

**PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO**

**RECONHECIMENTO DE
PLACAS DE CARRO EM UM
SISTEMA EMBARCADO**

**DANIEL ANTONIAZZI AMARANTE E
MATTHIAS OLIVEIRA NUNES**

Trabalho de Conclusão II apresentado
como requisito parcial à obtenção
do grau de Bacharel em Ciência da
Computação na Pontifícia Universidade
Católica do Rio Grande do Sul.

Orientador: Prof. Roland Teodorowitsch

**Porto Alegre
2017**

RECONHECIMENTO DE PLACAS DE CARRO EM UM SISTEMA EMBARCADO

RESUMO

O controle e identificação de veículos é usado nas mais diversas áreas, indo desde serviços de pagamentos automatizados, como pedágios, até aplicações mais críticas, como segurança de fronteiras e sistemas de vigilância de tráfego. Com o crescimento constante da frota de carros no Brasil, aplicações para auxiliar neste trabalho tornam-se cada vez mais necessárias. O fato de que há peculiaridades nas placas automotivas brasileiras, que impossibilitam a utilização de ferramentas configuradas para placas estrangeiras, também evidencia a necessidade de soluções locais para este problema. Com isso em mente, propõe-se neste trabalho uma solução de aplicação embarcada de reconhecimento de placas.

Palavras-Chave: Reconhecimento de placas de carro, Processamento de Imagens.

CAR LICENSE PLATE RECOGNITION IN A EMBEDDED SYSTEM

ABSTRACT

The identification and control of vehicles is used in many different areas, ranging from automated payment services, like toll booths, to more critical applications, such as border security and traffic vigilance systems. The fact that there are peculiarities in the brazilian license plates, that preclude the utilization of tools configured for international license plates, also points to the necessity of local solutions for that problem. With the constant growth of the Brazilian car fleet, applications used to help in this line of work become more necessary every day. With that in mind, it is proposed in this paper a solution of an embedded application for automatic license plate recognition.

Keywords: Automatic License Plate Recognition, Image Processing.

LISTA DE FIGURAS

Figura 2.1 – Método de Otsu para definir o limiar	14
Figura 2.2 – Dilatação em (b) com elemento estruturante de 3x3 e em (c) com elemento estruturante de 5x5	16
Figura 2.3 – Erosão em (b) com elemento estruturante de 3x3 e em (c) com elemento estruturante de 5x5	16
Figura 2.4 – Exemplo de aplicação do K-Nearest Neighbors	18
Figura 3.1 – Imagem com as bordas encontradas e buracos preenchidos	23
Figura 4.1 – O Raspberry Pi 3 model B	26
Figura 5.1 – Quatro estágios do reconhecimento de placa	28
Figura 5.2 – Placa de um carro	29
Figura 5.3 – Placa de uma moto	29
Figura 5.4 – Tipografia das placas	29
Figura 5.5 – Imagem em escala de tons de cinza.	31
Figura 5.6 – Aplicação de filtro bilateral em uma imagem em escala de tons de cinza	32
Figura 5.7 – Aumento de contraste usando equalização de histograma adaptativo	33
Figura 5.8 – Imagem Binarizada	33
Figura 5.9 – Detecção de borda pelo operador Sobel	34
Figura 5.10 – Dilatação morfológica	35
Figura 5.11 – Após preenchimento dos buracos	35
Figura 5.12 – Detecção das áreas candidatas da placa	36
Figura 5.13 – Placa extraída	37
Figura 5.14 – Placa aprimorada	37
Figura 5.15 – Preenchimento dos caracteres	38
Figura 5.16 – Máscara gerada	38
Figura 5.17 – Caracteres a serem extraídos	39
Figura 5.18 – Exemplo de classificação de caractere extraído	40
Figura 5.19 – Pipeline implementado para otimização	41
Figura 6.1 – Veículo que o algoritmo tem dificuldade de reconhecer	43
Figura 6.2 – Placa inclinada que o sistema foi capaz de identificar	44
Figura 6.3 – Placa com caractere mal reconhecido por causa de reflexo	44
Figura 6.4 – Placa desgastada	45

Figura 6.5 – Placa que confundiu os caracteres O e Q 45

Figura 6.6 – Performance da aplicação 47

LISTA DE TABELAS

Tabela 6.1 – Resultados obtidos	42
Tabela 6.2 – Resultados obtidos em diferentes ambientes	43
Tabela 6.3 – Resultados relativos da fase de extração	46

SUMÁRIO

1	INTRODUÇÃO	9
2	FUNDAMENTAÇÃO TEÓRICA	11
2.1	PROCESSAMENTO DE IMAGENS	11
2.1.1	FILTRO BILATERAL	12
2.1.2	DETECÇÃO DE BORDAS	12
2.1.3	LIMIARIZAÇÃO	13
2.1.4	OPERAÇÕES MORFOLÓGICAS	14
2.2	RECONHECIMENTO ÓTICO DE CARACTERES	17
2.2.1	K-NEAREST NEIGHBORS	17
3	TRABALHOS RELACIONADOS	19
3.1	RECONHECIMENTO DE PLACAS BRASILEIRAS	19
3.2	ESTUDOS COMPARATIVOS	19
3.3	EXTRAÇÃO DA PLACA	22
3.4	SEGMENTAÇÃO DOS CARACTERES	24
3.5	RECONHECIMENTO EM SISTEMA EMBARCADO	24
3.6	ANÁLISE DOS TRABALHOS	24
4	AMBIENTE DE DESENVOLVIMENTO	26
4.1	RASPBERRY PI	26
4.2	OPENCV	27
5	SOLUÇÃO DESENVOLVIDA	28
5.1	PLACA DE TRÂNSITO BRASILEIRA	28
5.2	AQUISIÇÃO DAS IMAGENS	29
5.3	EXTRAÇÃO DA PLACA	30
5.3.1	CONVERSÃO DE RGB PARA ESCALA DE TONS DE CINZA	31
5.3.2	REMOÇÃO DE RUÍDO POR FILTRO BILATERAL	31
5.3.3	AUMENTO DE CONTRASTE USANDO EQUALIZAÇÃO DE HISTOGRAMA ADAPTATIVO	32
5.3.4	BINARIZAÇÃO DA IMAGEM	33
5.3.5	DETECÇÃO DE BORDA PELO OPERADOR SOBEL	34

5.3.6	OPERAÇÃO DE DILATAÇÃO SOBRE AS BORDAS	34
5.3.7	PREENCHIMENTO DE BURACOS NA IMAGEM	35
5.3.8	DETECÇÃO DE ÁREAS DE PLACA CANDIDATAS POR OPERAÇÕES MORFOLÓGICAS DE ABERTURA E FECHAMENTO	36
5.3.9	EXTRAÇÃO DA ÁREA DA PLACA REAL	36
5.3.10	APRIMORAMENTO DA REGIÃO EXTRAÍDA	37
5.4	SEGMENTAÇÃO DOS CARACTERES	37
5.5	RECONHECIMENTO DOS CARACTERES	39
5.6	OTIMIZAÇÃO COM MULTIPROCESSAMENTO EM PIPELINE	40
6	RESULTADOS OBTIDOS	42
6.1	EXTRAÇÃO DA PLACA E CARACTERES	42
6.2	RECONHECIMENTO DOS CARACTERES	43
6.3	PERFORMANCE EM SISTEMA EMBARCADO	45
7	CONCLUSÃO E TRABALHOS FUTUROS	48
	REFERÊNCIAS	49

1. INTRODUÇÃO

O controle e identificação de veículos é usado nas mais diversas áreas, indo desde serviços de pagamentos automatizados, como pedágios, até aplicações mais críticas, como segurança de fronteiras, sistemas de vigilância de tráfego [2] e sistema de busca por carros roubados. Uma solução para a identificação de placas é, inclusive, parte do Plano de Governo do atual prefeito eleito de Porto Alegre, Nelson Marchezan. Ele pretende utilizar os sistemas de controle de velocidade da cidade para também monitorar as placas de carros com o objetivo de identificar carros roubados [18].

Com o crescimento constante da frota de carros no Brasil, aplicações para auxiliar neste trabalho tornam-se cada vez mais necessárias. Com isso em mente, propõe-se neste trabalho uma solução de aplicação embarcada de reconhecimento de placas, visando à crescente necessidade de controle na área. O fato de que há peculiaridades nas placas automotivas brasileiras, que impossibilitam a utilização de ferramentas configuradas para placas estrangeiras, também evidencia a necessidade de soluções locais para este problema.

O reconhecimento automático de placas de carros (*Automatic License-Plate Recognition*, ALPR) corresponde à extração das informações das placas de veículos a partir de uma imagem ou de uma sequência de imagens. A sua utilização na vida real precisa de um processamento rápido e bem sucedido de placas sob diferentes condições ambientais. Deve-se considerar as variações entre as placas de diferentes nações, que terão cores, fontes, símbolos, padrões e línguas diferentes. Também é preciso superar casos nos quais as placas possam estar parcialmente cobertas com sujeira, luzes e acessórios dos carros, e também a iluminação do ambiente e qualidade da imagem adquirida [19].

O objetivo deste trabalho é desenvolver um *software* embarcado em um *Raspberry Pi* (um computador de tamanho reduzido) equipado com módulo de câmera que seja capaz de reconhecer veículos e identificá-los baseando-se em sua placa. A solução proposta tem como diferencial o fato de permitir a análise e processamento das imagens em tempo real. Isso é realizado utilizando um software embarcado, que coleta as imagens ao mesmo tempo que as analisa. Essa abordagem evitaria ter que enviar a imagem inteira para ser processada em um servidor, ocupando menos largura de banda na transmissão. Dessa maneira é possível que esses dados sejam úteis para uma tomada de decisão imediata das "autoridades" de controle de tráfego ou policiamento.

Um exemplo de uma aplicação, na qual a velocidade e a disponibilidade imediata das informações é crucial para a viabilidade do produto, seria em um *software* de identificação de carros roubados. O sistema analisaria as placas dos carros que trafegam em uma rodovia e identificaria quais daqueles carros têm placas que correspondem a placas de carros roubados. Para que o *software* seja eficiente é necessário que o processamento

seja feito em tempo real. Qualquer atraso na identificação do veículo permite que ele se distancie muito, impedindo que as autoridades reajam a tempo.

Uma abordagem alternativa ao processamento em tempo real, seria fazer a captura das imagens em um momento, e o processamento em outro. Entretanto, desta maneira, a utilidade do *software* é limitada, pois nas aplicações citadas, estes dados precisam estar disponíveis imediatamente.

A solução desenvolvida foi testada e analisada em duas partes: A primeira parte foi o teste de funcionamento, que tem como objetivo avaliar se o programa consegue identificar as placas, com qual frequência e em quais condições. A segunda foi o teste de performance, que tem como objetivo avaliar se o programa consegue identificar as placas em tempo aceitável para ser utilizado em situações reais. Assim foi possível aprimorar a solução para garantir melhor funcionamento.

Para o teste do funcionamento da detecção foram geradas múltiplas imagens, em condições e ângulos levemente diferentes. Com essas imagens foi possível analisar casos em que a detecção funciona e casos em que a detecção não funciona, para poder avaliar os motivos e fazer melhorias.

Para o teste da performance, foi posicionado o computador com a câmera em um ponto estratégico de um estacionamento para analisar os carros que passavam. O programa pode, assim, coletar informações referentes ao valor da placa dos carros que passavam por aquele ponto. Dessa maneira foi possível medir o desempenho do programa tanto na velocidade da execução quanto na quantidade de acertos obtidos, podendo assim, descobrir qual a viabilidade da implementação em casos reais.

Este trabalho foi organizado da seguinte maneira: no Capítulo 2, serão mostrados e explicados os algoritmos e os termos técnicos utilizados no trabalho. No Capítulo 3, serão apresentados trabalhos relacionados aos temas de reconhecimento de placas de carros. No Capítulo 4, serão apresentadas as tecnologias utilizadas e o ambiente de desenvolvimento. No Capítulo 5, serão especificados os passos seguidos no desenvolvimento da aplicação, explicando cada etapa do projeto. No Capítulo 6 serão apresentados os resultados práticos obtidos com este projeto. No Capítulo 7 será feita uma conclusão sobre o resultado do trabalho e serão sugeridos futuros trabalhos que podem surgir com base na pesquisa feita.

2. FUNDAMENTAÇÃO TEÓRICA

Diversas técnicas e algoritmos são utilizadas nas aplicações de reconhecimento automático de placas de carros. Neste capítulo é feito um aprofundamento teórico sobre as técnicas utilizadas em todas as etapas do desenvolvimento do trabalho, explicando os principais algoritmos e conceitos aplicados na construção da ferramenta.

2.1 Processamento de Imagens

Uma imagem pode ser definida como uma função $f(x, y)$, na qual as variáveis x e y representam as coordenadas em um plano e o valor de f em qualquer par de coordenadas (x, y) representa a cor da imagem naquele ponto, se a imagem for colorida, ou a intensidade do cinza, em uma imagem em tons de cinza. Essas imagens digitais são compostas de um número finito de elementos, cada um com uma posição diferente e um valor. Estes elementos são conhecidos como *pixels* [6].

O campo do processamento digital de imagens se refere ao processamento de imagens feito por um computador. Segundo Gonzalez [6], não existe um consenso geral entre os autores sobre onde o processamento de imagens acaba e onde começam outros campos, como a análise de imagens e a visão computacional. As fronteiras entre estes campos não são muito claras, mas é possível dividir estes processos em três tipos distintos:

- Processos de baixo nível envolvem operações primitivas, como pré-processamento para redução de ruídos e aumento de contraste.
- Processos de nível médio envolvem tarefas como segmentação, descrição e reconhecimento de objetos. Estes processos são caracterizados pelo fato de que recebem como entrada uma imagem, mas têm como saída atributos extraídos dessa imagem, como contornos, bordas e a identidade de objetos nela existentes.
- Processos de alto nível visam obter um significado de um conjunto de objetos reconhecidos, fazendo funções normalmente associadas à visão.

No reconhecimento digital de placas de carro, os três tipos diferentes de processos categorizados por Gonzalez [6] estão presentes. Processos de baixo nível, para pré-processar as imagens extraídas que contêm os carros. Processos de nível médio, para extrair elementos dessas imagens, como as bordas e as regiões de interesse. Processos de alto nível, que buscam, com base nestes elementos extraídos, obter informações referentes a placa e aos caracteres.

Nas subseções seguintes serão listadas algumas técnicas de processamento de imagens que foram utilizadas na solução criada. Na Subseção 2.1.1 é fundamentada a aplicação do Filtro Bilateral, na Subseção 2.1.2 será fundamentada a detecção de bordas em uma imagem, na Subseção 2.1.3 será fundamentada a Limiarização e na Subseção 2.1.4 serão fundamentadas as operações morfológicas em imagens.

2.1.1 Filtro Bilateral

Filtros podem ser a operação mais fundamental de processamento de imagens de acordo com Tomasi e Manduchi [21]. Na operação de filtro, o valor da imagem filtrada em uma determinada localização é uma função dos valores da imagem de entrada em um pequeno conjunto da mesma localização. No filtro gaussiano de passa-baixa, por exemplo, é calculada uma média ponderada dos *pixels* da região onde o peso diminui com a distância do centro do conjunto. A ideia por trás disso é que imagens geralmente variam pouco no espaço, por isso, o ruído é atenuado e o sinal é preservado. Essa ideia da baixa variação dos *pixels* próximos não funciona bem nas bordas, fazendo com que acabem ficando borradadas na imagem [21].

Uma solução de filtro que contorna esse problema é o Filtro Bilateral. O Filtro Bilateral é um método não iterativo para suavizar imagens preservando as suas bordas. Ele utiliza uma combinação não linear de valores próximos da imagem, combinando os níveis de cinza ou cores baseado em sua proximidade geométrica e similaridade fotométrica [21].

2.1.2 Detecção de Bordas

Detecção de bordas é um método de processamento de imagem desenvolvido para detectar *pixels* de borda. Os *pixels* de borda são *pixels* em que a intensidade da imagem muda abruptamente, e as bordas são conjuntos de *pixels* de borda conexos [6].

O gradiente de uma imagem é uma troca de intensidade ou cor de uma imagem. Ele é computado pelas variações da imagem nas direções x e y. Pode-se considerar que uma borda acontece quando o gradiente está em seu máximo, ou seja, está havendo uma troca de intensidade. O gradiente pode ser dado pela seguinte fórmula:

$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

Um método utilizado para calcular os gradientes de uma imagem é utilizando o Operador de *Sobel*. Essa é a técnica utilizada na implementação do reconhecedor de placas deste trabalho, e ela funciona da seguinte maneira:

Considerando-se que I é a imagem a ser processada, são calculadas duas derivadas. As mudanças horizontais são computadas fazendo a convolução de I com um filtro G_x de tamanho ímpar. A convolução é uma técnica em que se desloca uma máscara sobre uma imagem e calcula-se a soma dos produtos em cada local. Utilizando um filtro de tamanho 3, o cálculo do gradiente horizontal seria da seguinte maneira:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I$$

Já para as mudanças verticais, utilizando um filtro de tamanho 3, o cálculo seria da seguinte maneira:

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

Estes filtros estimam os gradientes nas direções horizontal e vertical, e a magnitude do gradiente pode ser calculada somando os dois gradientes. O cálculo da aproximação do gradiente em cada ponto é dado pela seguinte equação:

$$G = \sqrt{G_x^2 + G_y^2}$$

2.1.3 Limiarização

Seja uma determinada imagem em tons de cinza composta de objetos claros em um fundo escuro. Uma maneira simples de extrair os objetos seria criar um valor de limiar e separar os *pixels* com intensidade menor que o limiar dos *pixels* com intensidade maior que o limiar. Elementos com intensidade menor que o limiar seriam classificados como "objetos" e os elementos com intensidade maior que o limiar seriam classificados como "fundo". Essa é a base da técnica chamada de limiarização [6].

A técnica mais simples de limiarização utiliza um limiar global T . Com essa técnica, a imagem é escaneada *pixel a pixel* classificando cada *pixel* como objeto ou fundo, dependendo se sua intensidade é maior ou menor do que o limiar global T . O sucesso desse método depende inteiramente de o quanto bem a imagem pode ser particionada, sendo bem utilizado em imagens bimodais, que têm dois tons de cinza que se destacam [6].

Imagens com iluminação desigual podem fazer com que imagens perfeitamente segmentáveis tenham péssimos resultados utilizando o limiar global. Para superar essas dificuldades foi criada a limiarização adaptativa. Com ela as imagens são divididas em imagens menores, tendo um limiar diferente para cada parte da imagem dividida. As dificuldades da limiarização adaptativa são: como dividir a imagem; e como escolher os limiares das partes da imagem [6].

Dadas as opções, neste trabalho é utilizada uma estratégia de limiarização com limiar global. As imagens dos carros tendem a ser bimodais, com um tom bem significativo vindo da lataria do carro, e um segundo tom significativo vindo da cor da placa. Para identificar se a imagem é bimodal pode-se calcular o histograma da imagem, que é a representação da divisão de frequências da imagem. Se o histograma da imagem tiver dois picos, essa imagem permite o uso de limiar global.

Para calcular o limiar utiliza-se o método de Otsu. Essa técnica calcula o ponto médio entre os dois picos e utiliza esse valor como limiar. Este valor é calculado automaticamente com base no histograma da imagem [14]. A aplicação do método de Otsu para definir o limiar de uma imagem bimodal pode ser vista na Figura 2.1

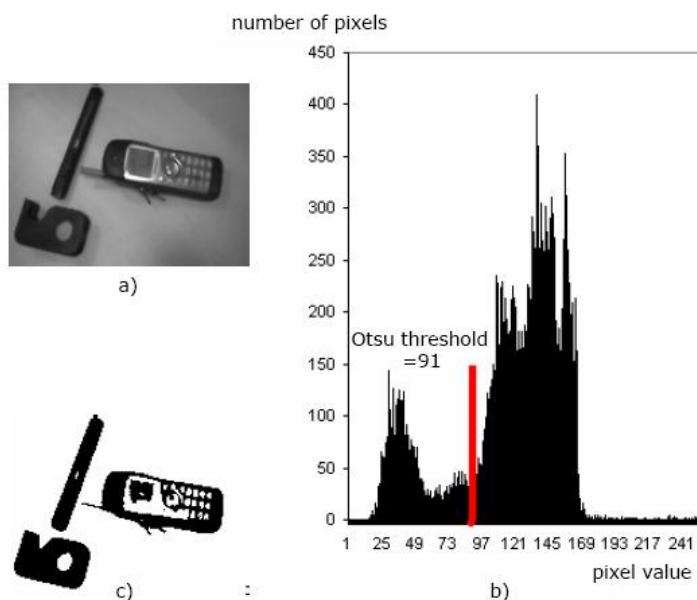


Figura 2.1 – Método de Otsu para definir o limiar

Fonte: Intel [9]

2.1.4 Operações Morfológicas

Morfologia matemática é uma ferramenta para extrair componentes da imagem que são úteis para representação e descrição. É um método da análise de imagens que utiliza

teoria dos conjuntos, podendo prover uma descrição quantitativa de estruturas geométricas. A maior parte das operações morfológicas são baseadas em operações de expansão e encolhimento, e são principalmente utilizadas em imagens binárias [17].

Essas transformações envolvem interações entre uma imagem a ser processada, e um conjunto estruturante, chamado de elemento estruturante. Este elemento costuma ser um disco circular no plano, mas pode ter qualquer forma [17].

Algumas operações são importantes para descrever as operações morfológicas, elas são: A translação, a reflexão e o complemento.

A translação de B por x é denominada B_x . Sejam A e B subconjuntos de um conjunto Z^2 , ela é definida por:

$$B_x = \{c : c = b + x, \text{ for } b \in B\}$$

A reflexão de B , denominada \hat{B} é definida por:

$$\hat{B} = \{x : x = -b, \text{ for } b \in B\}$$

O complemento de A é denominado A_c e a diferença entre os conjuntos A e B é denominado $A - B$.

As principais operações morfológicas são a erosão e a dilatação. Essas operações são fundamentais para o processamento morfológico. Grande parte das operações morfológicas são baseadas nessas duas operações [6].

A dilatação de uma imagem A pelo elemento estruturante B é dada por:

$$A \oplus B = \{x : \hat{B}_x \cap A \neq \emptyset\}$$

Na dilatação, o objetivo é dilatar as fronteiras de um objeto, ou seja, aumentar a espessura dos objetos em primeiro plano de uma imagem. A dilatação ocorre ao percorrer a imagem de entrada com o elemento estruturante, se ao menos um *pixel* sob o elemento estruturante tiver seu valor sendo 1, na imagem original, todos os *pixels* sob o elemento estruturante vão ser marcados como 1 na imagem de saída [15]. Na Figura 2.2 pode-se ver um exemplo da operação de dilatação em uma imagem.



Figura 2.2 – Dilatação em (b) com elemento estruturante de 3x3 e em (c) com elemento estruturante de 5x5

Fonte: Efford [5]

A erosão de uma imagem A pelo elemento estruturante B é dada por:

$$A \ominus B = \{x : B_x \subseteq A\}$$

A ideia por trás da erosão é, tal qual a erosão do solo, erodir as fronteiras de um objeto em primeiro plano. O elemento estruturante vai percorrer a imagem original e vai manter os *pixels* da imagem de saída em 1 somente se todos os *pixels* sob o elemento estruturante têm o valor de 1, caso contrário, essa parte da imagem é erodida, tem seus *pixels* zerados. Dessa maneira os *pixels* próximos das bordas vão ser descartados com base no tamanho do elemento estruturante e a espessura dos objetos em primeiro plano diminui [15]. Na Figura 2.3 pode-se ver um exemplo da operação de erosão em uma imagem.

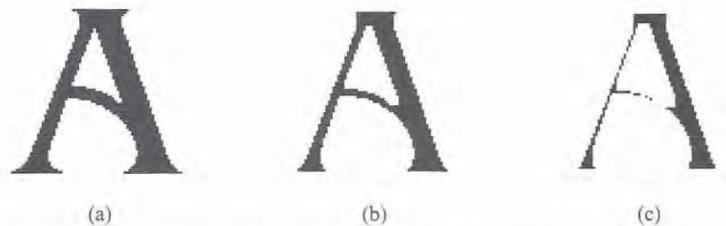


Figura 2.3 – Erosão em (b) com elemento estruturante de 3x3 e em (c) com elemento estruturante de 5x5

Fonte: Efford [5]

Utilizando as operações de erosão e dilatação, é possível fazer as operações de abertura e fechamento. A abertura de uma imagem A, dado um elemento estruturante B, simbolizada por $A \circ B$, é feita pela erosão de A por B, seguida da dilatação de A por B, e pode ser representada pela equação:

$$A \circ B = (A \ominus B) \oplus B$$

O fechamento de uma imagem A , dado um elemento estruturante B , simbolizada por $A \bullet B$, é feito pela dilatação de A por B , seguida da erosão de A por B , e pode ser representada pela equação:

$$A \bullet B = (A \oplus B) \ominus B$$

Assim como a abertura, o fechamento também suaviza os contornos dos objetos, mas, ao contrário da abertura, ele geralmente funde as protusões, eliminando pequenos buracos e preenchendo espaços no contorno [6].

2.2 Reconhecimento Ótico de Caracteres

Reconhecimento Ótico de Caracteres (*Optical Character Recognition*, OCR) resulta na conversão de textos em formato de imagem para o formato reconhecido por máquina. É o método mais eficiente para fazer o processamento de imagem para texto de acordo com Mohit et al. [12].

Uma ferramenta conhecida de OCR é o *Tesseract*¹. É um *software open source* de reconhecimento ótico de caracteres que suporta múltiplas línguas. É essencialmente um algoritmo de comparação de *templates*, e permite o auto treinamento de suas amostras de caracteres [8].

Neste trabalho, será implementado um *software* de reconhecimento ótico de caracteres focado especificamente em reconhecer caracteres de placas de trânsito brasileiras utilizando aprendizado de máquina com o algoritmo *K-Nearest Neighbors* que será fundamentado na Seção 2.2.1

2.2.1 K-Nearest Neighbors

K-Nearest Neighbors é um dos algoritmos mais simples de classificação disponíveis para aprendizado supervisionado em aprendizado de máquina [16]. Aprendizado supervisionado é tarefa de inferir uma função a partir de dados de treinamento rotulados. O algoritmo recebe um conjunto de exemplos como dados de treinamento e faz predições para os pontos não vistos com base nestes dados [13]. A ideia do algoritmo *K-Nearest Neighbors* é encontrar a combinação mais próxima de um determinado dado de teste em um espaço de dados [16].

¹<https://github.com/tesseract-ocr/tesseract>

Em um caso hipotético de aplicação do algoritmo existem duas classes: A classe dos quadrados azuis e a classe dos triângulos vermelhos. Este exemplo está ilustrado na Figura 2.4:

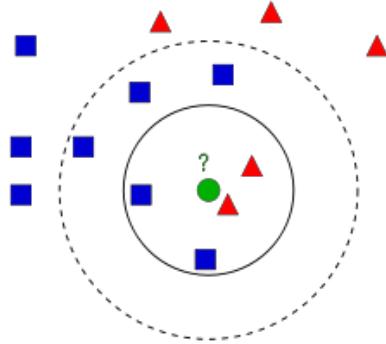


Figura 2.4 – Exemplo de aplicação do K-Nearest Neighbors

Fonte: Understanding k-Nearest Neighbour [16]

Pontos representando estas classes são espalhados em um espaço chamado de *feature space*. Estes espaços são os espaços onde todos os dados estão projetados. Em um espaço de duas dimensões, como no exemplo, cada dado tem duas características, x e y . Em um espaço de três dimensões, cada dado teria 3 características, em N dimensões, N características.

Na adição de um novo dado no *feature space*, ele deve ser classificado como uma das duas classes. Este é o chamado de processo de classificação, no qual o algoritmo é aplicado.

Uma forma é o de verificar quem é o vizinho mais próximo. Na figura 2.4, este seria o triangulo vermelho. Este método é chamado de *Nearest Neighbor*, ou o vizinho mais próximo, pois a classificação só depende de um vizinho.

Outro método seria checar múltiplos vizinhos para ver quantos vizinhos de cada classe o novo dado possui. Na imagem, o triangulo vermelho é o vizinho mais próximo, mas, considerando múltiplos vizinhos, é possível classificar o novo dado na classe dos quadrados azuis, pois existem mais vizinhos desta classe. Este método é chamado de *k Nearest Neighbors*, pois ele considera uma quantidade predeterminada k de vizinhos para classificar seus novos dados [16].

Neste projeto o algoritmo *K-Nearest Neighbors* é utilizado na implementação do reconhecedor ótico de caracteres encontrados em placas de transito brasileiras. São utilizados como dados de teste as letras maiúsculas e números na fonte *Mandatory*, que é a fonte utilizada nas placas de carro brasileiras, e são aproximados os valores dos caracteres segmentados da placa com base nestes. Como haverá apenas um valor para cada classe, cada classe representando um caractere diferente, será utilizado apenas o vizinho mais próximo para classificar os caracteres. Tendo um conjunto de treinamento maior, é possível aumentar este valor.

3. TRABALHOS RELACIONADOS

O reconhecimento de placas de carro é uma área bastante estudada com diversas publicações sobre o tema. Neste capítulo são apresentadas algumas destas publicações, organizadas por sua relevância nas diferentes etapas do reconhecimento.

3.1 Reconhecimento de placas brasileiras

Uma solução para detectar e reconhecer placas de licenciamento brasileiras foi proposta por Serro [20] na PUCRS. Uma das motivações descritas na concepção do trabalho foi a falta de publicações focadas na placa de transito brasileira, que, por suas peculiaridades, torna difícil a adaptação de soluções estrangeiras. Neste trabalho foram utilizadas técnicas de segmentação de imagens, histograma, cisalhamento de imagens e reconhecimento ótico de caracteres utilizando a ferramenta *Tesseract*. Este último não foi o foco principal do projeto. A metodologia utilizada consistiu das seguintes etapas: Calibração do sistema para definir a região de interesse e o ângulo de cisalhamento; detecção da placa; segmentação dos caracteres; e aplicação do reconhecedor ótico de caracteres.

Com a solução proposta por Serro [20] foi obtida uma taxa de acerto de aproximadamente 54%, com tempo médio de execução de 0,062 segundos por imagem. A baixa taxa de acerto pode ter sido por problemas de foco e nitidez, o tamanho da placa em *pixels* nas imagens e a existência de outros objetos em cena. Todos esses problemas foram citados no desenvolvimento do projeto.

Devido a falta de padronização das placas ao redor do mundo existe uma alternativa que permite ao usuário moldar a solução para se adaptar a todos os casos. O software *open source Openalpr*¹ é uma ferramenta de reconhecimento de placas que autoriza a comunidade a contribuir no desenvolvimento, treinando o reconhecedor de caracteres a reconhecer as placas de seus países.

3.2 Estudos comparativos

Alguns autores fizeram estudos comparando as técnicas mais comuns para solucionar o problema. Em Ahmad et al. [2] foi feito um destes estudos comparativos. Segundo estes autores, o processo de ler o conteúdo de uma placa passa por três estágios. O primeiro é a localização ou extração da placa, que consiste no processo de localizar a placa

¹<https://github.com/openalpr/openalpr>

do carro na imagem. O segundo estágio é a separação dos caracteres, na qual cada caractere individual é separado dos outros para reconhecimento. E o terceiro e último estágio é o reconhecimento do caractere em si, na qual os caracteres extraídos da imagem são identificados.

Nesse trabalho foram implementados três diferentes métodos de localização de placa e dois diferentes métodos de reconhecimento de caracteres, resultando em 6 diferentes abordagens para o reconhecimento de placas. Todas essas combinações foram, então, testadas contra diferentes conjuntos de dados. Os três métodos de localização da placa são:

- O primeiro método de extração estudado por Ahmad et al. [2] foi um método baseado em extrair as bordas verticais da imagem utilizando o operador de Sobel. Nele, a localização da placa se dá pela comparação das distâncias das bordas encontradas, que devem satisfazer um valor mínimo e máximo predefinido.
- O segundo método de extração utiliza histogramas verticais para encontrar as faixas mais prováveis de conter uma placa. A altura das faixas deve estar em uma variação razoável, então os que não se encaixam neste padrão são descartados.
- O terceiro método consiste de três passos: Primeiro, pontos próximos são unidos para formar linhas. Depois, quando possível, linhas são unidas para formar retângulos. Por fim estes retângulos são aceitos ou rejeitados.

Os dois métodos de reconhecimento de caracteres analisados por Ahmad et al. [2] são:

- O primeiro método é baseado em comparação de modelos. Neste método uma determinada imagem é binarizada e depois comparada *pixel a pixel* com padrões binários guardados em disco para encontrar o mais semelhante.
- O segundo método é mais complexo. Ele se baseia em uma rede neural probabilística. Este método funciona primeiro encontrando um bom mapeamento do espaço de imagem para o espaço dos caracteres, e então mapeando a nova imagem para este espaço. A rede neural então é treinada utilizando todas as instâncias de caracteres no espaço dos caracteres.

Os resultados obtidos por Ahmad et al. [2] na experimentação não foram muito animadores, variando entre 20 e 40 por cento de acerto na identificação da placa. Um dos motivos para os resultados ruins foi a variedade de parâmetros nas imagens do conjunto de dados de teste. Tais parâmetros incluem variações na distância, ângulo, iluminação e ambiente. Há a possibilidade de que estes erros poderiam ser mitigados em sistemas reais com uma câmera com resolução fixa e de boa qualidade. A variação do tamanho da placa

afetou o desempenho de alguns algoritmos, mas em uma câmera fixa é possível obter uma consistência e conseguir resultados mais aceitáveis.

Outros motivos para a baixa taxa de acerto foram a falta de pré-processamento das imagens, que a análise não considerou, e a falta de utilização de mais dados de aprendizado para o reconhecimento ótico de caracteres.

Em S Du et al. [19] também foi feito um estudo semelhante. Foi feita uma pesquisa categorizando os métodos existentes de acordo com as características de cada um. Destes métodos, foram analisados seus prós e contras, e foram comparados sua performance de reconhecimento e velocidade de processamento. Foram estudadas seis técnicas de extração da placa, cinco técnicas de segmentação dos caracteres e duas técnicas de reconhecimento dos caracteres.

As técnicas de extração de placas analisadas por S Du et al. [19] são:

- Extração da placa utilizando características de fronteiras. Utilizando o conhecimento prévio de que o contorno da placa é retangular e procurando por contornos retangulares. Um método simples e rápido, mas dificilmente aplicável a imagens complexas por ser muito sensível a bordas indesejadas.
- Extração da placa utilizando características globais da imagem. Buscando por objetos conectados cujas dimensões se assemelham a uma placa de carro. É uma abordagem direta e independente da posição da placa, porém pode gerar objetos quebrados.
- Extração da placa utilizando características de textura. Técnica baseada na frequente troca de cores em uma placa de carro. Com ela pode-se detectar a placa mesmo que suas bordas estejam deformadas, porém é muito complexa computacionalmente.
- Extração da placa utilizando características de cores. Essa estratégia busca por uma cor específica da placa. Tem a mesma vantagem que a técnica anterior de detectar placas com bordas deformadas, mas é muito limitada a problemas de iluminação.
- Extração da placa utilizando características dos caracteres. Esta maneira necessita que haja caracteres na placa. É uma técnica bem lenta e produz erros quando há mais texto na imagem além da placa.
- Extração da placa utilizando múltiplas características. Utilizando-se de mais de uma das técnicas anteriores se conseguem resultados mais confiáveis, porém é mais complexa computacionalmente.

As técnicas de segmentação de caracteres analisadas por S Du et al. [19] são:

- Segmentação utilizando conectividade de *pixels*. Procura por *pixels* conexos para identificar um caractere. Estratégia simples e direta, robusta contra rotação da placa. Sua desvantagem está quando os caracteres na imagem estão quebrados ou unidos.

- Segmentação utilizando perfis de projeção. Essa estratégia independe da posição dos caracteres e também consegue lidar com rotação, porém, ruídos na imagem podem afetar o valor da projeção.
- Segmentação utilizando conhecimento prévio dos caracteres. Um método simples que assume a posição dos caracteres com base em conhecimento existente. Mudanças podem afetar o resultado.
- Segmentação utilizando contornos dos caracteres. Encontrando os contornos dos caracteres é possível delimitar as fronteiras exatas dos caracteres, mas é uma técnica lenta e pode gerar contornos incompletos ou distorcidos.
- Segmentação utilizando características combinadas. Utilizar mais de uma estratégia. Assim como na segmentação da placa, é uma técnica mais confiável, porém computacionalmente mais complexa.

Por fim, as técnicas de reconhecimento de caracteres analisadas por S Du et al. [19] são:

- Reconhecimento por combinação de modelos (*Template Matching*). É uma estratégia que pega os caracteres prontos e compara os caracteres a serem reconhecidos com eles. É um método simples, útil para reconhecer caracteres de uma fonte única e tamanho fixo.
- Reconhecimento utilizando características extraídas. É uma alternativa a combinação de modelos. Esta técnica extrai características da imagem, como, por exemplo, formas do caractere. Isso diminui o tempo de processamento pois nem todos os *pixels* estão mais envolvidos.

3.3 Extração da placa

Kaur [11] propõe um método eficiente de extração que se baseia em detecção de bordas. A imagem adquirida é tratada por um pré-processamento que aplica operações de redução de ruídos e aumento de contraste com técnicas de filtro bilateral, equalização de histograma adaptativo e abertura morfológica seguida de subtração da imagem.

Após o pré-processamento da imagem, é aplicada uma binarização utilizando o método de Otsu e uma detecção de bordas utilizando o operador de Sobel. O objetivo da detecção de bordas é tentar encontrar a borda fechada ao redor da placa do carro. Com as bordas encontradas, os buracos da imagem são preenchidos utilizando a operação *imfill* do software MATLAB². Com o preenchimento dos buracos, a borda gerada pela placa será

²<https://www.mathworks.com/products/matlab.html>

preenchida com *pixels* conexos. Tendo essa região preenchida, é aplicada uma operação morfológica de abertura e fechamento para fazer desaparecer os conjuntos de *pixels* em grupos menores, mantendo apenas candidatos a placa. A Figura 3.1 mostra a etapa do processamento do algoritmo feito por Kaur [11] onde as bordas são preenchidas. Nela é possível observar uma das etapas mais importantes deste método, na qual a região da placa a ser extraída já está bem visível.



Figura 3.1 – Imagem com as bordas encontradas e buracos preenchidos

Fonte: Ahmed et al. [22]

O método proposto por Kaur [11] foi testado com 120 diferentes imagens de veículos e obteve sucesso em 118 deles, mantendo um percentual de 98.33%. O método implementado neste trabalho para a detecção da área da placa se baseia neste método.

Wafy e Madbouly [22] propõe outra alternativa para a detecção da região da placa. Os autores se baseiam na distribuição semi-simétrica dos pontos de canto nas imagens de carros e placas e nas características morfológicas da região da placa.

O algoritmo de detectar as regiões candidatas consiste em quatro módulos. O primeiro módulo prepara a imagem para detecção aplicando pirâmides gaussianas para suavizar e remover ruídos da imagem. O segundo módulo detecta pontos de canto utilizando o detector de Harris. O terceiro módulo encontra uma linha vertical de simetria se baseando no fato de que uma imagem contendo uma placa de carro tem uma distribuição semi-simétrica de pontos de canto na placa. O quarto módulo procura verticalmente nas linhas simétricas para encontrar a região candidata que tem o maior número de pontos de canto.

A solução proposta por Wafy e Madbouly [22] teve uma taxa de acerto de 97,5% no processo de detecção, com o maior tempo de execução para um dos processos sendo de 0,3s. Com estes resultados seria possível utilizar este método em aplicações de tempo real.

3.4 Segmentação dos caracteres

Em Abtahi et al. [1] foram feitas novas abordagens para a segmentação de caracteres em imagens. De acordo com eles, o método padrão de segmentação baseado em projeção sofre com variações consideráveis na região da placa ao redor dos caracteres, portanto estes autores propuseram duas abordagens. A primeira é feita adaptando um método de aprendizado por reforço, criando um agente que consiga achar os melhores caminhos para a segmentação. A segunda abordagem usa um método híbrido que utiliza a simplicidade e velocidade do método de projeção, mas com o poder do aprendizado por reforço.

3.5 Reconhecimento em sistema embarcado

Com relação ao uso de sistema embarcado para executar o reconhecimento, Arth et al. [3] trabalharam no desenvolvimento de um sistema de reconhecimento de placas de carro em um processador de sinal digital (*Digital Signal Processor*, DSP). *DSPs* são microprocessadores especializados em processar sinais digitais como áudio ou vídeo em tempo real [23]. O processador utilizado neste trabalho específico foi um *Texas Instruments C64* com *1MB* de cache *RAM* e um outro bloco de memória mais lento, *SDRAM* de *16MB*. O processador não possui câmera integrada mas permite a conexão de uma fonte de vídeo analógica ou digital. Na solução implementada foi utilizada uma câmera com resolução de *352x288 pixels*.

Com sua implementação, Arth et al. [3] foram capazes de conseguir localizar a placa em *7,30 ms*, levando mais aproximadamente *1 ms* para identificar cada caractere. Não é informado, no artigo, a taxa de sucesso de cada reconhecimento. Os autores ainda concluem que pelo tempo de detecção da placa ser superior ao tempo de reconhecimento dos caracteres, este algoritmo deve ser melhorado.

3.6 Análise dos trabalhos

Analizando os trabalhos feitos na área, nota-se que por mais que os algoritmos variem, a base da detecção de placas permanece parecida. Eles costumam ser divididos em pelo menos duas partes, a localização da placa e a detecção dos caracteres. Inclusive, Ahmad et al [2] misturou diferentes algoritmos, utilizando a localização de um e a detecção de outro, demonstrando que os dois passos são bem independentes.

Pode-se notar que o reconhecimento de placas de carros é uma área bastante estudada, com diversas abordagens diferentes e grande variação de resultados. Entretanto, um problema que ainda existe é a grande variação nas placas de diferentes países, no estilo, fonte, caracteres utilizados e padrão do texto. A solução deste problema é a criação de um método local de reconhecimento de placas.

4. AMBIENTE DE DESENVOLVIMENTO

Para o desenvolvimento do trabalho foram utilizados alguns componentes externos, tanto de *hardware* quanto de *software*. Foi utilizado o computador *Raspberry Pi* e seu módulo de câmera, a linguagem de programação *Python* e a biblioteca de visão computacional *OpenCV* juntamente com outras bibliotecas auxiliares que permitam integrar o *OpenCV* com a linguagem de programação *Python* e o módulo de camera do *Raspberry Pi*.

4.1 Raspberry Pi

Raspberry Pi é um computador construído em uma placa de circuito do tamanho de um cartão de crédito desenvolvido pela Raspberry Pi Foundation¹. Existem diversos modelos de Raspberry Pi no mercado, o utilizado no trabalho é um dos mais recentes, o Raspberry Pi 3 model B. Foi utilizado o sistema operacional *Raspbian*, que é o sistema operacional oficial suportado pela *Raspberry Pi Foundation*.



Figura 4.1 – O Raspberry Pi 3 model B

O computador é limitado, pois tem um processador quad-core ARMv8 de 1.2GHz e apenas 1GB de memória RAM. Pela limitação do *hardware* é possível que algumas aplicações fiquem mais lentas do que ficariam em um computador mais potente, apesar disso, o Raspberry Pi é um computador bem completo e capaz de exercer todas as funções de um computador normal.

¹<https://www.raspberrypi.org/>

4.2 OpenCV

OpenCV (*Open Source Computer Vision Library*) é uma biblioteca *open source* de visão computacional e aprendizado de máquina. Contém mais de 2500 algoritmos otimizados nessas áreas, incluindo algoritmos clássicos e recentes. A biblioteca é escrita nativamente em C++, e dispõe de interfaces para C, C++, Python, Java e MATLAB, suportando os sistemas operacionais Windows, Linux, Android e Mac OS.²

No desenvolvimento deste trabalho foi utilizada a linguagem de programação *Python*. O motivo da escolha dessa linguagem se dá por ser uma linguagem muito usada para OpenCV, contendo muita documentação. Algumas outras bibliotecas são utilizadas para trabalhar com *OpenCV* e *Python*. A biblioteca *numpy*³ é uma biblioteca de computação científica em *Python*, que inclui funções de processamento numérico e vetores que são utilizados pelo *OpenCV* para representar as imagens.

Para as aplicações de aprendizado de máquina foi utilizada a biblioteca *ml* que é um módulo do *OpenCV*. Esta biblioteca contém um conjunto de classes e métodos para classificação estatística, regressão e agrupamento de dados.⁴

Também foi utilizado o pacote *picamera*⁵, que possui uma interface em *Python* para se comunicar com o módulo de camera do *Raspberry Pi*. É possível utilizar as funções próprias de captura de imagem da camera do *OpenCV*, mas foi optado por utilizar o *picamera*, por ser uma interface mais específica para a camera utilizada.

As escolhas de ferramentas e bibliotecas foram feitas com objetivo de maximizar os resultados ao final do trabalho, para criar um balanço entre facilidade de implementação e qualidade do reconhecimento.

²<http://opencv.org/>

³<http://www.numpy.org/>

⁴<http://docs.opencv.org/2.4/modules/ml/doc/ml.html>

⁵<https://picamera.readthedocs.io/en/release-1.12/>

5. SOLUÇÃO DESENVOLVIDA

O trabalho foi dividido nos quatro passos propostos por S Du [19]: aquisição da imagem, extração da placa, segmentação dos caracteres e reconhecimento dos caracteres. Na figura 5.1 está disposto um exemplo do fluxo a ser seguido

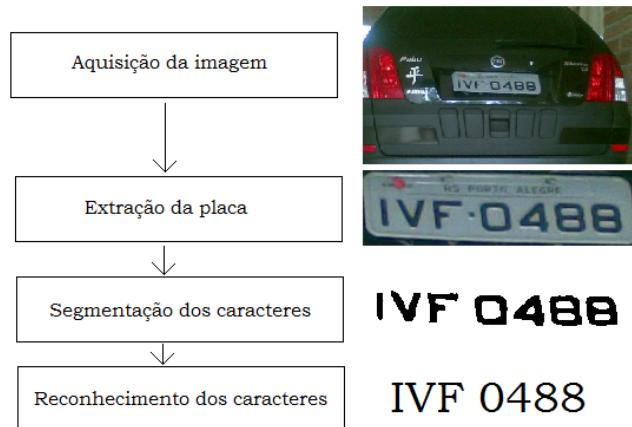


Figura 5.1 – Quatro estágios do reconhecimento de placa

5.1 Placa de Trânsito Brasileira

Segundo o código de trânsito brasileiro [4], os veículos nacionais são identificados por meio de duas placas, dianteira e traseira, com exceção dos veículos de duas ou três rodas, que são dispensados da placa dianteira. As placas são identificadas por uma tarja na parte superior contendo a sigla do estado e o nome do município, e pelo código de identificação único, composto por três letras, seguidas por quatro dígitos, separados por um hífen.

Veículos particulares, de aluguel, oficial, de experiência, de aprendizagem e de fabricante têm suas dimensões de $130mm \times 400mm$ e altura dos caracteres de 63mm (Figura 5.2). Caso a placa não caiba no receptáculo ela pode ser reduzida em até 15%. As placas de motocicleta, motoneta, ciclomotor e triciclos autorizados têm dimensões de $136mm \times 187mm$ e altura de caracteres de 42mm (Figura 5.3).



Figura 5.2 – Placa de um carro



Figura 5.3 – Placa de uma moto

A tipologia dos caracteres das placas utiliza a fonte *Mandatory* (conforme mostra a Figura 5.4), e as placas de categorias diferentes de veículos são diferenciadas pelas suas cores.



Figura 5.4 – Tipografia das placas

5.2 Aquisição das imagens

O primeiro passo para o reconhecimento de uma placa é a extração das imagens. Diversos fatores externos podem afetar o reconhecimento da placa, como a iluminação, a distância e o ângulo da imagem. A escolha da posição onde as imagens serão adquiridas deve ser feita com o objetivo de minimizar esses problemas.

A aquisição das imagens será feita utilizando o módulo de câmera do *Raspberry Pi*. Trata-se uma câmera de 5 *megapixels* capaz de gravar vídeos em 1080p, gerando imagens com dimensão de 1920×1080 *pixels*. Uma das vantagens de utilizar essa combinação do *Raspberry Pi* e seu módulo de câmera é a sua portabilidade. Por serem pequenos e leves, caso a câmera tenha dificuldade em capturar imagens que facilitam o reconhecimento das placas, é possível movê-los e colocá-los em posições privilegiadas para otimizar o processo.

5.3 Extração da placa

A fase de extração é a fase mais importante em um sistema de reconhecimento de placas, porque todas as outras fases dependem da exata extração da área da placa. Essa extração é difícil pois influencia na precisão do sistema como um todo [11]. Muitas dificuldades podem ocorrer durante a fase de extração pelos seguintes motivos:

- A eficiência da extração é afetada pela complexidade da cena.
- Diferentes veículos possuem suas placas em diferentes posições.
- Pode ocorrer ruído durante a captura da câmera.
- Condições do tempo podem influenciar no ruído.
- Hora do dia afeta na luminosidade e resulta em erros de contraste.
- Outros caracteres, quadros e parafusos podem introduzir confusão.
- Mal posicionamento da câmera, ou placa, pode resultar em distorção que afeta na eficiência.
- Luminosidade baixa, ou desigual, imagem desfocada, baixa resolução, reflexão e sombra afetam a eficiência da extração.

O método implementado é baseado em Kaur [11] e segue o seguinte fluxograma:

1. Conversão de RGB para escala de tons de cinza.
2. Remoção de ruído por filtro bilateral.
3. Aumento de contraste usando equalização de histograma adaptativo.
4. Binarização da Imagem.
5. Detecção de borda pelo operador *Sobel*.
6. Operação de dilatação sobre as bordas.

7. Preenchimento de buracos na imagem.
8. Detecção de área de placa candidata por operações morfológicas de abertura e fechamento.
9. Extração da área da placa real.
10. Aprimoramento da região extraída.

5.3.1 Conversão de RGB para escala de tons de cinza

A imagem capturada está no formato RGB. O primeiro passo do pré-processamento é converter essa imagem para a escala de tons de cinza. O objetivo dessa conversão é reduzir o número de cores. Os componentes R, G e B são separados do valor de cor de 24 bits de cada pixel (i,j) , e um valor de 8 bits em cinza é calculado. Uma imagem capturada e convertida para tons de cinza pode ser vista na Figura 5.5.



Figura 5.5 – Imagem em escala de tons de cinza.

5.3.2 Remoção de ruído por filtro bilateral

O filtro bilateral é um filtro não linear, que preserva as arestas e reduz ruído. Ele funciona substituindo o valor de intensidade de cada *pixel* em uma imagem por uma média ponderada dos valores de intensidade dos *pixels* próximos. O objetivo básico da filtragem é remover ruído e distorção da imagem. O ruído pode ocorrer durante a captura pela câmera ou pelas condições do tempo. No método que Kaur [11] propõe, o filtro bilateral iterativo é utilizado para remover ruído. Ele é não linear, provê mecanismo para remoção de ruído

enquanto preserva bordas mais efetivamente que o filtro mediano. O filtro mediano funciona substituindo o valor de cada *pixel* pela mediana dos *pixels* vizinhos. Na Figura 5.6 pode-se ver a imagem em tons de cinza com a remoção de ruído por filtro bilateral aplicada.



Figura 5.6 – Aplicação de filtro bilateral em uma imagem em escala de tons de cinza

É necessário utilizar um filtro de remoção de ruído que tenha capacidade de preservar as bordas, pois a detecção de bordas é um passo muito importante na extração da placa. A detecção de bordas será abordada na Seção 5.3.5.

5.3.3 Aumento de contraste usando equalização de histograma adaptativo

Contraste é definido como a diferença entre o nível mais baixo e alto de intensidade. Equalização de histograma é o método para distribuir de forma mais efetiva o histograma de *pixels*. Equalização de histograma adaptativo mostra melhor contraste em relação a equalização de histograma. Na Figura 5.7 foi aplicado o aumento de contraste utilizando equalização de histograma adaptativo sobre a imagem anterior (Figura 5.6).



Figura 5.7 – Aumento de contraste usando equalização de histograma adaptativo

5.3.4 Binarização da Imagem

Nesta operação a imagem de escala de cinza pré-processada é convertida em imagem binária. Para isso é utilizada uma técnica de limiarização. Na implementação de Kaur [11], foi escolhida uma técnica de limiarização por limiar global, utilizando o método de *Otsu* para calcular esse limiar.

O algoritmo do método de *Otsu* assume que a imagem contém duas classes de *pixels* seguindo um histograma bi-modal. Ele calcula o limiar ótimo separando as duas classes para que o seu espalhamento combinado seja mínimo. Tendo o limiar calculado, a imagem de escala de cinza subtraída é convertida em imagem em preto e branco.

O objetivo final desse passo, é fazer com que, na imagem binarizada, haja uma diferença entre a cor da placa e a cor do resto do carro, deixando a placa demarcada na imagem. O resultado dessa binarização pode ser visto na Figura 5.8.



Figura 5.8 – Imagem Binarizada

5.3.5 Detecção de borda pelo operador Sobel

Neste passo as bordas da imagem são detectadas pelo operador de *Sobel*. A detecção de bordas em uma imagem diminui显著mente a quantidade de dados a ser processados e remove informações desnecessárias, enquanto preserva importantes propriedades estruturais na imagem [7].

Para que esse passo tenha sucesso, é necessário que, na binarização, a placa e o resto do carro tenham obtido valores diferentes. Com isso, a detecção de bordas terá como objetivo desenhar o contorno da placa do carro, incluindo, ou não, outros elementos não relevantes da imagem. O resultado da detecção de bordas aplicada à imagem binarizada é mostrado na Figura 5.9.

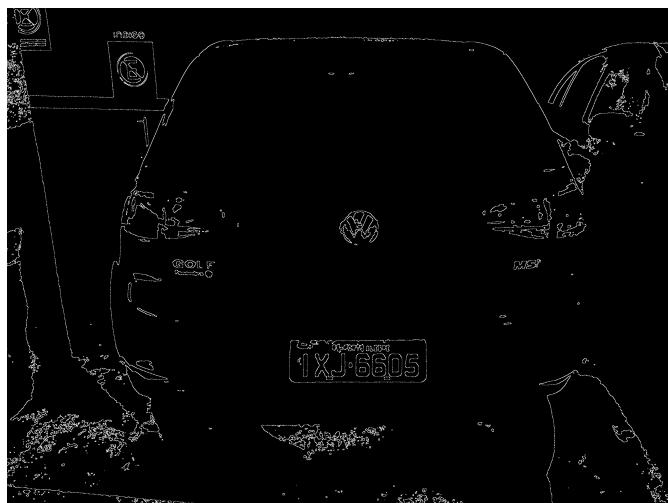


Figura 5.9 – Detecção de borda pelo operador Sobel

5.3.6 Operação de dilatação sobre as bordas

Tendo as bordas detectadas deve-se aplicar uma operação de dilatação sobre a imagem processada. O motivo deste passo é deixar as bordas detectadas mais fortes. Na operação anterior é possível que existam pequenos furos nas bordas geradas, segmentando a linha desenhada em diversos pedaços. Aplicando a dilatação sobre essas bordas, as linhas se fecham deixando as bordas mais definidas.

As bordas precisam ser bem definidas para garantir que o retângulo gerado pela placa esteja completo. Espaços abertos nessa área podem causar complicações nas etapas seguintes. Na Figura 5.10 é possível ver as bordas dilatadas e mais definidas.



Figura 5.10 – Dilatação morfológica

5.3.7 Preenchimento de buracos na imagem

Com as bordas dilatadas e bem definidas é feito o preenchimento das áreas fechadas na imagem. Se as bordas ao redor da placa estiverem corretamente demarcadas, nesta etapa, sobre a área da placa, haverá um retângulo completo fechado. Essa é a etapa da detecção das áreas de placa onde é possível identificar visualmente os candidatos a placa. Estes são os pontos onde existem grandes conjuntos de *pixels*, as áreas com bordas fechadas que foram preenchidas. Na Figura 5.11 pode-se ver a imagem com os buracos preenchidos.



Figura 5.11 – Após preenchimento dos buracos

5.3.8 Detecção de Áreas de Placa Candidatas por Operações Morfológicas de Abertura e Fechamento

Com as operações morfológicas, os objetos indesejados na imagem são removidos. As operações de abertura e fechamento são usadas para a detecção exata da área da placa. Essa etapa acaba se tornando um pouco limitada, pois o elemento estruturante deve ser de certa forma proporcional ao tamanho da placa na imagem. Com um elemento estruturante muito pequeno, sobram muitos candidatos falsos, pois eles não são removidos nas operações morfológicas. Com um elemento estruturante muito grande, a área da placa pode ser removida junto com o ruído. A Figura 5.12 contém as regiões candidatas da placa.

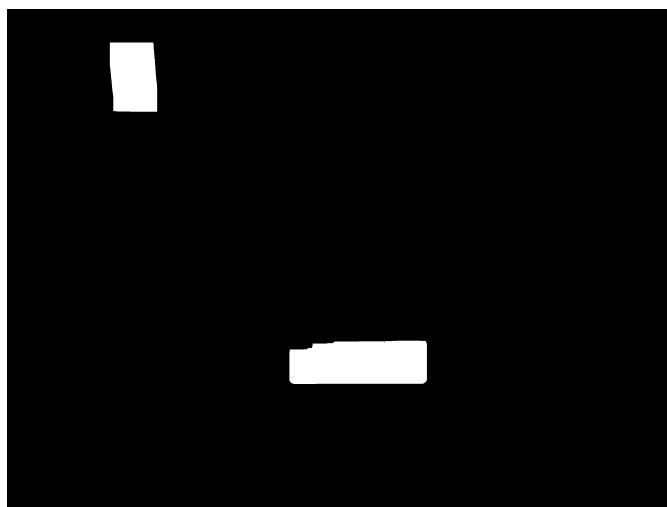


Figura 5.12 – Detecção das áreas candidatas da placa

5.3.9 Extração da área da placa real

Ao localizar as regiões candidatas, pode-se tanto encontrar uma placa, quanto regiões falsas, onde não existe placa. Para excluir estas regiões, Jia [10] sugere que características sejam extraídas para diferenciar corretamente as regiões de placa por outras.

Diversas características podem ser utilizadas com este propósito. Elas incluem o tamanho da região, a altura e largura da região, a orientação dos caracteres posteriormente extraídos, a intensidade das bordas e a posição da região.

Após a detecção da área da placa, essa área é extraída da imagem. Então os índices de linha e coluna da área da placa são encontrados por análise de componentes conectados. Na Figura 5.13 podemos ver a imagem placa extraída da imagem original.

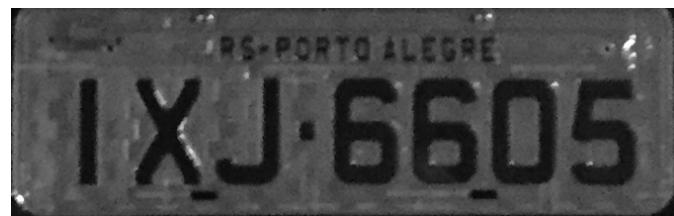


Figura 5.13 – Placa extraída

5.3.10 Aprimoramento da região extraída

A placa extraída deve ser convertida para um formato que seja processável pelo algoritmo. É aplicada sobre a imagem a conversão para tons de cinza seguida pela binarização. Apesar de a placa binarizada conter bastante ruído, não são feitas mais operações sobre a imagem da placa. O método seguinte, de segmentação de caracteres, Seção 5.4, vai lidar com os ruídos. O motivo da não utilização de algoritmos de remoção de ruído é para não deformar os caracteres a serem lidos pelo reconhecedor de caracteres na Seção 5.5. A placa aprimorada e pronta para segmentação dos caracteres pode ser vista em 5.14.

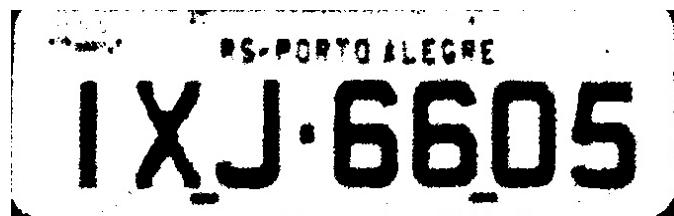


Figura 5.14 – Placa aprimorada

5.4 Segmentação dos caracteres

O terceiro passo para o reconhecimento da placa é a segmentação dos caracteres, a qual consiste na extração dos caracteres utilizando estratégias como projetar as suas informações de cores, rotulá-los ou comparar suas posições com modelos. A placa extraída no passo anterior pode conter problemas de inclinação ou iluminação, mas o algoritmo de segmentação deve superar todos esses problemas com pré-processamento [19].

Os principais algoritmos utilizados para a segmentação de caracteres são: segmentação utilizando conectividade de *pixels*, segmentação utilizando perfis de projeção, segmentação utilizando conhecimento anterior dos caracteres, segmentação utilizando contorno dos caracteres e segmentação utilizando características combinadas [19].

O método une a segmentação utilizando a conectividade de *pixels* com algum conhecimento anterior dos caracteres. É feita uma detecção de *pixels* conectados para

detectar quais elementos são possíveis caracteres e aplicando o conhecimento sobre a placa para descartar pontos muito pequenos ou muito próximos às bordas da placa.

O primeiro passo é o preenchimento dos espaços em branco da placa previamente tratada. Este passo existe para evitar que, ao detectar os contornos dos caracteres por meio da conexão dos *pixels*, não aconteça uma dupla detecção em caracteres que contenham espaços internos abertos. Por exemplo o número 0, que possui um contorno interno e um contorno externo. Na Figura 5.15 é possível ver a placa com os caracteres preenchidos por este passo.

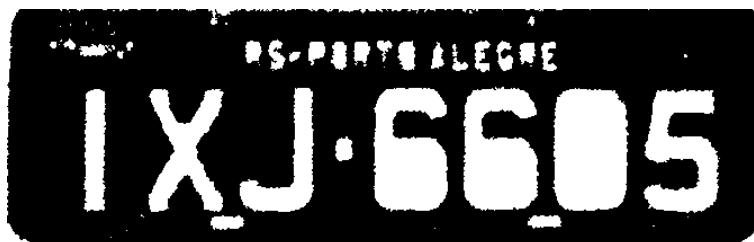


Figura 5.15 – Preenchimento dos caracteres

Neste ponto são detectados os contornos com base na conexão dos *pixels*. Muitos *pixels* conectados não fazem parte de nenhum caractere da placa. Para descartar os *pixels* indesejados, é feito um filtro baseado no tamanho dos contornos e na sua posição. A altura dos contornos não pode ser menor do que metade da altura da imagem completa. Isso descarta os conjuntos pequenos de *pixels* indesejados. Outro critério é que o contorno encontrado não pode estar encostado na borda da imagem. Em uma placa corretamente extraída os caracteres nunca vão estar encostando na borda. Este critério exclui sombras ou áreas que sobram na lateral da placa na extração, que poderiam ser erroneamente confundidas com caracteres.

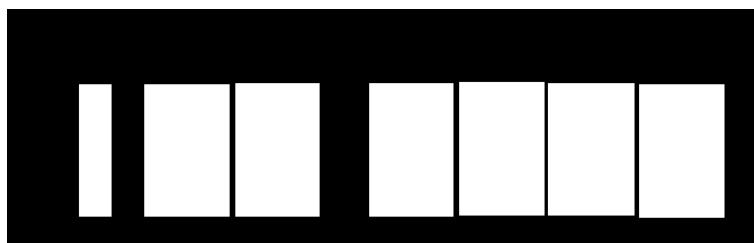


Figura 5.16 – Máscara gerada

Encontrados os contornos dos caracteres corretos, uma máscara, mostrada na Figura 5.16, é feita a partir deles. Essa máscara será aplicada sobre a imagem da placa extraída original, recortando os *pixels* marcados por ela. Os caracteres recortados por essa operação serão extraídos, ordenados com base em sua posição x, e enviados para a etapa de reconhecimento de caracteres. A Figura 5.17 mostra os caracteres a serem extraídos.



Figura 5.17 – Caracteres a serem extraídos

5.5 Reconhecimento dos caracteres

O último passo para identificar a placa é o reconhecimento ótico dos caracteres extraídos. Um dos maiores desafios da extração de caracteres é o fato de os caracteres extraídos não terem o mesmo tamanho e espessura, devido ao *zoom* da câmera. Outro problema, ao criar um reconhecedor de propósito geral, é a variedade nas fontes das placas ao redor do mundo [19]. Este segundo problema não será um obstáculo para este trabalho pois só há interesse em reconhecer placas brasileiras. Outro desafio é o reconhecimento de caracteres semelhantes, como o D e o 0 [8]. Esse problema será mitigado com o conhecimento prévio das placas brasileiras, uma vez que se sabe onde é possível haver letras e onde é possível haver números.

Tendo em mente os possíveis problemas, foi implementado um reconhecedor de caracteres especificamente para a solução deste problema. Foi utilizado o algoritmo *K-Nearest Neighbors* para classificar os caracteres possíveis e utilizados os caracteres na fonte *Mandatory* dispostos na Figura 5.4 como dados de treinamento. O *feature space* consiste em um caractere apenas por classe, portanto é utilizado apenas um vizinho para classificar o novo dado.

Além de focar na fonte na qual as placas estão escritas, o reconhecedor é calibrado também para o padrão. Sabendo que a placa de trânsito tem três letras e quatro números, mantendo esta ordem, o reconhecedor nunca vai confundir uma letra com um número. Para conseguir este resultado são utilizados dois diferentes *feature spaces*, um para os números e um para as letras. Assim, um caractere novo só pode ser classificado entre os possíveis candidatos.

Para que a classificação tenha sucesso, tanto as imagens de teste quanto as imagens a serem classificadas devem ter o mesmo tamanho. O *feature space* deve ter N dimensões não variáveis. Para isso as imagens são tratadas para que tenham todas o tamanho de 100x100 *pixels*, preenchendo o espaço restante com *pixels* brancos.

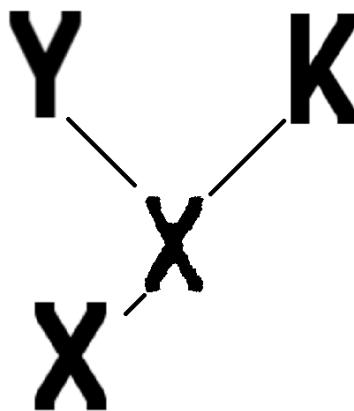


Figura 5.18 – Exemplo de classificação de caractere extraído

Na Figura 5.18 são mostrados os três vizinhos mais próximos de um determinado caractere retirado do processamento de uma imagem. A sua projeção visual é hipotética e foi criada para representar de uma forma mais didática a seleção dos vizinhos. Pode-se reparar a semelhança entre os caracteres e entender porque eles foram selecionados pelo algoritmo ao visualizar a imagem.

5.6 Otimização com Multiprocessamento em Pipeline

Ao processar imagens em um sistema embarcado, é preciso analisar e se adaptar às limitações do sistema. O computador *Raspberry Pi* é bastante limitado, possuindo um processador *Quad Core 1.2GHz Broadcom BCM2837 64bit CPU*. Por essa razão, para maximizar o ganho, foi necessário utilizar técnicas de otimização. Dado o fato de ele possuir um *Quad Core*, contendo quatro núcleos, é possível executar quatro processos em paralelo sem perder desempenho.

A técnica utilizada na extração da placa e reconhecimento dos caracteres é uma técnica sequencial, que segue uma série de passos. Sabendo disso, é possível implementar um *Pipeline* de execução. Um *Pipeline* é um conjunto de elementos de processamento de dados conectados em série, onde a saída de um elemento é a entrada do próximo. *Pipelines* podem ser executados em paralelo, pois o processamento de um não depende do outro.

As etapas de processamento de uma imagem foram divididas em quatro partes com tempo de processamento semelhante. Essas quatro etapas vão ser processadas cada uma em um núcleo diferente do processador do *Raspberry Pi*, otimizando o seu tempo. O *Pipeline* pode ser visto na Figura 5.19 encontrada abaixo:

Tempo ->	1	2	3	4	5	6	7	8
Processo 1	Imagen 1	Imagen 2	Imagen 3	Imagen 4	Imagen 5			
Processo 2		Imagen 1	Imagen 2	Imagen 3	Imagen 4	Imagen 5		
Processo 3			Imagen 1	Imagen 2	Imagen 3	Imagen 4	Imagen 5	
Processo 4				Imagen 1	Imagen 2	Imagen 3	Imagen 4	Imagen 5

Figura 5.19 – Pipeline implementado para otimização

Com a utilização do *Pipeline*, o tempo de processamento da primeira imagem vai continuar o mesmo, mas para as próximas imagens, o tempo vai diminuir. Nessas etapas seguintes, se o tempo de processamento de cada etapa for igual, cada imagem vai terminar de ser processada no tempo de processamento de uma etapa, desconsiderando o tempo da comunicação entre os processos.

Os quatro processos foram divididos como se segue:

1. O primeiro processo comprehende desde a aquisição da imagem até a aplicação do filtro bilateral.
2. O segundo processo comprehende desde a equalização do histograma até o preenchimento dos espaços.
3. O terceiro processo apenas executa a abertura morfológica.
4. O quarto processo comprehende desde o fechamento morfológico até o reconhecimento dos caracteres.

Os procedimentos mais lentos são a aplicação do filtro bilateral e as operações de abertura e fechamento, por isso foram divididos em processos diferentes. Os demais procedimentos foram também divididos de forma que os processos tenham tempo de processamento semelhante.

6. RESULTADOS OBTIDOS

Para a análise da solução desenvolvida e identificação de problemas foram feitos testes variados. Este capítulo foi dividido em seções que representam as descobertas mais relevantes ao longo da pesquisa e desenvolvimento. A Seção 6.1 vai descrever os resultados obtidos na experimentação com a extração da placa e dos caracteres nas imagens. São os casos onde o algoritmo detectou a presença de um carro e identificou uma placa. A Seção 6.2 vai abordar os resultados obtidos no reconhecimento dos caracteres, avaliar se a placa e caracteres extraídos foram lidos corretamente. Por fim, a Seção 6.3 vai abordar como performa o algoritmo no sistema embarcado no computador *Raspberry Pi*.

Foram coletadas imagens de 48 veículos distintos para analisar a capacidade de reconhecimento do sistema. A tabela 6.1 mostra o número de imagens processadas, o número de imagens onde a placa foi encontrada com sucesso e o número de imagens onde foi possível reconhecer o valor da placa corretamente. Existe uma diferença entre os casos onde não foi possível detectar uma placa e os casos onde foi possível mas ela foi detectada erroneamente. No segundo há um falso positivo, identificando um veículo que não estava no local.

Tabela 6.1 – Resultados obtidos

Placas	Resultado
Total de Imagens	48
Imagens onde a placa foi encontrada	22
Imagens onde a placa foi reconhecida corretamente	19

6.1 Extração da placa e caracteres

Das 48 imagens adquiridas, apenas em 22 imagens foi possível extrair a placa e os caracteres. Desses 48 imagens, 27 foram adquiridas ao ar livre, durante o dia e 21 foram adquiridas em local fechado, com iluminação baixa. Com essa diferenças foi possível analisar o sistema nos dois tipos de ambientes. Nas imagens ao ar livre, com boa iluminação, houve um percentual de 55% de acerto, enquanto nas imagens em ambiente fechado, com baixa iluminação, houve um percentual de acerto de 33%. Os resultados comparativos podem ser vistos na Tabela 6.2. Os exemplos são poucos para tirar conclusões, mas indicam que pode existir mais dificuldade para reconhecer placas em imagens mais escuras.

O algoritmo de extração da placa é limitado com relação a distância que o carro está na imagem. Na etapa de abertura e fechamento morfológicos, que tem o objetivo de detectar as áreas de placas candidatas, o tamanho do elemento estruturante influencia

Tabela 6.2 – Resultados obtidos em diferentes ambientes

Placas	Resultado
Imagens ao ar livre com boa iluminação	27
Total de acertos com boa iluminação	15
Percentual de acertos com boa iluminação	55%
Imagens em ambiente fechado com baixa iluminação	21
Total de acertos com baixa iluminação	7
Percentual de acertos com baixa iluminação	33%

diretamente a qualidade da extração. Se utilizado um elemento estruturante muito grande, é possível que na etapa de abertura, a placa seja erodida junto com o ruído. Se utilizado um elemento estruturante muito pequeno, muitas regiões sobram, diminuindo a performance do reconhecimento.

Outro problema foi observado na extração de placas de carros cuja cor da lataria é muito semelhante à cor da placa. Carros de cor prata ou branco são maioria nos casos em que a placa não foi encontrada. Isso ocorre pois, na fase de limiarização, não foi possível dividir o tom da lataria e da placa em grupos diferentes. Foram feitos testes utilizando limiarização adaptativa para comparar os resultados, mas ela obteve menos sucesso que utilizando limiar global, em geral. Detalhes significativos de cores diferentes também dificultam a extração. Na Figura 6.1 há um exemplo de veículo que o sistema teve dificuldade de reconhecer por estes motivos.



Figura 6.1 – Veículo que o algoritmo tem dificuldade de reconhecer

6.2 Reconhecimento dos caracteres

Inicialmente foi utilizado o software *Tesseract* para fazer o reconhecimento dos caracteres, mas os resultados utilizando a ferramenta pura não foram bons. Mesmo limitando

os caracteres possíveis, informando ao *software* se o caractere seria um número ou uma letra, ainda havia dificuldade no reconhecimento.

Com a substituição do reconhecedor de caracteres *Tesseract* para o implementado com o algoritmo *K-Nearest Neighbors* os resultados foram bem melhores, obtendo sucesso no reconhecimento de 86% das placas extraídas. Até mesmo em imagens de placas inclinadas, como é o caso da Figura 6.2, foi possível identificar os caracteres.



Figura 6.2 – Placa inclinada que o sistema foi capaz de identificar

Das 22 placas extraídas, três delas o reconhecedor de caracteres não teve capacidade de ler. Dois destes casos foram causados por problemas na imagem e o outro foi causado por semelhança de caracteres.

Placas com reflexo causado pelo ambiente, como a luz solar, podem impactar no reconhecimento. O sol refletido no caractere da imagem pode dificultar o processo de limiarização. Neste processo, ao diferenciar os caracteres escuros do fundo cinza, a cor do caractere na imagem fica mais clara, sendo ele parcialmente confundido com fundo. Na figura 6.3 é possível observar uma placa que o reflexo impediu o reconhecimento correto.



Figura 6.3 – Placa com caractere mal reconhecido por causa de reflexo

Em placas desgastadas, pelo tempo ou pelo carro ter passado por condições climáticas que o degradaram, também houve dificuldade de reconhecimento. Assim como no caso anterior, os caracteres ficaram desfigurados na fase da limiarização, por terem uma variação muito grande nos tons de cinza. Isso faz com que parte do caractere seja confundido com o fundo. Imagem de placa de carro adquirida onde a placa estava desgastada dificultando o reconhecimento pode ser vista na Figura 6.4.



Figura 6.4 – Placa desgastada

Somente em um caso, nas imagens de teste adquiridas, ocorreu um erro de reconhecimento de caractere por motivos não causados pelo ambiente ou qualidade da placa. Em uma imagem adquirida com a placa levemente inclinada, houve confusão, por parte do algoritmo, entre os caracteres "O" e "Q". Estes caracteres já são bastante semelhantes por si só, mas é possível, também, que o fato de a placa estar inclinada tenha amplificado o erro. A imagem que não foi reconhecida corretamente, ao confundir o "O" e o "Q", pode ser vista na Figura 6.5.



Figura 6.5 – Placa que confundiu os caracteres O e Q

Apesar dos casos em que houve erro, o resultado do reconhecedor de caracteres foi satisfatório. Mesmo tendo sido utilizados apenas um exemplo para cada caractere na fase de treinamento, somente em um caso houve confusão no reconhecimento dos caracteres bem segmentados. Com mais exemplos seria possível um reconhecimento ainda mais preciso.

6.3 Performance em sistema embarcado

A implementação do algoritmo obteve maus resultados com relação a performance quando executada como sistema embarcado no *Raspberry Pi*. O seu tempo de execução para processar cada imagem foi muito alto, tornando o software inviável para o uso em casos reais.

Dado o fato de a solução desenvolvida ser dividida em etapas, é possível avaliar quais os passos levam mais tempo para serem executados, que podem ser considerados os gargalos da aplicação. Em uma visão mais ampla, fica claro que a etapa de extração da placa é a que necessita de mais recursos computacionais. Isso se dá principalmente

por ter que processar uma imagem maior que a imagem da placa. O tempo utilizado para processar a placa extraída é, relativamente, bem menor.

Para melhor analisar os resultados comparativos, foram calculadas as médias de tempo de execução de cada etapa em múltiplas imagens e calculados seus valores percentuais em relação ao tempo total de processamento de uma placa. Esses valores independem do *hardware* utilizado, permitindo uma visão diferente dos resultados. Os resultados comparativos podem ser vistos na Tabela 6.3, as etapas estão dispostas na ordem em que são executados pela aplicação. A divisão das etapas em processos para fazer o *Pipeline* foi feita com base nestes dados.

Tabela 6.3 – Resultados relativos da fase de extração

Etapa	Tempo Percentual
Conversão para tons de cinza	9%
Filtro Bilateral	22%
Equalização do Histograma	9%
Limiarização	5%
Detecção de Bordas	10%
Dilatação	6%
Preenchimento de espaços	5%
Abertura	16%
Fechamento	16%
Extração das Regiões	2%

Ao aplicar o *Pipeline* em múltiplos processos, com o objetivo de utilizar mais núcleos do processador do *Raspberry Pi*, houve melhora, mas não muito significativa, no tempo. Por motivos de comparação, foi feito uma série de testes no *Raspberry Pi* e em um *Macbook Pro*, com processador *Intel i7-4770HQ*. Foi avaliado o tempo de processamento sequencial, sem utilizar múltiplos processos de 10 imagens, e o tempo de processamento, utilizando o *Pipeline* de processos em 10 imagens. Cada um desses testes foi feito 10 vezes e o seu tempo médio foi calculado. Os resultados podem ser vistos na Figura 6.6 abaixo.

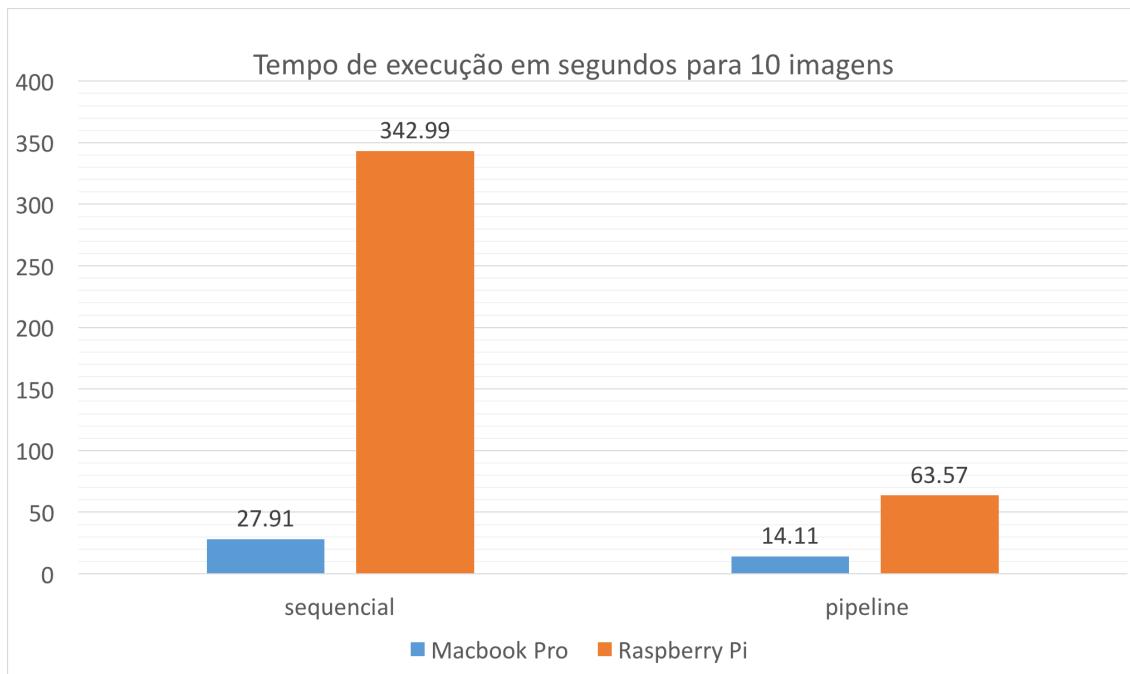


Figura 6.6 – Performance da aplicação

Em teste com a utilização do módulo de câmera em simulação próxima do real, não foi possível identificar placas de veículos por causa da performance. As imagens adquiridas ficam muito espaçadas, entre um processamento e o outro, e muito raramente consegue-se uma imagem que contenha um carro, deixando muitos passarem sem identificação.

7. CONCLUSÃO E TRABALHOS FUTUROS

O reconhecimento automático de placas de carro mostrou-se ser uma área bastante vasta, com diversos estudos sobre o tema e variadas técnicas para alcançar seu objetivo. Dos trabalhos relacionados pesquisados não foi encontrado nenhum que tenha sucesso em todas as suas tentativas, e os estudos comparativos levam a crer que não há uma técnica que funcione bem para todos os casos.

A solução desenvolvida nesse trabalho obteve baixa performance no computador *Raspberry Pi*, podendo ser vista apenas como um protótipo. O fato de ele ser um computador de propósito geral contribuiu com essa baixa performance. Um computador especializado em processamento de imagens, que possua uma placa de vídeo, teria resultados mais satisfatórios.

Com relação à quantidade de placas corretamente reconhecidas, o software também não teve uma taxa de acertos muito alta, ficando abaixo dos 50%. O baixo resultado se dá, principalmente, pela parte de extração da placa na imagem, onde a maioria dos erros se concentra. Como as etapas do processamento são bem distintas, é possível substituir a extração da placa por outras técnicas com facilidade, podendo assim, facilmente avaliar outras estratégias.

Como trabalhos futuros fica sugerida a aplicação do software desenvolvido em outros sistemas embarcados com maior poder de processamento que o *Raspberry Pi*. Um computador especializado no processamento de imagens teria resultados melhores que um computador de propósito geral.

Outra maneira de combater o problema de performance que pode vir a ser estudado seria uma maneira de analisar a imagem antes do processamento. Ao processar um vídeo quadro a quadro, muitas imagens que são processadas não vão ter um bom resultado devido a fatores externos e a posição do carro na imagem. Se for possível fazer uma análise de alta performance na imagem, para avaliar se ela está em boas condições para ser processada ou não, seria possível otimizar o processo.

Outro tema proposto como trabalho futuro, para melhorar a eficácia do reconhecimento, seria otimização da técnica para poder extrair com mais qualidade placas de carros em diferentes distâncias. Com essa melhoria é possível analisar mais imagens em um vídeo de um carro que se locomove em direção a câmera, podendo ter mais precisão no reconhecimento.

REFERÊNCIAS BIBLIOGRÁFICAS

- [1] Abtahi, F.; Zhu, Z.; Burry, A. M. "A deep reinforcement learning approach to character segmentation of license plate images". In: Machine Vision Applications (MVA), 2015 14th IAPR International Conference on, 2015, pp. 539–542.
- [2] Ahmad, I. S.; Boufama, B.; Habashi, P.; Anderson, W.; Elamsy, T. "Automatic license plate recognition: A comparative study". In: 2015 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT), 2015, pp. 635–640.
- [3] Arth, C.; Limberger, F.; Bischof, H. "Real-time license plate recognition on an embedded dsp-platform". In: 2007 IEEE Conference on Computer Vision and Pattern Recognition, 2007, pp. 1–8.
- [4] BRASIL. "Lei nº 9.503", 23 de setembro de 1997.
- [5] Efford, N. "Digital image processing: a practical introduction using java". Addison-Wesley Longman Publishing Co., Inc., 2000.
- [6] Gonzalez, R.; Wintz, P. "Digital image processing", 1977.
- [7] Ha, P. S.; Shakeri, M. "License plate automatic recognition based on edge detection". In: Artificial Intelligence and Robotics (IRANOPEN), 2016, 2016, pp. 170–174.
- [8] Ho, C.-Y.; Lin, H.-Y.; Wu, L.-T. "Intelligent speed bump system with dynamic license plate recognition". In: 2016 IEEE International Conference on Industrial Technology (ICIT), 2016, pp. 1669–1674.
- [9] Intel. "Developer zone". Acesso em 21/05/2017.
- [10] Jia, W.; Zhang, H.; He, X. "Region-based license plate detection", *Journal of Network and computer Applications*, vol. 30–4, 2007, pp. 1324–1333.
- [11] Kaur, S.; Kaur, S. "An efficient approach for number plate extraction from vehicles image under image processing", *International Journal of Computer Science and Information Technologies*, vol. 5–3, 2014, pp. 2954–2959.
- [12] MOHIT BANSAL, BINAY BINOD KUMAR, P. V. "Designing of licensed number plate recognition system using hybrid technique from neural network & template matching", *IEEE*, 2015, pp. 1–6.
- [13] Mohri, M.; Rostamizadeh, A.; Talwalkar, A. "Foundations of machine learning". MIT press, 2012.
- [14] OpenCV. "Image thresholding". Acesso em 21/05/2017.

- [15] OpenCV. “Morphological operations”. Acesso em 21/05/2017.
- [16] OpenCV. “Understanding k-nearest neighbour”. Acesso em 21/05/2017.
- [17] Owens, R. “Mathematical morphology; from computer vision lectures”. Acesso em 11/06/2017, outubro 1997.
- [18] PSDB. “Marchezan destaca propostas para a segurança pública em porto alegre”. Acesso em 19/11/2016.
- [19] S Du, B. “Automatic license plate recognition (alpr): A state of the art review”, *Circuits and Systems for Video Technology, IEEE Transactions on*, –99, 2013, pp. 311–325.
- [20] Serro, R. M. “Detecção e reconhecimento de placas de carros em imagens digitais”, 2012.
- [21] Tomasi, C.; Manduchi, R. “Bilateral filtering for gray and color images”. In: Computer Vision, 1998. Sixth International Conference on, 1998, pp. 839–846.
- [22] Wafy, M.; Madbouly, A. M. “Efficient method for vehicle license plate identification based on learning a morphological feature”, *IET Intelligent Transport Systems*, 2016.
- [23] Yovits, M. C. “Advances in computers”. Indianapolis, Indiana: Academic Press, 1993, vol. 37.