

INSTITUTO FEDERAL DO ESPÍRITO SANTO  
CURSO SUPERIOR DE ENGENHARIA ELÉTRICA

**GUSTAVO ORNÉLAS PEREIRA**

**PROPOSTA DE SISTEMA DE DETECÇÃO E RECONHECIMENTO DE ROTAS DE  
VEÍCULO DE TRANSPORTE COLETIVO, TIPO ÔNIBUS, PARA AUXÍLIO A  
PESSOA COM DEFICIÊNCIA VISUAL**

VITÓRIA

2022

**GUSTAVO ORNÉLAS PEREIRA**

**PROPOSTA DE SISTEMA DE DETECÇÃO E RECONHECIMENTO DE ROTAS DE  
VEÍCULO DE TRANSPORTE COLETIVO, TIPO ÔNIBUS, PARA AUXÍLIO A  
PESSOA COM DEFICIÊNCIA VISUAL**

Trabalho de Conclusão de Curso apresentado  
à Coordenadoria do Curso de Engenharia  
Elétrica do Instituto Federal do Espírito Santo,  
como requisito parcial para a obtenção do  
título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Me. Arnaldo Paterline  
Togneri

VITÓRIA

2022

Dados Internacionais de Catalogação na Publicação (CIP)  
(Biblioteca Nilo Peçanha do Instituto Federal do Espírito Santo)

P436p Pereira, Gustavo Ornélas.

Proposta de sistema de detecção e reconhecimento de rotas de veículo de transporte coletivo, tipo ônibus, para auxílio a pessoa com deficiência visual / Gustavo Ornélas Pereira – 2022.

85 f. : il. ; 30 cm

Orientador: Arnaldo Togneri Paterline.

Monografia (graduação) – Instituto Federal do Espírito Santo, Coordenadoria de Engenharia Elétrica, Curso Superior de Engenharia Elétrica, 2022.

1. Engenharia elétrica. 2. Sistema de controle inteligente -- Transporte. 3. Sistemas de reconhecimento de padrões. 4. Ônibus. 5. Pessoas com deficiência visual – Orientação e mobilidade – Transporte. 6. Deficientes visuais. 7. Acessibilidade ao transporte local. I. Paterline, Arnaldo Togneri. II. Instituto Federal do Espírito Santo. III. Título.

CDD 21 – 621.3

Elaborada por Ronald Aguiar Nascimento – CRB-6/MG – 3.116

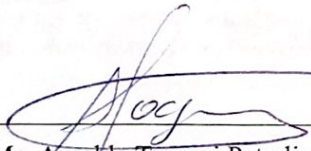
GUSTAVO ORNELAS PEREIRA

**PROPOSTA DE SISTEMA DE DETECÇÃO E RECONHECIMENTO DE ROTAS DE  
VEÍCULO DE TRANSPORTE COLETIVO, TIPO ÔNIBUS, PARA AUXÍLIO A PESSOA  
COM DEFICIÊNCIA VISUAL**

Trabalho de Conclusão de Curso apresentado à Coordenadoria de Engenharia Elétrica do Instituto Federal do Espírito Santo, como requisito parcial para obtenção do título de Bacharel em Engenharia Elétrica.


Aprovado em 28 de Julho de 2022

**COMISSÃO EXAMINADORA**



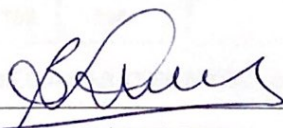
---

Prof. Me. Arnaldo Togneri Paterline  
Instituto Federal do Espírito Santo



---

Prof. Dr. Leandro Bueno  
Instituto Federal do Espírito Santo



---

Prof. Dr. Reginaldo Barbosa Nunes  
Instituto Federal do Espírito Santo

## **AGRADECIMENTOS**

Agradeço, primeiramente, à Deus por toda força e perseverança que me foi dada nesta fase final da graduação.

Agradeço aos meus pais, Maurina e Augusto, à minha irmã, Caroline, à minha companheira, Gisele, à minha tia Tereza, e aos meus tios Nivaldo e Elizabete por todo suporte e compreensão fundamentais na minha caminhada.

Agradeço aos amigos que fiz durante a graduação, aos amigos de longa data e aos colegas de trabalho pelas palavras de incentivo.

Agradeço também ao meu orientador, Arnaldo Togneri, pela preocupação em fazer deste trabalho uma oportunidade de aprendizado da vida cotidiana e da vida acadêmica.

Sem o apoio de todos vocês esse momento não seria possível.

## RESUMO

De acordo com o Instituto Brasileiro de Geografia e Estatística, a baixa acuidade visual é o tipo de deficiência mais frequente no país. Para identificar quando um ônibus está se aproximando a audição exerce papel complementar à visão, no entanto saber a rota do ônibus é um grande desafio para pessoa com deficiência visual. Para oferecer uma alternativa de auxílio, o objetivo desta pesquisa foi avaliar parâmetros do processamento digital de imagens que influenciem na detecção e reconhecimento de rotas de ônibus. Nesse sentido, a metodologia aplicada consistiu em verificar a influência de parâmetros pertinentes ao processamento de imagens na segmentação da face do ônibus e no reconhecimento da rota. Após análise, foram realizados testes utilizando *HaarCascade*, ajuste de perspectiva, suavização, realce de bordas através do detector de bordas *Canny*, erosão, e o *Tesseract*. Foram testadas 6 combinações de parâmetros diferentes para reconhecimento da rota do ônibus em vídeos. O vídeo com a maior taxa de detecção teve 88,5% dos *quadros* com face de ônibus identificada, enquanto as maiores taxas de reconhecimento e de acurácia da rota foram 80% e 80% dos vídeos, respectivamente, com média de 4 *quadros* por segundo.

**Palavras-chave:** Sistema. Leitura automática de rota. Deficiente visual. Ônibus de transporte urbano. Reconhecimento de caracteres.

## ABSTRACT

According to the Instituto Brasileiro de Geografia e Estatística, low visual acuity is the most frequent disability in the country. To identify when the bus is approaching hearing is complement to vision, however knowing the bus route it is a big challenge to the visually impaired. In order to offer an alternative to aid the individual affected by these incapability, this research objective was the study of digital image processing for detection and bus route recognition. Therefore, the methodology applied consists on verifying the influence of image processing parameters for bus facade segmentation and route recognizing. After analysis, were run tests using HaarCascade, perspective adjust, blurring, edge enhancement by Canny edge detector, erosion, and *Tesseract*. A combination of 6 different parameters were tested for bus route recognition in videos. The video with the biggest detection rate got 88.46% of frames with identified bus facade, while the biggest rates of recognition and route accuracy were 80% and 80%, respectively, with average of 4 frames per second.

**Key-words:** System. Automatic bus route lecture. Visually impaired. Urban bus. Character recognizing.

## LISTA DE ILUSTRAÇÕES

Figura 1 - Percentual de deficiência física no Brasil em 2010 .....	13
Figura 2 – Exemplo de imagem com limiarização global .....	21
Figura 3 – Filtragem espacial com filtro de média .....	23
Figura 4 – Imagem em escala de cinza (a), bordas detectadas (b) e imagem realçada (c) .....	24
Figura 5 – Exemplo de imagem com caracteres conectados .....	24
Figura 6 – Exemplo da representação integral de imagem com dois, três ou quatro retângulos.....	27
Figura 7 – Reconhecimento facial feito a partir de apenas duas características.....	27
Figura 8 – Representação do neurônio artificial .....	28
Figura 9 – Exemplo de uma palavra com caracteres igualmente espaçados.....	30
Figura 10 – Caracteres escritos à mão em Javanês .....	31
Figura 11 – Detecção de textos em ambiente não controlado .....	32
Figura 12 – Detecção dos números da rota de ônibus.....	33
Figura 13 – Correção de perspectiva para detecção de rota de ônibus .....	33
Figura 14 – Detecção de rota em ponto de ônibus.....	34
Figura 15 – Interface de seleção dos parâmetros testados.....	41
Figura 16 – Fluxograma do processamento comum aos testes .....	43
Figura 17 – Quadro em RGB (a) e em escala de cinza (b), respectivamente .....	44
Figura 18 – Face do ônibus detectada .....	45
Figura 19 – Rota do ônibus binarizada.....	45
Figura 20 – Fluxograma do teste 1.....	47
Figura 21 – Fluxograma do teste 2.....	48
Figura 22 – Fluxograma do teste 3.....	49
Figura 23 – Fluxograma do teste 4.....	50
Figura 24 – Fluxograma do teste 5.....	51
Figura 25 – Fluxograma do teste 6.....	52
Figura 26 – Vídeos com menor e maior taxa de detecção de face, respectivamente .....	54
Figura 27 – Exemplo de vídeos com acerto no reconhecimento da rota.....	56
Figura 28 – Exemplo de vídeos com erro no reconhecimento da rota .....	58



## LISTA DE TABELAS

Tabela 1 – Detalhamento dos vídeos da base de dados .....	36
Tabela 2 – Resultado do ponto ótimo de redimensionamento para detecção da face do ônibus.....	38
Tabela 3 – Resultado do ponto ótimo de redimensionamento para reconhecimento da rota do ônibus.....	39
Tabela 4 – Resumo das aferições mínima, média e máxima de detecção de faces de ônibus.....	54
Tabela 5 – Taxa de detecção de faces de ônibus por vídeo .....	55
Tabela 6 – Resumo dos resultados de detecção e reconhecimento por teste .....	57
Tabela 7 – Resumo das aferições mínima, média e máxima de FPS por teste .....	59
Tabela 7 – Resumo das aferições médias do tempo de processamento do primeiro quadro por teste .....	59

## LISTA DE SIGLAS

CETURB	<i>Companhia de Transportes Urbanos</i>
CMOS	<i>Complementary metal–oxide–semiconductor</i>
F-HOG	<i>Fuzzy Histogram of Oriented Gradients</i>
FPS	<i>Frames por segundo</i>
HSV	<i>Hue Saturation Value</i>
IBGE	<i>Instituto brasileiro de Geografia e Estatística</i>
IEEE	<i>Institute of Electrical and Eletronic Engineers</i>
KNN	<i>K-Nearest Neighbor</i>
LDA	<i>Linear Discriminant Analysis</i>
LDACE	<i>Linear Discriminant Analysis Cauchy Estimator</i>
LSTM	<i>Long-Short Term Memory</i>
MSER	<i>Maximally Stable External Regions</i>
OCR	<i>Optical Character Recognition</i>
OMS	<i>Organização Mundial da Saúde</i>
PCA	<i>Principal Component Analysis</i>
RFID	<i>Radio Frequency Identification</i>
RGB	<i>Red, Green, Blue</i>
SVM	<i>Support Vector Machine</i>

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO .....</b>	<b>13</b>
1.1	OBJETIVOS .....	15
1.1.1	<b>Objetivo Geral .....</b>	<b>15</b>
1.1.2	<b>Objetivos específicos .....</b>	<b>15</b>
1.2	ESTADO DA ARTE.....	15
1.3	ORGANIZAÇÃO DO TRABALHO .....	17
<b>2</b>	<b>REFERENCIAL TEÓRICO .....</b>	<b>18</b>
2.1	PROCESSAMENTO DE IMAGENS.....	18
2.1.1	<b>Representação de uma imagem digital.....</b>	<b>18</b>
2.1.2	<b>Modelos de cores.....</b>	<b>19</b>
2.1.3	<b>Limiarização .....</b>	<b>20</b>
2.1.4	<b>Filtros.....</b>	<b>21</b>
2.1.5	<b>Redimensionamento de imagens .....</b>	<b>25</b>
2.2	DETECÇÃO DE OBJETOS COM O ALGORITMO DE VIOLA E JONES.....	26
2.3	REDES NEURAIS ARTIFICIAIS .....	28
2.3.1	<b>Tesseract .....</b>	<b>29</b>
2.4	TRABALHOS RELACIONADOS.....	31
<b>3</b>	<b>METODOLOGIA .....</b>	<b>35</b>
3.1	BASE DE DADOS.....	35
3.2	TESTE DE REDIMENSIONAMENTO .....	37
3.2.1	<b>Teste de redimensionamento para análise da detecção de faces de ônibus 37</b>	
3.2.2	<b>Teste de redimensionamento para análise da detecção de rota ...38</b>	
3.3	ESCOLHA DOS PARÂMETROS DE CONTROLE.....	39
3.4	INTERFACE DE TESTE DO SISTEMA.....	40
3.5	TESTE DOS PARÂMETROS ESCOLHIDOS .....	41
<b>4</b>	<b>RESULTADOS .....</b>	<b>54</b>
<b>5</b>	<b>CONCLUSÃO .....</b>	<b>60</b>
5.1	TRABALHOS FUTUROS.....	61
	<b>REFERENCIAS.....</b>	<b>62</b>

**APÊNDICE A.....67**

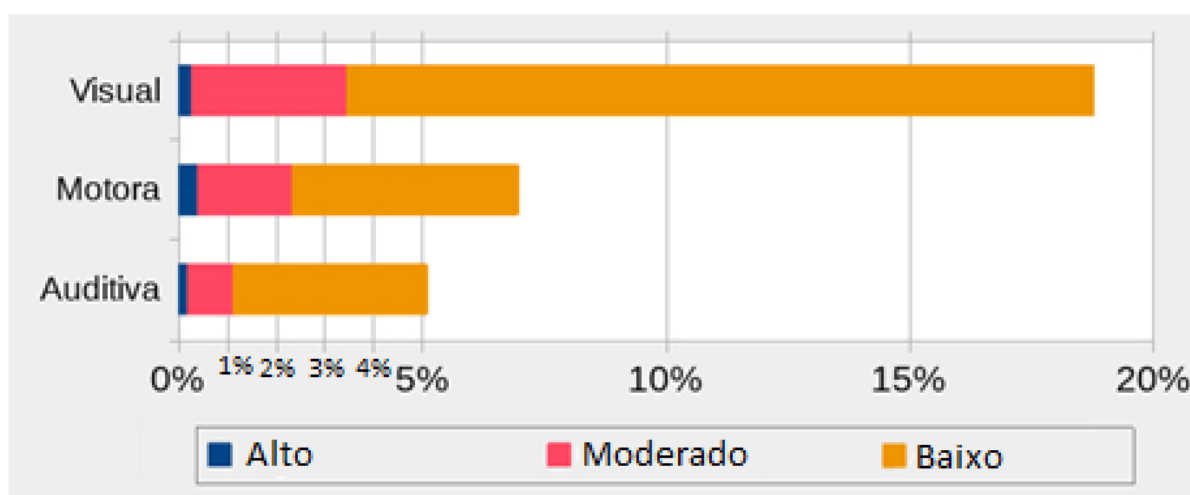
## 1 INTRODUÇÃO

O ramo da computação que estuda a aquisição de imagens, seu reconhecimento e processamento é a visão computacional. Ela é uma subárea da IA (Inteligência Artificial) que, para identificar objetos, pessoas, lugares, cores e caracteres, utiliza câmeras como meio de monitoramento. Essa tecnologia tem o potencial de auxiliar pessoas com deficiência visual, como por exemplo, promovendo acessibilidade aos meios de transporte público (PANCHAL, VARDE e PANSE, 2016).

A visão, segundo Panchal, Varde e Panse (2016), é um dos sentidos mais importantes para os seres humanos e a falta dela cria desafios, tanto na vida do deficiente visual quanto na de seus familiares. A cegueira e a baixa visão transformam várias tarefas básicas do cotidiano da pessoa com deficiência em verdadeiros obstáculos.

O gráfico da Figura 1 divide o número de pessoas com deficiência de acordo com o grau e o tipo da deficiência. No caso da deficiência visual, tanto as pessoas que não enxergam de modo algum quanto as que tem dificuldade em nível moderado, isto é, indivíduos com menos de 10% da visão, são consideradas pessoas com deficiência e somam 3,45% da população brasileira (IBGE, 2010). De acordo com o último censo demográfico do IBGE (Instituto Brasileiro de Geografia e Estatística), a deficiência visual é o tipo de deficiência física mais ocasional no Brasil (IBGE, 2010).

Figura 1 - Percentual de deficiência física no Brasil em 2010



Fonte: Adaptado de IBGE (2010).

Ainda segundo o IBGE, através da Pesquisa de Informações Básicas Municipais (Munic), de 2017, apenas 11,7% dos 1.649 municípios, que dispõem de transporte coletivo intramunicipal, possuem a frota totalmente adaptada para pessoas com deficiência. A falta de alternativas que possibilitem maior acessibilidade de deficientes visuais registrada no Brasil vai de encontro às estatísticas apresentadas pela OMS (Organização Mundial da Saúde) no Relatório Mundial Sobre a Visão de 2021, que alerta para o aumento da população idosa no mundo e à adoção de hábitos cada vez menos saudáveis, que estão diretamente relacionados a doenças oculares e perda da visão (OMS, 2021).

Diante desses problemas, a proposta de novos trabalhos de acessibilidade para deficientes visuais é dar uma alternativa de inclusão através da tecnologia. Dentre as alternativas de inclusão existentes, uma das mais discutidas pela literatura é a utilização de câmeras. O OCR (*Optical Character Recognition* – Reconhecimento Óptico de Caracteres) em ambientes não controlados tem um papel importante devido à demanda crescente de aplicações em tempo real (turismo, cidades inteligentes, carros inteligentes etc) (QIN, SHIVAKUMARA, *et al.*, 2016).

A eficácia de um sistema de reconhecimento de caracteres está diretamente relacionada a parâmetros que podem ser melhorados por meio de técnicas de processamento de imagens (YE, 2015). Com o intuito de oferecer uma maior qualidade de vida por meio da tecnologia, a contribuição deste trabalho é a avaliação de parâmetros do processamento digital de imagens a partir de um sistema de detecção e reconhecimento de rotas de ônibus em vídeos. Desse modo, a implementação de um sistema semelhante em um *smartphone* seria capaz de identificar um ônibus se aproximando e qual a sua rota. Um conversor de texto em voz, por exemplo, pode informar a linha identificada e dar ao usuário mais independência para transitar.

## 1.1 OBJETIVOS

### 1.1.1 Objetivo Geral

O objetivo geral deste trabalho é avaliar parâmetros do processamento digital de imagens para detecção e reconhecimento de rotas de ônibus a partir do desenvolvimento de um sistema.

### 1.1.2 Objetivos específicos

Com intuito de alcançar o objetivo geral, os seguintes objetivos específicos foram propostos:

- Levantar uma base dados de vídeos com ônibus se aproximando do ponto de parada para o processamento.
- Escolher parâmetros pertinentes ao processamento a partir dos vídeos da base de dados para possíveis melhorias na visibilidade da face do ônibus e da rota.
- Desenvolver um sistema aplicando técnicas de processamento de imagens para detecção da face do ônibus e reconhecimento da rota.
- Avaliar o desempenho do sistema no reconhecimento da rota a partir da variação dos parâmetros escolhidos e de testes utilizando os vídeos da base de dados.

## 1.2 ESTADO DA ARTE

Compreender o cenário científico no qual está inserido este trabalho é fundamental para filtrar os métodos eficazes e alinhados com os objetivos gerais e específicos descritos no item anterior. Para isso, é necessário pesquisar, dentre as várias alternativas, o que tem sido aplicado pela comunidade acadêmica nos últimos anos.

Tanto a detecção de ônibus quanto o reconhecimento de caracteres podem ser conduzidos com uma abordagem estatística. Um exemplo desse tipo de abordagem seria o método proposto por Viola e Jones (2001), que consiste em extrair características de bordas e linhas de um grupo de imagens e utiliza-las para detecção de objetos. Luo, Zhao e Ogai (2017) detectaram ônibus em vídeos para controle de tráfego urbano com a metodologia proposta por Viola e Jones.

Para reconhecimento de caracteres com análise estatística, Naik e Ali (2018) empregaram o PCA (*Principal Component Analysis*) a fim de reconhecer algarismos escritos à mão representando as características das imagens com autovetores e autovalores. Guo (2018) abordou em seu estudo o método LDACE (*Linear Discriminant Analysis Cauchy Estimator*), que se difere do LDA (*Linear Discriminant Analysis*) por maximizar a distância entre o conjunto de características de cada caractere.

Por outro lado, o uso de redes neurais para classificação, agrupamento e reconhecimento, tornou-se mais popular recentemente quando comparado aos modelos convencionais de regressão e estatística (ABIODUN, JANTAN, *et al.*, 2018). Sein, Htet et al. (2020) visaram a detecção de veículos (carros sedan, ônibus, caminhões, motos, bicicletas e pedestres) em vídeos utilizando Redes Neurais Convolucionais através do algoritmo conhecido como YOLO (*You Only Look Once – Você Só Olha Uma Vez*).

Já Jaderberg et al. (2014) adotaram, como ferramenta de reconhecimento de palavras, Redes Neurais Convolucionais Profundas em imagens a céu aberto. A estrutura do método proposto não detecta necessariamente o texto no primeiro momento, mas seleciona todos os possíveis candidatos e os lança como entrada da rede. Outro algoritmo de reconhecimento de caracteres que possui redes neurais é o *Tesseract*, que emprega a LSTM (*Long-Short Term Memory – Memória de Curto Longo Prazo*) e foi recentemente testado por Robby, Tandra et al. (2019) para reconhecer caracteres em Javanês.



### 1.3 ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado em cinco capítulos nos quais é discutido o que segue:

- Capítulo 1: é apresentado o contexto em que o trabalho está inserido bem como a motivação e os objetivos.
- Capítulo 2: é composto pela revisão de literatura de artigos pertinentes à detecção de objetos e reconhecimento de caracteres.
- Capítulo 3: aqui são expostos os procedimentos utilizados para realização e validação dos testes estudados.
- Capítulo 4: são apresentados os resultados obtidos.
- Capítulo 5: conclusão e recomendações para trabalhos futuros.

## **2 REFERENCIAL TEÓRICO**

Neste capítulo serão apresentados conceitos importantes para o entendimento não só da motivação deste trabalho, mas também da metodologia adotada. A execução deste trabalho envolve o processamento de imagens, a detecção de objetos e o reconhecimento de caracteres utilizando redes neurais.

### **2.1 PROCESSAMENTO DE IMAGENS**

Na década de 1920, quando foram enviadas as primeiras imagens via cabeamento submarino de Londres a Nova Iorque, iniciou-se o processamento de imagens devido à má qualidade das imagens enviadas. A implementação das atuais ferramentas de processamento, no entanto, só foi possível devido à concepção do computador de John von Neumann, na década de 1940, e aos avanços tecnológicos na área de circuitos eletrônicos. (GONZALEZ e WOODS, 2009).

Com o advento dos transistores na década de 1960 houve uma nova evolução na área do processamento de imagens, e tal evolução deu origem a novas técnicas e algoritmos para manipulação de imagens (GONZALEZ e WOODS, 2009).

A fim de alcançar os objetivos gerais e específicos propostos neste trabalho, serão aplicadas as técnicas de transformação de modelo de cores, limiarização, redimensionamento, suavização, detecção de bordas, detecção de objetos e reconhecimento de caracteres.

#### **2.1.1 Representação de uma imagem digital**

Para permitir o processamento digital de imagens através de algoritmos computacionais, antes, é necessário que se possua as imagens em formato digital.

Uma imagem digital pode ser entendida como um conjunto finito de elementos com localização e valor específicos que compõe uma matriz. Esses elementos, chamados de *pixels*, estão distribuídos no plano espacial da imagem (GONZALEZ e WOODS, 2009). O valor, ou intensidade, de um *pixel* é um número decimal que varia dentro de um intervalo definido pelo número de bits da imagem. Um pixel de uma imagem digital de 8 bits, por exemplo, poderá assumir 256 valores de intensidade diferentes.

No processamento de imagens, as imagens digitais são separadas em: imagens coloridas; escala de cinza; e imagens binárias (GONZALEZ e WOODS, 2009). No caso de uma imagem em escala de cinza de 8 bits, os pixels podem assumir valores de intensidade de 0 a 255 onde o número 0 corresponde a cor preta, o número 255 a cor branca e cada valor dentro do intervalo representa um tom de cinza diferente. Já na imagem binária, os pixels são representados apenas pelas cores preta, nível zero, e branca, nível um. Diferentemente das imagens em escala de cinza e binária, são utilizadas também imagens coloridas que são compostas por três matrizes, como será visto na seção 2.1.2.

### **2.1.2 Modelos de cores**

Os modelos de cores mais utilizados no processamento de imagens são o RGB (*red, green, blue* - vermelho, verde, azul) e o HSI (*hue, saturation, intensity* - matiz, saturação, intensidade). Cada um desses modelos para uma aplicação diferente, mas de maneira geral têm como objetivo viabilizar a especificação de cores através de um padrão (GONZALEZ e WOODS, 2009).

No caso da imagem em RGB, cada uma das três matrizes da imagem é, na verdade, uma imagem em escala de cinza e a combinação da intensidade de cada cor primária compõe a cor do pixel. Existe uma relação de transformação, entre uma imagem colorida em RGB e a mesma imagem em escala de cinza, que poderá ser feita matematicamente utilizando a Equação 1, proposta pela ITU (*International Telecommunication Union* – União Internacional de Telecomunicações),

$$(0,3 \times R) + (0,59 \times G) + (0,11 \times B) = Y \quad (1)$$

onde R, G e B correspondem às matrizes da imagem colorida e Y corresponde à imagem em escala de cinza. Os coeficientes mostrados na Equação 1 são determinados a partir do diagrama de cromaticidade através de padrões de representação de cores em sistemas de televisão analógica (INTERNATIONAL TELECOMMUNICATION UNION, 2011).

O processamento de imagens pode ser feito tanto em imagens coloridas quanto em imagens em escala de cinza. Para imagens coloridas que utilizam o modelo de cores RGB, por exemplo, o processamento deve ser feito nas três matrizes da imagem (GONZALEZ e WOODS, 2009). Portanto, uma vantagem da utilização da imagem em escala de cinza em relação à imagem colorida é a utilização de apenas uma matriz para o processamento.

Em sistemas de reconhecimento de caracteres, a conversão de imagem colorida para imagem em escala de cinza é uma etapa inicial que faz parte do pré-processamento. As demais técnicas de pré-processamento, apresentadas a seguir, são aplicadas para melhoria da eficácia do sistema.

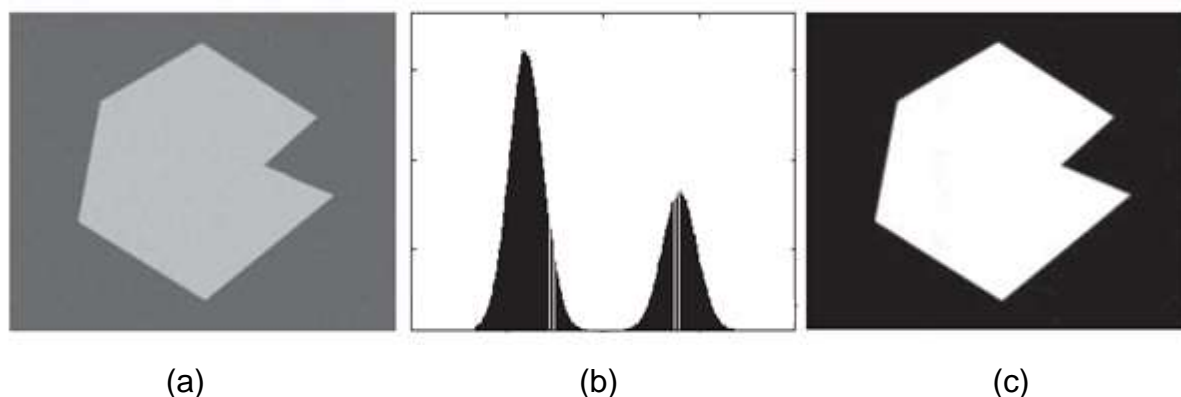
### **2.1.3 Limiarização**

Para Gonzales e Woods (2009), a limiarização pode ser entendida como uma tomada de decisão ao tentar separar grupos de pixels de acordo com a sua intensidade. A binarização é uma técnica de limiarização aplicada no processamento de imagens que transforma uma imagem em tons de cinza em uma imagem binária e auxilia na detecção de objetos. A binarização acontece ao selecionar um limiar “L”, tal que qualquer ponto da imagem com intensidade maior que “L” pertença ao objeto de interesse e, caso contrário, pertence ao fundo da imagem. A tomada da decisão pelo limiar adequado, no entanto, nem sempre é trivial. Nesses casos utiliza-se o histograma da imagem e algoritmos como por exemplo o de Otsu.

O histograma é uma ferramenta que auxilia na escolha do limiar ao organizar graficamente a quantidade de pixels de acordo com o respectivo nível de intensidade. Quando o histograma da imagem possui modas visivelmente bem definidas o limiar é

um valor de intensidade constante escolhido de maneira criteriosa entre os valores modais. Esse processo, ilustrado na Figura 2, chama-se limiarização global.

Figura 2 – Exemplo de imagem com limiarização global



Fonte: Adaptado de Gonzales e Woods (2009).

A limiarização global, exemplificada na Figura 2 (c), separa o objeto de interesse do plano de fundo aplicando o limiar escolhido entre as modas do histograma da Figura 2 (b). (GONZALEZ e WOODS, 2009).

Um exemplo de algoritmo de binarização adaptativa baseada em uma função estatística foi proposto por Otsu (1979) e sua aplicação é observada em trabalhos de reconhecimento de caracteres como o de Chen et al (2011) e Harraj e Raissouni (2015). O método simplifica a escolha do limiar tomando a variância entre as modas do histograma como parâmetro discriminante. Ao maximizar esta variância o método de Otsu permite escolher o limiar mais adequado para separar o plano de fundo de uma imagem com histograma multimodal.

#### 2.1.4 Filtros

Para remover elementos indesejáveis de uma imagem digital é possível lançar mão de ferramentas computacionais, dentre elas os filtros. Para remover elementos de alta frequência, tais como ruídos e pequenos detalhes irrelevantes, utiliza-se os filtros passa-baixa. Como após a conversão de imagem colorida para escala de cinza, a imagem convertida poderá ainda apresentar ruídos, provenientes da câmera que fez

a aquisição das imagens ou de uma fonte de iluminação não uniforme, utiliza-se assim os filtros que irão auxiliar na remoção desses ruídos (PANCHAL, VARDE e PANSE, 2016).

Um dos métodos de filtragem mais utilizados é a filtragem espacial onde uma máscara é convolvida com a imagem digital. Segundo Gonzalez e Woods (2009), a filtragem espacial também é uma ferramenta básica para aplicação de técnicas de realce, suavização de imagem, entre outras aplicações. Uma filtragem espacial consiste basicamente em atribuir ao pixel o valor resultante da operação predefinida utilizando os pixels vizinhos (GONZALEZ e WOODS, 2009).

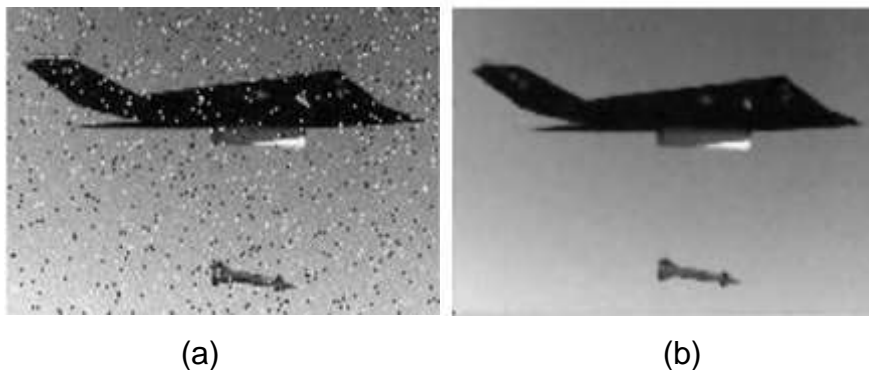
As seções a seguir mostrarão os filtros passa-baixa, passa-alta e morfológico.

#### 2.1.4.1 Filtro passa-baixa

O filtro passa-baixa, conhecido também como filtro de suavização, é utilizado, principalmente, para redução de ruído e conexão de pequenas discontinuidades em linhas. O uso de filtros de suavização para melhoria da eficiência de sistemas de reconhecimento de caracteres pode ser observado no trabalho de Naganjaneyulu, Narasimhadhan e Venkatesh (2014). Dentre os vários tipos de filtros de suavização, o filtro de média é vantajoso devido à facilidade de implementação e o baixo custo computacional. Um exemplo da aplicação desse filtro é visto no trabalho de Panchal, Varde e Panse (2016) que fizeram um sistema de reconhecimento de caracteres voltado para pessoas com deficiência visual.

Na aplicação de um filtro de suavização utilizando a média, cada pixel da imagem original, após a aplicação do filtro de média, é uma média dos pixels contidos na vizinhança da máscara de filtragem. Como efeito visual tem-se uma imagem menos nítida e com redução das transições abruptas de intensidade, isto é, com redução do ruído, conforme mostrado na Figura 3 (GONZALEZ e WOODS, 2009).

Figura 3 – Filtragem espacial com filtro de média



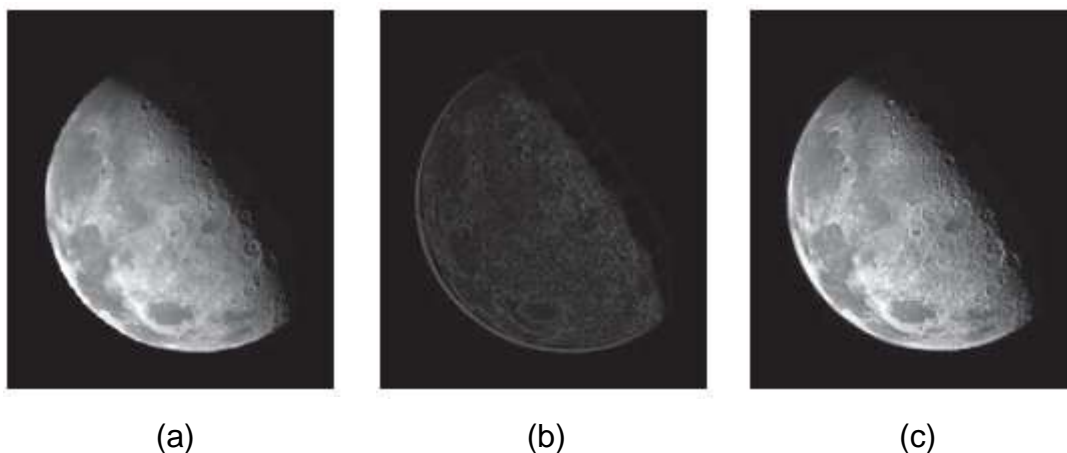
Fonte: Adaptado de Panchal, Varde e Panse (2016).

#### 2.1.4.2 Filtro passa-alta

O filtro passa-alta viabiliza o processo de aguçamento das bordas de uma imagem digital. Em um sistema de reconhecimento de caracteres o realce das bordas, através de um filtro passa-alta, é uma das técnicas que podem ser utilizadas para melhorar a taxa de acertos (YE, 2015). Normalmente o realce entre as bordas dos textos e o plano de fundo de uma imagem digital são mais visíveis, como exemplo cita-se: textos com caracteres pretos sobre fundo branco. Entretanto, quando o texto está contido em objetos que se encontram em movimento, estes caracteres poderão apresentar contornos menos definidos e, sendo assim, a alternativa para a solução deste problema é o aguçamento do texto da imagem utilizando, como filtro passa-alta, o detector de bordas de Canny, como visto no trabalho de Chen et al (2011).

Para efetuar a detecção pelo método de Canny é feita a convolução de uma máscara de perfil gaussiano com a imagem em escala de cinza que permite não só a localização, como também a identificação das direções das bordas (CANNY, 1986). O realce de bordas, exemplificado na Figura 4 (c), é feito somando-se as bordas detectadas, na Figura 4 (b), à imagem original vista na Figura 4 (a) (GONZALEZ e WOODS, 2009).

Figura 4 – Imagem em escala de cinza (a), bordas detectadas (b) e imagem realçada (c)



Fonte: Adaptado de Gonzales e Woods (2009).

#### 2.1.4.3 Filtro morfológico

O processo de binarização, explicado na seção 2.1.3, consiste em transformar uma imagem em escala de cinza em uma imagem binária útil para o reconhecimento de caracteres. Por outro lado, essa transformação pode gerar uma imagem degradada. Em imagens degradadas, normalmente, os pixels de um caractere estão conectados aos pixels do caractere adjacente, como mostra a Figura 5 (YE, 2015).

Figura 5 – Exemplo de imagem com caracteres conectados



Fonte: Adaptado de Nomura et al (2005).

A Figura 5 mostra uma imagem binarizada de uma placa de carro com o dígito 8 e o dígito 6 conectados. A separação do texto em caracteres é um dos desafios para algoritmos de reconhecimento de caracteres. Uma ferramenta que possibilita a desconexão de caracteres conectados em imagens degradadas é o filtro morfológico, como visto no trabalho de Nomura et al (2005).



O filtro morfológico, analogamente ao processo de convolução, depende de uma máscara que neste caso é chamada de elemento estruturante. O elemento estruturante pode ser entendido como um conjunto que define a vizinhança utilizada para processar os pixels de uma imagem binária. Na operação morfológica de erosão, um elemento estruturante previamente escolhido percorrerá todos os pixels e a imagem resultante terá apenas os pixels em que o elemento estiver totalmente contido em cada conjunto de pixels da imagem original (GONZALEZ e WOODS, 2009).

No processamento de imagem morfológico, o design do elemento estruturante, sua forma e tamanho, são importantes para o sucesso da operação morfológica. Quando combinados com os elementos estruturantes adequados, os operadores morfológicos de erosão são capazes de gerar imagens com contornos suavizados, eliminar ruídos e detalhes irrelevantes sem que a imagem perca informações úteis (GONZALEZ e WOODS, 2009).

Ao contrário da erosão que elimina partes da imagem, existe também a dilatação que pode ser entendida como uma operação morfológica que aumenta os elementos de uma imagem binária. O formato e o tamanho a ser aumentado depende também do elemento estruturante utilizado. No processamento morfológico de imagens é possível também combinar sucessivas operações de erosão e dilatação dando origem a operações de abertura e fechamento, por exemplo. As operações de abertura e fechamento podem ser empregadas para preencher pequenos vazios entre as bordas dos objetos de uma imagem além de também minimizar ruídos (GONZALEZ e WOODS, 2009).

### **2.1.5 Redimensionamento de imagens**

Outra técnica amplamente utilizada no processamento de imagens é o redimensionamento de imagens. A redução de imagens tem, entre outras, as funções de: aumentar a velocidade do processamento de vídeos em tempo real e de diminuir ruídos de alta frequência de maneira idêntica aos filtros de média e de erosão explicados na seção 2.1.4 (GONZALEZ e WOODS, 2009).

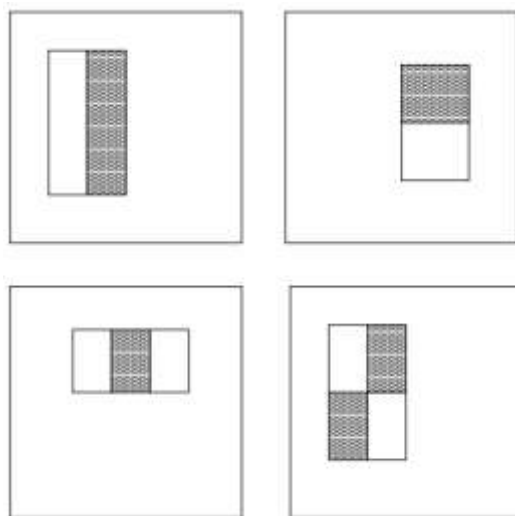
A redução de imagens pode ser entendida como um método de subamostragem. No entanto, ao alterar a resolução de uma imagem digital através da subamostragem, é importante minimizar as perdas de conteúdo da imagem. Nesse sentido, aplicam-se métodos de interpolação, tais quais a interpolação por vizinho próximo e a interpolação linear, que são uma ferramenta de auxílio na reconstrução da imagem (GONZALEZ e WOODS, 2009).

## 2.2 DETECÇÃO DE OBJETOS COM O ALGORITMO DE VIOLA E JONES

Dentro da visão computacional, detectar objetos é importante em várias atividades. Um dos algoritmos de detecção mais utilizados é o de Viola e Jones (2001) que trouxe uma nova perspectiva entre os acadêmicos da época ao representar as imagens baseando-se em características simples e facilmente classificáveis, tais como a diferença de intensidade dos pixels de uma região. A decisão de usar características ao invés do valor dos pixels se deve, principalmente, ao fato de que características podem ser melhor definidas dentro de um número finito de exemplares além de serem processadas mais rápido (VIOLA e JONES, 2001).

Ao analisar as características de uma imagem, a interpretação, isto é, o entendimento do contexto da imagem e dos elementos presentes nela, independe da resolução (MALLAT, 1989). Com isso, Viola e Jones (2001), estabeleceram a representação integral de imagem, calculando a subtração do somatório das intensidades dos pixels dentro de regiões retangulares. Os retângulos, exemplificados na Figura 6, têm o mesmo formato e tamanho e são horizontalmente ou verticalmente adjacentes.

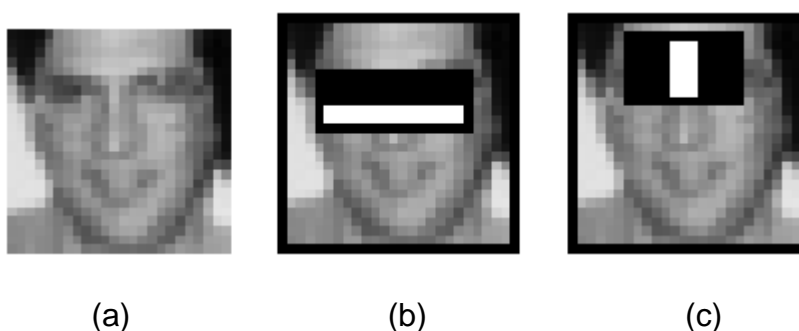
Figura 6 – Exemplo da representação integral de imagem com dois, três ou quatro retângulos



Fonte: Adaptado de Viola e Jones (2001).

A representação de características por retângulos, mostrada na Figura 6, permite que o algoritmo de Viola e Jones seja sensível à presença de bordas em uma imagem. Com isso, há um ganho de desempenho no aprendizado de máquina pois, como visto na Figura 7, o treinamento do algoritmo é feito com imagens em baixa resolução. (VIOLA e JONES, 2001).

Figura 7 – Reconhecimento facial feito a partir de apenas duas características



Fonte: Adaptado de Viola e Jones (2001).

No caso da detecção de faces, por exemplo, os autores viram que os retângulos de características são mais significativos na região central dos olhos e entre os olhos e a parte superior das bochechas mostrados na Figura 7 (b) e Figura 7 (c), respectivamente. Dessa forma, imagens digitais que apresentem o mesmo padrão de regiões claras e escuras serão detectadas como faces humanas. Este mesmo princípio de representação de características através de retângulos proposto por Viola

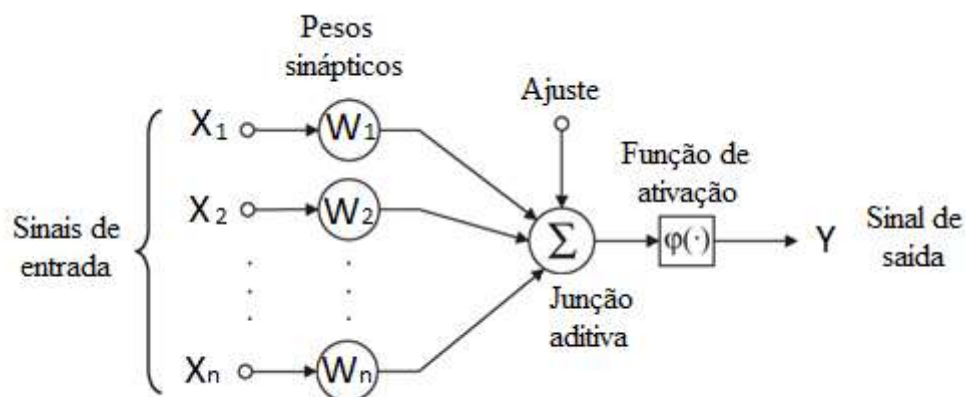
e Jones pode ser empregado para detecção de faces de ônibus, como visto no trabalho de Guida, Comanducci e Colombo (2011).

## 2.3 REDES NEURAIS ARTIFICIAIS

As redes neurais artificiais são um tipo de ferramenta matemática utilizada para tomada de decisão. Essas redes, que simulam características das redes neurais biológicas do cérebro humano, possuem o potencial de processar imagens, reconhecer caracteres, entre outras aplicações (HAYKIN, 2001; YE, 2015).

O cérebro humano é formado por células estruturais chamadas de neurônios que ao trabalhar em conjunto são capazes de processar informações complexas em milésimos de segundos. Esse sistema de processamento, não-linear e paralelo, é imitado pelo neurônio artificial. De maneira análoga ao neurônio biológico, o neurônio artificial, mostrado na Figura 8, também é entendido como unidade básica e é composto por entradas cujos pesos são chamados de pesos sinápticos e o somatório da combinação linear de cada entrada com seu respectivo peso passa por uma função de ativação que limita o valor do sinal de saída (HAYKIN, 2001; MCCULLOCH e PITTS, 1943).

Figura 8 – Representação do neurônio artificial



Fonte: Adaptado de Simon Haykin (2001).

O conjunto das unidades básicas, os neurônios artificiais, forma a rede neural da qual pode-se afirmar que:

Uma rede neural é um processador (...) constituído de unidades de processamento simples que tem a propensão natural para armazenar conhecimento experimental e torna-lo disponível para o uso (HAYKIN, 2001, p. 28).

Assim como acontece no cérebro, uma rede de neurônios artificiais também adquire conhecimento a partir do ambiente. Esse comportamento é ideal para previsão não linear de resultados e reconhecimento de caracteres, por exemplo, como visto no trabalho de Yann LeCun (1990). A rede neural de LeCun, nomeada de LeNet, foi o primeiro estudo a aplicar redes neurais para processamento de dados bidimensionais através do reconhecimento de dígitos escritos à mão em imagens.

Desde então, outras redes neurais também foram aplicadas para executar tarefas similares. Exemplo disso é trabalho de Zou, Zhang *et al.* (2020), que utilizou a rede LSTM (*Long-Short Term Memory* – Memória de Curto Prazo Longa) para reconhecimento de caracteres em placas de carro. Uma característica atrativa da rede utilizada é a memória associativa que armazena e utiliza as informações passadas em cálculos futuros (HAYKIN, 2001; BAHl e ZATNI, 2019). Outro exemplo é o trabalho de Robby *et al.* (2019) utilizou o *Tesseract*, um OCR que emprega a LSTM. O funcionamento deste OCR, que também será utilizado neste trabalho, encontra-se na seção 2.3.1.

### **2.3.1 Tesseract**

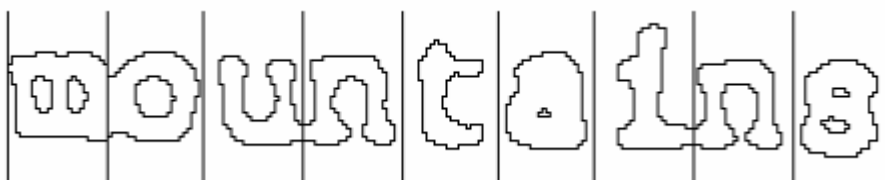
O *Tesseract* é um OCR de código aberto desenvolvido entre 1984 e 1994 por Ray Smith. A partir de então esta ferramenta, utilizada para o reconhecimento de caracteres brancos com fundo preto, vem sendo melhorada e, em 2016, teve a rede LSTM acrescentada a sua estrutura (SMITH, ABDULKADER, *et al.*, 2020).

O *Tesseract* segue uma estrutura dividida em etapas executadas paralelamente. Na primeira etapa ocorre uma análise da conectividade dos pixels onde cada elemento delimitado por contornos fechados é chamado de *blob* (“bolha”). Em seguida, as

bolhas passam por um processo de filtragem a partir da altura média. Dessa forma, assumindo que os caracteres do texto possuem uma dimensão similar, o filtro utiliza a altura média dos caracteres para ignorar as bolhas abaixo da média uma vez que estes elementos correspondem majoritariamente aos sinais de ponto final e de ruídos (SMITH, 2007).

Após filtradas, as bolhas são agrupadas em linhas e, então, separadas em palavras e em caracteres. Quando o espaçamento e o tamanho dos caracteres são uniformes, a separação é feita de maneira trivial utilizando um espaçamento horizontal fixo como referência, conforme exposto na Figura 9 (SMITH, 2007).

Figura 9 – Exemplo de uma palavra com caracteres igualmente espaçados



Fonte: Smith (2007).

A separação de palavras em caracteres torna-se mais difícil quando não existe uniformidade de espaçamento. Nesses casos, é calculada a distância vertical entre a linha de base, abaixo dos caracteres, e a linha que demarca a altura média com a finalidade de viabilizar a segmentação das linhas de texto em palavras (SMITH, 2007).

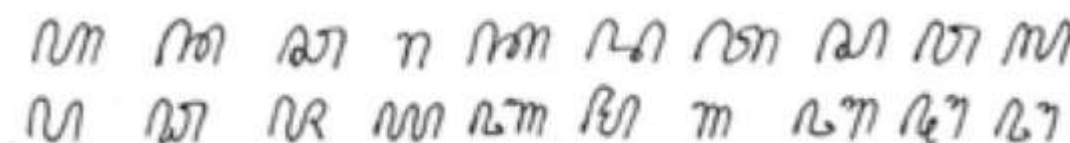
Então, são feitas duas tentativas de reconhecimento das palavras segmentadas. Na primeira, cada palavra é combinada com a base linguística do *Tesseract* buscando: palavras mais frequentes; as principais palavras no dicionário; números; palavras em caixa alta; palavras em caixa baixa; e as mais escolhidas pelo algoritmo de classificação. As palavras reconhecidas são passadas novamente ao classificador para que, na segunda tentativa, o reconhecimento seja feito com maior precisão já que novas características foram aprendidas durante a primeira tentativa. Finalmente, caso resultado do reconhecimento das palavras ainda não seja satisfatório, será feita uma última etapa de reconhecimento de cada caractere (SMITH, 2007).

## 2.4 TRABALHOS RELACIONADOS

Parte dos trabalhos voltados para deficientes visuais concentram-se no reconhecimento de caracteres em ambientes não controlados, como o de Panchal, Varde e Panse (2016) que utilizaram LDA e SVM no reconhecimento e CRF (*Conditional Random Field* – Campo Aleatório Condicional) para agrupar os caracteres reconhecidos em frases. Ao final do processamento, as frases agrupadas foram convertidas em áudio através do software de simulação LabView.

O estudo de Robby et. al. (2019) consistiu em utilizar o *Tesseract* para reconhecimento de caracteres em Javanês, como os mostrados na Figura 10, através de um aplicativo para telefone móvel.

Figura 10 – Caracteres escritos à mão em Javanês



Fonte: Adaptado de Robby et al. (2019).

Neste trabalho não foi necessária uma etapa de detecção pois a base de dados utilizada para reconhecimento possuía imagens contendo apenas os textos manuscritos em fundo branco. Já o trabalho de Minetto et. al. (2011), que também aplicou o *Tesseract*, demandou um trabalho de detecção uma vez que as imagens utilizadas no reconhecimento eram majoritariamente placas de identificação em cenas urbanas, como visto na Figura 11, que são mais difíceis de reconhecer.

Figura 11 – Detecção de textos em ambiente não controlado



Fonte: Adaptado de Minetto et al. (2011).

De fato, a detecção dos textos em ambiente não controlado é um gargalo para o *Tesseract* devido à presença de elementos indesejados que diminuem a taxa de acertos no reconhecimento de caracteres. A solução encontrada por Minetto et. al. (2011) foi utilizar SVM para separar possíveis textos e descritores F-HOG (*Fuzzy Histogram of Oriented Gradients* – Histograma de Gradientes Orientados Fuzzy) para eliminar falsos positivos atingindo 73% de acertos.

Outros trabalhos também voltados para auxiliar cegos a serem mais independentes, estão relacionados à detecção de rotas de ônibus. Uma solução conhecida para este fim é o uso de sensores, como o RFID (*Radio Frequency Identification* – Identificação por Rádio Frquência), e sistemas baseados em fotos. Cheng, Tsai e Yeh (2014) exploraram um método para detecção de números de rota de ônibus baseado em binarização adaptativa, subtraindo o quadro atual do quadro anterior, e na conectividade dos componentes da imagem. A Figura 12 exemplifica a detecção da rota feita pelos autores.



Figura 12 – Detecção dos números da rota de ônibus



Fonte: Adaptado de Cheng, Tsai e Yeh (2014).

Nesse estudo os parâmetros de filtragem e segmentação dos caracteres foram aferidos empiricamente. A taxa de detecção de face de ônibus pelo método proposto foi de 73,79% e de detecção do número da rota foi de 100%. Os resultados, segundo os autores, apontam que o sistema pode ser útil para uma pessoa esperando pelo ônibus, dada qualidade dos vídeos e o tempo de execução do algoritmo.

Wongta, Kobchaisawat e Chalidabhongse (2016) propuseram um sistema automático de reconhecimento de caracteres de rota de ônibus utilizando redes neurais convolucionais. A primeira etapa consistiu numa correção de perspectiva da área contendo o número da linha do ônibus, mostrada na Figura 13.

Figura 13 – Correção de perspectiva para detecção de rota de ônibus



Fonte: Adaptado de Wongta, Kobchaisawat e Chalidabhongse (2016).

Na área recortada é feita a detecção dos caracteres e a segmentação dos dígitos com auxílio do algoritmo MSER (*Maximally Stable External Regions*) de Neumann e Matas

(2012). Na fase final do método proposto, 78% do texto foi detectado e os 73,47% dos caracteres reconhecidos corretamente são convertidos em áudio.

Pan, Yi e Tian (2013) apresentaram um sistema que detecta e reconhece rotas de ônibus a partir de fotos tiradas em pontos de ônibus, mostrado na Figura 14.

Figura 14 – Detecção de rota em ponto de ônibus



Fonte: Adaptado de Pan, Yi e Tian (2013).

O sistema, utiliza o detector de bordas *Canny* para eliminar o fundo contendo céu, estrada, pedestres e veículos. Em seguida, o histograma de gradiente orientado (*Histogram of oriented gradient* – HOG) é extraído do recorte que possivelmente possui a face de um ônibus e comparado com histogramas de faces de ônibus através de uma SVM. Para detecção de texto novamente é empregado o detector de bordas *Canny* onde os possíveis candidatos são separados em regiões e filtrados. Por fim, as regiões são reconhecidas com auxílio de um OCR e convertidas em voz. O estudo teve um desempenho de 80,93% na detecção de faces de ônibus e 98,11% na detecção de textos na face do ônibus.

O Quadro 1 reúne os autores e os parâmetros escolhidos para detecção e reconhecimento de rotas de ônibus.

Quadro 1 –Técnicas utilizadas pelos autores pesquisados para detecção e reconhecimento de rota

Autor (es)	Técnicas
Cheng, Tsai e Yeh	Binarização adaptativa no espaço de cores YCbCr
Wongta, Kobchaisawat e Chalidabhongse	Correção de perspectiva, detecção de texto com MSER e reconhecimento com CNN
Pan, Yi e Tian	Detector de bordas <i>Canny</i> e extração de características HOG

Fonte: Autor (2022).

### 3 METODOLOGIA

Neste capítulo serão apresentados a metodologia e o desenvolvimento deste trabalho. As técnicas e informações relacionadas ao processamento de imagens foram explicadas na seção 2.

A execução deste trabalho envolveu o teste de parâmetros que influenciam no reconhecimento de caracteres e para viabilização dos testes foi criado um algoritmo que processa vídeos de ônibus se aproximando de um ponto de parada. Na Quadro 2 estão relacionadas as etapas adotadas e uma breve descrição do que será feito em cada etapa.

Quadro 2 – Etapas adotadas na metodologia deste trabalho

<b>Etapas</b>	<b>Descrição</b>
1	Gravar vídeos para base de dados
2	Testar redimensionamento para detecção de face
3	Testar redimensionamento para detecção de rota
4	Definir parâmetros de controle
5	Implementar interface automática de testes
6	Testar os parâmetros escolhidos

Fonte: Autor (2022).

#### 3.1 BASE DE DADOS

A criação de uma base de dados é um elemento importante não só neste trabalho, como também em outros que envolvam visão computacional. Ela tem como objetivo validar as hipóteses referentes ao problema proposto e será utilizada para aferir as taxas de detecção e reconhecimento.

Para compor a base de dados, foram gravados 22 vídeos de ônibus da CETURB (Companhia de Transportes Urbanos) do estado do Espírito Santo. Os vídeos, detalhados na Tabela 1, foram capturados em pontos de parada no município de

Cariacica utilizando um smartphone cuja câmera grava vídeos no formato mp4 com resolução de 1920x1080 pixels.

Tabela 1 – Detalhamento dos vídeos da base de dados

<b>Vídeo</b>	<b>Duração (s)</b>	<b>Resolução</b>	<b>Número de quadros</b>
1	2	1920x1080	81
2	2	1920x1080	85
3	4	1920x1080	117
4	4	1920x1080	98
5	8	1920x1080	247
6	7	1920x1080	207
7	5	1920x1080	160
8	5	1920x1080	161
9	5	1920x1080	166
10	2	1920x1080	74
11	1	1920x1080	50
12	4	1920x1080	142
13	6	1920x1080	185
14	4	1920x1080	124
15	3	1920x1080	119
16	1	1920x1080	52
17	5	1920x1080	161
18	2	1920x1080	76
19	3	1920x1080	91
20	3	1920x1080	93
21	1	1920x1080	51
22	2	1920x1080	77

Fonte: Autor (2022).

Visando uma melhor condição de iluminação ambiente, para realização deste trabalho os vídeos utilizados foram gravados durante o dia, entre os meses de Janeiro e Fevereiro de 2022. Cada vídeo mostra o ônibus se aproximando do ponto de parada e possui duração de 1 a 8 segundos.

## 3.2 TESTE DE REDIMENSIONAMENTO

O tamanho do quadro exerce influência na detecção de faces de ônibus e na detecção de rotas além de diminuir a quantidade de informação a ser processada. Portanto é necessário analisar o percentual de redimensionamento que fornece os melhores resultados de detecção de faces de ônibus e detecção de rotas antes de fazer efetivamente os testes dos parâmetros. Para isso a base de dados será separada em um grupo de 17 vídeos para teste e 5 vídeos, selecionados aleatoriamente, para validação dos parâmetros.

### 3.2.1 Teste de redimensionamento para análise da detecção de faces de ônibus

A detecção da face do ônibus no vídeo será feita com o algoritmo classificador *HaarCascade*. Este classificador, que está disponível na biblioteca de código aberto *OpenCV* para linguagem Python, é baseado no trabalho de Viola e Jones (2001). Dessa forma, diante de uma imagem em escala de cinza o algoritmo compara as características do objeto de interesse com as características pré-treinadas através um arquivo de texto.

Para análise do desempenho do método proposto e tendo em vista a otimização do custo computacional, será realizado um teste com 17 vídeos da base de dados diminuindo a resolução do quadro para detecção com o *HaarCascade*. No teste o quadro será redimensionado gradativamente com a função *resize* do *OpenCV* e a taxa de detecção de faces de ônibus aferida de acordo com a Equação 2:

$$Taxa\ de\ detecção\ de\ faces = \frac{Quadros\ com\ onibus\ detectado}{Total\ de\ quadros} \times 100 \quad (2)$$

onde a taxa de detecção é expressa em percentual.

Também será medida a taxa de FPS (*Frames Per Second* – Quadros Por Segundo) utilizando a Equação 3:

$$FPS = \frac{Total\ de\ quadros}{Tempo\ de\ processamento\ do\ vídeo} \quad (3)$$

onde o tempo de processamento do vídeo é dado em segundos.

O resultado do teste de validação do percentual de redimensionamento pode ser observado na Tabela 2.

Tabela 2 – Resultado do ponto ótimo de redimensionamento para detecção da face do ônibus

Redimensionamento (%)	FPS	Detecção (%)
10%	37,4	3,4
20%	22,8	19,8
30%	14,4	37,4
40%	9,8	43,5
50%	6,5	47,3
60%	4,7	46,5
70%	3,3	47,9
80%	2,3	46,6
90%	1,8	45,9
100%	1,6	42,5

Fonte: Autor (2022).

Apesar da taxa de detecção ser maior com redimensionamento em 70%, a taxa de quadros por segundo é mais baixa se comparado à percentuais menores. Portanto o valor coerente de redimensionamento com maior taxa de detecção e FPS é 50%.

### 3.2.2 Teste de redimensionamento para análise da detecção de rota

Na etapa de reconhecimento de caracteres com *Tesseract* também foi aplicado um teste de validação para encontrar o ponto ótimo de resolução utilizando a biblioteca própria do OCR para *Python*. O teste consistiu em monitorar o número de detecções de rotas também em 17 vídeos reduzindo a resolução da face do ônibus em 10%. O resultado do teste, visto na Tabela 3, revela que a maior taxa de detecções de rota acontece com percentual de redimensionamento em 90%.

Tabela 3 – Resultado do ponto ótimo de redimensionamento para reconhecimento da rota do ônibus

Redimensionamento (%)	Média de FPS	Deteção de rota (%)
10%	4,6	0
20%	4,1	0
30%	2,3	17,6
40%	2,5	35,3
50%	2,3	52,9
60%	2,3	64,7
70%	2,3	70,6
80%	2,4	70,6
90%	2,5	82,4
100%	2,5	76,5

Fonte: Autor (2022).

### 3.3 ESCOLHA DOS PARÂMETROS DE CONTROLE

Na seção 2 foram apresentadas técnicas de processamento de imagens para manipulação de parâmetros pertinentes ao reconhecimento de caracteres. Tendo em vista as técnicas abordadas e outros trabalhos de reconhecimento de caracteres, os parâmetros escolhidos para reconhecimento da rota de ônibus neste trabalho foram: redimensionamento dos quadros, ajuste de perspectiva, realce de contraste, filtro de média e filtro morfológico.

A distância do ônibus em relação ao observador influencia na resolução das imagens capturadas pela câmera registrando ora caracteres maiores, ora caracteres menores. Para utilizar o *HaarCascade* os vídeos da base de dados foram convertidos em escala de cinza e redimensionados para 50% da resolução original. Bahi e Zatni (2019) observaram reduzir o tamanho do quadro mantendo a proporção entre a largura e o comprimento da imagem não altera significativamente o desempenho no reconhecimento de caracteres ao passo que diminui a quantidade de informação a ser processada.

Como o ônibus está em movimento nos vídeos, a perspectiva do observador pode contribuir para um resultado de reconhecimento distante do original, conforme

observado no estudo de Zhu, Sun e Wang (2020). Por conta disso, esse parâmetro também foi escolhido.

Imagens com alto índice de luminosidade possuem pouco contraste e, com isso, a segmentação de objetos torna-se mais complicada. O realce pode ser feito utilizando um filtro passa altas que identifica as transições abruptas de intensidade, como bordas e ruídos (GONZALEZ e WOODS, 2009). Os vídeos da base de dados possuem essa característica pelo fato de terem sido gravados durante o dia. Nesse sentido, foi determinado o uso do algoritmo detector de bordas *Canny*.

Por fim, a escolha do último parâmetro aconteceu devido ao ruído inerente à aquisição das imagens. A presença dele interfere na segmentação e consequentemente altera a forma dos caracteres. O emprego da operação morfológica de erosão e do filtro de média suaviza os contornos e elimina o ruído de alta frequência (GONZALEZ e WOODS, 2009).

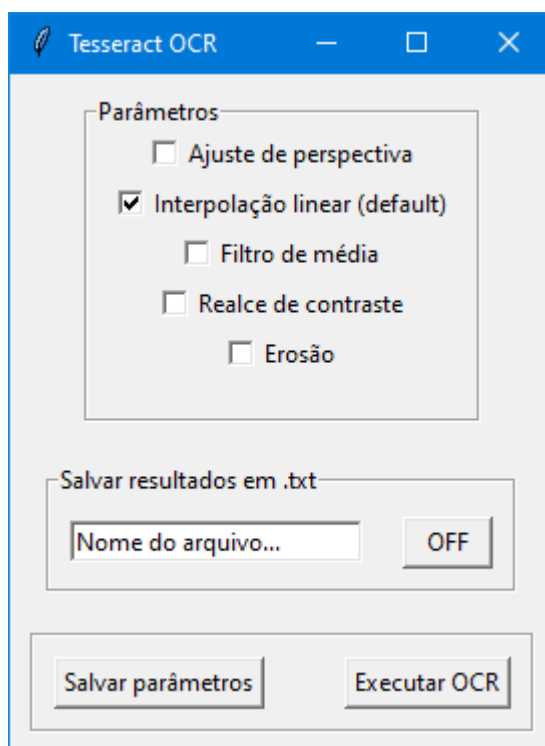
### 3.4 INTERFACE DE TESTE DO SISTEMA

O sistema proposto neste trabalho foi desenvolvido utilizando Python versão 3.7.0 e algumas bibliotecas disponíveis gratuitamente na internet. Dentre as bibliotecas utilizadas, a *pytesseract*, baseada no algoritmo *Tesseract*, possibilita o reconhecimento de caracteres e palavras em 123 idiomas diferentes. Outra biblioteca empregada foi *OpenCV* dedicada ao processamento de imagens. É válido mencionar que estas bibliotecas dispensaram qualquer treinamento para a base de dados deste trabalho.

Com intuito de conduzir os testes de maneira automática, foi criada uma interface de usuário. A interface, que implementa a biblioteca em Python *tkinter*, contém 3 divisões, como observado na Figura 15.



Figura 15 – Interface de seleção dos parâmetros testados



Fonte: Autor (2022).

Na interface mostrada na Figura 15, cada *check-box* na primeira divisão corresponde a um parâmetro que poderá ser selecionado para teste. Na divisão do meio existe um campo onde deverá ser preenchido o nome do arquivo em formato de texto para salvar o resultado do teste e um botão para habilitar ou desabilitar esta função. Na terceira divisão encontram-se os botões para salvar os parâmetros e iniciar o teste.

### 3.5 TESTE DOS PARÂMETROS ESCOLHIDOS

Ao todo foram realizados 6 testes, utilizando como *hardware* uma *CPU Intel Core i3-5005U* com uma placa de vídeo *Intel HD Graphics 5500*, para escolher o melhor conjunto de características e para validar os parâmetros escolhidos. Os testes, mostrados na Quadro 3, foram feitos primeiro com apenas um parâmetro e posteriormente com uma combinação dos parâmetros com melhores resultados de reconhecimento da rota.

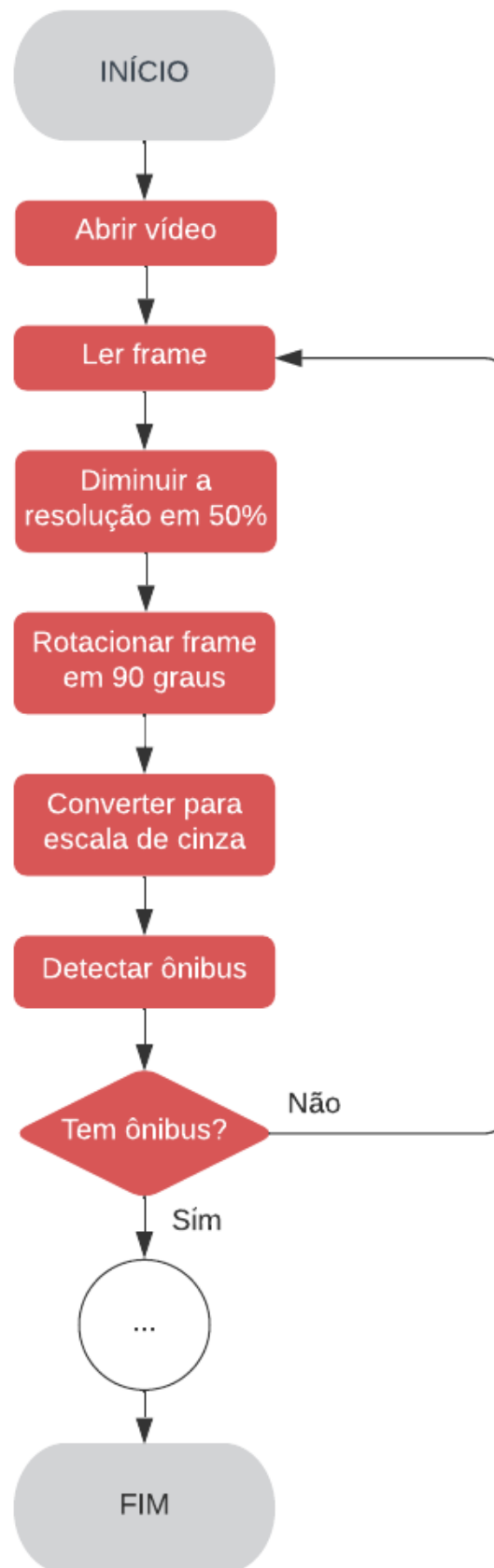
Quadro 3 – Resumo das combinações de parâmetros testadas

<b>Teste</b>	<b>Parâmetros avaliados</b>
Teste 1	Nenhum
Teste 2	Filtro de média
Teste 3	Ajuste de perspectiva
Teste 4	Erosão
Teste 5	Filtro de média e realce de bordas
Teste 6	Filtro de média, realce de bordas e ajuste de perspectiva

Fonte: Autor (2022).

Mesmo explorando parâmetros diferentes, o processamento inicial é o mesmo para todos os testes. A Figura 16 mostra o fluxograma desta parte do algoritmo.

Figura 16 – Fluxograma do processamento comum aos testes



A Figura 16 mostra que cada quadro é alterado três vezes antes da etapa de detecção com *HaarCascade*. Primeiramente, após abrir o vídeo, o quadro é reduzido pela metade, em conformidade com o teste de redimensionamento mencionado na seção 3.2, conservando a proporção de largura e comprimento. A redução diminui o custo computacional da detecção da face do ônibus. Depois o quadro é rotacionado em noventa graus para adequar o posicionamento. Em seguida, o quadro, inicialmente em RGB, é convertido para escala de cinza, como visto na Figura 17.

Figura 17 – Quadro em RGB (a) e em escala de cinza (b), respectivamente



(a)



(b)

Fonte: Autor (2022)

A partir do quadro em escala de cinza é feita a detecção da face, conforme observado na Figura 18. É importante lembrar que podem ser detectadas mais de uma face em um mesmo quadro. Nesse ponto do algoritmo, é feito um teste para verificar a

presença de mais de um ônibus no quadro e, caso existam, apenas a maior entre as duas faces mais próximas é recortada. O recorte é feito utilizando o quadro em tamanho original que, em seguida, será reduzido para 90% conforme teste de redimensionamento.

Figura 18 – Face do ônibus detectada



Fonte: Autor (2022)

Os parâmetros escolhidos para cada teste são então aplicados na face detectada de maneira sequencial com a finalidade de melhorar o resultado final da segmentação e, consequentemente, a taxa de detecção e reconhecimento da rota.

Em seguida, a parte superior da face do ônibus, exemplificada na Figura 19, é recortada, uma vez que o número da rota se encontra nessa região. Finalmente, o recorte da rota é binarizado através o método de Otsu variando o limiar de binarização, com passo de 10%, entre 100% e 140% do limiar de Otsu.

Figura 19 – Rota do ônibus binarizada



Fonte: Autor (2022)

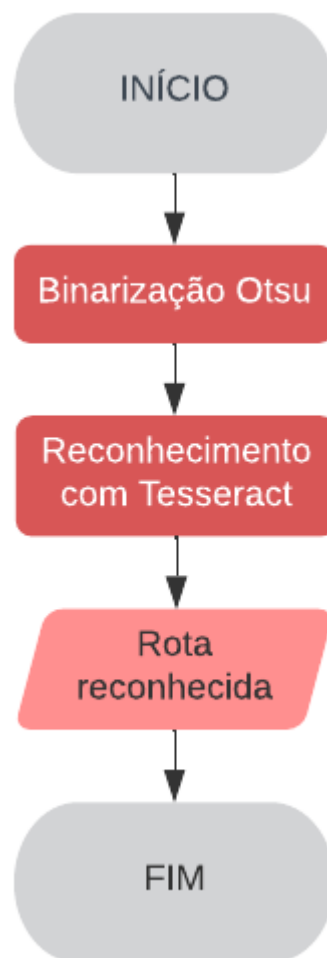
A partir da imagem binária é que acontece o reconhecimento com o *Tesseract*. Para avaliar o desempenho dos parâmetros escolhidos a taxa de detecção de rota será calculada, em cada teste, segundo a equação a seguir.

$$\text{Taxa de detecção de rota} = \frac{\text{Vídeos com rota detectada}}{\text{Total de vídeos}} \times 100 \quad (4)$$

Vale salientar que o resultado do reconhecimento de caracteres nem sempre fornece apenas números da rota. Por conta disso foi feito um filtro que checa se os caracteres reconhecidos são números e se os números formam uma rota com três dígitos.

No primeiro teste o processamento e a segmentação foram conduzidos sem teste de parâmetros adicionais. Este teste foi essencial para o desenvolvimento do algoritmo uma vez que a partir dele foram feitas análises de histograma tanto nos canais da imagem colorida quanto da imagem em escala de cinza. Com isso a hipótese de filtragem no espaço de cores foi descartada além da observação da necessidade de melhoria na taxa de quadros por segundo que foi feita utilizando metade dos quadros com ônibus detectado para o reconhecimento da rota. No fluxograma da Figura 20 é possível ver as etapas do teste.

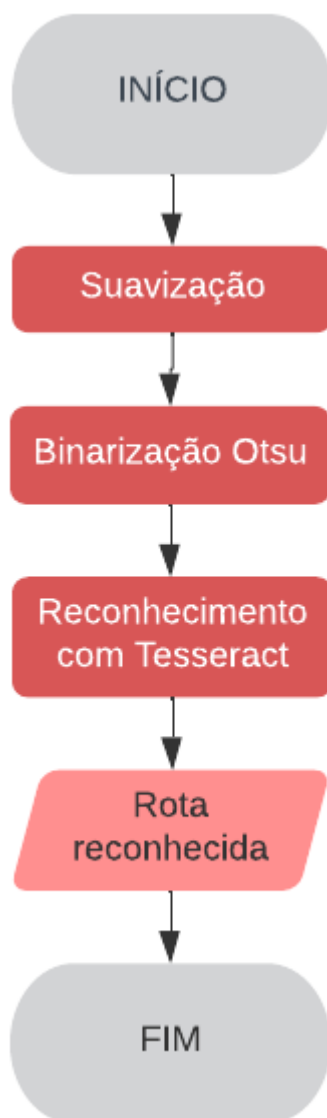
Figura 20 – Fluxograma do teste 1



Fonte: Autor (2022)

No segundo teste, cujo fluxograma encontra-se na Figura 21, foi acrescentada a etapa de suavização dos contornos com filtro espacial de média utilizando uma matriz de tamanho 20 vezes menor que a altura do recorte da parte superior da face do ônibus.

Figura 21 – Fluxograma do teste 2

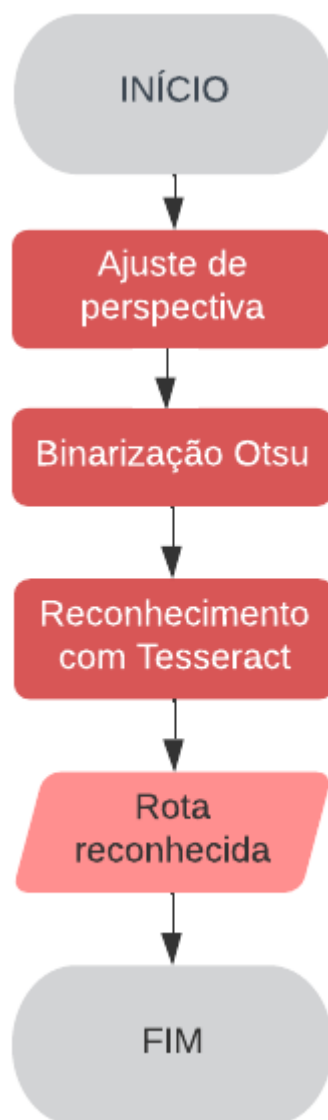


Fonte: Autor (2022).

Na sequência, ajuste de perspectiva foi avaliado e o fluxograma do terceiro teste encontra-se Figura 22.



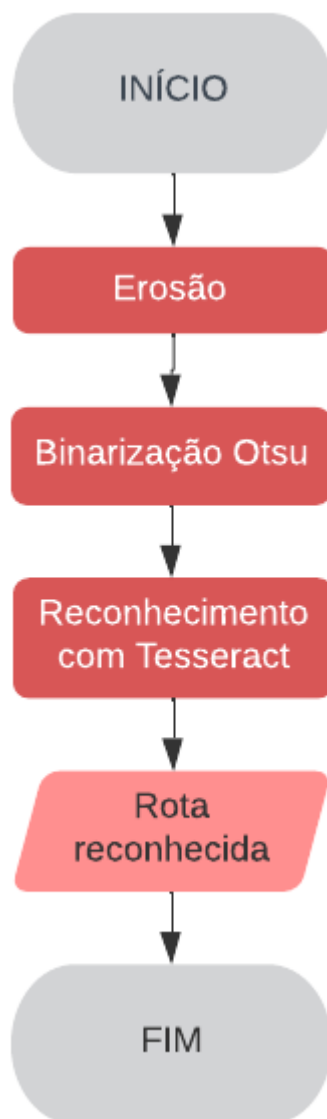
Figura 22 – Fluxograma do teste 3



Fonte: Autor (2022).

A etapa de erosão foi incluída no quarto teste, observado na Figura 23. Para isso foi utilizado um elemento estruturante quadrado com lado correspondendo a 5% da altura do recorte da parte superior da face do ônibus.

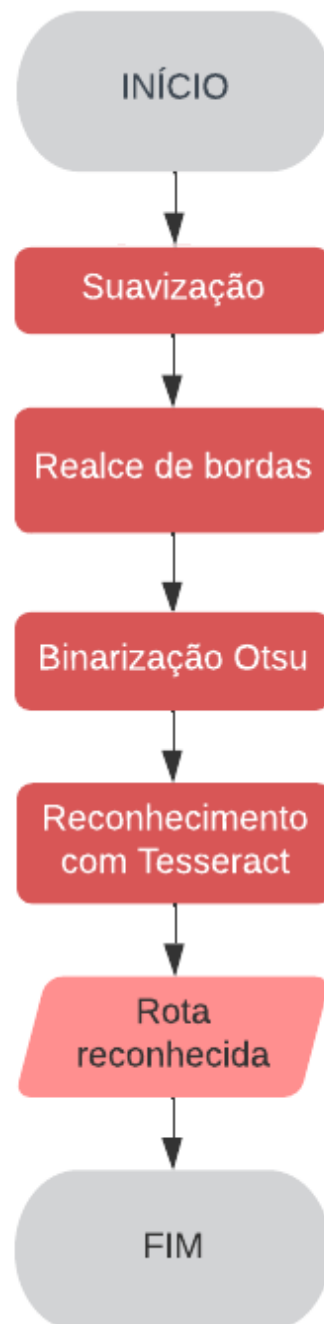
Figura 23 – Fluxograma do teste 4



Fonte: Autor (2022).

No quinto teste foram combinados o filtro de média e o realce de bordas, como mostra o fluxograma da Figura 24.

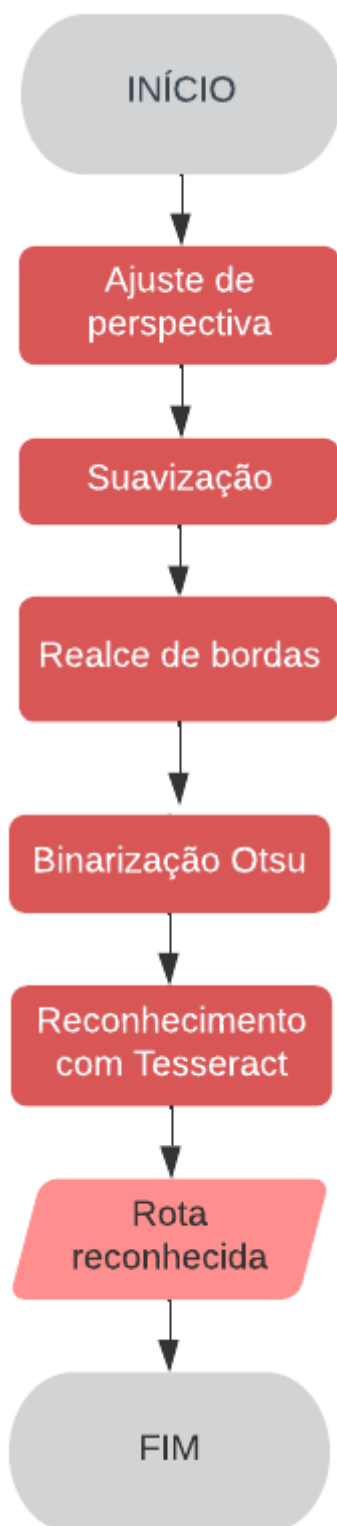
Figura 24 – Fluxograma do teste 5



Fonte: Autor (2022).

Finalmente, o fluxograma da Figura 25 mostra as etapas do teste 6 onde os parâmetros de ajuste de perspectiva, o filtro de média e o realce de contraste foram investigados.

Figura 25 – Fluxograma do teste 6



Fonte: Autor (2022).

Vale ressaltar que outros testes foram feitos considerando a redução do quadro com diferentes tipos de interpolação e diferentes combinações de parâmetros. No entanto, devido ao baixo desempenho esses testes foram desconsiderados.

## 4 RESULTADOS

Após validadas as taxas de redimensionamento, foram medidas as taxas de detecção de ônibus em cada vídeo, bem como a taxa de *quadros* por segundo, detecção e reconhecimento de rota em cada teste.

No geral, a face dos ônibus foi detectada ao menos metade dos *quadros* de cada vídeo. A média de detecção, mostrada na tabela a seguir, foi de 49,6%.

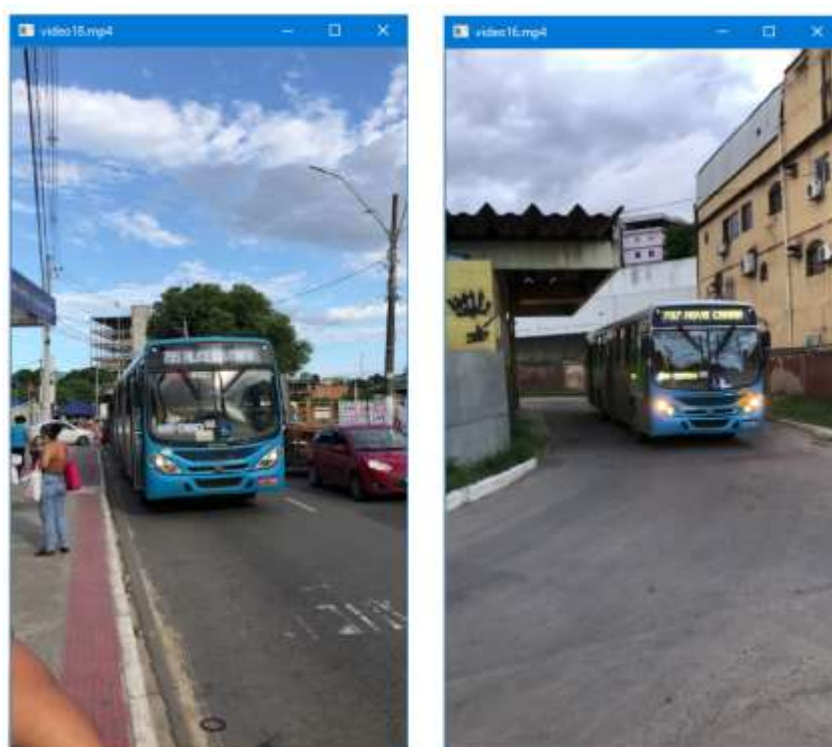
Tabela 4 – Resumo das aferições mínima, média e máxima de detecção de faces de ônibus

Detecção mínima (%)	Detecção média (%)	Detecção máxima (%)
1,32	49,6	88,5

Fonte: Autor (2022).

Contudo, a taxa foi diretamente influenciada por fatores como luminosidade, distância e posicionamento do ônibus como pode ser observado na Figura 26, contendo quadros dos vídeos 16 e 18.

Figura 26 – Vídeos com menor e maior taxa de detecção de face, respectivamente



Fonte: Autor (2022).

Os vídeos foram gravados em horários diferentes e, consequentemente, possuem níveis de luminosidade diferentes. É notável que o quadro à esquerda da figura anterior tem poucas nuvens e maior reflexo do sol no para-brisa do ônibus enquanto no quadro à direita o ônibus reflete menos luz e o céu está nublado. Vale ressaltar ainda que os modelos dos ônibus são diferentes.

Na tabela abaixo são expostas as taxas de detecção de face de ônibus em cada vídeo.

Tabela 5 – Taxa de detecção de faces de ônibus por vídeo

<b>Vídeo</b>	<b>Taxa de detecção (%)</b>
<b>Vídeo 1</b>	80,3
<b>Vídeo 2</b>	80
<b>Vídeo 3</b>	70,9
<b>Vídeo 4</b>	81,7
<b>Vídeo 5</b>	38,9
<b>Vídeo 6</b>	66,7
<b>Vídeo 7</b>	85
<b>Vídeo 8</b>	60,9
<b>Vídeo 9</b>	77,7
<b>Vídeo 10</b>	58,1
<b>Vídeo 11</b>	54
<b>Vídeo 12</b>	26,1
<b>Vídeo 13</b>	40
<b>Vídeo 14</b>	4,03
<b>Vídeo 15</b>	9,24
<b>Vídeo 16</b>	88,5
<b>Vídeo 17</b>	45,3
<b>Vídeo 18</b>	1,32
<b>Vídeo 19</b>	28,6
<b>Vídeo 20</b>	24,7
<b>Vídeo 21</b>	33,3
<b>Vídeo 22</b>	36,4

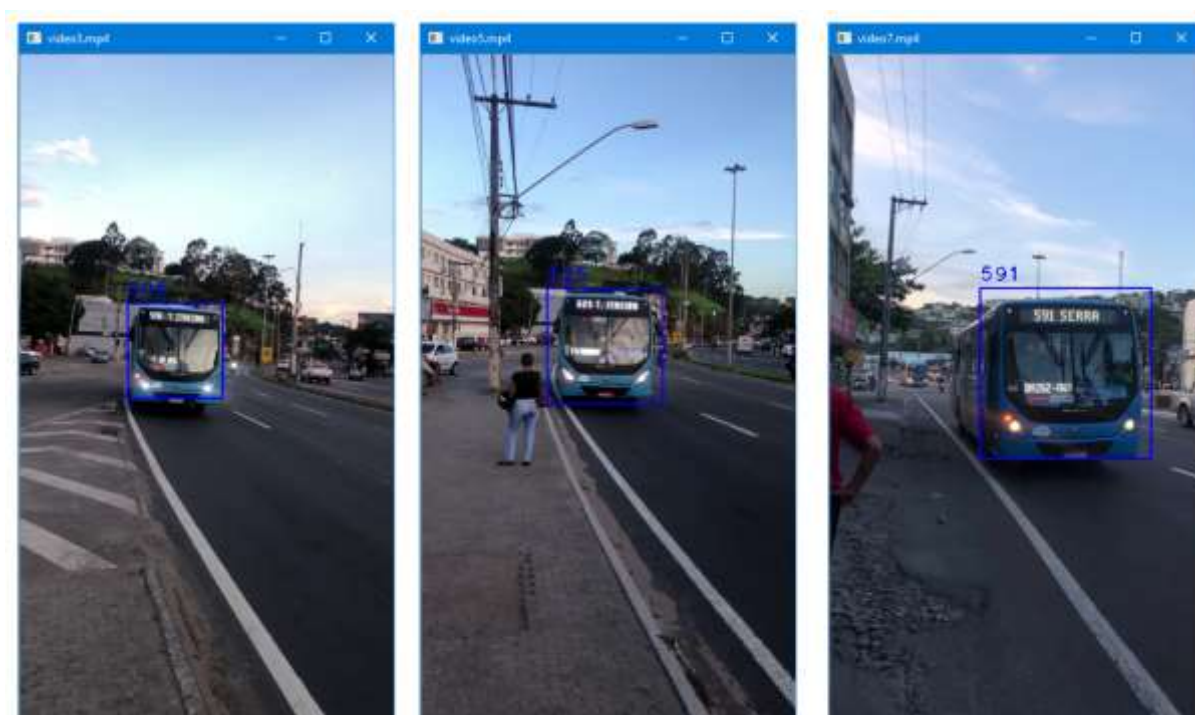
Fonte: Autor (2022).

A biblioteca do *Tesseract* para *Python*, *pytesseract*, possui 14 opções de *psm* (*page segmentation mode* – modo de segmentação da página). Com isto é possível determinar de que forma o texto está disposto na imagem, o que economiza tempo de processamento na detecção dos caracteres. Em todos os testes foi configurado o modo de segmentação de página *config='--psm 6'*, assumindo, portanto, que existe um único bloco uniforme de texto na imagem.

Mesmo considerando a imagem contendo um bloco único de texto, por vezes o reconhecimento do *Tesseract* retornou mais de uma linha de texto, letras e caracteres especiais como resultado. Para contornar essa falha e facilitar a contagem de acertos, foi estabelecido filtro que descarta as letras e os caracteres especiais. Com isso, apenas grupos de 3 algarismos são retornados e contabilizados como rota detectada.

Depois de detectada, a rota é comparada com as rotas da base de dados de vídeos. A Figura 27 mostra vídeos onde a rota foi reconhecida corretamente.

Figura 27 – Exemplo de vídeos com acerto no reconhecimento da rota



Fonte: Autor (2022).

Para validação dos parâmetros escolhidos, cada teste foi aplicado apenas nos vídeos previamente selecionados. Foram escolhidos aleatoriamente os vídeos 3, 5, 9, 16 e 20. Na tabela a seguir estão reunidos os resultados de detecção e reconhecimento de rota de todos os testes.



Tabela 6 – Resumo dos resultados de detecção e reconhecimento por teste

<b>Teste</b>	<b>Vídeo com rota detectada (%)</b>	<b>Vídeos com rota correta (%)</b>	<b>Parâmetros avaliados</b>
<b>Teste 1</b>	80	40	Nenhum
<b>Teste 2</b>	80	40	Filtro de média
<b>Teste 3</b>	80	80	Ajuste de perspectiva
<b>Teste 4</b>	80	40	Erosão
<b>Teste 5</b>	80	60	Filtro de média e realce de bordas
<b>Teste 6</b>	80	60	Filtro de média, realce de bordas e ajuste de perspectiva

Fonte: Autor (2022).

No primeiro teste não houve aplicação de nenhuma etapa adicional. Assim sendo, a segmentação dos caracteres ficou exclusivamente por conta da binarização variando o limiar de Otsu. Neste teste houve uma taxa de detecção de rota em 80% dos vídeos e em 2 dos 5, ou seja, 40% a rota detectada foi reconhecida corretamente.

A filtragem espacial de média foi empregada no teste 2 com intuito de suavizar a imagem. É comum utilizar um filtro passa baixas seguido de uma binarização uma vez que a intensidade dos objetos menores se mistura ao fundo tornando objetos maiores mais facilmente detectáveis. Embora os contornos fiquem borrados, esse filtro passa baixas contribui para diminuição de ruído (GONZALEZ e WOODS, 2009). No entanto, a taxa de detecção de rota e reconhecimento permaneceram iguais aos do teste 1, 80% e 40%, respectivamente.

O terceiro teste consistiu em aplicar um ajuste de perspectiva na região da face do ônibus que contem a rota, provocando um leve achatamento. Isso acontece quando o ônibus está fazendo uma curva e a perspectiva da face do ônibus em relação ao observador não é mais perpendicular. Com isso, foi possível segmentar melhor a rota em alguns vídeos onde a face estava inclinada resultando em 80% de detecção de rota e 80% de acertos no reconhecimento.

O teste quatro é análogo ao teste dois, visto que ambos buscam diminuir o ruído. Entretanto, diferentemente do teste dois que borrava a imagem diminuindo a transição das bordas, a erosão elimina os elementos da imagem menores que o elemento

estruturante. O resultado da erosão com um elemento estruturante quadrado foi de 80% na detecção de rota e 40% no reconhecimento.

Os dois últimos testes são os únicos formados por combinações de mais de um parâmetro. No teste cinco a imagem teve as bordas realçadas após passar por uma suavização com filtro de média. O realce foi feito somando as bordas detectadas com o filtro passa altas *Canny* à imagem em escala de cinza. O teste seis, por sua vez, também teve bordas realçadas, mas além disso, foi adicionada uma etapa ajuste de perspectiva. Em ambos os testes a taxa de reconhecimento da rota foi de 60% e a taxa de detecção de rota foi 80%.

A Figura 28 mostra algumas rotas com caracteres reconhecidos com erros.

Figura 28 – Exemplo de vídeos com erro no reconhecimento da rota



Fonte: Autor (2022).

Tão importante quanto as taxas de detecção e reconhecimento de rota são as medidas de *quadros* por segundo. Por conta disso, foram aferidos os valores mínimo, médio e máximo de FPS, afim de melhorar a compreensão acerca do desempenho dos testes. Na Tabela 7 estão os valores registrados em cada teste.

Tabela 7 – Resumo das aferições mínima, média e máxima de FPS por teste

Teste	Mínimo FPS	Média FPS	Máximo FPS	Parâmetros avaliados
Teste 1	1,9	3,0	4	Nenhum
Teste 2	2,1	2,4	4,9	Filtro de média
Teste 3	2,9	4,0	5	Ajuste de perspectiva
Teste 4	2,5	4,0	5	Erosão
Teste 5	2,9	4,1	5	Filtro de média e realce de bordas
Teste 6	2,4	4,1	5	Filtro de média, realce de bordas e ajuste de perspectiva

Fonte: Autor (2022).

Também foi medido o tempo médio de processamento do primeiro quadro dado que este tempo é importante para a compatibilização entre o tempo do algoritmo e um ônibus se aproximando do ponto de parada.

Tabela 8 – Resumo das aferições médias do tempo de processamento do primeiro quadro por teste

Teste	Tempo médio de processamento do primeiro quadro (s)	Parâmetros avaliados
Teste 1	0,19	Nenhum
Teste 2	0,2	Filtro de média
Teste 3	0,21	Ajuste de perspectiva
Teste 4	0,21	Erosão
Teste 5	0,2	Filtro de média e realce de bordas
Teste 6	0,27	Filtro de média, realce de bordas e ajuste de perspectiva

Fonte: Autor (2022).

## 5 CONCLUSÃO

O processamento de imagens pode ser utilizado com diversas finalidades no âmbito da tecnologia, como por exemplo, identificação de textos para fins de orientação nas ruas. Uma aplicação proeminente é a transcrição e conversão em tempo real de frases em voz, que é de grande ajuda não só para turistas como também pessoas com deficiência visual.

Nesse estudo, a utilização do processamento de vídeos no reconhecimento da rota de um ônibus foi feita para que, futuramente, o sistema possa proporcionar independência aos deficientes visuais na hora de pegar um ônibus.

O sistema proposto possui duas etapas centrais: detecção da face do ônibus e reconhecimento da rota. Para detecção da face foi utilizado o *HaarCascade* cujo funcionamento baseia-se no trabalho de Viola e Jones (2001). Com a finalidade de ganho no custo computacional, foi realizado um teste de validação para diminuir a resolução do quadro mantendo a maior taxa de detecção e maior taxa de FPS. Para reconhecimento da rota foram testadas 6 combinações de parâmetros escolhidos previamente. Em cada teste, a imagem processada foi segmentada variando o limiar de Otsu, após reconhecimento com *Tesseract*, foram registradas as taxas de detecção de face do ônibus por vídeo, taxa de detecção de rota, taxa de acerto no reconhecimento da rota detectada e os valores mínimo, médio e máximo da taxa de FPS.

Conclui-se que, para detecção e reconhecimento de rota de ônibus em vídeos, o melhor parâmetro testado foi o ajuste de perspectiva. Embora o teste quatro tenha alcançado a mesma média de FPS e de tempo de processamento do primeiro quadro, o teste três obteve melhor desempenho. A escolha desse parâmetro afetou positivamente o resultado já que permitiu que ônibus mais afastados e em movimento de curva também fossem detectados.

## 5.1 TRABALHOS FUTUROS

A partir dos resultados obtidos, novos estudos podem aplicar o processamento de imagens para auxiliar deficientes visuais a reconhecer rotas de ônibus testando o sistema proposto em diferentes condições de tráfego ou ainda em horários com exposição à luz diferentes dos já testados. Nos horários de muita luz, uma alternativa seria o ajuste de luminosidade que pode aumentar o contraste da imagem. Além disso, novas combinações de parâmetros no pré-processamento, como operações morfológicas de abertura, fechamento e isolamento de caracteres, podem ser empregadas para avaliar o desempenho do *Tesseract*.

Neste estudo, foi implementado um classificador já treinado fornecido pela biblioteca *OpenCV* para detecção de ônibus. Uma base de dados contendo um número significativo de imagens de ônibus com modelos diferentes e ângulos de captura diferentes pode ser criada e treinada afim de averiguar se há alguma melhora na detecção de faces de ônibus pelo *HaarCascade*.

Outro fator importante observado foi o reconhecimento de mais de uma linha de texto pelo *Tesseract*. Uma alternativa para esse comportamento seria investigar outros modos de segmentação de página para o *Tesseract*. Para contornar os caracteres identificados erroneamente, uma possibilidade seria treinar a rede neural do *Tesseract* para identificar padrões de números e caracteres de acordo com o utilizado nos ônibus.

O tempo de processamento do algoritmo também pode ser otimizado. Após análise, o reconhecimento de caracteres neste trabalho foi feito em metade das faces de ônibus detectadas afim de melhorar a taxa de FPS. Esta melhoria é fundamental para viabilizar seu uso em tempo real. Nesse sentido, podem ser estudadas alternativas que acelerem o processamento dos quadros. A partir disso, pode ser feito um aplicativo para que o deficiente visual seja apto a identificar rotas de ônibus com o próprio *smatphone*. Pensando em melhorar ainda mais a experiência do usuário, é interessante implementar serviços online, como geolocalização ou ainda conectividade com aplicativos de gerenciamento de transporte público urbano já em

vigor, para aumentar a precisão da identificação e oferecer mais funcionalidades ao aplicativo.

## REFERÊNCIAS

- ABIODUN, O. I. et al. State-of-the-art in artificial neural network applications: A survey. **Heliyon**, v. 4, n. 11, p. 1-41, Novembro 2018. ISSN 10.1016/j.heliyon.2018.e00938.
- BAHI, H. E.; ZATNI, A. Text recognition in document images obtained by a smartphone based on deep convolutional and recurrent neural network. **Multimedia Tools and Applications**, n. 78, p. 26453–26481, Junho 2019. ISSN 10.1007/s11042-019-07855-z.
- CHEN, H. et al. Robust text detection in natural images with edge-enhanced Maximally Stable Extremal Regions. **18th IEEE International Conference on Image Processing**, p. 2609-2612, 2011. ISSN 10.1109/ICIP.2011.6116200.
- CHENG, C.-C.; TSAI, C.-M.; YEH, Z.-M. Detection of Bus Route Number via Motion and YCbCr Features. **International Symposium on Computer, Consumer and Control**, 2014. ISSN 10.1109/IS3C.2014.21.
- GONZALEZ, R. C.; WOODS, R. C. **Processamento digital de imagens**. 3ª. ed. [S.l.]: Pearson, 2009.
- GUIDA, C.; COMANDUCCI, D.; COLOMBO, C. Automatic Bus Line Number Localization and Recognition on Mobile Phones—A Computer Vision Aid for the Visually Impaired. **Image Analysis and Processing – ICIAP**, v. 6979, 2011. ISSN 10.1007/978-3-642-24088-1\_34.
- GUO, Y. Linear Discriminant Analysis Cauchy Estimator for Single Chinese Character Font Recognition. **Journal of Physics: Conference Series**, v. 1069, Junho 2018. ISSN 10.1088/1742-6596/1069/1/012177.
- HARRAJ, A. E.; RAISSOUNI, N. OCR ACCURACY IMPROVEMENT ON DOCUMENT IMAGES THROUGH A NOVEL PRE-PROCESSING APPROACH. **Signal & Image Processing : An International Journal (SIPIJ)**, v. 6, n. 4, 2015. ISSN 10.5121/sipij.2015.6401.
- HAYKIN, S. **Redes neurais: princípios e práticas**. 2. ed. Porto Alegre: Bookman, 2001.
- INTERNATIONAL TELECOMMUNICATION UNION. **BT.601**: Studio encoding parameter of digital television for standard 4:3 and wide screen 16:9 aspect ratios, 2011. Disponível em: <<https://www.itu.int/rec/R-REC-BT.601-7-201103-I/en>>. Acesso em: 20 Junho 2022.
- JADERBERG, M. et al. Reading Text in theWild with Convolutional Neural Networks. **International Journal of Computer Vision**, v. 116, p. 1-20, Dezembro 2014. ISSN 10.1007/s11263-015-0823-z.
- LUO, X.; ZHAO, H.; OGAI, H. A Novel Video Detection Design Based On Modified Adaboost Algorithm and HSV Model. **2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)**, Chongqing, p. 2328-2331, Março 2017. ISSN 978-1-4673-8979-2/17/\$31.00.

MALLAT. A theory for multiresolution signal decomposition: the wavelet representation. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 11, n. 7, p. 674–693, 1989. ISSN 10.1109/34.192463.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **Bulletin of Mathematical Biophysics**, v. 5, p. 115–133, Dezembro 1943.

MINETTO, R. et al. Text Detection and Recognition in Urban Scenes. **IEEE International Conference on Computer Vision Workshops (ICCV Workshops)**, Barcelona, p. 277-234, Novembro 2011. ISSN 10.1109/ICCVW.2011.6130247.

NAGANJANEYULU, G. V. S. S. K. R.; NARASIMHADHAN, A. V.; VENKATESH, K. Performance evaluation of OCR on poor resolution text document images using different pre processing steps. **TENCON 2014 - 2014 IEEE Region 10 Conference**, Bangkok, p. 1-4, 2014. ISSN 10.1109/TENCON.2014.7022357.

NAIK, Y.; ALI, A. A. A. PCA Based English Handwritten Digit Recognition. **International Journal of Advanced Research in Computer Science**, v. 8, n. 5, p. 1426-1429, Maio 2018. ISSN 0976-5697.

NEUMANN, L.; MATAS, J. Real-Time Scene Text Localization and Recognition. **25th IEEE Conference on Computer Vision and Pattern Recognition**, p. 3538-3545, Junho 2012. ISSN 10.1109/CVPR.2012.6248097.

NOMURA, S. et al. A novel adaptive morphological approach for degraded character image segmentation. **Pattern Recognition**, v. 38, n. 11, p. 1961– 1975, 2005.

PADILLA, R.; , S. L. N.; SILVA, E. A. B. D. A Survey on Performance Metrics for Object-Detection Algorithms. **2020 International Conference on Systems, Signals and Image Processing (IWSSIP)**, Niteroi, p. 237-242, Julho 2020. ISSN 10.1109/iwssip48289.2020.9145130.

PAN, H.; YI, C.; TIAN, Y. A PRIMARY TRAVELLING ASSISTANT SYSTEM OF BUS DETECTION AND RECOGNITION FOR VISUALLY IMPAIRED PEOPLE. **IEEE International Conference on Multimedia and Expo Workshops (ICMEW)**, California, Julho 2013. ISSN 10.1109/ICMEW.2013.6618346.

PANCHAL, A. A.; VARDE, S.; PANSE, M. S. Character Detection and Recognition System for Visually Impaired People. **IEEE International Conference On Recent Trends In Electronics Information Communication Technology**, p. 1492-1497, Maio 2016. ISSN 978-1-5090-0774-5/16/\$31.00.

RELATÓRIO Mundial sobre a Visão. **Organização Mundial da Saúde**, 2021. Disponível em: <<https://apps.who.int/iris/bitstream/handle/10665/328717/9789241516570-por.pdf>>. Acesso em: 14 Novembro 2021.

ROBBY, A. et al. Implementation of Optical Character Recognition using Tesseract with the Javanese Script Target in Android Application. **Procedia Computer Science**, v. 157, p. 499–505, Setembro 2019. ISSN 10.1016/j.procs.2019.09.006.

SCHUMANN, J.; GUPTA, P.; LIU, Y. Application of Neural Networks in High Assurance Systems: A Survey, p. 1-19 , 2010. ISSN 10.1007/978-3-642-10690-3.



SEIN, M. M. et al. Object Detection, Classification and Counting for Analysis of Visual Events. **2020 IEEE 9th Global Conference on Consumer Electronics (GCCE)**, Kobe, p. 274-275, Dezembro 2020. ISSN 10.1109/GCCE50665.2020.9292058.

SMITH, R. An Overview of the Tesseract OCR Engine. **Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)**, Parana, n. 9, p. 629-633, Setembro 2007. ISSN 10.1109/ICDAR.2007.4376991.

SMITH, R. et al. Tesseract User Manual. **GitHub**, 2020. Disponível em: <<https://github.com/tesseract-ocr/tessdoc>>. Acesso em: 12 out. 2020.

VIOLA, P.; JONES, M. Rapid Object Detection using a Boosted Cascade of Simple Features. **Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition**, p. I-I, 2001. ISSN 10.1109/CVPR.2001.990517.

WONGTA, P.; KOBCHAISAWAT, T.; CHALIDABHONGSE, T. H. An Automatic Bus Route Number Recognition. **13th International Joint Conference on Computer Science and Software Engineering (JCSSE)**, p. 1-6, 2016. ISSN 10.1109/JCSSE.2016.7748910.

YE, Q. & D. D. Text Detection and Recognition in Imagery: A Survey. **IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE**, v. 37, n. 7, p. 1480-1500, Julho 2015. ISSN 10.1109/TPAMI.2014.2366765.

ZHU, Y.; SUN, C.; WANG, Y. Research on the Influence of Perspective Angle on Document Image Correction Results. **5th Information Technology and Mechatronics Engineering Conference (ITOEC)**, Chongqing, Junho 2020.

ZOU, Y. et al. A Robust License Plate Recognition Model Based on Bi-LSTM. **IEEE Access**, v. 8, p. 211630 - 211641, Novembro 2020. ISSN 10.1109/ACCESS.2020.3040238.

## APÊNDICE A

Neste Apêndice encontra-se o código desenvolvido que está dividido em três partes. A parte principal é o *ocr\_tesseractc\_test.py* que implementa os outros trechos de código para mostrar a interface de testes e funções úteis. Os outros arquivos são o *ocr\_test\_generator.py* e o *busDetectorFunctions.py*.

Segue o código do arquivo *ocr\_tesseractc\_test.py*:

```
import cv2

from numpy import append

import busDetectorFunctions

import ocr_tesseract_test_generator

import numpy as np

import time


def runTest(video,percentualOfOtsu,processingParametersOCR):

    #abre video

    cap = cv2.VideoCapture(video)

    #variaveis para estatisticas das taxas de detecção da face
do ônibus, reconhecimento da rota e fps

    framesWithBusDetection = 0

    framesWithBusRouteDetection = False

    framesWithRightBusRoute = False

    totalFrames = 0

    totalProcessingTime = 0

    tesseractResult = []

    totalFrameTime = 0
```

```

#contador para utilizar apenas metade dos quadros

counter = 1

#tempo inicial

initalVideoTime = time.time()

while True:

    initialFrameTime = time.time()

    #le o video frame a frame

    frameReadingStatus, frame = cap.read()

    #teste da variavel que controla se realmente foi
retornado um frame

    if frameReadingStatus == False:

        break

    resizedFrame =
cv2.resize(frame, (int(frame.shape[1]*0.5),
int(frame.shape[0]*0.5)), interpolation=cv2.INTER_LINEAR)

    #rotaciona o frame em 90 graus

    resizedFrame = cv2.rotate(resizedFrame,
cv2.ROTATE_90_CLOCKWISE)

    frame = cv2.rotate(frame, cv2.ROTATE_90_CLOCKWISE)

    #converte o quadro em tamanho original para escala de
cinza

    convertedFrameToGrayScale = cv2.cvtColor(frame,
cv2.COLOR_BGR2GRAY)

    #converte o quadro reduzido para escala de cinza

    convertedResizedFrameToGrayScale =
cv2.cvtColor(resizedFrame, cv2.COLOR_BGR2GRAY)

    #detecta faces de onibus

```

```

        detectedBusFacade =
busDetectorFunctions.detectBusFacade(convertedResizedFrameToGr
ayScale)

        if(len(detectedBusFacade)>0 and counter%2==0):

            detectedBusFacade = 2*detectedBusFacade

            #recorte da face do onibus indentificado

            busFacade =
busDetectorFunctions.cropBusFacade(detectedBusFacade,converted
FrameToGrayScale)

            #redimensiona a face do onibus

            resizedBusFacade =
busDetectorFunctions.resizeFrame(busFacade,processingParameter
sOCR)

            #ajusta a face do onibus

            ajustedBusFacade =
busDetectorFunctions.ajustPerspective(resizedBusFacade,process
ingParametersOCR[2])

            #filtro de média

            blurredBusFacade =
busDetectorFunctions.blurFilter(ajustedBusFacade,processingPar
ametersOCR[3])

            #aplica o detector de bordas canny e soma com a
imagem original para realce de contraste

            cannyBusFacade =
busDetectorFunctions.cannyEdges(ajustedBusFacade,blurredBusFaca
de,processingParametersOCR[4])

            #binarização da face do onibus

```

```

segmentedBusFacade =
busDetectorFunctions.segmentImage(cannyBusFacade,percentualOf0
tsu)

#aplica erosão na face do onibus

busFacadeEroded =
busDetectorFunctions.erodeImage(segmentedBusFacade,processingP
arametersOCR[5])

#reconhece os caracteres da linha com tesseract

recognizedString =
busDetectorFunctions.recognizeBusRoute(busFacadeEroded)

if(len(recognizedString)==3):

    #rotas reconhecidas

    tesseractResult.append(recognizedString)

    framesWithBusRouteDetection = True

    #verificação da rota

    framesWithRightBusRoute =
busDetectorFunctions.verifyBusRoute(recognizedString)

    framesWithBusDetection += 1

counter += 1

totalFrames += 1

k = cv2.waitKey(1) & 0xff

if k == 27:

    break

videoTime = time.time()

```

```

        totalProcessingTime = np.round(videoTime -
        initalVideoTime,2)

        #salva as estatisticas no arquivo em format txt

        busDetectorFunctions.saveStringOnTxt("\nTotal de frames " +
        video + ": " + str(totalFrames) +

        ".\nTaxa de onibus detectado: " +
        str(np.round((framesWithBusDetection/totalFrames)*100,1)) + "%"
        +

        "\nTaxa de tempo por frame: " +
        str(np.round((totalProcessingTime/totalFrames),2)) + "s" +

        "\nTaxa de fps: " +
        str(np.round((totalFrames/totalProcessingTime),1)) +

        "\nResultado tesseract: " +
        str(tesseractResult),processingParametersOCR)

#retorna a quantidade de quadros com face detectada e o numero
de acertos da rota para estatisticas

        return framesWithBusRouteDetection,framesWithRightBusRoute

```

Segue o código do arquivo *ocr\_test\_generator.py*:

```

from tkinter import *

import ocr_tesseract_test01 as ocr

import busDetectorFunctions

import numpy as np

import cv2

```

```

root = Tk()

root.title("Tesseract OCR")


#variáveis que serão usadas para gerar o teste

linearInterpolation = BooleanVar(value=True)

nearstInterpolation = BooleanVar()

perspectiveAjust = BooleanVar()

blur = BooleanVar()

canny = BooleanVar()

erosion = BooleanVar()

processingParametersOCR = [False, False, False, False, False,
False,False,""]

saveResultsButtonStatus = BooleanVar()

fileNameBox = StringVar()


#base de dados de validacao

videoList
=["video3.mp4", "video5.mp4", "video9.mp4", "video16.mp4", "video2
0.mp4"]

#base de dados de teste

#videoList
=
["video1.mp4", "video2.mp4", "video4.mp4", "video6.mp4", "video7.m
p4", "video8.mp4", "video10.mp4", "video11.mp4", "video12.mp4", "vi
deo13.mp4", "video14.mp4", "video15.mp4", "video17.mp4", "video18.
mp4", "video19.mp4", "video21.mp4", "video22.mp4"]

```

```

def saveParameters():

    #parametro do resize interpolação linear
    processingParametersOCR[0] = linearInterpolation.get()

    #parametro de ajuste de perspectiva
    processingParametersOCR[2] = perspectiveAjust.get()

    #parametro de filtro de media
    processingParametersOCR[3] = blur.get()

    #parametro de canny
    processingParametersOCR[4] = canny.get()

    #parametro de erosao
    processingParametersOCR[5] = erosion.get()

    #salvar resultados
    processingParametersOCR[6] = saveResultsButtonStatus.get()

    #nome do arquivo
    processingParametersOCR[7] = fileNameBox.get()

    print("-----\n" +
        "Interpolação linear = " + str(processingParametersOCR[0])
+
        ".\nAjuste de perspectiva = " +
str(processingParametersOCR[2]) +
        ".\nFiltro de média = " + str(processingParametersOCR[3]) +
        ".\nCanny = " + str(processingParametersOCR[4]) +
        ".\nErosão = " + str(processingParametersOCR[5]) + "." +
        "\n-----")

```



```
#salva configuracoes selecionadas
```

```
def saveResults():
```

```
    if saveResultsButton.config('text')[-1] == 'ON':
```

```
        saveResultsButton.config(text='OFF')
```

```
        saveResultsButtonStatus.set(False)
```

```
    else:
```

```
        saveResultsButton.config(text='ON')
```

```
        saveResultsButtonStatus.set(True)
```

```
#inicia o processamento e faz um print das estatísticas no  
terminal ao final do processamento de cada vídeo da base de dados
```

```
def runOCR():
```

```
    busDetectorFunctions.saveStringOnTxt("\n-----  
-----\n" +
```

```
    "Interpolação linear = " + str(processingParametersOCR[0])  
+
```

```
    ".\nInterpolação por vizinho próximo = " +  
str(processingParametersOCR[1]) +
```

```
    ".\nAjuste de perspectiva = " +  
str(processingParametersOCR[2]) +
```

```
    ".\nFiltro de média = " + str(processingParametersOCR[3]) +
```

```
    ".\nRealce de contraste = "  
str(processingParametersOCR[4]) +
```

```

        ".\nErosão = " + str(processingParametersOCR[5]) + "." +

        "\n-----"
\n",processingParametersOCR)

videoWithBusRouteDetectionRate = 0

videoWithRightBusRouteRate = 0


for video in videoList:

    videoWithBusRouteDetection = False

    videoWithRightBusRoute = False

    print("Processando " + video + " ...")

    busDetectorFunctions.saveStringOnTxt("\nProcessando " +
video + " ...",processingParametersOCR)

    for percentualOfOtsu in np.arange(1,1.5,0.1):

        print("-----" + " +
str(np.round(percentualOfOtsu*100,1)) + "% " + " of otsu -----
-")

        busDetectorFunctions.saveStringOnTxt("\n-----
---- " + str(np.round(percentualOfOtsu*100,0)) + "% " + " of
otsu -----",processingParametersOCR)

        framesWithBusRouteDetection,
framesWithRightBusRoute =
ocr.runTest(video,percentualOfOtsu,processingParametersOCR)

        cv2.destroyAllWindows()

        if framesWithBusRouteDetection == True:

            videoWithBusRouteDetection = True

        if framesWithRightBusRoute == True:

            videoWithRightBusRoute = True

```

```

        if videoWithBusRouteDetection == True:

            videoWithBusRouteDetectionRate += 1

        if videoWithRightBusRoute == True:

            videoWithRightBusRouteRate += 1

        print(video + " salvo com sucesso")

        busDetectorFunctions.saveStringOnTxt("\n-----
-----" +

        "\nTaxa de videos com rota detectada: " +
str(np.round((videoWithBusRouteDetectionRate/len(videoList))*1
00,1)) + "%" +

        "\nTaxa de videos com rota correta: " +
str(np.round((videoWithRightBusRouteRate/len(videoList))*100,1
)) + "%" +

        "\n-----
\n",processingParametersOCR)

#subdivisoas da interface

frame1 = LabelFrame(root, text="Parâmetros")

frame1.grid(row=0, column=0, padx=10, pady=10, ipadx=10,
ipady=10)

frame2 =LabelFrame(root,text="Salvar resultados em .txt")

frame2.grid(row=2, column=0,padx=10, pady=10,ipadx=10)

fileNameBox = Entry(frame2,borderwidth=2)

fileNameBox.pack(side=LEFT,padx=10, pady=10,ipadx=10)

fileNameBox.insert(0,"Nome do arquivo...")

frame3 =LabelFrame(root)

```

```
frame3.grid(row=3, column=0, padx=10, pady=10, ipadx=10)
```

```
#checkbox do ajuste de perspectiva
```

```
Checkbutton(frame1,
```

```
text = "Ajuste de perspectiva",
```

```
variable = perspectiveAjust,
```

```
onvalue = 1,
```

```
offvalue = 0).pack()
```

```
#checkbox da Interpolação linear
```

```
Checkbutton(frame1,
```

```
text = "Interpolação linear (default)",
```

```
variable = linearInterpolation,
```

```
onvalue = 1,
```

```
offvalue = 0).pack()
```

```
#checkbox do filtro de média
```

```
Checkbutton(frame1,
```

```
text = "Filtro de média",
```

```
variable = blur,
```

```
onvalue = 1,
```

```
offvalue = 0).pack()
```

```
#checkbox de bordas canny
```

```
Checkbutton(frame1,  
text = "Realce de contraste",  
variable = canny,  
onvalue = 1,  
offvalue = 0).pack()
```

```
#checkbox da Erosão  
Checkbutton(frame1,  
text = "Erosão",  
variable = erosion,  
onvalue = 1,  
offvalue = 0).pack()
```

```
#botoes
```

```
saveResultsButton = Button(frame2, text="OFF", width=5,  
command=saveResults)
```

```
saveResultsButton.pack(pady=10)
```

```
runButton = Button(frame3, text="Salvar parâmetros",  
command=saveParameters)
```

```
runButton.pack(side=LEFT, padx=10, pady=10)
```

```
runButton = Button(frame3, text="Executar OCR", command=runOCR)
```

```
runButton.pack(side=RIGHT, padx=10, pady=10)
```

```
root.mainloop()
```

Segue o código do arquivo *busDetectorFunctions.py*:

```
import cv2

import numpy as np

import pytesseract as ocr

import openpyxl as op


#local onde esta salvo o arquivo contendo as características que
serao usadas para deteccao das faces de onibus posteriormente

busClassifier =
cv2.CascadeClassifier('C:\\Users\\...\\AppData\\Local\\Program
s\\Python\\Python37\\Lib\\site-
packages\\cv2\\data\\haarcascade_onibus.xml')


def detectBusFacade(convertedFrameToGrayScale):

    #detecção da face do onibus

    detectedBuses =
busClassifier.detectMultiScale(convertedFrameToGrayScale)

    return detectedBuses


def drawBusContour(detectedBusFacade, sourceFrame, string):

    #desenha um retangulo em torno da face do onibus

    cv2.rectangle(sourceFrame,

(int(detectedBusFacade[0][0]/2),int(detectedBusFacade[0][1]/2)
),
```

```
(int(detectedBusFacade[0][0]/2+detectedBusFacade[0][2]/2),int(
detectedBusFacade[0][1]/2+detectedBusFacade[0][3]/2)),

    (255,0,0),

    2)
```

```
cv2.putText(sourceFrame,string,(int(detectedBusFacade[0][0]/2)
,int(detectedBusFacade[0][1]/2)-
10),cv2.FONT_HERSHEY_PLAIN,2,(255,0,0),2)
```

```
def resizeFrame(sourceFrame,interpolation):
```

```
    '''
```

```
    interpolation[0] = habilitar interpolação linear
```

```
    '''
```

```
    if interpolation[0] == True:
```

```
        return
```

```
cv2.resize(sourceFrame,(int(sourceFrame.shape[1]*0.9),
int(sourceFrame.shape[0]*0.9)), interpolation=cv2.INTER_LINEAR)
```

```
    else:
```

```
        return sourceFrame
```

```
def cropBusFacade(detectedBusFacade,sourceFrame):
```

```
    '''
```

essa função corrige a perspectiva de no maximo 2 onibus identificados pelo haarcascade

```
    detectedBusFacade[0][0] == coordenada x
```

```
    detectedBusFacade[0][1] == coordenada y
```

```

        detectedBusFacade[0][2] == comprimento w

        detectedBusFacade[0][3] == altura h

        '''

        if len(detectedBusFacade)>=2:

            if detectedBusFacade[0][2]*detectedBusFacade[0][3] >
detectedBusFacade[1][2] * detectedBusFacade[1][3]:

                return

            sourceFrame[detectedBusFacade[0][1]:int(detectedBusFacade[0][1]
)+detectedBusFacade[0][2)],detectedBusFacade[0][0]:int(detectedBusFacade[0][0]+detectedBusFacade[0][3])]

        else:

            return

            sourceFrame[detectedBusFacade[1][1]:int(detectedBusFacade[1][1]
)+detectedBusFacade[1][2)],detectedBusFacade[1][0]:int(detectedBusFacade[1][0]+detectedBusFacade[1][3])]

        return

        sourceFrame[detectedBusFacade[0][1]:int(detectedBusFacade[0][1]
)+detectedBusFacade[0][2)],detectedBusFacade[0][0]:int(detectedBusFacade[0][0]+detectedBusFacade[0][3])]

def ajustPerspective(sourceFrame,enable):

    #ajuste da perspectiva

    if enable == True:

        pts1 = np.float32([[int(sourceFrame.shape[1]*0.15),
int(sourceFrame.shape[0]*0.05)],

        [int(sourceFrame.shape[1]*0.9),
int(sourceFrame.shape[0]*0.05)],

```



```

        [sourceFrame.shape[1]*0.9, sourceFrame.shape[0]*0.5],

        [int(sourceFrame.shape[1]*0.15),
sourceFrame.shape[0]*0.5]])

    pts2 = np.float32([[0, 0],

        [int(sourceFrame.shape[1]), 0],

        [int(sourceFrame.shape[1]),
int(sourceFrame.shape[0]*0.5)],

        [0, int(sourceFrame.shape[0]*0.5)]]])

    perspective = cv2.getPerspectiveTransform(pts1, pts2)

    return cv2.warpPerspective(sourceFrame, perspective,
(sourceFrame.shape[1], int(sourceFrame.shape[0]*0.2)))

    return
sourceFrame[int(sourceFrame.shape[0]*0.05):int(sourceFrame.shap
e[0]*0.2),int(sourceFrame.shape[1]*0.15):int(sourceFrame.shap
e[1]*0.9)]

def blurFilter(sourceFrame,enable):

    #filtro de media

    if enable == True:

        if (np.round(sourceFrame.shape[0]/20)) % 2 == 0:

            maskSize = int(np.round(sourceFrame.shape[0]/20)) +
1

        else:

            maskSize = int(np.round(sourceFrame.shape[0]/20))

        return cv2.medianBlur(sourceFrame,maskSize)

    return sourceFrame

```

```

def cannyEdges(sourceFrame,bluredBusFacade,enable):

    #drealce de bordas com detector de canny

    if enable == True:

        cannyEdges = cv2.Canny(bluredBusFacade,0,200)

        return cannyEdges + sourceFrame

    return sourceFrame


def segmentImage(sourceFrame,percentualOfOtsu):

    #binarizacao com percentual de otsu

    otsu , _ = cv2.threshold(sourceFrame,0,255,cv2.THRESH_OTSU)

    _ , _ , segmentedImage =
cv2.threshold(sourceFrame,int(otsu*percentualOfOtsu),255,cv2.T
HRESH_BINARY)

    return segmentedImage


def erodeImage(sourceFrame,enable):

    #erosao da imagem binaria

    if enable == True:

        structuringElement =
cv2.getStructuringElement(cv2.MORPH_RECT,(int(np.round(sourceF
rame.shape[0]/20)),int(np.round(sourceFrame.shape[0]/20))))

        erodedImage =
cv2.morphologyEx(sourceFrame,cv2.MORPH_ERODE,structuringElemen
t)

        return erodedImage

    return sourceFrame

```

```

def recognizeBusRoute(sourceFrame):

    '''

    Page segmentation modes:

    0    Orientation and script detection (OSD) only.

    1    Automatic page segmentation with OSD.

    2    Automatic page segmentation, but no OSD, or OCR.

    3    Fully automatic page segmentation, but no OSD. (Default)

    4    Assume a single column of text of variable sizes.

    5    Assume a single uniform block of vertically aligned
text.

    6    Assume a single uniform block of text.

    7    Treat the image as a single text line.

    8    Treat the image as a single word.

    9    Treat the image as a single word in a circle.

    10   Treat the image as a single character.

    11   Sparse text. Find as much text as possible in no
particular order.

    12   Sparse text with OSD.

    13   Raw line. Treat the image as a single text line,
bypassing hacks that are Tesseract-specific.

    '''

    recognizedRoute = ocr.image_to_string(sourceFrame, lang =
'por',config='--psm 6')

    busRoute = ""

    lastCharacter = 0

```

```

for character in recognizedRoute:

    if character.isdigit():

        busRoute = busRoute + character

        lastCharacter += 1

    else:

        busRoute = ""

        lastCharacter = 0

    if lastCharacter == 3:

        break

return busRoute

```

```

def verifyBusRoute(busRoute):

    #verifica rota reconhecida com a base de rotas possiveis

    busRouteMatch = False

    with open('busRouteList.txt') as busRouteList:

        for route in busRouteList.readlines():

            if busRoute in route:

                busRouteMatch = True

                break

    return busRouteMatch

```

```

def saveStringOnTxt(string,processingParametersOCR):

    #salva resultados em um arquivo de texto

    if processingParametersOCR[6] == True:

        file = open(processingParametersOCR[7] + ".txt","a")

```

```
file.write(string)
```

```
file.close
```