CENTRO UNIVERSITÁRIO DE BRASÍLIA - UniCEUB

CURSO DE ENGENHARIA DE COMPUTAÇÃO

JEAN DIAS DO NASCIMENTO

DETECÇÃO E RECONHECIMENTO DE PLACA AUTOMOTIVA COM BAIXO CUSTO

Orientador: Prof. Ms. C. Francisco Javier De Obaldía Díaz

Brasília

dezembro, 2012

JEAN DIAS DO NASCIMENTO

DETECÇÃO E RECONHECIMENTO DE PLACA AUTOMOTIVA COM BAIXO CUSTO

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação. Orientador: **Ms. C. Francisco Javier De Obaldía Díaz**.

Brasília

dezembro, 2012

JEAN DIAS DO NASCIMENTO

DETECÇÃO E RECONHECIMENTO DE PLACA AUTOMOTIVA COM BAIXO CUSTO

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação. Orientador: **Ms. C. Francisco Javier De Obaldía Díaz**.

Este trabalho foi julgado adequado para a obtenção do Título de Engenheiro de Computação, e aprovado em sua forma final pela Faculdade de Tecnologia e Ciências Sociais Aplicadas - FATECS.

_____

Prof. Abiezer Amarilia Fernandes

Coordenador do Curso

Banca Examinadora:

_____	_____
Prof. Francisco Javier De Obaldía Díaz, Ms.	Prof. Flávio Antônio Klein, Ms.
Orientador	UnB
_____	_____
Prof. Fabiano Mariath D'Oliveira, Ms.	Prof. João Marcos Souza Costa, Esp.
UniCEUB	UniCEUB

Diz no livro de Êxodo, 35:35 – "Encheu-os de sabedoria do coração, para fazer toda a obra de mestre, até a mais engenhosa, e a do gravador, em azul, e em púrpura, em carmesim, e em linho fino, e do tecelão; fazendo toda a obra, e criando invenções."

Este trabalho só tem significado memorável quando existem pessoas que repassam sabedoria e dão companheirismo a cada dia. Dedico esta obra:

Aos meus pais, Jorge Jean e Alda Celeste, pela orientação do caminho a percorrer com amor e fraternidade. A minha amada esposa, Marcela Zago, pela comunhão de nossa família. A minha irmã, Ana Cláudia, que sempre renova o significado de harmonia e amizade. A Deus, pelo qual tudo é possível e acontece.

AGRADECIMENTOS

Pelo apoio e orientação, professores Francisco Javier e Abiezer Fernandes que colaboraram de forma relevante para que o trabalho fosse realizado.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

A/D – Analógico/Digital

ALGOL – ALGOrithmic Language

API – Application Programming Interface

ASCII – American Standard Code for Information Interchange

BCPL – Basic Combined Programming Language

CLU – CLUsters Language

CNSeg – Confederação Nacional das Empresas de Seguros Gerais, Previdência Privada e Vida, Saúde Suplementar e Capitalização

CONTRAN – Conselho Nacional de Transito

FISCA – Sistema de Fiscalização Inteligente

HSL – Hue Saturation Lightness

IP – Interface Protocols

LCD – Liquid Crystal Display

LED – Light Emitting Diode Displays

ML – Metalanguage

MSVS – Microsoft Visual Studio

OCR – Optical Character Recognition

RAM – Random Access Memory

RGB – Red-Green-Blue

SURF – Speedef-Up Robust Features

SVN – Subversion

TCP – Transmission Control Protocol

UF – Unidade Federativa

UTF-8 – 8-bit Unicode Transformation Format

VGA – Video Graphics Array

RESUMO

A proposta desse trabalho é apresentar uma solução de visão computacional para detecção e reconhecimento de placa automotiva utilizando câmeras de baixo custo e bibliotecas *open source* de tratamento de imagem OpenCV e de reconhecimento de caracteres Tesseract OCR. Sendo assim, é criado um modelo seguindo os conceitos da visão computacional abordado em seis etapas, todas comentadas em nível de instalação e implementação, sejam elas: a captura, o pré-processamento, a localização, a validação, a segmentação e por final transcrevendo a imagem da placa do veículo em caracteres. Após o desenvolvimento das etapas, são realizadas três aferições: do nível de exatidão na detecção da placa do veiculo, do desempenho do algoritmo e da exatidão na conversão dos caracteres da imagem para o formato texto. Neste trabalho são apresentadas todas as dificuldades que foram encontradas no transcorrer do projeto, desde a fase de concepção aos resultados obtidos, colecionando assim, vários materiais técnicos para se trabalhar com a visão computacional e proporcionando um exemplo de aplicação que ajudará entender melhor cada etapa.

Palavras Chave: visão computacional, OpenCV, Tesseract OCR, placa automotiva, detecção, reconhecimento.

ABSTRACT

The purpose of this study is to present a computer vision solution for detection and recognition of automotive plate using inexpensive cameras and opensource libraries like OpenCV processing image and Tesseract OCR character recognition. Thus, a model is created following the concepts of computer vision approached in six steps, all commented on the level of installation and implementation, they are: the capture, preprocessing, location, validation, targeting and end transcribing the image of the characters on board the vehicle. After the development of the steps are performed three measurements: the level of accuracy in detecting plate of the vehicle, the algorithm's performance and accuracy in converting the characters from the image into the text format. This work presents all the difficulties that were encountered in the course of the project, from the design stage to the results obtained, collecting so many technical materials to work with computer vision and providing an example application that will help better understand each step.

Keywords: computer vision, OpenCV, Tesseract OCR, automotive plate, detection, recognition.

1. INTRODUÇÃO

1.1. Apresentação do Problema

Os seres humanos podem reconhecer, sem muito esforço, objetos em movimento utilizando o sentido da visão. E o avanço da tecnologia, em capacidade de processamento, memória e sensoriamento, além do conjunto de bibliotecas de softwares, envolvendo melhores práticas e técnicas, vem reconstruindo a possibilidade dessa capacidade humana. Porém, a tarefa de reconhecimento de um objeto específico pelo computador em uma imagem, é um dos tópicos mais complexos no campo da programação visual. O reconhecimento de uma placa veicular é um pequeno desafio nessa área, devido aos diferentes tipos, formas e cores. Também, dos diversos meios de exposição luminosa, ângulo e distância durante aquisição da imagem. Além disso, para esse tipo de aplicação, é fundamental utilizar uma codificação que satisfaça um rápido processamento para reconhecimento dos padrões, típico de sistemas que operam em tempo real.

Diante do exposto questiona-se: como é possível localizar e determinar padrões de objetos através de uma câmera? Como por exemplo: localizar uma placa veicular em uma rodovia.

1.2. Objetivo do Trabalho

1.2.1. Objetivo geral

A proposta desse trabalho é desenvolver um software que identifique a placa veicular por vídeo imagem utilizando uma câmeras de baixo custo, como por exemplo, de um celular, adotando melhores técnicas e estudos acadêmicos em filtros, detecção de padrões e bibliotecas de software, utilizando como base as interfaces de programação de aplicativos (API) OpenCV e Tessaract OCR, na linguagem de programação C/C++.

1.2.2. Objetivo específico

Para alcançar o objetivo geral do projeto, foram traçados alguns objetivos específicos que primeiramente precisavam ser atendidos:

- Identificar as melhores práticas para a programação em C/C++;
- Conhecer o funcionamento de sistema de visão: sistema ótico, sistema de iluminação, sistema de aquisição de imagens, sistema de processamento de dados, propriedades das imagens, pré-processamento, técnicas de segmentação, filtros e reconhecimento de padrões;
- Integrar as bibliotecas de tratamento de imagem OpenCV e de reconhecimento de caracteres Tesseract OCR, aplicando-as.

1.3. Justificativa e Importância do Trabalho

Essa necessidade de reconhecer uma placa veicular vem ganhando espaço comercial, onde cada vez mais, o sistema de transporte e segurança vem incrementando benefícios à vida dos cidadãos, uma vez que pode ser utilizado em inúmeras aplicações, como em controle de acesso, monitoramento e segurança. Exemplos: utilizando em acessos aos estacionamentos, controle de tráfego em rodovias, em pedágios, em detecção e controle de veículos pelo departamento de trânsito etc.

1.4. Escopo do Trabalho

Sendo assim, este trabalho dará ênfase somente à implementação do aplicativo: detecção e leitura da placa veicular para carros e caminhões - respeitando as possibilidades impostas pelas configurações mínimas do meio físico (distâncias, ângulos, condições climáticas e iluminação) e dispositivos eletrônicos (câmera e microcomputador), esclarecidas ao longo desse trabalho. A figura 1.1 a seguir representa a proposta global:

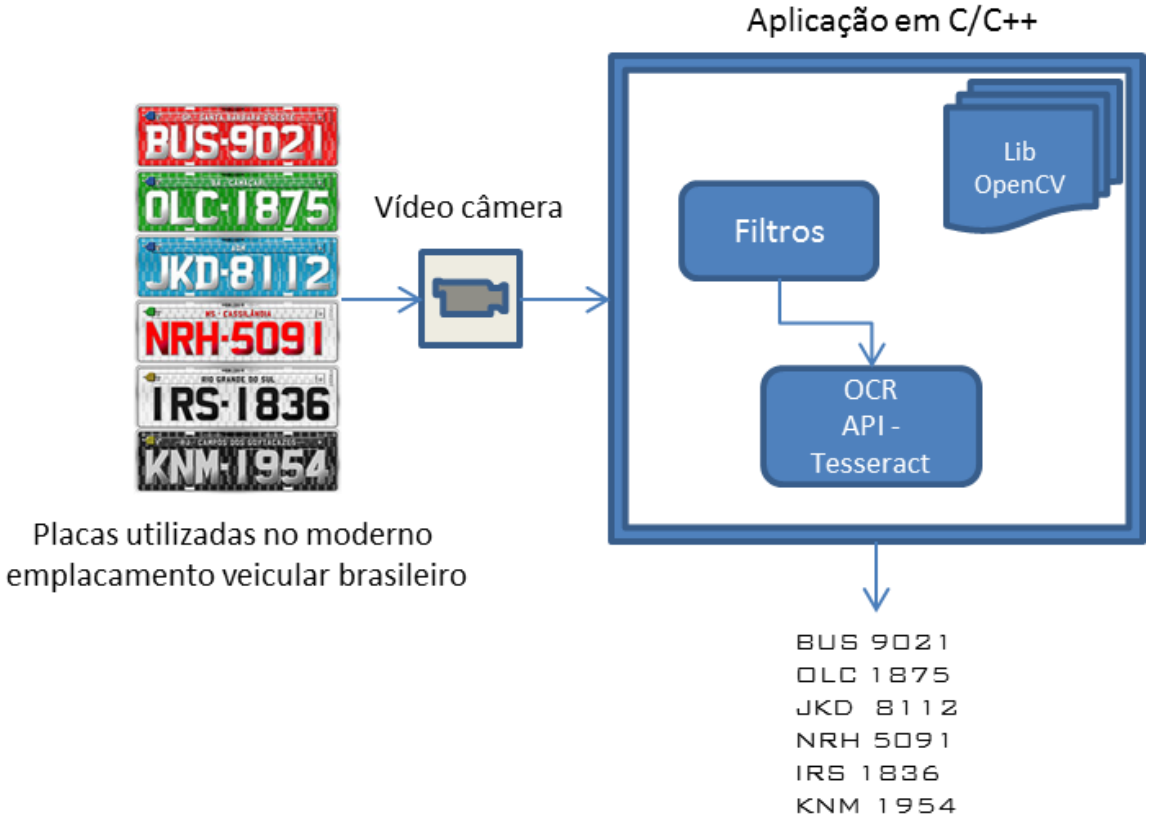

Figura 1.- Arquitetura do projeto

Fonte: Autor.

1.5. Resultados Esperados

O projeto final tem por finalidade:

- A entrada de um vídeo em tempo real ou gravado;
- A detecção da placa do veículo;
- A leitura da placa detectada para o formato texto.

Espera-se também com esse trabalho dar profundidade ao assunto, aplicando técnicas cada vez mais elaboradas para detecção de objetos e padrões, com possibilidade de futura comercialização.

1.6. Estrutura do Trabalho

Além deste capítulo introdutório, esse trabalho está estruturado em mais cinco capítulos, organizados da seguinte forma:

- Capítulo 2 – Apresentação do Problema: Nesse capítulo é apresentada uma descrição do problema que o projeto pretende resolver. Primeiramente serão mostrados alguns dados atuais e como o problema vem sendo resolvido por alguns profissionais da área. Em seguida são mostrados os benefícios da solução apresentada.

- Capítulo 3 – Bases Metodológicas para a Resolução do Problema: Aqui a monografia apresenta todo o referencial teórico e tecnológico que embasa o projeto. Além de aplicar conteúdos vistos nas disciplinas para a resolução do problema proposto.

- Capítulo 4 – Modelo Proposto: Esse capítulo detalha todas as etapas e passos necessários para a resolução do problema apresentado no capítulo 2.

- Capítulo 5 – Aplicação do Modelo Proposto: É apresentada nesse capítulo uma aplicação prática, envolvendo um caso real de aplicação, mostrando a viabilidade da proposta de resolução sugerida. Aqui são demonstrados os testes, a simulação e a avaliação do projeto.

- Capítulo 6 – Conclusão: Com este termina-se a documentação do trabalho, permitindo uma análise completa de todo o desenvolvimento do projeto e da monografia. Em seguida são apresentadas sugestões para trabalhos futuros.

2. APRESENTAÇÃO DO PROBLEMA

2.1. O Problema

Para muitos especialistas, as estatísticas oficiais no Brasil não relatam a real situação do trânsito no país. A demora em compilar os dados obtidos e a forma como são notificados, até mesmo pelos estados e municípios, comprometem o resultado final das estatísticas. Por exemplo, sabe-se que atualmente existe um aumento desenfreado da frota de veículos e que a malha viária das grandes cidades já não suporta essa capacidade de crescimento, porém não se sabe muito bem como está sendo distribuído o fluxo de veículos que trafegam naquele determinado instante ou período em uma via central de uma capital, inviabilizando a tomada de decisões instantâneas para controlar o trafego. Ou, não tem informações mais elaboradas da quantidade de carros que trafegam em uma via sem o devido licenciamento do veículo.

Uma situação que muito vem se discutindo também é sobre o controle da segurança pública. Estatísticas do mês de junho de 2012, elaborado pela CNseg (Confederação Nacional das Empresas de Seguros Gerais, Previdência Privada e Vida, Saúde Suplementar e Capitalização), revelou que 19.987 veículos foram furtados no Brasil. Esses elevados números refletem a ineficiência dos sistemas de segurança.

Portanto, na tendência natural, é previsto que no decorrer do tempo estes números aumentem ainda mais, tornando o monitoramento e a identificação dos veículos um problema complexo, trabalhoso ou mesmo ineficiente.

Sendo assim, o intuito deste projeto é desenvolver um aplicativo que ajude a realizar o reconhecimento da placa do veículo, auxiliando, assim, trabalhos futuros que consigam solucionar os problemas que envolvem a fiscalização urbana, a gestão do trânsito e a segurança pública.

2.2. Soluções Existentes

2.2.1. Radar e lombada eletrônica

Atualmente, a forma mais comum de monitoramento e fiscalização dos veículos são os radares fixos ou móveis que existem nas ruas das grandes cidades.

Existem quatro tipos de radares que multam carros, o radar fixo é um deles. Seu funcionamento é bastante simples e acontece sem qualquer intervenção humana. São três laços detectores em cada faixa de uma via. Quando um veículo passa, os laços calculam sua velocidade. Se o motorista estiver acima da velocidade permitida no local, a câmera fotográfica é acionada. Para não ofuscar o motorista durante a noite, alguns radares utilizam flash infravermelho. O sistema é independente em cada faixa, ou seja, se dois carros passarem ao mesmo tempo um do lado do outro em um radar, ele consegue diferenciar a velocidade.

Os outros três tipos de radares também funcionam de modo parecido, mas com algumas diferenças. No primeiro, há totens na calçada que mostram a velocidade do veículo para o motorista utilizando um laço detector que calcula a velocidade do carro. O segundo é o radar estático com tripé – popularmente conhecido como o "radar móvel" – dois feixes de laser perpendiculares à pista calculam a velocidade dos carros que passam. Esse sistema é incapaz de diferenciar dois carros que passem juntos por um feixe. Assim, quando isso acontece, o radar anula a medição, mesmo que um dos veículos esteja em alta velocidade. O último tipo de radar é o único realmente móvel e também o único que é, de fato, um radar, pois usa o efeito doppler para cálculo da velocidade. Apontado para um veículo, ele lança uma onda eletromagnética que, após atingir o veículo e voltar, mostra a velocidade em que ele estava.

2.2.2. FISCA – Sistema de Fiscalização Inteligente

A empresa Search Tecnologia desenvolveu um produto chamado FISCA visando fiscalizar os veículos cadastrados na base local e de outras UF's do Departamento de Transito (DETRAN), propondo a detecção instantânea das irregularidades dos veículos que trafegam.

O sistema captura a placa dos veículos que cruzam pela barreira (BLITZ), por meio de câmara. O procedimento pode capturar as placas dianteiras ou traseiras dos veículos.

O sistema transmite a placa em forma de imagem para uma estação de reconhecimento de caracteres onde é convertida para arquivo texto e encaminha para o sistema central de processamento que por sua vez, verifica as informações de débitos, informações de bloqueios, informações de roubo/furto, informações de seguro obrigatório e ainda os dados do condutor se este for o proprietário do veiculo.

A figura 2.1 a seguir, exemplifica o funcionamento.

Figura 2.- FISCA - Bliz Inteligente.

Fonte: extraído da internet no endereço, em 13/10/12: http://detran.ap.gov.br/produtos/fisca/fisca.jsp

2.3. Benefícios da Solução Apresentada

A ideia de trazer a inteligência na fiscalização com a prévia triagem dos veículos, identificando possíveis usuários com problemas e prestando serviços para corrigirem suas pendências com os órgãos arrecadadores, ou na possibilidade de identificação e localização imediata de veículo com registro de roubo e bloqueios judiciais para apreensão do veículo,

são alguns dos benefícios para o sistema. Essa possibilidade aumentaria a eficiência das autoridades de fiscalização, acabando com a seleção aleatória de veículos e prestando um serviço de segurança eficiente.

3. BASES METODOLÓGICAS PARA RESOLUÇÃO DO PROBLEMA

Nas próximas seções serão apresentados conceitos importantes para o desenvolvimento do projeto.

3.1. O que é Visão Computacional?

Visão computacional é a transformação de dados de uma câmera fotográfica ou de vídeo em qualquer uma decisão ou em uma nova representação, segundo os autores Trucco (1998) e Forsyth (2003). Todas essas transformações são feitas para alcançar algum objetivo particular. Os dados de entrada podem incluir uma informação de contexto como “tem um carro nesse local” ou “o laser localizador indica um objeto está a 1 metro de distância". A decisão pode ser “o carro está em movimento" ou “existem 14 células tumorais neste slide”. A nova representação pode transformar uma imagem colorida em escala de cinza ou remoção de uma cena a partir de uma sequência de imagens.

Como criaturas visuais, é fácil o ser humano se enganar pensando que o computador realizará tarefas de forma simples. Quão difícil é encontrar, por exemplo, um carro quando alguém está olhando para ele em uma imagem? O cérebro humano divide a visão em muitos canais diferentes, onde exprimem uma grande quantidade de informações em seu cérebro. O cérebro humano também tem um sistema de detecção que cruza informações ao ver uma parte de um objeto, utilizando uma espécie de realimentação que cruza informações associadas ao longo da experiência vivida.

Num sistema de visão artificial, no entanto, um computador recebe grandes quantidades de números a partir de uma câmera ou de um meio de armazenamento de informação. Que, para a maior parte, não há reconhecimento do padrão interno ou nenhum controle automático de foco e abertura. Também não há associações cruzadas com anos de experiência. Assim, por inferência ao exposto, conclui-se que a área da visão computacional é pouco desenvolvida, exigindo assim, vários estudos complexos de física, matemática e computação.

A Figura 3.1 mostra a imagem de um automóvel. Nesse quadro, existe um retrovisor do lado do motorista. No entanto, o que o computador "vê" é apenas uma grade de números.

Estes números trazem poucas informações. Existe assim, a difícil missão de transformar esta grande quantidade de números em sentido, o que faz o estudo da visão computacional uma tarefa difícil.

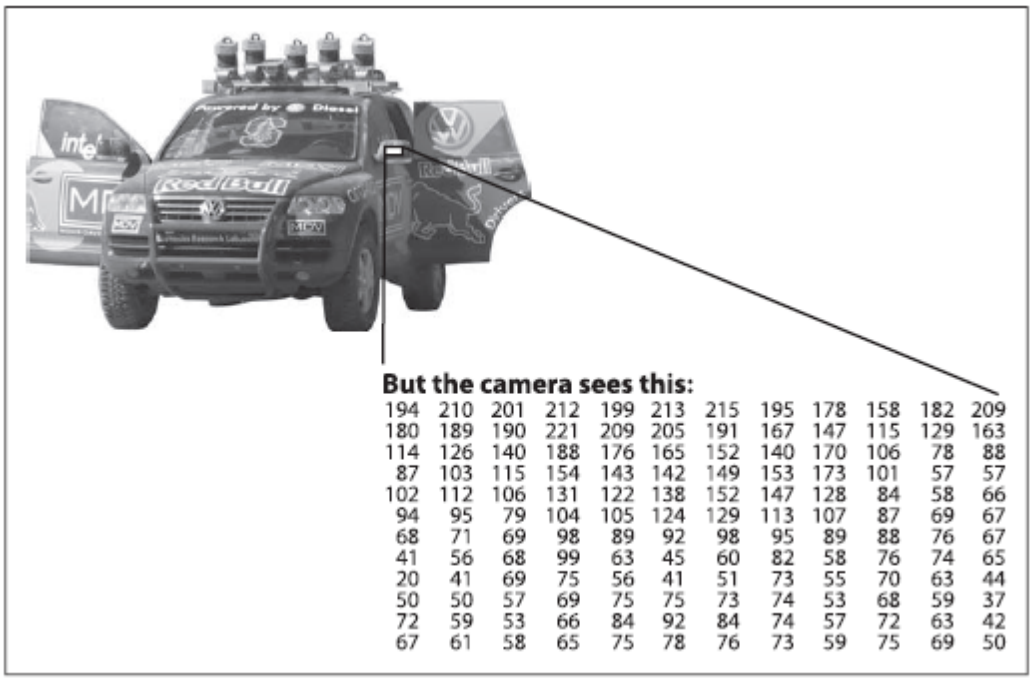

Figura 3.- para o computador o retrovisor de um carro é apena uma grade de números

Fonte: Bradski e Kaehler (2008).

Outro motivo importante é que os dados são corrompidos por ruído e distorção. Tal fato decorre de variações no mundo (clima, iluminação, reflexos, movimentos), imperfeições na lente e configuração mecânica, o tempo de integração com o sensor (borrão na imagem), o ruído elétrico no sensor ou outros aparelhos eletrônicos e artefatos de compressão após captura de imagem. Dado os desafios, como obter algum progresso?

3.2. Enfrentando os Desafios

Para o autor E. Roy Davies (2005), a visão computacional depende de vários métodos para ter uma margem de sucesso considerável. Desde a aquisição da imagem à tomada de decisão. A seguir, algumas resumidamente:

- **Aquisição de imagem**: atualmente existem de vários modelos de sensores de imagem, como por exemplo: as câmeras fotográficas, dispositivos de infravermelho e ultravioleta, sensores gamas etc. Que tem por finalidade extrair um conjunto de informações bidimensional ou tridimensional.
- **Pré-processamento**: antes que algum método de visão computacional possa ser aplicado, a fim de extrair alguma informação específica, geralmente é necessário processar os dados de modo a assegurar que a mesma satisfaça certas hipóteses impostas pelo método. Exemplos são: re-amostragem, a fim de assegurar que o sistema de coordenadas está correta. A redução do ruído, a fim de assegurar que o ruído do sensor não introduza informações falsas. Aumento de contraste, para assegurar que a informação relevante seja detectada.
- **Segmentação**: um ponto importante para detecção de um objeto é conhecer as regiões da imagem, pontuando-as por níveis de interesse ou relevância.
- **Alto nível de processamento**: nessa etapa, tipicamente utiliza-se um pequeno conjunto de informações, assumindo que contenha o objeto específico de pesquisa. Como por exemplo: verifica-se se os dados satisfazem o modelo básico ou um pressuposto específico das informações. Assim como os parâmetros de observação, como o tamanho do objeto ou a posição onde este se encontra. O reconhecimento da imagem, classificando-a como um objeto. E o registro da informação, comparando ou combinando duas visões de um mesmo cenário.
- **Tomada de decisão**: nesse nível, o sistema toma a decisão final. Como por exemplo: encontrou ou não o objeto de estudo. Se, passa ou descarta o produto, pela inspeção automática de qualidade. Sinaliza o cenário, indicando se existem as características informadas, como a detecção da placa do automóvel ou localização de uma determinada pessoa em um local.

3.3. Componentes de um Sistema de Visão Computacional

Sistemas de visão integram, em uma única solução, uma série de tecnologias diferentes, permitindo a grande flexibilidade no desenvolvimento de aplicações em diversas áreas do conhecimento.

A organização destes sistemas em diferentes partes/componentes é conveniente, pois permite o estudo de cada tecnologia empregada no sistema em separado.

Gonzalez and Woods (2002) apresentam uma estrutura de componentes interligados para a organização de sistemas de processamento de imagens como pode ser visto na figura 3.2.

Segue uma breve descrição de cada componente do modelo:

- **Sensores**: dispositivo físico sensível à energia irradiada pelo objeto do qual se pretende adquirir uma imagem, convertendo-a num sinal elétrico proporcional à intensidade da energia recebida.
- **Hardware de Aquisição**: dispositivo responsável pela digitalização do sinal proveniente do sensor e por realizar algum pré-processamento sobre estes dados antes de enviá-los ao computador.
- **Computador**: é o elemento principal do sistema, que coordena todas as atividades desempenhadas, desde o acionamento do hardware de aquisição da imagem, as tarefas do software de processamento de imagem, o armazenamento dos dados, cópia backup em mídias, até a visualização dos resultados em um display. De acordo com o tipo de aplicação, o computador pode ser um PC comum, ou um super computador, ou até mesmo uma unidade de processamento dedicada ao controle das tarefas do sistema, podendo estar embutido totalmente junto ao sensor e ao hardware de aquisição das imagens (câmeras inteligentes).
- **Software de Processamento**: consiste de uma cadeia de algoritmos de processamento de imagem ordenada de forma a solucionar o problema em questão, retornando dados importantes para o computador, o qual deve tomar as decisões em relação aos resultados apresentados.
- **Armazenamento**: espaço de memória dedicada ao armazenamento das imagens e dados adquiridos pelo sistema para possibilitar seu processamento de forma rápida e otimizada. Compreende o espaço de memória RAM no computador, memória no próprio hardware de aquisição ou até mesmo as demais mídias de acesso mais lento do computador.

- **Displays**: dispositivos responsáveis por apresentar um resultado gráfico do estado de evolução do sistema ao usuário. Pode ser um monitor, LCDs, telas de plasma, dentre outros.
- **Meios de cópia**: dispositivos que fazem a cópia física segura da informação adquirida e processada pelo sistema. Podem ser impressoras, mídias óticas, cartões de memória, dentre outros.
- **Rede**: a rede é um componente essencial do sistema principalmente quando os resultados do sistema têm de ser enviados a uma estação remota distante do local de operação do sistema.

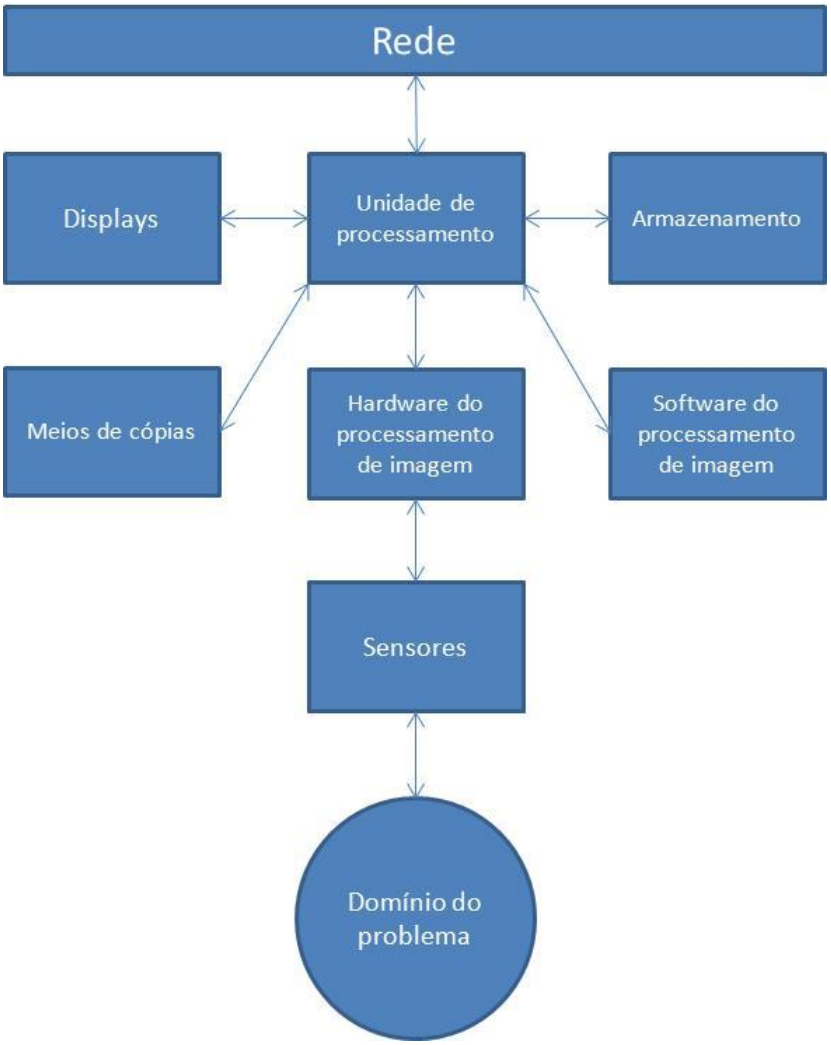

Figura 3.- Componentes gerais de um sistema de processamento de imagens.

Fonte: Gonzalez and Woods (2002).

Já para a organização modular de um sistema de visão completo, segundo o autor Jähne (1999) apresenta uma estrutura de componentes visando separar metodicamente as diferentes tecnologias contidas no sistema.

- **Fonte de Radiação**: iluminação adequada para o realce das características do objeto em estudo é necessária, tendo em vista que a maioria dos objetos não emitem luz própria, sendo esta importante para sensibilizar o sensor de aquisição da imagem.
- **Câmera**: direciona e coleta a radiação proveniente do objeto em estudo para o sensor de aquisição.
- **Sensor**: converte o sinal de radiação recebido em outro compreensível à unidade de processamento.
- **Unidade de Processamento**: coleta, organiza e processa os sinais recebidos do sensor, extraindo características e tomando decisões a partir destas informações.
- **Atores**: reagem ao resultado das observações realizadas pela unidade de processamento.

3.4. Sistema Ótico

No desenvolvimento de uma aplicação de visão, deve-se tomar muito cuidado na escolha de cada componente do sistema. Um princípio básico para que se possa escolher bons algoritmos de processamento de imagens e alcançar bons resultados na interpretação das informações, é obter como entrada do sistema uma imagem de boa qualidade (Davies, 2005); (Jain, Kasturi e Schunck, 1995). Ou seja, a etapa de aquisição da imagem deve ser bem configurada, o que implica a escolha correta e precisa do hardware do sistema de visão. Conforme explicado anteriormente, erros na configuração destes equipamentos, como por exemplo, obtenção de imagens escuras ou com sombras perturbadoras, foco inapropriado, magnificação insuficiente, tempo de aquisição impróprio e ruídos demasiados podem inviabilizar a implementação da aplicação.

Alguns parâmetros de um sistema de visão exercem normalmente maior influência sobre a configuração do sistema ótico. Estes parâmetros são ditos parâmetros fundamentais, sendo necessário familiarizar-se com os mesmos para de iniciar um projeto.

A figura 3.3 ilustra estes parâmetros.

Figura 3.- Parâmetros fundamentais para definicação de um sistema ótico.

Fonte: Edmund Optics em http://www.edmundoptics.com/

Segue uma descrição detalhada dos mesmos parâmetros segundo os autores Jain, Kasturi and Brian (1995):

- **Campo de Visão** (*Field of Vision – FOV*): representa a área visível do objeto em estudo que incide sobre o sensor, ou seja, a porção do objeto que preenche e sensibiliza a área do sensor.
- **Distância de Trabalho** (*Working Distance –WD*): representa a distância da parte frontal das lentes até a superfície do objeto. Trata-se normalmente de uma faixa de valores (máximo e mínimo).
- **Profundidade de Campo** (*Depth of Field – DOF*): representa a maior distância (em termos profundidade no campo de visão) que pode ser mantida em foco no objeto de estudo para uma determinada distância de trabalho. Também pode ser vista como a quantidade de movimento permitida ao objeto que ainda conserve foco na área ou superfície inspecionada.

- **Resolução** (*Resolution – R*): representa a menor porção do objeto em estudo que pode ser distinguida pelo sistema. É normalmente visualizada em pares de linha, ou em número de *pixels*, e também é bem conhecida pela expressão “resolução espacial”.
- **Tamanho do Sensor** (*Sensor Size – SS*): representa o tamanho da área ativa do sensor, especificada em sua dimensão horizontal.

Desta forma, o primeiro passo de um projeto de sistema de visão é identificar os parâmetros fundamentais na aplicação, que permitem inferir a respeito dos demais dispositivos necessários ao sistema. A figura 3.4 apresenta uma metodologia sistemática para o projeto de um sistema de visão do ponto de vista do hardware do sistema (Deschamps, 2004).

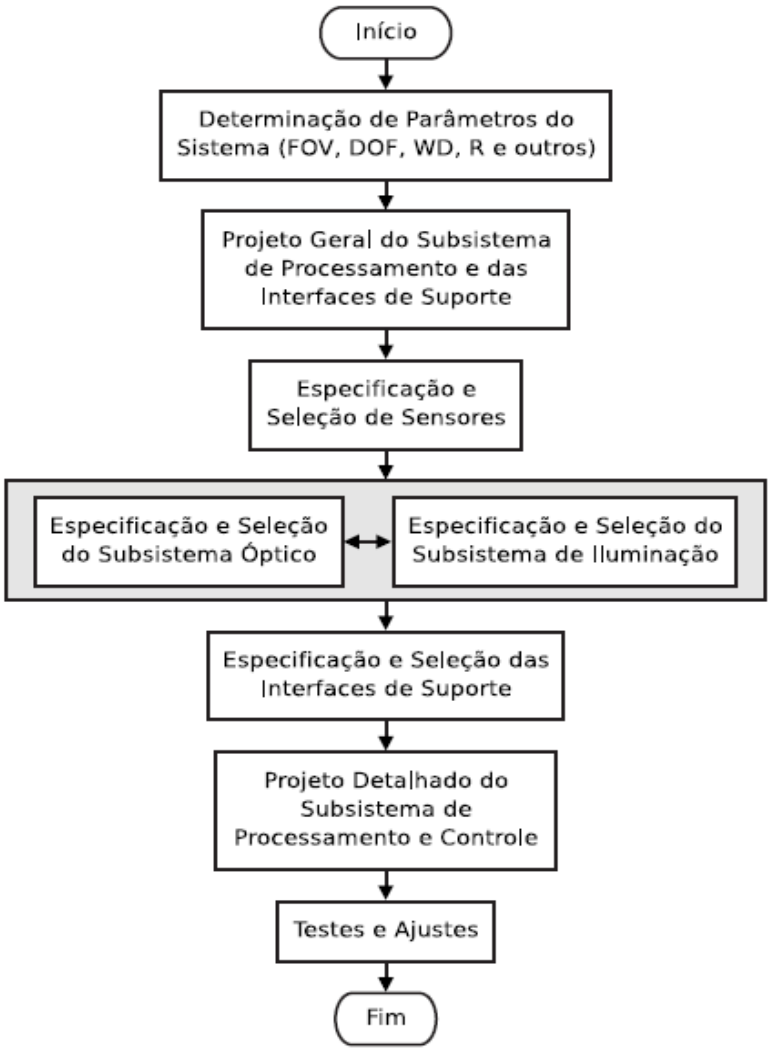

Figura 3. - Metodologia sistemática para o projeto de um sistema de visão computacional.

Fonte: Deschamps (2004).

Observa-se no diagrama que o projeto inicia sempre com a identificação dos parâmetros fundamentais para a boa formação das imagens (*FOV*, *DOF*, *WD*, *R*, *SS* e derivados). Na sequência, um pré-projeto do sistema de processamento pode ser inferido, de acordo com as restrições temporais da aplicação. Em geral nesta fase já foram adquiridas algumas imagens do objeto de estudo, e pode-se tentar inferir soluções preliminares para o processamento destas.

Um sensor para aquisição das imagens deve ser escolhido, levando-se em conta o tipo de varredura, taxa de aquisição e resolução necessárias para atacar o problema. O conjunto ótico e de iluminação devem ser ajustados para permitir correta magnificação, foco e destaque para o objeto em estudo na imagem. Em geral a escolha de ambos estes sistemas é feita de maneira paralela pela forte ligação de algumas características das lentes e a quantidade de luminosidade que afeta o sistema.

3.5. Sistema Ótico Propriamente Dito

O sistema ótico é formado pela composição de lentes, tubos extensores e de montagem, filtros, difusores, polarizadores, espelhos, prismas e fibras óticas, sendo responsável pelo direcionamento e dimensionamento dos feixes luminosos que provêm do objeto para o sensor ótico (Jähne, 1999).

As lentes definem a superfície de foco e a magnificação (ampliação ou redução) da imagem do objeto em estudo. Podem ser côncavas, convexas, plano-convexas, telecêntricas, dentre outras. Os tubos extensores (ou espaçadores) auxiliam na magnificação da imagem do objeto em estudo, porém, diminuem a quantidade de iluminação incidente sobre o sensor e limitam a distância de trabalho. Já os tubos de montagem servem para adaptar diferentes padrões de lentes e câmeras. Filtros extraem frequências específicas do espectro luminoso que incide no sensor, como por exemplo, filtros de cores visíveis ao olho humano, filtros infravermelhos e ultravioletas, dentre outros. Difusores são aplicados no espalhamento ou dispersão dos raios luminosos incidindo sobre os objetos, para dar um aspecto mais homogêneo de iluminação. Já os polarizadores filtram a intensidade dos feixes luminosos incidentes sobre o objeto em estudo. Espelhos são usados na manipulação da trajetória dos feixes luminosos incidentes na cena em estudo por reflexão, sendo muito usados para simular dupla aquisição de imagens, ou seja, aquisição feita por duas câmeras, porém, realizada com

apenas uma câmera, para economizar custos do sistema. Os prismas também possuem características reflexivas, porém, costumam ser usados para realizar a divisão dos feixes luminosos em mais de uma direção, ou até mesmo a separação de componentes da luz.

Abaixo alguns conceitos para entendimento do sistema ótico:

- **Abertura de lentes** (f /# ou *f-number*): representa o controle de abertura da íris das lentes, proporcionando uma medida da quantidade de luminosidade que sensibiliza o sensor ótico. À medida que a íris se fecha, *f* /# aumenta. Há uma relação íntima entre a abertura de lentes e a profundidade de campo do sistema, como pode ser visto na figura 3.5. Há um aumento na profundidade de campo do sistema à medida que a abertura da íris diminui. Em contrapartida, deve-se compensar a quantidade luminosa incidente no sensor aumentando a potência da fonte luminosa.

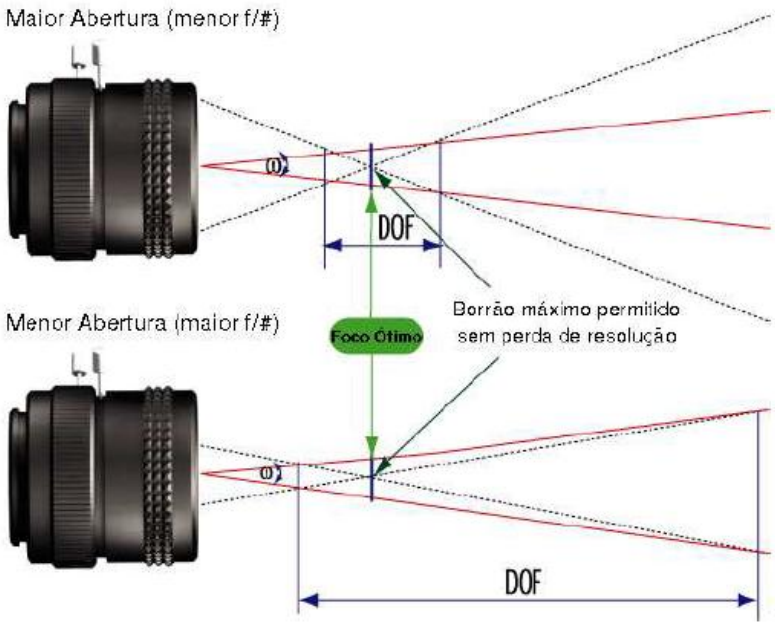

Figura 3.- Relacionamento entre a profundidade de campo e a abertura de lentes.

Fonte: Edmund Optics em http://www.edmundoptics.com/

- **Magnificações primária**: representam a relação de proporção entre as dimensões do objeto em estudo no mundo real e as imagens projetadas no sistema. Representa o quanto o objeto em estudo será aumentado (ou diminuído) de acordo com o tamanho da imagem projetada no sensor.

- **Limites focais da lente**: é influenciada diretamente pela distância de trabalho do sistema, ou seja, é necessário conhecer a distância do objeto em estudo em relação às lentes para poder determinar onde queremos manter foco para a aquisição das imagens. Os limites focais significam na verdade uma faixa de valores de distâncias em que o sistema alcança foco sobre o objeto. O termo limite focal também é conhecido como profundidade de foco na literatura de visão de Griot (1999).

- **Contraste**: é uma comparação entre as diversas tonalidades de cinza (intensidade luminosa) da imagem adquirida, que auxilia a identificar e separar objetos do fundo da imagem. É fortemente influenciado pelas características da lente, da câmera e principalmente da iluminação do sistema. O contraste é normalmente definido em termos de porcentagem. Por exemplo, uma linha preta desenhada sobre uma superfície branca resulta em 100% de contraste entre a linha e a superfície.

- **Distorções e erros de perspectiva**: fenômenos como o de distorção e os erros de perspectiva alteram a qualidade final da imagem. A distorção é um fenômeno (ou aberração de natureza geométrica) que ocorre com as lentes devido as suas características esféricas, produzindo diferenças de magnificação em pontos distintos da imagem, fazendo com que os objetos sejam dispostos incorretamente na imagem em relação ao seu centro. Ou seja, a distorção não causa perda de informações na imagem, mas sim um posicionamento incorreto de partes da imagem. A distorção (D%) pode ser corrigida por software calculando a porcentagem de erro de distorção ou deslocamento ocasionado pelas lentes, através dos valores de distância atuais (AD) e previstos (PD) nas imagens adquiridas. Já os erros de perspectiva (conhecidos por *parallax*) são fenômenos bem conhecidos da visão humana, sendo de fato, o que permite o cérebro humano interpretar as informações 3D do mundo real. Estes erros são mudanças na magnificação de um objeto, dando a impressão de que as regiões mais próximas da lente tenham dimensões maiores do que as que se encontram mais distantes.

Os conceitos apresentados nos tópicos de sistema ótico (3.4) e sistema ótico propriamente dito (3.5), não fazem parte do projeto físico deste trabalho e tem por finalidade, citar conceitos mais amplos sobre o assunto. O autor deste projeto utiliza um sistema ótico de baixo custo, a exemplo de vídeos câmeras de celulares (Iphone 4S) e *webcam* (resolução VGA) – cujo sistema ótico projetado foi otimizado à necessidade do próprio aparelho.

3.6. Sistema de Iluminação

O objetivo do sistema de iluminação é a projeção de luz sobre o objeto em estudo, pois em geral estes não emitem luz própria, que é necessária para a sensibilização do sensor ótico. Quando se menciona em luz, entende-se qualquer faixa do espectro luminoso, e não apenas a faixa do espectro visível ao olho humano. Existem muitas soluções de visão que requerem inclusive a aplicação de luz incidente em faixas não visíveis do espectro luminoso, como, por exemplo, as imagens de tomografia médica, algumas imagens astronômicas e imagens infravermelho de curvas de calor. A figura 3.6 ilustra toda a faixa do espectro luminoso. Repara-se como é pequena a faixa visível ao olho humano, e quão limitada seria a tecnologia de visão caso se restringisse apenas a esta estreita faixa do espectro.

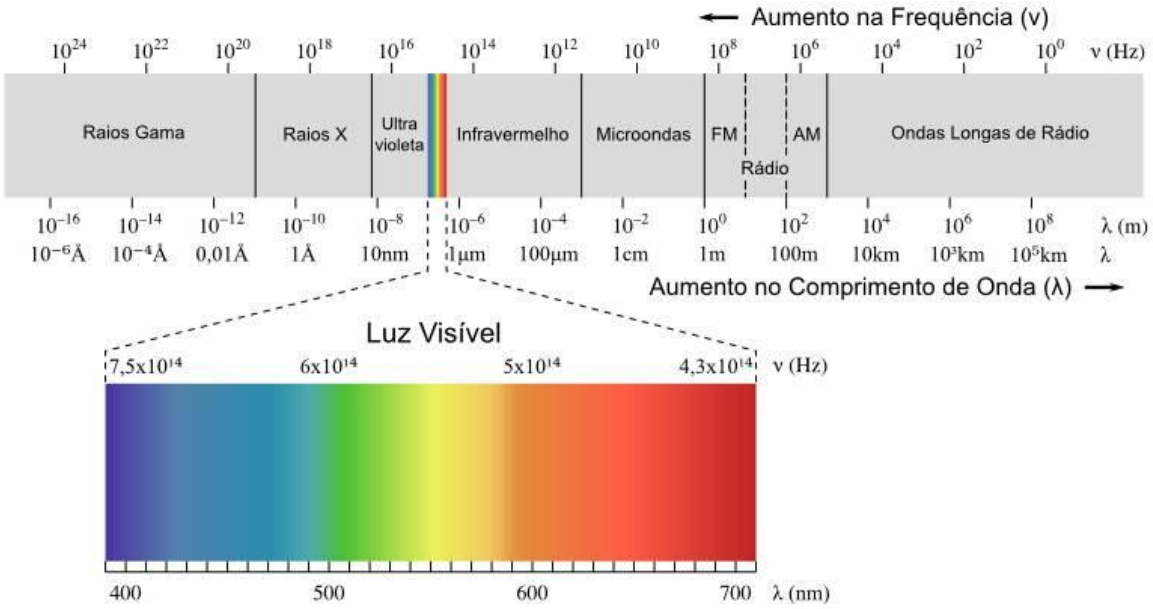

Figura 3. - faixas do espectro luminoso.

Fonte: extraído da internet no endereço, em 21/11/2012: http://dan-scientia.blogspot.com.br/2010/03/relacao-da-frequencia-com-o-comprimento.html

Sendo assim, a escolha de um tipo de iluminação correto para o ambiente da aplicação torna-se extremamente importante, pois se o objeto em estudo não for destacado (apresentar bom contraste) em relação às demais informações da cena, dificilmente consegue-se progredir na solução, e o projeto pode ser totalmente inviabilizado. Em contrapartida, caso uma boa

iluminação seja alcançada, as etapas de processamento das informações tornam-se muito mais fáceis conforme Erhardt-Ferron (2000).

Em geral, a escolha de uma fonte luminosa e de uma técnica de iluminação adequada são influenciadas pelas características superficiais do objeto em estudo (geometria, estrutura, cor, transparência, reflectância), onde o objetivo é normalmente incidir sobre a superfície do objeto uma iluminação homogênea e constante ao longo do tempo, conforme Erhardt-Ferron (2000) e Jähne (1999).

O projeto de iluminação de um sistema de visão consiste normalmente de três etapas, Deschamps (2004):

- Determinar o tamanho exato do campo de visão (junto ao projeto do sistema ótico), para que se compreenda o tamanho e as características superficiais da área que se deseja iluminar.
- Determinar o tipo de fonte luminosa adequada à aplicação (tungstênio, fluorescente, halogênio, LED, laser), escolhendo uma faixa de frequência apropriada do espectro luminoso conforme as características superficiais do objeto em estudo vistas no passo anterior.
- Determinar a geometria (posição da fonte em relação ao objeto e sensor, direção dos feixes luminosos), a potência, eficácia e características temporais da fonte luminosa, necessárias para realçar as partes desejáveis do objeto de forma homogênea e constante ao longo do tempo.

3.7. Fundamentos de Imagem Digital [1]

Este tópico tem por objetivo apresentar as principais características das imagens digitais.

Uma imagem monocromática pode ser descrita matematicamente por uma função *f(x,y)* da intensidade luminosa, sendo seu valor, em qualquer ponto de coordenadas espaciais

[1] MARQUES FILHO, Ogê; VIEIRA NETO, Hugo. *Processamento Digital de Imagens*, Rio de Janeiro: Brasport, p.19-20, 1999.

(x,y), proporcional ao brilho (ou nível de cinza) da imagem naquele ponto. A figura 3.7 mostra uma imagem monocromática e a convenção utilizada para o par de eixos *(x,y)*.

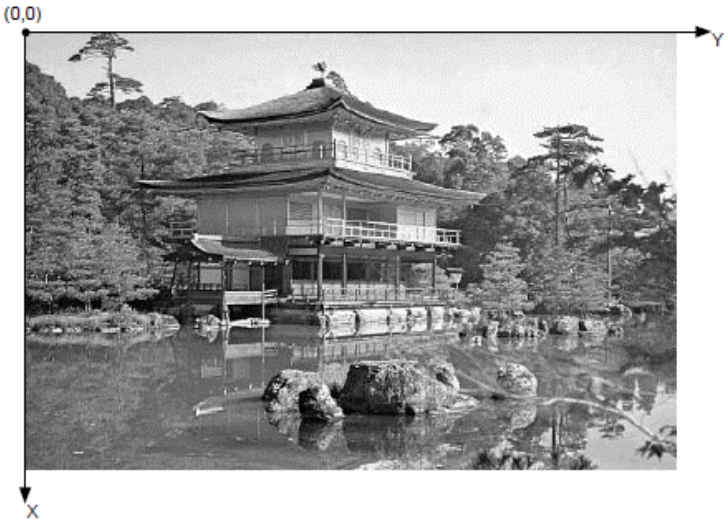

Figura 3. - Uma imagem monocromática e a convenção utilizada para o par de eixos (x,y).

Fonte: MARQUES FILHO, Ogê; VIEIRA NETO (2009).

A função *f(x,y)* representa o produto da interação entre a iluminância *i(x,y)* – que exprime a quantidade de luz que incide sobre o objeto – e as propriedades de refletância ou de transmitância próprias do objeto, que podem ser representadas pela função *r(x,y)*, cujo valor exprime a fração de luz incidente que o objeto vai transmitir ou refletir ao ponto *(x,y)*. Estes conceitos estão ilustrados na figura 3.8. Matematicamente:

$$f(x,y) = i(x,y) \cdot r(x,y) \qquad (3.1)$$

com:

$i(x,y) > 0$

$0 < r(x,y) < 1$

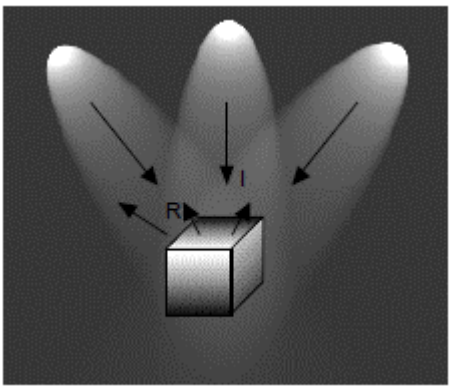

Figura 3. - Os componentes iluminância (I) e refletância (R) de uma imagem.

Fonte: MARQUES FILHO, Ogê; VIEIRA NETO (2009).

As tabelas 3.1 e 3.2 apresentam valores típicos de iluminância e refletância.

Tabela 3.- Exemplos de valores para i(x,y) [em lux ou lúmen/m2].

i(x,y)	
900	dia ensolarado
100	dia nublado
10	iluminação média de escritório
0,001	noite clara de lua cheia

Fonte: MARQUES FILHO, Ogê; VIEIRA NETO (2009).

Tabela 3.- Exemplos de valores para r(x,y).

r(x,y)	
0.93	neve
0.8	parede branco-fosca
0.65	aço inoxidável
0.01	veludo preto

Fonte: MARQUES FILHO, Ogê; VIEIRA NETO (2009).

No caso de uma imagem que possui informações em intervalos ou bandas distintas de frequência, é necessária uma função f(x,y) para cada banda. É o caso de imagens coloridas padrão RGB, que são formadas pela informação de cores primárias aditivas, como o vermelho (R - *Red*), verde (G - *Green*) e azul (B - *Blue*).

No processo de digitalização, o sinal analógico de vídeo obtido através da saída do dispositivo de aquisição deve ser submetido a uma discretização espacial e em amplitude para tomar o formato desejável ao processamento computacional.

A amostragem é o processo de discretização espacial e dar-se o nome de quantização ao processo de discretização em amplitude.

Basicamente, a amostragem converte a imagem analógica em uma matriz de M por N pontos, cada qual denominado pixel (ou elemento de imagem):

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & \dots & f(1,N-1) \\ \vdots & \vdots & \dots & \vdots \\ f(M-1,0) & f(M-1,0) & \dots & f(M-1,N-1) \end{bmatrix} \qquad (3.2)$$

Maiores valores de M e N implicam em uma imagem de maior resolução.

Por outro lado, a quantização faz com que cada um destes pixels assuma um valor inteiro, na faixa de 0 a 2n-1. Quanto maior o valor de *n*, maior o número de cores presentes na imagem digitalizada.

Do ponto de vista eletrônico, a digitalização consiste em uma conversão analógica digital na qual o número de amostras do sinal contínuo por unidade de tempo indica a taxa de amostragem e o número de bits do conversor A/D utilizado determina o número cores resultantes na imagem digitalizada.

Na especificação do processo de digitalização deve-se decidir que valores de N, M e *n* são adequados, do ponto de vista de qualidade da imagem e da quantidade de bytes necessários para armazená-la. A tabela 3.3 fornece uma ideia estimada do número de bytes necessários para armazenar uma imagem de N x N pixels com *2n* tons de cinza, calculados como: N x N x *n* / 8.

Tabela 3. - Número de bytes necessários para armazenar uma imagem digital NxN com 2n níveis.

N \ *n*	1	2	3	4	5	6	7	8
32	128	256	512	512	1.024	1.024	1.024	1.024
64	512	1.024	2.048	2.048	4.096	4.096	4.096	4.096
128	2.048	4.096	8.192	8.192	16.384	16.384	16.384	16.384
256	8.192	16.384	32.768	32.768	65.536	65.536	65.536	65.536
512	32.768	65.536	131.072	131.072	262.144	262.144	262.144	262.144
1.024	131.072	262.144	393.216	524.288	655.360	786.432	917.504	1.048.576

Fonte: MARQUES FILHO, Ogê; VIEIRA NETO (2009).

Assume-se que um pixel estará inteiramente contido em um byte, mesmo que isto signifique que alguns bits de cada byte permaneçam vazios. Por exemplo, para *n* igual a 5, assume-se que cada pixel ocupa um byte, restando 3 bits sem utilização em cada byte.

Do ponto de vista qualitativo, pode-se perguntar: quantos pontos e níveis de cinza serão necessários para que a versão digitalizada de uma imagem apresente qualidade comparável à imagem original? Parece evidente que quanto maiores os valores de M, N e *n*, melhor a imagem digital resultante. Mas sabendo que elevados valores de M, N e *n* implicarão em maiores custos de digitalização e armazenagem. Deve existir uma forma de definir valores adequados à qualidade desejada. Convém observar ainda que "qualidade de imagem" é um conceito altamente subjetivo, que também depende fortemente dos requisitos da aplicação dada.

3.8. Ruído [2]

Na prática, as imagens digitais são corrompidas por ruídos durante sua aquisição ou transmissão, requerendo que essas imagens sejam filtradas antes de seu processamento. A definição do termo ruído não é precisa, mas geralmente ruído é considerado informação espúria (indesejada) na imagem. Em sentido amplo, o termo filtragem implica na manipulação

[2] ROSITO JUNG, Cláudio. Filtragem de Imagens com Preservação das Bordas Usando a Transformada Wavelet, Porto Alegre: p. 19-20, 2002.

do conteúdo de frequência da imagem, porém aqui é utilizado para denotar a remoção (ou atenuação) do ruído em uma imagem.

Em particular, é de extrema importância que as bordas sejam preservadas durante a filtragem. As bordas da imagem delimitam as fronteiras entre dois objetos que compõem uma cena, e são fundamentais tanto na percepção do sistema visual humano segundo Hubel (1962), quanto na análise de imagens (por exemplo, na segmentação da imagem em seus objetos componentes). Infelizmente, a grande maioria das técnicas de filtragem existentes na literatura não se mostram eficientes na preservação das bordas. A distinção entre ruído e bordas é muito difícil para um sistema de visão computacional, embora seja normalmente uma tarefa trivial para o sistema visual humano. Na verdade, os processos de filtragem e detecção de bordas são interdependentes, como será visto a seguir.

A caracterização matemática do ruído não é simples, pois existem diferentes tipos de ruídos e com origens distintas. Entretanto, o ruído em uma imagem monocromática é caracterizado geralmente por variações de alta frequência espacial na intensidade de tons de cinza. Por outro lado, as bordas da imagem representam as fronteiras entre os objetos que compõem uma cena, e são caracterizadas por variações bruscas nos tons de cinza (correspondentes às transições entre duas regiões aproximadamente homogêneas associadas a dois objetos distintos), que também correspondem às altas frequências espaciais.

Em geral, as variações de intensidade correspondentes às bordas têm amplitude maior do que as variações de intensidade associadas ao ruído. Contudo, essa hipótese não é válida para bordas de baixo contraste (como, fronteiras entre regiões que tem aproximadamente a mesma média nos tons de cinza), ou quando a quantidade de ruído presente na imagem é grande. Nesses casos, a distinção entre bordas e ruído pode ser bastante complexa, resultando em falsas bordas (ruído erroneamente detectado como borda) e/ou falhas na detecção das bordas de baixo contraste.

Para que a filtragem seja eficiente, o ruído deve ser atenuado nas regiões homogêneas da imagem, mas as bordas não devem ser suavizadas. Para tal, é importante que se conheça a localização das bordas. Por outro lado, a detecção das bordas requer uma imagem filtrada, pois a detecção das mesmas em imagens ruidosas apresenta resultados errôneos. Como exemplo, a figura 3.9 (a) mostra uma imagem contaminada por ruído Gaussiano aditivo, Larson and Shubert (1979). A figura 3.9 (b), mostra as bordas detectadas pelo método de Prewitt, Pratt (1991), enquanto que a figura 3.9 (c) mostra o resultado da filtragem pela

convolução com filtro Gaussiano (Jain, 1989). Pode-se notar que as bordas da imagem filtrada estão borradas, e que a detecção delas não foram eficientes (contornos abertos e varias falsas bordas detectadas).

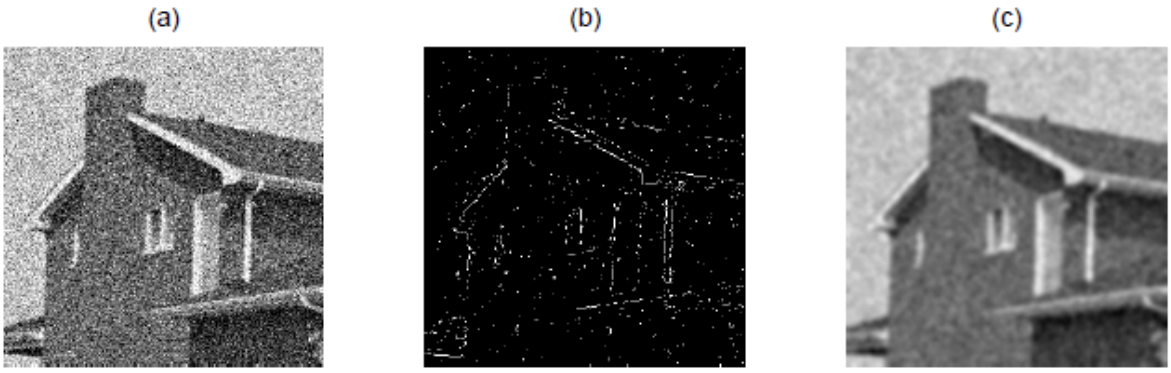

Figura 3. - (a) Imagem ruidosa. (b) Bordas detectadas pelo método de Prewitt. (c) Filtragem pela convolução com uma Gaussina.

Fonte: Jung, Cláudio (2002).

3.9. Filtro Gaussiano [3]

Um dos filtros mais utilizados para atenuação de ruído é o baseado na distribuição Gaussiana. O filtro gaussiano também é muito utilizado para suavização de imagens, com a diferença de não preservar as arestas uma vez que não considera a diferença das intensidades. Ele possui dois parâmetros, a dimensão da janela e um valor para o desvio padrão máximo sigma. Seu comportamento é similar ao filtro passa-baixa, isto é, suavização de imagens. O quanto a imagem será suavizada está relacionada ao desvio padrão sigma, isto é, quanto maior o sigma, mais a imagem é suavizada, não dependendo muito do parâmetro referente a dimensão da janela. Quanto maior o sigma, maior o número de pixels cujo valor é diferente de zero, o que leva os pixels vizinhos a terem maior influência em cada ponto, realizando uma suavização maior na imagem, o que não significa, necessariamente, uma melhoria na qualidade da imagem, conforme mostra a figura 3.10.

[3] Texto modificado, retirado em Joint Bilateral Upsample (disponível em: <http://lvelho.impa.br/ip09/demos/jbu/filtros.html > acesso em 13/10/2012).

Visto que é um filtro utilizado para atenuação do ruído, o método deve ser bem empregado para trabalhar com a detecção de bordas, visto que a qualidade da imagem diminui devido a suavização, dificultado a localização precisa do objeto.

O filtro Gaussiano é definido por:

$$GB[I]_p = \sum_{q \in S} G_\sigma \left(||p - q|| \right) I_q \qquad (3.3)$$

Onde,

$$G\sigma(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \qquad (3.4)$$

A figura 3.10, a seguir, mostra o resultado da aplicação do filtro gaussiano com diferentes σ.

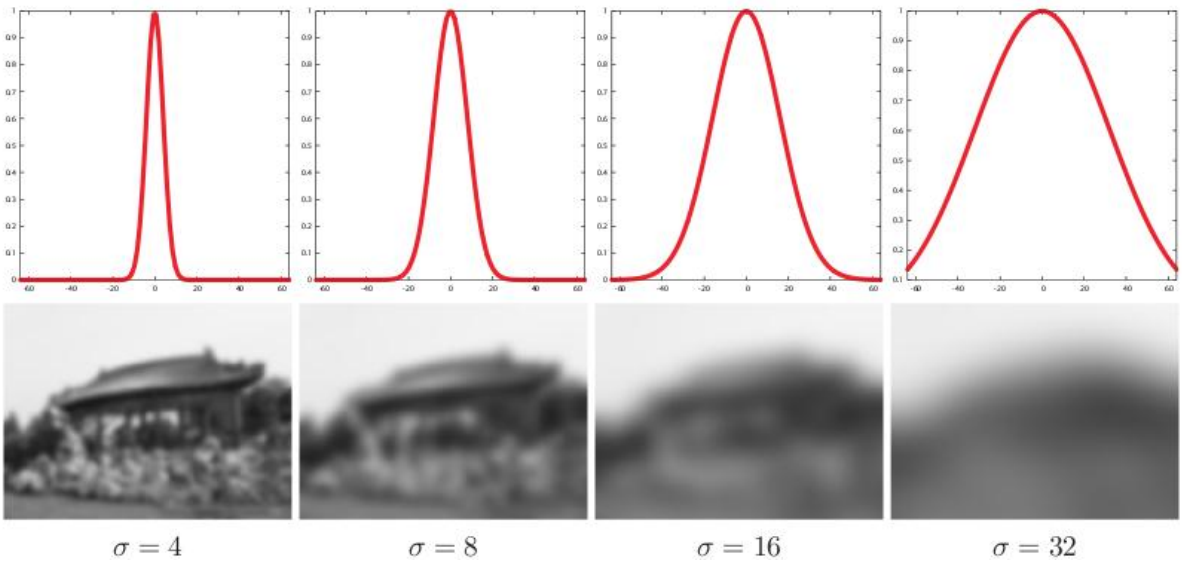

Figura 3. - Filtro Gaussiano

Fonte: extraído da internet no endereço, em 13/10/12: http://lvelho.impa.br/ip09/demos/jbu/filtros.html

3.10. Segmentação Baseada em Bordas [4]

Na detecção de bordas são analisadas as descontinuidades nos níveis de cinza. Neste trabalho ela é empregada para delimitar os objetos encontrados na imagem. Definindo os padrões dos objetos, o sistema é capaz de identificá-los.

Uma borda é o limite entre duas regiões com propriedades relativamente distintas de nível de cinza. As bordas na imagem caracterizam os contornos dos objetos e são bastante úteis para segmentação e identificação de objetos na cena. Porém, quando a imagem é diferenciada, todas as variações dos níveis de cinza são detectadas e, por consequência, detectam-se também bordas espúrias, que é uma forma indesejável de variação. Para que as bordas espúrias, provenientes de ruído ou textura da imagem, não sejam detectadas, deve-se suavizar a imagem antes da detecção. Contudo, existem efeitos inoportunos ligados à suavização, como a perda de informação e o deslocamento de estruturas de feições relevantes na imagem. Além disso, existem diferenças entre as propriedades dos operadores diferenciais comumente utilizados, ocasionando bordas diferentes. Logo, é difícil formular um algoritmo de detecção de bordas que possua um bom desempenho em diferenciados contextos e capture os requisitos necessários aos estágios subsequentes de processamento (Ziou e Tabbone, 1997). Consequentemente, no tocante ao processamento de imagem digital, uma variedade de detectores de bordas tem sido desenvolvida visando diferentes propósitos, com formulações matemáticas diferenciadas e com propriedades algorítmicas distintas.

Com base nos problemas acima mencionados, Canny (1986), desenvolveu um processo de detecção de bordas a partir de critérios de quantificação de desempenho de operadores de bordas conhecidos como os critérios de detecção e de localização.

Canny menciona três critérios básicos. O primeiro deles é denominado Taxa de Erro ou Detecção, consistindo na maximização da razão sinal/ruído (SNR). Quanto maior for o SNR, maior é a probabilidade de se detectar as bordas verdadeiras da imagem. Assim, o detector de bordas deveria detectar somente bordas e nenhuma poderia faltar. O segundo critério especifica que pontos de borda devem estar bem localizados, isto é, as distâncias entre os pontos extraídos pelo detector e as respectivas posições verdadeiras devem ser minimizadas. Tem-se então o critério de Localização (L), definido como sendo o inverso da

[4] Texto extraido: VALE , G. e DAL POZ, A. Processo de detecção de bordas de Canny. Curitiba: Bol. Ciênc. Geod., sec. Artigos, v. 8, no 2, p.67-78, 2002.

distância entre um ponto detectado e a respectiva posição verdadeira. Portanto, quanto maior for L, mais próximos das posições verdadeiras estarão os pontos detectados pelo filtro. Pelo exposto, o projeto de um filtro para a detecção de bordas arbitrárias envolve a maximização de ambos os critérios, o que é equivalente à maximização do produto entre ambos (SNR e L), ficando (Canny, 1986):

$$\left(\frac{\left| \int_{-W}^{W} G(-x) f(x)\, dx \right|}{n_0 \sqrt{\int_{-W}^{W} f^2(x)\, dx}} \right) \cdot \left(\frac{\left| \int_{-W}^{W} G'(-x) f'(x)\, dx \right|}{n_0 \sqrt{\int_{-W}^{W} f'^2(x)\, dx}} \right) \qquad (3.5)$$

onde “f(x)” é a resposta de impulso do filtro definido no intervalo [-w; w], “G(x)” é uma borda unidimensional e “n0” a quantificação do ruído da imagem. Assume-se que a borda está centrada em x = 0. Na equação 3.5, a primeira quantidade entre parêntesis corresponde ao SNR e a segunda à L. A condição de filtro ótimo de Canny deve ainda atender a um terceiro critério, denominado critério de resposta múltipla. A idéia básica é que deve haver um único ponto de borda onde existe uma única borda verdadeira. Seja (Canny, 1986):

$$x_{max} = 2\pi \left(\frac{\int_{-\infty}^{+\infty} f'^2(x)\, dx}{\int_{-\infty}^{+\infty} f''^2(x)\, dx} \right)^{1/2} \qquad (3.6)$$

a expressão matemática para a distância (xmax) entre máximos adjacentes na resposta do filtro f(x) devido ao ruído. Assim, ao maximizar a condição dada pela equação 3.5, deve-se também garantir que xmax seja maior possível, aumentando a possibilidade de separação de máximos verdadeiros dos falsos na saída do filtro f(x).

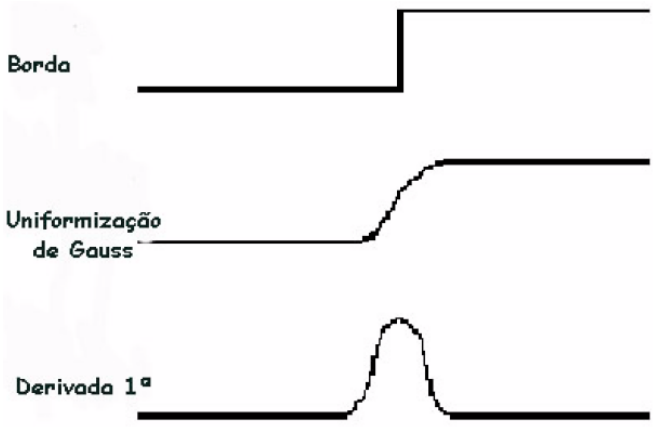

Figura 3.- Detecção de bordas de Canny

Fonte: Autor.

Sendo assim, se considerar uma borda de uma dimensão variando no contraste (um degrau) e então convolucionando a borda com a função de uniformização de Gauss, o resultado será uma variação contínua do valor inicial ao final, com uma inclinação máxima no ponto onde existe o degrau. Se esta continuidade é diferenciada em relação a x, esta inclinação máxima será o máximo da nova função em relação a original (figura 3.11).

Os máximos da convolução da máscara e da imagem indicarão bordas na imagem. Este processo pode ser realizado através do uso de uma função de Gauss de 2-dimensões na direção de x e y. Os valores das máscaras de Gauss dependem da escolha do sigma na equação de Gauss, visto na fórmula 3.4 deste trabalho.

A aproximação do filtro de Canny para detecção de bordas é G'. Convolucionando a imagem com G' se obterá uma imagem *I* que mostrará as bordas, mesmo na presença de ruído. A convolução é relativamente simples de ser implementada, mas é cara computacionalmente, especialmente se for em 2-dimensões. Entretanto, uma convolução de Gauss de 2-dimensões pode ser separada em duas convoluções de Gauss de 1-dimensão.

A intensidade computacional do detector de bordas de Canny é relativamente alta, e os resultados são geralmente pós-processados para maior clareza. Entretanto, o algoritmo é mais eficiente no processamento de imagens com ruídos ou com bordas difusas.

Algoritmo de Canny:

- Ler a imagem (I) a ser processada;

- Criar uma máscara de Gauss de 1-D (G) para convolucionar I. O desvio (S) de Gauss é um parâmetro para o detector de bordas;
- Criar uma máscara de 1-D para a primeira derivada de Gauss nas direções x e y; nomear como Gx e Gy. O mesmo valor S é usado;
- Convolucionar a imagem I com G percorrendo as linhas na direção x (Ix) e percorrer as colunas na direção y (Iy);
- Convolucionar Ix, com Gx, para dar I´x (o componente x). De I convolucionado com a derivada de Gauss. E convolucionar Iy, com Gy para dar I´y;
- Neste ponto, o resultado dos componentes x e y devem ser "combinados". A magnitude do resultado é computada para cada pixel (x,y).

3.11. Operações de Dilatação e Erosão da imagem

Duas operações básicas são fundamentais para o tratamento da imagem que auxiliarão posteriormente o processo de segmentação e leitura dos caracteres da placa.

- **Dilatação**: é a aplicação de um elemento estruturante de forma concêntrica sobre um conjunto definido de pontos (brancos ou pretos) em uma imagem, de maneira que o elemento estruturante adicione informação sobre a vizinhança destes pontos. Ou seja, pode-se imaginar que o elemento estruturante desliza sobre um conjunto de pontos dilatando sua vizinhança numa proporção que varia conforme as dimensões do elemento estruturante (Gonzalez and Woods, 2002). Esta operação é utilizada principalmente para preencher intervalos e lacunas indesejáveis na imagem.
- **Erosão**: é o inverso da dilatação. A aplicação do elemento estruturante ocorre analogamente à operação anterior, porém, ao invés de dilatar a vizinhança do ponto percorrido inserindo informação, o elemento retira informação (gerando erosão nas áreas percorridas) (Gonzalez and Woods, 2002). Esta operação é utilizada principalmente para eliminar detalhes irrelevantes, como ruídos, e abrir intervalos ou lacunas em regiões de conexão indesejada.

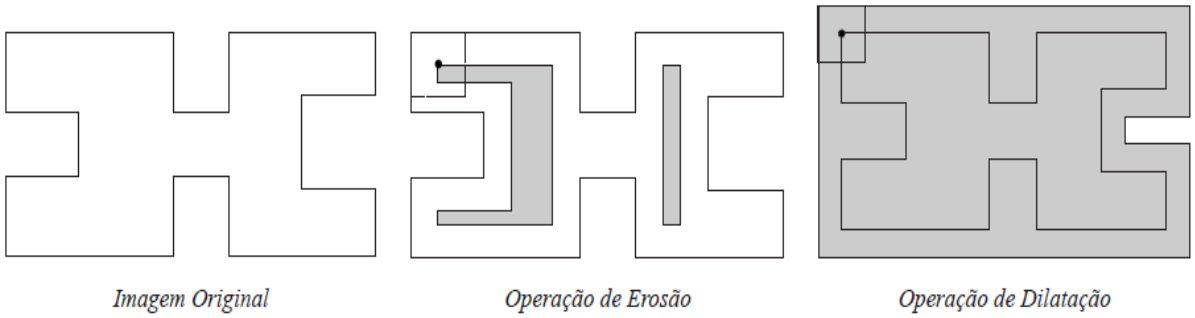

Figura 3. - Princípio de funcionamento de operadores morfológicos.

Fonte: Gonzalez and Woods (2002).

3.12. OpenCV

Conforme mencionado no objetivo do trabalho, o OpenCV é a interface de programação utilizada para o desenvolvimento do software.

É uma biblioteca de visão computacional "open source" (código aberto) escrito na linguagem de programação C e C++. Sua infraestrutura trabalha com um grau de desempenho em processamento - muito exigido no tratamento de imagens em tempo real – retirando proveito da tecnologia de multiprocessamento.

Atualmente com um pouco mais de 500 funções, proporciona uma série de facilidades no processo de entrada e saída de imagens e vídeos, estrutura de dados, álgebra linear, filtros, calibração de câmera, tratamento de imagem etc.

Desde seu lançamento em janeiro de 1999, várias aplicações e pesquisas estão utilizando o OpenCV. Alguns exemplos: na redução de ruído em imagens médicas, na análise de objeto, nos sistemas de segurança e detecção objeto, em aplicação militar e inspeção aérea.

A biblioteca é basicamente estruturada em cinco principais componentes, dentre as quais quatro são mostradas na figura 3.13.

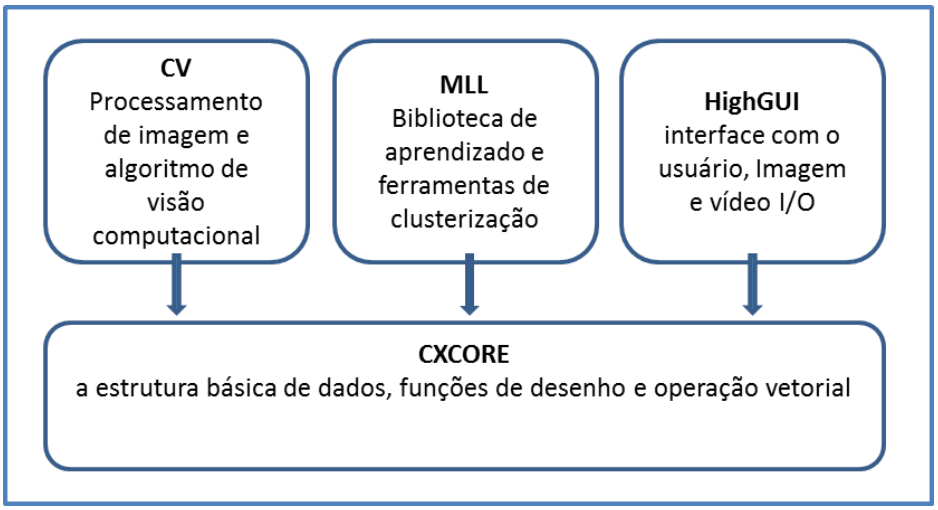

Figura 3. - A estrutura básica do OpenCV.

Fonte: Autor.

O módulo CV contêm as principais funcionalidades de processamento de imagem, contemplando o algoritmo de visão computacional. O módulo ML é a biblioteca de aprendizado, que inclui funções de estatísticas e ferramentas de clusterização. O módulo HighGUI contêm o controle de interface com o usuário e as rotinas de entrada e saída de vídeos e imagens. O CXCore contêm a estrutura básica de dados, funções de desenho e operação vetorial. A figura 3.13 não contempla o módulo CVAux integrando técnicas de reconhecimento de objetos, pois ainda está em fase experimental.

3.13. Tesseract OCR

No trabalho é empregada a biblioteca Tesseract OCR para leitura da placa automotiva.

OCR significa “Optical Character Recognition”, é o processo pelo qual o computador consegue “ler” o texto contido numa imagem. Quando é passada uma página de texto de um livro no scanner o resultado é uma foto. Essa foto precisa passar por um processo de OCR para extrair o texto dela.

Originalmente desenvolvido pela Hewlett Packard (HP) entre 1985 e 1995, o Tesseract OCR foi redescoberto pelo Google. A biblioteca também é “open source” e desenvolvida na linguagem C++, onde combinada com a biblioteca de processamento de imagem Leptonica,

pode ler uma grande variedade de formatos de imagem e convertê-los em texto em mais de 40 idiomas.

A partir da versão 3.x, a ferramenta fornece o módulo totalmente treinável, podendo lidar com quaisquer caracteres na codificação UTF-8 (8-bit Unicode Transformation Format).

Por conveniência, o que se segue é um breve resumo de como funciona Tesseract:

- Os contornos são analisados e armazenados;
- Os contornos são reunidos como bolhas;
- As bolhas são organizadas em linhas de texto;
- Linhas de texto são divididas em palavras;
- A primeira passagem do processo de reconhecimento tenta encontrar cada palavra, uma de cada vez;
- As palavras encontradas são então informadas ao treinador adaptativo;
- Existe uma segunda análise das palavras que não foram entendidas na primeira passagem
- Os espaços borrados do texto são considerados como pequenas marcações;
- As palavras encontradas são informadas;

Durante esses processos, o Tesseract utiliza:

- Algoritmos para detecção de linhas de texto de uma página distorcida;
- Algoritmos para detectar a proporção das palavras e símbolos (essa proporção verifica se todas as letras de uma palavra tem a mesma largura);
- Algoritmos para cortar caracteres unidos ou associar caracteres quebrados;
- Analisador linguístico para identificar as palavras mais utilizadas dentre um conjunto de caracteres;
- Dois classificadores de caracteres: um classificador estático, e um classificador adaptativo que utiliza os dados de treino, distinguindo caracteres maiúsculos e minúsculos.

O projeto Tesseract está no seguinte endereço da internet: http://code.google.com/p/tesseract-ocr/.

3.14. Linguagem de Programação C/C++[5]

No projeto proposto é utilizada uma mistura da linguagem de programação C e C++, pelo fato das bibliotecas empregadas: OpenCV e Tesseract OCR, serem desenvolvidas para estas linguagens.

Abaixo uma visão geral das linguagens C e C++, mostrando algumas características:

3.14.1. Linguagem C

O desenvolvimento inicial de C ocorreu nos laboratórios Bell da AT&T, entre 1969 e 1973. Segundo Ritchie, o período mais criativo ocorreu em 1972. Deu-se o nome "C" à linguagem porque muitas das suas características derivaram de uma linguagem de programação anterior chamada "B". Há vários relatos que se referem à origem do nome "B": Ken Thompson dá crédito à linguagem de programação BCPL mas ele também criou uma outra linguagem de programação chamada "Bom", em honra da sua mulher Bonnie. Por volta de 1973, a linguagem C tinha se tornado suficientemente poderosa para que grande parte do núcleo de Unix, originalmente escrito na linguagem de programação PDP-11/20 assembly, fosse reescrito em C. Este foi um dos primeiros núcleos de sistema operativo que foi implementado numa linguagem sem ser o assembly.

C é uma linguagem imperativa e procedural para implementação de sistemas. Seus pontos de design foram para ele ser compilado, fornecendo acesso de baixo nível à memória e baixos requerimentos do hardware. Também foi desenvolvido para ser uma linguagem de alto nível, para maior reaproveitamento do código. C foi útil para muitas aplicações que forma codificadas originalmente em Assembly. Essa propriedade não foi acidental; a linguagem C foi criada com o objetivo principal: facilitar a criação de programas extensos com menos erros, recorrendo ao paradigma da programação algorítmica ou procedimental, mas sobrecarregando menos o autor do compilador, cujo trabalho complica-se ao ter de realizar as características complexas da linguagem. Para este fim, a linguagem C possui as seguintes características:

---

[5] Texto modificado, retirado em Expert.net (disponível em: <http:// http://www.tiexpert.net/programacao/c/index.php > acesso em 13/10/2012).

- Uma linguagem nuclear extremamente simples, com funcionalidades não-essenciais, tais como funções matemáticas ou manuseamento de ficheiros (arquivos), fornecida por um conjunto de bibliotecas de rotinas padronizada;
- A focalização no paradigma de programação procedimental;
- Um sistema de tipos simples que evita várias operações que não fazem sentido;
- Uso de uma linguagem de pré-processamento, o pré-processador de C, para tarefas tais como a definição de macros e a inclusão de múltiplos ficheiros de código fonte;
- Um acesso de baixo-nível à memória do computador, através do uso de ponteiros;
- Parâmetros que são sempre passados por valor para as funções e nunca por referência (É possível simular a passagem por referência com o uso de ponteiros);
- Definição do alcance lexical de variáveis;
- Estruturas de variáveis (structs), que permitem que dados relacionados sejam combinados e manipulados como um todo.

3.14.2. Linguagem C++

O C++ foi inicialmente desenvolvido por Bjarne Stroustrup dos Bell Labs durante a década de 1980 com o objetivo de melhorar a linguagem de programação C ainda que mantendo máxima compatibilidade. Stroustrup percebeu que a linguagem Simula possuía características bastante úteis para o desenvolvimento de software, mas era muito lenta para uso prático. Por outro lado, a linguagem BCPL era rápida, mas possuía demasiado baixo nível, dificultando sua utilização em desenvolvimento de aplicações. Durante seu período na Bell Labs, ele enfrentou o problema de analisar o kernel UNIX com respeito à computação distribuída. A partir de sua experiência de doutorado, começou a acrescentar elementos do Simula no C. O C foi escolhido como base de desenvolvimento da nova linguagem pois possuía uma proposta de uso genérico, era rápido e também portável para diversas plataformas. Algumas outras linguagens que também serviram de inspiração para o cientista da computação foram ALGOL 68, Ada, CLU e ML.

Vantagens:

- Produção de código o quanto mais eficiente possível;
- Possibilidade em programação de alto e baixo nível;
- Alta flexibilidade, portabilidade e consistência;
- Adequado para grandes projetos;
- Ampla disponibilidade e suporte, devido principalmente à grande base de desenvolvedores;
- Não está sob o domínio de uma empresa (em contraste do Java — Sun ou Visual Basic — Microsoft);
- Padronização pela ISSO;
- Grandes possibilidades para a metaprogramação e programação genérica;
- Compatilidade com C, resultando em vasta base de códigos.

Desvantagens:

- Compatilidade com o C herdou os problemas de entendimento de sintaxe do mesmo;
- Os compiladores atuais nem sempre produzem o código mais otimizado, tanto em velocidade quando tamanho do código;
- Grande período para o aprendizado;
- A biblioteca padrão não cobre áreas importantes da programação, como threads, conexões TCP/IP, interface gráfica e manipulação de sistemas de arquivos, o que implica na necessidade de criação de bibliotecas próprias para tal, que pecam em portabilidade;
- Devido à grande flexibilidade no desenvolvimento, é recomendado o uso de padrões de programação mais amplamente que em outras linguagens.

4. MODELO PROPOSTO PARA DETECÇÃO E RECONHECIMENTO DE PLACA AUTOMOTIVA

Neste capítulo será detalhado o desenvolvimento da aplicação, demonstrando os passos utilizados desde a instalação das bibliotecas bases para o projeto aos métodos utilizados para detecção e leitura da placa veicular.

4.1. Apresentação Geral do Modelo proposto

Conforme mostrado de modo superficial na arquitetura do projeto (vide figura 1.1), para obtenção dos caracteres alfanumérico da placa, foi desenvolvida a seguinte estratégia baseada nas metodológicas supracitadas no capítulo anterior. Na figura 4.1 abaixo, observa-se o modelo proposto da visão computacional empregado neste projeto:

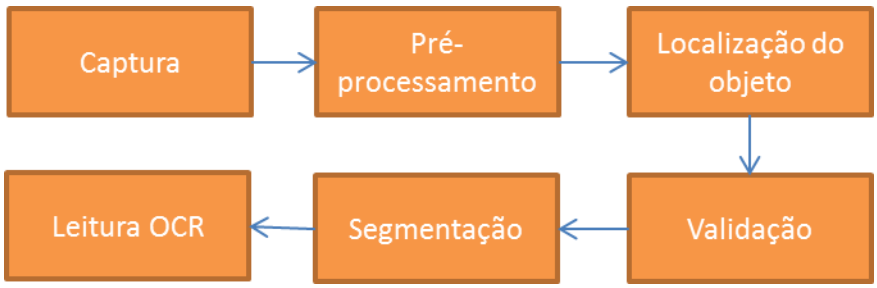

Figura 4.- Modelo proposto no projeto

Fonte: Autor.

4.2. Descrição das Etapas do Modelo

Captura: a imagem pode ser adquirida por um dispositivo que filma em tempo real; por exemplo, de uma *webcam*, câmera infravermelha de segurança, câmera profissional de alta resolução etc. Ou, através de uma gravação armazenada em uma memória; no caso, um *pendrive* ou um *hard disk*. É importante enfatizar, que no método utilizado, não é pré-estabelecido que já se tenha a imagem da placa do veículo fotografado, visto que o objetivo aqui é a detecção automática da placa.

Pré-processamento: nessa etapa são utilizadas algumas técnicas para o tratamento da imagem, aumentando as condições para o sistema de localização do objeto. Neste trabalho empregaram-se técnicas de diminuição de ruído de Gauss, conversão de cores do formato RGB (com três canais, vermelho-verde-azul) para escala de cinza (único canal), experimentando também a técnica de conversão em HSL (matriz-saturação-luz) para manipulação das cores.

Localização do objeto: aqui serão detectadas as bordas dos objetos presentes na imagem. Nesse processo, aplicou-se a técnica de detecção e segmentação de bordas de Canny. Tentou-se também utilizar outras técnicas de detecção de borda, como Sobel e Wavelet 2D, onde a técnica de Sobel foi implementada (vide apêndice A), porém sem êxito significante ao trabalho devido ao reduzido tempo de projeto e pesquisas.

Validação: após a localização dos objetos, os mesmo serão validados, procurando localizar a placa do carro. Aqui são utilizadas como parâmetros as dimensões da placa, como a altura e o comprimento. Assim, objetos retangulares e proporcionais pela razão do comprimento e da altura serão considerados como objetos válidos. Serão apresentadas também outras condições de validação da placa.

Segmentação: é um processo de recorte do objeto localizado e validado.

Leitura OCR: nessa fase a imagem é preparada para o processo de leitura da imagem onde o reconhecedor de caracteres irá retornar os valores alfanuméricos da placa.

4.3. Ferramentas Utilizadas

Para o desenvolvimento deste trabalho, foi utilizado o sistema operacional Windows 7, não precisando ter uma experiência aprofundada sobre este sistema operacional. Abaixo é demonstrado o processo de instalação das ferramentas utilizadas.

4.3.1. Microsoft Visual Studio 2008

Conforme explicado anteriormente, as bibliotecas aqui utilizadas (OpenCV e o Tesseract OCR), tem suporte a linguagem de programação C/C++. Além disso, essas bibliotecas podem ser configuradas para utilização em diversos ambientes de

desenvolvimento (IDE) empregando para isso o programa CMake. Nesse trabalho, as bibliotecas foram configuradas para o Microsoft Visual Studio 2008 (MSVS), devido a um suporte completo e vasto conteúdo literário.

O MSVS na versão Professional Edition foi instalado utilizando o instalador no modo *custom* e desmarcando a opção de instalação do SQL Server Express, pois não se fará proveito neste trabalho. É importante observar também se as linguagens C e C++ estão selecionadas para instalação, na opção *Language Tools*.

O processo de instalação não apresentou nenhuma dificuldade ou problema.

Na figura 4.2 abaixo, exibe a interface do instalador.

Figura 4. - Instalador do Microsoft Visual Studio 2008

Fonte: Autor.

4.3.2. OpenCV

Resumo da instalação:

- Optou-se utilizar a versão mais recente do OpenCV a partir do repositório SVN do desenvolvimento, na versão 2.4.2. A ideia era obter o código fonte e conhecer o processo de compilação da biblioteca;

- Instalação do CMake: utilizado para criação do projeto OpenCV para o formato MSVS 2008, a partir dos fontes da biblioteca. No final desse processo será gerado o arquivo “sln”, extensão MSVS utilizado para organiza projetos, itens de projeto e itens de solução, fornecendo ao ambiente, referências a seus locais no disco.
- Compilação do projeto OpenCV no MSVS 2008;
- Finalmente, a configuração no MSVS para utilizar a biblioteca OpenCV compilada.

4.3.2.1. Baixando o OpenCV

Para obter o código fonte que é atualizado quase diariamente, é necessário baixar a partir do repositório SVN, no endereço: “*http://code.opencv.org/svn/opencv/trunk/opencv*”.

Nesse caso, o programa TortoiseSVN realizou esta tarefa no dia 20/09/2012 baixando a versão 2.4.2, conforme figura 4.3, abaixo:

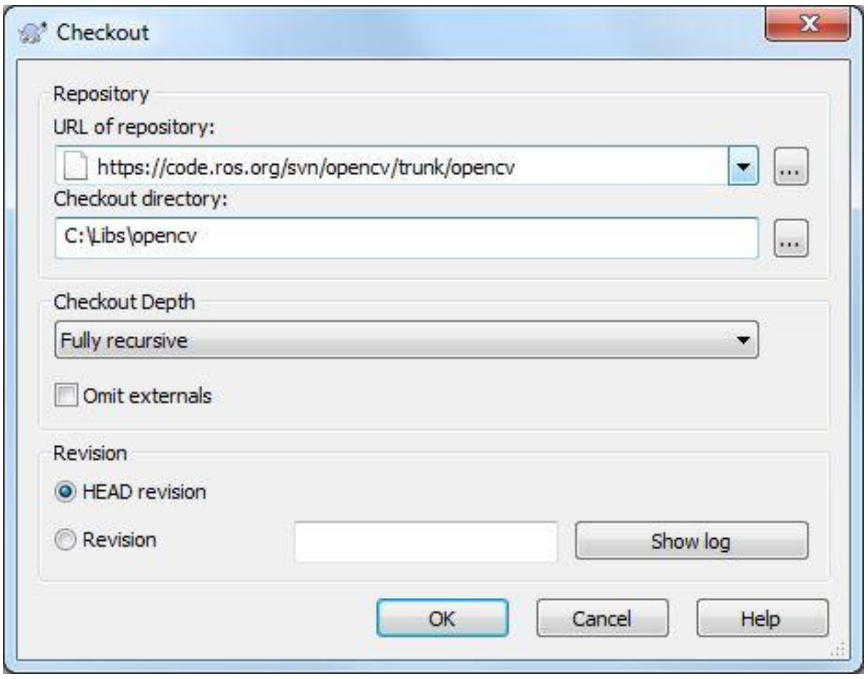

Figura 4. - TortoiseSVN

Fonte: Autor.

Seguindo os passos:

- Deve-se localizar a pasta onde deseja fazer o download dos códigos fontes. Ex.: “C:\opencv”;
- Com o botão direito do mouse escolha “SVN checkout” no menu de contexto;
- Preencher o URL: *“http://code.opencv.org/svn/opencv/trunk/opencv”;*
- Escolher o diretório de checkout. Ex.: “C:\opencv”;
- Clicar em “OK”. O programa começa a baixar os fontes.

4.3.2.2. CMake

CMake criará os arquivos de projeto para versão MSVS 2008 necessários para construir posteriormente os fontes do OpenCV.

A versão utilizada foi a 2.8.9, baixado no endereço: *“http://www.cmake.org/”*.

Seguindo os passos:

- Abrir CMake (cmake-gui);
- Escolher o caminho do código fonte (onde foi baixado OpenCV). Neste diretório deve conter o arquivo “CMakeLists.txt”. Ex.: “C:\opencv”;
- Escolher o diretório de saída (onde construir os binários). Ex.: “C:\opencv\msvc2008”; Clique em “Sim” para criar o diretório se não existir;
- Clicar no botão “Configurar”;
- Escolher o compilador / IDE que deseja usar. Neste caso, escolha Visual Studio 9.
- Escolher as opções que deseja usar. Como construir exemplos, construir opencv_core, construir opencv_highgui, etc;
- Clicar no botão “Configurar” até que todos os conflitos sejam resolvidos (todas as linhas vermelhas mudam a sua cor para branca);
- Clicar em “Gerar”.

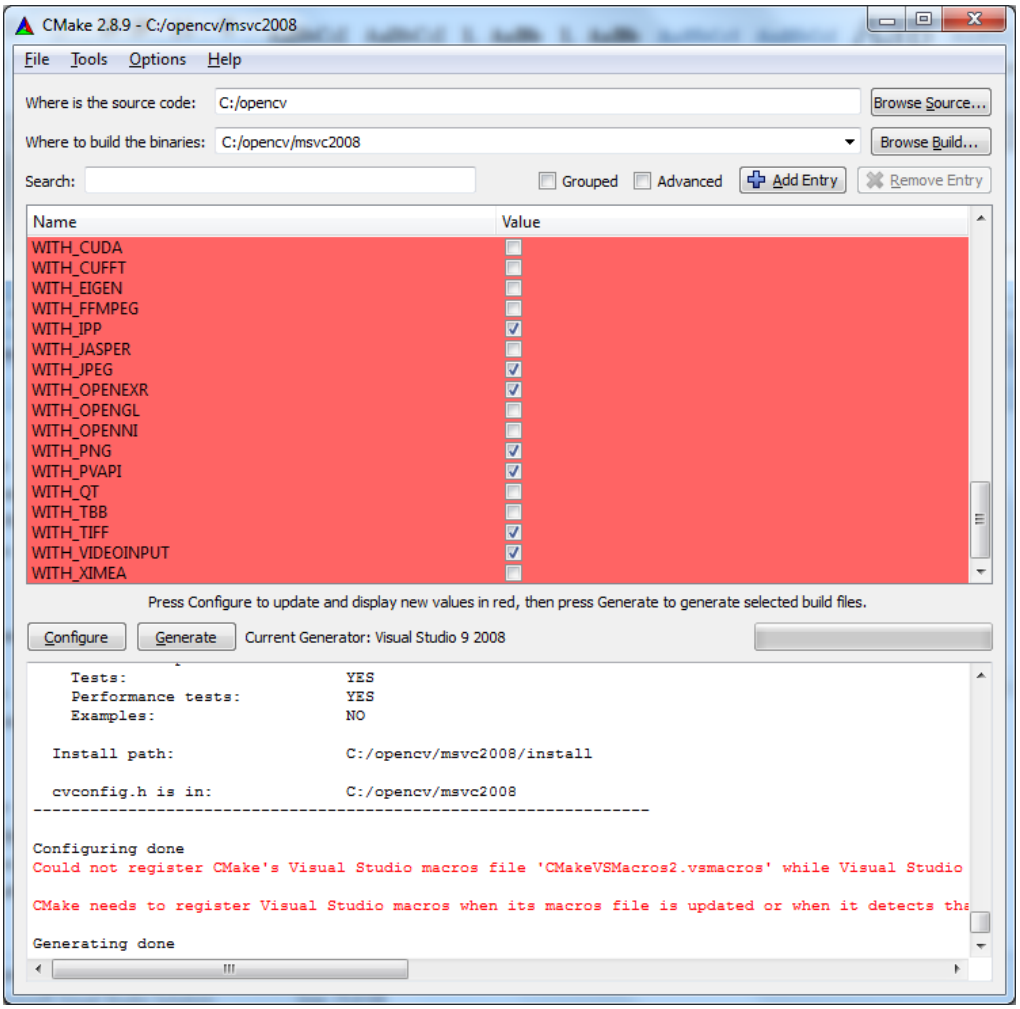

Figure 4. - CMake

Fonte: Autor.

4.3.2.3. Compilando com o MSVS 2008

Para compilar o código fonte do OpenCV com o Visual Studio 2008:

- Abrir o arquivo de solução, localizado no diretório de saída informado no CMaker. Ex.: "C:\opencv\msvc2008\OpenCV.sln";
- Aguardar até que todos os arquivos sejam completamente carregados;
- Pressionar F7 para criar a solução.

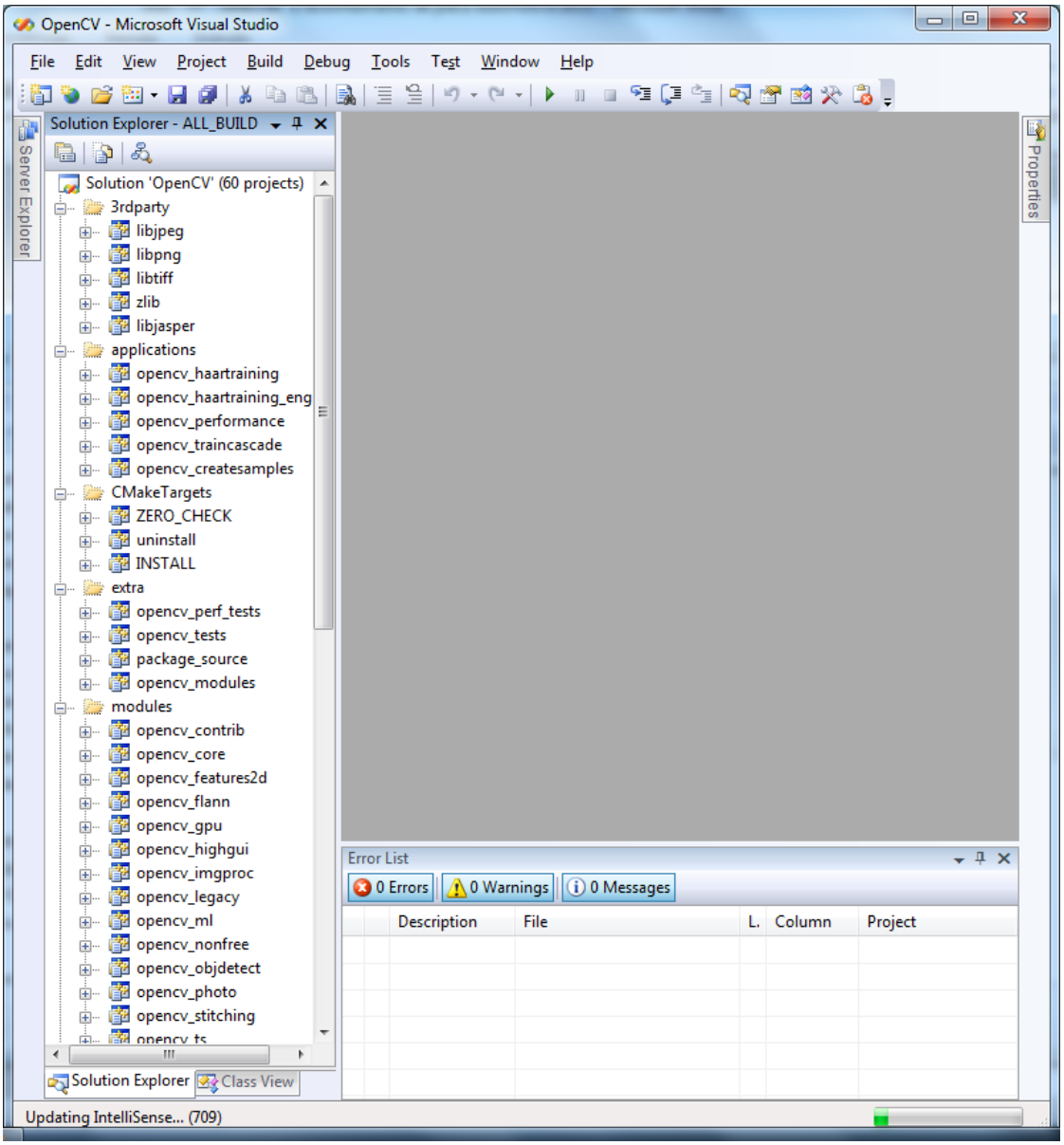

Figura 4. - Compilação do OpenCV

Fonte: Autor.

No final desse processo, será gerado um diretório. Como, por exemplo: “C:\opencv\build”.

4.3.2.4. Configurando a biblioteca OpenCV no MSVS 2008

1. Configurando o Visual Studio

- Abrir *VC++ Directories: Tools > Options… > Projects and Solutions > VC++ Directories*;
- Escolher "*Show directories for: Include files*";

Adicionar "$openCVDir\include"

- Escolher "*Show directories for: Library files*";

Adicionar "$openCVDir\lib"

- Escolher "*Show directories for: Source files*";

Adicionar "$openCVDir\bin"

2. Configurando o projeto de reconhecimento de placa veicular

- Abrir Propriedades do Projeto: NomedoProjeto > *Properties*;
- Escolher "*Linker"*: *Configuration Properties > Linker > Input*;

Em *Configuration* , selecionar o modo*: Debug* | *Release*;

- Para *Configuration Debug:*

No "*Additional Dependencies*" , colocar em cada linha:

opencv_calib3d242d.lib

opencv_contrib242d.lib

opencv_core242d.lib

opencv_features2d242d.lib

opencv_flann242d.lib

tbb.lib

opencv_gpu242d.lib

opencv_haartraining_engined.lib

opencv_highgui242d.lib

opencv_imgproc242d.lib

opencv_legacy242d.lib

opencv_ml242d.lib

opencv_nonfree242d.lib

opencv_objdetect242d.lib

opencv_photo242d.lib

opencv_stitching242d.lib

opencv_ts242d.lib

opencv_video242d.lib

opencv_videostab242d.lib

- Para *Configuration Release:*

No "*Additional Dependencies*", colocar em cada linha:

opencv_contrib242.lib

opencv_core242.lib

opencv_features2d242.lib

opencv_flann242.lib

opencv_gpu242.lib

opencv_haartraining_engine.lib

opencv_highgui242.lib

opencv_imgproc242.lib

opencv_legacy242.lib

opencv_ml242.lib

opencv_nonfree242.lib

opencv_objdetect242.lib

opencv_photo242.lib

opencv_stitching242.lib

opencv_ts242.lib

opencv_video242.lib

opencv_videostab242.lib

Feito isso, o projeto reconhecerá todas as bibliotecas do OpenCV.

4.3.3. Tesseract OCR

Resumo da instalação:

- Conforme explicado anteriormente, o Tesseract OCR compartilha recursos da biblioteca Leptonica, necessitando a prévia instalação dessa biblioteca;
- Para instalação do Tesseract OCR, a equipe de suporte ao produto recomenda baixar o código fonte para manter o produto atualizado. Sendo assim, o mesmo processo utilizado para o OpenCV foi realizado, utilizando o repositório SVN;
- Compilar a biblioteca no MSVS2008;
- Configurar a biblioteca no MSVS para utilização no projeto.

4.3.3.1. Leptonica

A versão utilizada foi a 1.68 pré-compilada, no endereço:

"http://code.google.com/p/leptonica/downloads/detail?name=leptonica-1.68-win32-lib-include-dirs.zip";

Feito isso, descompacte-a para "C:\opencv\tesseract\".

4.3.3.2. Baixando o Tesseract OCR

A partir do repositório SVN, o programa TortoiseSVN realizou a mesma tarefa utilizada para o OpenCV, conforme figura 4.6, abaixo:

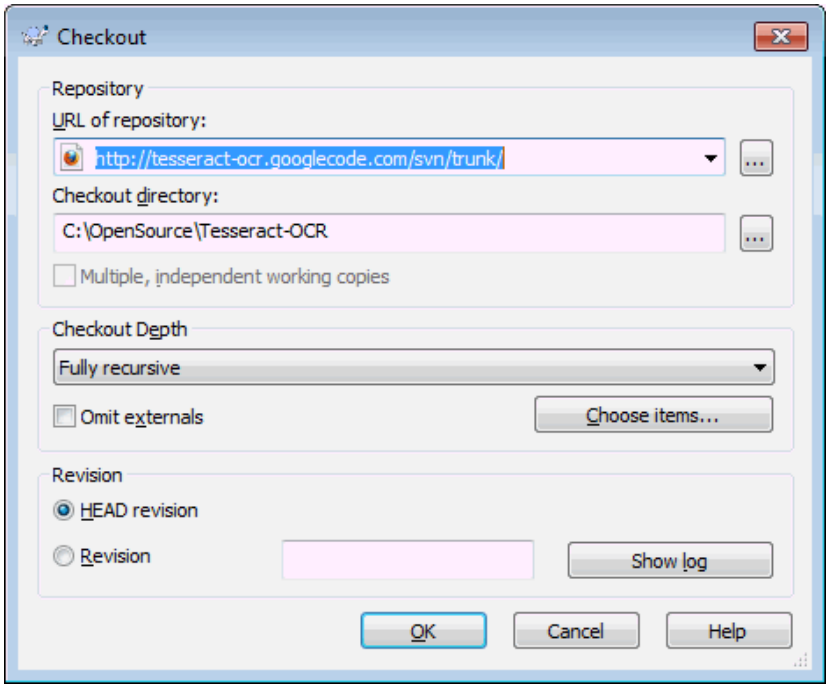

Figura 4. - Tortoise Tesseract OCR

Fonte:Autor

Seguindo os passos:

- Ir até a pasta onde deseja fazer o download das fontes. Ex.: "C:\OpenSource\Tesseract-OCR";
- Com o botão direito do mouse escolher "SVN checkout" no menu de contexto;
- Preencher o URL: *"http://tesseract-ocr.googlecode.com/svn/trunk/"*;
- Escolher o diretório de checkout. Ex.: "C:\OpenSource\Tesseract-OCR";
- Clicar em OK. Ele começará a baixar os códigos fontes.

A versão da biblioteca é a 3.0.2.

4.3.3.3. Compilando o Tesseract OCR

Para construir o Tesseract com o Visual Studio 2008:

- Abra o arquivo de solução, localizado no diretório de saída. Ex.: "C:\ OpenSource\Tesseract-OCR\vs2008\tesseract.sln";
- Aguarde até que todos os arquivos sejam completamente carregados;
- Abrir Propriedades do Projeto: libtesseract302 > *Properties*;
- Em *Configuration,* selecionar o modo*: DLL_Debug* | *DLL_Release* | Lib_Debug | Lib_Release;
- Compilar para cada configuração acima o projeto *libtesseract302*, clicando com o botão direito do mouse o e selecione *Build*;

No final desse processo serão gerados quatro diretórios na pasta "C:\ OpenSource\Tesseract-OCR\vs2008\", chamados: *LIB_Debug*, *LIB_Release*, *DLL_Debug e DLL_Release*;

4.3.3.4. Configurando a biblioteca Tesseract OCR no MSVS 2008

1. Configurando o Visual Studio
   - Abrir *VC++ Directories: Tools > Options... > Projects and Solutions > VC++ Directories*;
   - Escolher "*Show directories for: Include files*";

   Adicionar:

   "C:\OpenSource\Tesseract-OCR\include",

   "C:\OpenSource\Tesseract-OCR\include\leptonica";

   "C:\OpenSource\Tesseract-OCR\tesseract-ocr\ccutil";

   "C:\OpenSource\Tesseract-OCR\tesseract-ocr\ccstruct";

   "C:\OpenSource\Tesseract-OCR\tesseract-ocr\ccmain";

   "C:\OpenSource\Tesseract-OCR\tesseract-ocr\api"

2. Configurando o projeto de reconhecimento de placa veicular
   - Abrir Propriedades do Projeto: NomedoProjeto > *Properties*
   - Escolher "*Linker"*: *Configuration Properties > Linker > Input*

   Em *Configuration*, selecionar o modo*: Debug* | *Release*

   - Para *Configuration Debug*:

   libtesseract302d.lib

   - Para *Configuration Release*:

   libtesseract302.lib

3. Adicionando a biblioteca ao projeto
   - Adicionar os seguintes arquivos no pasta Debug do projeto:

   Libfftw3-3.dll

   Liblept168.dll

   Libtesseract302d.dll

Completada todas as instalações e configurações dos pré-requisitos de *software*, será possível desenvolver o projeto aqui proposto.

4.4. Descrição da Implementação

A seguir, será descrito a implementação do software desenvolvido neste projeto, de acordo com as etapas do modelo proposto. Com isso, se compreenderá melhor os conceitos da visão computacional e da própria solução final, de forma a contribuir com os futuros trabalhos acadêmicos que dê esse enfoque. No apêndice A, contém o código fonte completo. A figura a seguir representa a estrutura do programa implementado.

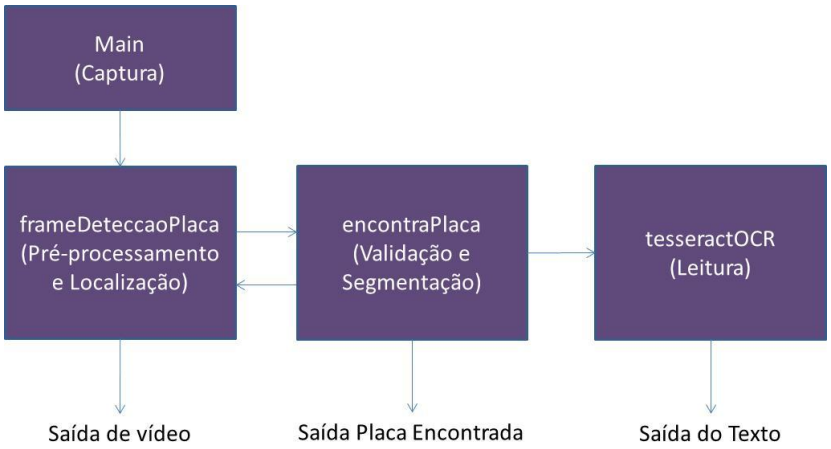

Figura 4. Estrutura do programa implementado

Fonte: Autor.

1. O programa é inicializado pela estrutura *Main* que realiza o processo de captura da imagem e chama o método frameDeteccaoPlaca;
2. O método frameDeteccaoPlaca lê cada *frame* do vídeo e é responsável pelo pré-processamento e localização das bordas da imagem;
3. A função encontraPlaca valida todas as bordas encontradas tentando encontrar objetos que tenham a dimensão de uma placa de carro ou caminhão. Caso positivo, realiza a segmentação e passa esta imagem para o método tesseractOCR;
4. O método tesseractOCR lê a imagem segmentada e a transforma para o formato texto (caracteres).

4.4.1. A captura

O algoritmo de captura da biblioteca OpenCV foi otimizado para trabalhar com diversas operações de processamento de imagem. Também provê interface para diferentes modelos de câmera.

Nesse trabalho, foi utilizado uma câmera de baixíssima resolução (640x480), 8 bits por pixel (256 cores), no padrão VGA (*Video Graphics Array*), provendo otimização no processamento da imagem, porém dificultando proporcionalmente o processo de reconhecimento de padrões. O motivo pela escolha foi a falta de recurso do autor na obtenção

de uma câmera de alta qualidade. Por outro lado, funcionando em câmeras de baixa resolução, é de se esperar um melhor resultado com câmeras que trabalham em alta resolução.

É importante destacar, que para um projeto com finalidade comercial é fundamental a escolha de um sistema ótico adequado, exigindo um estudo mais complexo de câmeras, conforme o anunciado do tópico 3.4.

A seguir, o trecho de código para captura de vídeo na câmera (em tempo real) extraído do código fonte utilizado nesse trabalho:

```
24    CvCapture *capture = NULL;
30    //capture = cvCaptureFromCAM( -1 );
```

ou

```
28    capture = cvCreateFileCapture( "teste1.wmv" );
```

A variável "*capture*" foi declarada utilizando a estrutura de vídeo chamada *CvCapture* apontando para uma área vazia da memória. Esta estrutura não tem uma interface pública e é usada somente como parâmetro para as funções de captura de vídeo. A variável recebe então a função *cvCaptureFromCAM*, onde inicializa a câmera. Da mesma forma, caso se queira utilizar um vídeo gravado, utiliza-se a função *cvCreateFileCapture*.

```
22 IplImage *frame;
36 frame = cvQueryFrame( capture );
```

A variável "*frame*" foi declarada utilizando a estrutura *IplImage*. Essa estrutura é a mais completa para trabalhar com imagem no OpenCV, definindo propriedades da imagem, como: o tamanho, quantidade de canais, modelo de cor etc. É importante lembrar, que uma imagem é uma matriz de informações ordenadas por uma quantidade de linhas e colunas. O método *cvQueryFrame* lê as propriedades do vídeo e retorna à variável.

```
32    if( capture )
33    {
34          while( true )
35          {
36                frame = cvQueryFrame( capture );
37
38                if(frame)
39                {
```

```
40                      startT=clock();
41                      frameDeteccaoPlaca( frame );
42                      endT=clock();
43                      cout << (double) (endT-startT) << endl;
44                }
45                else
46                {
47                      cout << "FIM da captura" << endl;
48                      break;
49                }
50
51                if( cvWaitKey(5) == 27 ) { break; }
52          }
53    }
```

Por final, se faz a validação da captura do vídeo, para chamada do método *frameDeteccaoPlaca*, que a cada *frame* reproduzido, inicializa as rotinas do programa de detecção da placa.

4.4.2. O pré-processamento

Conforme visto anteriormente, precisamos tratar a imagem, reduzindo assim, o ruído e melhorando as características da imagem que servirá de entrada no processo de localização da placa.

O OpenCV trabalha com inúmeras funções e métodos de pré-processamento. Veja agora algumas utilizadas no trabalho.

```
75    cvCvtColor( frame, frame_gray, CV_BGR2GRAY );
```

```
273   cvCvtColor( frame, frame_HLS, CV_BGR2HLS );
337   cvCvtColor( frame_HLS, frame, CV_HLS2BGR );
```

O método *cvCvtColor* faz diversas conversões de imagem. Nos dois casos mostrados acima, o código de conversão *CV_BGR2GRAY*, converte uma imagem com codifição de cores tanto em RGB ou BRG para escala de cinza. Já o código de conversão *CV_BGR2HLS*, converte uma imagem com codifição de cores tanto em RGB ou BRG para representação de cores em HLS (*Hue-Lightness-Saturation*), ou vice-versa, com a codifição *CV_HLS2BGR*.

É importante aplicar o código de conversão em escala de cinza, reduzindo a quantidade de canais da imagem, de três (RGB) para um (cinza), reduzindo, assim, o espaço de memória utilizado no processamento da imagem e ajudando a otimizar o método de localização de bordas, posteriormente aplicado.

A conversão em HLS, serve para inúmeras finalidades. Neste trabalho, foi utilizado algumas vezes para alterar o contraste da imagem e trabalhar também no processo de mudança de cores. Alterando o contraste da imagem, pode-se aumentar as caracteristicas que distingue os objetos na imagem. Já na modificação de cores, pode-se trabalhar com diferentes tipos de placas adotadas pelo Conselho Nacional de Trânsito (CONTRAN), figura 4.7, transformando-as em um formato único de cores.

No processo de alteração de cores, não obteve-se um sucesso desejado, muito relativo ao ruido capturado pela camera VGA. Vejamos na figura 4.8, um exemplo que altera a cor de uma caneta na cor azul para branco. Nesse processo, percebe-se os pixels alterados em alguns pontos, não cobrindo completamente a caneta, como era o esperado. Essa interferência pode ser atribuido também ao tipo de iluminação do meio.

Figure 4.- Modelos de placas adotas pelo CONTRAN.

Fonte:Autor.

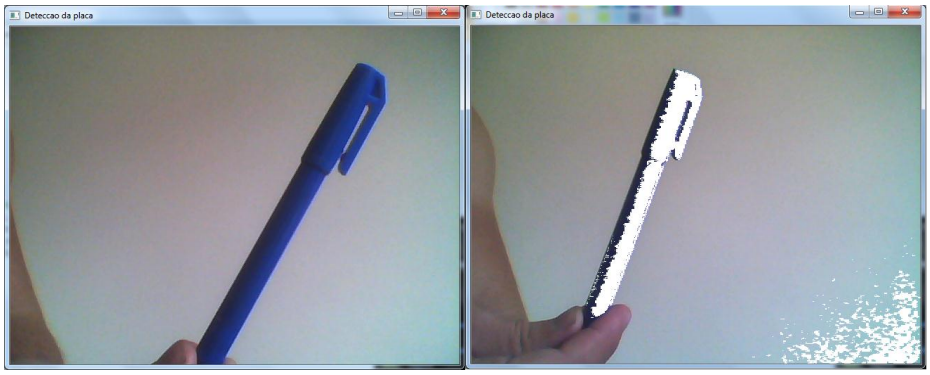

Figure 4.- Transformação de cores em HLS.

Fonte:Autor.

Seguindo assim, a ideia de diminuição de ruído explicado no tópico 3.8 desse trabalho, veremos abaixo outro método da biblioteca OpenCV utilizado para diminuição do ruido:

```
77    cvSmooth(frame_gray, frame_gray, CV_GAUSSIAN, 3,3);
```

Nesse método é utilizado o filtro Gaussiano sobre uma área de cinco por cinco centralizada em cada pixel que será varrido, com objetivo de suavizar a transição entre os valores dos pulsos, reduzindo o ruído. Este parâmetro da dimensão de área do filtro de "suavização" dependerá primordialmente da qualidade da captura da imagem. A figura 4.9, abaixo, mostra a processo de suavização.

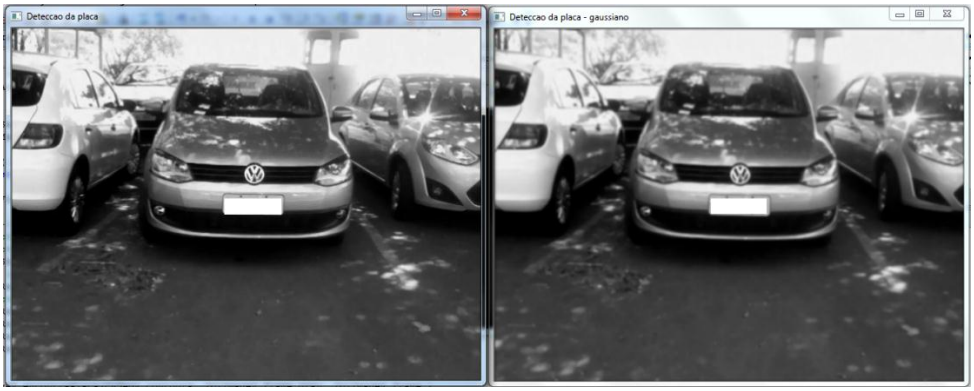

Figura 4. - Suavização Gaussiana.

Fonte: Autor.

Aqui é importante reforçar, que o processo de suavização pode causar efeitos inoportunos, como a perda de informação e o deslocamento de estruturas de feições relevantes na imagem.

4.4.3. A localização do objeto

Na etapa de localização do objeto é preciso primeiramente distinguir cada um dos objetos presentes na imagem. Desta forma, a técnica utilizada no trabalho foi a de detecção de bordas de Canny.

```
cvCanny( frame_gray, frame_canny, 50, 70, 3);
90    //cvCanny( frame, frame_canny2, 0, 70, 3 );

92    // Soma dos filtros
93    //cvAddWeighted(frame_canny, 0.5, frame_canny2, 0.5, 0, frame_merge);
```

Ao trabalhar com a função cvCanny do OpenCV, precisa-se configurar três parâmetros fundamentais: o limiar mínimo (min *threshold*), o limiar máximo (max *threshold*) e o tamanho de abertura (*aperture*). Definido os limiares, caso o pixel tenha um gradiente maior que o limiar máximo, ele é considerado como elemento de borda. Caso o pixel tenha um gradiente menor que o limiar mínimo definido, ele é descartado. Caso o gradiente do pixel esteja entre os limiares definidos (mínimo e máximo), será aceito com a condição de que o gradiente do pixel adjacente pertença a condição do limiar máximo. O parâmetro de abertura deverá ser impar (1, 3, 5 ou 7) e representa o tamanho da matriz utilizado para calcular a derivada.

Canny recomenda a razão de máximo e mínimo, na proporção de 5:1 ou 3:2, respectivamente.

No trabalho a função cvCanny foi utilizada duas vezes. Um para a imagem na escala cinza e outra para imagem colorida, para somar as duas respostas utilizando o método *cvAddWeighted* e conseguir uma maior precisão da localização do objeto. Invariavelmente isso torna o processo mais lento – muito devido aos três canais da imagem colorida (RGB).

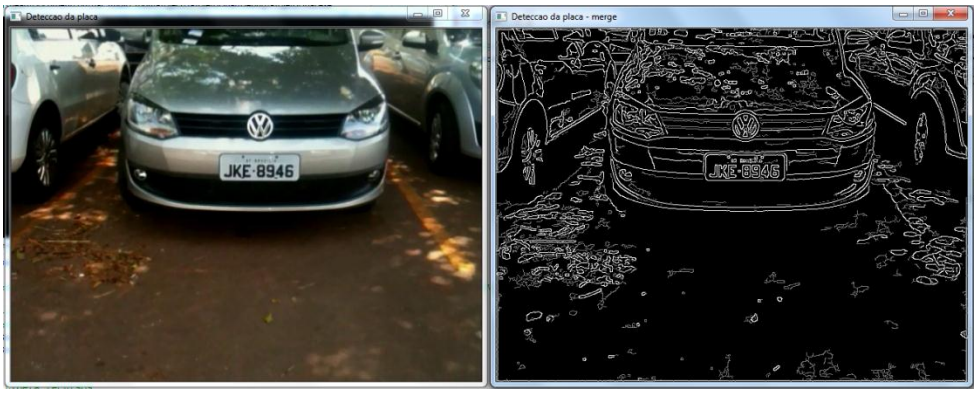

Figura 4.- Detecção de bordas de Canny.

Fonte:Autor.

```
95    // declara contorno e area na memoria
96    CvSeq* contours = 0;
97    CvMemStorage* storage = NULL;
98
99    if( storage==NULL ) storage = cvCreateMemStorage(0);
100   else cvClearMemStorage(storage);
101
102   cvFindContours( frame_canny, storage, &contours, sizeof(CvContour),
103                   CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE);
```

A variável *"contours"* foi declarada utilizando a estrutura de sequência CvSeq. Esta estrutura tem similaridades às classes de *containeres* (ou modelo de classe de contêineres) que existem em outras linguagens de programação. Essa estrutura foi construída para adicionar ou excluir um objeto, além de realizar um rápido acesso ao objeto. Essa estrutura tem alguns elementos importantes, como por exemplo, a quantidade total dos nodos da sequência. Também existem quatro elementos importantes para percorrer os nodos: *h_prev*, *h_next*, *v_prev*, e *v_next*. Estes quatros ponteiros fazem parte do modelo de estrutura chamada *CV_TREE_NODE_FIELDS*, que são utilizados para indicar elementos dentro da sequência, em vez de ligar sequências diferentes um no outro. As variáveis *h_prev* e *h_next* da estrutura podem ser utilizadas isoladamente para criar uma lista simples. Os outros dois, *v_prev* e *v_next*, podem ser usados para criar topologias mais complexas que se relacionam uns com os outros nodos. É por meio desses quatro ponteiros que o método *cvFindContours* será capaz de navegar por todos os contornos encontrados da estrutura de sequência.

Para utilizar o método *cvFindContours* é necessário também alocar um espaço reservado de memória (*cvCreateMemStorage*). É declarada então uma variável chamada *"storage"* do tipo *CvMemStorage*.

Na figura 4.11 abaixo, mostra os contornos encontrados a partir de uma imagem.

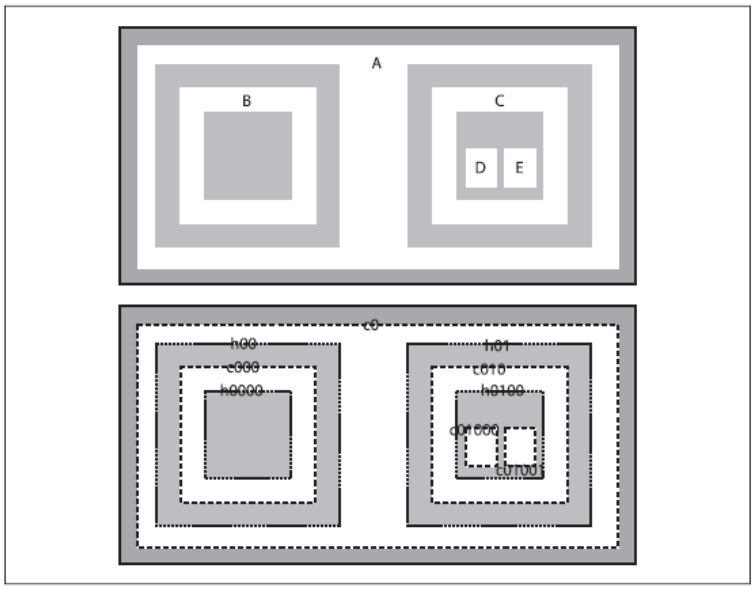

Figura 4.- Contornos encontrados pelo método cvFindContours, utilizando o modo CV_RECT_TREE e método CV_CHAIN_APPROX_SIMPLE de busca.

Fonte: Learn OpenCV.

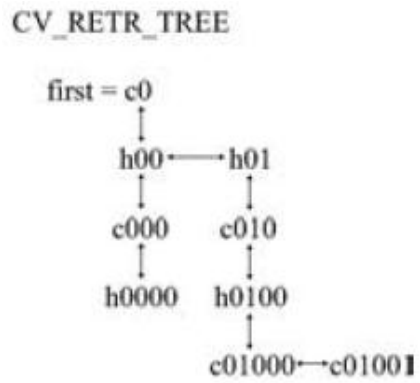

Figura 4.- Modo CV_RETR_TREE. Recupera todos os contornos e reconstrói a hierarquia completa de contornos aninhados.

Fonte: Learn OpenCV.

4.4.4. A validação

Após o processo de localização dos contornos dos objetos, a validação tem for finalidade encontrar o objeto desejado. Sendo assim, é necessário conhecer basicamente duas características:

- A forma: se é retangular (poligonal), circular etc;
- As dimensões: a altura, a largura, a profundidade e os ângulos;

A placa automotiva para carros e caminhões tem as seguintes especificações, conforme o Conselho Nacional de Trânsito (CONTRAN), figura 4.13:

Forma: polígono quadrilátero;

Dimensões: altura (a) 130 mm e largura (l) 400 mm; Ângulo das bordas: entre 80º e 110º.

Figura 4. – Especificação da Placa automotiva em mm para carros e caminhões.

Fonte: CONTRAN, 2007.

```
123   for ( ;contours!= NULL; contours = contours->h_next)
124   {
125         CvSeq* approxContour = cvApproxPoly(contours,
126                                             contours->header_size ,
127                                             contours->storage,
128                                             CV_POLY_APPROX_DP,
129                                   cvContourPerimeter(contours)*0.05,
130                                              0);
131
132
133         if (approxContour->total >=4 &&
134               fabs(cvContourArea(approxContour,CV_WHOLE_SEQ)) > 2000 &&
135               cvCheckContourConvexity(approxContour))
```

```
136         {
137               CvBox2D box = cvMinAreaRect2(approxContour);
138               double whRatio = (double)box.size.width /
139                                          box.size.height;
140               if ((!(2.7 < whRatio && whRatio < 3.4))    )  //||
141 abs(box.angle)>20 razao da largura e altura do objeto (essa proporcao é
142 para o formato brasileiro).
143            {
144                     CvSeq* child = contours->v_next;
145               if (child != NULL)
146                           encontraPlaca(child, frame_gray, frame);
147                     continue;
                  }
                /*...*/
          }
        /*...*/
      }
```

O laço *for* percorrerá todos os contornos que foram localizados na etapa anterior.

Para cada contorno percorrido pelo laço, a função *cvApproxPoly* tem por finalidade aproximar as formas curvas para poligonais. O método utilizado é o *CV_POLY_APPROX_DP*, que corresponde ao algoritmo de Douglas e Peucker (1973), que reduz a quantidade de pontos em uma curva tornando-a mais linear. E o parâmetro de precisão, que utiliza a função *cvContourPerimeter*, retorna o perímetro do contorno original, na qual, multiplicado por um fator, aumenta a precisão de aproximação. A variável *"approxcontour"* armazena o novo contorno.

Como o objetivo é encontrar contornos na forma de polígono quadrilátero é especificada a quantidade total de quatro vértices. E com intuito de reduzir a quantidade de polígonos, utiliza-se uma condição mínima para o tamanho da área aceitável. A função *cvCheckContourConvexity*, informa também que o contorno deve ser simples, não havendo convexidade (ou sem auto-interseções).

Após isso, deve-se verificar se o retângulo encontrado tem a proporção da placa automotiva. A variável *"whRatio"* significa aqui, a razão do comprimento pela altura, onde calculado, dá um valor aproximado de 3 (três). Essa variável é testada, mantendo uma variação mínima e máxima de -0.3 e +0.3 respectivamente, representando 10% (dez por cento) de variação.dd

Caso o objeto encontrado tenha as características mínimas para uma placa segundo o modelo do CONTRAN, é desenhado o retângulo na cor vermelha indicando a localização, conforme a figura 4.14.

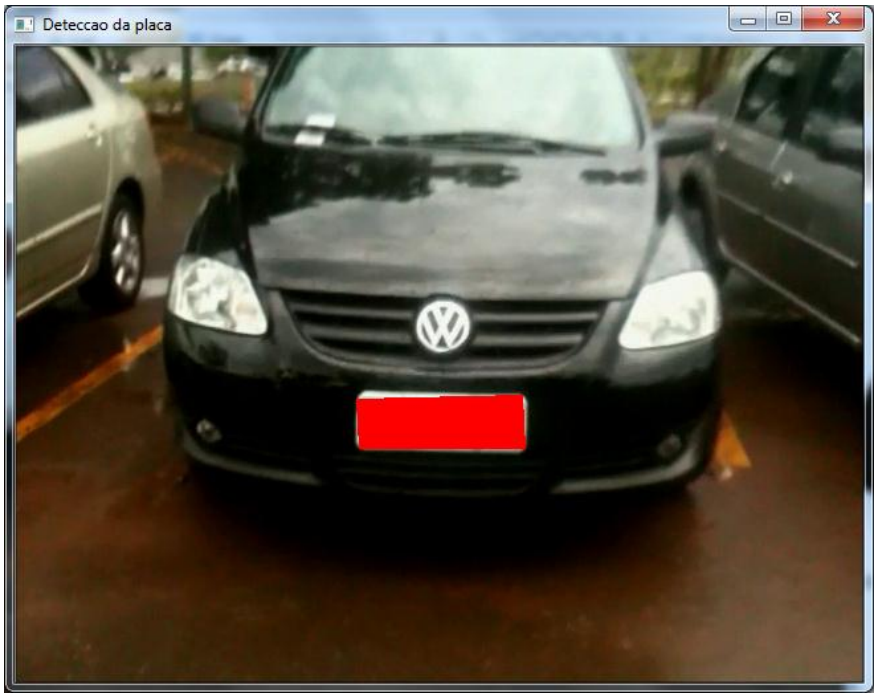

Figura 4.- Localização da placa.

Fonte: Autor.

4.4.5. A segmentação

```
182                CvRect box2 = cvBoundingRect(approxContour);
…
207                //recorta imagem
208                cvSetImageROI(frame_gray, box2);
209                cvResize(frame_gray, licenca, CV_INTER_LINEAR);
210                cvResetImageROI(frame_gray);
```

Nesta etapa, é feito o recorte da placa que foi localizada na imagem utilizando o método *cvSetImageROI* que seleciona o local informado pela variável chamada *"box2"* do tipo *CvRect* – estrutura utilizada para trabalhar com polígonos retangulares, contendo as informações de localização, largura e altura do objeto.

O método *cvResize* tem por finalidade redimensionar a imagem utilizando a técnica de interpolação bilinear, *CV_INTER_LINEAR*. O redimensionamento do recorte ampliará a placa, facilitando o processo de reconhecimento de detalhes da imagem na etapa seguinte.

Figura 4.- Placa recordada e redimensionada.

Fonte: Autor.

4.4.6. A leitura OCR

Antes de iniciar o processo de OCR é necessário realizar alguns passos, como:

- Transformar a imagem para o formato preto e branco;
- Utilizar técnicas de dilatação e erosão da imagem, para separar os caracteres que por ventura estejam próximos uns dos outros;
- Realizar o processo de treinamento dos caracteres pelo Tesseract OCR.

Para transformar a figura 4.15 em preto e branco, utilizam-se as seguintes funções:

```
212                 //trata placa comum
213                 cvThreshold(licenca,licenca,0,255,
214                                  CV_THRESH_TRUNC | CV_THRESH_OTSU);
215                 cvThreshold(licenca,licenca,0,255,
216                                 CV_THRESH_BINARY | CV_THRESH_OTSU);
217                 //trata placa colorida
218                 //cvThreshold(licenca,licenca,0,255,
219                                 CV_THRESH_BINARY_INV | CV_THRESH_OTSU);
```

A função *cvThreshold* é normalmente usado para obter dois níveis de imagem (binário), convertendo a imagem da escala de cinza para preto e branco, ou na remoção de ruídos na imagem, isto é, filtrando pixels com valores muito pequenos ou muito grandes. As limiarizações utilizadas neste trabalho foram o *CV_THRESH_TRUNC*, o *CV_THRESH_BINARY* e o *CV_THRESH_BINARY_INV*.

- *CV_THRESH_TRUNC*

$$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{threshold} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

Se um pixel estiver acima do valor limiar, será definido o valor limiar indicado. Se o pixel estiver abaixo do valor limiar, continua com o mesmo valor. É geralmente usado quando se quer destacar pixels acima do valor limiar, deixando outras originais.

- *CV_THRESH_BINARY*

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxValue} & \text{if } \texttt{src}(x,y) > \texttt{threshold} \\ 0 & \text{otherwise} \end{cases}$$

Se um pixel estiver acima do valor limiar, será definido o valor máximo indicado. Se o pixel estiver abaixo do valor limiar, será ajustado para o valor zero. Poderia ser usado para realçar pixels acima de um limiar de uma imagem, ao remover os outros.

- *CV_THRESH_BINARY_INV*

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{threshold} \\ \texttt{maxValue} & \text{otherwise} \end{cases}$$

O aposto do CV_THRESH_BINARY.

A figura 4.16, exemplifica.

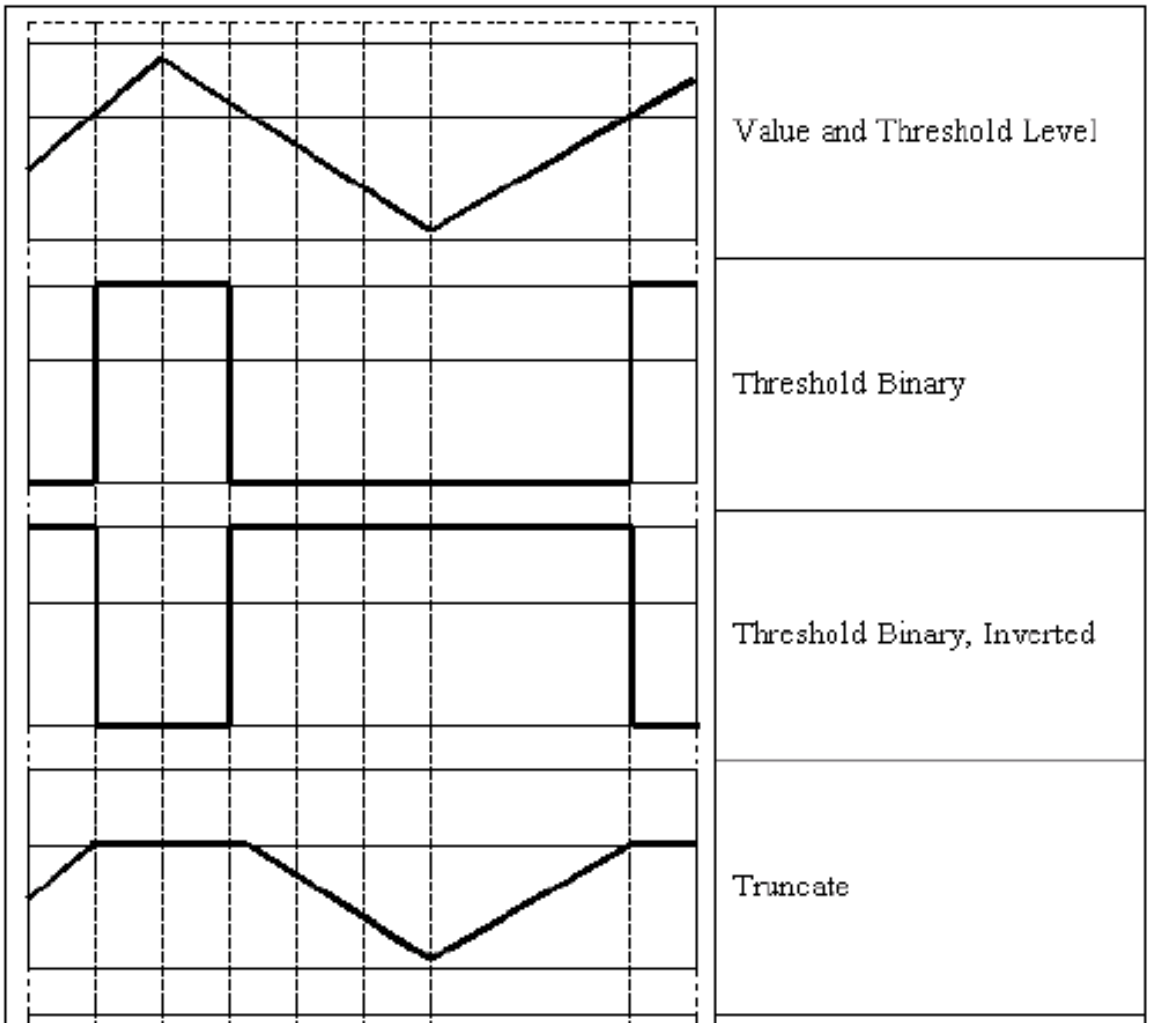

Figura 4.- Tipos de limiarizações utilizadas.

Fonte: BRADSKI and KAEHLER; Learning OpenCV, (2008).

Além disso, o valor especial CV_THRESH_OTSU pode ser combinado com um dos valores acima indicados. Neste caso, a função determina o valor do limiar ótimo usando o algoritmo de Otsu e o usa, em vez do limiar especificado. A função retorna o valor limite computadorizado. Atualmente, o método de Otsu é implementado apenas para imagens de 8 bits. O algoritmo de Otsu baseia-se nas médias dos valores dos pixels dentro da imagem.

Para a placa capturada na figura 4.15, observa-se o modelo de placa particular (fundo na cor cinza e letra na cor preta), utilizando dessa forma o tipo de limiarização binária. Já para os casos de modelo de placas de aluguel, de experiência ou de fabricantes (fundo com cor e letra na cor branca), utiliza-se o tipo de limiarização binária inversa.

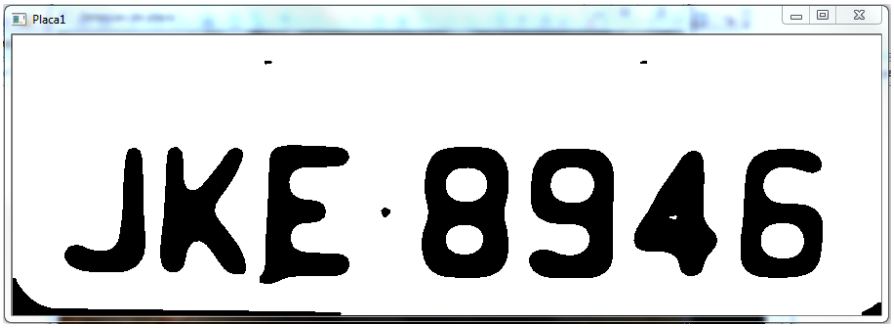

Figura 4.- Placa em preto e branco após o threshold.

Fonte: Autor.

A figura 4.17 exemplifica o resultado da transformação, de forma que os caracteres já são claramente visíveis no formato preto e branco, podendo-se melhorar com as operações de dilatação e erosão da imagem, a seguir:

```
221                cvDilate(licenca,licenca,NULL,2);
222                cvErode(licenca,licenca,NULL,2);
```

O método cvDilate irá expandir a área em branco da imagem, diminuindo as bordas dos caracteres. Dessa maneira, busca-se separar os caracteres para que não prejudique futuramente a leitura dos caracteres. Já o método cvErode trabalha de forma inversa do método cvDilate. O valor "2" utilizado especifica a quantidade de iterações repetidas a variável "licença" do tipo *IplImage*.

Figura 4.- Placa após o Dilate e Erode. Preparado para leitura.

Fonte: Autor.

A seguir, já com a imagem tratada e com os caracteres legíveis na imagem, é necessário realizar o treino da ferramenta de reconhecimento de caracteres. Feito o treino do Tesseract, a imagem poderá ser convertida em texto.

Na figura 4.19 é especificado o tipo de fonte utilizado nas placas, conforme o Conselho Nacional de Trânsito.

Figura 4.- Especificação de fonte para placas.

Fonte: CONTRAN, 2007.

Observa-se assim, uma grande diferença do tipo de fonte especificado pelo órgão regulador e o capturado na imagem. Essa diferença deve ser tratada, caso contrário o Tesseract OCR reconhecerá caracteres estranhos ou errôneos, justamente porque a biblioteca de treinamento padrão não tenha "aprendido" o tipo de letra.

Além disso, quando é realizado o processo de aprendizado da linguagem, nota-se visivelmente um aumento da performance no reconhecimento dos caracteres.

A seguir será demonstrado especificamente o processo de treinamento dos caracteres exemplificados neste trabalho, nas seguintes etapas:

- Criação das amostras dos caracteres;
- Correção dos caracteres errados encontrados pelo processo padrão de reconhecimento;
- Compilar uma nova biblioteca de caracteres.

No processo de criação das amostras, foi realizado o arquivamento das imagens geradas na etapa de preparação dos caracteres para leitura. Este passo é fundamental, pois para cada imagem gerada, diferenciam-se as formas para os mesmos caracteres encontrados. Para isso, o Tesseract utiliza um tipo de arquivo com formato *"TIFF"* - um formato de ficheiro gráfico *bitmap*.

Das imagens arquivadas no formato *"JPEG"* pelo método *saveImage* (vide apêndice A), foram extraídos manualmente os caracteres utilizando a ferramenta "Paint" (copiando e colando) no novo arquivo nomeado de "placa.arial.exp0.tif", conforme vemos na figura 4.20 o resultado final:

Figura 4.- Arquivo "placa.arial.exp0.tif". Caracteres utilizados para o treinamento da ferramenta de OCR.

Fonte:Autor.

A partir do arquivo "TIFF", com as amostras montadas, são executados os comandos de aprendizagem:

```
$ tesseract [lang].[fontname].exp[num].tif [lang].[fontname].exp[num]
batch.nochop makebox
```

No exemplo:

```
$ tesseract placa.arial.exp0.tif placa.arial.exp0 batch.nochop makebox
```

A parte mais trabalhosa no processo de aprendizagem é editar manualmente o arquivo com codificação UTF-8 gerado, "placa.arial.exp0.box". Este arquivo informa a letra encontrada e o local em pixel da localização. Nesse estágio é necessário corrigir os caracteres indevidamente informados pelo processo padrão de reconhecimento.

Feito os ajustes, por fim o arquivo terá a seguinte sequência, baseado na figura 4.20:

```
J 13 976 97 1103 0
K 114 973 201 1103 0
E 202 961 307 1104 0
- 335 1028 352 1044 0
8 380 969 468 1103 0
9 488 969 577 1102 0
. 582 957 608 965 0
4 596 969 683 1101 0
6 704 969 792 1102 0
J 12 801 95 940 0
K 113 799 201 940 0
E 202 789 308 936 0
- 338 856 354 875 0
8 381 798 469 933 0
9 491 792 575 931 0
4 596 793 681 931 0
6 703 792 786 931 0
J 12 618 95 753 0
K 112 614 198 752 0
E 203 601 307 748 0
- 337 670 352 687 0
8 382 609 468 744 0
9 487 608 573 744 0
4 593 606 679 741 0
6 701 604 783 739 0
J 12 410 94 545 0
K 114 404 200 543 0
E 204 393 305 543 0
- 338 465 348 479 0
8 383 402 466 541 0
9 488 401 573 539 0
4 595 401 680 533 0
6 703 401 785 536 0
J 9 213 93 347 0
K 112 207 197 346 0
E 200 199 304 344 0
- 337 269 349 283 0
8 380 207 466 343 0
9 486 206 574 342 0
4 593 207 680 340 0
6 702 206 786 340 0
J 13 36 98 166 0
K 114 34 202 167 0
E 203 22 308 166 0
- 336 92 353 111 0
8 382 33 468 166 0
9 487 31 573 166 0
4 578 21 679 166 0
6 702 34 787 167 0
```

Agora é necessário compilar uma nova biblioteca. Seguem os comandos que executam essa tarefa:

1 – Para cada arquivo “TIFF” treinado, executar:

```
$tesseract     [lang].[fontname].exp[num].tif     [lang].[fontname].exp[num]
nobatch box.train
```

O resultado desse comando, é um arquivo chamado [lang].[fontname].exp[num].tr , que contêm todas as propriedades de cada caractere treinado.

2 – O Tesseract precisa ainda conhecer o conjunto de caracteres de saída. Para gerar o arquivo *unicharset*, é preciso executar o extrator:

```
$ unicharset_extractor [lang].[fontname].exp[num].box ...
```

O resultado desse comando, é um arquivo chamado [lang].unicharset , representando as características de cada caractere, como o tipo (se é pontuação, um número, um alfabético etc.) e a representação em hexadecimal.

3 – A partir da versão 3.0.1 do Tesseract é preciso criar um arquivo declarando as propriedades da fonte utilizada, como: o nome da fonte, se está em negrito, itálico etc. Definiu-se então criar o um arquivo chamado *“font_properties”* conforme o modelo, a seguir:

```
<fontname> <italic> <bold> <fixed> <serif> <fraktur>
```

Exemplo:

```
Arial 0 1 0 0 0
```

No exemplo: fonte Arial, não tem itálico, tem negrito, não é fixado, não é *serif* e não é *fraktur*.

4 – É necessário juntar o(s) arquivo(s) treinado(s) clusterizando-o(s), em dois comandos: *mftraining* e *cntraining* .

Executando:

```
$ mftraining -F font_properties -U unicharset -O [lang].unicharset
[lang].[fontname].exp[num].tr ...
```

Exemplo:

```
$ mftraining -F font_properties -U unicharset -O placa.unicharset
placa.arial.exp0.tr ...
```

Este comando gerará dois arquivos: [lang].*inttemp* e [lang].*pffmtable* .

E executando:

```
$ cntraining [lang].[fontname].exp[num].tr ...
```

Exemplo:

```
$ cntraining placa.arial.exp0.tr ...
```

Este comando gera um arquivo: [lang].*normproto* .

5 – Combinando todos os arquivos, executa o comando:

```
$ combine_tessdata [lang].
```

Feito todos esses procedimentos, o treinamento nomeado “placa” está criado e pronto para ser utilizado no programa de reconhecimento de caracteres, conforme a seguir:

```
248 void tesseractOCR(IplImage *placa, int x, int y, int width, int height)
249 {
250   TessBaseAPI tess;
251   tess.SetVariable("tessedit_char_whitelist",
252                    "ABCDEFGHIJKLMNOPRSTUVWXYZ1234567890");
253   tess.Init(NULL,"placa" );
254   tess.SetImage((uchar*)placa->imageData, placa->width, placa->height,
255                 placa->nChannels, placa->widthStep);
256   tess.SetRectangle(x,y,width,height);
257   //tess.GetBoxText(0);
258   tess.Recognize(0);
259   //tess.TesseractRect
260
261   const char* out = tess.GetUTF8Text();
262   tess.End();
263
264   cout << out << endl;
265 }
```

A variável *“tess”* é declarada pelo tipo de estrutura chamada *TessBaseAPI* da biblioteca do Tesseract. O processo de leitura OCR somente inicializa quando chamado o método *Init* que especifica a base de treinamento (por padrão o inglês), que ocorreu no processo de aprendizado, nomeado anteriormente de “placa”.

O método *Recognize* inicializa o reconhecimento dos caracteres da imagem.

Já o método *getUTF8Text* imprime o texto no formado ASCII.

5. APLICAÇÃO PRÁTICA DA SOLUÇÃO PROPOSTA

Conforme explicado no escopo do trabalho no capítulo 1, o protótipo foi desenvolvido especificamente para identificar a placa veicular de automóveis e caminhões, obedecendo ao modelo adotado oficialmente pelo Conselho Nacional de Trânsito.

Sendo assim, o objetivo principal do capítulo é apresentar a operação do sistema de detecção com testes e os respectivos resultados.

5.1. Apresentação da área de Aplicação da Solução

Logicamente, por se tratar de um protótipo, o projeto não está apto para utilização comercial, pois envolveria obrigatoriamente um projeto de captura da imagem e um projeto de iluminação adequada ao local. Dessa maneira, este software é direcionado unicamente para fins acadêmicos.

Existem inúmeras áreas para aplicação desse modelo, contemplando até mesmo trabalhos mais complexos como na segurança pública, monitorando os veículos que trafegam em vias. Assim como, para aplicações mais específicas que identifique a placa de veículo em um determinado cenário ou situação.

5.2. Descrição da Aplicação da Solução

A seguir, serão mostrados os testes realizados no modelo, descrevendo o passo-a-passo, além dos equipamentos utilizados.

No processo de captura, utilizaram-se dois equipamentos de filmagem: uma webcam que filma em qualidade VGA (figura 5.1) e uma câmera de oito Mega pixel do iPhone 4S (figura 5.2). A webcam foi utilizada para testes de desenvolvimento, aferindo em tempo-real o funcionamento e a qualidade da implementação do algoritmo e da própria biblioteca de imagem utilizada. A câmera do iPhone 4S serviu para gravar as vídeos-imagens dos veículos

que estivessem na via, para posteriormente utilizá-los no software, evitando levar o computador até os ambientes de testes.

Também não foi utilizado nenhum sistema de iluminação, a não ser a “natural” do cenário, no caso: luz solar, lâmpadas fluorescentes e incandescentes. É importante mencionar aqui, que a grande maioria dos dispositivos de captura já possui um modo automático que controla a intensidade luminosa que chega ao foto sensor, colaborando na etapa de pré-processamento da imagem.

Figura 5.-Motion Eye do Sony Vaio.

Fonte: Autor.

Figura 5.- Câmera iSight é de 8 megapixels

Fonte: Autor.

Já os processos de pré-processamento, localização, validação, segmentação e leitura da imagem, foram realizadas com o apoio de um notebook com as seguintes configurações:

Processador:

- Tipo: Intel® Core™ i5-2410M;
- Velocidade: 2.3GHz with Turbo Boost Technology up to 2.90GHz;
- Cache : 3MB.

Processador Gráfico:

- Processador : AMD Radeon™ HD 6630M;
- RAM : 1GB dedicado.

Memória:

- Instalado: 4GB (2GB x 2);
- Frequência: DDR3-1333MHz.

Passo-a-passo para utilização do programa:

1 – Captura da imagem utilizando o iPhone 4S na resolução 1080p;

2 – Apontar na linha de código do programa o endereço do arquivo de vídeo;

3 – Executar o programa;

4 – O programa pode salvar as placas encontradas, que serão utilizadas futuramente para criar os arquivos de aprendizados do Tesseract OCR;

5 – O programa informa em formato texto, os caracteres das placas encontradas.

5.3. Resultados da Aplicação da Solução

Foram realizados três tipos de testes: de nível de exatidão na detecção da placa do veículo, da performance do algoritmo utilizando as bibliotecas OpenCV e Tesseract OCR, e da exatidão na conversão dos caracteres da imagem para o texto. Foram obtidos os resultados, apresentados nas tabelas 5.1, 5.2 e 5.3:

Tabela 5.- Exatidão na detecção da placa

Arquivo	Resolução	Amostras	Sucesso	Falha	Proporção(%)
teste1.wmv	480p	2	2	0	100 %
teste2.wmv	1080p	17	12	5	70,5 %
teste3.wmv	1080p	30	19	11	63.3 %

Tabela 5.- Performance do Algoritmo

Arquivo	Resolução	Tempo médio (ms)
teste1.wmv	480p	~ 60
teste2.wmv	1080p	~ 95
teste3.wmv	1080p	~105

Tabela 5.- Exatidão do Tesseract OCR

Arquivo	Resolução	Amostras	Sucesso	Falha	Proporção(%)
teste1.wmv	480p	2	2	0	100 %
teste2.wmv	1080p	12	12	0	100 %
teste3.wmv	1080p	19	19	0	100 %

Observa-se na tabela 5.1, que a solução de detecção da placa é funcional. Porém não foi previsto no início do projeto um fator muito importante que contribuiu para as falhas encontradas nos testes, refletido no resultado. Isto é, para carros nas cores branca e prata com placas do modelo particular, na cor cinza, o sistema não detectou corretamente nos testes que foram realizados (filmados). Tentou-se contornar isso, utilizando a técnica de alteração de cores da placa no modelo HLS conforme explicado no item 4.4.2 da descrição da implementação, mostrado na figura 4.8, não obtendo sucesso devido ao alto nível de ruído detectado pela câmera VGA. Também se tentou utilizar a técnica de detecção de borda de Sobel implementada (vide apêndice A), porém a técnica não é compatível com a função de detecção de contorno, precisando utilizar técnicas de limiarização, sobrecarregando ainda mais o algoritmo. Dessa maneira, pode-se inferir por comparação, que o mesmo problema ocorra para carros com tons de cores que sejam parecidas com o fundo da placa, por exemplo, carros vermelhos com o modelo de placa de aluguel, também na cor vermelho, ou carros, com tons de cores brancos e pratas que utilizam o modelo de placa de aprendizado ou oficial, com fundo na cor branca. Houve um caso particular de sucesso de detecção para um carro na cor

prata utilizando a placa do modelo particular: na placa existia a borda sutilmente preta, que encobria o material de alumínio que normalmente não é pintada.

Na performance do algoritmo, verifica-se que o modelo rodou no máximo de 16,6 frames/segundo e um mínimo de 9,5 frames/segundo, considerado inferior ao previsto para utilizar em tempo real. No caso, a configuração do notebook utilizado neste projeto não mostrou desempenho suficiente para a aplicação em tempo real. Uma maneira de otimizar o processo é criando um algoritmo que melhor utilize a característica de multiprocessamento do processador.

Utilizando o Tesseract OCR obteve-se um bom resultado, desde que seja realizado previamente o treinamento dos caracteres baseado no processo de segmentação e leitura OCR, esclarecidos nos itens 4.4.5 e 4.4.6 desse trabalho. No mesmo item 4.4.6 é importante destacar que o processo de aprendizagem dos caracteres ajuda na performance da biblioteca.

Na tabela 5.4 a seguir, é ilustrado a performance dos sistemas de detecção de placas automotivas, encontradas em algumas literaturas.

Tabela 5. - Performances dos sistemas de detecção de placas automotivas, nas literaturas.

Referencial	**Taxa de reconhecimento** **DP: detecção da placa** **RC: reconhecimento dos caracteres** **T: sucesso total**	**Plataforma e processador**	**Tempo (seg)**	**Conhecimento cientifico envolvido no método** **P: método de detecção da placa** **R: método de reconhecimento dos caracteres**
[2]	100% (DP) 92.5% (RC)	Matlab 6.0, P II , 300MHz	~ 4	P: analise de componentes conectados. R: 2 PNNs (um para letras e outra para números). Teste: 40 imagens.
[3]	80.4% (T)	AMD K6, 350MHZ	~ 5	P: Morfologia matemática. R: Hausdoff distance.
[4]	99.6% (DP)	P IV, 1.7GHz	~0.1	P: Morfologia e estatística de borda. Testes: ~10000 imagens.
[6]	94.2% (DP), 98,6% (RC)	C++, P II, 300MHz	-	P: Magnitude de gradientes verticais e componentes geométricos. Testes: 104 imagens.
[7]	92.3% (DP), 95.7% (RC)	-	~1.6	P: Tecnica baseada em processamento e referencia de imagens. Motocicletas também foram identificadas.
[8]	93.1% (T)	-	-	P: Processamento em escala de cinza. R: projeção dinâmica.
[9]	90% (T)	P III, 1GHz	~0.3	P: Combinação de informações de cores e

				formas das placas.
[10]	97% (T no dia), 90% (T a noite)	-	0.1 – 1.5	Unidade de Computação de alta performance para reconhecimento de placa.
[11]	91% (T)	RS/6000 mod. 560	~1.1	P: Informação de contraste. R: Template Matching. Teste: 3200 imagens.
[12]	80% (T)	Intel 486/66MHz	~15	P: Processamento em escala de cinza. R: Feedforward ANN.
[13]	89% (DP)	C++, P III 800MHz	~1.3	P: Suport Vector Machine (SVM), continuous adaotive meanshift algorithm (CAMShift) in color measurements.
[14]	98.5% (DP)	AMD Athlon 1.2	~34	P: Um classificador supervisionado treinados sobre as características de textura. R: "kd-tree" data and "approximate nearest neighbour". Teste: 131 imagens.
[15]	95% (T)	Visual C++, P IV	2	P: método de múltiplo entrelaçamento. R: ANN.
[16]	97% (DP)	Workstation SG – INDIGO 2	2	P: Regras de lógica Fuzzy. Teste:100 imagens.
[17]	75.4% (DP), 98.5% (RC)	-	-	P: Logica Fuzzy. R: Rede neural. Teste 10000 imagens.
[18]	97.6% (DP) 95.6% (RC)	P IV, 1.6 GHz	~2.5	P: Regras de lógica Fuzzy. R: Self Organized Neural Network. Teste: 1065 imagens de 71 veiculos.
[19]	98% (DP)	P IV, 1.4 GHz	3.12	P: Gabor Transform and Vector Quantization. Executa somente a localização da placa e a segmentação de caracteres (94.2%).
[20]	-	P III, 600 MHz	0.15	P: Sub-machine-code Genetic Programming.
[21]	80.6% (DP) average	P III, 500 MHz	0.18	P: Real-coded genetic algorithm (RGA).
[22]	-	-	-	P: Markov Random Field and Genetic Algorithm in video sequence.
[23]	98.8% (DP), 95.2% (RC)	Visual C++, P IV 1.4 GHz	0.65 (plates) 0.1 (per character)	P: Hough transform. R: Hidden Markov Model.
[24]	95.6% (DP)	-	-	P: AdaBoost (Adaptive HBoostingH) meta - algorithm.
[25]	92.4% (DP)	Borland C++, P IV 1.6GHz	-	P: Wavelet transform.
[26]	93.6% (DP)	Visual C++, AMD Athlon 900Mhz	~1.3	P: Generalized Symmetry Transform and Image Warping.
[27]	85% (DP)	-	-	P: Discrete Time Cellular Neural Networks. R: Multi-Layer Perceptron.
[28]	85% (DP)	Pulse-Coupled NN Processor	-	P: Pulse Coupled Neural Network (PCNN).
[29]	100% (DP), 94.7% (RC)	Visual C++, P II	~1	P: 2 Time-Delay Neural networks (TDNNs). R: support vector machine (SVM)-based character recognizer. Teste:

				1000 imagens.
[30]	98% (DP)	C++, P I, 200 MHz	0.2	P: Vector Quantization (VQ) for picture compression and effective license plate location.
[31]	97.0-99.3%	-	-	P: Sensing system with a wide dynamic range. Teste: 1000 imagens.
[32]	84.5% (RC segmentation)	-	-	Character segmentation: Feature vector extraction and mathematical morphology.
[33]	96.8% (RC)	Visual C++, P IV 2.4GHz	-	P: Block based local adaptive binarization method.
[34]	88% (DP), 95.7% (RC)	-	-	P: Connected components. R: Hidden Markov Models.
[36]	99%	-	-	R: Character recognition-PNN. Teste: 680 imagens.
[37]	-	-	~1.5	P:Gradientanalysis,R:Templatematching

Comparando as técnicas utilizadas por alguns autores, este trabalho deu um foco maior à literatura desenvolvida pelos autores Bai Hongliang e Liu Changping (2004), focando na localização de bordas e características morfológicas da placa.

5.4. Avaliação Global da Solução

O modelo de processamento de imagens empregado nesse trabalho obteve resultados positivos em partes, conforme explicado no item anterior. Trabalhar com a visão computacional exige um trabalho muito minucioso e conhecimento de diversas áreas que a compreende.

Abaixo são listadas algumas dificuldades encontradas ao longo das execuções deste trabalho que afetaram diretamente o processo de detecção da placa e leitura dos caracteres:

- **Detecção da placa**: utilização de vários métodos de segmentação de bordas. Principalmente devido ao problema das cores dos automóveis e a cor do fundo das placas.
- **Tratamento da imagem da placa**: utilizando os processos de dilatação e erosão, geram alterações nos formatos dos caracteres, aumentando ou diminuindo as bordas. Isso pode gerar alteração nos padrões dos caracteres já catalogados no processo de aprendizado.
- **Iluminação insuficiente do meio**: geram caracteres imprevistos e não catalogados pelo banco de aprendizado

- **Velocidade inadequada de gravação da câmera**: com a quantidade frames por segundo insuficiente, é gerado borrões nas imagens, efeito “blur”.
- Outro fator que deve ser levado em consideração para o projeto ter eficácia é a velocidade de processamento do computador. A princípio, o sistema deve ser ajustado para trabalhar com uma velocidade de processamento equivalente a velocidade de gravação da câmera, permitindo a detecção das placas em tempo real. Caso não seja possível, procura-se trabalhar com o processamento, adequando à velocidade para acima da máxima permitida pela via. Por exemplo, se o sistema captura a placa numa distância de 30 metros entre os pontos de mínima e máxima detecção e a velocidade da via seja de 30m/s (108Km/h), calcula-se que em um segundo que o carro percorra os 30 metros, o sistema deva capturar no mínimo um quadro perfeito.

6. CONCLUSÃO

6.1. Conclusões

A utilização da visão computacional com a finalidade de aproximá-la à visão humana mostrou-se realmente um desafio ao utilizá-la no desenvolvimento do projeto de detecção e reconhecimento de placa automotiva. Diversas áreas de estudo foram exigidas para elaboração deste trabalho, cuja interdisciplinaridade agrega diversas técnicas matemáticas e tecnológicas. As disciplinas como física, métodos matemáticos, programação e processamento digital de imagem, são exemplos que compõem a parte teórica. A câmera que filma em um formato e em uma determinada resolução e o computador que processa e armazena dados, foram aqui consideradas como tecnologias fins para obtenção de informações. Assim, para a construção do software de detecção e reconhecimento de placas para carros e caminhões, aplicou-se o modelo proposto na figura 4.1, baseado em um projeto de visão computacional utilizando as bibliotecas *open source*, OpenCV e o Tesseract OCR.

No decorrer da fase de desenvolvimento do software, não faltaram problemas para serem solucionados. Por exemplo, no processo de captura da imagem não foi utilizada uma câmera adequada que trouxesse qualidade à imagem, além da parte de iluminação – sem a luz e um equipamento eficaz, outras fases do modelo são afetadas diretamente. Dessa maneira, a fase de pré-processamento exigiu um tratamento das imagens, aplicando filtros para diminuição de ruídos. No processo de localização, empregou-se a técnica de segmentação de bordas que melhor oferecesse performance ao algoritmo. No processo de validação, foram ajustados parâmetros para conferir maior precisão para a detecção da placa automotiva. Já no processo da leitura e reconhecimento de caracteres, destacou-se a necessidade prévia do aprendizado dos caracteres, por um método bastante trabalhoso atualmente.

Dessa maneira, o objetivo principal do projeto foi cumprido para a maioria dos casos de testes, salvo nos casos particulares apresentados na sessão dos resultados da aplicação do capítulo da aplicação prática, onde foram encontrados casos em que o método de detecção de borda é falha e necessita uma atenção especial nos equipamentos de captura.

Para aperfeiçoar o trabalho de detecção de objeto por bordas, tentou-se utilizar também em conjunto outras técnicas de segmentação, como a de Sobel e a wavelet 2D, porém

os resultados não foram expressivos, visto o curto tempo de pesquisa e desenvolvimento do projeto. Em complemento, o projeto de um sistema de iluminação eficaz em conjunto ao sistema ótico e a bons equipamentos para processamento de dados, trariam excelentes resultados para o trabalho.

6.2. Sugestões para Trabalhos Futuros

Foram referenciadas ao longo deste trabalho, inúmeras técnicas para detecção e reconhecimento de placa automotiva, que poderão ser exploradas em complemento a técnica de detecção de bordas apresentadas. Dessa maneira, este trabalho pode ser complementado utilizando outras técnicas de bordas como de Sobel ou Wavelet. Além disso, podem ser criadas interfaces que tragam facilidades para ajustes às variáveis do programa, agilizando a configuração dos métodos empregados no sistema, adaptando-os consequentemente ao meio ambiente utilizado. Ou, integrando este sistema a outros que controlam e/ou monitoram acessos de veículos, exemplo: *shopping*.

Outra metodologia para localização de objetos que tem se expandido é utilizando o treinamento de imagens, como *haartraining* e o *Speeded-Up Robust Features* (*SURF)* que o OpenCV já disponibiliza. Essas técnicas utilizam critérios de repetição das características do objeto treinado sob diferentes cenários que ocorrem, construindo assim, um “banco de informações”.

REFERÊNCIAS

CANNY, J.; A Computational Approach To Edge Detection, IEEE Trans. Pattern Analysis and Machine Intelligence, 1986.

BRADSKI, KAEHLER Adrian; Learning OpenCV, (2008).

DAVIES, E. Roy. Machine Vision: Teoria, Algoritmos, Practicalities. Morgan Kaufmann, 2005.

DESCHAMPS, Fernando. Contribuições para o Desenvolvimento de um Sistema de Visão aplicado ao Monitoramento do Desgaste de Ferramentas de Corte - O Sistema ToolSpy. Dissertação de mestrado, Universidade Federal de Santa Catarina - Programa de Pós Graduação em Engenharia Elétrica, Florianópolis - SC - Brasil, 2004.

ERHARDT-FERRON. Theory and Applications of Digital Image Processing. University of Applied Sciences Offenburg, 1 edition, 2000.

FORSYTH, J. Ponce, Computer Vision: A Modern Approach, Englewood Cliff s, NJ: Prentice-Hall, 2003.

GONZALEZ, RICHARD E. Woods. Digital Image Processing. Pearson Education, 2 edition, 2002.

GRIOT, Melles. The Practical Application of Light. Estados Unidos, 1999.

HUBEL, WIESEL; Receptive Fields, binocular interaction and functional architecture in the cat´s visual cortex. J. Physiol, 1962.

JÄHNE, HAUSSECKER, PETER Geissler. Handbook of Computer Vision and Applications - Sensors and Imaging, volume 1. Academic Press, San Diego - CA, 1 edition, 1999.

JAIN, A. K. Fundamentals of digital image processing. Englewood Cliffs, NJ: Prentice Hall, 1989.

JAIN, KASTURI, BRIAN G. Schunck. Machine Vision. McGraw Hill, 1 edition, 1995.

LARSON, H. J.; SHUBERT, B. O. Probabilistic Models in engineering sciences 1979, v.1

MARQUES FILHO, Ogê; VIEIRA NETO, Hugo. Processamento Digital de Imagens, Rio de Janeiro:Brasport, 1999.

MORRIS, Tim. Computer Vision and Image Processing. Palgrave Macmillan, 2004.

PRATT, W. K. Digital image processing. New York: John Wiley and Sons, 1991.

SHAPIRO, GEORGE Stockman. Computer Vision. Prentice Hall, 2001.

TRUCCO, A. Verri, Introductory Techniques for 3-D Computer Vision, Englewood Cliff s, NJ: Prentice-Hall, 1998.

ZIOU D., TABBONE S. Edge Detection Techniques - An Overview. International Journal of Pattern Recognition and Image Analysis,Vol. 8, No. 4, pp. 537-559, 1998. also Technical Report, No. 195, Dept. Math. Et Informatique, Université de Sherbrooke, 41p, 1997.

Links externos:

TheComputerVision é uma comunidade para os pesquisadores de visão por computador e profissionais para compartilhar, discutir, conectar e colaborar.

Disponível em: The Computer Vision <http://cvisioncentral.com/>

Acesso em: 10 de nov. 2012.

Uma boa fonte para códigos-fonte, pacotes de software, conjuntos de dados, etc relacionada à visão de computacional.

Disponível em: Computer Vision Online <http://www.computervisiononline.com/>

Acesso em: 10 de nov. 2012.

Bob Fisher's Compendium of Computer Vision.

Disponível em: CVonline < http://homepages.inf.ed.ac.uk/rbf/CVonline/>

Acesso em: 10 de nov. 2012.

Image Processing - Grupo de Pesquisa

Disponível em: IPRG < http://iprg.co.in/index.php>

Acesso em: 10 de nov. 2012.

Referencial da tabela 5.4:

[2] C. Anagnostopoulos, E. Kayafas, V. Loumos, “Digital image processing and neural networks for vehicle license plate identification”, Journal of Electrical Engineering, vol. 1, No.2, p.p. 2-7, 2000. [Online]. Available: http://www.medialab.ntua.gr/people/canag/journals.php.

[3] F. Martín, M. García, J. L. Alba, (2002, Jun.), New Methods for Automatic Reading of VLP's (Vehicle License Plates), presented at IASTED International Conference Signal Processing, Pattern Recognition, and Applications, SPPRA 2002. [Online]. Available: http://www.gpi.tsc.uvigo.es/pub/papers/sppra02.pdf.

[4] Bai Hongliang and Liu Changping, “A hybrid License Plate Extraction Method Based On Edge Statistics and Morphology”, in proc. Conf. on Pattern Recognition (ICPR), 2004, pp. 831-834.

[5] Danian Zheng, Yannan Zhao, Jiaxin Wang, “An efficient method of license plate location”, Pattern Recognition Letters (2005), Volume 26, Issue 15, November 2005, Pages 2431-2438.

[6] SherrZheng Wang and HsMian Lee, “Detection and Recognition of License Plate Characters with Different Appearences”, in proc. Conf. Intelligent Transportation Systems, vol. 2, 2003, pp. 979 - 984.

[7] Hsi-Jian Lee, Si-Yuan Chen and Shen-Zheng Wang, “Extraction and Recognition of License Plates of Motorcycles and Vehicles on Highways”, in proc Int. Conf. on Pattern Recognition (ICPR), 2004, pp. 356 – 359.

[8] Tsang-Hong Wang, Feng-Chou Ni, Keh-Tsong Li, Yon-Ping Chen, “Robust License Plate Recognition based on Dynamic Projection Warping”, in proc. IEEE Int. Conf. on Networking, Sensing & Control, 2004, pp.784-788.

[9] Xifan Shi, Weizhong Zhao, and Yonghang Shen, “Automatic License Plate Recognition System Based on Color Image Processing”, Lecture Notes on Computer Science 3483, O. Gervasi et al., Ed. Springer-Verlag, 2005, pp. 1159 – 1168.

[10] Dai Yan, Ma Hongqing, Liu Jilin and Li Langang, “A High Performance License Plate Recognition System Based on the Web Technique”, in proc. Conf. Intelligent Transportation Systems, 2001, pp. 325-329.

[11] P. Comelli, P. Ferragina, M.N. Granieri, F. Stabile, “Optical recognition of motor vehicle license plates”, IEEE Trans. On Vehicular Technology, vol. 44, No. 4, pp. 790-799, 1995.

[12] S. Draghici, “A neural network based artificial vision system for license plate recognition”, International Journal of Neural Systems, vol. 8, no. 1, pp. 113-126, 1997.

[13] Kwang In Kim, Keechul Jung, and Jin Hyung Kim , “Color Texture-Based Object Detection: An Application to License Plate Localization”, Lecture Notes on Computer Science 2388, S.-W. Lee and A. Verri, Ed., Springer-Verlag, pp. 293-309.

[14] J. Cano and J. C. Perez-Cortes, “Vehicle License Plate Segmentation in Natural Images”, Lecture Notes on Computer Science 2652, F.J. Perales et al., Ed. Springer-Verlag, 2003, pp. 142–149.

[15] A.Broumandnia, M.Fathy, (2005, Dec.) , Application of pattern recognition for Farsi license plate recognition, to be presented at ICGST International Conference on Graphics, Vision and Image Processing (GVIP-05). [Online]. Available: http://www.icgst.com/gvip /v2/P1150439001.pdf.

[16] N. Zimic, J. Ficzko, M. Mraz, J. Virant, “The Fuzzy Logic Approach to the Car Number Plate Locating Problem”, in proc. Intelligent Information Systems (IIS’97), pp.227-230, 1997.

[17] J. A. G. Nijhuis, M. H. ter Brugge, K.A. Helmholt, J.P.W. Pluim, L. Spaanenburg,R.S. Venema, M.A. Westenberg, “Car License Plate Recognition with Neural Networks and Fuzzy Logic”, in proc.. IEEE International Conference on Neural Networks, vol.5, 1995, pp. 2232-2236.

[18] Shyang-Lih Chang, Li-Shien Chen, Yun-Chung Chung, and Sei-Wan Chen, “Automatic License Plate Recognition”, IEEE Trans. on Intelligent Transportation Systems, Vol. 5, No. 1, pp. 42-53, 2004.

[19] Fatih Kahraman, Binnur Kurt, and Muhittin Gökmen , “License Plate Character Segmentation Based on the Gabor Transform and Vector Quantization”, Lecture Notes on Computer Science 2869, A. Yazici and C. Sener, Ed., Springer-Verlag, 2003, pp. 381-388.

[20] G. Adorni, S. Cagnoni, M. Gori, and M. Mordonini, “Access control system with neuro-fuzzy supervision”, in proc. Conf. Intelligent Transportation Systems, 2001, pp. 472-477.

[21] Seiki Yoshimori, Yasue Mitsukura, Minoru Fukumi, and Norio Akamatsu, “License Plate Detection Using Hereditary Threshold Determine Method”, Lecture Notes in Artificial

Intelligence 2773, V. Palade, R.J. Howlett, and L.C. Jain, Ed. Springer-Verlag, 2003, pp. 585–593.

[22] Yuntao Cui, Qian Huang, “Extracting characters of license plates from video sequences”, Machine Vision and Applications, vol. 10, pp. 308-320, 1998.

[23] Tran Duc Duan, Tran Le Hong Du, Tran Vinh Phuoc, Nguyen Viet Hoang, “Building an Automatic Vehicle License-Plate Recognition System”, in proc. Intl. Conf. in Computer Science (RIVF), 2005, pp. 59-63.

[24] Louka Dlagnekov, (2004, Mar.), License Plate Detection Using AdaBoost, Computer Science & Engineering Dpt., San Diego, La Jolla. [Online]. Available: http://www.cse.ucsd.edu/classes/ fa04/cse252c/projects/louka.pdf.

[25] Ching-Tang Hsieh, Yu-Shan Juan, Kuo-Ming Hung, “Multiple License Plate Detection for Complex Background”, in Proc. Int Conf. on Advanced Information Networking and Applications (AINA), vol. 2, 2005, pp. 389-392.

[26] Dong-Su Kim, Sung-I1 Chien, “Automatic Car License Plate Extraction Using Modified Generalized Symmetry Transform and Image Warping”, in proc. Int. Symposium on Industrial Electronics (ISIE), 2001 , pp. 2022-2027.

[27] M. H. ter Brugge, J. H. Stevens, J.A. G. Nijhuis, L. Spaanenburg, “License Plate Recognition Using DTCNNs”, in proc. IEEE lnt. Workshop on Cellular Neural Networks and their Applications, 1998, pp. 212-217.

[28] Mario I. Chacon M., Alejandro Zimmerman S., “License Plate Location Based on a Dynamic PCNN Scheme”, in proc. Int. Joint Conf. on Neural Networks, vol. 2, 2003, pp. 1195 – 1200.

[29] K.K.Kim K.I.Kim J.B.Kim H. J.Kim, “Learning-Based Approach, for License Plate Recognition”, in proc. IEEE Signal Processing Society Workshop, Neural Networks for Signal Processing, Vol. 2, 2000, pp. 614 - 623.

[30] R. Zunino, S. Rovetta, “Vector quantization for license-plate location and image coding”, IEEE Trans. on Industrial Electronics, February, vol.47, No.1, pp.159-167, 2000.

[31] T. Naito, T. Tsukada, K. Yamada, K. Kozuka, S. Yamamoto, “Robust license-plate recognition method for passing vehicles under outside environment”, IEEE Trans. on Vehicular Technology, Vol. 49, No.6, pp. 2309-2319, 2000.

[32] Shigueo Nomura, Keiji Yamanaka, Osamu Katai, Hiroshi Kawakami, Takayuki Shiose, “A novel adaptive morphological approach for degraded character image segmentation”, Pattern Recognition, Vol. 38, Issue 11, November 2005, pp.1961-1975.

[33] Byeong Rae Lee, Kyungsoo Park, Hyunchul Kang, Haksoo Kim, and Chungkyue Kim, “Adaptive Local Binarization Method for Recognition of Vehicle License Plates”, Lecture Notes on Computer Science 3322 R. Klette and J. Žunić, Ed. Springer-Verlag, 2004, pp. 646–655.

[34] David Llorens, Andres Marzal, Vicente Palazon, and Juan M. Vilar , “Car License Plates Extraction and Recognition Based on Connected Components Analysis and HMM Decoding”, Lecture Notes on Computer Science 3522, J.S. Marques et al. , Ed. Springer-Verlag, 2005, pp. 571–578.

[35] C. Anagnostopoulos, T. Alexandropoulos, S. Boutas, V. Loumos, E. Kayafas, “A template-guided approach to vehicle surveillance and access control”, in proc. IEEE Conf. on Advanced Video and Signal Based Surveillance, 2005., pp. 534 – 539.

[36] Yafeng Hu, Feng Zhu, and Xianda Zhang, “A Novel Approach for License Plate Recognition Using Subspace Projection and Probabilistic Neural Network”, Lecture Notes on Computer Science 3497, J. Wang, X. Liao, and Z. Yi, Ed. Springer-Verlag 2005, LNCS 3497, pp. 216–221.

[37] Yo-Ping Huang, Shi-Yong Lai and Wei-Po Chuang, “A Template-Based Model for License Plate Recognition”, in proc. IEEE Int. Conf. on Networking, Sensing & Control, 2004, pp. 737-742.

APÊNDICE

APENDICE A – Algoritmo do projeto – este algoritmo foi comentado no item 4.4, no descritivo da implementação, assim como em alguns trechos do código.

```
1 #include <stdio.h>
2 #include <highgui.h>
3 #include <cv.h>
4 #include <baseapi.h>
5
6
7 using namespace cv;
8 using namespace std;
9 using namespace tesseract;
10
11 void frameDeteccaoPlaca( IplImage* frame );
12 IplImage* encontraPlaca(CvSeq* contours, IplImage* frame_gray,
13                         IplImage* frame );
14 void trataImagemHLS(IplImage* frame);
15 void tesseractOCR(IplImage *placa, int x, int y, int width, int height);
16 void saveImg(IplImage *img);
17 double angle( CvPoint* pt1, CvPoint* pt2, CvPoint* pt0 );
18 IplImage* sobelImg(IplImage* frame);
19
20 int main(int argc, char** argv)
21 {
22    IplImage *frame;
23
24    CvCapture *capture = NULL;
25
26    clock_t startT, endT;
27
28    capture = cvCreateFileCapture( "teste1.wmv" );
29
30    //capture = cvCaptureFromCAM( -1 );
31
32    if( capture )
33    {
34          while( true )
35          {
36                frame = cvQueryFrame( capture );
37                if( !frame ) break;
38                if( !frame->imageSize==0 )
39                {
40                      startT=clock();
41                      frameDeteccaoPlaca( frame );
42                      endT=clock();
43                      cout << (double) (endT-startT) << endl;
44                }
45                else
46                {
47                      cout << "Problema na webcam! Sem captura" << endl;
48                      break;
49                }
50
```

```
51                 if( cvWaitKey(5) == 27 ) { break; }
52             }
53     }
54     cvReleaseCapture( &capture );
55     cvDestroyWindow("Deteccao da placa");
56 }
57
58 void frameDeteccaoPlaca( IplImage* frame )
59 {
60     //trata imagem utilizando o tipo HLS (HUE, Light, Saturation)
61     //trataImagemHLS(frame);
62
63     //IplImage* frame_gray_s = cvCreateImage( cvSize( frame->width,frame-
64 >height ), IPL_DEPTH_8U, 1); // gray smooth
65     IplImage* frame_gray = cvCreateImage( cvSize( frame->width,
66                                         frame->height ), IPL_DEPTH_8U, 1);
67     IplImage* frame_canny = cvCreateImage( cvSize( frame->width,
68                                         frame->height ), IPL_DEPTH_8U, 1);
69     IplImage* frame_canny2 = cvCreateImage( cvSize( frame->width,
70                                         frame->height ), IPL_DEPTH_8U, 1);
71     IplImage* frame_merge = cvCreateImage( cvSize( frame->width,
72                                         frame->height ), IPL_DEPTH_8U, 1);
73
74     // Escala cinza
75     cvCvtColor( frame, frame_gray, CV_BGR2GRAY );
76     // Retira ruidos
77     cvSmooth(frame_gray, frame_gray, CV_GAUSSIAN, 3,3);
78
79     //Sobel
80     //IplImage* frame_sobel = cvCreateImage( cvSize( frame->width,
81                                        frame->height ), IPL_DEPTH_8U, 1);
82     //frame_sobel = sobelImg(frame_gray);
83
84     //cvConvertScaleAbs(frame_sobel, frame_sobel);
85
86     //cvShowImage("Sobel", frame_sobel);
87
88     // Filtros canny cinza e canny colorida
89     cvCanny( frame_gray, frame_canny, 0, 255, 3);
90     //cvCanny( frame, frame_canny2, 0, 70, 3 );
91
92     // Soma dos filtros
93     //cvAddWeighted(frame_canny, 0.5, frame_canny2, 0.5, 0, frame_merge);
94
95     // declara contorno e area na memoria
96     CvSeq* contours = 0;
97     CvMemStorage* storage = NULL;
98
99     if( storage==NULL ) storage = cvCreateMemStorage(0);
100    else cvClearMemStorage(storage);
101
102    cvFindContours( frame_canny, storage, &contours, sizeof(CvContour),
103                      CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE);
104
105    //-- Mostra imagem
106    cvShowImage("Deteccao da placa",
107                    encontraPlaca(contours,frame_gray,frame) );
108    //cvShowImage("Deteccao da placa", frame);
109
110    // Libera recursos
111    cvReleaseImage(&frame_gray);
```

```
112   //cvReleaseImage(&frame_gray_s);
113   cvReleaseImage(&frame_canny);
114   cvReleaseImage(&frame_canny2);
115   cvReleaseImage(&frame_merge);
116   cvReleaseMemStorage(&storage);
117 }
118
119 IplImage* encontraPlaca(CvSeq* contours, IplImage* frame_gray,
120                         IplImage* frame )
121 {
122
123   for ( ;contours!= NULL; contours = contours->h_next)
124   {
125         CvSeq* approxContour = cvApproxPoly(contours,
126                                             contours->header_size ,
127                                             contours->storage,
128                                             CV_POLY_APPROX_DP,
129                                    cvContourPerimeter(contours)*0.05,
130                                               0);
131
132
133         if (approxContour->total >=4 &&
134               fabs(cvContourArea(approxContour,CV_WHOLE_SEQ)) > 2000 &&
135               cvCheckContourConvexity(approxContour))
136         {
137               CvBox2D box = cvMinAreaRect2(approxContour);
138               double whRatio = (double)box.size.width /
139                                      box.size.height;
140               if ((!(2.7 < whRatio && whRatio < 3.4))    )  //||
141 abs(box.angle)>20 razao da largura e altura do objeto (essa proporcao é
142 para o formato brasileiro).
143            {
144                     CvSeq* child = contours->v_next;
145               if (child != NULL)
146                           encontraPlaca(child, frame_gray, frame);
147                     continue;
148            }
149
150               double s = 0;
151            for( int i = 0; i <= 4; i++ )
152            {
153                 // find minimum angle between joint
154                 // edges (maximum of cosine)
155                     if( i >= 2 )
156                 {
157                           double t = fabs(angle(
158                            (CvPoint*)cvGetSeqElem(approxContour, i),
159                            (CvPoint*)cvGetSeqElem(approxContour, i-2),
160                            (CvPoint*)cvGetSeqElem(approxContour, i-1)));
161
162                           s = s > t ? s : t;
163                 }
164            }
165
166            // if cosines of all angles are small
167            // (all angles are ~90 degree) then write quandrange
168            // vertices to resultant sequence
169            if( !(s < 0.3) )
170               {
171                     CvSeq* child = contours->v_next;
172                     if (child != NULL)
```

```
173                         encontraPlaca(child, frame_gray, frame);
174                   continue;
175             }
176
177             // desenha retangulo vermelho
178             cvDrawContours( frame , approxContour, CV_RGB(255,0,0),
179                               CV_RGB(255,0,0), -1, CV_FILLED, 8 );
180
181             //amplia imagem
182             CvRect box2 = cvBoundingRect(approxContour);
183             int m_width = (int)9000/box2.width; //multiplicador de
184 largura
185             int height_size, width_size, width_char, height_char;
186
187             if (m_width < 1)
188             {
189                   height_size = 300;
190                   width_size = 900;
191                   height_char = 159; //53%
192                   width_char = 102; //11.4%
193
194             }else
195             {
196                   height_size = m_width * box2.height / 10;
197                   width_size = m_width * box2.width / 10;
198                   height_char = height_size * 46/100; // altura da
199                                                       sessao dos caracteres
200                   width_char = width_size * 12/100; // largura
201             }
202
203             IplImage* licenca = NULL;
204             licenca = cvCreateImage( cvSize( width_size,
205                                     height_size ), IPL_DEPTH_8U, 1);
206
207             //recorta imagem
208             cvSetImageROI(frame_gray, box2);
209             cvResize(frame_gray, licenca, CV_INTER_LINEAR);
210             cvResetImageROI(frame_gray);
211
212             //trata placa comum
213             cvThreshold(licenca,licenca,0,255,
214                           CV_THRESH_TRUNC | CV_THRESH_OTSU);
215             cvThreshold(licenca,licenca,0,255,
216                           CV_THRESH_BINARY | CV_THRESH_OTSU);
217             //trata placa colorida
218             //cvThreshold(licenca,licenca,0,255,
219                           CV_THRESH_BINARY_INV | CV_THRESH_OTSU);
220
221             cvDilate(licenca,licenca,NULL,4);
222             cvErode(licenca,licenca,NULL,2);
223
224
225             //separa os caracteres
226             CvRect box_char;
227             box_char.x = width_size*5/100;
228             box_char.y = height_size*33/100;
229             box_char.width =  width_size*88/100;
230             box_char.height =  height_size*57/100;
231
232             //saveImg(licenca);
233
```

```
234                cvShowImage("Placa", licenca);
235
236                tesseractOCR(licenca, box_char.x, box_char.y ,
237                                box_char.width , box_char.height);
238
239          }
240          cvClearSeq(approxContour);
241    }
242
243    return frame;
244 }
245
246
247
248 void tesseractOCR(IplImage *placa, int x, int y, int width, int height)
249 {
250    TessBaseAPI tess;
251    tess.SetVariable("tessedit_char_whitelist",
252                        "ABCDEFGHIJKLMNOPRSTUVWXYZ1234567890");
253    tess.Init(NULL,"placa" );
254    tess.SetImage((uchar*)placa->imageData, placa->width, placa->height,
255                      placa->nChannels, placa->widthStep);
256    tess.SetRectangle(x,y,width,height);
257    //tess.GetBoxText(0);
258    tess.Recognize(0);
259    //tess.TesseractRect
260
261    const char* out = tess.GetUTF8Text();
262    tess.End();
263
264    cout << out << endl;
265 }
266
267 void trataImagemHLS(IplImage* frame)
268 {
269    //transforma para o formato HLS
270    IplImage* frame_HLS = cvCreateImage( cvSize( frame->width,
271                                            frame->height ), IPL_DEPTH_8U,
272 3);
273    cvCvtColor( frame, frame_HLS, CV_BGR2HLS );
274
275    /**
276     * Pega os valores da imagem de entrada
277     */
278    unsigned int height = frame_HLS->height;
279    unsigned int width = frame_HLS->width;
280    unsigned int step = frame_HLS->widthStep;
281    unsigned int channels= frame_HLS->nChannels;
282    uchar *data = (uchar*) frame_HLS->imageData;
283
284     /**
285      * Aqui vamos varrer os pixels da imagem.
286      * No for externo percorremos a altura e no
287      * interno a largura.
288      */
289
290    for(unsigned int i=0; i < height; i++)
291    {
292          for(unsigned int j=0; j < width; j++){
293
```

```
294                 float H = float(data[i*step + j*channels]); // escala de
295 0 a 360
296                 float L = float(data[i*step + j*channels+1]); // escala
297 de 0 a 100
298                 float S = float(data[i*step + j*channels+2]); // escala
299 de 0 a 100
300
301
302                 // o branco fica mais cinza
303                 //if (!( (S <=30) && (L >= 70) )){
304                 //    H = 0;
305                 //    L = 0;
306                 //    S = 0;
307                 //
308
309                 // o verde fica branco
310                 //if ( !((H >= 0) && (H <=140) && (S >= 50)) ){
311                 //    L = 100;
312                 //    S = 0;
313                 //}
314
315
316                 // o azul fica branco
317                 //if ((H >= 90) && (H <=125) &&(L >= 0) && (S >= 100) ){
318
319                 //    L = 255;
320                 //    S = 0;
321                 //}
322
323                 // o vermelho fica branco
324                 //if ( (H>=330) && (H <=20) && (L >= 50) && (S >= 50)){
325                 //    L = 100;
326                 //    S = 0;
327                 //}
328
329
330                 data[i*step + j*channels]= uchar(H);
331                 data[i*step + j*channels+1]= uchar(L);
332                 data[i*step + j*channels+2]= uchar(S);
333
334         }
335     }
336   //Converte para o formato BGR (padrao)
337   cvCvtColor( frame_HLS, frame, CV_HLS2BGR );
338   cvReleaseImage(&frame_HLS);
339 }
340
341 double angle( CvPoint* pt1, CvPoint* pt2, CvPoint* pt0 )
342 {
343    double dx1 = pt1->x - pt0->x;
344    double dy1 = pt1->y - pt0->y;
345    double dx2 = pt2->x - pt0->x;
346    double dy2 = pt2->y - pt0->y;
347    return (dx1*dx2 + dy1*dy2)/
348            sqrt((dx1*dx1 + dy1*dy1)*(dx2*dx2 + dy2*dy2) + 1e-10);
349 }
350
351 void saveImg(IplImage *img)
352 {
353   time_t t = time(0);
354   struct tm tstruct;
```

```
355   char buf[80];
356   tstruct = *localtime (&t);
357   strftime(buf, sizeof(buf), "%m-%d-%H-%M-%S.jpg" , &tstruct);
358      cvSaveImage(buf,img);
359 }
360
361 IplImage* sobelImg(IplImage* frame) //utilizando a escala de cinza
362 {
363   IplImage* frame_x = cvCreateImage( cvSize( frame->width,
364                                      frame->height ), IPL_DEPTH_8U, 1);
365
366   IplImage* frame_y = cvCreateImage( cvSize( frame->width,
367                                      frame->height ), IPL_DEPTH_8U, 1);
368   IplImage* frame_sobel = cvCreateImage( cvSize( frame->width,
369                                      frame->height ), IPL_DEPTH_8U, 1);
370
371   //em X
372   cvSobel(frame, frame_x, 2, 0);
373
374   //em Y
375   cvSobel(frame, frame_y, 0, 2);
376
377   cvAddWeighted( frame_x, 0.5, frame_y, 0.5, 0, frame_sobel );
378
379   cvReleaseImage(&frame_x);
380   cvReleaseImage(&frame_y);
381
382   return frame_sobel;
383 }
```