UNIVERSIDADE FEDERAL DO RECÔNCAVO DA BAHIA
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
ENGENHARIA DE COMPUTAÇÃO

IMPLEMENTAÇÃO DE DETECÇÃO DE PLACAS VEICULARES UTILIZANDO OPENCV

MARCOS VINICIUS BIÃO CERQUEIRA

CRUZ DAS ALMAS, 2016

UNIVERSIDADE FEDERAL DO RECÔNCAVO DA BAHIA
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
ENGENHARIA DE COMPUTAÇÃO

IMPLEMENTAÇÃO DE SEGMENTAÇÃO DE PLACAS VEICULARES UTILIZANDO OPENCV

Trabalho de conclusão de curso apresentado à universidade federal do recôncavo da Bahia como parte dos requisitos para obtenção do título de **engenheiro de computação**

Orientador (a): Prof. Yuri Tavares dos Passos

MARCOS VINICIUS BIÃO CERQUEIRA

CRUZ DAS ALMAS, 2016

UNIVERSIDADE FEDERAL DO RECÔNCAVO DA BAHIA
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
ENGENHARIA DE COMPUTAÇÃO

IMPLEMENTAÇÃO DE SEGMENTAÇÃO DE PLACAS VEICULARES UTILIZANDO OPENCV

Aprovada em: 26/02/2016

EXAMINADORES:

Prof. Yuri Tavares dos Passos ASS _____

Prof. Tiago Palma Pagano ASS _____

Prof. Igor Dantas dos Santos Miranda ASS _____

Prof. Jose Valentim dos Santos Filho ASS _____

MARCOS VINICIUS BIÃO CERQUEIRA

CRUZ DAS ALMAS, 2016

AGRADECIMENTOS

Agradeço aos meus pais pelo apoio.

Agradeço a Luana, sem ela nada seria possível.

Agradeço a Yuri pela orientação

Agradeço aos colegas e professores por toda a ajuda no curso.

UNIVERSIDADE FEDERAL DO RECÔNCAVO DA BAHIA
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
ENGENHARIA DE COMPUTAÇÃO
IMPLEMENTAÇÃO DE DETECÇÃO DE PLACAS VEICULARES UTILIZANDO OPENCV

RESUMO

Esse trabalho apresenta um sistema para detecção de placas veiculares utilizando a biblioteca OpenCV. Foi feito uma revisão bibliográfica a cerca do tema expondo possíveis abordagens. O sistema utiliza as técnicas de operação morfológica como forma de encontrar regiões candidatas. Utilizou-se também o operador de Canny e a transformada de Hough, porém essas não obtiveram sucesso. A implementação proposta foi testada em 100 imagens de veículos com diferentes níveis de luminosidade e obteve-se sucesso em 40 imagens. Imagens com luminosidade alta foram mais fáceis de serem reconhecidas e imagens com sombra ou carros que possuem adesivos apresentaram maior dificuldade.

Palavras-chave: OpenCV, Detecção, Placa Veicular, Visão Computacional

UNIVERSIDADE FEDERAL DO RECÔNCAVO DA BAHIA
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
ENGENHARIA DE COMPUTAÇÃO
IMPLEMENTAÇÃO DE DETECÇÃO DE PLACAS VEICULARES UTILIZANDO OPENCV

ABSTRACT

This work presents a vehicle license plate detection system using OpenCV library and Java programming language. It was made a literature review on the subject exposing possible approaches. The system uses the morphological operation techniques in order to find candidate regions. We also used the Canny operator and Hough transform, but these were unsuccessful. The proposed implementation was tested in 100 vehicle images with different brightness levels and success was obtained in 40 images. Images with high brightness levels were more easily recognized and shadowed images or cars that have adhesives had greater difficulty.

Keywords: OpenCV, detection, vehicle license plate, computer vision.

Lista de Figuras

Sumário

1. Introdução

Visão computacional, segundo Souza, Capovilla e Eleoterio (2010), é a área da ciência que se dedica a desenvolver teorias e métodos voltados para extração automática de informações úteis contidas em uma determinada imagem. A criação de sistemas que envolvam Visão Computacional, seja para aplicações industriais ou até para navegação de robôs, envolvem um conjunto de dados obtidos através de câmeras ou scanners para então serem processadas.

De acordo com Stemmer et al (2005), sistemas de visão integram em uma única solução tecnologias diferentes, permitindo grande flexibilidade no desenvolvimento de aplicações em diversas áreas de conhecimento, com por exemplo, cálculo de distâncias, reconhecimento de placas veiculares, reconhecimento facial, entre outros. Como a visão artificial abrange um vasto leque de temas e abordagens distintas, não existe uma fórmula padrão de como deve-se fazer ou estudá-lo. A literatura propõe que os sistemas sejam feitos de forma modular, pois é possível testar e utilizar várias técnicas diferentes em uma mesma etapa do software.

Uma das técnicas bastante difundida para trabalhar-se com Visão Computacional é a biblioteca Open-CV. Segundo Marengoni e Stringhini (2009), a biblioteca OpenCV foi desenvolvida pela Intel e possui mais de 500 funções. Foi idealizada com o objetivo de tornar a Visão Computacional acessível a usuários e programadores em áreas tais como a interação humano-computador em tempo real e a robótica. A biblioteca está disponível com o código fonte e os executáveis (binários) otimizados para os processadores Intel. Essa biblioteca possui vasta documentação pela internet, podendo ser utilizada e estudada por todos. Nesse trabalho propõe-se desenvolver uma ferramenta de detecção de placas veiculares utilizando a biblioteca *OpenCV.*

1.1 Descrição do problema

O problema consiste em realizar a localização da placa no veículo seja em uma câmera em tempo real ou em um quadro de imagem. Depois de localizada, deve-se separar os caracteres da placa em novas imagens, para então fazer o reconhecimento do conjunto.

Há vários fatores que influenciam negativamente os resultados desse processo, como condições atmosféricas, condições de iluminação ou danos físicos nas placas. Outro problema encontrado é a possibilidade de inclinações que a placa pode-se encontrar em relação ao eixo da imagem. Além disso, é possível que a imagem da placa esteja obstruída parcial ou completamente ou ainda o carro não possua placa.

1.2 Objetivo

Desenvolver um detector da placa do carro e isolá-lo, dado uma imagem contendo a parte traseira de um veículo.

1.2.1 – Objetivos específicos

1. Pesquisar na literatura sobre soluções para o problema da detecção de placas em imagens de veículos.

1.3 Justificativa

A necessidade de reconhecer placas veiculares vem ganhando força nos últimos anos. Cada vez mais sistemas comerciais são desenvolvidos com esse propósito. É dispendioso o uso de recursos humanos para anotação da placa de todos os veículos que passem em um determinado local, como um estacionamento privado por exemplo. Além disso, há a possibilidade de erro humano e o trabalho de editar manualmente uma planilha eletrônica ou um banco de dados de imagens e placas.

1.4 Estrutura do texto

No capítulo 2 desse trabalho encontra-se uma revisão bibliográfica a cerca do que se trata Visão Computacional, detecção e extração de placas veiculares.

O capítulo 3 aborda trabalhos correlacionados com o tema

A descrição dos métodos desenvolvidos é tratada no capítulo 4, mostrando os pontos positivos e problemas encontrados.

O capítulo 5 mostra os resultados e algumas discussões desse trabalho e a conclusão no capitulo 6.

2. Revisão bibliográfica

Segundo Correia (2012), os dados estatísticos 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 ainda, não há informações mais elaboradas da quantidade de carros que trafegam em uma via sem o devido licenciamento do veículo.

Com o aumento do número de veículos nas cidades, aumenta-se proporcionalmente o numero de infrações e a quantidade de carros nos estacionamentos públicos e privados, fazendo com que seja necessário um controle mais preciso de toda essa frota. Porém apenas com o uso de recursos humanos não é mais suficiente para atender a crescente demanda.

A correta diferenciação entre veículos se dá através da informação contida na placa de identificação do mesmo, que apresenta 7 caracteres, sendo 3 letras seguidas de 4 números. Com essa informação, é possível identificar unicamente um veículo, salvo situações de fraudes na obtenção da placa.

Em alguns locais ainda existem o sistema de uma pessoa ficar olhando os veículos que passam e então o valor da placa é anotada em papéis que, por sua vez, são armazenados em arquivos. Esse processo é ruim por vários motivos. Dependendo da velocidade dos veículos, não é possível uma única pessoa consiga visualizar todas as placas, podendo algum carro passar sem ter sua placa anotada. Outro problema é a tabulação das informações, pois muitas folhas de papel teriam que ser lidas para mensurar a quantidade de veículos que passou em um determinado dia. Também há a possibilidade de alguém escrever o valor da placa errado. Segundo Campos (2001), um ser humano devidamente treinado necessita de um tempo da ordem de 5 segundos para conseguir identificar a placa de um automóvel, para, posteriormente, inseri-la em um

sistema, tempo este que, para ambientes de fluxo intenso de veículos, é inviável. Em resumo, o erro humano é um fator que traz desvantagens nesse processo manual.

Por isso faz-se necessário à utilização de um sistema automático e computadorizado, do qual se excluiria o erro humano, tornando todo o processo mais rápido e conseguindo organizar as informações com mais exatidão. Os métodos utilizados para a implementação de tal sistema podem ser encontrados na Visão Computacional.

2.1 – Visão Computacional

Segundo Marengoni e Stringhini (2009), não é clara a fronteira entre o processamento de imagens e Visão Computacional. Pode-se dizer que processamento de imagens é um processo onde a entrada do sistema é uma imagem e a saída é um conjunto de valores numéricos, que podem ou não compor outra imagem. A Visão Computacional procura emular a visão humana, portanto também possui como entrada uma imagem, porém, a saída é uma interpretação da imagem como um todo, ou parcialmente. Os processos de Visão Computacional geralmente iniciam com o processamento de imagens.

Ainda segundo Marengoni e Stringhini (2009), o espectro que vai do processamento de imagens até a Visão Computacional pode ser dividido em três níveis: baixo nível, nível médio e alto nível. Os processos de baixo nível envolvem operações primitivas, tais como a redução de ruído ou melhoria no contraste de uma imagem. Os processos de nível médio são operações do tipo segmentação (particionamento da imagem em regiões) ou classificação (reconhecimento dos objetos na imagem). Os processos de alto nível estão relacionados com as tarefas de cognição associadas com a visão humana.

Figura 1: Aplicação do Processamento de Imagens e Visão Computacional

A Figura 1 mostra a aplicação de processamento de imagem e a Visão Computacional. Na imagem da esquerda a foto está escura, já na foto da direita, utilizou-se Processamento de Imagem (PI) para deixar a foto mais nítida, tornando a visualização dos caracteres da placa mais simples. E a técnica de Visão Computacional (VC) reconheceu os caracteres contidos na placa. Para tal, é possível utilizamos a biblioteca *OpenCV*.

2.2 – OpenCV

Segundo Delai e Coelho (2012), *OpenCV* é um conjunto de ferramentas de programação para desenvolvimento de aplicações com Visão Computacional. Ela engloba também outro conceito importante, especialmente no meio acadêmico: o software livre. A biblioteca é possui código aberto (*open-source* – isto é, seu código fonte é disponível para o público), é distribuída gratuitamente e aberta a colaborações de quaisquer indivíduos ou empresas voluntários.

A gratuidade da *OpenCV*, o baixo custo do poder de máquina e a crescente qualidade das câmeras torna possível o desenvolvimento de sistemas sofisticados de Visão Computacional, com baixo investimento e custo de operação. A utilização de câmeras pode inclusive substituir outros sensores e sistemas mais caros, complexos e menos genéricos. Isso evidencia não só o crescimento da área da Visão Robótica, mas também a tendência de aproximação dos computadores à forma como os humanos

entendem o ambiente ao seu redor.

Um ponto interessante sobre o *OpenCV* é a diversidade de plataformas que ele pode atuar. Segundo o site oficial (http://opencv.org/) as plataformas que suportam o *OpenCV* são Windows, Linux, Mac OS, iOS e Android. Foi projetado para a eficiência computacional e com um forte foco em aplicações de tempo real. Escrito em C / C ++, a biblioteca pode tirar proveito do processamento multicore.

A *OpenCV* foi desenvolvida pela Intel e possui mais de 500 funções. Foi idealizada com o objetivo de tornar a Visão Computacional acessível a usuários e programadores em áreas tais como a interação humano-computador em tempo real e a robótica. A biblioteca está disponível com o código fonte e os executáveis (binários) otimizados para os processadores Intel. Um programa *OpenCV*, ao ser executado, invoca automaticamente uma DLL *(Dynamic Linked Library)* que detecta o tipo de processador e carrega, por sua vez, a DLL otimizada para este. Juntamente com o pacote *OpenCV* é oferecida a biblioteca IPL *(Image Processing Library)*, da qual a *OpenCV* depende parcialmente, além de documentação e um conjunto de códigos exemplos.

A biblioteca está dividida em cinco grupos de funções: Processamento de imagens; Análise estrutural; Análise de movimento e rastreamento de objetos; Reconhecimento de Padrões e Calibração de câmera e reconstrução 3D.

Muitos projetos de várias áreas de aplicação utilizam a *OpenCV* em sua codificação:

- Marengoni e Stringhini (2009) utiliza-o para reconhecimento de padrões;
- Kinjo e Soares (2013) utiliza-o para rastreamento de objetos;
- Quintans, Carnevali e Escola (2010) utiliza-o para reconhecimento facial;
- Lima e Reis (2013) utiliza-o para reconhecimento de gestos;
- Nascimento (2014) utiliza-o para fazer robôs moveis se localizarem em ambientes fechados;
- Sousa (2013) utiliza-o em dispositivos moveis para auxílio de pedestres com deficiência visual.

2.3 – Reconhecimento de padrões

Segundo Carvalho (2006), pegando-se uma base de dados qualquer, constituída de diversos exemplos, cada exemplo possui um tipo específico de padrão associado a ele. Um padrão nada mais é do que o tipo do exemplo, ou seja, um rótulo que o caracterize ou que o classifique. Os exemplos das bases de dados são geralmente medições ou observações sobre determinado assunto, definindo o domínio do processo de aprendizagem.

Um método de reconhecimento de padrões deve, baseado no conhecimento extraído dos exemplos de uma base, classificar um exemplo novo, desconhecido até então, ao padrão que mais reflete as suas características. Problemas de reconhecimento de dígitos, reconhecimento de faces, predição de tendências em séries financeiras, predição de falha sem equipamentos, e muitos outros, englobam o universo do reconhecimento de padrões. Esta área é muito extensa e surgem frequentemente novas aplicações, fazendo com que métodos poderosos sejam cada vez mais necessários.

Segundo Bianchi (2006), o interesse na área de reconhecimento de padrões tem crescido muito devido as aplicações que, além de serem desafiantes, são também cada vez mais exigentes computacionalmente. Com o avanço e a disponibilidade de vários recursos computacionais, tornou-se fácil o projeto e a utilização de elaborados métodos de análise e classificação de padrões. Em muitas aplicações, não existe somente uma única abordagem para classificação que seja “ótima” e, por isso, a combinação de várias abordagens de classificadores é uma prática bastante usada.

A escolha de uma abordagem para o reconhecimento de padrões não é uma tarefa simples e muitas vezes ela conta com a experiência do projetista. Segundo Passos (2008) e Bianch (2006) existem quatro abordagens no reconhecimento de padrões, casamento de padrões *(template matching)*, casamento sintático ou estrutural *(syntatic or structural matching)*, redes neurais *(neural networks)* e classificação estatística *(statistical classification)*.

A abordagem de casamento de padrões estuda técnicas que usam um protótipo de

padrão para detectar os diversos padrões em um conjunto de dados. A abordagem de casamento sintático ou estrutural adota uma perspectiva hierárquica onde o padrão é visto como sendo composto de subpadrões mais simples, os quais são compostos por subpadrões ainda mais simples. A abordagem de redes neurais visa utilizar redes neurais artificiais como elemento principal para o reconhecimento de padrões. Na abordagem de classificação estatística, o padrão é representado em termos de um vetor de características, que ocupam um espaço d-dimensional. Nessa abordagem descobrir o padrão é o mesmo que descobrir as regiões disjuntas formadas pelos vetores d-dimensionais. A efetividade da descoberta de um padrão na classificação estatística esta relacionada com o quão as diferentes regiões no espaço d-dimensional estão distanciadas. Essas abordagens não são independentes e existem diversas tentativas de projetar sistemas de reconhecimento de padrões que são um misto de algumas dessas abordagens

Segundo Passos (2008), mesmo com mais de 50 anos de estudo existente sobre Reconhecimento de Padrões, ainda não existe um sistema computacional de reconhecimento de padrões que seja capaz de reconhecer qualquer tipo de padrão. A causa principal disso é´ a existência de mistérios sobre algumas etapas do funcionamento do reconhecimento de padrões no ser humano. Estes mistérios são os mesmos encontrados em outras ciências e influenciam o Reconhecimento de Padrões por ser uma ciência baseada também nos estudos de outras que estudam o funcionamento dos sistemas visual, auditivo e cognitivo dos seres humanos.

2.4 – Etapas do reconhecimento de placas veiculares

Esse trabalho assume que já se tem armazenado as imagens a serem processadas. Segundo Campos (2001) e Trentini, Godoy e Marana (2010), são necessários quatro etapas para o sucesso do processo, pré-processamento, identificação da placa, extração dos caracteres e reconhecimento dos caracteres.

Figura 2: Etapas do reconhecimento de placa veicular

Na Figura 2, é apresentado um fluxograma com a sequência das principais etapas do processo de identificação de placas veiculares.

2.4.1 – Pré-processamento

Como as imagens utilizadas pelos computadores são capturadas por outros dispositivos como, por exemplo, câmeras fotográficas ou câmeras de vídeo, às vezes não sabendo a sua qualidade de imagem elas estão passíveis a erros na captura e posterior representação digital.

Segundo Trentini, Godoy e Marana (2010), para que as fases de processamento de imagens possam executar suas tarefas de maneira mais eficiente, uma fase de pré-processamento pode ser realizada antes do início das fases de identificação da placa e da

segmentação de caracteres. Essa fase é responsável por efetuar correções de distorções geométricas as quais foram geradas no momento da captura e também a eliminação de ruídos. Segundo Campos (2001), no pré-processamento são usados algoritmos para realçar dados de interesse e eliminar dados que podem dificultar a segmentação. Dentre os quais existem a limiarização, exclusão de linhas e a suavização.

2.4.2 – Segmentação da placa

Essa fase caracteriza-se pelo isolamento da placa perante toda imagem, a literatura também chama essa fase de identificação da região da placa. Todo o processo é feito para que se consiga identificar a placa para poder retirá-la do restante da imagem.

Grande parte dos trabalhos, no que diz respeito à localização, baseia-se em algumas características das placas que a destacam na imagem de um carro. A bibliografia apresentada na seção 4 mostra várias abordagens para essa etapa como a utilização da forma geométrica da placa ou da diferença de cores.

2.4.3 – Segmentação dos caracteres

Também conhecido como extração dos caracteres, a segmentação dos caracteres consiste num processo cujo objetivo é separar em imagens distintas, cada caractere. Essa etapa permitirá a identificação em separado de cada letra ou número que compõem a placa.

O uso de métodos mais simples ou mais complexos, bem como a taxa de sucesso deles, depende muito de como foi feita a localização da placa e da qualidade dos algoritmos de pré-processamento das imagens, principalmente de binarização.

2.4.4 – Reconhecimento dos caracteres

Segundo Oliveira e Gonzaga (2012),o reconhecimento de caracteres e a etapa em que um objeto segmentado é analisado e associado a um único caractere alfanumérico. Diversos autores utilizam redes neurais para o processo de Reconhecimento. O objetivo

básico de sistemas baseados em redes neurais é realizar um treinamento prévio do programa com um banco de dados especifico para o treinamento, para então fazer a análise das placas de outro banco de dados.

2.5 – Métodos utilizados nesse trabalho

2.5.1 – Operador Canny: detector de bordas

Segundo a documentação do OpenCV, o detector de Bordas Canny, foi desenvolvido por John F. Canny em 1986. Também conhecido por muitos como o *detector ideal,* o algoritmo de Canny visa satisfazer três critérios principais:

- Baixa taxa de erros: Significa uma boa detecção das arestas existentes;
- Boa localização: A distância entre os pixels detectados e os pixels reais, é minimizada;
- Resposta mínima: Apenas uma resposta do detector por aresta.

Segundo Marengoni e Sringhini (2009), *Canny* conseguiu representar estas ideias matematicamente e, utilizando otimização numérica, definiu um detector de borda baseado na primeira derivada de uma Gaussiana. Numa primeira etapa a imagem é convoluída por uma função Gaussiana, em seguida são determinadas a magnitude e a direção, conforme indicado pelas expressões na Equação 1.

$$g(x,y) = f(x,y) ** G(x,y)$$
$$M(x,y) = \sqrt{g_x^2 + g_y^2}$$
$$\alpha(x,y) = \tan^{-1}\left(\frac{g_y}{g_x}\right)$$
$$g_i = \frac{\partial g(x,y)}{\partial i}$$

Equação 1: Canny

Onde:

- g(x,y): Resultado da convolução
- G(x,y): Função Gaussiana
- gx, gy:Gradiente no ponto (x,y)

- f(x,y): Imagem original
- M(x,y): Magnitude de pixels
- α(x,y): Direção da normal de cada pixel
- i pode ser X ou Y

Os valores de g_i, podem ser obtidos a partir dos operadores de Prewitt ou de Sobel. M(x, y) é uma imagem que contém a informação da magnitude em cada pixel e α(x, y) é uma imagem que contém a direção da normal à borda para cada pixel. A ideia é, para cada pixel p, verifica se pelo menos um dos vizinhos de p possui a mesma direção que p, se sim, marca o pixel com M(x, y) (as coordenadas de p), senão marca a imagem de saída com 0.

Finalmente utiliza-se um operador de corte para reduzir pontos de borda falsos. O operador de corte, neste caso, é baseado em histerese, segundo Marengoni e Sringhini (2009). O operador de corte baseado em histerese possui dois valores, um valor chamado de alto e outro chamado de baixo. São feitas as operações de corte usando estes valores e são obtidas duas imagens binárias, estas imagens são subtraídas uma da outra para se obtiver a imagem final. O tamanho da Gaussiana utilizada é o mesmo definido para o operador de Marr-Hildreth. A Figura 3 mostra um exemplo de do operador de Canny.

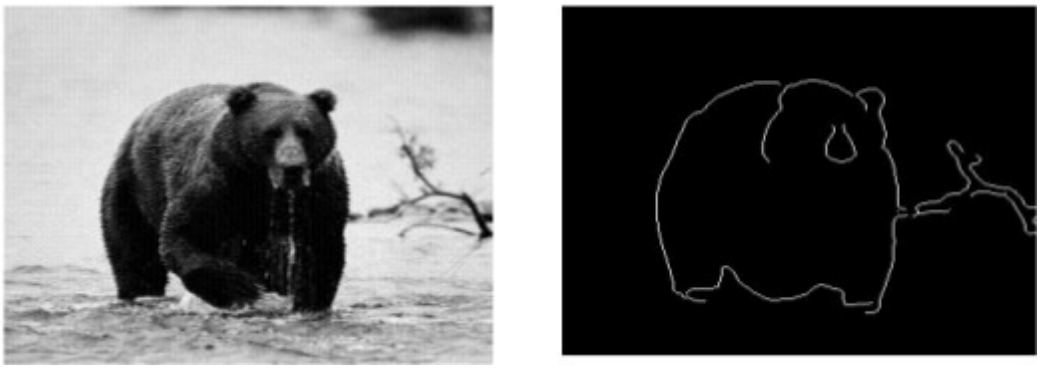

Figura 3: Exemplo do método de Canny

2.5.2 – Transformada de Hough

Segundo Jamundá (2000), a transformada de Hough foi desenvolvida por Paul Hough em 1962 e patenteada pela IBM. Originalmente, foi elaborada para detectar

características analiticamente representáveis em imagens binarizadas, assim como linhas, círculos e elipses. Na última década tornou-se uma ferramenta de uso comum na visão artificial para o reconhecimento destas características. A transformada de *Hough* (HT) é um método padrão para detecção de forma que são facilmente parametrizadas (linhas, círculos, elipses, etc.) em imagens computacionais. Em geral, a transformada é aplicada após a imagem sofrer um pré-processamento, comumente a detecção de bordas.

O conceito principal da HT está em definir um mapeamento entre o espaço de imagem e o espaço de parâmetros. Cada borda de uma imagem é transformada pelo mapeamento para determinar células no espaço de parâmetros, indicadas pelas primitivas definidas através do ponto analisado. Essas células são incrementadas, e indicarão no final do processo, através da máxima local do acumulador, quais os parâmetros correspondentes a forma especificada.

Segundo Moreira (2012), o método original de Hough utilizava a parametrização inclinação intersecção (slope-intercept) para mapear um ponto na imagem para o plano de parâmetros empregando a Equação 2:

$$y = mx + b$$

Equação 2:Parametrização

Onde m e b são os coeficientes angular (inclinação) e linear (intersecção), respectivamente, da reta, enquanto que x e y são os valores da projeção do ponto sobre o eixo x e y, respectivamente. Todavia, tanto a inclinação como a intersecção é ilimitada e essa parametrização tem a desvantagem de ser sensível à escolha do eixo de coordenadas no plano da imagem, o que dificulta a aplicação da técnica.

Ainda segundo Moreira (2012), Duda e Hart (2001) avaliaram que um dos problemas mais constantes do processamento de imagens era a detecção de linhas retas. Eles perceberam o problema na abordagem de Hough e propuseram uma solução para a aplicação da técnica. Os autores utilizaram coordenadas polares para definir o segmento de reta, trabalhando com os parâmetros ângulo e raio ao invés de inclinação e

intersecção.

No plano da imagem, uma reta parametrizada pode ser representada por dois parâmetros: ρ sendo à distância da normal até a linha da origem da imagem e θ sendo o ângulo dessa normal com o eixo horizontal. A equação da reta correspondente a essa geometria é dada pela Equação 3:

$$\rho = x\cos\theta + ysen\theta$$

Equação 3: Equação da reta em coordenada polar

Como os novos parâmetros utilizados para representar o espaço são definidos agora por ρ e θ, o problema de detectar pontos colineares pode ser convertido no problema de se encontrar curvas concorrentes.

Ainda segundo Moreira (2012) apud Deans (1983), é percebido que a Transformada de Hough é um caso especial da transformada Radon. Uma transformação que passou quase despercebido durante meio século, mas agora está sendo amplamente explorada, especialmente no campo da tomografia computadorizada. A Transformada Radon é uma função f(x,y) definida em um plano bidimensional Euclidiano como:

$$R(\rho,\theta) = \iint f(x,y)\delta(\rho - x\cos\theta - ysen\,\theta)dxdy$$

Equação 4: Transformada de Radon

Entretanto para se calcular a transformada de Hough utiliza-se uma matriz que inicialmente tem o valor zero, as dimensões da matriz dependem dos intervalos previamente fixados pelos eixos ρ e θ do espaço de parâmetros.

Para cada ângulo encontrado do intervalo [0, π] e para cada ponto da imagem (x,y) da imagem de entrada, um novo valor de ρ é encontrado, a partir da equação:

$$\rho = x\cos\theta + ysen\theta$$

Equação 5:Parametrização

Após o cálculo, a célula da matriz acumuladora na posição θ, ρ é acrescentada no espaço de Hough e construída uma curva senoidal.

Para identificar retas, é utilizados um aprimoramento da transformada de Hough, a Transformada de Hough probabilística. Essa variação analisa aleatoriamente os pixels de acordo com uma função de densidade uniforme, definida em cima da imagem. Isto é equivalente a computar a transformada padrão de uma amostra da imagem.

2.5.3 – Operações morfológicas

Segundo Cavalho (2006), A fundamentação teórica da morfologia matemática binária é a teoria dos conjuntos. Em imagens binárias, os pontos no conjunto são chamados de primeiro plano (FOREGROUND) e aqueles do conjunto do complemento são chamados de fundo (BACKGROUND). Além de tratar das operações de conjunto usuais de união e de interseção, a morfologia depende amplamente das operações de união, interseção, complemento, diferença, translação, reflexão de conjuntos (GONZALES &WOODS, 2001).

Sejam A e B são conjuntos em um espaço euclidiano n dimensional:

1. A união dos conjuntos A e B, é denotado por C = A ∪ B , sendo C o conjunto de todos os elementos pertencentes a A , B e a ambos.
2. A interseção dos conjuntos A e B, é denotado por D = A ∩ B , sendo D o conjunto de todos os elementos pertencentes a ambos.
3. O complemento do conjunto A é o conjunto dos elementos não contidos em A

   definido como $A^c = \{x \mid x \notin A\}$.

4. A diferença de dois conjuntos A e B denotado por A − B, é definido como A − B = {x

$| X \in A , x \notin B\} = A \cap B^c$.

5. A translação de uma dada imagem A, para uma posição x, onde $x \in E$, é definida pela translação de todos os elementos de A , denotado por A_x . Assim a translação é definida por $A_x = \{c \mid c = a + x , a \in A\}$.

2.5.3.1 – Elemento estruturante

Segundo Carvalho (2006), o elemento estruturante (EE) é um conjunto utilizado para percorrer uma imagem a ser estudada operando uma transformação morfológica. Geralmente sua forma é escolhida de acordo com um conhecimento prévio da geometria procurada na imagem, tipicamente uma parte da própria imagem. É necessário definir a origem do elemento estruturante. A partir dessa origem, o EE será considerado ao ser operado na imagem. A forma do EE é adaptada para as medições que se deseja obter. Comumente, defini-se o ponto central do EE.

2.5.3.2 – Dilatação

Dada uma imagem binária A, a operação de dilatação de A pelo elemento estruturante B, ambos os conjuntos em Z^2, denotado por $\delta_B (A)$, é definida pela Equação 6:

$$\delta_B(A) = \{x : A \cap B_x = \varnothing\} = A \oplus \hat{B}$$

Equação 6: Dilatação

Podemos considerar na definição acima que a dilatação de A por B equivale à soma de Minkowski, segundo Wang e Bertrand (1988) com $\hat{B}$, se o conjunto B for invariante em relação à simetria. Assim também podemos definir dilatação sendo equivalente a uma união de todas as translações da imagem original pelo elemento estruturante

$$A \oplus B = \bigcup_{b \in B} A_b$$

Equação 7: União das translações

Figura 4: Processo de dilatação

2.5.3.3 – Erosão

Dada uma imagem binária A, sua erosão pelo elemento estruturante B, ambos os conjuntos em Z^2, denotado por $\varepsilon_B(A) = A \ominus \hat{B} = \{x : B_x \subset A\}$ é denotado por $A \ominus B$, e definido na Equação 8:

$$\varepsilon_B(A) = A \ominus \hat{B} = \{x : B_x \subset A\}$$

Equação 8: Erosão

Como visto anteriormente a dilatação pode ser representada como uma união de todas as translações, a erosão pode ser representada como a interseção da translação de A por todos os elementos de B invertidos e também podemos aplicar a subtração de Minkowski, segundo Wang e Bertrand (1988):

$$A \ominus B = \bigcap_{b \in B} A_{-b}$$

Equação 9: Interseção das translações

onde -b é a multiplicação escalar do vetor b por -1, o que equivale a refletir B em relação à origem dos eixos.

Figura 5:Processo de erosão

2.5.3.4 – Abertura

Abertura de uma imagem binária A por um elemento estruturante B, denotado por A^B , é definido na Equação 10:

$$A_B = \delta_B\left(\varepsilon\left(A\right)\right) = (A \ominus \hat{B}) \oplus B$$

Equação 10: Abertura

A imagem original sofre a erosão primeiramente e em seguida é dilatada. Consequentemente, pode-se, intuitivamente, pensar como o "deslocamento do elemento estruturante sobre o limite interno da imagem". A abertura também pode ser representada por $A \circ B$ e da interpretação geométrica obtêm-se:

$$A \circ B = A_B = \bigcup \{B_x \mid y \in E, B_x \subset A\}$$

Equação 11: Interpolação geométrica

A operação de abertura afeta a imagem de maneira a eliminar objetos pequenos e finos quebrando objetos nos pontos estreitos, geralmente deixando os contornos dos grandes objetos lisos e uniformes, sem mudanças bruscas em sua extensão.

Figura 6:Processo de abertura

2.5.3.5 – Fechamento

O Fechamento de uma imagem binária A pelo elemento estruturante B, é denotado por A^B, é definido pela Equação 12:

$$A^B = \varepsilon_B\left(\delta_B\left(A\right)\right) = \left(A \oplus \hat{B}\right) \ominus B$$

Equação 12: Fechamento

A operação de fechamento afeta a imagem de maneira a preencher buracos pequenos e finos presentes no objeto. Ela conecta objetos vizinhos a partir de seu ponto mais próximo e geralmente os contornos dos objetos ficam lisos e uniformes sem mudanças bruscas em sua extensão.

Figura 7:Processo de fechamento

2.5.3.6 – Top Hat por abertura

A operação de Top Hat por abertura é utilizada para detectar picos em uma

imagem. O Top Hat de abertura de uma imagem f pelo elemento estruturante *g*:

$$h^g(f) = f^g(f) - f$$

Equação 13:Top Hat por abertura

Como a abertura é um processo anti-extensivo, o seu resultado fica abaixo do sinal original. Utilizando um elemento estruturante adequado, o processo permite a eliminação dos picos existentes. Aplicando-se a diferença entre o sinal original e o resultado da abertura, permitindo, assim, a exclusão de ruídos.

Figura 8: Processo de Top Hat

2.5.3.7 – Black Hat (Top Hat por fechamento)

A técnica de Black Hat é utilizada para a detecção de vales em uma imagem. O tophat por fechamento de um sinal f pelo elemento estruturante g é definido:

$$h^g(f) = f^g(f) - f$$

Equação 14: Black Hat

Como o fechamento é um processo extensivo, o seu resultado da transformação é sempre positivo.

Com o uso de um elemento estruturante apropriado, o processo de fechamento permite a eliminação dos vales. Com a subtração, entre o sinal original e o resultado do fechamento, elimina-se o ruído e suprime-se a falta de homogeneidade, ou seja, ressalta-se a informação dos vales da imagem.

Figura 9:Processo de Black Hat

3 – Trabalhos relacionados

A literatura traz diversas pesquisas em torno desse tema. Abordagens diferentes para situações diversas são trabalhadas nesses trabalhos.

Souza, Capovilla e Eleoterio (2010) não trabalha com uma problemática em especifica em torno do reconhecimento de placas veiculares, porém aborda várias técnicas de processamento de imagens, transformações, histogramas e filtros, falando também sobre segmentação de imagem e reconhecimento de padrões.

Campos (2001) desenvolve um sistema composto por oito etapas, sendo que as cinco primeiras são responsáveis pela localização da placa e as três últimas pela extração e reconhecimento dos caracteres. A imagem original é convertida de colorida para monocromática, depois um algoritmo de análise por variação tonal faz uma varredura em busca do contraste entre o fundo da placa e os caracteres. Então binariza-se a imagem e executa-se um algoritmo que busca elementos que possuam dimensões compatíveis com um padrão esperado. Uma vez que o local da placa tenha sido encontrado, é necessário extrair os caracteres do restante da imagem. Isso é feito por um algoritmo de agrupamento por similaridade, que tem a finalidade de separar cada elemento encontrado e outro algoritmo que avalia os elementos extraídos, descartando possíveis ruídos. Depois desse processo, utiliza uma rede neural do tipo *feedforward* para a identificação dos caracteres. A Figura 10 apresenta a aplicação desenvolvida por Campos (2001).

Trentini, Godoy e Marana (2010) faz uma conversão da imagem para monocromática, assim como Campos (2001), e depois usa o filtro Smooth para suavizar a imagem. Logo após realiza-se a detecção de bordas pelo operador de Canny. Para realizar a segmentação, busca-se por um padrão em específico, um objeto retangular com quatro vértices, deve-se também ter uma área predeterminada por testes anteriores e duas arestas consecutivas devem ser de 90°. Na extração dos caracteres foi desenvolvido um método que verifica a quantidade de pixels pretos contidos em cada coluna, fazendo em seguida o corte da imagem, gerando 7 novas imagens, como exemplifica na Figura 11. Na fase de reconhecimento de caracteres, utiliza-se o algoritmo *Random Trees* ou *Random Forests*, que é um classificador baseado em árvore de decisão. Este método conseguiu reconhecer 93,7% das letras e 86,9% dos números.

Figura 10:Sistema desenvolvido por Campos(2001)

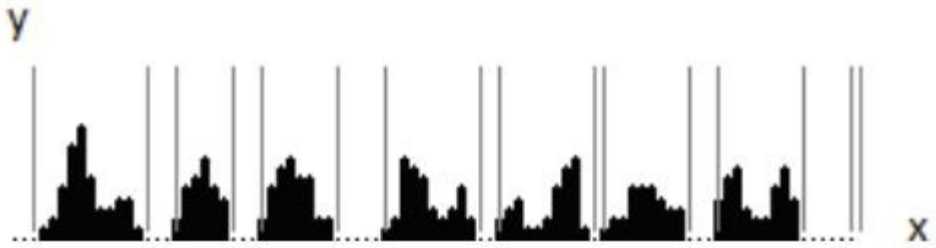

Figura 11: Histograma vertical

Conci e Monteiro (2003) aplica diversos métodos de computação visual combinando aspectos de computação gráfica, processamento de imagens e inteligência artificial. Em um primeiro momento a imagem é binarizada, tornando os pixels importantes mais fáceis de serem identificados. Logo após usou-se o processo de erosão, para corrigir um problema que aparecia com bastante frequência no processo de identificação. Esse processo limpa determinados ruídos encontrados na imagem depois dela ser binarizada. Para a segmentação das imagens foi desenvolvido um algoritmo que basicamente verifica a vizinhança de cada pixel e a partir daí vai identificando quem faz parte, ou não, do mesmo objeto, isso é está com a mesma cor que ele. Como o sistema desenvolvido para o reconhecimento objetiva 100% de acerto, foram usados duas formas de extração de características geométricas de cada um dos caracteres. A primeira utiliza a forma geométrica extraída da fase anterior, já a segunda considera apenas as características essenciais dos caracteres e necessita que eles passem por um processamento prévio de afinamento ou esquematização. Nos testes, foram utilizadas 505 imagens sendo que

apenas 49 apresentaram erro em alguma das etapas do trabalho.

Pereira (2014), inicialmente aplicou a conversão da imagem de entrada para escala de cinza, seguido pela aplicação de um filtro de suavização. Na etapa de localização da placa, etapa que autor julga como o maior desafio para esse tipo de problema, aplicou-se o algoritmo de detecção de bordas de Canny, em seguida foi executada uma sequência de operações morfológicas e por ultimo foi executado uma algoritmo de busca de contornos. Após a aplicação das operações morfológicas, utilizou-se uma função chamada *findContours* que tem como entrada uma imagem binária e após sua execução gera uma vetor contendo os conjuntos de pontos que definem as bordas das regiões retangulares presentes na imagem de entrada. Após a identificação dos caracteres, o algoritmo identificou 94% das 100 imagens utilizadas.

Ferreira (2012) usou uma abordagem diferente dos anteriores, ele utilizou a cor do fundo da placa para conseguir identificar a sua localização. A cor da placa é procurada levando em consideração a sua conectividade com os vizinhos similares, bem como os padrões de cores das placas veiculares. Para isso cada pixel terá suas componentes de cores avaliadas, e se estiver atendendo ao critério de busca, o pixel em questão é marcado como uma região candidata. Um algoritmo foi desenvolvido para percorrer todos os pixels da imagem original analisando se a tonalidade tem possibilidade de ser uma placa de cor cinza. Após isso o algoritmo verifica e marca as células candidatas que são conexas com outras candidatas a serem parte da placa. Posteriormente as células são percorridas novamente e agrupadas por uma identificação de grupo conexo, formando um conjunto de elementos com as mesmas características. Com os agrupamentos formados, calcula-se a média aritmética dos pixels vizinhos. Um algoritmo encontra as coordenadas dos quatros cantos de cada polígono. Polígonos com menos de 100 pixels na horizontal são descartados. Caso não exista polígono candidato para a placa de cor cinza, as operações são repetidas para vermelho, branco, preto e azul, sendo que o filtro aplicado para busca do pixel candidato será específico para cada cor de fundo. Com a placa segmentada, é feito uma correção do nivelamento e uma binarização, fazendo assim, surgir espaços vazios entre os elementos da placa que permitem localizar os caracteres e elementos da mesma. Para segmentação dos caracteres, uma distribuição de pixels horizontal para identificar o espaço entre os caracteres alfanuméricos e a informação da cidade/estado. Depois é feito uma distribuição de pixels vertical, conseguindo encontrar a posição de cada caractere para realizar a extração. Por fim, foi utilizado duas redes

neurais *perceptron* de 3 camadas com o treinamento realizado com *backpropagation*, uma para identificar as letras (26 células na camada de saída) e outra para identificar os números (10 células na camada de saída). Todas as letras e números que serviram de teste obtiveram sucesso de 100% no reconhecimento, contudo somente testes com placas sem artefatos foram realizados. Uma identificação em PC convencional leva cerca de 0,25 s para as letras, e mais 0,16 s para os números.

Carvalho (2006), utilizou quatro bases de imagens diferentes. Elas se diferem pela quantidade de fotos, pelo ângulo de captura, pela iluminação e pela distancia da câmera em relação ao carro. Inicialmente converte-se a imagem em escala de cinza e aplica a operação de Top Hat, para simplificação da imagem. Realiza-se o fechamento com um elemento estrutural horizontal linear, uma abertura com um elemento estrutural vertical linear para eliminar os objetos de altura inferior à altura mínima dos caracteres. Depois se faz outra abertura para eliminar os objetos de altura inferior a altura máxima dos caracteres e uma subtração da ultima imagem com a penúltima. Por fim, aplica-se uma dilatação para assegurar de que se obtém a placa inteira. Carvalho (2006) também faz um comparativo de diversas situações que a imagem pode apresentar:

- Fator de iluminação
- Imagem com vários carros
- Carros com adesivo

Em media, o algoritmo proposto reconheceu 73,23% das imagens apresentadas 12,81% delas foram reconhecidas parcialmente, sobre um total de 324 imagens

Aragão (2012) considerou já ter as imagens da placa segmentadas, começando assim o seu trabalho a partir da etapa de segmentação dos caracteres. Para isso, foi utilizado a técnica de histograma vertical. Para cada caractere, foi feito a projeção vertical, calculando-se a media dos resultados. O algoritmo comparara os valores atuais com as medias calculadas anteriormente, podendo assim identificar a posição dos caracteres e segmentá-los. Para o reconhecimento, utilizou-se uma rede neural perceptron de múltiplas camadas, pois conseguem resolver problemas complexos, segundo Aragão (2012) apud Russel e Norvig, 2004. O treinamento foi realizado usando o algoritmo de retropropagação (*backpropagation*). Duas redes foram utilizadas, uma para letras e outra para números. O método proposto conseguiu segmentar 53% das placas testadas, desse número 82,9% das letras foram reconhecidas e 84,6% dos números.

4 – Descrições do método proposto

Neste capítulo será apresentada a metodologia usada no desenvolvimento do trabalho e também a descrição de cada uma das etapas que o constitui. Aqui se buscou a confecção de uma base de dados, implementação e a experimentação do método proposto.

A plataforma de desenvolvimento empregada na implementação deste trabalho foi o *Java (OpenJdk 1.8.0_65)*. Utilizou-se a versão 2.4.9 do *OpenCV* para Java encontrado no link: *http://opencv.org/*.

4.1 – Bases de dados

Para o desenvolvimento de um trabalho de análise de imagens é necessária à utilização de uma base de dados, pois a disponibilidade de um número significativo de imagens é um fator importante para qualquer projeto de pesquisa experimental nesta área.

Os testes e verificações foram realizados em uma base de dados contendo um total de 2400 imagens. Conseguiu-se essa base de dados junto a SMTT de Aracaju, onde todas foram falsos positivos, o sistema de radar fez a captura da imagem em momento indevido. Na Figura 12, encontra-se algumas imagens de exemplo.

Figura 12: Exemplo da base de imagens

4.2 – Desenvolvimento

Os passos de processamento de imagens para obter-se a posição da placa do veículo a partir de uma imagem digital em níveis de cinza serão mostrados a seguir.

4.2.1 – Segmentação

A primeira implementação seguiu o fluxograma apresentado na Figura 13:

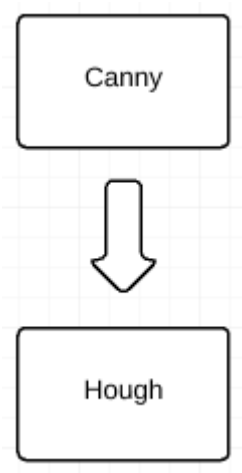

Figura 13: Fluxograma da primeira implementação

A ideia inicial era utilizar o operador *Canny*, para detectar as bordas e depois usar a transformada de *Hough* para traçar as retas e a transformada de *Corner Harris*, podendo identificar a região da placa.

Como as imagens já estavam em tons de cinza, não se fez necessário essa etapa no processamento. A chamada do método de *Canny* é a seguinte:

Canny (grayImage, cannyImage, lowThreshold, highThreshold, kernel_size);

Onde:

- grayImage: imagem de origem
- cannyImage: imagem de destino
- lowThreshold: primeiro limite para o limiar
- highThreshold:segundo limite para o limiar

- kernel_size:tamanho de abertura para luminosidade

Empiricamente, os valores escolhidos para *lowThreshold* foi 100, para o *highThreshold* foi 100 e para o *kernel_size* foi 3. Na Figura 14, tem-se a imagem original e o resultado da aplicação do operador de *Canny*.

Figura 14: Aplicação do operador Canny

Com esses parâmetros, conseguiu-se destacar as bordas da placa sem uma grande quantidade de ruídos, como ocorre na Figura 15, que apresenta um exemplo de aplicação do operador Canny com parâmetros diferentes dos utilizados.

Figura 15: Resultado da aplicação do Canny com muitos ruídos

Com as bordas destacadas, utilizou-se a transformada de *Hough*, com o seguinte

método:

HoughLinesP (cannyImage, OutputArray, rho, theta, threshold, minLineLength, maxLineGap)
onde:

- *cannyImage:* imagem de entrada (imagem em bordas)
- *OutputArray:* array de saída (posição das retas a serem traçadas)
- *rho:* Distancia dos pixels
- *theta:* ângulo do acumulador em radianos
- *threshold:* parâmetro do acumulador
- *minLineLength: numero* mínimo de pontos para identificar uma linha
- *maxLineGap:* diferença máxima entre 2 pontos para se considerar uma reta

Os resultados não foram satisfatórios. Inicialmente utilizou-se os seguintes parâmetros, rho = 1, theta= π/180, *threshold= 100, minLineLength=2, maxLineGap = 2.* Um exemplo do resultado é apresentado na Figura 16.

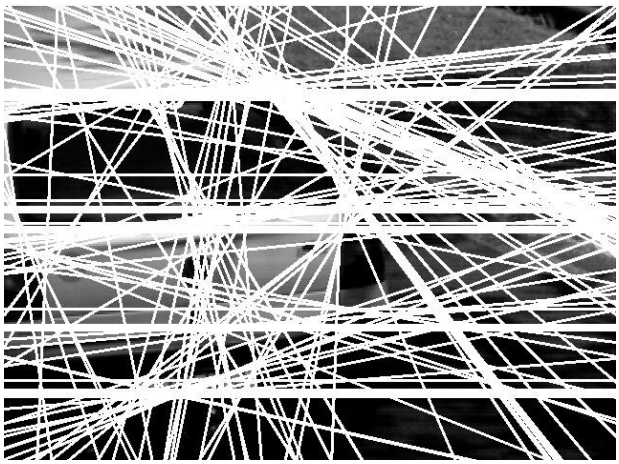

Figura 16: Resultado das linhas traçadas

Os testes com esta abordagem não foram satisfatórios (descritos na seção 5). Por conta disto, novos métodos foram pesquisados e o escolhido foi o método utilizado por Conci, Carvalho e Rauber (2009). Na Figura 17 encontra-se o fluxograma da 2ª implementação

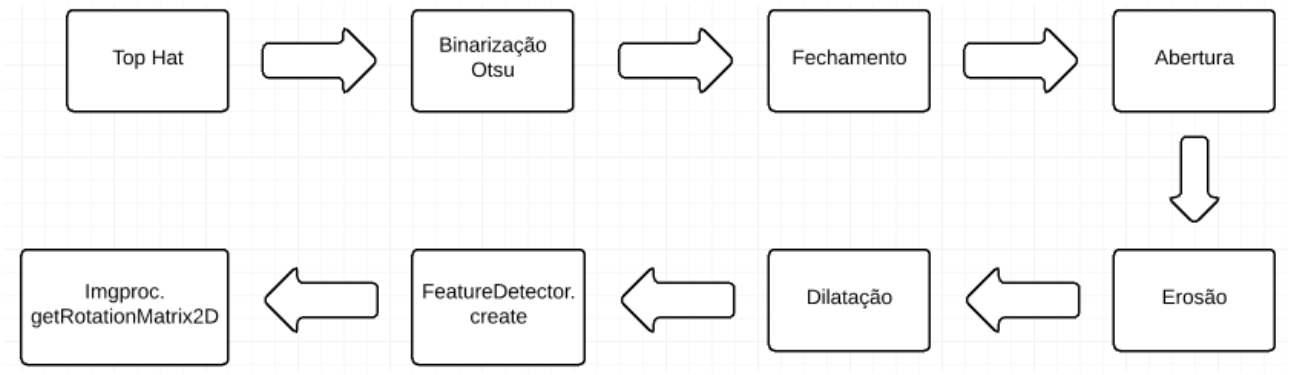

Figura 17: Fluxograma da segunda implementação

Inicialmente carrega-se a imagem e para ressaltar as variações na placa aplica-se a operação morfológica Top Hat com um elemento estruturante circular de tamanho 12x12. Um exemplo desta aplicação encontra-se na Figura 18. É necessário eliminar a maior quantidade de ruído presente na imagem. Para a redução da complexidade é feita a binarização da imagem pelo método de Otsu, o resultado encontra-se na Figura 19. Em seguida, aplica-se a operação morfológica de fechamento (*close*), com isso, agrupa-se as regiões em branco que estão próximas, de acordo com o elemento estruturante. O EE utilizado foi retangular com tamanho 10x5 (Figura 20).

Figura 18: Aplicação de TopHat

Figura 19: Imagem com binarização de Otsu

Após a operação de fechamento, aplica-se a operação de abertura com um elemento estruturante de abertura com tamanho 20x20. Para eliminar os diversos pixels acesos que não satisfazem as características da placa, definem-se parâmetros que representam a altura mínima e máxima e a largura mínima e máxima dos caracteres. Faz-se uma abertura com um elemento estruturante vertical linear eliminando os objetos de altura inferior à altura mínima dos caracteres. Depois se faz outra abertura para eliminar os objetos de altura inferior a altura máxima dos caracteres (Figura 21).

Figura 20: Imagem após operação de fechamento

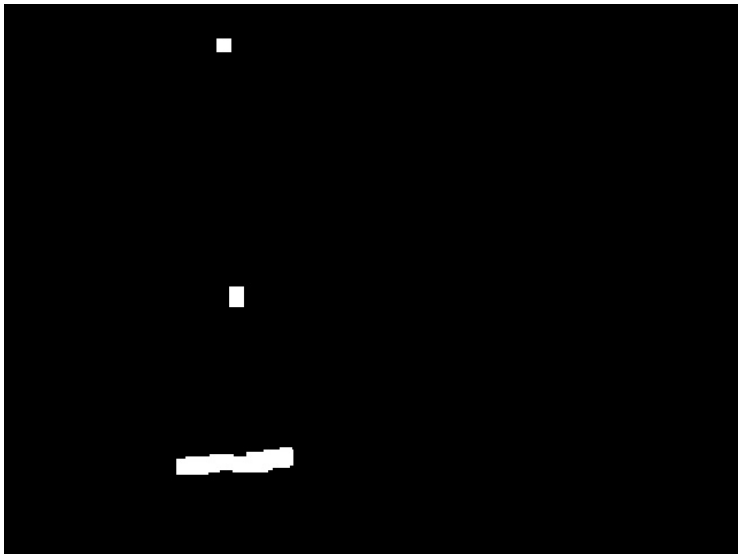

Figura 21: Imagem após operação de abertura

Em seguida, aplica-se o operador erosão. Com esse operador morfológico espera-se eliminar ruídos menores que a região da placa. O elemento estruturante utilizado foi um retangular com tamanho 20x15 (Figura 22).

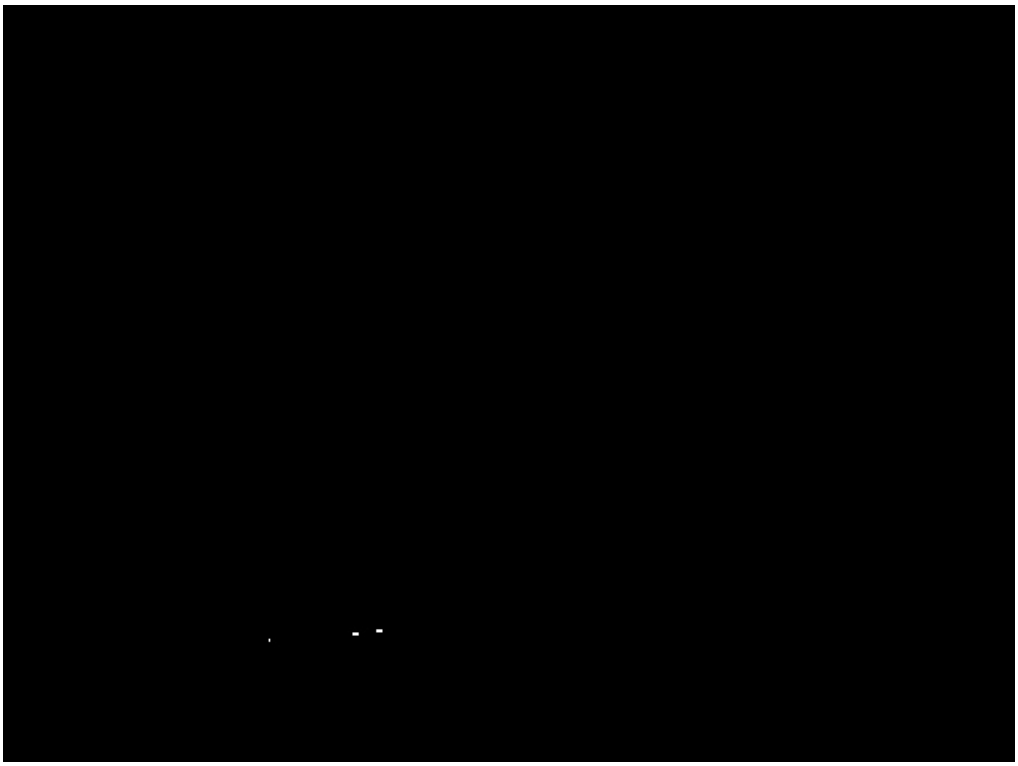

Figura 22:Imagem após operação de erosão

Para garantir que toda a região da placa seja reconhecida, aplica-se a operação de

dilatação, onde ela essa operação ira expandir as regiões em branco de acordo com o elemento estruturante. O EE que foi utilizado foi um retangular com tamanho 30x25 (Figura 23).

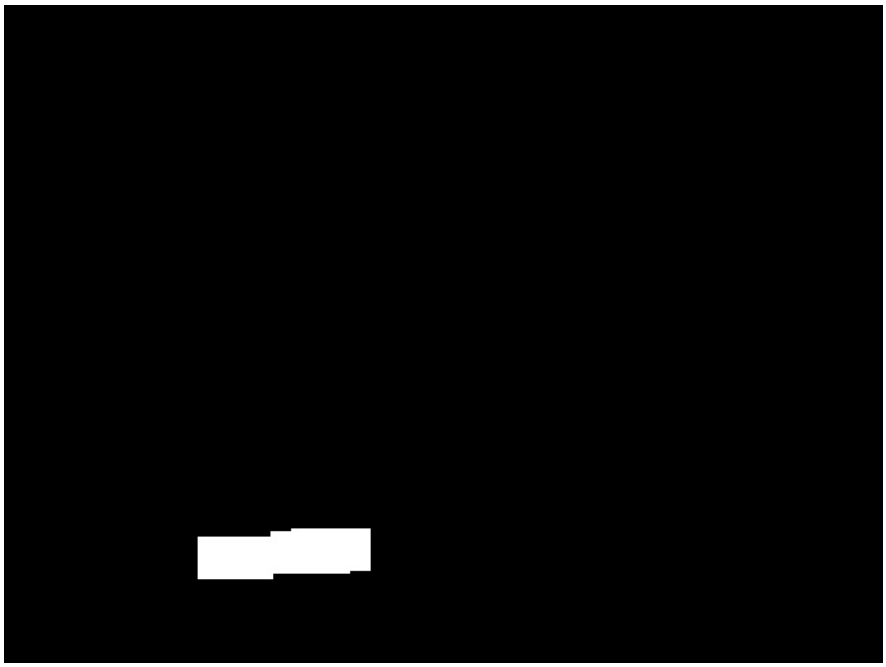

Figura 23: Imagem após operação de dilatação

Com a região identificada, precisa-se agora retirar essa região da imagem original. Para isto, utiliza-se o método FeatureDetector.create (FeatureDetector.Harris). *Esse método retorna todos os pontos que correspondem a quinas. Esses pontos podem ser visualizados utilizando o método* Features2d.drawKeypoints, cujos argumentos são a imagem da placa e um array contendo todos os pontos detectados anteriormente (Figura 24). Porém não será necessário todos os pontos, precisa-se apenas de 4 pontos, os extremos superiores direito e esquerdo e o extremos inferiores direito e esquerdo. Para isso percorreu-se os pontos marcados buscando pelos 2 menores valores de X, encontrando assim os extremos da esquerda e depois buscou-se os 2 maiores X, encontrando os pontos da extrema esquerda. Como exemplo desta etapa, utilizando novamente o método *Features2d.drawKeypoints* com os quatros pontos encontrados tem-se a imagem da Figura 25.

Nessa etapa, pega-se o menor e o maior X e o menor e o maior Y para determinar uma região retangular a ser retirada da imagem original, onde a largura é igual à subtração do maior X e menor X e a altura é a subtração do maior Y e o menor Y. Com isso, pode-se realizar a extração. Um exemplo da imagem resultante encontra-se na

Figura 24:Pontos encontrados pelo FeatureDetector.create

Figura 26.

Porém a placa ainda não está alinhada. É preciso fazer uma rotação para fazer esse alinhamento. Para isto, é calculado o angulo da reta superior e da reta inferior. Esse ângulo é calculado através do arco tangente. Depois de calculado, é escolhido o menor deste dois ângulos para fazer a rotação. No OpenCV, isto é feito com método *Imgproc.getRotationMatrix2D* seguido do método *Imgproc.warpAffine* para realizar a rotação.Um exemplo de imagem encontra-se na Figura 27. Com isso conclui-se a etapa de detecção da placa.

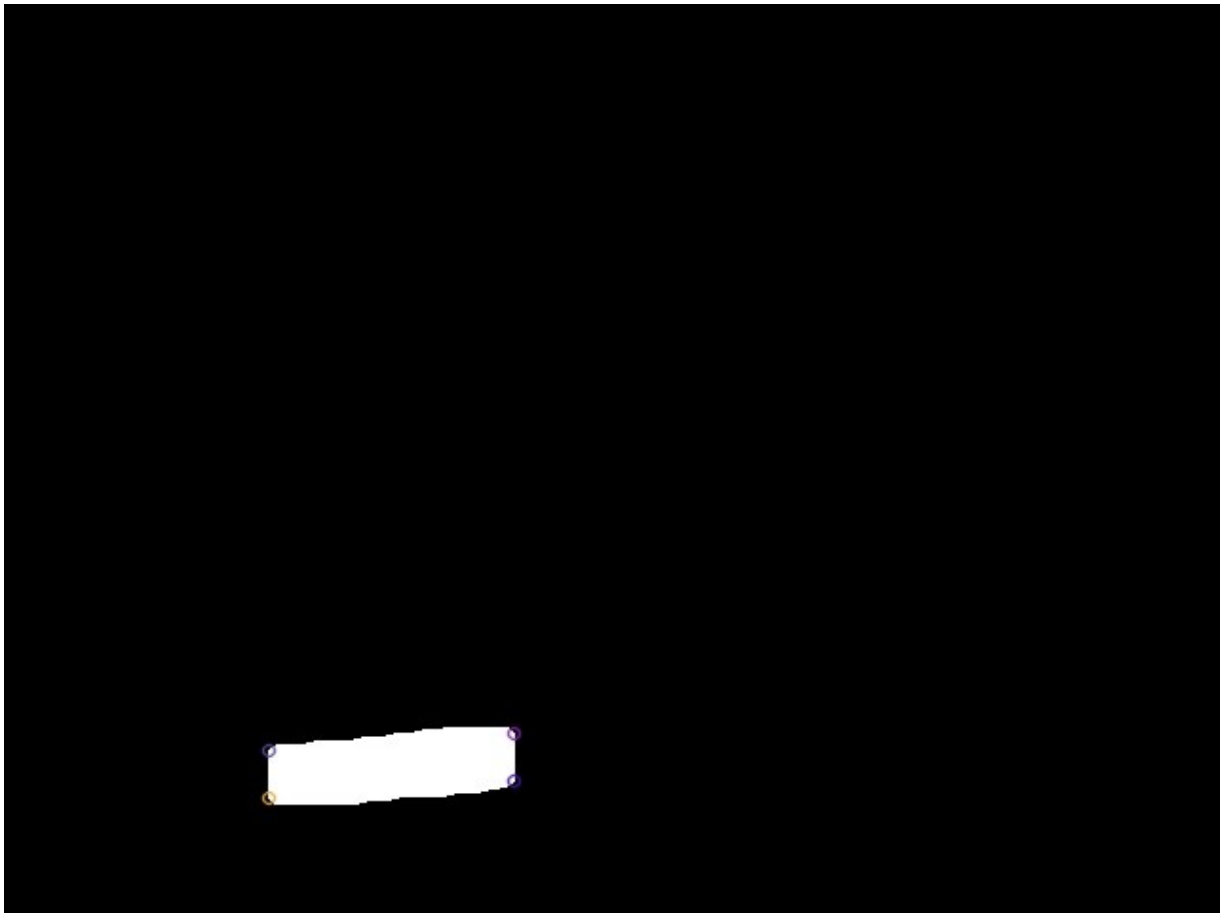

Figura 25:Os quatros pontos das extremidades

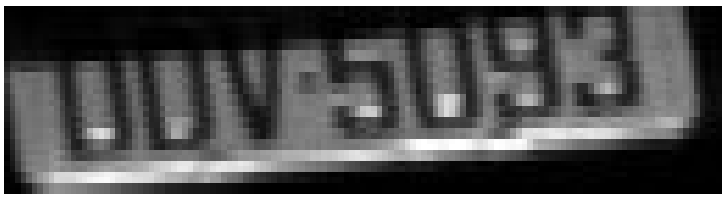

Figura 26:Placa segmentada

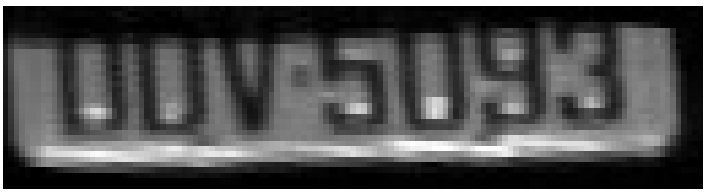

Figura 27: Placa rotacionada

5 – Resultados e discussões

Nesta seção, são discutidos dois resultados. O primeiro é um teste preliminar feito com o operador Canny e Transformada de Hough. O segundo é o teste usando a abordagem com operadores morfológicos e detecção de quinas.

5.1 Testes com operador Canny e Transformada de Hough.

Depois desse resultado, tentou-se reavaliar os parâmetros, porem sem sucesso. Então se decidiu fazer vários laços de repetição para testar uma grande quantidade de combinações e avaliar se método seria viável. Os parâmetros utilizados estão na Tabela 1.

lowThreshold (canny)	1	50	100	150	200	300			
highThreshold (canny)	1	50	100	150	200	300			
kernel_size (canny)	3	5	7						
threshold (HoughLines)	5	10	15	20	30	50	80	100	150
minLineLength (HoughLines)	2	3	4	5	6	7	8	9	10
maxLineGap (HoughLines)	2	3	4	5	6	7	8	9	10

Tabela 1: Parâmetros para a tranformada de Hough e operador Canny.

Alguns dos resultados apresentaram melhora em relação aos resultados anteriores, porém eles ainda não eram satisfatórios. Na Figura 28, encontram-se algumas imagens dos resultados.

Para validar a eficiência desta a abordagem, foi usada uma imagem com um fundo preto e quatro linhas brancas formando um retângulo, e rodou-se o algoritmo para analisar o resultado neste cenário mais simples e com menos ruído. Como o algoritmo não conseguiu identificar as quatro linhas, esse método foi descartado. Na Figura 29 encontra-se a imagem original e um dos resultados. O código fonte dessa tentativa encontra-se no Anexo 1.

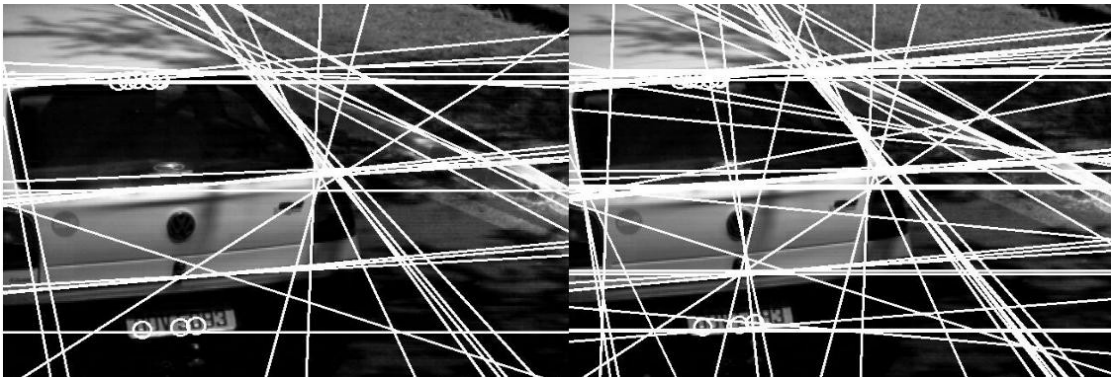

Figura 28: Novos resultados

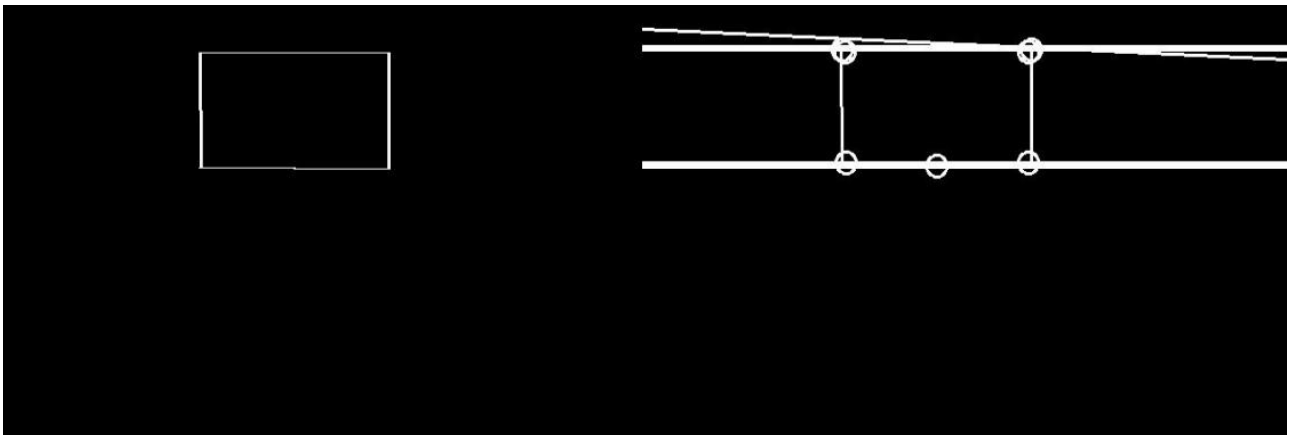

Figura 29:Quadrado simples e resultado de execução

5.2 Testes com operadores morfológicos e detecção de quinas

Como o objetivo era avaliar experimentalmente o algoritmo de detecção de placas, foram realizados testes envolvendo as imagens em condições reais, descritas no tópico 4.1. O objetivo é se aproximar o máximo da situação real.

A Tabela 2 mostra que foram testadas 100 imagens e apenas 40% das imagens tiveram um resultado satisfatório. O tempo de processamento foi para todas as imagens foi de menos de 1 minuto, considerando a leitura ate a escrita da placa segmentada em um computador com Core i5 2.53GHz com 4GB de RAM. Os métodos presentes no OpenCV apresentam um tempo satisfatório, então não se fez necessário otimizá-los, mas podem ser aperfeiçoados para melhorar o desempenho do processo de localização das placas.

Algoritmo	Numero de placas testadas	Sucesso	Percentual de sucesso
Segmentador de placas	100	40	40%

Tabela 2: Estatística de sucesso

A Figura 30 mostra imagens que foram bem localizadas. Porem a Figura 31 mostra algumas imagens que não foram bem sucedidas. Esses erros ocorreram devido aos operadores morfológicos não conseguirem eliminar todo o ruído presente na imagem, então ao buscar os quatro pontos das extremidades, segmentou-se uma região muito grande, a qual não fazia parte apenas a placa.

Figura 30:Placas segmentadas com sucesso

Figura 31:Placas mal localizadas

Outro problema que o algoritmo encontrou foi em relação aos adesivos no carro. Por várias vezes o adesivo prevaleceu em relação à placa do veiculo (Figura 32).

Figura 32: Captura de adesivos

Existiram também situações onde o algoritmo não identificou nada. A operação morfológica de erosão não deixou nenhuma região para ser identificada, ocasionando que não restava nenhuma região candidata.

As imagens que foram melhor identificadas foram as que tinham maior luminosidade, facilitando muito a aplicação dos operadores morfológicos. Algumas delas apresentam sombra parcial na placa, tornando a identificação um pouco difícil. Acredita-se que uma mudança dos parâmetros resulta no reconhecimento desse tipo de situação, contudo, o resultado da detecção em imagens bem iluminadas seria prejudicado. Este resultado sugere que a utilização de um método de reparametrização adaptativa baseado no nível de iluminação possa melhorar a detecção nesta base de imagens. Encontraram-se também placas com sombreamento parcial, dificultando a execução do algoritmo (Figura 33).

Dentre as 100 imagens testadas, 66 eram bem iluminadas. Considerando apenas estas imagens, algoritmo obteve 60.1% de acerto.

Figura 33: Carro com sombra parcial na placa

6 - Conclusão

Esse trabalho apresentou uma bibliografia relacionada à área de Processamento de Imagens Digitais, Visão Computacional e Reconhecimento de Padrões, com intuito de definir as etapas necessárias para ser desenvolvido o sistema de reconhecimento de placas de veículos. O objetivo principal deste trabalho foi desenvolver uma das fases deste sistema: a detecção da placa numa imagem contendo a parte traseira de um veículo. Para implementação desta fase, utilizou-se o OpenCV, que é uma biblioteca rica em recursos e em constante expansão. Sua comunidade é ativa, facilitando a utilização e o aprendizado dessa ferramenta.

O estudo pretendia abarcar todas as etapas do processo de reconhecimento de identificação, porém realizou-se apenas a detecção da placa. Os testes mostraram que a utilização de Transformada de Hough com operador Canny não produziram resultados satisfatórios. Contudo, um algoritmo com uso de operadores morfológicos e detecção de quinas obteve 40% de sucesso, um numero pequeno, já que na literatura encontra-se trabalhos que obtém mais de 80% de sucesso. Também notou-se que este algoritmo com os parâmetros utilizados tem resultados bons somente para alto nível de luminosidade das imagens de entrada. Dentre as 100 imagens testadas, 66 eram bem iluminadas. Considerando apenas estas imagens, algoritmo obteve 60.1% de acerto. Uma mudança nos parâmetros usados nos operadores morfológicos pode ser feita para trabalhar com imagens com baixo nível de luminosidade, contudo, para o sistema se comportar bem em qualquer situação, deve-se utilizar um método de mudança adaptativa destes parâmetros, dada a luminosidade da imagem de entrada.

Um ponto importante a ser notado é a diferença nas bases de dados utilizadas pelos autores, Esse trabalho faz uso de imagens capturadas por radares de transito, estas mais próximas de uma situação real. Já alguns trabalhos, os próprios autores fazem a captura e criam seu banco de imagens.

Como trabalhos futuros, pretende-se usar as técnicas descritas em outras referências para produzir um sistema completo de reconhecimento de placas e fazer um estudo de viabilidade para utilização de algoritmos com esta finalidade em um sistema embarcado.

Referencias

ARAGÃO, Maria Géssica dos Santos. **Reconhecimento e segmentação de caracteres em placas de veículos usando análise de projeção e redes neurais.** 2013. 10 f. TCC (Graduação) - Curso de Ciência da Computação, Universidade Tiradentes, Aracaju, 2013.

BIANCHI, Marcelo Franceschi de. **Extração de características de imagens de faces humanas através de wavelets, PCA e IMPCA.** 2006. Disponível em: <http://www.teses.usp.br/teses/disponiveis/18/18133/tde-10072006-002119/en.php>. Acesso em: 10 abr. 2006.

CAMPOS, Tatiane Jesus de. **Reconhecimento de Caracteres Alfanuméricos de Placas em Imagens de Veículos.** 2001. 120 f. Dissertação (Mestrado) - Curso de Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2001.

CARVALHO, Jonh Edson Ribeiro de. **Uma Abordagem de Segmentação de Placas de Automóveis Baseada em Morfologia Matemática.** 2006. 101 f. Dissertação (Mestrado) - Curso de Computação, Universidade Federal Fluminense, Rio de Janeiro, 2006.

CONCI, Aura; CARVALHO, John; RAUBER, Thomas. A Complete System for Vehicle Plate Localization, Segmentation and Recognition in Real Life Scene. **IEEE Latin America Transactions,** Niterói, v. 7, n. 5, p.497-506, set. 2009.

CONCI, Aura; MONTEIRO, Leonardo Hiss. RECONHECIMENTO DE PLACAS DE VEÍCULOS UTILIZANDO PROCESSAMENTO DE IMAGEM. **Engevista,** Rio de Janeiro, v. 5, n. 10, p.31-43, dez. 2003. Disponível em: <http://www.uff.br/engevista/seer/index.php/engevista/article/view/117/20>. Acesso em: 16 fev. 2016

CORREIA, Ronaldo Dias. **VISÃO COMPUTACIONAL.** Barra Mansa: Slide, 2012. 13 slides, color.

Deans, S. R, The Radon Transform and Some of Its Applications 1983.

DELAI, Riccardo Luigi; COELHO, Alessandra Dutra. **VISÃO COMPUTACIONAL COM A OPENCV – MATERIAL APOSTILADO E VEÍCULO SEGUIDOR AUTÔNOMO.** São Caetano do Sul: Slide, 2012. 6 slides, color.

Duda, R. O.; Hart, P. E. Use of the Hough Transformation to detect lines and curves in pictures. Graphics and Image Processing, 2001.

FERREIRA, Luismar Sebastião; SOARES, Luciano Pereira. Reconhecimento Automático de Placas Veiculares. In: WORKSHOP DE VISÃO COMPUTACIONAL, 8., 2012, Goiana. **Anais... .** Goiana: Bdbcomp, 2012. v. 8, p. 53 – 62.

GONZALEZ, R. C; WOODS, R. E. **Processamento de imagens digitais**. São Paulo: Edgard Blucher, 2000.

JAMUNDÁ, T. **Reconhecimento de Formas: A Transformada de Hough.** 2000. Disponível em: < https://www.inf.ufsc.br/~visao/2000/Hough/ >Acessado em: 15/01/2016

KINJO, P. & SOARES, T. OpenCV Rastreando Objetos, 2013. Disponível em: http://www.academia.edu/4654828/OpenCV_Rastreando_Objetos> Acessado em 13/11/2015.

LIMA, Warlem Lucio, REIS, Augusto Sergio Teixeira. CONGRESSO NACIONAL DE INICIAÇÃO CIENTIFICA, 13., 2013, Campinas. **APLICAÇÃO PARA RECONHECIMENTO DE GESTOS, USANDO A CAMADA INFERIOR OPENCV.** Campinas: A, 2013. 11 p. CD-ROM.

MARENGONI, Maurício; STRINGHINI, Denise. Tutorial: Introdução à Visão Computacional usando OpenCV. **Revista de Informática Teórica e Aplicada,** Porto Alegre, v. 16, n. 1, p.125-160, abr. 2009.

MOREIRA, Breno Barbosa. **A Transformada de Hough aplicada à Difração de Elétrons Retroespalhados.** 2012. 71 f. Monografia (Especialização) - Curso de Matemática,

Universidade Federal de Minas Gerais, Belo Horizonte, 2012.

NASCIMENTO, Rafaella Cristianne Alves do. **Localização de Robôs Móveis em Ambientes Fechados em Tempo Real Utilizando Câmeras Montadas no Teto.** 2014. 60 f. Dissertação (Mestrado) - Curso de Programa de Pós-graduação em Engenharia Elétrica e de Computação, Universidade Federal do Rio Grande do Norte, Natal, 2014.

OLIVEIRA, Leonardo Augusto; GONZAGA, Adilson. Localização, Segmentação e Reconhecimento de caracteres em Placas de Automóveis. **Avanços em Visão Computacional,** [s.l.], p.283-302, 18 dez. 2012. Omnipax Editora. DOI: 10.7436/2012.avc.15.

MOREIRA, Breno Barbosa. **A Transformada de Hough aplicada à Difração de Elétrons Retroespalhados.** 2012. 71 f. Monografia (Especialização) - Curso de Matemática Para Professores Com Ênfase em Cálculo, Universidade Federal de Minas Gerais, Belo Horizonte, 2012.

PASSOS, Yuri Tavares dos. **Uma Abordagem Bio-Inspirada para o Problema de Reconhecimento de Placas de Licença.** 2008. 76 f. TCC (Graduação) - Curso de Ciência da Computação, Universidade Federal de Sergipe, Aracaju, 2008.

PEREIRA, Jorge Fernando Silva. **Localização de Placas de Licenciamento Veicular em tempo real utilizando OpenCV com CUDA.** 2014. 15 f. TCC (Graduação) - Curso de Análise e Desenvolvimento de Sistemas, Instituto Federal de Educação, Ciência e Tecnologia da Bahia, Salvador, 2014.

QUINTANS, Maria; CARNEVALI, Felipe; ESCOLA, João. Implementação de um software para reconhecimento facial utilizando HaarCascade através da OpenCV. In: ESCOLA REGIONAL DE INFORM TICA DO RIO DE JANEIRO, 1., 2010, Rio de Janeiro. **Anais... .** Rio de Janeiro: UFRJ, 2010. v. 1, p. 72 - 75.

Russell, S. e Norvig, P. (2004) “Inteligência Artificial”, Rio de Janeiro: Elsevier.

SOUSA, Kelly Aparecida Oliveira. **USO DE VISÃO COMPUTACIONAL EM DISPOSITIVOS MÓVEIS PARA AUXÍLIO À TRAVESSIA DE PEDESTRES COM DEFICIÊNCIA VISUAL.** 2013. 85 f. Dissertação (Mestrado) - Curso de Programa de Pós Graduação em Engenharia Elétrica, Universidade Presbiteriana Mackenzie, São Paulo,

2013.

SOUZA, Caio Felipe de; CAPOVILLA, Felipe; ELEOTÉRIO, Thiago Castro. **VISÃO COMPUTACIONAL.** Taquaritinga: A, 2010. 51 p.

STEMMER, Marcelo R.; ORTH, Alexandre; ROLOFF, Márcio L.; DESCHAMPS, Fernando; PAVIM, Alberto X. Apostila de sistemas de visão. Disponível em: <http://s2i.das.ufsc.br/harpia/downloads/apostila-sistemas-visao.pdf>. Acessado em 13/11/2015.

TRENTINI, Vinicius Bergoli; GODOY, Lucas Antonio Toledo; MARANA, Aparecido Nilceu. Reconhecimento Automático de Placas de Veículos. In: WORKSHOP DE VISÃO COMPUTACIONAL, 6., 2010, Presidente Prudente. **Anais.** Presidente Prudente: A, 2010. p. 267 - 272.

WANG, X.; Bertand, G.; An algorithm for a generalized distance transformation based on Minkowski operation. 9th ICPR, pag 1163 – 1167, 1988.

Anexo 1

```
package placaVeicular;

import java.awt.image.BufferedImage;
import java.io.File;
import javax.imageio.ImageIO;
import org.opencv.core.Core;
import org.opencv.core.Mat;
import org.opencv.highgui.Highgui;

public class entrada {
	public static void main(String[] args) {

		try {
			// carregando a biblioteca
			System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
			System.out.println("começando");
			int[] mm = { 10,100 };
			int[] nn = { 10,100,300 };
			int[] aa = { 1 };
			int[] bb = { 5};
			int[] cc = { 5,10,15 };
			int[] dd = { 3 };
			int[] ee = {200};
			for (int m = 0; m < mm.length; m++) {
			for (int n = 0; n < nn.length; n++) {
			for (int a = 0; a < aa.length; a++) {
			for (int b = 0; b < bb.length; b++) {
			for (int c = 0; c < cc.length; c++) {
			for (int d = 0; d < dd.length; d++) {
				for(int e = 0; e < ee.length;e++){
				// carregando a imagem original, ja em escala de cinza
				Mat imageGray =
```

```
Highgui.imread("/home/biao/workspace/placaVeicular/HI00007076.jpg",Highgui.CV_LOAD_IMA
GE_GRAYSCALE);
                                BufferedImage temp = reconhecimento
                                                .reconhecedorPlaca(imageGray, mm[m], nn[n],
                                                                aa[a], bb[b], cc[c], dd[d],ee[e]);
                                File outputfile = new File("final" + aa[a]
                                                + "x" + bb[b] + "x" + cc[c] + "x" + dd[d] + "x" +ee[e]
+"x"
                                                + ".jpg");
                                ImageIO.write(temp, "jpg", outputfile);
                        }}}}}}}

                } catch (Exception e) {
                        System.out.println("Error: " + e.getMessage());
                }
        }
}


package placaVeicular;

import java.awt.image.BufferedImage;

import org.opencv.core.Core;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.Scalar;
import org.opencv.core.Size;
import org.opencv.highgui.Highgui;
import org.opencv.imgproc.Imgproc;

public class reconhecimento {

        public static int ver =0;
        public static BufferedImage reconhecedorPlaca(Mat matrix,int m,int n,int a,int b,int c,int d,
```

```
int e ) {

        int cols = matrix.cols();
    int rows = matrix.rows();

    int elemSize = (int)matrix.elemSize();

    byte[] data = new byte[cols * rows * elemSize];

    int type;

                                Mat imagemResultanteCanny = new
Mat(matrix.rows(),matrix.cols(),CvType.CV_8UC1);

    Imgproc.blur(matrix, imagemResultanteCanny,new Size(3,3));
    Imgproc.Canny(imagemResultanteCanny, imagemResultanteCanny, m, n, 3, true);
    Highgui.imwrite("canny"+m+"x"+n+"x3ver"+ver+".jpg", imagemResultanteCanny);
    ver++;

    Mat lines = new Mat();
    Imgproc.HoughLinesP(imagemResultanteCanny, lines, 2, Math.PI/180, b, c, d);
    System.out.println(lines.size());


    for (int x = 0; x < lines.cols(); x++)
    {
        double[] vec = lines.get(0, x);

        org.opencv.core.Point start = new org.opencv.core.Point();
      start.x = (int) vec[0];
      start.y = (int) vec[1];

        org.opencv.core.Point end = new org.opencv.core.Point();
      end.x = (int) vec[2];
      end.y = (int) vec[3];
```

```
                Core.line(matrix, start, end, new Scalar(255,255,255), 2);
            }

            Mat imagemResultanteCorner = new Mat();

            Imgproc.cornerHarris(imagemResultanteCanny, imagemResultanteCorner, 2, 3, 0.04, 1);

            Mat n_norm = new Mat();
                Core.normalize(imagemResultanteCorner,  n_norm,0,255,Core.NORM_MINMAX,
CvType.CV_32FC1);

                Mat s_norm = new Mat();
                Core.convertScaleAbs(n_norm, s_norm);

          Highgui.imwrite("corner"+m+"x"+n+"x"+a+"x"+b+"x"+c+"x"+d+"x"+e+"ver"+ver+".jpg",
s_norm);
            for(int y = 0; y < imagemResultanteCorner.height(); y++){

               for(int x = 0; x < imagemResultanteCorner.width(); x++){

                if(s_norm.get(y,x)[0] > e ){
                   Core.circle(matrix, new org.opencv.core.Point(x,y),10, new Scalar(255), 2,8,0);
                }
               }
            }

            switch (matrix.channels()) {
              case 1:
                type = BufferedImage.TYPE_BYTE_GRAY;
                break;
              case 3:
                type = BufferedImage.TYPE_3BYTE_BGR;
                byte bI;
                for(int i=0; i<data.length; i=i+3) {
```

```
                bI = data[i];
                data[i] = data[i+2];
                data[i+2] = bI;
              }
              break;
            default:
              return null;
          }
        matrix.get(0, 0, data);
    BufferedImage image2 = new BufferedImage(cols, rows, type);
    image2.getRaster().setDataElements(0, 0, cols, rows, data);
    return image2;

  }
}
```