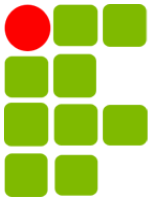


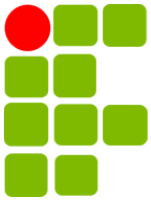
Instituto Federal de Educação, Ciência e Tecnologia da Paraíba
Campus Campina Grande
Coordenação do Curso Superior de Tecnologia em Telemática

DETECÇÃO AUTOMÁTICA DE PLACAS DE VEICULOS IRREGULARES EM BLITZES

DIEGO ALVES DUARTE

Orientador: Fagner de Araujo Pereira

Campina Grande, outubro de 2018



Instituto Federal de Educação, Ciência e Tecnologia da Paraíba
Campus Campina Grande
Coordenação do Cursos Superior de Tecnologia em Telemática

DETECÇÃO AUTOMÁTICA DE PLACAS DE VEICULOS IRREGULARES EM BLITZES

DIEGO ALVES DUARTE

Monografia apresentada à Coordenação do Curso de Telemática do IFPB - Campus Campina Grande, como requisito parcial para conclusão do curso de Tecnologia em Telemática.

Orientador: Fagner de Araujo Pereira

Campina Grande, outubro de 2018

DETECÇÃO AUTOMÁTICA DE PLACAS DE VEICULOS IRREGULARES EM BLITZES

DIEGO ALVES DUARTE

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Orientador

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Membro da Banca

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Membro da Banca

Campina Grande, Paraíba, Brasil

Outubro/2018

Dedico este trabalho a minha esposa, Thamirys Islanny.

Não só isso, mas também nos gloriamos nas tribulações, porque sabemos que a tribulação produz perseverança, E a paciência a experiência, e a experiência a esperança.

Romanos 5:3,4

Agradecimentos

Agradeço primeiramente a Deus pelo dom da vida, pela força que Ele me proporcionou durante essa trajetória fazendo com que eu superasse todas as adversidades que ocorreram pelo caminho. A minha esposa Thamirys que sempre me incentivou a continuar, a não desistir, sempre mostrando que eu posso ir além. Aos meus pais, Assis e Lena, toda gratidão.

Resumo

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.

Palavras-chave: XXXXXXXXXXXXXXXXXXX.

Abstract

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.

Keywords: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.

Sumário

Lista de Abreviaturas

API	*Application Programming Interface*
BPL	*Bandpass Liftering* (Filtragem passa-faixa)
CFG	*Context-Free Grammar* (Gramática Livre de Contexto)
CORDIC	*COordinate Rotation DIgital Computer*
DARPA	*Defense Advanced Research Projects Agency*
DTW	*Dynamic Time Warping* (Alinhamento Dinâmico no Tempo)

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

As imagens são produzidas por uma variedade de dispositivos físicos, tais como câmeras e vídeo câmeras, equipamentos de radiografia, microscópios eletrônicos, magnéticos e de força atômica, radares, equipamento de ultra-som, entre vários outros. A produção e utilização de imagens podem ter diversos objetivos, que vão do puro entretenimento até aplicações militares, médicas ou tecnológicas. O objetivo da análise de imagens, seja por um observador humano ou por uma máquina, é extrair informações úteis e relevantes para cada aplicação desejada.

> Em geral, a imagem pura, recém adquirida pelo dispositivo de captura, necessita de transformações e realces que a torne mais adequada para que se possa extrair o conteúdo de informação desejada com maior eficiência. O Processamento Digital de Imagens (PDI) é uma área da eletrônica/teoria de sinais em que imagens são convertidas em matrizes de números inteiros, sendo que cada elemento desta matriz é composto por um elemento fundamental: o pixel (uma abreviação de picture element). A partir desta matriz de pixels que representa a imagem, diversos tipos de processamento digital podem ser implementados por algoritmos computacionais. A aplicação destes algoritmos realiza as transformações necessárias para que se possa, por exemplo, obter uma imagem com os realces pretendidos ou extrair atributos ou informações pertinentes [Andrade e Márcio 2003, P.1].

Sistemas de detecção e processamento de imagens podem ser utilizados em vários campos do nosso dia a dia. Com o avanço da tecnologia, dos meios de processamento e do conjunto de bibliotecas de softwares Open Source, processar imagens estáticas ou não e extrair delas informações úteis tem sido um campo bastante amplo nas pesquisas acadêmicas que trouxeram e ainda trazem benefícios para a sociedade. O reconhecimento de um placa de carro, por exemplo, é um dos pequenos desafios que existem na área de visão computacional, devido aos diferentes tipos, formas, cores, ângulo e iluminação.

1.1 Justificativa e Relevância do Trabalho

Os índices de roubos a veículos só crescem a cada dia em nossa sociedade, tornando esse um problema de segurança pública. Em apenas um ano, o número de veículos roubados

aumentou 59% entre 2016 e 2017 - dados da Secretaria de Segurança e Defesa Social (Seds) da Paraíba. Devido ao alto índice de roubo de veículos, as blitzes, que tem como objetivo combater qualquer tipo de ilegalidade, tornaram-se um dos principais meios de combate a esse tipo de problema. Em nossa cidade é comum vermos essas blitzes sendo efetuadas em locais bem específicos e com êxito conseguindo apreender um grande número de veículos com algum tipo de irregularidade ou não. Porém, como o quadro de agentes numa blitz não é proporcional ao número de veículos que ali trafega alguns veículos conseguem passar pela blitz mesmo sem terem sido vistoriados. Nesse caso, muitas vezes, veículos irregulares.

Com esse problema do alto índice de roubos a veículos, acrescido do baixo efetivo de profissionais, posicionando alguns metros antes da blitz, uma câmera irá detectar a placa do veículo verificando automaticamente em um banco de dados se o mesmo tem alguma restrição, podendo diminuir ou até mesmo eliminar a chance de um veículo com restrição passar sem ser autuado.

1.2 Objetivos

1.2.1 Objetivo Geral

Desenvolver um aplicação que identifique uma placa veicular por vídeo imagem utilizando uma câmera de baixo custo, adotando códigos Open Source, técnicas de filtragem e detecção de padrões.

1.2.2 Objetivos Específicos

- Identificar as melhores práticas de detecção de uma imagem.
- Aplicar as diferentes funcionalidades da biblioteca OpenCV e da OCR do Google.
- Conhecer um pouco da visão computacional, captura de imagens, pré-processamento, segmentação, binarização, limiarização filtros e reconhecimento de padrões.

  itemize

1.3 Metodologia

Em um banco de dados prévio contendo imagens de placas posicionadas da melhor forma possível para identificação da mesma. Os resultados da extração da numeração da placa irão ser consultados em um banco de dados local-fictício.

1.4 Organização do Documento

Este trabalho está dividido em 5 capítulos. No capítulo 1 teremos a Introdução onde apresentamos o tema e contextualizamos a problemática seguido dos objetivos gerais e específicos a serem atingidos e a metodologia utilizada. No capítulo 2 apresentamos as principais etapas para o processamento digital de uma imagem. No capítulo 3 apresentamos a técnica utilizada juntamente com o algoritmo para o processamento da imagem das placas. Logo após no capítulo 4 apresentaremos os resultados obtidos e posteriormente no capítulo 5 as propostas para continuidade deste trabalho.

Capítulo 2

Fundamentação Teórica

2.0.1 O que é o processamento de Imagem.

> Uma das primeiras aplicações remonta ao começo deste século, onde buscavam-se formas de aprimorar a qualidade de impressão de imagens digitalizadas transmitidas através do sistema Bartlane de transmissão de imagens por cabo submarino entre Londres e Nova York. Os primeiros sistemas Bartlane, no início da década de 20, codificavam uma imagem em cinco níveis de intensidade distintos. Esta capacidade seria expandida já em 1929, para 15 níveis, ao mesmo tempo em que era desenvolvido um método aprimorado de revelação de filmes através de feixes de luz modulados por uma fita que continha informações codificadas sobre a imagem. [Marques e Vieira 1999].

Figura 2.1: *A figura acima foi transmitida dessa maneira e reproduzida numa impressora telegráfica contendo caracteres para a simulação de padrões de tons intermediários. Fonte: Google 2018*

O processamento de imagem é qualquer forma de se processar um dado, onde entrada e saída sejam imagens, estáticas ou frames de um vídeo. A área de processamento de imagem vem sendo bastante discutida e difundida por permitir e viabilizar um grande número de aplicações. Devemos diferenciar o tratamento de imagens, o qual se preocupa somente na manipulação de figuras para representá-las posteriormente, do

processamento de imagem, o qual podemos chamá-lo de um estágio para novos processamento de dados, como aprendizagem de máquina ou reconhecimento de padrões. Hoje temos toda essa tecnologia na palma de nossas mãos, ao fazermos um reconhecimento biométrico posicionando a mão em um leitor de caixa eletrônico para transações bancárias.

Nos dias atuais, em contrapartida do que foi citado por Hugo e Vieira Neto, conseguimos codificar uma imagem colorida em 256 níveis de cinza, trabalhá-la da forma que desejamos e decodificá-la obtendo uma imagem rica em detalhes.

Figura 2.2: *Retirada da informação da numerção da placa de um veículo.*
Fonte: Google 2018

2.1 Etapas do processamento de imagem.

> O processamento de imagens tem como funções facilitar a visualização da imagem ou adequá-la para análises quantitativas através de correções de defeitos ou realces das regiões de interesse nas imagens; e a extração e tratamento de dados quantitativos, feitos pelo próprio computador. [Gomes 2001].

2.1.1 Aquisição da imagem

Nessa etapa, recorremos ao objeto que irá capturar a imagem, uma câmera, que é um dispositivo eletrônico. As câmeras digitais utilizam o circuito CCD e um sistema de digitalização interno para a geração de uma imagem digital, a qual pode ser transferida diretamente para um computador. Uma imagem digital pode ser descrita como uma matriz matemática, cujo os índices linha e coluna representam um ponto na imagem. Cada coordenada, linha e coluna, representa um elemento da matriz, que são os pixels, possuindo um valor que corresponde ao nível de cinza, ou a cor, naquele ponto da imagem. Por fornecer uma descrição global da aparência de uma imagem, uma das formas mais utilizadas para apresentar a intensidade de um pixel em uma imagem é o histograma.

"O gráfico do histograma é plotado com a intensidade dos pixels para 256 tons no eixo horizontal e, no eixo vertical, a probabilidade de ocorrência dos tons de cinza na imagem. " [Gonzales 2002].

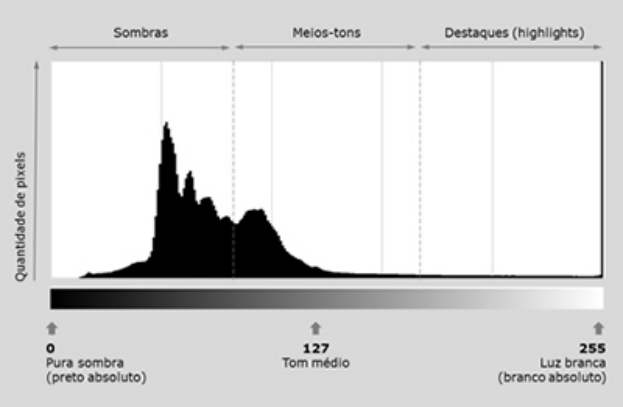


Figura 2.3: *Histograma de uma imagem.*
Fonte:Google 2018

2.1.2 Pré-Processamento

A partir do bloco de aquisição de imagens, a maioria das funções realizadas em cada etapa do PADI(Processamento e Análise Digital de Imagens) pode ser implementada em software, sendo expressa em forma de algoritmo.

"O pré-processamento tem como objetivo melhorar a imagem, corrigindo algum defeito proveniente de sua aquisição e/ou realçando detalhes importantes para a análise." [Gomes 2001].

Se o procedimento de captura for realizada de forma cautelosa, em boas condições de iluminação e ângulo por exemplo, não se faz necessário muitas operações de correção nas imagens adquiridas.

2.1.3 Segmentação

A segmentação é a etapa crítica do fluxograma de PADI, onde se obtém a imagem a partir da qual alguma informação será extraída. O seu principal objetivo é subdividir uma imagem em regiões ou objetos de interesse, formados pelo agrupamento de pixels contíguos.

A imagem de saída da segmentação é uma imagem binária, onde os pixels pretos representam o fundo ou objetos que não são de interesse na imagem, e os pixels brancos constituem os objetos de interesse, os quais serão quantificados, ou vice-versa. [Iglesias 2001].

Existem diversos tipos de algoritmos de segmentação e geralmente são baseados em duas propriedades básicas de valores de cinza: Descontinuidade e Similaridade. A descontinuidade realiza a detecção de pontos isolados e detecção de linhas e bordas na imagem, dividindo a imagem conforme ocorre mudanças bruscas nos níveis de cinza, enquanto a similaridade se baseia na ideia de limiarização, crescimento de regiões e, divisão e fusão de regiões.

2.1.4 Pós-Processamento

Muitas vezes os resultados obtidos da etapa de segmentação, não estão adequados para que os grupos de pixels segmentados sejam representados e descritos em termo de suas características nas etapas posteriores, sendo necessário um pós-processamento.

> A separação de objetos que se tocam, a eliminação de objetos de que não se deseja extrair nenhuma informação e o agrupamento de objetos para a formação de objetos mais complexos são exemplos de procedimentos realizados na etapa de pós-processamento. Estes procedimentos são realizados através de operações lógicas e morfológicas [Gomes 2001].

A fase de parametrização identifica, ou seja, rotula cada um dos objetos segmentados calculando alguns parâmetros (pré-determinados), como exemplo de algum parâmetro podemos citar o perímetro ou a área de uma determinada forma).

2.1.5 Extração de atributos

Nessa etapa inicia-se a análise da imagem propriamente dita. São realizadas medidas na imagem segmentada ou pós processada, ou até mesmo na imagem em tons de cinza. Com essas medidas, os grupos de pixels são caracterizados por atributos característicos, gerando dados quantitativos para o objetivo final. Podemos dividir a extração de atributos em dois tipos de medidas: Medidas de campo e medidas de região.

> As medidas de campo se referem ao campo como um todo, como na medição de número de objetos, área total dos objetos e fração de área, gerando como resultado apenas um valor por medida. As medidas de região se referem aos objetos individualmente, ou seja, é extraído um parâmetro de cada objeto na imagem, como por exemplo, tamanho, forma e posição das partículas. [Paciornick 2001].

2.1.6 Reconhecimento de padrões e classificação

Nesta última etapa do processo iremos classificar de forma automática os objetos a partir de informações encontradas na imagem, geralmente com um banco de dados

previamente estabelecido. A classificação parte da premissa que a similaridade entre objetos implica que eles possuam características similares, formando classes/agrupamentos.

2.2 OCR (Reconhecimento Ótico de Caracteres)

OCR significa Optical Character Recognition ou Reconhecimento Ótico de Caracteres, é o processo pelo qual um computador ou dispositivos eletrônico com uma câmera consegue ler/extrair o texto contido em uma imagem. O reconhecimento ótico humano é feito através do glob ocular, os olhos. Enquanto este reconhecimento é feito pelos olhos (entrada), a interpretação (processamento) varia de pessoa para pessoa de acordo com muitos fatores como qualidade da câmera do dispositivo, porcentagem de ruído das imagens, formato dos caracteres de entrada, entre outros.

As primeiras versões de OCR eram simples e requeriam calibragem do sistema. Esta calibragem constituía-se da prévia programação de imagens associadas a cada caractere e utilizando-se de apenas um tipo de fonte. Implementações atuais abordando OCR abrangem tanto caracteres do alfabeto latino quanto caracteres orientais como o chinês, japonês, etc. A grande maioria dessas implementações são de código aberto e possuem grande quantidade de colaboradores em todo mundo, porém, ainda apresemtam alguns problemas de integração com os diferentes sistemas operacionais. O OCR é utilizado para a entrada automática de dados em um computador, armazenamento, compartilhamento ou processamento. Um sistema de OCR é composto basicamente de: Escaneamento Ótico, Localização e Segmentação, Pré-processamento, Extração de características, Classificação e Pós-processamento, e sua funcionalidade pode ser abstraida de duas formas: Existem duas abordagens para o reconhecimento do caracter. A primeira consiste em comparar o caracter com uma base prévia de símbolos e, então, fazer o reconhecimento desses padrões. E na segunda abordagem, cada característica do texto (curvas, linhas retas e outros "pedaços" que diferenciam cada caractere) é extraída, compondo seu formato e convergindo para a identificação que parece ser a mais próxima.

2.2.1 Biblioteca OpenCV - Pytesseract-OCR.

O OpenCV é uma biblioteca multiplataforma que segui o modelo de licença BSD da Intel. A biblioteca possui módulos de Processamento de Imagens e Vídeo I/O, Estrutura de dados, Álgebra Linear, GUI (Interface Gráfica do Usuário) básica com sistema de janelas independentes, controle de mouse e teclado, além de mais de 350 algoritmos de Visão computacional como: filtros de imagem, calibração de câmera, reconhecimento de objetos, análise estrutural e outros. Foi iniciado pela Intel em 1999 por Gary

Bradsky e o primeiro lançamento foi lançado em 2000. Atualmente, o OpenCV suporta muitos algoritmos relacionados à visão de computador e à aprendizagem de máquinas e está expandindo dia a dia. O Python-tesseract é uma ferramenta de reconhecimento óptico de caracteres (OCR) para python, o que faz dele uma ferramenta indispensável na hora de extrair texto de imagens. Além de ser um excelente pacote para o mecanismo Tesseract-OCR do Google, o mesmo é capaz de ler todos os tipos de imagem suportados pela biblioteca do python. O Tesseract foi originalmente desenvolvido na Hewlett-Packard Laboratories Bristol e na Hewlett-Packard Co, Greeley Colorado entre 1985 e 1994, com algumas mudanças feitas em 1996 para suportar ambiente Windows, e algumas C ++ em 1998. Em 2005 a Tesseract estava com seu codigo aberto fornecido pela HP. De 2006 até os dias atuais é desenvolvido pelo Google. A versão estável mais recente (baseada no LSTM) é a 4.0.0, lançada em 29 de outubro de 2018. O último código-fonte 4.0 está disponível no branch master do GitHub. A versão mais recente do 3.5 é 3.05.02, lançada em 19 de junho de 2018. O código-fonte mais recente para o 3.05 está disponível no branch 3.05 do GitHub.

O funcionamento do Tesseract, utiliza-se de uma entrada de dados (imagem) na qual é binarizada, caso já não esteja pré-processada. Separando os objetos em blocos de texto, linhas e figuras onde esses objetos são analisados delimitando linhas em torno dos objetos referentes ao texto

Capítulo 3

Desenvolvimento

3.1 Aquisição e tratamento da imagem.

Todas as funções de processamento das imagens descritas e utilizadas no desenvolvimento desse projeto foram obtidas através da biblioteca OpenCV que está disponível para diversas linguagens de programação. Para a programação deste projeto, foi utilizado a linguagem de programação Python. Python é uma linguagem de programação multiplataforma desenvolvido sob uma licença de código aberto, que é administrada pela Python Software Foundation e aprovada pela OSI20, tornando-o livremente utilizável e distribuível, mesmo para uso comercial.

O primeiro passo para iniciar este trabalho foi a captura ou aquisição das imagens que posteriormente seriam processadas. Foram utilizadas imagens de placas não oficiais(placas ideais, sem ruido, sombra, etc.) meramente para estudos. Porém que se utilizam dos padrões reais, tanto como cor, tamanho e fonte.

Devido a grande complexidade para obtenção de uma imagem perfeita que obdecesse as caracteristicas minimas para um resultado satisfatorio, foi considerado uma imagem com aproximadamente 80kb de tamanho em disco e com 1366x768 de dimensões para servir de imagem de um cenário ideal.

Após obter a imagem, a submetemos por três etapas cruciais para seu melhoramento como também deixá-la apta a passar pelas diferentes funções que exigem tal tratamento. Três filtros serão usados, Gray Scale (Escala de cinza), Thresholding (Limiarização) e Gaussian Blur (desfoque) seguindo essa ordem.

No primeiro passo convertemos a imagem em escala de cinza. Imagens em nível de cinza são diferentes de imagens binárias em preto e branco, que contém apenas duas cores; imagens em nível de cinza podem conter diversos tons de cinza em sua composição dessa forma todos os pixels da imagem possuirão níveis que irão variar de 0 a 255, dependendo da sua intensidade. Quanto mais escuro(próximo do preto) o

pixel, menor o valor, consequentemente quanto mais brilho(próximo do branco) o pixel tiver mais perto de 255 seu valor estará. Para a escala de cinza foram utilizados a função cvtColor da biblioteca OpenCV, passando como parâmetro a constante 21 cv2.COLOR$_B GR2GRAY$.

Figura 3.1: *Imagem em escala de cinza.*
Fonte: Autoria própria.

Com a imagem devidamente em escala de cinza, para a segunda etapa realizamos a binarização, que consiste em criar um limiar entre os níveis de cinza da imagem obtendo assim regiões que evidenciam regiões delimitadas entre o branco e o preto. Para isso a função utilizada da biblioteca OpenCV foi método threshold recebendo os seguintes parâmetros: O primeiro argumento é a imagem de origem, que deve ser uma imagem em escala de cinza, neste caso o resultado da operação anterior. O segundo argumento é o valor do limiar que é usado para classificar os valores de pixel. O terceiro argumento é o maxVal que representa o valor a ser dado se o valor do pixel for maior que (às vezes menores do que) o valor do limite. Para esse resultado do passo 2 da Fig. 3.2, foram utilizados osseguintes parâmetros de (imagem, 129, 255, cv2.THRESH$_B INARY$).

Figura 3.2: *Imagem binarizada.*
Fonte:Autoria própria.

Na terceira etapa de melhoramento da imagem, aplicamos um leve desfoque que ajuda a retirar arestas, a Filtragem Gaussiana ou Glausian Blur, é um filtro de suavização de imagem, mais conhecido como desfoque, aplicando na imagem com um kernel de filtro de passa baixa que é bastante útil para remover os ruídos e conteúdo de alta frequência. Neste processo foi utilizado a função do OpenCV conhecida como GaussianBlur, passando como parâmetros nesse caso o resultado binarizado visto anteriormente. Deve ser especificado uma largura e uma altura do kernel ou matriz, para esse caso foram utilizados um kernel 5x5. Por último é preciso especificar o desvio padrão nas direções X e Y, respectivamente. Se apenas X for especificado, o Y é tomado como igual a X. Se ambos forem dados como zeros, eles são calculados a partir do tamanho do kernel, no caso deste projeto utilizamos o valor 0.

Figura 3.3: *Imagem desfocada.*
Fonte: Autoria própria.

A imagem a seguir nos permite visualizar os trechos do codigo o qual a imagem passa.

```
20  # Convertendo a imagem em escala de cinza
21  cinza = cv2.cvtColor(img_in, cv2.COLOR_BGR2GRAY)
22
23  #Binarizando imagem
24  _, binarizada = cv2.threshold(cinza, 129 , 255, cv2.THRESH_BINARY)
25
26  #Aplicando filtro Gaussiano
27  desfoque = cv2.GaussianBlur(binarizada, (5, 5), 0)
28
```

Figura 3.4: *Imagem desfocada.*
Fonte: Autoria própria.

Concluído assim a etapa de processamento, seguimos agora para a segmentação e extração de características da imagem de entrada.

3.2 Segmentação da imagem.

Nessa próxima etapa do trabalho, o processo de segmentação pode ser entendido como o particionamento/divisão de uma imagem em regiões que apresentem propriedades semelhantes, como textura ou cor. O princípio da segmentação foi apresentado por psicólogos

alemães, quando foi mostrado que o ser humano, no processo de visão, realiza naturalmente o agrupamento de regiões por critério baseados na proximidade, similaridade e continuidade, trazendo para nós uma forma de nos basearmos nesses critérios, e, artificialmente/computacionalmente criar nossos própios meios de segmentação.

Existem diferentes técnicas de se encontrar padrões em imagens digitais, a mais conhecida quando o assunto é Computação Gráfica e Processamento Digital de Imagens, é a utilização de classificadores, os haar-cascades ou classificadores de cascata. Esse tipo de classificador utiliza aprendizagem de máquina em busca de padrões.

Para a utilização desse classificador faz-se necessário um grande poder computacional e uma vasta coleção de imagens pré-carregadas de diferentes formas, cores e tamanhos para o treinamento do arquivo final utilizado para o reconhecimento, que no caso trata-se de um arquivo XML com todos os dados obtidos. Tentando contornar essa problemática partimos para o desenvolvimento de um algoritmo que reconhecesse o padrão de placas de veículos através do seu formato retangular, utilizando de padrões e dimensões como largura e altura. Partindo do processamento anterior que nos conseguiu melhoras nos padrões da imagem, utilizando-se da função findContours, foi possível encontrar esses padrões em uma imagem. Essa função é capaz de encontrar contornos e bordas, criando uma curva que une todos os pontos contínuos ao longo do limite com a mesma cor e intensidade,sendo assim uma técnica bastante utilizada para se encontrar padrões e objetos em uma imagem digital.

3.2.1 Etapas para segmentação da placa.

Para que fosse possível a extração da placa de uma imagem, foram utlizadas as seqguintes funções, respectivamente:

- Função findContours.
- Função drawContours
- Função arcLength
- Função approxPolyDP

3.2.2 Função findContours.

A Função recebe alguns argumentos como parâmetros, o primeiro é a imagem de origem, o segundo é o modo de recuperação de contorno, o terceiro é o método de aproximação de contorno. O método retorna uma imagem, contornos e suas hierarquias. Esses contornos na verdade são uma lista de todos os contornos obtidos da imagem. Cada um desses contornos individualmente é representado por uma matriz de coordenadas (x, y) de pontos de fronteira do objeto.

3.2.3 Função drawContours.

Função drawContours é a responsável por desenhar os contornos obtidos na imagem depois da etapa anterior.

3.2.4 Função arcLength.

Essa função irá verificar se os contornos encontrados na etapa anterior formam um perímetro, ou seja, se o mesmo é fechado.

> "Ele se aproxima de uma forma de contorno a outra forma com menos número de vértices, dependendo da precisão que forem especificados". " [MORDVINTSEV 2013].

3.2.5 Função approxPolyDP.

Essa função fará a aproximação dos contornos, assim, caso os perímetros dos contornos forem problemáticos ou seja com menos número de vértices, haja uma aproximação para seu formato mais semelhante.

Através das funções acimas citadas, foi alcançado o objetivo de encontrar um placa em uma determinada imagem. Para que isso fosse possível, foi utilizado um laço "for" como uma estrutura de repetição afim de que a cada interação verifique através das funções arcLength e approxPolyDP o perímetro, e seja constatado que o mesmo se trata de um retângulo validando a característica buscada. Esta condição de perimetro $>$ 120 é um parâmetro testado para um retângulo satisfatório, representando todos os retângulos encontrados na imagem eliminando todos os perímetros fora da condição, assim sendo possível fazer uma verificação dos retângulos que possivelmente poderão ser a placa. Na figura 3.5 podemos observar uma parte do código implementado que posteriormente resulta na correto perimetro que envolve apenas a região da placa, imagem 3.6.

```
30
31      if perimetro > 120:#pegando retangulos grandes e descartando os menores
32          #criar uma variavel para verificar a forma e transforma-la  na forma mais proxima que o contorno tem originalmen
33          aprox = cv2.approxPolyDP(c, 0.03 * perimetro, True)
34
35              #E ai verifico se tem 4 lados == 4
36          if len(aprox) == 4:
37
38                  #aproximar essa minha area de um retangulo ou de um quadrado
39                  (x, y, alt, lar) = cv2.boundingRect(c)
40                  cv2.rectangle(img_in, (x, y), (x+alt, y+lar), (0,255,0), 3)#contornando de verde
41
42                  P_E = img_in[y:y + lar, x:x + alt]#Capturando apenas a regiao da placa
43                  cv2.imwrite('ReriaoDaPlaca.jpg', P_E)
```

Figura 3.5: *Trecho do código.*
Fonte: Autoria própria.

Figura 3.6: *Contorno obtido.*
Fonte: Autoria própria.

Finalizado a localização apenas da placa, recortamos o perímetro encontrado, passando as coordenadas (x, y) encontradas nos contornos e assim escrevendo-as com o "nome.jpg"no disco para posterior tratamento.

Capítulo 4

Resultados parciais obtidos.

Obtida a imagem e passada pelos tratamentos de filtragem e segmentação, o que obtivemos nesta etapa foi um segmento da imagem original, ou seja, a placa a qual será interpretada pelo modulo de reconhecimento ótico de caracteres, o OCR.

Figura 4.1: *Contorno obtido.*
Fonte: Autoria própria.

A partir dessa segmento de imagem, importamos ela em outra classe do codigo que irá, novamente, refazer todo o processo de melhoramento da imagem. Para uma boa interpretação do OCR fez-se necessário ampliar a imagem de entrada, para isso utlizamos a função cv2.resize(img, (300,100), 0.5, 0.5), que irá receber como parametro img que é a imagem de entrada, redimensiona-la para uma escala de 300x100 pixels obedecendo uma proporção de 0.5.

Após o redimensionamento da imagem podemos prosseguir com as mesmas etapas descritas no item 3.1, Gray Scale (Escala de cinza), Thresholding (Limiarização) e Gaussian Blur (desfoque) seguindo essa mesma ordem. Abaixo podemos visualizar como ficou a imagem em cada uma das três etapas.

Figura 4.2: *Contorno obtido.*
Fonte: Autoria própria.

4.0.1 Resultados obtidos pelo OCR.

Com a imagem tratada, submetemos a função pytesseract.image_to_string que lerá o texto presente na imagem. O resultado obtido no console da IDE Pycharm foi NXL 4223, como pode ser visto na imagem 4.3. Como é possível perceber na imagem o reconhecimento dos caracteres teve um precissão de 100% para essa determinada imagem. Com essa informação em arquivo de texto é totalmente possível buscar a numeração da placa em um banco de dados vendo se esse veiculo possui ou não restrição e assim aborda-lo para as medidas previstas nas leis de trânsito.

Com isso obtemos um resultado parcial satisfatório que findará com a busca em banco de dados previamente criado apenas para fins de didatica.

Figura 4.3: *Resultado da leitura da placa.*
Fonte: Autoria própria.

Apêndice A

Base de Dados

Capítulo opcional usado para que o autor coloque um texto,documento ou informação referente ao assunto do trabalho, mas que não deve interromper a leitura.

Referências Bibliográficas

[Andrade e Márcio 2003] ANDRADE, I.; MáRCIO, P. Hidden Markov Models for speech recognition. *Technometrics*, v. 34, n. 4, p. 1, 2003. 1

[Gomes 2001] GOMES, B. H. Hidden Markov Models for speech recognition. *Technometrics*, v. 34, n. 4, p. 251–276, 2001. 5, 6

[Gomes 2001] GOMES, I. Hidden Markov Models for speech recognition. *Technometrics*, v. 34, n. 4, p. 1, 2001. 7

[Gonzales 2002] GONZALES, I. Hidden Markov Models for speech recognition. *Technometrics*, v. 34, n. 4, p. 1, 2002. 6

[Iglesias 2001] IGLESIAS, I. Hidden Markov Models for speech recognition. *Technometrics*, v. 34, n. 4, p. 1, 2001. 6

[Marques e Vieira 1999] MARQUES, B. H.; VIEIRA, L. R. Hidden Markov Models for speech recognition. *Technometrics*, v. 33, n. 3, p. 251–272, 1999. 4

[MORDVINTSEV 2013] MORDVINTSEV, K. A speaker-independent, syntax-directed, connectedd word recognition system based on hidden Markov models and level building. *ieeeassp*, v. 33, n. 3, p. 0 – 89, 2013. 14

[Paciornick 2001] PACIORNICK, I. Hidden Markov Models for speech recognition. *Technometrics*, v. 34, n. 4, p. 1, 2001. 7