da metade da imagem para baixo...
33                if (y > halfImage) {
34                    if (crest < histogram[y]) {
35                        crest = histogram[y];
36                        posCrest = y;
37                    }
38                }
39                //System.out.println(y + ";" + histogram[y]);
40            }
41            media = (int) (tempMedia / height);
42        } else {
```

Quadro 14 – Processamento do histograma no eixo `y`

Imprimindo o processamento realizado através da linha 39 pode-se gerar um gráfico da soma das bordas detectadas (eixo `y`) pela altura da imagem (eixo `x`) apresentado no Quadro 15. É possível observar visualmente que a região horizontal candidata está aproximadamente

entre a altura 520 a 580.

Quadro 15 – Histograma eixo `y`

Tendo estas informações é utilizada a classe `HorizontalPosition` que extrai a região horizontal da imagem candidata a placa. A inicialização da classe é formada pelos seguintes parâmetros conforme (Quadro 16):

a) `pixel`: contém a imagem original em um *array* de inteiros;
b) `his`: corresponde a classe `Histogram` apresentada nos Quadro 13 e Quadro 14;
c) `w`: largura da imagem;
d) `h`: altura da imagem;
e) `m`: média da detecção de borda apresentada no Quadro 15;
f) `p`: corresponde a posição na imagem onde ocorreu a maior concentração de bordas detectadas horizontalmente na imagem.

```
public HorizontalPosition(int[] pixel, int[] his, int w, int h, int m, int p){
    arrayPixel = pixel;
    histogram = his;
    width = w;
    height = h;
    media = m;
    posCrest = p;
}
```

Quadro 16 – Inicialização do `HorizontalPosition`

Após a inicialização, é invocado o método `getHorizontalPosition()` que retorna a região horizontal da imagem (Quadro 17). Ainda no Quadro 17, da linha 26 a 34, é obtida a posição inicial da região horizontal, onde o corte ocorre no quinto *pixel* que sucede a média dos valores da detecção de borda representada no Quadro 15. A posição horizontal final é realizada da linha 36 a 44, seguindo a mesma lógica realizada para obter a posição horizontal

inicial da placa do veículo. Por fim, o retorno da imagem é realizado na linha 46 do Quadro 17.

```
21      public BufferedImage getHorizontalPosition(){
22          int posIni = 0, posFim = 0, count = 0;
23
24          //Obtem a posição inicial da imagem;
25          count = 0;
26          for(int y=posCrest; y>0; y--) {
27              if (histogram[y] < media){
28                  count++;
29                  if (count > 5){
30                      posIni = y;
31                      break;
32                  }
33              }
34          }
35          count = 0;
36          for(int y=posCrest; y<height;y++) {
37              if (histogram[y] < media){
38                  count++;
39                  if (count > 5){
40                      posFim = y;
41                      break;
42                  }
43              }
44          }
45
46          return getImage(posIni, posFim);
47      }
```

Quadro 17 – Método `getHorizontalPosition()`

O método `getImage` é detalhado no Quadro 18, onde da linha 53 a 57, é realizada uma consistência verificando se a posição final da imagem foi identificada. Caso negativo, a posição final é deduzida pela quantidade de *pixels* que foram utilizados para realizar o corte inicial da imagem. A segunda consistência é realizada da linha 63 a 65, onde caso não foi identificada a posição inicial ou a mesma é muito grande, a posição inicial é deduzida pela quantidade de pixels que foram utilizados para realizar o corte final da imagem. Por fim, da linha 77 a 80 são realizados os tratamentos necessários para o retorno, onde primeiramente o *array* de inteiros é convertido para uma *Image*, e posteriormente é convertido para *BufferedImage*.

```
49      private BufferedImage getImage(int posIni, int posFim){
50          Image newImage = null;
51
52          //Se não encontrou pela média... Isso pode ocorrer quando o carro possui sombra
53          if (posFim == 0){
54              posFim = posCrest + (posCrest-posIni) + 5;
55              if (posFim > height)
56                  posFim = height;
57          }
58
59          int difPosIni = posCrest - posIni;
60          int difPosFim = posFim - posCrest;
61
62          //não achou posição inicial, ou uma posição inicial muito grande...
63          if ((posIni == 0) || (difPosIni > (difPosFim*3))){
64              posIni = (posCrest - (posFim - posCrest)) + 5;
65          }
66          int newHeight = posFim - posIni;
67
68          //Array de Bytes de retorno
69          int[] output = new int[width*newHeight];
70
71          for(int x=0; x<width;x++) {
72              for(int y=0; y<newHeight;y++) {
73                  output[y*width+x] = arrayPixel[(y+posIni)*width+x];
74              }
75          }
76          //Converte os bytes para Imagem.
77          newImage = Toolkit.getDefaultToolkit().createImage(new MemoryImageSource(width, newHeight, output, 0, width));
78          //Converte a imagem para bufferedImage
79          ToBufferedImage bfImage = new ToBufferedImage(newImage);
80          return  bfImage.getBufferedImage();
81      }
82  }
```

Quadro 18 – Método `getImage()`

O retorno do método `getImage()` é apresentado na Figura 19.

Figura 19 – Posição horizontal obtida da imagem original

3.3.1.4 Aplicação do gradiente no eixo horizontal

A aplicação do gradiente no eixo horizontal é realizado na imagem da Figura 19. O procedimento a ser executado é o mesmo da seção 3.3.1.2, porém agora para o eixo `x` utilizando a máscara de Sobel apresentada no Quadro 19.

1	2	1
0	0	0
-1	-2	-1

Quadro 19 – Máscara de Sobel utilizada para detecção de bordas no eixo `x`

Após a aplicação do operador de gradiente no eixo `x`, obtém-se apenas as bordas horizontais apresentadas na Figura 20.

Figura 20 – Resultado da aplicação do gradiente no eixo `x`

3.3.1.5 Morfologia matemática

Conforme mencionado na seção 2.3.2, a operação de fechamento da morfologia matemática pode ser utilizada para as seguintes situações:

a) gerar fusão em ligações quebradas;

b) eliminar pequenos buracos na imagem;

c) preencher lacunas faltantes na imagem.

É possível observar na Figura 20 que a aplicação do fechamento morfológico pode ser utilizada para preencher as lacunas na placa do veículo a fim de destacar a região da placa. Para a aplicação deste processamento foi extraído o código do processamento morfológico da ferramenta ImageJ[6] (IMAGEJ, 2008). Para realizar a operação de fechamento, é realizada a operação de dilatação seguida da operação de erosão, conforme apresentado no Quadro 20 linhas 54 e 55.

```
37    public byte[] getCloseOperation() {
38
39        int width = input.getWidth(null);
40        int height = input.getHeight(null);
41
42        // Converte os pixels para byte
43        byte[] pixels = new byte[width * height];
44        PixelGrabber pGrabber = new PixelGrabber(input, 0, 0, width, height, false);
45        try {
46            pGrabber.grabPixels();
47        } catch (InterruptedException e) {
48            System.err.println(e);
49        };
50
51        int []pixelsRGB = (int[]) pGrabber.getPixels();
52        pixels = convertRGBToByte(pixelsRGB);
53
54        pixels = fastDilate(pixels, width, height);
55        pixels = fastErode(pixels, width, height);
56
57        return pixels;
58    }
```

Quadro 20 – Método `getCloseOperation()`

O resultado final do método `getCloseOperation()` é apresentado na Figura 21. A região destacada em branco é a região candidata a placa na imagem.

Figura 21 – Retorno do método `getCloseOperation()`

[6] ImageJ é uma ferramenta de processamento de imagens baseada em Java com domínio público.

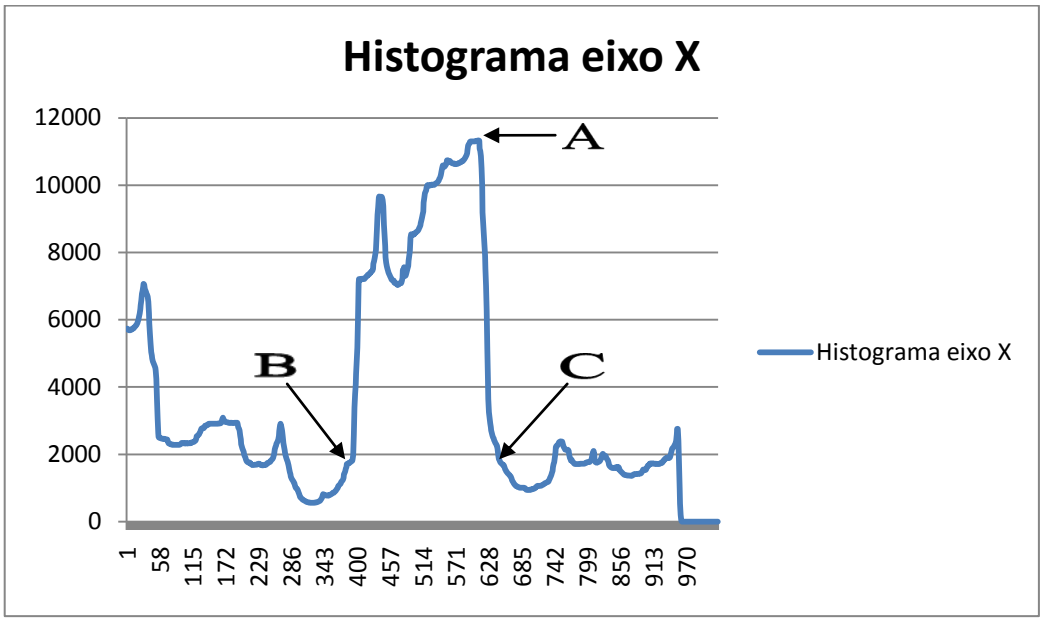
3.3.1.6 Obtendo a posição vertical da placa

Para obter a posição vertical da placa, é utilizado o retorno do método `getCloseOperation()` apresentado na Figura 21 para que seja processado o histograma da imagem através da classe `Histogram` cuja inicialização desta classe foi apresentada no Quadro 13. Após a inicialização da classe, inicia-se o processamento para o eixo `x` da imagem conforme o Quadro 21. O primeiro passo é varrer todos os *pixels* da imagem no eixo `x` (vertical) a fim de criar um histograma da distribuição dos tons de cinza para identificar as regiões de interesse. Essa verificação é realizada da linha 51 a 61. Da linha 65 a 88 são selecionadas as regiões candidatas a placa, sendo que é considerada como uma região candidata aquela que possuir a soma dos *pixels* no eixo `x` maior do que a média da soma dos *pixels* do eixo `x` de toda a imagem. Após esta etapa, da linha 90 até a linha 111 verificada a existência de regiões candidatas a placa do veículo. Existindo mais do que uma região candidata é selecionada a que possuir o maior comprimento, pois as outras regiões possivelmente são caracteres informativos do veículo como a cilindrada, modelo ou então adesivos em geral.

```
...
42          } else {
43              boolean acimaMedia = false;
44              int[] faixa = new int[2];
45              ArrayList<int[]> array = new ArrayList<int[]>();
46
47              // Soma no eixo X
48              histogram = new int[width];
49              double tempMedia = 0;
50
51              for (int x = 0; x < width; x++) {
52                  for (int y = 0; y < height; y++) {
53                      histogram[x] += (arrayPixel[y * width + x] & 0xff);
54                  }
55                  tempMedia += histogram[x];
56                  if (crest < histogram[x]) {
57                      crest = histogram[x];
58                      posCrest = x;
59                  }
60                  //System.out.println(x + ";" + histogram[x]);
61              }
62              media = (int) (tempMedia / width);
63
64              // verifica a maior faixa acima da média
65              for (int x = 0; x < histogram.length; x++) {
66                  if (histogram[x] > media) {
67                      if (acimaMedia) {
68                          continue;
69                      } else {
70                          // Início de uma faixa candidata
71                          acimaMedia = true;
72                          faixa[0] = x;
73                      }
74
75                  } else {
76                      if (!acimaMedia) {
77                          continue;
78                      } else {
79                          // Final de uma faixa candidata
80                          acimaMedia = false;
81                          faixa[1] = x;
82                          int[] f = new int[2];
83                          f[0] = faixa[0];
84                          f[1] = faixa[1];
85                          array.add(f);
86                      }
87                  }
88              }
89
90              if (array.size() > 1) {
91                  // pesquisa qual a maior faixa da placa para buscar o pico
92                  int ini = 0, fim = 0;
93                  for (int i = 0; i < array.size(); i++) {
94
95                      int[] placa = (int[]) array.get(i);
96
97                      if ((fim - ini) < (placa[1] - placa[0])) {
98                          ini = placa[0];
99                          fim = placa[1];
100                     }
101                 }
102
103                 // tendo a maior faixa, busca o maior pico
104                 crest = 0;
105                 for (int x = ini; x < fim; x++) {
106                     if (crest < histogram[x]) {
107                         crest = histogram[x];
108                         posCrest = x;
109                     }
110                 }
111             }
112         }
113     }
```

Quadro 21 – Processamento do histograma no eixo x

Um exemplo do histograma gerado na linha 60 (do Quadro 21) é apresentado no Quadro 22, onde visualmente é possível identificar que a região da placa está aproximadamente entre as posições 400 e 630 (`B` e `C` do Quadro 22).



Quadro 22 – Histograma eixo `x`

Com o histograma processado, é utilizada a classe `VerticalPosition` para retirar da imagem a região da placa através do método `getVerticalPosition()` conforme o Quadro 23. O mesmo principio utilizado para identificar a posição horizontal da placa é aplicado para identificar a posição vertical. Nas linhas 26 a 34 do Quadro 23 é identificada a coordenada inicial a esquerda do pico do histograma (Quadro 22, A). Ao final do processamento, obtém-se a posição inicial da placa do veículo (Quadro 22, B). Da linha 37 a 45 do Quadro 23, a partir do pico do histograma, obtém-se a coordenada final da placa do veículo (Quadro 22, C). Após definida a posição inicial e final, na linha 46 do Quadro 23 é invocado o método `getImage()` passando como parâmetro a posição `x` inicial e final da placa para que então seja retornada apenas a placa do veículo.

```
21    public BufferedImage getVerticalPosition(){
22        int posIni = 0, posFim = 0, count = 0;
23        int mediaComp = (int) (media * 1.35);
24        //Obtem a posição inicial da imagem;
25        count = 0;
26        for(int x=posCrest; x>0; x--) {
27            if (histogram[x] < mediaComp){
28                count++;
29                if (count > 15){
30                    posIni = x;
31                    break;
32                }
33            }
34        }
35        count = 0;
36        //Obtem a posição final da imagem;
37        for(int x=posCrest; x<width;x++) {
38            if (histogram[x] < mediaComp){
39                count++;
40                if (count > 15){
41                    posFim = x;
42                    break;
43                }
44            }
45        }
46        return getImage(posIni, posFim);
47    }
```

Quadro 23 – Método `getVerticalPosition()`

O resultado final da etapa de localização vertical da placa do veículo é apresentado na Figura 22.

Figura 22 – Resultado da extração da placa do veículo

3.3.1.7 Reconhecendo os caracteres

Para o reconhecimento dos caracteres foi utilizada a biblioteca Microsoft Office Document Imaging (MODI), que é uma biblioteca fornecida pela Microsoft para os usuários que possuem o pacote Office. De acordo com (MICROSOFT, 2008), a biblioteca possibilita o desenvolvimento de aplicações para reconhecimento de caracteres texto em imagens digitalizadas. Para fazer a *interface* entre a biblioteca e a aplicação, foi construída uma DLL através da ferramenta *Borland Delphi Enterprise 7.0*. No Quadro 24 é apresentado o código responsável pelo reconhecimento.

Primeiro é criada uma instância da biblioteca MODI, e passado como parâmetro o local em disco em que está a imagem a ser reconhecida (através da variável `aLocalFoto`). Em seguida é invocado o método `OCR` com os seguintes parâmetros:

a) `LangId`: identificador do idioma a ser utilizado para o reconhecimento. Embora não sejam reconhecidos textos que formam palavras de um determinado idioma, foi informado como idioma a língua portuguesa;
b) `OCROrientImage`: especifica se o motor OCR precisa determinar a orientação da página a ser reconhecida;
c) `OCRStraightenImage`: determina se o motor OCR precisa corrigir pequenos ângulos de desalinhamento em relação a vertical.

Após a execução do `OCR`, são obtidos os caracteres reconhecidos. Caso a biblioteca não consiga realizar o reconhecimento, a exceção é tratada e a DLL retorna a string 'Erro' para a aplicação, conforme apresentado no Quadro 24.

```
try
  doc := IDispatch(CreateOleObject('MODI.Document')) as IDocument;
  doc.create(aLocalFoto);
  doc.OCR(miLANG_PORTUGUESE,false,true);
  Img := IDispatch(doc.Images[0]) as IImage;
  Layout := IDispatch(Img.Layout) as ILayout;
  xString := Layout.Text;
except
  xString := 'Erro';
end;
```

Quadro 24 – Utilização da biblioteca MODI através da DLL construída em Delphi

Na ferramenta a classe `OcrPlate` é a responsável por obter os caracteres reconhecidos através da DLL conforme o Quadro 25. Na linha 17 é invocado o método da DLL passando como parâmetro o diretório e nome do arquivo em que a Figura 22 está armazenada. Na linha 18 são retirados os espaços em branco antes e depois dos caracteres reconhecidos. Como é de conhecimento, conforme informado na seção 2.1, a placa do veículo é formada por sete (7) caracteres alfanuméricos individualizados, sendo o primeiro grupo composto por três (3), resultante do arranjo, com repetição de vinte e seis (26) letras, tomadas três a três. O segundo grupo é composto por quatro (4), resultante do arranjo, com repetição, de dez (10) algarismos, tomados quatro a quatro.

Dessa forma, da linha 19 a 43 (Quadro 25) é realizado um filtro sobre os caracteres reconhecidos a fim de eliminar caracteres que não fazem parte da placa, aumentando assim a eficiência final do reconhecimento.

```
14    public String Recognize(){
15        String retorno, xresult = "";
16        int count=1;
17        retorno = Recognize (parametro);
18        retorno = retorno.trim();
19        for (int i = retorno.length() - 1; i >= 0; i--) {
20            //The last four character are the numbers of license
21            if (count > 4) {
22                char letter;
23                letter = retorno.substring(i, i+1).charAt(0);
24                if (Character.isLetter(letter)){
25                    xresult = retorno.substring(i, i+1) + xresult;
26                    count++;
27                if (count > 7)
28                    break;
29                }else{
30                    continue;
31                }
32
33            }else{
34                char number;
35                number = retorno.substring(i, i+1).charAt(0);
36                if (Character.isLetterOrDigit(number)){
37                    xresult = retorno.substring(i, i+1) + xresult;
38                    count++;
39                }else{
40                    continue;
41                }
42            }
43        }
44        return xresult;
45    }
```

Quadro 25 – Método `Recognize()` da classe `OcrPlate`

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 Abrindo uma foto para reconhecimento dos caracteres

Na tela principal da ferramenta, na aba `File`, o usuário tem a opção para abrir uma foto ou sair do programa. A Figura 23 demonstra esta situação.

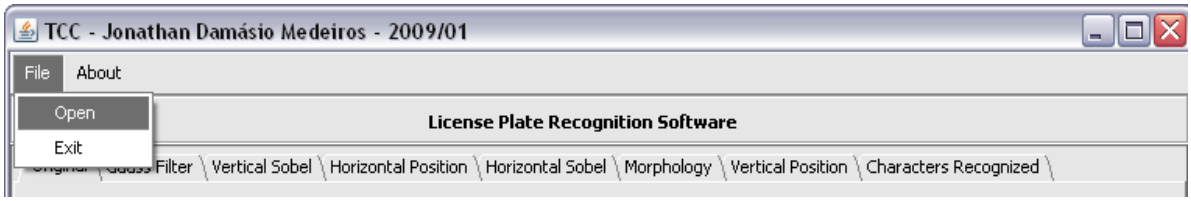


Figura 23 – Tela menu `File`

Ao selecionar uma imagem, a resolução mínima aceita pela ferramenta é de 800x600 *pixels*. Caso o usuário selecione uma imagem com resolução menor do que a mínima exigida, uma mensagem é exibida conforme Figura 24.

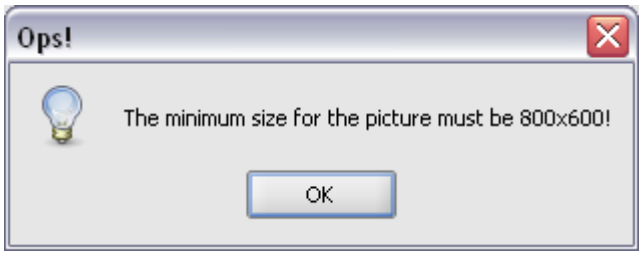


Figura 24 – Tela de informação para resolução abaixo do mínimo exigido

Ao selecionar uma imagem, se o arquivo selecionado não corresponder a uma imagem, é exibida uma mensagem conforme Figura 25.

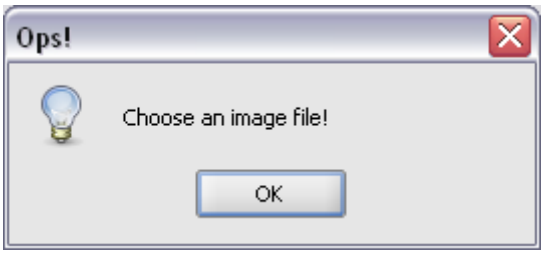


Figura 25 – Tela de informação para arquivos que não sejam imagens

Se as condições referentes à Figura 24 e Figura 25 foram satisfeitas, então a ferramenta exibe a foto selecionada pelo usuário conforme Figura 26.

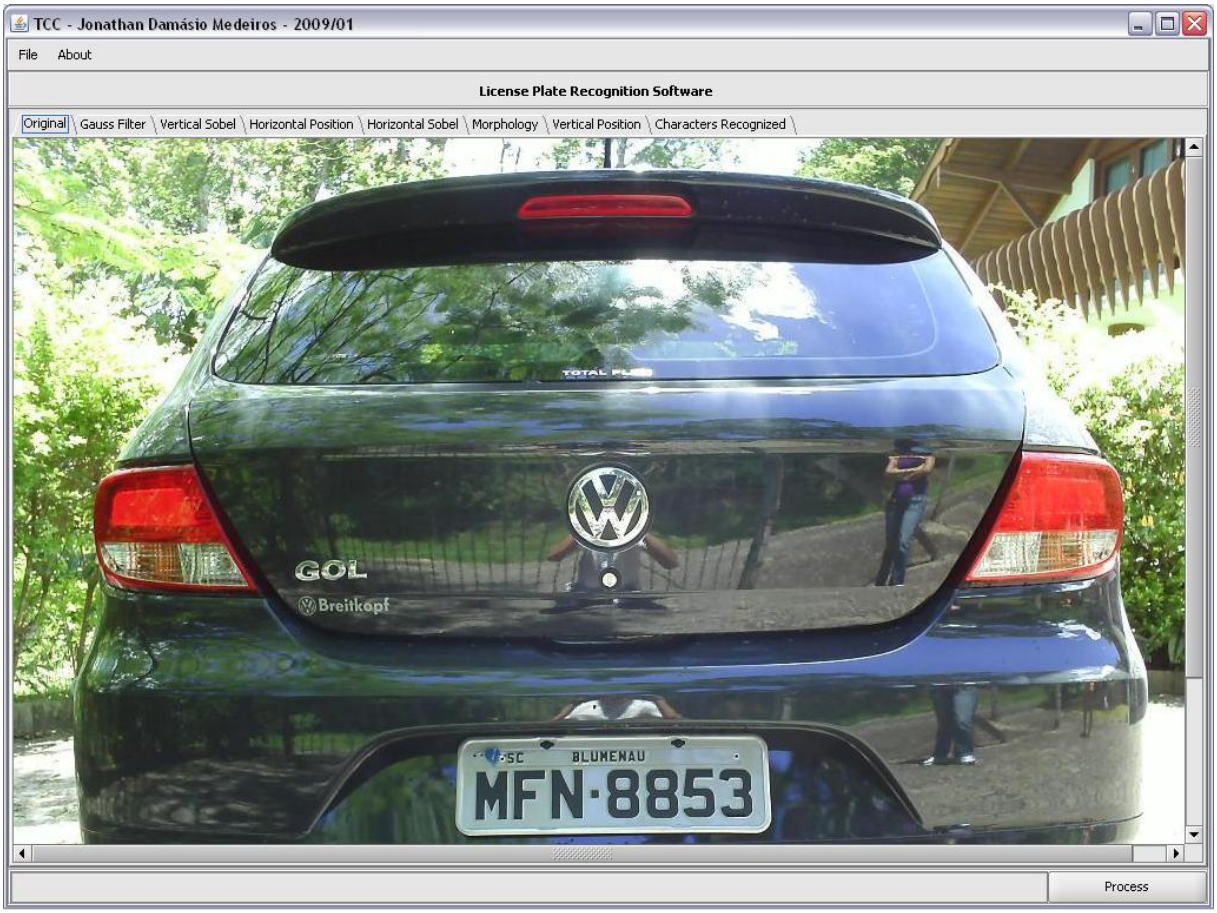


Figura 26 – Tela principal da ferramenta com a foto selecionada pelo usuário

3.3.2.2 Realizando o processamento na imagem selecionada

Para iniciar o processamento é necessário ter selecionado uma imagem, caso contrário o botão `Process` não é habilitado. Ao clicar no botão `Process` a ferramenta realiza os seguintes procedimentos:

a) todas as etapas descritas no item 3.3.1, apresentando os resultados em guias;
b) atualização em tempo real da duração do processamento de cada etapa;
c) andamento do processamento através da barra de progressão;
d) mensagem ao usuário informando o fim do processamento conforme Figura 27.

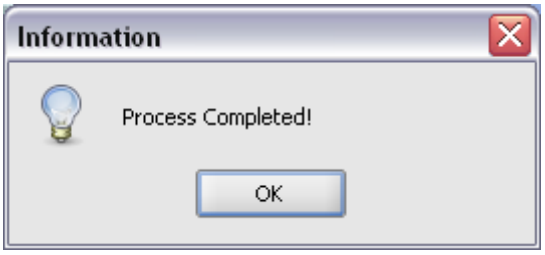


Figura 27 – Tela informando a conclusão do processamento

3.3.2.3 Verificando o resultado do processamento

O resultado do processamento pode ser observado nas guias da ferramenta. A primeira é a aplicação do filtro de Gauss conforme Figura 28.

Figura 28 – Resultado da aplicação do filtro de Gauss

Na Figura 29 é apresentado o resultado da aplicação do gradiente utilizando a máscara de Sobel para detectar as bordas verticais da imagem.

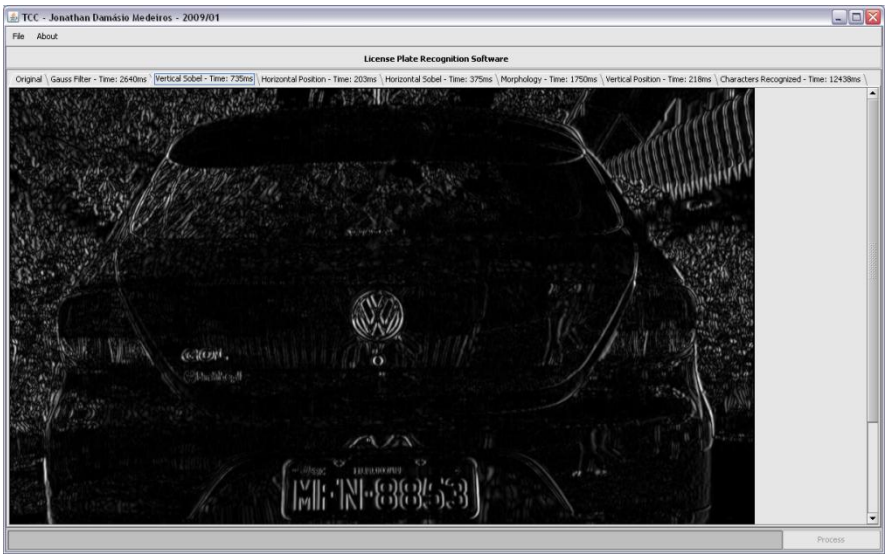


Figura 29 – Resultado da aplicação do gradiente utilizando Sobel

Na Figura 30 é apresentada a imagem horizontal contendo a placa do veículo a ser reconhecida.

Figura 30 – Resultado localização horizontal da placa do veículo

Na Figura 31 é apresentado o resultado da aplicação do gradiente utilizando a máscara de Sobel para detectar as bordas horizontais da imagem.

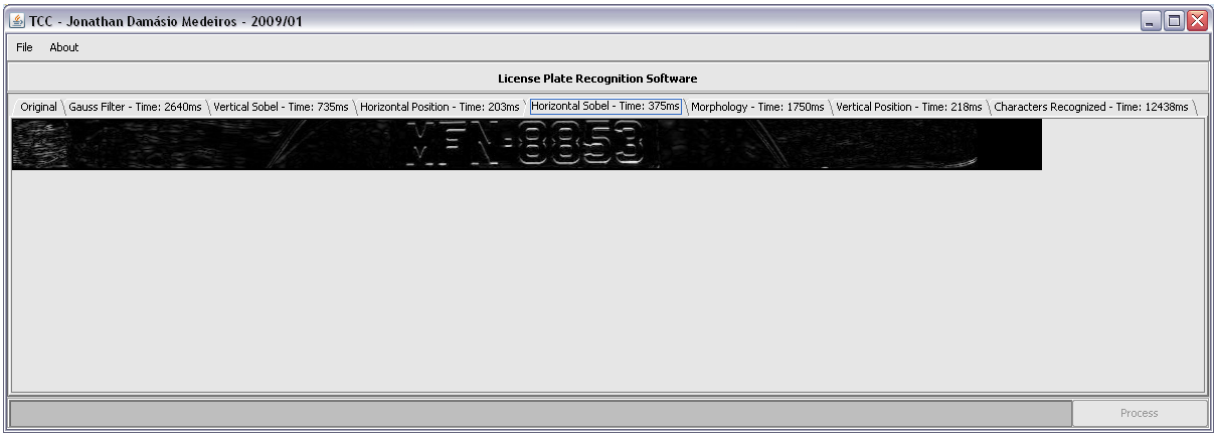


Figura 31 – Resultado da aplicação do gradiente utilizando Sobel

Na Figura 32 é apresentado o resultado da morfologia matemática, usando a operação de fechamento.

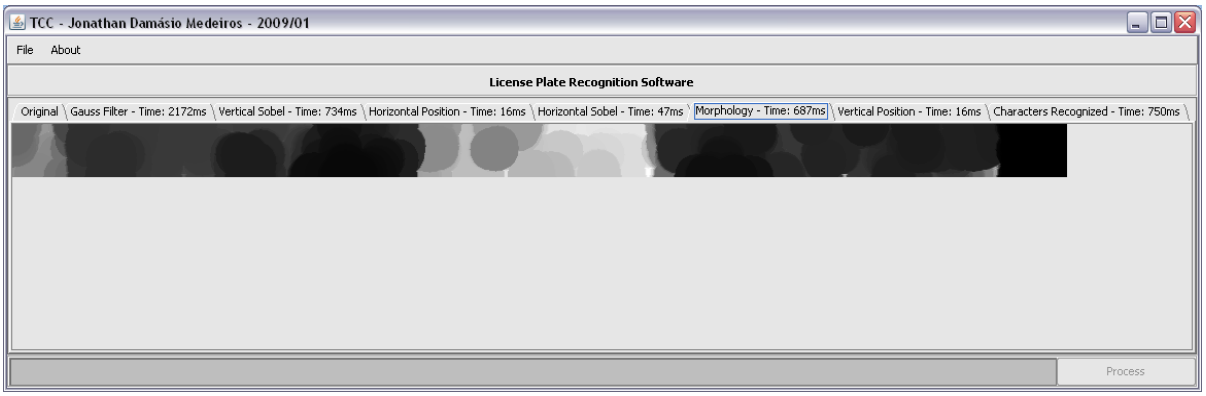


Figura 32 – Resultado da morfologia matemática

O resultado da posição vertical da placa é apresentado na guia `Vertical Position` conforme a Figura 33.

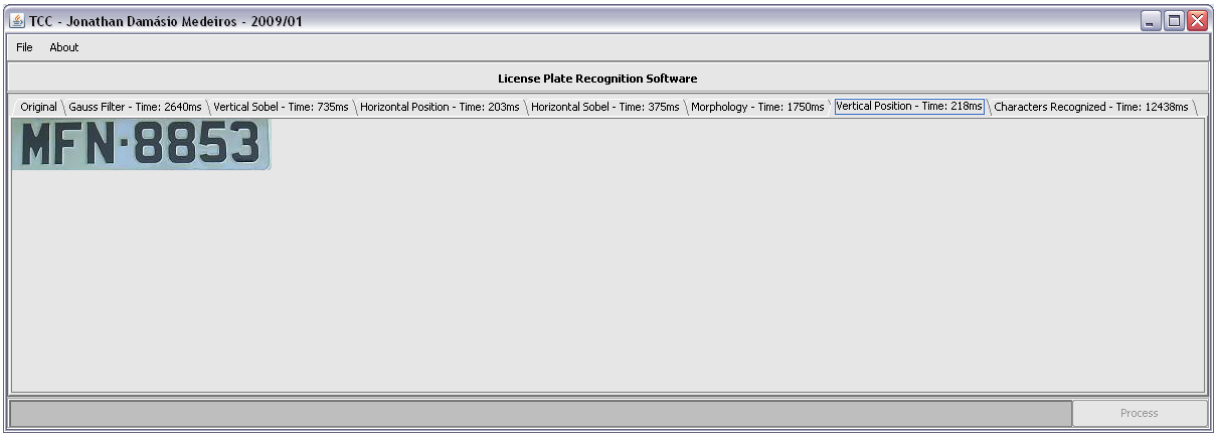


Figura 33 – Extração da placa da imagem

Na Figura 34 são apresentados os caracteres reconhecidos pela biblioteca através da utilização da DLL, conforme descrito no item 3.3.1.7.

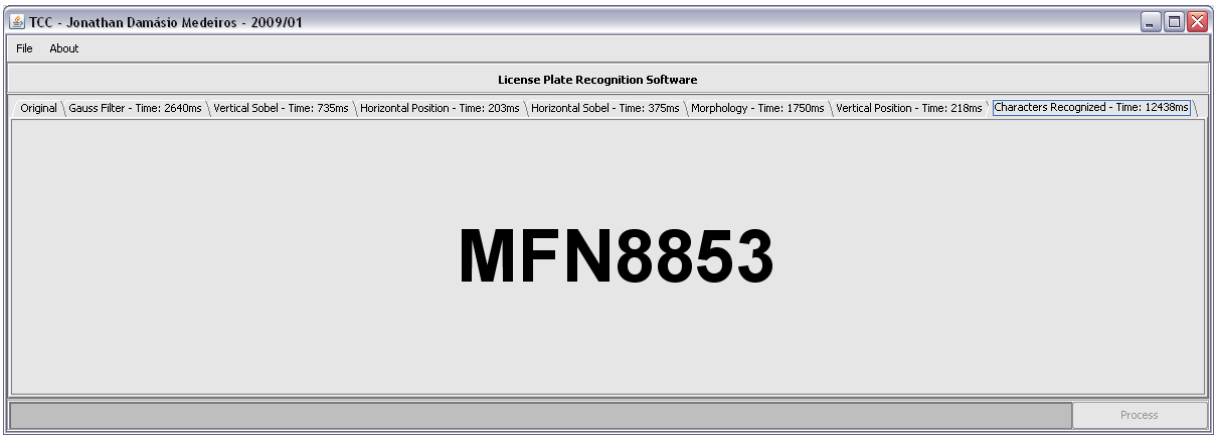


Figura 34 – Caracteres reconhecidos

3.4 RESULTADOS E DISCUSSÃO

Esta seção tem por objetivo apresentar os resultados e discussões obtidos através de testes realizados na ferramenta. Nas próximas seções serão abordadas questões de performance e precisão no reconhecimento.

3.4.1 Tempo de processamento

Para obter o tempo total de reconhecimento de uma imagem, foi utilizada uma imagem de tamanho 1024x768 para o teste. Antes de iniciar a tomada de tempo, foram descartados os

dois primeiros processamentos, pois nos testes realizados, a partir do segundo processamento a ferramenta apresenta estabilidade no tempo do reconhecimento. A média do tempo de processamento foi de 3.672 milissegundos. Para obter o tempo de processamento por *pixel* multiplica-se a base vezes a altura da imagem (1024x768) que é igual a 786.432 *pixels*. Após isto, é realizada uma regra de três onde obtém-se o tempo de processamento para 1 *pixel*, que é de 0,00467 milissegundos. Dessa forma, para uma imagem de 800x600, o tempo estimado de processamento é de 2.242 milissegundos.

3.4.2 Desempenho da ferramenta

Para verificar o tempo de processamento de cada etapa descrita no item 3.3.1, foram realizadas tomadas de tempo no início e no final de cada processamento, a fim de verificar qual é a etapa em que está sendo consumido o maior e o menor tempo de processamento. Os resultados dos testes estão apresentados no Quadro 26.

Picture	Size (pixels)	Gauss(ms)	Ver. Sobel(ms)	Hor. Pos.(ms)	Hor. Sobel(ms)	Mor.(ms)	Ver. Pos.(ms)	Car. Rec.(ms)	Total
Veículo 1	800x600	875	172	16	1	344	15	1078	2501
Veículo 2	800x601	891	172	62	16	422	15	998	2576
Veículo 3	1024x768	1453	531	16	31	625	16	1000	3672
Veículo 4	1024x768	1422	547	10	47	1125	15	1141	4307
Processamento total (ms):		4641	1422	104	95	2516	61	4217	13056
Processamento total (%):		**35,55**	**10,89**	**0,80**	**0,73**	**19,27**	**0,47**	**32,30**	**100,00**

Quadro 26 – Resultado dos testes de performance

O tempo total de processamento não depende apenas do tamanho da imagem, mas também da sub-imagem obtida na etapa da localização horizontal (seção 3.3.1.3). No Quadro 26 é apresentada esta situação, onde embora o tamanho das imagens dos veículos três e quatro serem os mesmos (1024x768), o tempo de processamento variou aproximadamente 17%. Isto ocorre pelo motivo de que a etapa de morfologia matemática (seção 3.3.1.5) é processada para uma quantidade maior ou menor de *pixels*. Como exemplo, a Figura 35 apresenta a região horizontal da placa dos veículos três e quatro. Devido ao tamanho maior da sub-imagem do veículo quatro em relação ao três, o tempo de processamento do veículo quatro na etapa de morfologia matemática é maior.

Figura 35 – Localização horizontal veículo três e quatro

Podemos observar também (Quadro 26) que as rotinas que apresentaram maior tempo

de processamento foram o filtro de Gauss com 35,55% e o reconhecimento dos caracteres com 32,30%, somando juntos 67,85% do tempo de processamento da ferramenta.

3.4.3 Precisão no reconhecimento da placa do veículo

Para o reconhecimento, é importante que a placa do veículo esteja bem focalizada na imagem. Identificou-se que o algoritmo não reconhece a região da placa quando a superfície em que o veículo está é formada por muitas bordas, conforme a Figura 36.

Figura 36 – Foto do veículo sobre uma superfície com várias bordas

Observa-se na Figura 36 que a região que mais apresentou destaque na detecção das bordas foi a calçada e não a placa do veículo. Para contornar esta situação, foi aplicado na imagem através da ferramenta ImageJ (IMAGEJ, 2008) o ajuste de brilho e contraste através das teclas de atalho (Ctrl+Shift+C), apresentado na Figura 37.

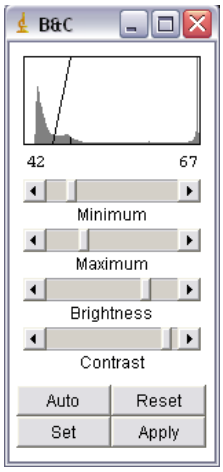


Figura 37 – Ajuste de brilho e contraste aplicado na imagem

Com o ajuste aplicado na Figura 37, a ferramenta realizou o reconhecimento da região horizontal da placa do veículo corretamente, conforme Figura 38.

Figura 38 – Região horizontal da placa identificada pela ferramenta

3.4.4 Precisão no reconhecimento dos caracteres da placa

O reconhecimento da placa do veículo utilizando a biblioteca a MODI apresentou um resultado satisfatório, porém em algumas situações a biblioteca não consegue realizar o reconhecimento de imagens que são visualmente semelhantes. Um exemplo disto é a Figura 39, que a biblioteca não consegue realizar o reconhecimento, gerando como resposta os caracteres ‘uLR3OZz’. Mesmo realizando ajustes no contraste e no brilho da imagem, o resultado não foi satisfatório.

Figura 39 – Reconhecimento não realizado corretamente

A Figura 40 é outra foto semelhante a da Figura 39, com a diferença de que a biblioteca MODI realizou o reconhecimento dos caracteres corretamente.

Figura 40 – Reconhecimento realizado corretamente

4 CONCLUSÕES

A concepção de uma ferramenta que realiza uma atividade através da simulação da visão humana em um ambiente dinâmico envolve o processamento de imagens. No reconhecimento de inúmeras placas de veículos, geralmente uma ferramenta desenvolve esta atividade melhor do que um ser humano. Pois, dependendo da jornada de trabalho da pessoa, devido a fadiga, cansaço e outras interferências, o nível de atenção da pessoa diminui contribuindo para erros na obtenção da placa do veículo. Neste contexto, a relevância computacional do projeto se traduz pela complexidade de simular a visão humana em ambientes dinâmicos.

As técnicas utilizadas se mostraram eficientes, em especial a utilização da máscara de Sobel para obtenção do gradiente da imagem. Este filtro apresentou estabilidade na detecção de bordas na imagem em diversos ambientes, e a morfologia matemática mostrou-se uma ferramenta eficiente para a identificação da placa do veículo. Devido a eficiência da localização da placa apresentada nos testes realizados, não foi necessário trabalhar com regiões candidatas para a obtenção da localização horizontal da placa do veículo.

A biblioteca utilizada para o reconhecimento dos caracteres MODI (MICROSOFT, 2008), mostrou-se eficiente para o reconhecimento de texto em documentos, visto que utiliza um dicionário de palavras para auxiliar o reconhecimento. Porém, devido a construção da biblioteca ser voltada para reconhecimento de texto e não de caracteres, o reconhecimento dos caracteres da placa do veículo não mostrou-se eficiente. A construção de uma biblioteca voltada para o reconhecimento dos caracteres de uma placa de veículo, além de possuir uma complexidade menor do que uma biblioteca de reconhecimento de texto, apresentará resultados melhores. Pois poderá utilizar, por exemplo, a heurística de que os caracteres a serem reconhecidos nas três primeiras posições são letras maiúsculas e que as quatro últimas são números. Dessa forma, restringindo o universo de pesquisa, diminui-se a complexidade de construção aumentando a eficiência no reconhecimento.

As ferramentas utilizadas na implementação mostraram-se eficientes, principalmente o *plugin* Jigloo (JIGLOO, 2004) para o Eclipse (ECLIPSE, 2008). Com ele, pode-se “desenhar” a interface *swing*, obtendo um código legível como saída e com uma grande variedade de componentes. A ferramenta para especificação JUDE (JUDE, 2008) atendeu todas as necessidades do projeto.

4.1 EXTENSÕES

Para versões futuras sugere-se que seja implementado um OCR próprio para a ferramenta, a fim de melhorar o reconhecimento dos caracteres através da aplicação do conhecimento da formação dos caracteres da placa do veículo que é três letras seguidas por quatro números. Como sugestão pode-se utilizar Stivanello (2004).

Analisar se é possível melhorar ou substituir o filtro de Gauss, onde nos testes realizados está consumindo quase 36% do tempo total de processamento da ferramenta.

Incluir a detecção do veículo na imagem, possibilitando que o reconhecimento da placa seja aplicado apenas na região do veículo, como sugestão pode-se utilizar Santos (2008). Dessa forma, a detecção de borda do cenário da imagem não é considerada, melhorando a precisão da localização da região horizontal da imagem.

Incluir a etapa de morfologia matemática para aprimorar a delimitação da região horizontal da placa do veículo, visto que a morfologia matemática através da operação de fechamento apresentou uma precisão considerável na detecção da região vertical da placa.

Obter a placa para mais de um veículo na imagem e estender o reconhecimento para vídeo. Dessa forma, a ferramenta poderia monitorar rodovias.

REFERÊNCIAS BIBLIOGRÁFICAS

ALECRIM, E. **Redes neurais artificiais**. [S.l.], 2004. Disponível em: <http://www.infowester.com/redesneurais.php >. Acesso em: 04 set. 2008.

CAMAPUM, J. F. **Mini-curso de processamento de imagens**. [Brasília], 2005. Disponível em: <http://www.ene.unb.br/~juliana/cursos/semana/index.html>. Acesso em: 03 set. 2008.

CARMEN FREEFLOW. **Plate recognition software technical specifications**. Hungary, [2008]. Disponível em: <http://www.arhungary.hu/contleft/1012/content.htm> Acesso em: 03 ago. 2008.

CONSELHO NACIONAL DE TRÂNSITO. Resolução n. 231, de 15 de março de 2007. Padroniza o formato das placas dos veículos e da providências correlatos. [S.l.], 2007. Disponível em: <http://www.denatran.gov.br/download/Resolucoes/RESOLUCAO_231.pdf>. Acesso em: 26 ago. 2008.

COSTA, L. F.; CESAR JR., R. M. **Shape analysis and classification**: theory and practice. Boca Raton: CRC Press, 2001.

ECLIPSE. [S.l.], 2008. Disponível em: <http://www.eclipse.org>. Acesso em: 20 nov. 2008.

GOMES, J.; VELHO, L. **Fundamentos da computação gráfica**. Rio de Janeiro: IMPA, 2003.

GONZALEZ, Rafael C.; WOODS, Richard E. **Digital image processing.** 3. ed. New Jersey: Person Prentice Hall, 2008.

GONZALEZ, Rafael C.; WOODS, Richard E. **Processamento de imagens digitais.** Tradução de Roberto Marcondes Cesar Junior, Luciano da Fontoura Costa. São Paulo: Edgard Blücher, 2000.

HUTTON, J.; DOWLING, B. **Electronics and computer science**. Southampton, 2005. Disponível em: < http://users.ecs.soton.ac.uk/msn/book/new_demo/>. Acesso em: 21 set. 2008.

IMAGEJ. [S.l.], 2008. Disponível em: <http://rsb.info.nih.gov/ij>. Acesso em: 22 maio 2008.

JAVAOCR TEAM. **JavaOCR 1.3**. [S.l.], 2008. Disponível em: <http://www.javaocr.com>. Acesso em: 11 jan. 2008.

JIGLOO. [S.l.], 2004. Disponível em: <http://www.cloudgarden.com/jigloo>. Acesso em: 22 nov. 2008.

JUDE. Tokyo, 2008. Disponível em: <http://jude.change-vision.com/jude-web/index.html>. Acesso em: 28 março 2009.

KUBIÇA, S.; FACON, J.; LETHELIER, E. **Processamento de imagens de documentos - Parte II**. [S.l.], [2008]. Disponível em: <http://www.pr.gov.br/batebyte/edicoes/1999/bb89/imagens.htm>. Acesso em: 02 set. 2008.

LOESCH, C.; SARI, S. T**. Redes neurais artificiais:** fundamentos e modelos. Blumenau: EdiFURB, 1996.

MICROSOFT. [S.l.], 2008. Disponível em: <http://msdn.microsoft.com/en-us/library/aa167607(office.11).aspx>. Acesso em: 10 nov. 2008.

NAKASHIMA, E. K. **Sistema de reconhecimento automático de placas de veículos**. 2004. 19 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação, Universidade de São Paulo, São Paulo. Disponível em: <http://www.linux.ime.usp.br/~cef/mac499-04/monografias/elcio/monografia.pdf>. Acesso em: 01 ago. 2008.

PERA, M. Q. **Fenabrave divulga números do mercado brasileiro de junho**. [S.l.], 2007. Disponível em: <http://www.autodiario.com.br/index.php?m=543&sc=6>. Acesso em: 02 set. 2008.

SANTOS, D. **Sistema óptico para identificação de veículos em estradas**. 2008. 65 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://www.bc.furb.br/docs/MO/2009/335632_1_1.pdf>. Acesso em: 05 maio 2009.

STIVANELLO, Maurício E. **Inspeção industrial através de visão computacional.** 2004. 77 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

TOPSOFT. **TopOCR for Windows**. [S.l.], 2008. Disponível em: <http://www.topocr.com/download.html>. Acesso em: 12 jan. 2008.

UML. **Unified modeling language specification**: version 1.4. [S.l], 2002. Disponível em: <http://www.omg.org>. Acesso em: 28 março. 2009.

XU, Z.; ZHU, H. An efficient method of locating vehicle license plate. In: NATURAL COMPUTATION, 3th, 2007, Haikou. **Proceedings...** Hainan: IEEE Computer Society, 2007. p. 180-183. Disponível em: <http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/4304497/4304498/04304542.pdf >. Acesso em: 27 ago. 2008.

YANG, F.; MA, Z. Vehicle license plate location based on histogramming and mathematical morphology. In: AUTOMATIC IDENTIFICATION ADVANCED TECHNOLOGIES, 4th, 2005, New York. **Proceedings...** Los Alamitos: IEEE Computer Society, 2005. p. 89-94. Disponível em: <http://ieeexplore.ieee.org/iel5/10364/32967/01544406.pdf>. Acesso em: 25 ago. 2008.