INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Dissertação de Mestrado

IDENTIFICAÇÃO E CLASSIFICAÇÃO DE SINALIZAÇÃO HORIZONTAL EM AUTOVIAS UTILIZANDO OPENCV

Francisco Alves de Oliveira Júnior
Mestrando

Suzete Élida Nóbrega Correia, D.Sc.
Orientadora

Carlos Danilo de Miranda Regis, D.Sc.
Coorientador

João Pessoa – PB,
Junho de 2016

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

IDENTIFICAÇÃO E CLASSIFICAÇÃO DE SINALIZAÇÃO HORIZONTAL EM AUTOVIAS UTILIZANDO OPENCV

FRANCISCO ALVES DE OLIVEIRA JÚNIOR

Dissertação de Mestrado submetida à Coordenação do Programa de Pós-Graduação em Engenharia Elétrica do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia Elétrica no domínio da Engenharia Elétrica.

Área de Concentração: Processamento de Sinais

Suzete Élida Nóbrega Correia, D.Sc.
Orientadora

Carlos Danilo de Miranda Regis, D.Sc.
Coorientador

João Pessoa – PB, Junho de 2016
Francisco Alves de Oliveira Júnior

Dados Internacionais de Catalogação na Publicação – CIP
Biblioteca Nilo Peçanha – IFPB, *campus* João Pessoa

O48i Oliveira Júnior, Francisco Alves de.
Identificação e classificação de sinalização horizontal em autovias utilizando OpenCV / Francisco Alves de Oliveira Júnior. – 2016.
155 f.: il.
Dissertação (Mestrado em Engenharia Elétrica) – Instituto Federal de Educação, Ciência e Tecnologia da Paraíba - IFPB, 2016 / Coordenação de Pós-Graduação em Engenharia Elétrica.
Orientador: Suzete Élida Nóbrega Correia.
Coorientador: Carlos Danilo de Miranda Regis.
1. Processamento de sinais. 2. Visão computacional.
3. Sistemas de assistência ao condutor.

CDU 621.391

FRANCISCO ALVES DE OLIVEIRA JÚNIOR

IDENTIFICAÇÃO E CLASSIFICAÇÃO DE SINALIZAÇÃO HORIZONTAL EM AUTOVIAS UTILIZANDO OPENCV

Dissertação de Mestrado submetida à Coordenação do Programa de Pós-Graduação em Engenharia Elétrica do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia Elétrica no domínio da Engenharia Elétrica.

BANCA EXAMINADORA

Suzete Elida Nóbrega Correia, D.Sc., IFPB
Orientadora

Carlos Danilo de Miranda Regis, D.Sc., IFPB
Coorientador

Cleumar da Silva Moreira, D.Sc., IFPB
Membro

Gutemberg Gonçalves dos Santos Júnior, D.Sc., UFCG
Membro

Silvana Luciene do Nascimento Cunha Costa, D.Sc., IFPB
Membro

João Pessoa – PB, junho de 2016
Francisco Alves de Oliveira Júnior

Dedico esse trabalho aos meus filhos ***Giovanna***, ***Leonardo*** e à minha esposa ***Rúbia***. Todos eles se sacrificaram muito para que eu alcançasse esse objetivo. Dedico também ao meu irmão ***Bruno*** e às minhas duas mães, ***Maria da Guia*** e, em especial, à memória de ***Maria do Carmo*** que, apesar de suas origens humildes, nunca mediram esforços para a minha educação.

“Navegar é preciso; viver não é preciso”.

Fernando Pessoa.

AGRADECIMENTOS

A DEUS, que sempre me deu forças para conquistar todos os objetivos.

A minha família, Rúbia, Giovanna, Leonardo, Maria da Guia, Maria do Carmo e Bruno, agradeço pelo apoio incondicional.

A minha orientadora, professora. Dra. Suzete Élida Nóbrega Correia, por seu apoio, dedicação e competência, fatores fundamentais para a conclusão desse trabalho.

Ao meu coorientador, professor Dr. Carlos Danilo de Miranda Regis pela atenção especial nas revisões e sugestões para o aprimoramento desse trabalho.

Aos professores doutores Franklin Pamplona e José Antônio Cândido Borges da Silva, que acreditaram nesse sonho e me ajudaram a ingressar no mestrado.

A todos os professores do mestrado que, de alguma forma, contribuíram para minha formação.

Aos meus colegas mestrandos, os quais estudaram comigo inúmeras disciplinas. Seria injusto nomeá-los por receio de esquecer alguém.

Ao Tribunal Regional Eleitoral da Paraíba, instituição na qual trabalho e que disponibilizou a Licença Capacitação que em muito viabilizou a conquista desse novo desafio.

Aos colegas de trabalho do TRE-PB, que também me incentivaram e não me deixaram desistir nos momentos difíceis.

Ao IFPB, à Coordenação do Mestrado, à coordenadora, Dra. Silvana Luciene do Nascimento Cunha Costa, ao ex-coordenador do Mestrado, Dr. Jefferson Costa e Silva e a todos os funcionários da instituição que, direta ou indiretamente, propiciaram a realização desse mestrado.

RESUMO

Acidentes de trânsito podem ser fatais, causando a morte ou invalidez de motoristas e pedestres. Desta forma, muitos pesquisadores estão desenvolvendo meios para deixar os veículos mais seguros, através do uso de sistemas de apoio à condução que auxiliem os motoristas nas mais diversas situações no trânsito. O objetivo deste trabalho é propor um sistema para detecção e classificação de linhas de sinalização horizontais em autovias. Sistemas deste tipo podem ajudar a diminuir a quantidade de acidentes de trânsito, auxiliando o condutor do veículo a permanecer em sua faixa e realizar ultrapassagens apenas em locais permitidos. As imagens das autovias, capturadas por uma câmera fixada ao para-brisa no interior do veículo, são analisadas quadro a quadro em tempo real. O sistema proposto foi desenvolvido na linguagem de programação C++, utilizando a biblioteca OpenCV, de código aberto, amplamente empregada em visão computacional. Dentre outras técnicas, utilizou-se o detector de bordas de Canny e a Transformada Probabilística de Hough para identificação de segmentos de reta. Adicionalmente, foram desenvolvidos métodos geométricos para otimização e eliminação de segmentos desnecessários e um algoritmo para estimação do ponto de fuga, o qual auxilia na identificação dos segmentos mais relevantes para o sistema. Foram realizados sete experimentos apresentando diferentes níveis de dificuldade. Acurácias na faixa de valores de 86,58% a 100% foram alcançadas. Em média, os experimentos obtiveram uma acurácia de 94,56% na classificação dos diferentes tipos de sinalizações horizontais.

Palavras-chave: Visão computacional, Detecção de faixa, OpenCV, Transformada probabilística de Hough, Sistemas de assistência ao condutor.

ABSTRACT

Traffic accidents may be fatal, causing death or disability of drivers and pedestrians. Thus, many researchers are developing ways to make vehicles safer through the use of driver assistance systems that help drivers in various traffic situations. The objective of this work is to propose a system for road lane detection and classification. Such systems can help reduce the number of traffic accidents, helping the driver of the vehicle to remain in his lane and performing overtaking only in allowed places. Images of roads, captured by a camera attached to the windshield inside the vehicle are analyzed frame by frame in real time. The proposed system was developed in C ++ programming language using the OpenCV library, open source, widely used in computer vision. Among other techniques, we used the Canny edge detector and Probabilistic Hough Transform method to identify line segments. Additionally, geometrical methods for optimization and disposal unnecessary segments, as well an algorithm for estimating the vanishing point were developed, which helps to identify the most relevant segments to the system. Seven experiments were conducted showing different levels of difficulty. The achieved accuracies ranging from 86.58% to 100%. On average, the experiments obtained an accuracy of 94.56% in the classification of different types of lane boundaries.

Key words: Computer vision, Lane detection, OpenCV, Probabilistic Hough transform, Driver assistance systems.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

OMS	-	Organização Mundial de Saúde;
IPEA	-	Instituto de Pesquisa Econômica Aplicada;
PIB	-	Produto Interno Bruto;
IEEE	-	Instituto de Engenheiros Eletricistas e Eletrônicos (*Institute of Electrical and Electronics Engineers*);
ITS	-	Sistemas Inteligentes de Transporte (*Intelligent Transportation Systems*);
ADAS	-	Sistemas Avançados de Assistência a Direção (*Advanced Driver Assistance Systems);*
GPS	-	Sistema de Posicionamento Global (*Global Positioning System*);
V2V	-	Veículo para Veículo (*Vehicle-to-Vehicle*);
OpenCV	-	Biblioteca para Visão Computacional de Código Aberto (*Open Source Computer Vision Library*);
SVM	-	Máquina de Vetores de Suporte (*Support Vector Machine*);
ROI	-	Região de Interesse (*Region over Interest*);
IPM	-	Mapeamento em Perspectiva Inversa (*Inverse Perspective Mapping*);
RANSAC	-	Consenso por Amostra Aleatória (*RANdom SAmple Consensus*);
PF	-	Ponto de Fuga;
VP	-	Ponto de Fuga (*Vanishing Point);*
CONTRAN	-	Conselho Nacional de Trânsito;
IBM	-	*Internacional Business Machines;*
BSD	-	Licença do tipo *Berkeley Software Distribution;*
IROADS	-	Conjunto dados composto por imagens de rodovias;
TPH	-	Transformada Probabilística de Hough;
cc	-	Código que representa a sinalização de linha contínua;
ss	-	Código que representa a sinalização de linha seccionada;
cc	-	Código que representa a sinalização de linha contínua-contínua;
cs	-	Código que representa a sinalização de linha contínua-seccionada;
sc	-	Código que representa a sinalização de linha seccionada-contínua;
ss	-	Código que representa a sinalização de linha seccionada-seccionada;
-	-	Código que representa a sinalização de linha Sem sinalização.

Lista de Símbolos

m	-	Coeficiente angular de uma reta;
b	-	Parâmetro utilizado em coordenadas retangulares que indica a posição em que uma reta intercepta o eixo y;
r	-	Parâmetro utilizado em coordenadas polares que representa a menor distância entre uma reta e a origem das coordenadas;
θ	-	Parâmetro utilizado em coordenadas polares que representa o ângulo formado entre o segmento de tamanho r e a horizontal;
α	-	Ângulo correspondente à inclinação da reta;
$\rho\rho$	-	Comprimento, distância entre um ponto e a origem do sistema de coordenadas polares no espaço de Hough;
$\theta\theta$	-	Ângulo utilizado para localizar um ponto do sistema de coordenadas polares no espaço de Hough;
P	-	Percentual de *pixels* utilizados para o cálculo da Transformada Probabilística de Hough;
PFx	-	Valor da abscissa do ponto de fuga;
PFy	-	Valor da ordenada do ponto de fuga;
$xmax$	-	Valor máximo da abscissa x alcançado na região de interesse;
$ymax$	-	Valor máximo da ordenada y alcançado na região de interesse;
AIQ	-	Amplitude Interquartil do boxplot;
$Q1$	-	Primeiro quartil do boxplot;
$Q2$	-	Segundo quartil, correspondente a mediana, do boxplot;
$Q3$	-	Terceiro quartil do boxplot;
N	-	Quantidade de segmentos de reta encontrados pela TPH;
$line(i)$	-	Vetor que define o i-ésimo segmento encontrado pela TPH;
i	-	Índice de cada segmento encontrado pela TPH;
x_1 e y_1	-	Coordenadas que representam o ponto inicial que forma o segmento;
x_2 e y_2	-	Coordenadas que representam o ponto final que forma o segmento;
Δy	-	Diferença entre as ordenadas dos pontos que formam o segmento: $y_2 - y_1$;

ε_v - Valor limite de Δy que indica se um segmento é ou não horizontal;

xb - Indica o valor da abscissa x de um ponto cuja ordenada corresponda à base da ROI, isto é, quando $y = ymax$;

xt - Indica o valor da abscissa x de um ponto cuja ordenada corresponda ao topo da ROI, isto é, quando $y = 0$;

m - Valor correspondente à inclinação de um segmento de reta;

Δxb - É um parâmetro de entrada que representa a menor distância entre linhas adjacentes na base da ROI

ε_b - Limiar que corresponde à distância limite entre os pontos xb_i e xb_j que indique que os segmentos i e j sejam colineares;

Δxt - É um parâmetro de entrada que representa a menor distância entre linhas adjacentes no topo da ROI;

ε_t - Limiar que corresponde à distância limite entre os pontos xt_i e xt_j, para indicar colinearidade entre os segmentos i e j;

$\wedge$ - Operação lógica AND;

Δyt - É um parâmetro de entrada que representa a menor distância entre segmentos separados, mas que pertencem a uma mesma reta;

ε_L - Distância limite para se considerar a união entre dois segmentos consecutivos e colineares;

βxb - Valor limite que define a distância máxima entre valores de xb consecutivos para formação de grupos;

xc - Valor central da abscissa x que divide a ROI em duas partes iguais;

W - Parâmetro que representa a distância entre as linhas de sinalização horizontal, na parte inferior da imagem;

$largura\_B$ - Largura entre as linhas de sinalização medida na base da ROI;

$largura\_T$ - Largura entre as linhas de sinalização medida no topo da ROI;

DXB - Valor da abscissa X no ponto onde a linha direita encontra a base da ROI;

EXB - Valor da abscissa X no ponto onde a linha esquerda encontra a base da ROI;

DXT - Valor da abscissa X no ponto onde a linha direita encontra o topo da ROI;

EXT - Valor da abscissa X no ponto onde a linha esquerda encontra o topo da ROI;

$Erro$ - Variável utilizada para acumular a ocorrência de erros na

identificação das linhas, ocorridos em quadros consecutivos;

$y\_desloca$	-	Parâmetro de entrada do sistema que corresponde ao deslocamento vertical da ROI em *pixels;*
$altura$	-	Parâmetro de entrada do sistema que corresponde à altura da ROI em *pixels*;
$largura\_faixa$	-	Parâmetro de entrada so sistema que representa a largura estimada da faixa em *pixels*;
$largura\_subgrupo$	-	Limiar utilizado para identificar subgrupos de segmentos;
A	-	Representa uma classe da Matriz de Confusão;
N_{ij}	-	Representa o número de amostras pertencente à classe A_i, mas classificada como classe A_j, na matriz de Confusão.

Sumário

1 Introdução

1.1 Formulação do Problema

Uma das principais observações encontradas no Relatório Global de Segurança no Trânsito 2013 [1], elaborado pela OMS – Organização Mundial de Saúde, indica que 1,24 milhão de pessoas morrem todos os anos, pelas estradas, em todo o mundo. O mesmo relatório informa que o número de pessoas que sofrem lesões não fatais alcança a marca de 50 milhões ao ano. Essas mortes e lesões causadas pelos acidentes de trânsito têm um impacto imensurável nas famílias afetadas, cujas vidas são irrevogavelmente modificadas por essas tragédias.

No relatório da OMS [1], há a informação que 37.594 pessoas morreram em decorrência de acidentes de trânsito no Brasil. Encontra-se também a estimativa do IPEA (Instituto de Pesquisa Econômica Aplicada) sobre o ano de 2005, que aponta que as perdas econômicas em decorrência dos acidentes de trânsito chegaram a 1,2% do PIB (Produto Interno Bruto) [1, 2].

Em 2010, a Assembleia Geral das Nações Unidas aprovou a Resolução 64/2551, que proclamou a Década de Ação para Segurança Viária. A meta para a década 2011 – 2020 é estabilizar e reduzir a tendência de aumento das mortes no trânsito, poupando cerca de 5 milhões de vidas ao longo do período [1]. Para isso, cada país participante se comprometeu a desenvolver políticas públicas que atuem sobre os seguintes pilares:

- Gestão da segurança rodoviária;
- Investimento em estradas seguras e em mobilidade;
- Desenvolvimento de veículos seguros;
- Conscientização dos usuários, para que primem pela segurança;
- Resposta pós-acidente.

Embora com aplicação deficiente, o Brasil possui legislação específica para todos os cinco fatores elencados no relatório. Mesmo assim, não conseguiu reduzir os índices de acidentes [2]. Segundo os dados estatísticos fornecidos pelo portal do Departamento de Polícia Rodoviária Federal, referentes ao primeiro semestre de 2013 [3], os principais tipos de acidente nas rodovias federais brasileiras estão listados na Tabela 1.

Tabela 1: Tipos de acidentes nas rodovias federais ocorridos no primeiro semestre de 2013.

Item	Tipos	Ocorrências	Percentual (%)
1	Colisão traseira	26957	30,30
2	Colisão lateral	15112	16,99
3	Saída de Pista	13197	14,83
4	Colisão Transversal	8842	9,94
5	Colisão com objeto fixo	4799	5,39
6	Capotamento	3628	4,08
7	Tombamento	3416	3,84
8	Colisão Frontal	3047	3,43
9	Queda de motocicleta / bicicleta / veículo	2743	3,08
10	Atropelamento de pessoa	2118	2,38
11	Atropelamento de animal	2080	2,34
12	Colisão com objeto móvel	848	0,95
13	Colisão com bicicleta	773	0,87
14	Danos eventuais	520	0,58
15	Derramamento de carga	463	0,52
16	Incêndio	420	0,47
	TOTAL	**88963**	**100,00**

Fonte: Departamento da Polícia Rodoviária Federal [3].

A Tabela 2 lista as principais causas dos acidentes descritos na Tabela 1. Os dados, fornecidos pelo portal do Departamento de Polícia Rodoviária Federal, são

referentes ao primeiro semestre de 2013 [4].

Tabela 2: Principais causas dos acidentes ocorridos nas rodovias federais no primeiro semestre de 2013.

Item	Causas	Ocorrências	Percentual(%)
1	Falta de atenção	30068	33,80
2	Outras	21248	23,88
3	Não guardar distância de segurança	10270	11,54
4	Velocidade incompatível	8679	9,76
5	Desobediência à sinalização	3660	4,11
6	Ingestão de álcool	3460	3,89
7	Defeito mecânico em veículo	3435	3,86
8	Animais na Pista	2409	2,71
9	Dormindo	2321	2,61
10	Ultrapassagem indevida	1983	2,23
11	Defeito na via	1430	1,61
	TOTAL	**88963**	**100,00%**

Fonte: Departamento da Polícia Rodoviária Federal [4].

De acordo com a Tabela 2, percebe-se que as causas: Falta de atenção (33,80%), Desobediência à sinalização (4,11%), Dormindo (2,61%) e a Ultrapassagem indevida (2,23%), quando contabilizadas juntas, representam 42,75% dos acidentes ocorridos nas rodovias federais no ano de 2013. Essas causas podem provocar uma saída indevida da faixa e, consequentemente, uma colisão frontal, que pode ser fatal.

1.2 Justificativa

A edição de janeiro de 2015 da revista Spectrum do IEEE [5] traz uma reportagem sobre a construção de rodovias inteligentes (*smart roads*), as quais permitem a comunicação entre os próprios carros e entre os carros e a rodovia, formando o que se denominou de ITS (*Intelligent Transportation Systems*).

No mesmo periódico, fala-se sobre o projeto de uma rodovia inteligente localizada na Europa, partindo de Roterdã, passando por Munique e Frankfurt até chegar à Viena, harmonizando os padrões de trânsito entre os três países

envolvidos e formando o que se designou de *Cooperative ITS Corridor.* Porém, o verdadeiro líder nessa corrida tecnológica é o Japão, uma vez que possui rodovias inteligentes que informam aos motoristas sobre as condições do trânsito e os limites de velocidade por sensores infravermelhos [5].

Utilizando uma abordagem diferente, os Estados Unidos não pretendem investir em rodovias inteligentes a curto prazo. O governo americano decidiu financiar pequenos projetos de pesquisa que dão ênfase à comunicação entre veículos, transferindo todo o custo de implementação às montadoras e aos consumidores finais [5].

Seguindo a ideia norte-americana, onde os veículos seriam, pelo menos inicialmente, a parte mais inteligente do sistema, deve-se levar em consideração tecnologias conhecidas como ADAS, do inglês *Advanced Driver Assistance Systems,* que são sistemas de apoio a condução de automóveis, que aumentam o nível de segurança dos carros e, consequentemente, da rodovia [6, 7]. Desenvolvidos para automatizar, adaptar, melhorar a segurança e a forma de se conduzir os veículos, tais sistemas também servem para evitar acidentes, oferecendo tecnologias que alertam o condutor sobre o surgimento de potenciais problemas. Alguns são capazes de acionar, de forma autônoma, controles do carro em situações de perigo no intuito de evitar colisões. Exemplos de sistemas ADAS:

- Controle automático das luzes;
- Piloto automático;
- Frenagem automática;
- GPS incorporado a alertas de trânsito;
- Conexão com *smartphones*;
- Alertar ao condutor sobre aproximação de outros carros ou perigos;
- Manutenção do veículo na faixa correta;
- Alertar perigos provenientes do ponto cego.

A tecnologia ADAS pode ser baseada em sistemas de visão/câmera, tecnologia de sensores, rede de comunicação de dados entre veículos V2V (*Vehicle-to-Vehicle*) ou comunicação entre os veículos e a infraestrutura das rodovias.

O desenvolvimento de sistemas ADAS constitui um dos segmentos de maior crescimento no ramo da eletrônica automotiva [8]. Seguindo esse raciocínio, a presente dissertação pretende apresentar um sistema para a identificação e classificação das sinalizações horizontais básicas presentes nas rodovias, utilizando-se de técnicas de visão computacional. O sistema proposto foi desenvolvido na linguagem C++ e pode ser adaptado para informar ao condutor sobre qualquer saída de faixa que lhe possa trazer perigo, quer seja por uma ultrapassagem em local proibido, quer seja por falta de atenção ou sonolência por parte do condutor. Esse é um tipo de sistema ADAS conhecido como "*Lane Departure Warning System*", que em português significa Sistema de Aviso de Saída de Faixa. Sistemas como esse ajudariam a diminuir as estatísticas de mortes causadas por colisões frontais.

Para a elaboração do programa foi utilizada a biblioteca OpenCV, criada pela Intel, gratuita tanto para uso acadêmico quanto comercial [9]. A biblioteca OpenCV é largamente utilizada em visão computacional, possuindo grande quantidade de funções úteis para o desenvolvimento do sistema proposto. O sistema deverá trabalhar em tempo real, analisando as características das sinalizações horizontais básicas (linhas contínuas e/ou seccionadas).

1.3 Estado da Arte

Muitos estudos estão sendo realizados sobre o desenvolvimento de sistemas ADAS. Na literatura, encontra-se diversas pesquisas sobre detecção das faixas (*Lane Detection*) e sobre saídas de faixa (*Lane Departures Warning*). Com relação à classificação da sinalização horizontal, tem-se:

- No trabalho desenvolvido por Paula e Jung [10], o sistema proposto classifica somente a linha à esquerda do veículo. Analisa-se uma região com dimensões de $0{,}03W$ *pixels* de altura por $0{,}20W$ *pixels* de largura, localizada na parte inferior e à esquerda da imagem. O parâmetro W representa a distância estimada entre as linhas de sinalização horizontal, na parte inferior

da imagem. A medida que o veículo se move, a evolução temporal das linhas fornece pistas sobre o tipo de marcação. Classificadores binários em cascata são utilizados para distinguir cinco tipos de sinalizações: linha seccionada, contínua, seccionada-contínua, contínua-seccionada e duplo-contínua. Como técnica de classificação, utilizou-se o SVM (*Support Vector Machine*) que é um modelo de algoritmo utilizado em aprendizagem de máquina (*Machine Learning*). O programa foi escrito em C++, utilizou-se a biblioteca OpenCV e imagens de uma câmera no para-brisa do veículo. Segundo Paula e Jung [10], os resultados globais apresentaram taxa de acerto da ordem de 78,07%;

- Em dezembro de 2015, um novo sistema foi proposto por Paula e Jung [11]. Este artigo é uma evolução do artigo Paula e Jung [10]. Uma acurácia em torno de 96% sobre o conjunto dos testes realizados foi alcançada. Tem-se a detecção e classificação das linhas de sinalização utilizando uma câmera *onboard*. Inicialmente, os limites da pista são detectados utilizando um modelo de pista linear-parabólica. Os pontos próximos são modelados como uma função linear e os pontos mais distantes, como funções parabólicas. Em seguida, um procedimento para calibração automática da câmera *on-the-fly* é aplicada. Então, um esquema de suavização adaptativa é aplicado para reduzir o ruído, unindo arestas próximas. Pares de valores máximos e mínimos locais do gradiente são utilizados como indicativos para identificar marcações na pista. Em seguida, um método dividido em duas fases é utilizado para classificar as marcações: na primeira fase, um classificador Bayesiano é aplicado para classificar as marcações das faixas presentes em cada quadro de uma sequência de vídeo como: seccionada, contínua, seccionada-contínua, contínua-seccionada ou duplo contínua. Finalmente, o segundo estágio faz a separação entre as linhas seccionada-contínua e contínua-seccionada;

- No trabalho desenvolvido por Ding et al. [12], o sistema identifica a localização das linhas de sinalização e as classifica. É realizada a combinação das imagens adquiridas por uma câmera no interior do veículo

com informações de GPS e dados de mapas digitais. Como métodos, são utilizadas duas ROIs (Regiões de Interesse), uma para imagens próximas e outra para imagens distantes. A técnica IPM (*Inverse Perspective Mapping*) é empregada para simular uma vista aérea da autovia. O valor de limiar utilizado é encontrado pelo algoritmo de Otsu[1]. Operações morfológicas são utilizadas para eliminar ruídos. O sistema classifica as combinações de linhas contínuas e seccionadas e, adicionalmente, identifica as cores amarela e branca. Os cálculos são realizados combinando 20 quadros. Segundo Ding et al. [12], foram testados 1072 quadros de diferentes sequências de vídeo. A taxa de acerto alcançada foi de 96,7%;

- O sistema proposto por Li et al. [13] identifica a localização das linhas de sinalização sem, no entanto, realizar a classificação das mesmas. O método divide verticalmente a ROI em duas partes, converte as imagens em tons de cinza e aplica as técnicas de corrosão e expansão. As linhas horizontais são suavizadas aplicando o operador de Sobel. As bordas são determinadas pelo algoritmo de Canny e a Transformada de Hough é aplicada para a identificação das retas. Nos testes, foram utilizados conjuntos de imagens virtuais com taxa de acerto de 80% e imagens reais com taxa de acerto de 85%.

Dentre os métodos mais utilizados, tem-se a detecção de bordas pelo algoritmo de Canny em [15–21], Transformada de Hough em [15–25], método RANSAC[2] em [20, 24–26]. Outro método importante é a estimação do ponto de fuga (v*anishing point)* que auxilia na identificação correta das linhas, utilizada em [19, 27].

Nesse trabalho, aplicou-se alguns dos métodos descritos anteriormente, tais como a escolha de uma Região de Interesse localizada na parte inferior da imagem, a estimação do ponto de fuga para a seleção de segmentos de reta que possam compor as sinalizações horizontais, o algoritmo de Canny para detecção de bordas e

1 O algoritmo de Otsu é um método de limiarização, proposto por Nobuyuki Otsu. Seu objetivo é, a partir de uma imagem em tons de cinza, determinar o valor ideal de um limiar que separe os elementos do fundo e da frente da imagem em dois aglomerados, atribuindo a cor branca ou preta para cada um deles;

2 RANSAC é uma abreviatura de "*RANdom SAmple Consensus*". É um método iterativo para a estimativa dos parâmetros de um modelo matemático.

a Transformada Probabilística de Hough para a identificação dos segmentos de reta.

Além dos métodos citados anteriormente, desenvolveu-se algoritmos para otimização/eliminação dos segmentos de reta duplicadas e/ou colineares, o uso do método Boxplot para determinação dos pontos de fuga, métodos para eliminação de segmentos que não convergem para o ponto de fuga, métodos para ordenação e identificação dos conjuntos de segmentos formadores das sinalizações à direita e à esquerda e métodos para a classificação das linhas identificadas.

1.4 Objetivos

1.4.1 Objetivo Geral

Desenvolver um sistema para auxílio a condutores, escrito na linguagem C++, que emprega a biblioteca OpenCV capaz de identificar e classificar os tipos de sinalizações horizontais presentes em autovias.

1.4.2 Objetivos Específicos

Os objetivos específicos para o projeto são os seguintes:

- Identificar as técnicas empregadas em trabalhos recentes ou de maior relevância sobre a identificação e classificação de sinalização horizontal em autovias;
- Desenvolver o sistema na linguagem C++, utilizando a biblioteca OpenCV para o processamento das imagens;
- Implementar algoritmos para o processamento, identificação e classificação dos tipos de sinalizações horizontais, linhas contínuas, seccionadas ou a combinação dessas, bem como a identificação da falta de linha;
- Validar o sistema implementado realizando experimentos e comparando os resultados alcançados com aqueles obtidos em outros estudos sobre o mesmo assunto.

1.5 Organização do Texto

O texto deste trabalho encontra-se dividido em 4 capítulos. No Capítulo 1, foi apresentada a formulação do problema, a justificativa e o estado da arte, bem como os objetivos gerais e específicos para esse trabalho.

O Capítulo 2 traz a formulação teórica. Inicialmente, são explicados os tipos de linhas de sinalização que se pretende identificar e classificar. Depois, são feitas explanações sobre o Detector de Bordas de Canny, a Transformada de Hough e, finalmente, sobre Transformada Probabilística de Hough.

O Capítulo 3 discorre sobre os materiais e métodos utilizados nesse trabalho. Inicia-se mencionando a biblioteca OpenCV e descrevendo a base de dados utilizada. Em seguida, aprofunda-se na metodologia aplicada. Aqui, são explicados os métodos desenvolvidos para esse estudo, tais como o método para se estimar a localização do ponto de fuga da imagem, o uso do ponto de fuga como filtro, o método para eliminação de segmentos colineares, o método para localização das linhas horizontais e os métodos para a classificação das sinalizações.

No Capítulo 4, são exibidos os resultados experimentais obtidos com o uso dos métodos introduzidos no Capítulo 3. No total, realizam-se 7 experimentos. Cada um deles tem um grau de dificuldade diferente, apresentando desafios para o sistema. Por fim, são feitas as considerações finais no Capítulo 5.

No final do trabalho, encontram-se as referências bibliográficas e os apêndices, contendo informações referenciadas no decorrer do texto. O último apêndice traz o código do programa desenvolvido para identificar e classificar as linhas de sinalização.

2 Formulação Teórica

Neste capítulo serão abordadas as informações necessárias para o desenvolvimento do projeto proposto, iniciando com a conceituação dos tipos de sinalizações horizontais em estudo. Em seguida, discute-se o detector de bordas de Canny, a Transformada de Hough e a Transformada Probabilística de Hough, métodos essenciais para o desenvolvimento desse trabalho.

2.1 Considerações Gerais

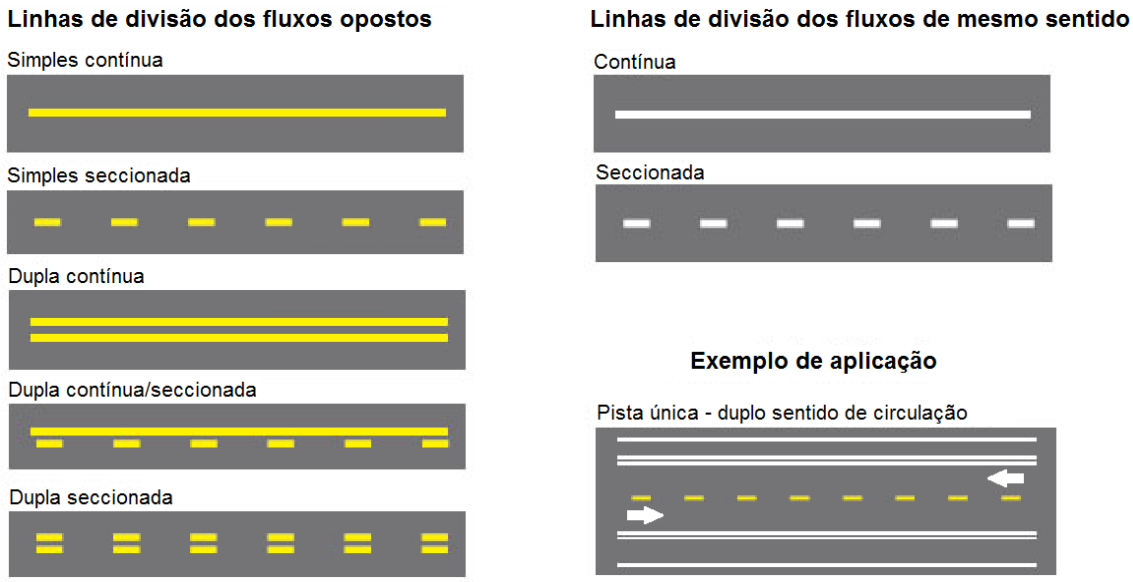
Para a compreensão dos termos empregados nesse estudo, faz-se necessário definir os tipos de sinalização horizontal existentes em rodovias, as quais pretende-se classificar. Para tanto, utilizam-se as informações contidas no Manual Brasileiro de Sinalização de Trânsito – VOLUME IV – Sinalização Horizontal – elaborado pelo CONTRAN (Conselho Nacional de Trânsito) [26].

No manual do CONTRAN [26], a sinalização horizontal é um subsistema da sinalização viária composta de marcas, símbolos e legendas, apostos sobre o pavimento da pista da autovia. Tem a finalidade de fornecer informações que permitam aos usuários adotarem comportamentos adequados, de modo a aumentar a segurança e fluidez do trânsito, ordenar o fluxo de tráfego, canalizar e orientar os usuários da via. Ela também deve ser reconhecida e compreendida por todo usuário, independentemente de sua origem ou da frequência com que utiliza a via.

2.1.1 Padrão de formas e cores

A sinalização horizontal é constituída por combinações de traçado e cores que definem os diversos tipos de marcas viárias conforme exibido na Figura 1.

Figura 1: Exemplos de sinalização horizontal.



As formas padrão utilizadas para sinalização são:

- **Contínua**: corresponde às linhas sem interrupção, aplicadas em trecho específico de pista;
- **Tracejada ou Seccionada**: corresponde às linhas interrompidas, aplicadas em cadência, utilizando espaçamentos com extensão igual ou maior que o traço;
- **Setas, Símbolos e Legendas**: correspondem às informações representadas em forma de desenho ou inscritas, aplicadas no pavimento, indicando uma situação ou complementando a sinalização vertical existente.

Nesse trabalho, pretende-se identificar o posicionamento e classificação das formas contínuas e tracejadas. Estudo sobre classificação de setas, símbolos, legendas e sinalização vertical será abordado em trabalhos futuros.

O padrão de cores é definido por sua utilização, sendo:

- **Amarela**:
  - Separar movimentos veiculares de fluxos opostos;
  - Regulamentar ultrapassagem e deslocamento lateral;
  - Delimitar espaços proibidos para estacionamento e/ou parada;
  - Demarcar obstáculos transversais à pista (lombada).
- **Branca**:
  - Separar movimentos veiculares de mesmo sentido;
  - Delimitar áreas de circulação;
  - Delimitar trechos de pistas destinados ao estacionamento regulamentado de veículos em condições especiais;
  - Regulamentar faixas de travessias de pedestres;
  - Regulamentar linha de transposição e ultrapassagem;
  - Demarcar linha de retenção e linha de “Dê a preferência”;
  - Inscrever setas, símbolos e legendas.

O manual do CONTRAN [26] também regulamenta sobre a utilização das cores vermelha, azul e preta na sinalização horizontal, que poderão ser identificadas pelo sistema em trabalhos futuros. No estudo atual, converte-se as imagens para níveis de cinza. Tal procedimento atende aos objetivos propostos, pois linhas contínuas, independente de serem brancas ou amarelas, indicam a proibição de ultrapassagem. Da mesma forma, as linhas seccionadas permitem a ultrapassagem, independente de sua cor.

2.2 Canny – Detecção de Bordas

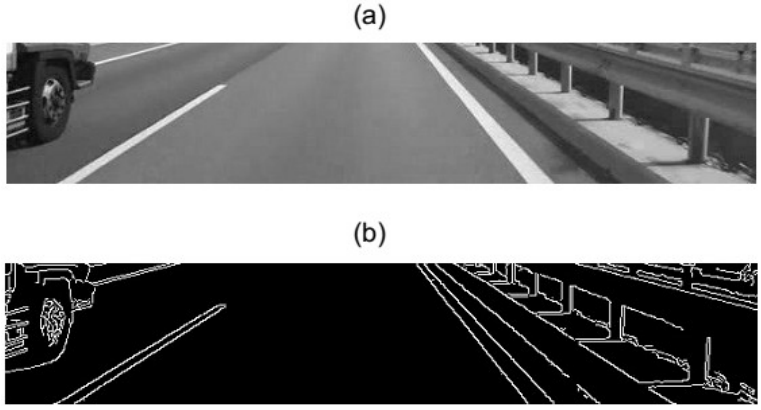
Segundo Gonzalez e Woods [27] a detecção de borda é o método usado mais frequentemente para segmentar as imagens com base nas variações abruptas (locais) de intensidade. O detector de bordas de Canny foi desenvolvido por John F. Canny em 1986 e é conhecido como “Detector Ótimo de Bordas” [28]. A abordagem do algoritmo de Canny baseia-se em três objetivos básicos [27]:

- **Baixa taxa de erros**: Todas as bordas deverão ser encontradas e não deve haver respostas espúrias;
- **Os pontos de borda devem estar bem localizados**: As bordas detectadas devem estar o mais próximas possível das bordas verdadeiras. Isto é, a distância entre um ponto marcado como uma borda pelo detector e o centro da borda verdadeira deve ser mínima;
- **Resposta de um único ponto por borda**: O detector deve retornar apenas um ponto para cada ponto de borda verdadeiro. Ou seja, o número de máximos locais em torno da borda verdadeira deve ser mínimo. Isso significa que o detector não deve identificar múltiplos *pixels* de borda em que apenas um único ponto de borda exista.

Ainda segundo Gonzalez e Woods [27], o algoritmo de Canny se resume em, primeiramente, suavizar a imagem aplicando uma função gaussiana, calcular o gradiente do resultado e, em seguida, usar a magnitude do gradiente e a direção para estimar a intensidade da borda e a direção em cada ponto.

A Figura 2 representa o resultado da aplicação do algoritmo de Canny a uma imagem em tons de cinza.

Figura 2: a) Exemplo de uma imagem de autovia em tons de cinza, obtida nesse projeto. b) Resultado encontrado após aplicação do algoritmo de Canny.



Como exposto anteriormente, na Figura 2, o algoritmo de Canny consegue identificar as bordas existentes na imagem, inclusive as bordas das sinalizações horizontais, sendo uma linha seccionada à esquerda e uma linha contínua à direita.

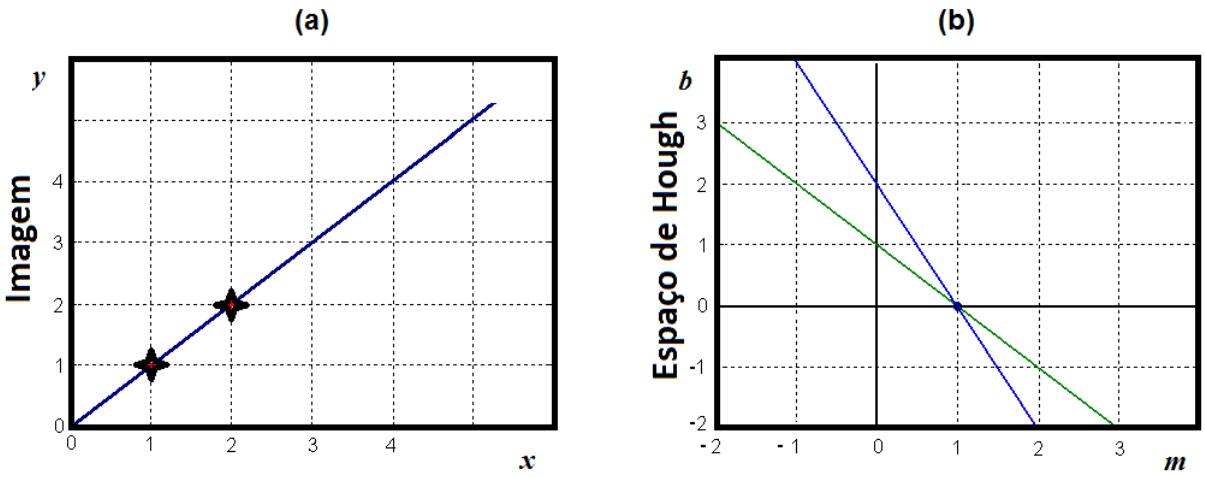
2.3 Transformada de Hough

A Transformada de Hough foi desenvolvida por Paul Hough em 1962 e patenteada pela IBM. Originalmente, foi elaborada para detectar características analiticamente representáveis em imagens binarizadas, assim como linhas, círculos e elipses. Na última década, tornou-se uma ferramenta de uso comum na visão artificial para o reconhecimento destas características [29].

Em geral, a transformada é aplicada após a imagem sofrer um pré-processamento para detecção de bordas, tal como realizado pelo método de Canny. De forma sucinta, a Transformada de Hough consiste em mapear os *pixels* da imagem em curvas no espaço de parâmetros.

Um exemplo mais simples do uso da Transformada de Hough pode ser visualizado considerando a imagem disposta em um plano cartesiano (coordenadas retangulares) como visto na Figura 3(a). Por simplificação será representado apenas dois pontos. Na Figura 3(b), tem-se a representação do espaço de parâmetros, também conhecido com espaço de Hough.

Figura 3: Exemplo de aplicação da Transformada de Hough. (a) Plano xy referente à imagem. (b) Espaço de parâmetros (Hough).



Para formar as retas do espaço de parâmetros da Figura 3, utilizam-se os pontos dispostos na Tabela 3. Nesse exemplo, fixa-se o ponto (1, 1) da imagem e varia-se os valores de m entre -2 e 3 para calcular os parâmetros b correspondentes. Assim, forma-se a linha verde no espaço de parâmetros. O mesmo procedimento é realizado para o ponto (2, 2) da imagem, formando a linha azul. Uma variável chamada de acumulador $A(m, b)$ incrementa a quantidade de vezes que um determinado par ordenado ocorre no espaço de parâmetros. As retas serão identificadas pelos acumuladores que obtiverem as maiores quantidades de incrementos. Nesse exemplo, verificou-se que o par ordenado $A(1, 0)$ ocorreu duas vezes, indicando a possível presença de uma reta que passa pelos pontos (1, 1) e (2, 2) da imagem.

Tabela 3: Cálculo dos pontos do espaço de parâmetros.

x	y	m	$b = -m.x + y$	**Acumulador $A(m, b)$ Incrementado**
1	1	-2	3	$A(-2, 3)$
1	1	-1	2	$A(-1, 2)$
1	1	0	1	$A(0, 1)$
1	1	1	0	$A(1, 0)$*
1	1	2	-1	$A(2, -1)$
1	1	3	-2	$A(3, -2)$
2	2	-2	6	$A(-2, 6)$
2	2	-1	4	$A(-1, 4)$
2	2	0	2	$A(0, 2)$
2	2	1	0	$A(1, 0)$*
2	2	2	-2	$A(2, -2)$
2	2	3	-4	$A(3, -4)$

* Único elemento do acumulador que é incrementado duas vezes, indicando colinearidade entre esses pontos.

Pela equação da reta, representada na Equação (1), verifica-se que o parâmetro m é coeficiente angular da reta e que o parâmetro b informa o deslocamento da reta no eixo y em relação à origem.

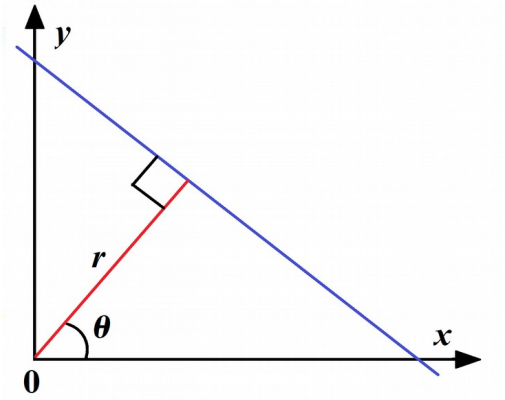
$$y = m x + b \tag{1}$$

Desta forma, para que os pontos (1, 1) e (2, 2) possam fazer parte de uma mesma reta, necessitam possuir os mesmos parâmetros m = 1 e b = 0 formando a Equação (2) que dá origem à reta da Figura 3(a).

$$y = x \tag{2}$$

Uma dificuldade prática com o uso de coordenadas retangulares é que a inclinação da reta se aproxima do infinito conforme sua inclinação se aproxima da vertical [27]. Uma maneira de contornar essa dificuldade é utilizar a representação da reta em coordenadas polares como mostrado na Figura 4.

Figura 4: Representação de uma reta em coordenadas polares.



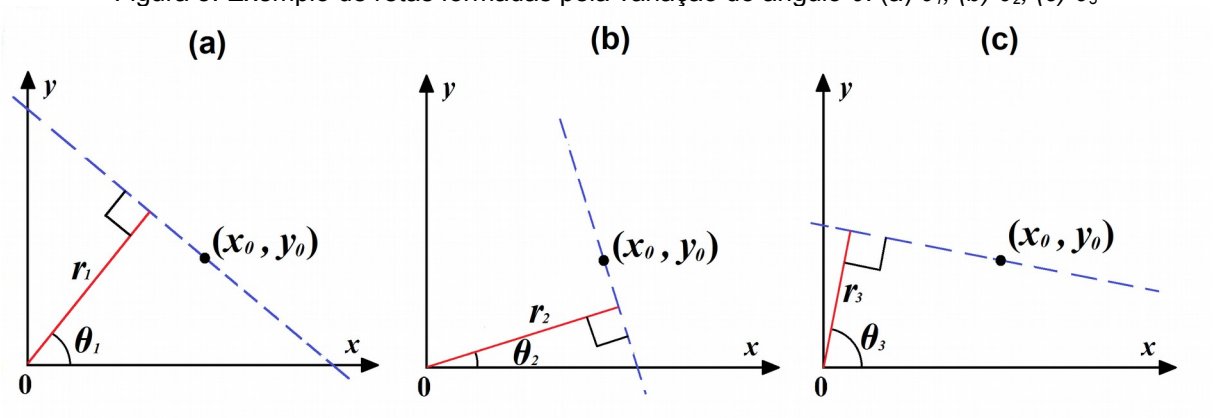
Em coordenadas retangulares foram utilizados os parâmetros m e b para formar o espaço de Hough. Em coordenadas polares, são utilizados os parâmetros r e θ. Em que r correspondem à menor distância entre a reta e a origem e θ representa o ângulo formado pelo segmento de tamanho r. Segundo o *website* OpenCV.org [30], no sistema polar, a equação da reta pode ser representada pela Equação (3). E, manipulando-se os termos, tem-se a Equação (4). A representação de uma reta em coordenadas polares, dada pelas Equações (3) e (4), é demonstrada no Apêndice C.

$$y = -\left(\frac{\cos\theta}{sen\,\theta}\right)x + \frac{r}{sen\,\theta} \tag{3}$$

$$r = y\,sen\,\theta + x\cos\theta \quad (4)$$

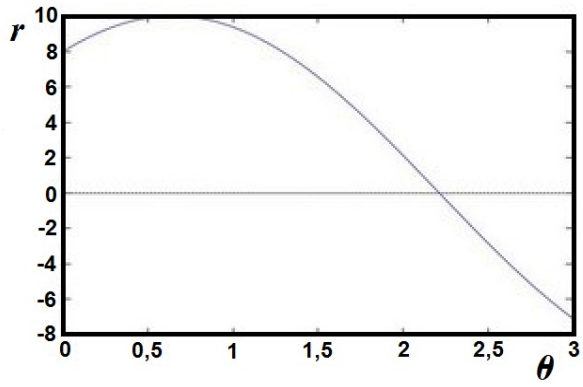
Para cada ponto (x, y) da Equação (4), varia-se o valor de θ e calcula-se o novo valor de r correspondente. A Figura 5 exemplifica a formação de três retas formadas pela variação do ângulo θ para um ponto (x_0, y_0) qualquer.

Figura 5: Exemplo de retas formadas pela variação do ângulo θ. (a) θ_1; (b) θ_2; (c) θ_3



Na Figura 5, se o ponto (x_0, y_0) fosse (8, 6), a variação dos parâmetros θ e r formaria a curva da Figura 6. Devem ser considerados apenas os valores para $r > 0$ e $0 < \theta < 2\pi$

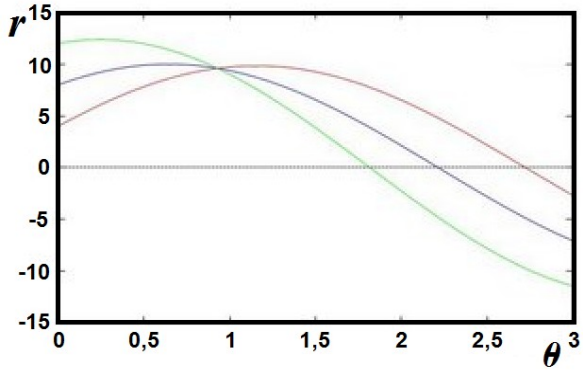
Figura 6: Curva formada pela variação dos parâmetros θ e r para o ponto (8, 6) [30]



Um exemplo interessante é proposto pelo *website* OpenCV.org [30]. Dados três pontos (4, 9), (8, 6) e (12, 6), obtém-se as três curvas exibidas na Figura 7. Cada curva foi obtida pela variação do parâmetro θ e pelo correspondente cálculo de r utilizando a Equação (4). O ponto de intersecção entre as mesmas indica os

valores dos parâmetros θ e r que formam a reta que passa pelos respectivos pontos (4, 9), (8, 6) e (12, 6).

Figura 7: Curvas plotadas no espaço de Hough. O ponto de intersecção entre as curvas indica o valor dos parâmetros da reta que passa pelos pontos (4, 9), (8, 6) e (12, 6) [30]



Desta forma, o ponto de intersecção da Figura 7 informa que os valores dos parâmetros da reta que passa pelos três pontos são: θ = 0,925 e r = 9,6. Como outro exemplo, a Figura 8 exibe a aparência do espaço de Hough aplicado a uma imagem qualquer que contenha dois segmentos de reta. Da mesma forma, os parâmetros das duas retas são identificados pelas coordenadas indicadas pelos dois pontos de intersecção.

Figura 8: Imagem ilustrativa do espaço de Hough apresentando dois pontos de intersecção. Esses pontos indicam a existência de dois segmentos de reta presentes na imagem de entrada.

2.3.1 Transformada Probabilística de Hough

A Transformada Probabilística de Hough (TPH) foi introduzida em 1990 por N. Kiryati, Y. Eldar e A. M. Bruckshtein. Para encontrar objetos, a TPH utiliza apenas uma quantidade P (0% < P < 100%) de *pixels* da imagem. Estes *pixels* são randomicamente escolhidos de acordo com uma função de densidade de probabilidade uniforme, definida a partir da imagem. Isto é equivalente a computar a transformada padrão de uma amostra da imagem [29]. Como o sistema proposto trabalhará em tempo real, optou-se por utilizar a Transformada Probabilística de Hough, por ser mais rápida.

No próximo capítulo, são descritos os métodos que serão utilizados na elaboração do sistema proposto. O produto final deverá ser um programa escrito na linguagem C++ e que se utiliza da biblioteca OpenCV para manipulação de imagens.

3 Materiais e Métodos

O sistema de identificação e classificação de sinalização horizontal proposto neste trabalho, foi desenvolvido na linguagem C++. A maioria das funções utilizadas pelo mesmo estão presentes na biblioteca OpenCV. A OpenCV (*Open Source Computer Vision Library*), originalmente, desenvolvida pela Intel, em 2000, é uma biblioteca amplamente utilizada para o desenvolvimento de aplicativos na área de visão computacional. É liberada sob a licença BSD (*Berkeley Software Distribution)* e, portanto, gratuita tanto para uso acadêmico quanto comercial, [31]. Possui interfaces C++, C, Python e Java. Aplicável aos ambientes Windows, Linux, Mac OS, iOS e Android. A OpenCV foi projetada para ter maior eficiência computacional, com um forte foco em aplicações em tempo real. Escrita em C e C++ otimizados, a biblioteca pode tirar proveito do processamento *multi-core* do processador [9, 29].

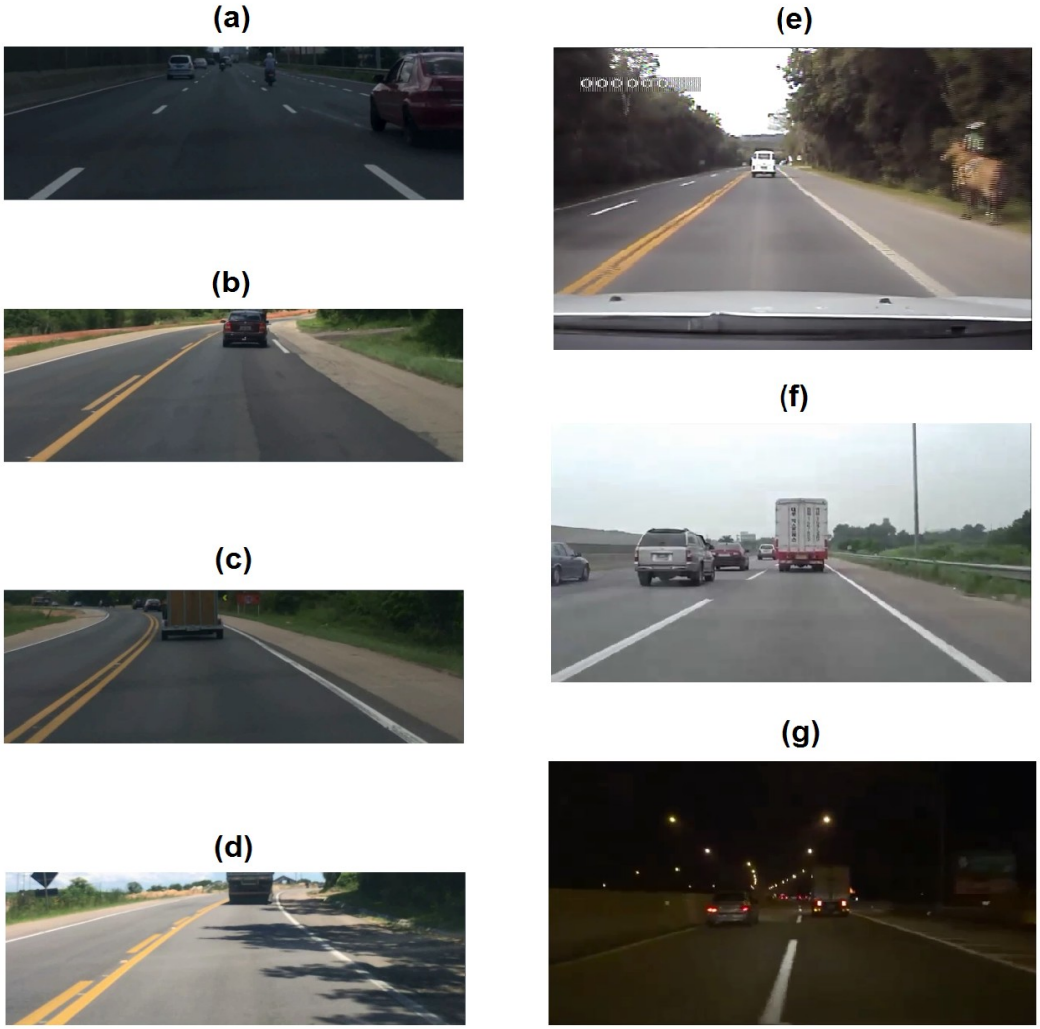
3.1 Base de Dados

Para o desenvolvimento desse estudo, foram utilizados conjuntos de dados contendo imagens de diversas rodovias. Esses conjuntos são amplamente utilizados em outros estudos com propósitos semelhantes. As imagens foram obtidas utilizando-se uma câmera instalada no para-brisa, no interior de um veículo.

Para os experimentos realizados no Capítulo 4 desse trabalho, foram

utilizados os vídeos disponibilizados por Paula e Jung [10, 11] e outros vídeos adquiridos nos endereços eletrônicos <http://youtu.be/qQHDvlsLu4c> e <http://youtu.be/vkwJnOi1RZY>. A Figura 9 mostra alguns exemplos de imagens utilizadas nos experimentos do Capítulo 4.

Figura 9: Exemplo de imagens do conjunto de dados utilizadas nos experimentos desse trabalho. (a) Vídeo “clip_85.avi”; (b) Vídeo “clip_94.avi”; (c) Vídeo “clip_96.avi”; (d) Vídeo “clip_104.avi”; (e) Vídeo “rs-040.mp4”; (f) Vídeo “japao_dia.mp4” (g) Vídeo “japao_noite.mp4”.



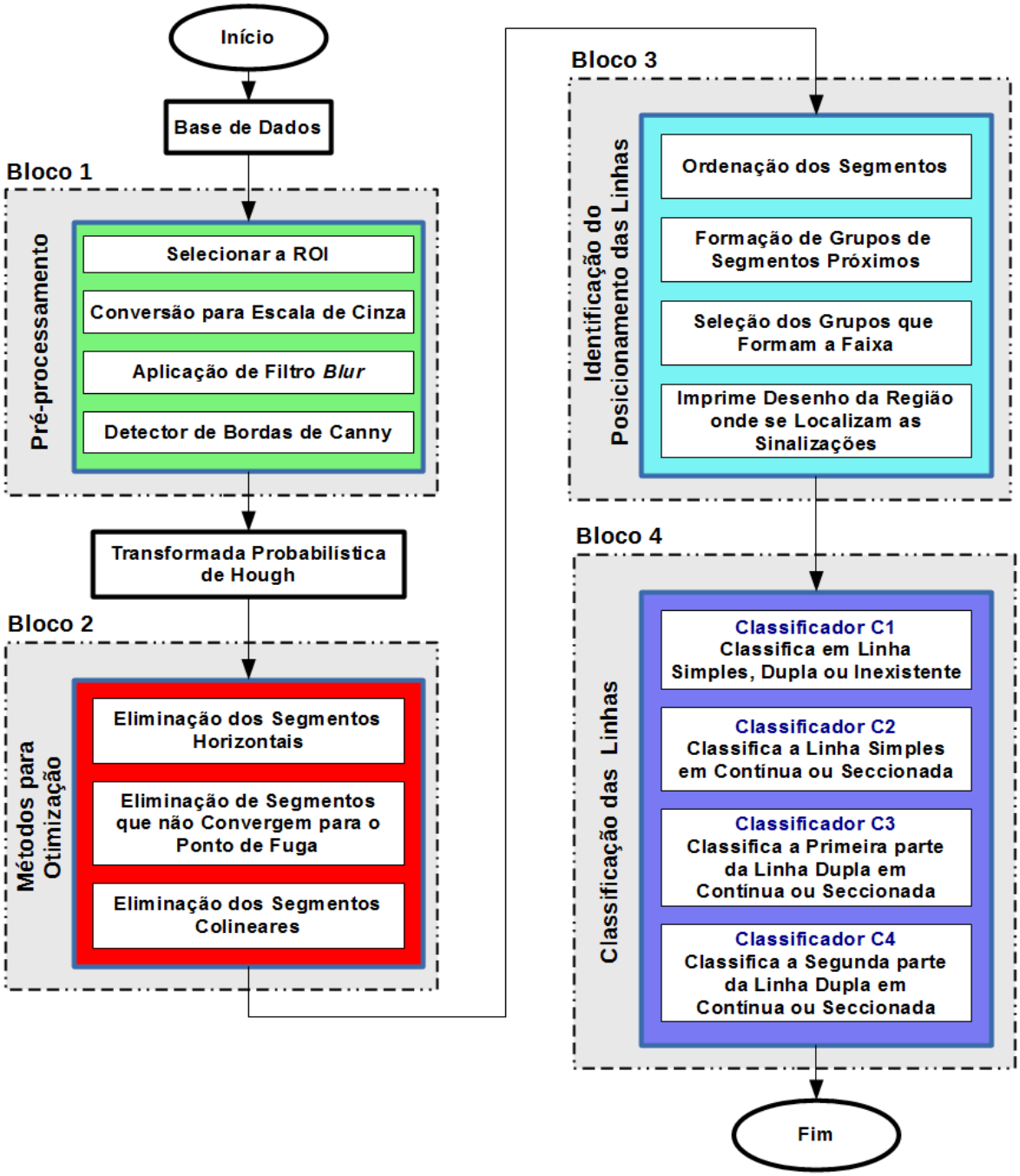
3.2 Metodologia

A Figura 10 exibe o fluxograma das operações realizadas pelo sistema. No Bloco 1, são apresentados os métodos que realizam o pré-processamento da imagem, desde a seleção da Região de Interesse (ROI) até a detecção de bordas pelo algoritmo de Canny. Em seguida, são encontrados os segmentos de reta utilizando a Transformada Probabilística de Hough (TPH). No Bloco 2, após a TPH, aplicam-se os métodos para a otimização do sistema, eliminando segmentos de reta que não convergem para o ponto de fuga. São eliminados, também, os segmentos horizontais e os segmentos duplicados e/ou colineares. No Bloco 3, encontram-se os estágios para a localização das linhas de sinalização na imagem. Por último, o Bloco 4 apresenta os métodos para a classificação das linhas da sinalização identificadas.

A contribuição desse trabalho pode ser percebida no desenvolvimento de novos algoritmos, tais como:

- Métodos para localização do ponto de fuga;
- Métodos para eliminação dos segmentos que não convergem para o centro de fuga;
- Métodos para eliminação dos segmentos duplicados/colineares;
- Métodos para identificação das linhas de sinalização a serem classificadas;
- Métodos para classificação (C1, C2, C3 e C4) das linhas selecionadas, inclusive, a opção que C1 oferece de detectar a ausência de sinalização.

Figura 10: Metodologia utilizada para a identificação e classificação das linhas de sinalização horizontais.



3.3 Pré-processamento

O pré-processamento é uma etapa crucial para o bom desempenho do sistema. Os métodos utilizados manipulam as imagens para realçar os contornos que formam as sinalizações horizontais. O pré-processamento inclui as seguintes etapas: seleção da região de interesse, conversão para a escala de cinza, aplicação do filtro *blur* e detecção das bordas de Canny, discutidas a seguir.

3.3.1 Região de Interesse

As imagens do conjunto de dados, além da pista de rolamento, exibem os carros, o céu e os canteiros laterais, representando uma dificuldade a mais na identificação da sinalização. Para amenizar esse tipo de problema, optou-se por selecionar uma Região de Interesse, ROI (*Region of Interest*), localizada na parte inferior da imagem, definida pelos parâmetros de entrada $y\_desloca$ e $altura$. O parâmetro $y\_desloca$ determina o deslocamento da ROI em relação ao topo da imagem e o parâmetro $altura$, como o nome indica, representa a altura da ROI. Com a escolha desses valores, garante-se que o sistema analisará a porção da pista que se estende desde a frente do veículo até próximo da linha do horizonte. Espaço suficiente para localização e classificação das sinalizações horizontais, como apresentado na Figura 11.

Figura 11: Seleção da Região de Interesse.

Com relação à largura da ROI, são subtraídos 2 *pixels* das margens esquerda e 2 *pixels* da direita, pois, em alguns testes, foi observado que o método de seleção de bordas de Canny tornava algumas margens visíveis.

Em resumo, a escolha de uma região de interesse também ajuda a diminuir a quantidade de cálculos exigidos na manipulação das imagens, reduzindo o custo computacional, favorecendo o processamento em tempo real. A Figura 11 demonstra o processo de seleção da ROI a partir da definição dos parâmetros de entrada *y_desloca* e *altura*.

3.3.2 Conversão para a Escala de Cinza

A ROI é convertida em uma imagem com tons de cinza. Imagens coloridas geradas por câmeras de vídeo, comumente, apresentam três canais de cores, RGB (*Red, Green e Blue*). Quando se converte imagens coloridas para a escala de cinzas, divide-se a quantidade de dados por 3, diminuindo o montante de operações a serem executadas nos processos futuros. A Figura 12 ilustra o resultado da transformação da ROI da Figura 11 em uma imagem em tons de cinza.

Figura 12: Imagem resultante em tons de cinza.

3.3.3 Aplicação do Filtro *Blur*

O filtro *Blur*, *Gaussian Blur* ou simplesmente Desfoque Gaussiano suaviza a imagem, reduzindo ruídos que poderiam ser detectados como bordas pelo algoritmo de Canny. Na Figura 13, pode-se visualizar a imagem desfocada resultante da aplicação do filtro *Blur*.

Segundo Gonzalez e Woods [27], o fato de um pequeno ruído visual poder causar um impacto significativo sobre a detecção de bordas é uma questão importante que deve ser levada em consideração, pois, na detecção de borda de Canny, que é a etapa seguinte do sistema, são calculadas as derivadas de primeira e de segunda ordem em relação à imagem.

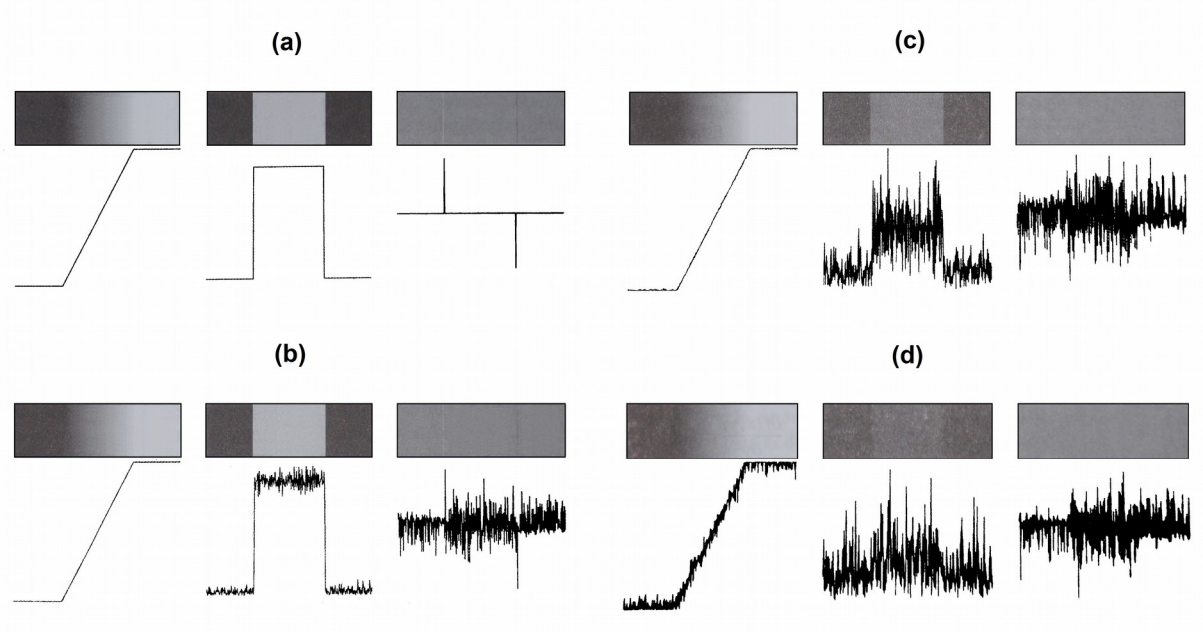
Figura 13: Imagem desfocada após o uso do filtro *Blur*.

A Figura 14 é uma adaptação de Gonzalez e Woods [27]. A mesma está dividida em quatro grupos de imagens com três colunas cada e demonstra como os ruídos podem afetar a análise das bordas. A primeira coluna de cada figura traz a representação de uma borda em declive que faz a transição de uma região escura à esquerda para uma região clara à direita. O segmento de imagem da primeira coluna da Figura 14(a) não apresenta ruído. As outras três imagens, na primeira coluna, estão corrompidas por um ruído gaussiano aditivo com média zero e desvio padrão 0,1, 1,0 e 10,0 níveis de intensidade, respectivamente. O gráfico abaixo de cada imagem é um perfil de intensidade horizontal que passa pelo centro da imagem. Todas as imagens têm resolução de níveis de cinza de oito bits, com o nível zero representando o preto e o nível 255 representando o branco.

A segunda coluna representa a imagem resultante da aplicação da primeira derivada e seus respectivos perfis de intensidade. O valor das derivadas nas regiões constantes é zero, resultando nas duas faixas escuras laterais. As derivadas nos pontos da rampa são constantes e iguais a sua inclinação. Conforme acrescenta-se o ruído, percebe-se que as derivadas ficam cada vez mais distintas se comparadas com a derivada do sinal original na Figura 14(a).

A terceira coluna mostra as imagens resultantes da segunda derivada e seus perfis de intensidade. Percebe-se que a segunda derivada é ainda mais sensível ao ruído. As linhas finas verticais são os elementos positivos e negativos da segunda derivada. Nessas imagens, o cinza representa o zero. A única imagem ruidosa de segunda derivada que se assemelha ao caso sem ruído é aquela correspondente a um ruído com desvio padrão de 0,1. As demais imagens de segunda derivada e seus respectivos perfis ilustram que seria difícil detectar seus componentes positivos e negativos, que são as características verdadeiramente úteis da segunda derivada em termos de detecção de bordas

Figura 14: Imagens de bordas em declives submetidas a um ruído gaussiano, os resultados da aplicação da derivada de primeira ordem, os resultados da aplicação da derivada de segunda ordem e os respectivos perfis de intensidade para cada caso: (a) Ruído com desvio padrão de 0,0; (b) Ruído com desvio padrão de 0,1; (c) Ruído com desvio padrão de 1,0; (d) Ruído com desvio padrão de 10,0.



Desta forma, ruídos imperceptíveis podem prejudicar a análise das derivadas. Sendo mais crítico para a derivada de segunda ordem. No intuito de amenizar este tipo de problema, o sistema proposto utiliza um filtro gaussiano antes da aplicação do algoritmo de Canny.

3.3.4 Utilização do Algoritmo de Canny

O algoritmo de Canny para detecção de bordas foi implementado através do comando *Canny*() pertencente à biblioteca OpenCV. Detalhamento sobre o uso deste comando pode ser visualizado no Apêndice A.

Dentre os parâmetros de entrada, o comando *Canny*() necessita de dois valores de limiar, *threshold1* e *threshold2*. No desenvolvimento do sistema, percebeu-se que a escolha correta desses valores é crucial para a eficiência da aplicação. Valores inadequados poderiam deixar o algoritmo muito sensível a ruídos, detectando bordas desnecessárias, fazendo com que o processamento ficasse lento. Ou, simplesmente, o método poderia diminuir sua sensibilidade, não detectando as linhas da sinalização horizontal.

Como primeira tentativa para se resolver o problema da escolha desses parâmetros, utilizou-se o limiar ótimo de Otsu como descrito em Otsu [33] e Ding et al. [12]. Em uma segunda abordagem, os valores de *threshold* foram calculados tendo como referência o valor médio de intensidade dos *pixels* de cada quadro. Em testes realizados em sequências de vídeo das bases de dados, compostas por imagens diurnas e noturnas, percebeu-se que a simples escolha dos valores 20 e 60 para os limiares *threshold1* e *threshold2*, respectivamente, apresentavam uma resposta melhor do que aquela alcançada ao se utilizar a média ou o limiar ótimo de Otsu como referência. Como exemplo do uso do algoritmo de Canny, tem-se a imagem da Figura 15.

Figura 15: Imagem resultante após aplicação do algoritmo de Canny.

3.3.5 Utilização da Transformada Probabilística de Hough

Com as bordas já detectadas, pode-se aplicar a Transformada Probabilística de Hough para se obter as coordenadas dos segmentos de retas da imagem. Em OpenCV, utiliza-se o comando $HougLinesP()$. Detalhes sobre esse comando podem ser encontrados no Apêndice B. Na Figura 16, os segmentos identificados estão representados na cor vermelha.

Figura 16: Imagem resultante final. Vê-se em vermelho os segmentos de reta detectados pela Transformada Probabilística de Hough.

São encontrados N segmentos de reta. Eles são representados pelas coordenadas dos pontos iniciais e finais de cada segmento e são armazenados no vetor "*line*", como visto na Equação (5):

$$line(i) = [x_1, y_1, x_2, y_2]_i \tag{5}$$

No vetor da Equação (5), a variável i identifica o número do segmento, variando de 0 a N – 1. As coordenadas $\boldsymbol{x_1}$ e $\boldsymbol{y_1}$ representam o ponto inicial que forma o segmento, sendo que $\boldsymbol{x_2}$ e $\boldsymbol{y_2}$, representam o ponto final.

De posse das coordenadas que definem os pontos iniciais e finais dos segmentos de reta, pode-se analisar as suas geometrias e descobrir se estes formam linhas contínuas e/ou seccionadas, bem como suas posições em relação ao veículo, identificando o primeiro conjunto de segmentos à esquerda e o primeiro à direita.

Descartar segmentos de reta desnecessários ajuda a evitar erros nos processos de identificação e classificação das sinalizações. Nas próximas seções, descreve-se como funcionam cada um dos métodos de eliminação de segmentos utilizados pelo sistema.

3.4 Eliminação dos Segmentos Horizontais

No detalhe em amarelo da Figura 17, percebe-se a presença de um dos segmentos horizontais da imagem. Para o objetivo dessa pesquisa, faz-se necessário eliminar esses tipos de segmentos. Para isso, basta verificar o tamanho do intervalo Δy conforme a Equação (6).

Figura 17: Imagem obtida após o uso da Transformada Probabilística de Hough. Os segmentos encontrados estão representados em vermelho. Em destaque, tem-se um dos segmentos horizontais.

$$\Delta y = y_2 - y_1 \tag{6}$$

$$|\Delta y| < \varepsilon_V \tag{7}$$

Assim, para se eliminar um segmento horizontal, deve ser atendida a condição da Inequação (7). Em que ε_V corresponde ao mínimo deslocamento vertical aceitável entre as alturas do ponto inicial e final que formam o segmento. Para esse estudo, escolheu-se o valor de limiar ε_V = 1% da altura da imagem em *pixels*. Como exemplo, para o vídeo de Tan et al. [22], que possui 360 *pixels* de altura, utilizou-se o limiar aproximado ε_V = 4 *pixels*.

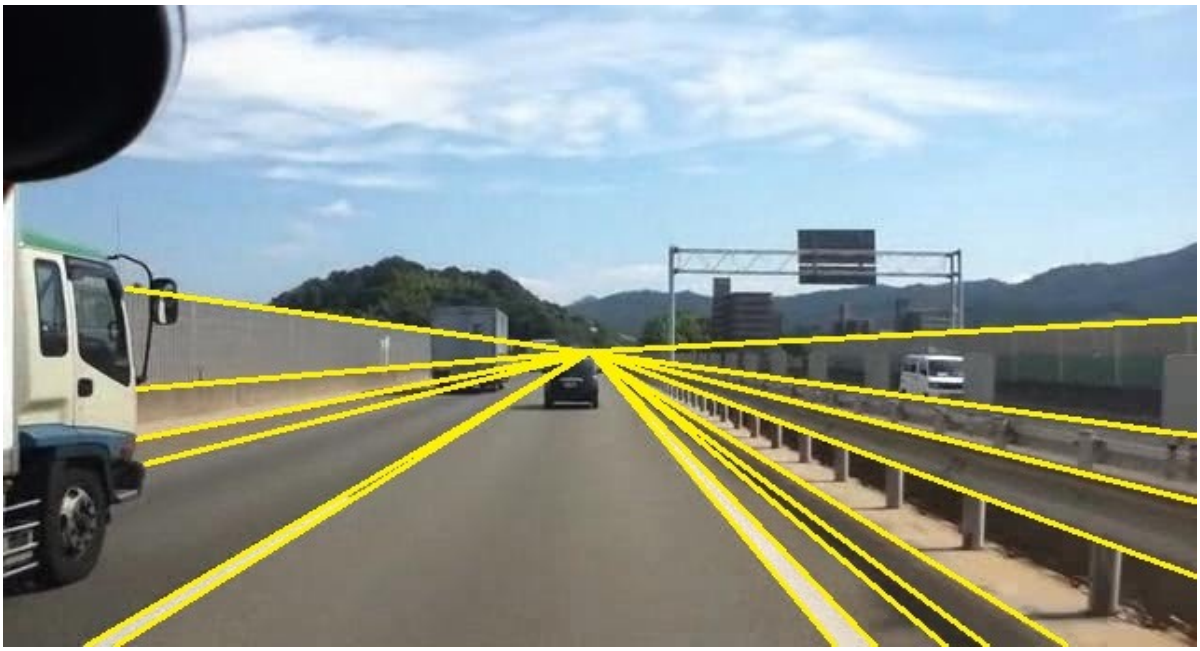
3.5 Ponto de Fuga

As sinalizações horizontais em estudo são formadas por linhas paralelas dispostas nas autovias. Quando visualizadas do ponto de vista do condutor, tais linhas se afunilam devido ao efeito da perspectiva, convergindo para uma região conhecida como ponto de fuga (PF). A Figura 18 evidencia a existência de um ponto

de fuga em um dos quadros da base de dados IROADS [34].

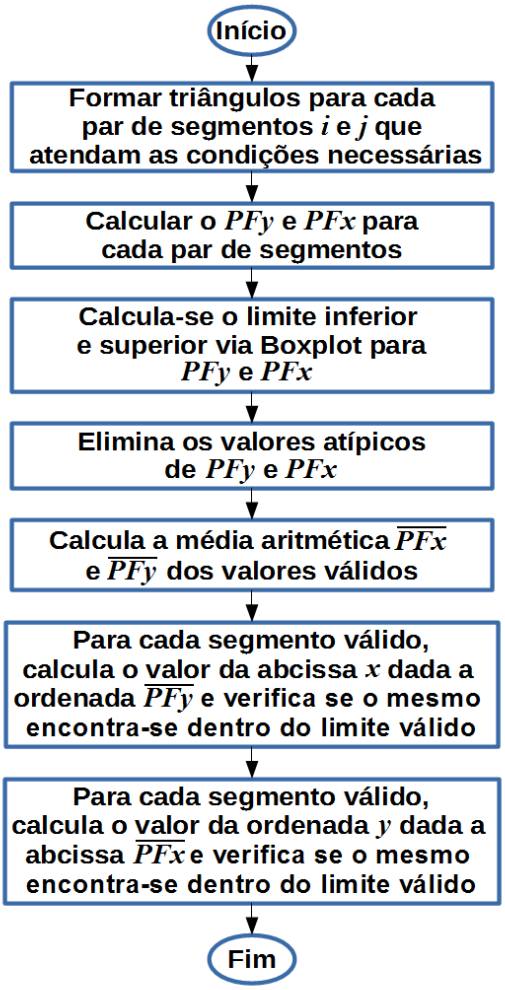
Devido a essa particularidade, o ponto de fuga pode ser utilizado para selecionar as linhas que possuem maior probabilidade de constituírem as sinalizações horizontais. Técnicas de seleção de linhas através da localização do PF são utilizadas em [25] e [17].

Figura 18: Convergência das linhas em direção ao ponto de fuga.



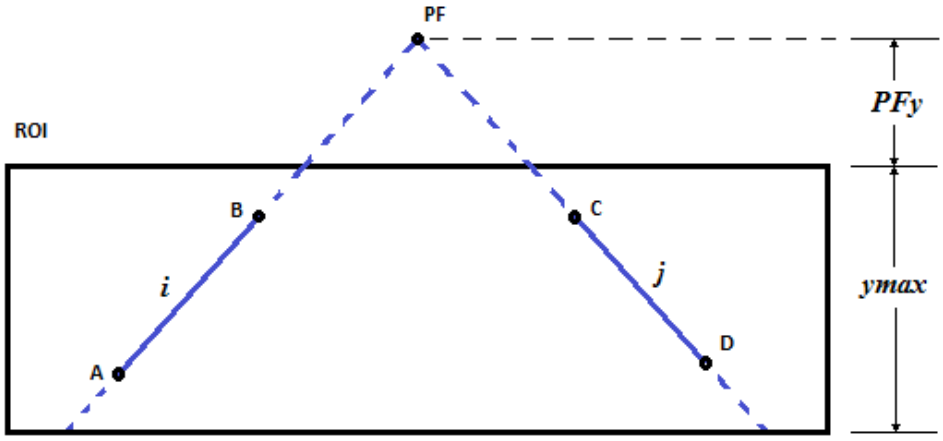
Pretende-se identificar a região mais provável onde se localiza o PF. Para isso, foram desenvolvidos os procedimentos descritos no fluxograma da Figura 19. De posse dos limites dessa região, selecionam-se apenas os segmentos de reta cuja projeção se aproxime do ponto de fuga.

Figura 19: Fluxograma para seleção dos segmentos que convergem para o ponto de fuga.



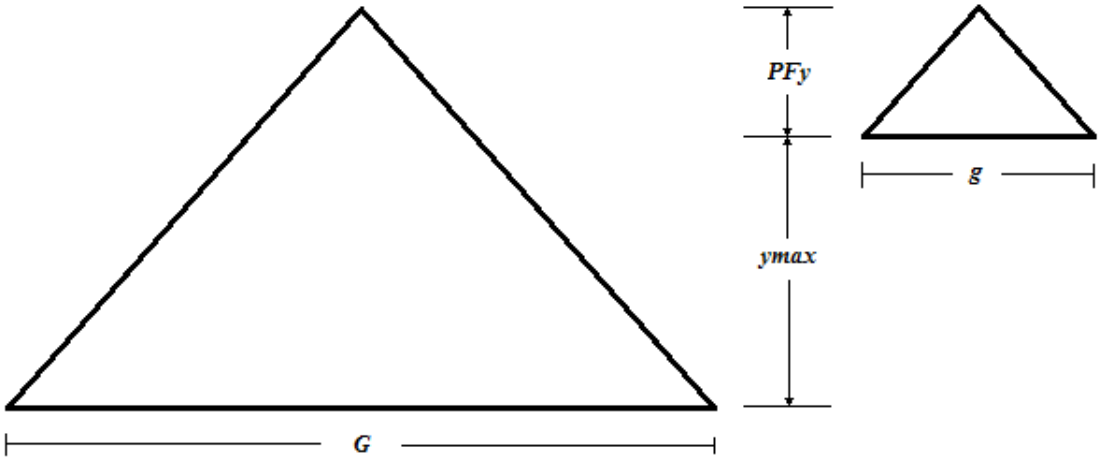
Para o cálculo do PF, utiliza-se a semelhança de triângulos. Os triângulos são formados pelo prolongamento de segmentos de reta i e j, conforme ilustrado na Figura 20.

Figura 20: Prolongamento de dois segmentos para o cálculo da altura do ponto de fuga PFy.



Na Figura 20, os prolongamentos dos segmentos $\overline{AB}$ e $\overline{CD}$ se cruzam no PF. A formação de triângulos semelhantes podem ser visualizados na Figura 21. Os valores G e g de suas bases são encontrados pela intersecção dos prolongamentos dos segmentos i e j com o limite superior e inferior da ROI.

Figura 21: Semelhança de triângulos utilizada no cálculo da ordenada do ponto de fuga PFy.



A Equação (14) demonstra como se calcula o valor PFy, que corresponde à ordenada y do ponto de fuga. Pela semelhança de triângulos, são verificadas as seguintes proporcionalidades:

$$PFy \sim g \tag{8}$$

$$(PFy + ymax) \sim G \tag{9}$$

Tem-se:

$$\frac{PFy}{g} = \frac{PFy + ymax}{G} \tag{10}$$

$$G \cdot PFy = g \cdot PFy + g \cdot ymax \tag{11}$$

$$G \cdot PFy - g \cdot PFy = g \cdot ymax \tag{12}$$

$$PFy \cdot (G - g) = g \cdot ymax \tag{13}$$

$$PFy = \frac{g \cdot ymax}{G - g} \tag{14}$$

Em que:

- PFy é o valor do ponto de fuga no eixo y tendo como referência o topo da ROI;
- g é a base do triângulo menor;
- G é a base do triângulo maior;
- $ymax$ corresponde a altura da região de interesse.

Após o cálculo de PFy, pode-se encontrar o valor da abscissa PFx**,** manipulando-se a equação que fornece o coeficiente angular m da reta. Para o segmento i ou j, tem-se que a inclinação da reta é dada pela Equação (15).

$$m = \frac{y_2 - y_1}{x_2 - x_1} \tag{15}$$

Como se conhece as coordenadas (x_1, y_1) e (x_2, y_2) do segmento, pode-se calcular o valor de m. Na Equação (15), substitui-se as coordenadas do ponto (x_2, y_2) por (PFx, PFy).

$$m = \frac{PFy - y_1}{PFx - x_1} \tag{16}$$

$$m(PFx - x_1) = PFy - y_1 \tag{17}$$

$$m \cdot PFx - m \cdot x_1 = PFy - y_1 \tag{18}$$

$$m \cdot PFx = m \cdot x_1 + PFy - y_1 \tag{19}$$

$$PFx = x_1 + \frac{(PFy - y_1)}{m} \tag{20}$$

Em que:

- PFx é o valor do ponto de fuga no eixo x;
- PFy é o valor do ponto de fuga no eixo y, calculado anteriormente;
- x_1 e y_1 é um dos pontos que formam o segmento;
- m é o valor conhecido que corresponde ao coeficiente angular do segmento de reta calculado na Equação (15).

Para se obter uma melhor estimativa da localização do PF, deve-se calcular a média aritmética dos diversos PFs encontrados pelas combinações dos segmentos existentes na imagem em análise. Porém, nem todos os segmentos apontam para uma mesma região da imagem.

Após se realizar todas as combinações de segmentos dois a dois, os valores de PFx e PFy serão analisados para eliminação de valores atípicos. Para isso, os dados são organizados no formato de boxplot.

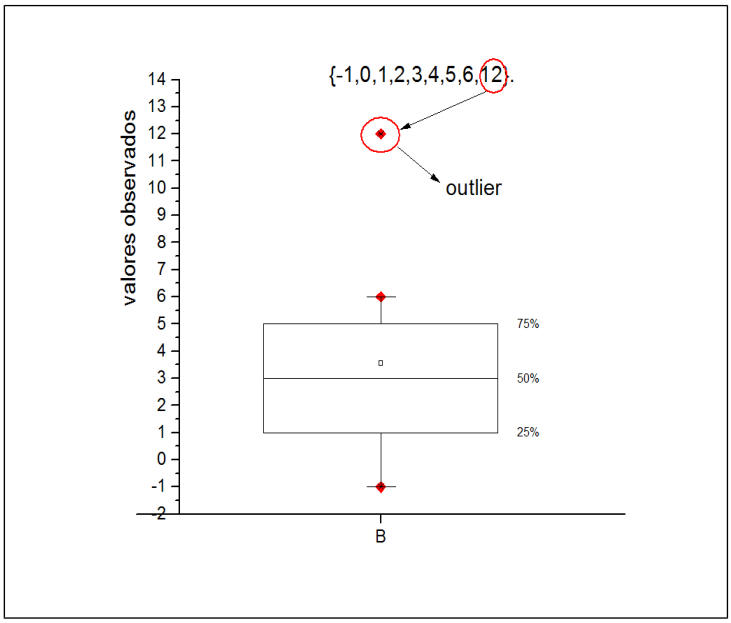
3.5.1 Boxplot

Segundo Farias [35], boxplot é um gráfico formado por uma caixa construída paralelamente ao eixo da escala dos dados (pode ser horizontal ou vertical). Ele é construído com base no resumo de cinco números, constituído por:

- Valor mínimo;
- Primeiro quartil ($Q1$);
- Mediana (segundo quartil $Q2$);
- Terceiro quartil ($Q3$);
- Valor máximo.

O tamanho da caixa se estende desde o primeiro quartil até o terceiro quartil e nela traça-se uma linha na posição da mediana. Essa caixa, que descreve os 50% centrais da distribuição, é comum a todas as variantes do boxplot. Pode-se acrescentar também uma linha, paralela à linha da mediana, para indicar a média. A Figura 22 ilustra um exemplo de representação de dados utilizando boxplot.

Figura 22: Exemplo de uso do boxplot.



Na Figura 22, percebe-se que alguns valores, chamados de *outliers*, ou valores atípicos, se distanciam do boxplot e deverão ser eliminados.

3.5.2 Valores Atípicos

A regra para identificação dos valores atípicos adotada nesse estudo, baseia-se na amplitude interquartil AIQ, definida como a distância entre o primeiro e o terceiro quartil: $AIQ = Q3 - Q1$**.** Em que AIQ é o comprimento da caixa. Quaisquer valores abaixo de $Q1$ ou acima de $Q3$ por mais de $1{,}5 \times AIQ$ serão considerados valores atípicos [35].

Após a eliminação dos valores atípicos, encontra-se a estimativa do ponto de fuga calculando-se a média aritmética dos valores válidos para PFx e PFy**,** como mostram as Equações (21) e (22).

$$\overline{PFx} = \frac{\sum_{i=0}^{n-1} PFx_i}{n} \tag{21}$$

$$\overline{PFy} = \frac{\sum_{i=0}^{n-1} PFy_i}{n} \tag{22}$$

3.5.3 Exclusão dos segmentos que não convergem para o ponto de fuga

Conforme já indicado, nem todos os segmentos de reta convergem para o ponto de fuga. Anteriormente, os valores atípicos já tinham sido desconsiderados para o cálculo das médias. Agora, faz-se necessário eliminar os segmentos que causaram tais distorções. Assim, para cada segmento, procede-se da seguinte forma:

- Calcula-se o valor da abscissa x dado o valor médio da ordenada $\overline{PFy}$. Se x estiver fora dos limites inferior e superior encontrados no boxplot, o segmento que o originou será eliminado;

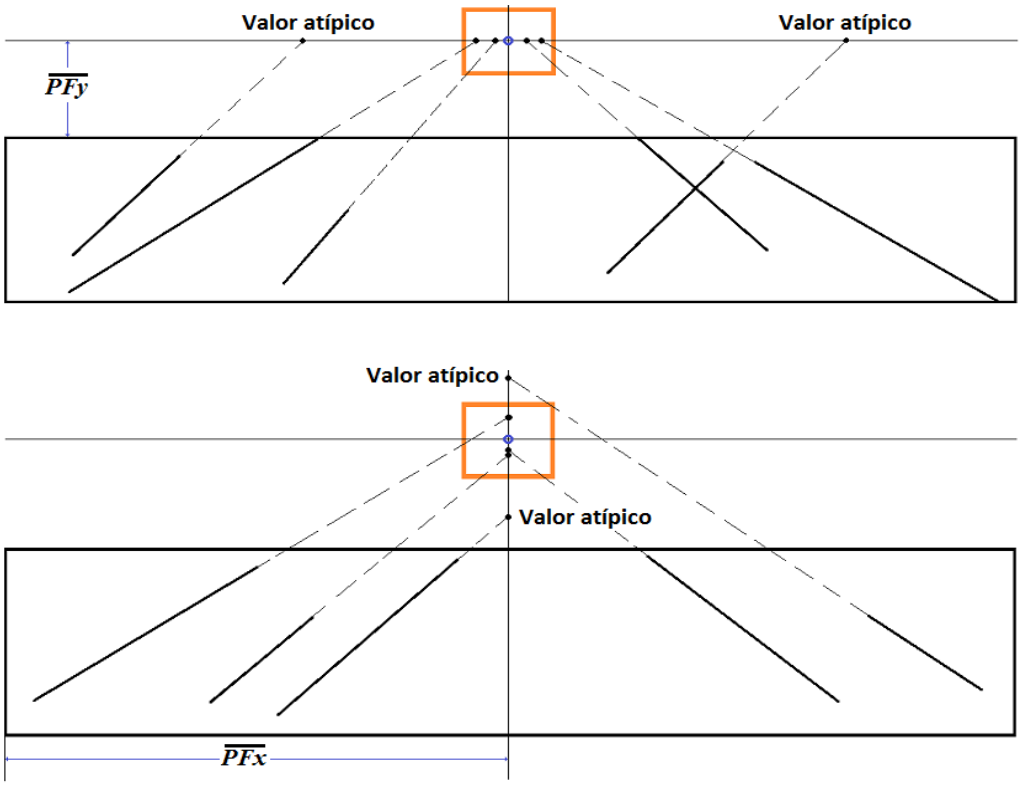
$$x = x_1 + \frac{(PFy - y_1)}{m} \tag{23}$$

- De forma análoga, calcula-se o valor da ordenada y dado o valor médio da abscissa $\overline{PFx}$. Se y estiver fora dos limites inferior e superior encontrados no boxplot, o segmento que o formou também será eliminado.

$$y = y_1 + m(PFx - x_1) \tag{24}$$

A Figura 23 ilustra o procedimento anterior.

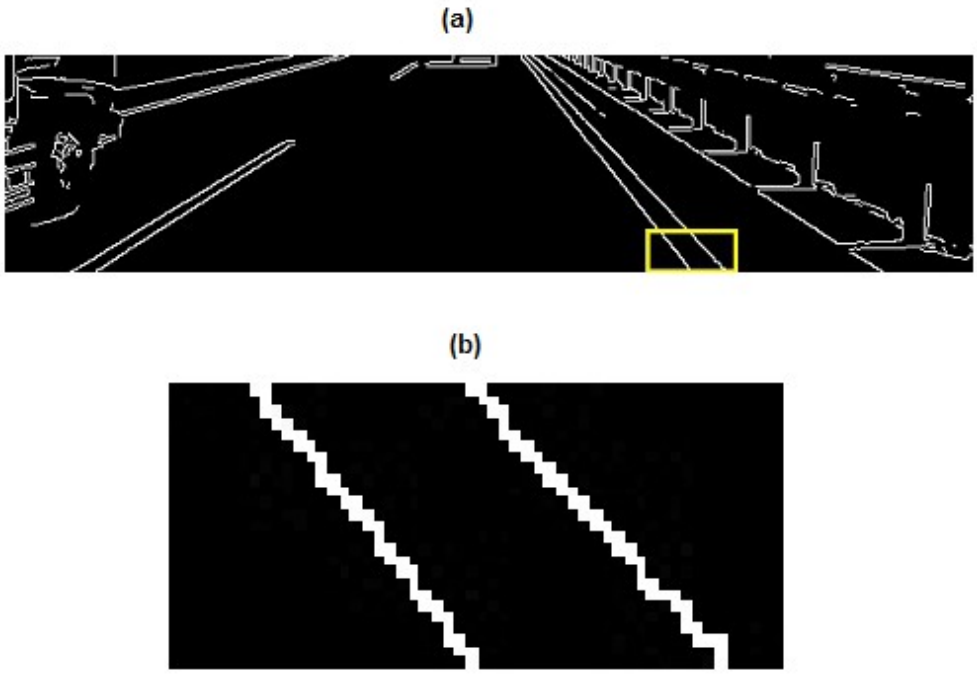
Figura 23: Seleção dos segmentos de reta que convergem para dentro dos limites do boxplot.



3.6 Exclusão de Segmentos Colineares

Após eliminar os segmentos horizontais e aqueles que não convergem para o ponto de fuga, deseja-se eliminar aqueles em duplicidade e/ou colineares. Para se entender como surgem esses segmentos desnecessários, deve-se observar a ação do algoritmo de Canny na Figura 24.

Figura 24: a) Resultado da detecção de bordas de Canny. b) Ampliação da imagem contida no retângulo amarelo para demonstração das imperfeições na formação das bordas.

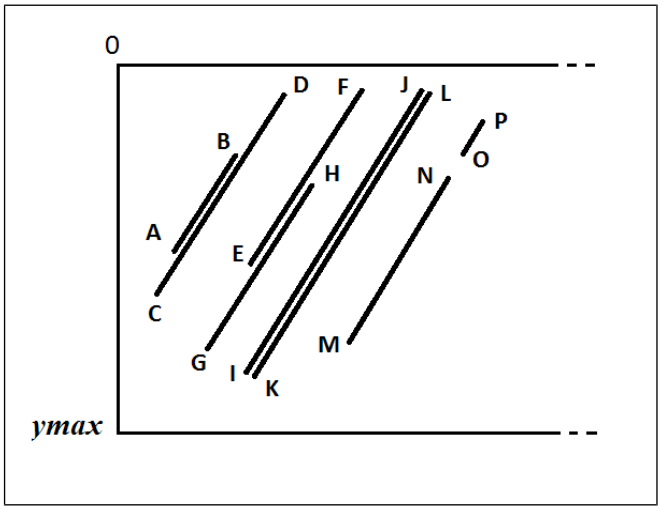


Ao se ampliar uma parte da imagem, Figura 24(b), percebe-se que as bordas selecionadas, frequentemente, possuem imperfeições que são detectadas pelo algoritmo de Hough, gerando linhas duplicadas, paralelas e/ou colineares, conforme Oliveira et al. [36].

A Figura 25 demonstra como as linhas duplicadas, paralelas ou colineares, podem surgir. O retângulo da Figura 24(a), onde se encontram os segmentos é uma representação da Região de Interesse (ROI). A ROI é composta por linhas e colunas. As linhas correspondem à altura e estão representadas pela ordenada y,

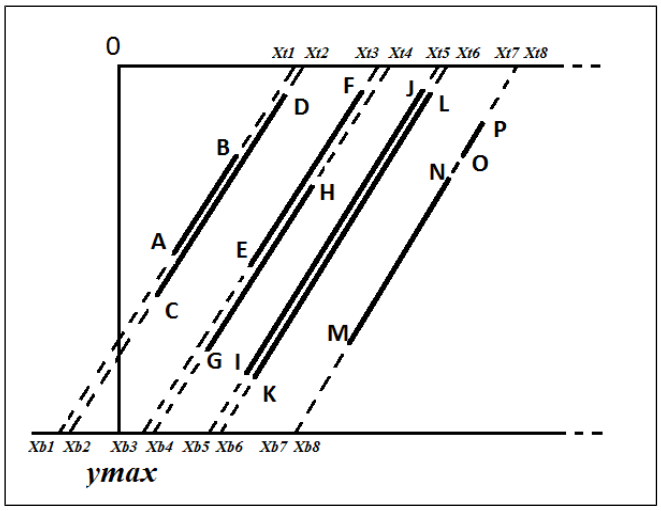
orientadas de cima para baixo, de 0 (zero) a $ymax$. A abscissa x representa as colunas e varia da esquerda para a direita, de 0 (zero) a $xmax$.

Figura 25: Exemplo de segmentos de retas presentes na ROI e que poderiam ser simplificados.



A Figura 26 ilustra o processo pelo qual se pode eliminar os segmentos indesejados. Para compará-los, faz-se necessário prolongar seus comprimentos até alcançarem as alturas 0 (zero) e $ymax$ como indicado na Figura 26. Definiu-se os valores de x no topo como xt, isto é, o valor de x quando y = 0. De forma análoga, os valores de x na base da imagem foram definidos como xb, isto é, o valor de x quando $y = ymax$.

Figura 26: Prolongamento dos segmentos entre a altura 0 (zero) e a altura $ymax$.



Para encontrar as ordenadas xb e xt de cada segmento, utilizou-se a equação da reta. O valor de m na Equação (25) corresponde à inclinação e deverá ser previamente calculada para todos os segmentos.

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \tag{25}$$

Manipulando a Equação (25), tem-se:

$$x_2 = x_1 + \frac{(y_2 - y_1)}{m} \tag{26}$$

Para se encontrar os valores de xb, substitui-se $x_2 = xb$ e $y_2 = ymax$ na Equação (26). Assim, encontra-se:

$$xb = x_1 + \frac{(ymax - y_1)}{m} \tag{27}$$

Da mesma forma, para xt, faz-se $x_2 = xt$ e y_2 = 0 na Equação (26). Desta forma:

$$xt = x_1 - \frac{y_1}{m} \tag{28}$$

De posse dos valores de xb e xt, pode-se aplicar uma rotina para comparar as distâncias na base e no topo. Considerando-se dois segmentos consecutivos quaisquer, i e j, como ilustrado na Figura 26, deve-se verificar se a seguinte condição é atendida:

$$\left(|xb_i - xb_j| < \varepsilon_b\right) \wedge \left(|xt_i - xt_j| < \varepsilon_t\right) \tag{29}$$

Em que:

- $i \neq j$;
- ε_b é um parâmetro que corresponde à distância limite entre os pontos xb_i e xb_j

que indique que os segmentos i e j sejam colineares;

- ε_t, analogamente, corresponde à distância limite entre os pontos xt_i e xt_j, para indicar colinearidade;
- O símbolo $\wedge$ representa a operação lógica AND.

Se a condição imposta pela Equação (29) for atendida, conclui-se que os segmentos são aproximadamente colineares. Porém, para eliminação ou junção destes, deve-se examinar a disposição entre os pontos que os formam. Se existir intersecção entre os segmentos, pode-se escolher os pontos mais externos para formação do novo segmento. Por exemplo, na Figura 26 tem-se:

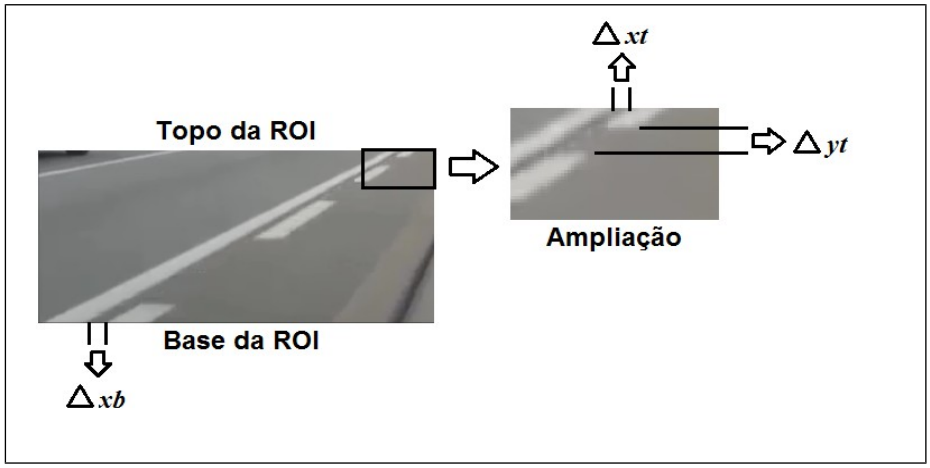
- $\overline{AB}\ U\ \overline{CD} = \overline{CD}$;
- $\overline{EF}\ U\ \overline{GH} = \overline{GF}$;
- $\overline{IJ}\ U\ \overline{KL} = \overline{IJ}$ ou $\overline{KL}$;
- Como não há intersecção entre os segmentos $\overline{MN}$ e $\overline{OP}$, deve-se encontrar as distâncias entre os pontos N e O e compará-la com um parâmetro ε_L (distância limite para se considerar a união entre dois segmentos de reta colineares). Caso a distância seja menor que ε_L, deve-se unir os dois segmentos formando $\overline{MP}$.

3.6.1 Escolha dos Limiares ε_b, ε_t e ε_L

Para se definir os valores dos limiares ε_b, ε_t e ε_L, deve-se considerar que as imagens analisadas encontram-se distorcidas pelo efeito da perspectiva. Objetos próximos à base apresentam dimensões superiores àqueles próximos ao topo.

Os valores adequados para os limiares ε_b e ε_t, devem ter como referência as menores distâncias obtidas pelas linhas de sinalização no topo, Δxt, e na base da ROI, Δxb. A Figura 27 ilustra como os mesmos são determinados. O parâmetro Δxb representa a menor distância entre linhas adjacentes na base da ROI. No detalhe ampliado da Figura 27, Δxt é a menor distância entre linhas adjacentes no topo da ROI.

Figura 27: Representação das menores distâncias encontradas na base da ROI, no topo da ROI e entre faixas tracejadas.



Para evitar que o método de eliminação altere ou anule linhas adjacentes válidas, propõe-se dividir as distâncias mínimas por um fator de dois. Conforme definido a seguir, os valores de ε_t e ε_b são dados por:

$$\varepsilon_t = \frac{\Delta xt}{2} \tag{30}$$

$$\varepsilon_b = \frac{\Delta xb}{2} \tag{31}$$

Desta forma, evita-se a eliminação de segmentos vizinhos realmente distintos, podendo utilizar os limiares ε_t e ε_b como critérios de proximidade entre segmentos adjacentes, como descrito na seção anterior. No detalhe ampliado da Figura 27, encontra-se o intervalo Δyt. Seguindo o mesmo raciocínio, ε_L pode ser definido como:

$$\varepsilon_L = \frac{\Delta yt}{2} \tag{32}$$

Os segmentos que formam sinalizações tracejadas não devem se fundir, sob pena de confundi-los com linhas contínuas. No topo da imagem da Figura 27, encontra-se uma representação do menor espaçamento entre sinalizações tracejadas. Assim, segmentos consecutivos podem ser recombinados, como descrito

na seção anterior, desde que a distância entre os mesmos não ultrapasse o valor de limiar ε_L definido pela Equação (32).

Os valores dos intervalos Δxb, Δxt e Δyt são parâmetros de entrada utilizados para calibrar o sistema. Mais detalhes sobre os parâmetros de entrada do sistema serão abordados na Seção 4.4.

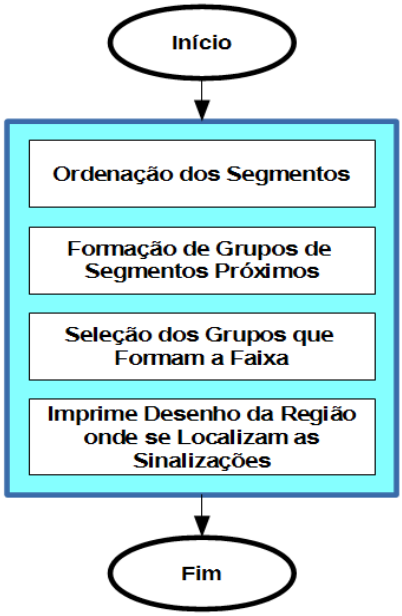
3.7 Identificação da Localização das Linhas de Sinalização Horizontal

Nesse estudo, identificar uma linha de sinalização horizontal é o processo de encontrar o posicionamento da mesma na imagem. Na Figura 28, identificam-se as linhas de sinalização à esquerda e à direita do veículo principal, automóvel no qual está instalada a câmera. As linhas à esquerda e à direita delimitam a faixa onde se encontra o veículo em questão.

Figura 28: A marcação na cor amarela é gerada pelo programa e indica a localização das linhas de sinalização horizontal.

Na Figura 28, a marcação na cor amarela é gerada pelo sistema. Para que o programa implementado consiga identificar as linhas corretas à esquerda e à direita, faz-se necessário realizar os métodos descritos na Figura 29.

Figura 29: Métodos utilizados para identificação das linhas de sinalização horizontal.



Primeiramente, é necessário ordenar os segmentos. A ordenação é realizada comparando os valores de xb dentre todos os segmentos de reta válidos. A Figura 30 ilustra um exemplo de linhas ordenadas.

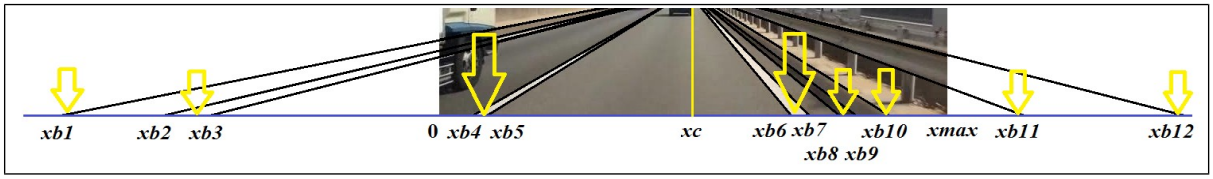
Figura 30: Exemplo de ordenação de linhas.

Após a ordenação, formam-se os grupos. Define-se um valor limite βxb como sendo a largura máxima para a formação de um grupo de segmentos. A partir de xb_1 até xb_n, calcula-se a diferença entre os valores dos xb consecutivos. Se os valores forem inferiores ao limite βxb, os segmentos pertencerão ao mesmo grupo. A condição para formação dos grupos pode ser visualizada na Inequação (33). As setas amarelas na Figura 29, indicam onde se localizam os grupos no exemplo dado.

$$xb_{n+1} - xb_n < \beta xb \tag{33}$$

Nos experimentos realizados, adotou-se βxb = 0,07 x *largura_faixa*. A variável *largura_faixa* é um parâmetro de entrada utilizado para calibrar o sistema. Mais detalhes sobre parâmetros de entrada serão abordados na Seção 4.4.

Figura 31: Formação dos grupos representados por setas amarelas. As setas maiores identificam o primeiro grupo à esquerda e o primeiro a direita do veículo principal. A reta amarela em *xc* divide a ROI em duas partes iguais.



O próximo passo é identificar o primeiro grupo à esquerda e o primeiro grupo à direita. Admitindo-se que a câmera esteja posicionada, aproximadamente, no centro do para-brisas e direcionada para o ponto de fuga, encontra-se o valor central da abscissa *xc* que divide a ROI em duas partes iguais pela Equação (34).

$$xc = \frac{xmax}{2} \tag{34}$$

Na Figura 31, o valor de *xc* está representado por uma reta amarela, marcando o centro da ROI. Ainda no exemplo da Figura 31, as setas amarelas maiores foram selecionadas como sinalização horizontal por estarem mais próximas do centro da imagem.

3.7.1 Medidas da Largura de Faixa

O sistema utiliza duas variáveis para contornar possíveis falhas de identificação das linhas de sinalização. Elas servem para determinar as larguras da faixa na qual se encontra o veículo. Essas variáveis são definidas como se segue:

- $largura\_B$ - Largura entre as linhas de sinalização medida na base da ROI. Sua representação pode ser visualizada pela maior linha azul da Figura 32;

$$largura\_B = DXB - EXB \tag{35}$$

- $largura\_T$ - Largura entre as linhas de sinalização medida no topo da ROI, representada pela menor linha azul da Figura 32;

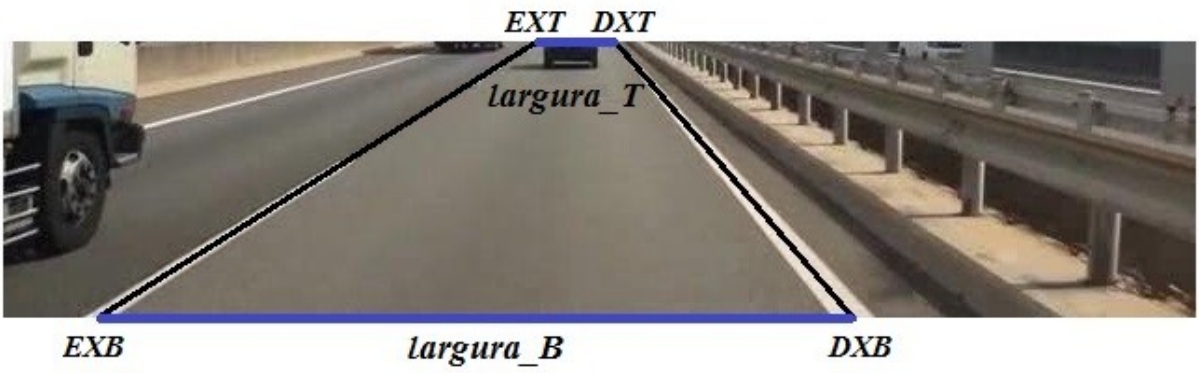
$$largura\_T = DXT - EXT \tag{36}$$

Em que:

- DXB – valor da abscissa X no ponto onde a linha direita encontra a base da ROI;
- EXB – valor da abscissa X no ponto onde a linha esquerda encontra a base da ROI;
- DXT – valor da abscissa X no ponto onde a linha direita encontra o topo da ROI;
- EXT – valor da abscissa X no ponto onde a linha esquerda encontra o topo da ROI;

A Figura 32 exibe a representação das medidas anteriores.

Figura 32: Representação das distâncias $largura\_T$ e $largura\_B$ na cor azul.



3.7.2 Autoajuste

A etapa de autoajuste é fundamental para o sistema proposto, pois o mesmo necessita de referências para realizar seus cálculos e descartar informações inconsistentes que possam prejudicar a localização das sinalizações. Inicialmente, o sistema encontra os valores de largura de faixa para poder operar de forma estável. Isto é, calcula as variáveis *largura_B* e *largura_T*. Após várias experimentações, percebeu-se que um período de autoajuste de três quadros consecutivos eram suficientes para determinação dessas variáveis. Isto é, se forem encontradas linhas laterais esquerda e direita por 3 quadros consecutivos, o sistema sairá do modo de autoajuste, utilizando as últimas larguras encontradas como padrão.

O autoajuste ocorrerá quando o sistema for inicializado ou, quando falhar na identificação das sinalizações, direita e esquerda, por cinco quadros consecutivos. Como exemplo, pode-se citar o caso em que o veículo esteja estacionado, não existindo linhas que sirvam de referências para que o sistema possa prosseguir com seus cálculos. Nesse caso, ele entra em modo de autoajuste e aguarda sucesso na identificação das linhas de ambos os lados por três quadros consecutivos.

3.7.3 Solucionando Problemas na Identificação de Linhas

No decorrer da análise dos quadros, percebeu-se que dependendo da intensidade da iluminação, se é dia ou noite, se está ou não chovendo e, principalmente, dependendo das condições de conservação da autovia, o sistema poderá falhar na localização da linha esquerda e/ou direita. Em alguns quadros, uma das linhas pode não ser detectada. Considerando-se que o sistema já se encontra ajustado, pode-se inferir a localização da linha faltante através do conhecimento da posição da linha detectada, somando-se ou subtraindo-se os valores padronizados de largura da faixa.

Por exemplo, caso não seja detectada a linha esquerda, calcula-se:

$$EXB = DXB - largura\_B \tag{37}$$

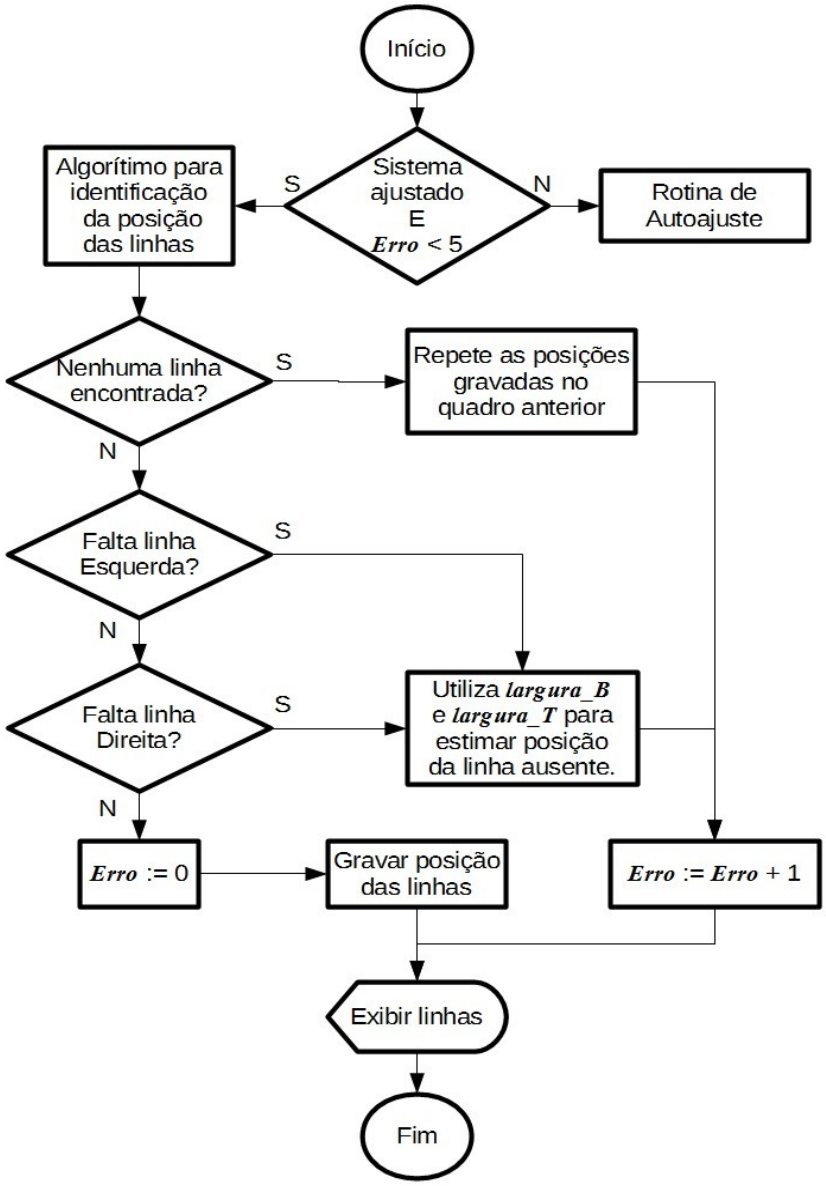
$$EXT = DXT - largura\_T \tag{38}$$

Nos experimentos realizados, percebeu-se que as posições das linhas identificadas varia muito pouco entre quadros consecutivos. Isso se deve à alta taxa de amostragem. Como exemplo, nos vídeos analisados nesse trabalho, as menores taxas são de 29 quadros por segundo. No exemplo anterior, de posse de EXB e EXT, calculados pelas Equações (37) e (38), pode-se localizar a linha de sinalização ausente.

Caso ambas as linhas não sejam detectadas, o sistema repetirá as posições encontradas no quadro anterior. Adotou-se a estratégia de se gravar as localizações encontradas à direita e à esquerda em cada quadro, que poderá servir para preencher falhas na localização da linha esquerda e direita no quadro seguinte. Se o problema persistir cinco vezes seguidas, o sistema entrará em modo de autoajuste eté detectar as linhas laterais por três quadros consecutivos.

A Figura 33 apresenta um fluxograma que retrata a forma como o sistema analisa cada quadro na busca da localização das sinalizações nas imagens. Primeiramente, verifica-se a necessidade de realizar o autoajuste. Isto é, quando se está no início da sequência ou quando ocorrerem 5 erros consecutivos. Ao se analisar o primeiro quadro, é possível que uma das linhas não seja identificada. Neste caso, ainda não existem posições de referência gravadas. Por isso o sistema entra em modo de autoajuste até detectar a sinalização esquerda e direita por três quadros consecutivos.

Figura 33: Fluxograma para resolução de problemas referentes à identificação das posições das sinalizações para cada quadro analisado.



Se o autoajuste não for necessário, aplicam-se os métodos descritos na Figura 29 para encontrar as linhas. Porém, nem sempre consegue-se identificar as linhas à esquerda ou à direita do veículo. A seguir, para cada dificuldade encontrada, aplica-se uma técnica diferente:

- Caso nenhuma linha seja identificada, o sistema repete as posições do quadro anterior, já que se encontra ajustado, e incrementa o contador $Erro$;
- Se faltar apenas a linha esquerda ou apenas a linha direita, o sistema tem

condições de estimar a linha ausente aplicando as Equações (37) e (38), pois são conhecidos os valores de $largura\_B$ e $largura\_T$. Da mesma forma, incrementa-se o contador $Erro$;

- Se foram identificadas as linhas da esquerda e da direita, no mesmo quadro, o sistema zera a variável $Erro$ e grava as posições das linhas caso o próximo quadro necessite dessa referência.

Após todas as ponderações, o sistema exibe as linhas de sinalização encontradas, procede a classificação e passa para a análise do próximo quadro. Na próxima seção, serão abordados os métodos desenvolvidos para a classificação das linhas de sinalização identificadas.

3.8 Classificação das Linhas de Sinalização Horizontal

O sistema realiza a classificação do grupo de linhas selecionados à esquerda e à direita. Analisando a disposição dos segmentos de reta e seus comprimentos, pode-se inferir sobre o tipo de sinalização que eles representam. As possibilidades são:

- Linha Simples:
  - Contínua;
  - Seccionada;
- Linha Dupla:
  - Contínua – Contínua;
  - Contínua – Seccionada;
  - Seccionada – Contínua;
  - Seccionada – Seccionada.
- Sem Linha.

O diagrama em blocos dos métodos utilizados na classificação pode ser visualizado na Figura 34. São utilizados 4 classificadores conforme ilustrado na Figura 35. A classificação C1 ocorre para identificar se a sinalização é simples (formada por uma linha), dupla (composta por duas linhas) ou inexistente. A classificação C2 ocorre apenas para linhas simples. Nela decide-se se a linha é contínua ou seccionada. As classificações C3 e C4 ocorrem quando a sinalização é dupla. C3 classifica a primeira parte em contínua ou seccionada e C4 classifica a segunda parte em contínua ou seccionada.

Figura 34: Diagrama de blocos dos métodos utilizados na classificação das linhas.

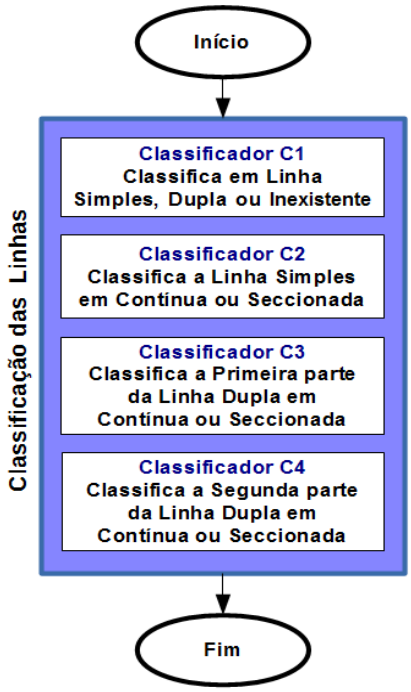
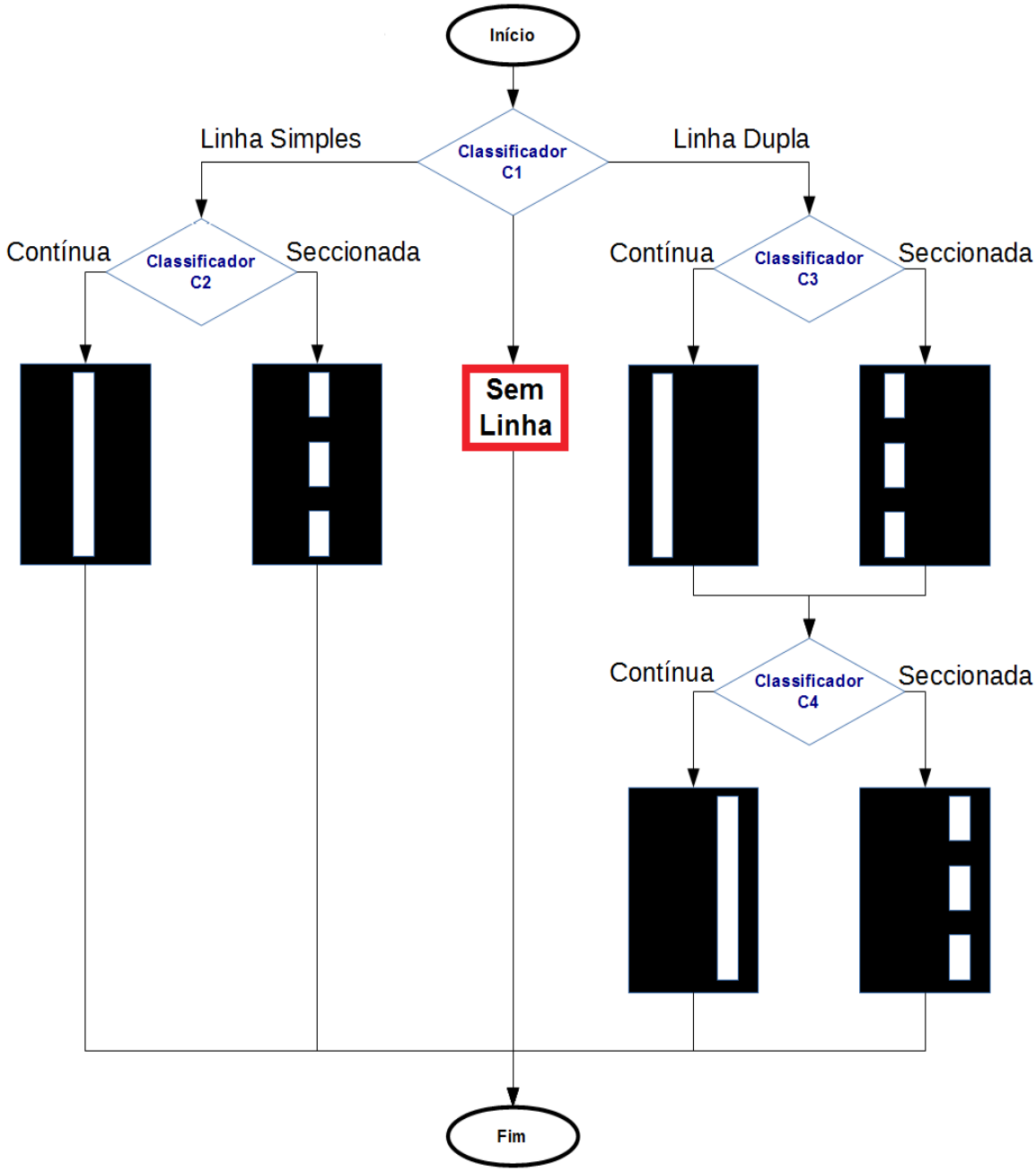


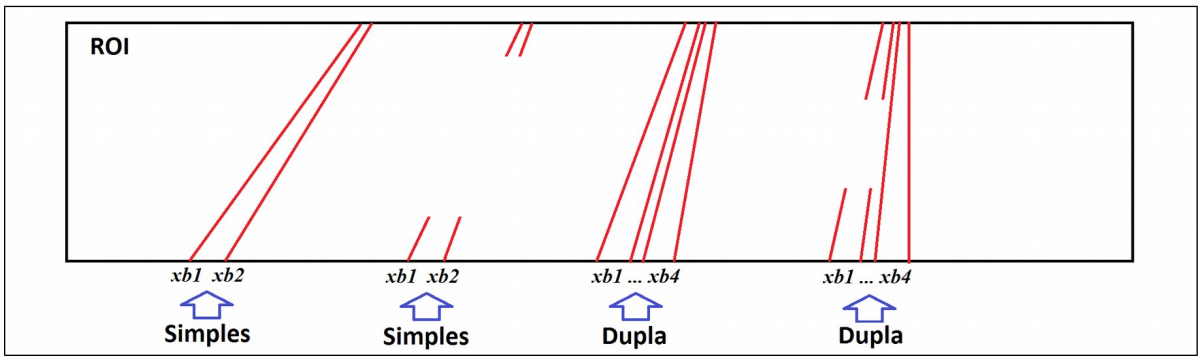
Figura 35: Fluxograma utilizado para a classificação das sinalizações horizontais.



3.8.1 Classificador C1

Inicialmente, o classificador C1 identifica se a sinalização horizontal é simples, dupla ou inexistente. Para isso, verifica-se a quantidade de segmentos presentes no grupo escolhido. Se existirem um ou dois segmentos, tem-se uma sinalização simples. Se existirem dois ou mais segmentos, tem-se uma sinalização dupla. A Figura 36 exemplifica a diferença entre as sinalizações simples e duplas.

Figura 36: Exemplos do uso do classificador C1 - Identificando se a sinalização é simples ou composta.

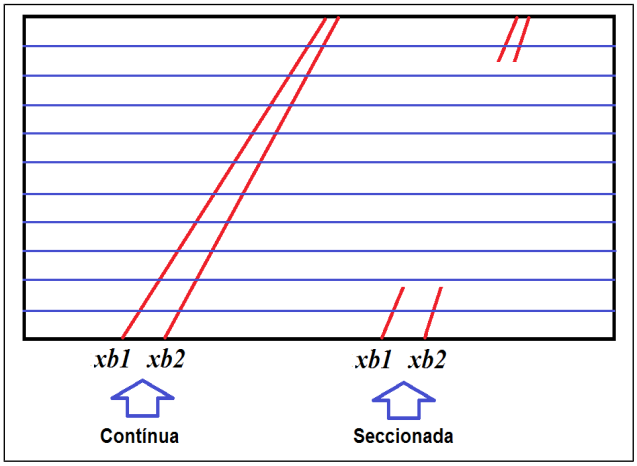


Se a quantidade de segmentos for igual a zero por 5 quadros consecutivos, o classificador C1 identifica a falta de linha, que pode ser causada por falhas na sinalização da pista ou por simples ausência de sinalização, como por exemplo, na passagem por um cruzamento.

Classificador C2

O classificador C2 é aplicado somente se a sinalização for simples. Ele identifica se a linha é contínua ou seccionada. Nesse caso, executa-se 10 (dez) cortes horizontais ao longo da altura da ROI e verifica se esses cortes interceptam os segmentos do grupo. Se pelo menos 80% dos cortes interceptarem os segmentos, classifica-se a sinalização horizontal como simples e contínua. Caso contrário será classificada como linha simples e seccionada. A Figura 37 exemplifica a metodologia aplicada na classificação C2.

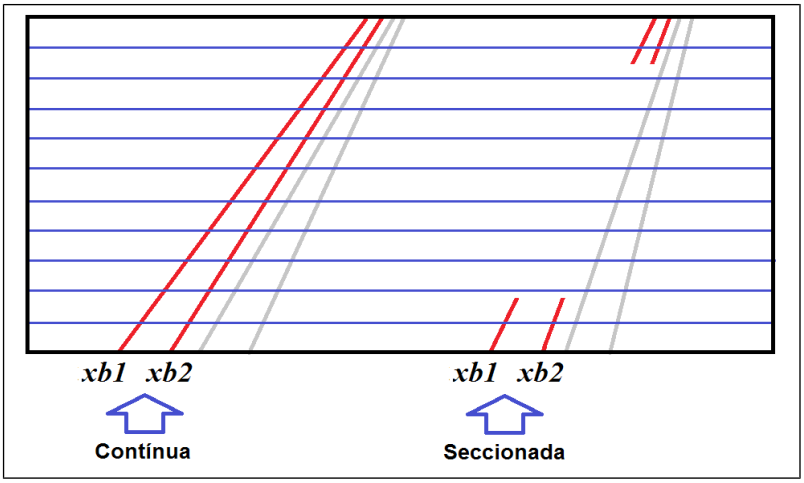
Figura 37: Classificador C2 - A ROI é dividida por 10 linhas. Verifica-se a quantidade de intersecções entre as linhas azuis e os segmentos vermelhos.



3.8.2 Classificador C3

O classificador C3 ocorre apenas quando a sinalização é dupla. Ele verifica se os primeiros segmentos, aqueles mais à esquerda, representam linhas contínuas ou seccionadas. Utiliza-se a mesma técnica aplicada em C2, porém apenas nos segmentos mais à esquerda, representados na cor vermelha, na Figura 38.

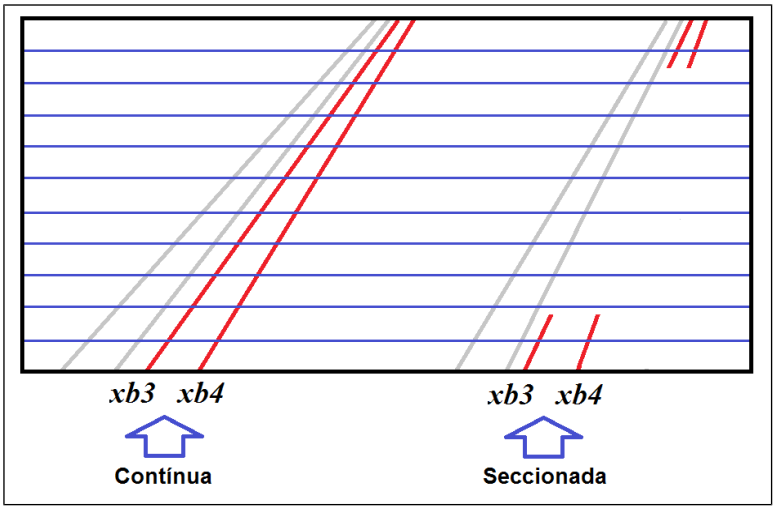
Figura 38: Classificador C3 - Utiliza a mesma metodologia do classificador C2, porém aplicada apenas à parte esquerda da sinalização dupla (cor vermelha).



3.8.3 Classificador C4

Esse classificador somente é aplicado às sinalizações duplas. Funciona de forma semelhante ao C3, porém aplicado aos últimos segmentos do grupo, aqueles mais à direita. A Figura 39 exemplifica a metodologia aplicada.

Figura 39: Classificador C4 - Verifica os segmentos que formam a última linha na sinalização dupla (em vermelho).



3.8.4 Estabilização das Classificações

O sistema classifica os tipos de linha, quadro a quadro. Em determinadas situações, condições ruins da tinta das sinalizações, buracos na pista, problemas causados por sombras ou má iluminação, podem interferir no processo de classificação, fazendo com que linhas contínuas pareçam seccionadas, ou linhas seccionadas pareçam inexistentes, em quadros aleatórios, causando ruído ao sistema. A Tabela 4 ilustra um exemplo de como problemas desse tipo podem prejudicar a classificação. Supõe-se que o veículo esteja trafegando por uma via com sinalização contínua. Os classificadores detectam essa condição e informam ao condutor o resultado encontrado.

Para o sistema, a letra “c” representa a linha contínua e a letra ”s”, a linha seccionada. Percebe-se que, nos quadros 5, 10 e 11, o classificador interpretou a linha como seccionada. Porém, deve-se recordar que, para vídeos com taxa de

exibição de 30 quados/s, seria impossível ocorrer uma mudança no tipo de faixa em um intervalo de tempo tão curto. Para amenizar esse tipo de problema, cada classificador compara o valor de classificação obtida no quadro atual com os valores obtidos nos últimos 20 quadros. Aplica-se o valor (+1) para classificações “c” e (-1) para “s”.

A Equação (39) exibe o cálculo da Estabilização, que corresponde ao somatório das 21 classificações obtidas recentemente. Em estatística descritiva, este cálculo é conhecido como “moda” e indica qual o valor que detém o maior número de observações. Nesse estudo, também pode ser considerado como um filtro passa-baixa, pois, elimina ruídos na saída dos classificadores. O valor de 21 classificações foi estipulado experimentalmente, pois reduziu o ruído do sistema sem causar grandes prejuízos à taxa de acurácia.

Tabela 4: Exemplo de ruído na classificação das linhas.

Quadro	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Saída(q)	c	c	c	c	s	c	c	c	c	s	s	c	c	c	c	c	c	c	c	c	c

$$Estabilização(q) = \sum_{n=0}^{20} saída(q-n) \tag{39}$$

Em que:

- q representa o número do quadro atual;
- Se $Estabilização(q) \geq 0$, tem-se que a sinalização é contínua;
- Se $Estabilização(q) < 0$, a sinalização é seccionada.

Uma observação importante deve ser feita. A cada mudança real de tipo de sinalização, o sistema apresentará um atraso na classificação correta que poderá durar por até 11 quadros. Para os vídeos estudados nesse trabalho, cuja taxa de exibição é de 29,97 quadros/s, corresponderia a um atraso de 0,367 segundos, que não prejudicaria a tomada de decisões do condutor do veículo.

No próximo capítulo, o sistema será testado em 7 experimentos. Cada um com um nível de dificuldade diferente. Serão exibidos os resultados obtidos e suas análises. Além disso, serão apresentadas as considerações finais e as sugestões para os próximos trabalhos.

4 Resultados Experimentais

Para analisar o sistema foram utilizadas cenas de vídeo disponíveis nas bases de dados de estudos semelhantes, compartilhados por Paula e Jung [10, 11] e outras cenas pesquisadas na internet. Além disso, foi necessário estabelecer as métricas necessárias para a análise do desempenho do sistema.

4.1 Métricas para Medição do Desempenho do Sistema

As métricas utilizadas para medir o desempenho se baseiam na análise da Matriz de Confusão ou Matriz de Erro. Segundo Deng et al. [37], a Matriz de Confusão contém informações sobre as classificações, reais e previstas, obtidas por um sistema de classificação. Ela possui duas dimensões. Uma dimensão é indexada pela classe atual do objeto e a outra é indexada pela classe que o classificador predisse. A Tabela 5 apresenta a estrutura básica de uma Matriz de Confusão para uma tarefa de classificação de múltiplas classes, A_1, A_2 até A_n. Na Matriz de Confusão, N_{ij} representa o número de amostras pertencente à classe A_i, mas classificada como classe A_j.

Tabela 5: Estrutura de uma Matriz de Confusão com múltiplas classes.

Matriz de Confusão		**Classe Prevista**		
		A_1	$\ldots A_j \ldots$	A_n
Classe Real	A_1	N_{11}	$\ldots N_{1j} \ldots$	N_{1n}
	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	A_i	N_{i1}	$\ldots N_{ij} \ldots$	N_{in}
	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	A_n	N_{n1}	$\ldots N_{nj} \ldots$	N_{nn}

Duas das métricas mais comuns para se medir a performance de um classificador são a acurácia e a taxa de reconhecimento. A acurácia corresponde ao valor percentual do total de previsões classificadas corretamente dividido pela soma de todas as amostras testadas. O cálculo da acurácia pode ser realizado pela Equação (40), que corresponde à soma dos valores da diagonal principal da matriz dividido pela soma de todos os valores.

$$Acurácia = \frac{\sum_{i=1}^{n} N_{ii}}{\sum_{i=1}^{n}\sum_{j=1}^{n} N_{ij}} \times 100 \tag{40}$$

A taxa de reconhecimento é uma medida de desempenho que corresponde ao percentual de amostras classificadas corretamente, de uma determinada classe, dividido pela soma dos valores da linha que pertence a essa mesma classe. Para cada classe, calcula-se a taxa de reconhecimento. A Equação (41) define o cálculo da taxa de reconhecimento para a classe A_i.

$$Taxa\,de\,Reconhecimento(A_i) = \frac{N_{ii}}{\sum_{k=1}^{n} N_{ik}} \times 100 \tag{41}$$

4.2 Programas Auxiliares

Para a verificação das respostas foi necessário criar dois programas auxiliares escritos na linguagem C++. O primeiro programa gera um arquivo chamado *padrao.txt,* contendo as classificações consideradas corretas para um determinado vídeo de entrada. Para gerar essa referência padrão, o operador analisa o vídeo quadro a quadro, atribuindo os códigos descritos na Tabela 6, correspondentes aos tipos de sinalizações. As informações são gravadas no arquivo *padrao.txt* na seguinte ordem:

- **Primeira coluna** – Número do quadro analisado;
- **Segunda coluna** – Classificação da sinalização esquerda;
- **Terceira coluna** – Classificação da sinalização direita;

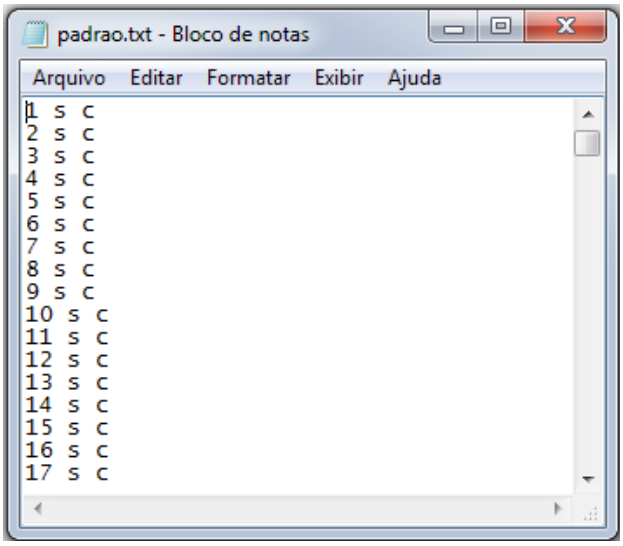
Na Tabela 6, encontram-se os códigos atribuídos a cada tipo de sinalização horizontal estudada. A Figura 40 mostra como são organizadas as informações dentro do arquivo *padrao.txt*.

Tabela 6: Códigos utilizados para os diversos tipos de sinalizações horizontais estudados.

Código	Significado
c	Contínua
s	Seccionada
cc	Contínua-contínua
cs	Contínua-seccionada
sc	Seccionada-contínua
ss	Seccionada-seccionada
-	Sem sinalização*

***** A falta de sinalização é representada pelo símbolo "**-**". Esse tipo de classificação ocorre quando não existem linhas a serem classificadas. Por exemplo, ao passar por um cruzamento ou quando as sinalizações não estão visíveis.

Figura 40: Exemplo de formatação das informações no arquivo *padrao.txt*.



Para cada quadro, o programa pede para o usuário analisar visualmente a imagem e informar o tipo de sinalização à esquerda e à direita. Para isso, o usuário deverá inserir os códigos apresentados anteriormente. Como exemplo, a primeira linha do arquivo *padrao.txt* da Figura 40 informa os códigos (1 s c), indicando que o quadro um é composto por uma sinalização simples à esquerda e outra contínua à direita.

Da mesma forma, o sistema principal gera um arquivo chamado *teste.txt* que contém todas as classificações de um determinado vídeo analisado. Com o intuito de verificar os resultados obtidos, foi desenvolvido um programa para realizar a comparação. Inicialmente, abrem-se os arquivos *teste.txt* e *padrao.txt.* Verificam-se, linha a linha, as classificações para a sinalização esquerda e para a direita. Por fim, são preenchidas as Matrizes de Confusão, calculando os valores de acurácia para a classificação média e as acurácias para cada classificador: C1, C2, C3 e C4. A Figura 41 mostra a tela de resultados gerado pelo programa de comparação.

Figura 41: Tela de saída gerada pelo programa de comparação aplicado ao Experimento 1.

```
Resultado

Acertos -> 200  200     = 400
Erros   -> 0    0       = 0

Total de testes realizados = 400

Percentuais Totais

Acuracia     =   100%
Erros        =   0%

========================= MATRIZ DE CONFUSAO ==============================

        c       s       cc      cs      sc      ss      (-)     Taxa de Rec.

c       0       0       0       0       0       0       0       --------------
s       0       400     0       0       0       0       0       100%
cc      0       0       0       0       0       0       0       --------------
cs      0       0       0       0       0       0       0       --------------
sc      0       0       0       0       0       0       0       --------------
ss      0       0       0       0       0       0       0       --------------
(-)     0       0       0       0       0       0       0       --------------

======= CLASSIFICAO C1  ==================================================

        simples dupla   (-)     Taxa de Rec.

simples 400     0       0       100%
dupla   0       0       0       --------------
(-)     0       0       0       --------------   Acuracia de C1 = 100%

======= CLASSIFICAO C2  ==================================================

        c       s       Taxa de Rec.

c       0       0       --------------
s       0       400     100%             Acuracia de C2 = 100%

======= CLASSIFICAO C3 - classifica a linha esquerda da dupla ===============

        cx      sx      Taxa de Rec.

cx      0       0       --------------
sx      0       0       --------------   Acuracia de C3 = Nao se aplica!

======= CLASSIFICAO C4 - classifica a linha direita da dupla ================

        xc      xs      Taxa de Rec.

xc      0       0       --------------
xs      0       0       --------------   Acuracia de C4 = Nao se aplica!

Pressione qualquer tecla para continuar. . .
```

4.3 Características dos Vídeos dos Experimentos

Nesse trabalho serão exibidos os resultados da análise de 7 vídeos, dos quais, 5 deles foram utilizados em outros estudos cujo foco é a classificação das linhas horizontais, possibilitando a comparação entre as acurácias alcançadas.

Os quatro primeiros vídeos: “clip_85.avi”, “clip_94.avi”, “clip_96.avi” e “clip_104.avi”, exibem rodovias brasileiras e foram utilizados nos estudos de Paula e Jung [11]. O quinto vídeo, “rs-040.mp4” foi utilizado nas pesquisas de Paula e Jung [10] e traz imagens de uma rodovia estadual brasileira, localizada no Rio Grande do Sul, a RS-040.

O sexto vídeo, “japao_dia.mp4”, foi adquirido pela internet e exibe imagens de uma rodovia japonesa. Por último, para verificar como o sistema se comporta na

análise de situações e ambientes desafiadores, analisa-se o vídeo “japao_noite.mp4”, que traz imagens noturnas e situações em que o veículo principal executa mudanças de faixa. O mesmo também foi filmado em uma rodovia japonesa. A Tabela 7 descreve as características de cada vídeo.

Tabela 7: Características dos vídeos utilizados nos experimentos de 1 a 7.

Nº	Vídeo	Resolução (*pixels* x *pixels*)	Taxa de exibição (quadros/s)	Quantidade de quadros	Sinalizações**	Trabalhos relacionados
1	clip_85.avi	1100 x 380*	29,97	200	**s**	[11]
2	clip_94.avi	1100 x 380*	29,97	1200	**s, c, sc, cs, -**	[11]
3	clip_96.avi	1100 x 380*	29,97	1000	**cs, c, cc**	[11]
4	clip_104.avi	1100 x 380*	29,97	1100	**sc, c, cc, cs**	[11]
5	rs-040.mp4	632 x 480	29,97	10797	**cc, c, cs, s, sc**	[10]
6	japao_dia.mp4	640 x 360	29	1000	**s, c**	-
7	japao_noite.mp4	640 x 360	29	1185	**s, sc, c**	-

***** A resolução original desses vídeos era de 1920 x 1080 *pixels*. A redução foi necessária para superar dificuldades técnicas explicadas no texto a seguir.
****** Nessa coluna aparecem os símbolos que representam as sinalizações como informado na Tabela 6.

Nos experimentos de número 1 a 4, pretende-se comparar os resultados desse estudo com aqueles obtidos por Paula e Jung [11]. Os vídeos originais utilizados para análise, possuem dimensão de 1920 x 1080 *pixels*. No entanto, limitações impostas pela versão 2.48 da biblioteca OpenCV não possibilitaram que imagens com resoluções de 1920 x 1080 *pixels* fossem exibidas utilizando a função “*imshow*()”. Apenas parte das imagens era exibida na saída do programa. Essa função não conseguia manipular imagens de tal ordem de grandeza. Acredita-se que essas limitações tenham sido superadas nas versões mais atuais da biblioteca.

Para contornar esse problema, resolveu-se gerar novos vídeos de entrada, a partir daqueles, selecionando-se a porção inferior dos mesmos com 1100 x 380 *pixels*. Assim, pôde-se utilizar a função *imshow*(). Essa solução manteve os tamanhos das linhas iguais aos dos vídeos originais, possibilitando a comparação do desempenho entre os mesmos. No entanto, impossibilitando a comparação com relação à velocidade de processamento, já que as dimensões não são as mesmas.

4.4 Parâmetros de Entrada do Sistema

Nesse estudo, utiliza-se diversos parâmetros para calibração do sistema. Alguns deles são fixos, tais como os limiares utilizados em Canny e os da Transformada Probabilística de Hough como visto no Capítulo 3. Porém identificou-se que alguns parâmetros dependem da qualidade do vídeo, das suas dimensões, da geometria das sinalizações que aparecem no mesmo e da posição e orientação da câmera que o filmou. Assim, o simples fato de modificar a orientação da câmera, como apontá-la mais para cima, para baixo, esquerda ou direita, pode prejudicar a localização do ponto de fuga, requerendo a entrada de novos parâmetros. No desenvolvimento desse sistema, até o momento, não foi escrito um código para realizar a autocalibração. Esse estudo limita-se na identificação de quais parâmetros influenciam a qualidade das respostas. Em estudos futuros, pretende-se trabalhar na autocalibração para melhorar o desempenho do sistema. Sendo assim, os parâmetros de entrada que variam de acordo com o vídeo estão dispostos a seguir:

- $y\_desloca$ – Esse parâmetro corresponde ao deslocamento vertical da ROI em *pixels*. Dependendo da posição da câmera, esse parâmetro também ajuda a evitar o aparecimento do capô do veículo na ROI;
- $altura$ – Corresponde à altura da ROI em *pixels*;
- $largura\_faixa$ – Representa a largura estimada da faixa em *pixels*;
- Δxb, Δxt e Δyt – Correspondem às distâncias utilizadas para calcular os valores dos limiares da Seção 3.6.1;
- $largura\_subgrupo$ – Limiar utilizado para identificar subgrupos de segmentos. Este parâmetro é útil no processo de identificação das sinalizações.

A Tabela 8 exibe os valores para os parâmetros de entrada utilizados nos experimentos das próximas seções. As unidades desses valores são expressas em *pixels*.

Tabela 8: Parâmetros de entrada utilizados nos Experimentos de 1 a 7.

Exp. nº	*y_desloca*	*altura*	*largura_faixa*	Δxb	Δxt	Δyt	*largura_subgrupo*
1	150	230	1000	5	3	5	15
2	150	230	1000	15	6	5	20
3	150	230	1000	15	5	5	25
4	150	230	1000	15	6	5	20
5	210	120	460	5	3	5	7
6	200	160	600	10	3	5	15
7	220	120	550	5	3	5	12

Obs.: Todos os valores são expressos em unidades de *pixels*.

4.5 Experimento 1

No Experimento 1, aplica-se o sistema ao vídeo “clip_85.avi”. O mesmo possui 200 quadros, com uma resolução de 1100 x 380 *pixels e* taxa de exibição de 29,97 quadros/s. Nesse vídeo, o veículo principal, aquele ao qual está fixada a câmera, trafega em linha reta, à luz do dia, em uma rodovia de mão única com três faixas e sem executar mudanças entre as faixas. As linhas de sinalização à esquerda e à direita do mesmo são simples e seccionadas. A Figura 42 exibe um dos quadros do “clip_85.avi”. Os parâmetros de entrada foram exibidos na Tabela 8.

Figura 42: Experimento 1 - Imagem de um dos quadros exibidos no vídeo "clip_85.avi".

Para cada quadro, analisam-se as sinalizações à esquerda e à direita, totalizando 400 classificações. A Tabela 9 exibe a Matriz de Confusão que informa a acurácia geral e as taxas de reconhecimento encontradas pelo sistema proposto. Da mesma forma, as Tabelas 10 e 11, exibem as Matrizes de Confusão para os classificadores C1 e C2, respectivamente. Como não foram detectadas linhas duplas, as classificações C3 e C4 não se aplicam.

Tabela 9: Matriz de Confusão do Experimento 1.

	c	**s**	**cc**	**cs**	**sc**	**ss**	**(-)**	**Taxa de Reconhecimento (%)**
c	0	0	0	0	0	0	0	----------
s	0	400	0	0	0	0	0	100
cc	0	0	0	0	0	0	0	----------
cs	0	0	0	0	0	0	0	----------
sc	0	0	0	0	0	0	0	----------
ss	0	0	0	0	0	0	0	----------
(-)	0	0	0	0	0	0	0	----------
								Acurácia Média = 100 %.

Tabela 10: Experimento 1 - Matriz de Confusão do Classificador C1.

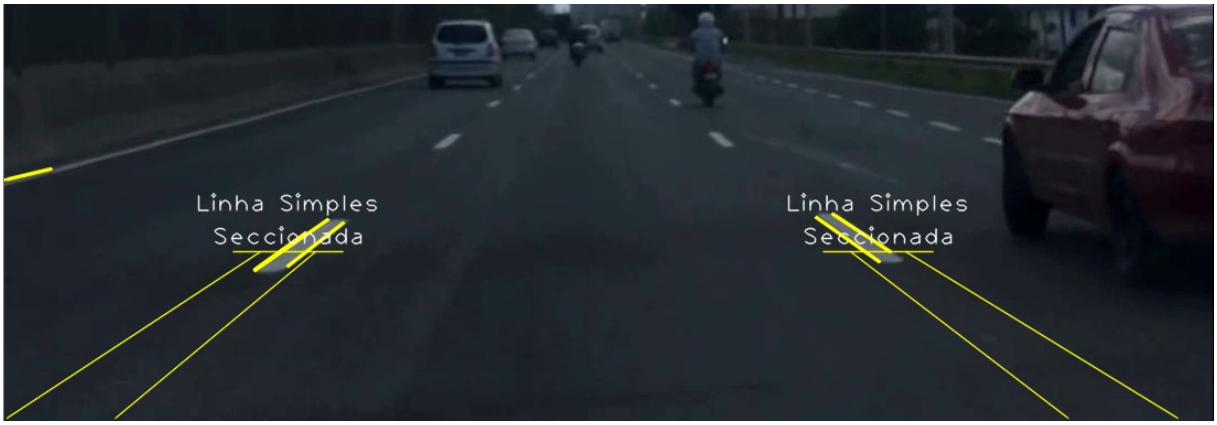
	Linha Simples	**Linha Dupla**	**Sem Linha (-)**	**Taxa de Reconhecimento (%)**
Linha Simples	400	0	0	100
Linha Dupla	0	0	0	----------
Sem Linha (-)	0	0	0	----------
				Acurácia do Classificador C1 = 100%.

Tabela 11: Experimento 1 - Matriz de Confusão do Classificador C2.

	Contínua	**Seccionada**	**Taxa de Reconhecimento (%)**
Contínua	0	0	----------
Seccionada	0	400	100
			Acurácia do Classificador C2 = 100%.

Analisando as Tabelas 9, 10 e 11, percebe-se que o sistema obteve 100% de acurácia na classificação C1, que identifica se a linha é dupla, simples ou inexistente. Da mesma forma, obteve 100% de acurácia na classificação C2, que identifica se uma linha simples é contínua ou seccionada. Como resultado, obteve-se 100% de acurácia na classificação média. Essa acurácia de 100% também foi alcançada nos estudos de Paula e Jung [11].

Figura 43: Experimento 1 - Exemplo de imagem de saída do sistema.



A Figura 43 exibe um exemplo de imagem de saída do sistema proposto. As linhas grossas amarelas indicam todos os segmentos da ROI selecionados após a filtragem pelo ponto de fuga. A identificação das localizações das sinalizações são indicadas por 3 linhas amarelas finas. Acima delas vem a descrição da classificação escrita em letras brancas. Observa-se que a presença dos veículos não interferiu na acurácia dos resultados.

4.6 Experimento 2

No Experimento 2, analisa-se o vídeo “clip_94.avi”. Esse vídeo possui 1200 quadros, resolução de 1100 x 380 *pixels* e taxa de exibição de 29,97 quadros/s. Nesse vídeo, o veículo principal trafega à luz do dia, em uma rodovia de mão dupla. Nesse experimento, surge a necessidade de se classificar a falta de linha, representada pelo símbolo (-) na Matriz de Confusão. Pois a pista apresenta alguns

trechos com sinalizações apagadas e outros sem sinalização. O sistema consegue compensar esses problemas em alguns quadros. Aumentando o grau de dificuldade, a rodovia apresenta curvas suaves à esquerda e à direita, tráfego à frente do veículo e em sentido contrário. Além disso, ocorrem cinco tipos diferentes de linhas de sinalização. A Figura 44 exibe um dos quadros desse vídeo de entrada.

Figura 44: Experimento 2 - Exemplo de imagem de entrada, exibindo curva suave à direita, tráfego no mesmo sentido do veículo e falta de sinalização horizontal à direita.

Foram realizadas 2400 classificações. A Tabela 12 exibe as taxas de reconhecimento encontradas para cada tipo de sinalização e a acurácia média para esse vídeo que foi de 88,42%. Na Tabela 13 verifica-se uma acurácia de 94,20% para o classificador C1. Percebe-se que a taxa de reconhecimento da falta de linha (-) ficou baixa. O motivo disso é que o sistema só considera falta de linha após esgotarem as chances de ser uma linha seccionada, pois a mesma possui intervalos sem tinta. Nesse caso, do total de 84 quadros sem linha, foram identificados apenas 30 e 54 foram classificados como linha simples, dos quais, 53 foram classificadas como linhas seccionadas. Da mesma forma, 21 linhas simples foram classificadas erroneamente como “sem linha” (Ver Tabela 12).

Tabela 12: Matriz de Confusão do Experimento 2.

	c	s	cc	cs	sc	ss	(-)	Taxa de Reconhecimento (%)
c	1030	33	0	0	12	0	0	95,814
s	47	199	0	0	0	6	21	72,8938
cc	0	0	0	0	0	0	0	----------
cs	47	0	21	415	9	13	0	82,1782
sc	1	1	3	4	448	6	0	96,7603
ss	0	0	0	0	0	0	0	----------
(-)	1	53	0	0	0	0	30	35.7143
								Acurácia Média = 88,42%.

Apesar da baixa taxa de reconhecimento, nesse caso, e a necessidade de aprimoramento, a detecção de falta de linha é uma funcionalidade a mais do sistema proposto. O vídeo “clip_94.avi” também foi analisado nos trabalhos de Paula e Jung [11] e obteve uma acurácia de 82,39%.

Tabela 13: Experimento 2 - Matriz de Confusão do Classificador C1.

	Linha Simples	Linha Dupla	Sem Linha (-)	Taxa de Reconhecimento (%)
Linha Simples	1309	18	21	97,1068
Linha Dupla	49	919	0	94,938
Sem Linha (-)	54	0	30	35,7143
				Acurácia do Classificador C1 = 94,08%

Com relação aos Classificadores C2, C3 e C4, observa-se que as acurácias alcançadas foram de 93,89%, 96,84% e 95,65%, respectivamente. Na Figura 45, pode-se perceber a classificação da linha dupla seccionada e contínua à esquerda e detecção da falta de linha, marcada como “Erro”, à direita.

Tabela 14: Experimento 2 - Matriz de Confusão do Classificador C2.

	Contínua	Seccionada	Taxa de Reconhecimento (%)
Contínua	1030	33	96,8956
Seccionada	47	199	80,8943
			Acurácia do Classificador C2 = 93,89%.

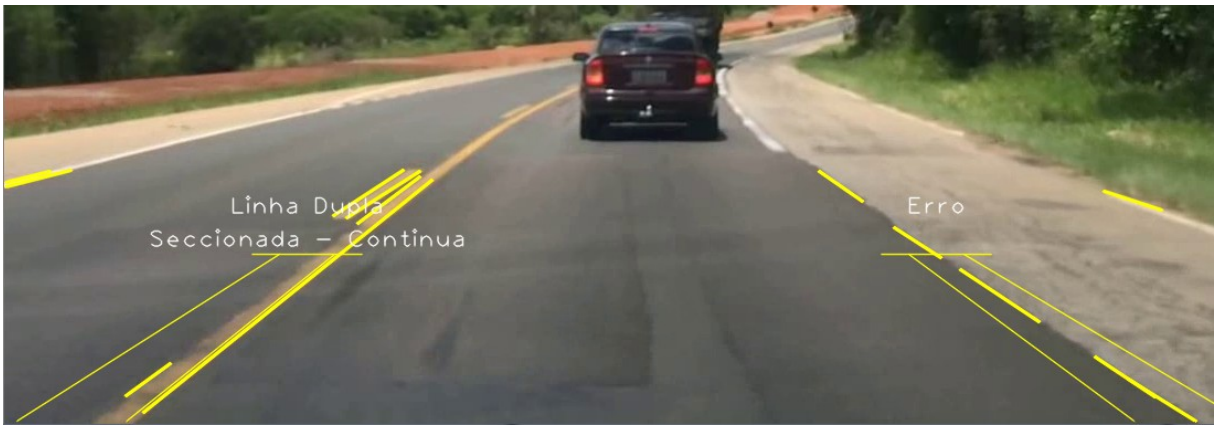
Tabela 15: Experimento 2 - Matriz de Confusão do Classificador C3.

	(Contínua) - (x)	(Seccionada) - (x)	Taxa de Reconhecimento (%)
(Contínua) - (x)	436	22	95,1965
(Seccionada) - (x)	7	454	98,4816
			Acurácia do Classificador C3 = 96,84%.

Tabela 16: Experimento 2 - Matriz de Confusão do Classificador C4.

	(x) - (Contínua)	(x) - (Seccionada)	Taxa de Reconhecimento (%)
(x) - (Contínua)	451	10	97,8308
(x) - (Seccionada)	30	428	93,4498
			Acurácia do Classificador C4 = 95,65%.

Figura 45: Experimento 2 - Exemplo de imagem de saída do sistema. À direita, percebe-se a detecção da falta de linha, indicada pela palavra “Erro”.



4.7 Experimento 3

O “clip_96.avi” é formado por 1000 quadros com resolução de 1100 x 380 *pixels* e taxa de exibição de 29,97 quadros/s. O veículo principal trafega em uma rodovia de mão dupla à luz do dia, apresentando uma curva suave à esquerda. Visualiza-se a presença de carros em ambos os sentidos. Ocorrem quatro formas de sinalizações. Um exemplo de imagem de entrada pode ser visto na Figura 46. Foram realizadas 2000 classificações. Na Tabela 17, verifica-se que a acurácia alcançada

foi de 98,38%, pouco abaixo da acurácia de 98,75% encontrada nos estudos de Paula e Jung [11].

Figura 46: Experimento 3 - Exemplo de imagem de entada.

Em geral, as taxas de reconhecimento e as acurácias foram altas, alcançando para C1, C2, C3 e C4, respectivamente, 99,95%, 100%, 97,9% e 98,6% de acurácia. A formação desses valores está detalhada nas Tabelas 18 a 21.

Tabela 17: Matriz de Confusão do Experimento 3.

	c	**s**	**cc**	**cs**	**sc**	**ss**	**(-)**	**Taxa de Reconhecimento (%)**
c	999	0	0	0	0	0	1	99,9
s	0	0	0	0	0	0	0	----------
cc	0	0	492	3	9	0	0	97,619
cs	0	0	8	247	0	0	0	96,8627
sc	0	0	9	3	229	0	0	95,0207
ss	0	0	0	0	0	0	0	----------
(-)	0	0	0	0	0	0	0	----------
								Acurácia Média = 98,35%.

Tabela 18: Experimento 3 - Matriz de Confusão do Classificador C1.

	Linha Simples	**Linha Dupla**	**Sem Linha (-)**	**Taxa de Reconhecimento (%)**
Linha Simples	999	0	1	99,9
Linha Dupla	0	1000	0	100
Sem Linha (-)	0	0	0	----------
				Acurácia do Classificador C1 = 99,95%.

Tabela 19: Experimento 3 - Matriz de Confusão do Classificador C2.

	Contínua	**Seccionada**	**Taxa de Reconhecimento (%)**
Contínua	999	0	100
Seccionada	0	0	----------
			Acurácia do Classificador C2 = 100%.

Tabela 20: Experimento 3 - Matriz de Confusão do Classificador C3.

	(Contínua) - (x)	**(Seccionada) - (x)**	**Taxa de Reconhecimento (%)**
(Contínua) - (x)	750	9	98,8142
(Seccionada) - (x)	12	229	95,0207
			Acurácia do Classificador C3 = 97,9%.

Tabela 21: Experimento 3 - Matriz de Confusão do Classificador C4.

	(x) - (Contínua)	**(x) - (Seccionada)**	**Taxa de Reconhecimento (%)**
(x) - (Contínua)	739	6	99,1946
(x) - (Seccionada)	8	247	96,8627
			Acurácia do Classificador C4 = 98,6%.

Uma das imagens de saída pode ser vista na Figura 47. Nela, percebe-se que as classificações funcionam corretamente mesmo na ocorrência de curvas suaves, pois as linhas próximas ao veículo principal podem ser aproximadas por linhas retas.

Figura 47: Experimento 3 - Exemplo de imagem de saída.

4.8 Experimento 4

Nesse experimento, aplica-se o sistema ao vídeo “clip_104.avi” composto por 1100 quadros, com resolução 1100 x 380 *pixels,* taxa de exibição de 29,97 quadros/s, totalizando 2200 classificações. O veículo principal trafega por uma rodovia de mão dupla, à luz do dia, com ocorrência de curvas suaves à direita, presença de veículos na mão, na contramão e ocorrência de sombras de árvores incidindo sobre trechos da sinalização direita. A Figura 48 ilustra um dos quadros de entrada no qual incidem sombras sobre a sinalização.

Figura 48: Experimento 4 - Exemplo de quadro de entrada. Observa-se à direita a presença de sombras sobre a sinalização.

Nos trabalhos de Paula e Jung [11] a análise desse vídeo obteve uma acurácia de 97,96%. Nesse estudo, observou-se na Tabela 22 uma acurácia de 95%. Já na Tabela 23, vê-se que o classificador C1 identificou corretamente as linhas duplas e simples, obtendo uma acurácia de 100%. No entanto, a ocorrência de sombras ocasionou a diminuição da acurácia no Classificador C2 para 92,82%. Ver Tabela 24. Pois 79 das 1100 linhas contínuas foram classificadas erroneamente como linhas seccionadas. Das Tabelas 25 e 26, tem-se que os classificadores C3 e C4 obtiveram acurácia de 97,36% e 99,82%, respectivamente. Um exemplo de classificação errada ocasionada pela ocorrência de sombras pode ser visualizado na Figura 49.

Tabela 22: Matriz de Confusão do Experimento 4.

	c	s	cc	cs	sc	ss	(-)	Taxa de Reconhecimento (%)
c	1021	79	0	0	0	0	0	92,8182
s	0	0	0	0	0	0	0	----------
cc	0	0	579	0	29	0	0	95,2303
cs	0	0	2	225	0	0	0	99,1189
sc	0	0	0	0	265	0	0	100
ss	0	0	0	0	0	0	0	----------
(-)	0	0	0	0	0	0	0	----------
								Acurácia Média = 95%.

Tabela 23: Experimento 4 - Matriz de Confusão do Classificador C1.

	Linha Simples	Linha Dupla	Sem Linha (-)	Taxa de Reconhecimento (%)
Linha Simples	1100	0	0	100
Linha Dupla	0	1100	0	100
Sem Linha (-)	0	0	0	----------
				Acurácia do Classificador C1 = 100%.

Tabela 24: Experimento 4 - Matriz de Confusão do Classificador C2.

	Contínua	Seccionada	Taxa de Reconhecimento (%)
Contínua	1021	79	92,8182
Seccionada	0	0	----------
			Acurácia do Classificador C2 = 92,82%.

Tabela 25: Experimento 4 - Matriz de Confusão do Classificador C3.

	(Contínua) - (x)	(Seccionada) - (x)	Taxa de Reconhecimento (%)
(Contínua) - (x)	806	29	96,5269
(Seccionada) - (x)	0	265	100
			Acurácia do Classificador C3 = 97,36%.

Tabela 26: Experimento 4 - Matriz de Confusão do Classificador C4.

	(x) - (Contínua)	(x) - (Seccionada)	Taxa de Reconhecimento (%)
(x) - (Contínua)	873	0	100
(x) - (Seccionada)	2	225	99,1189
			Acurácia do Classificador C4 = 99,82%.

Figura 49: Experimento 4 - Exemplo de imagem de saída. Observa-se a falha na classificação da sinalização direita causada pela incidência de sombras sobre o pavimento.

4.9 Experimento 5

Nesse experimento, analisa-se o vídeo “rs-040.mp4”, composto por 10797 quadros com resolução de 632 x 480 *pixels* e taxa de exibição de 29,97 quadros/s. O vídeo exibe um trecho da rodovia RS-040, do estado do Rio Grande do Sul. Esse é o experimento mais longo, totalizando 6 minutos e 19 segundos. A quantidade de classificações realizadas é de 21594. Esse vídeo foi escolhido para fazer uma comparação entre os resultados obtidos e os resultados de Paula e Jung [10]. As imagens mostram uma rodovia de mão dupla, à luz do dia, com a presença de veículos na mão e contramão, com curvas suaves à esquerda e à direita e a ocorrência de seis tipos de sinalização horizontal. Como desafio extra, a câmera está disposta de tal forma que o capô e o painel do veículo principal aparecem nas imagens. A Figura 50 mostra um dos quadros de entrada. As imagens do interior do veículo e de seu capô são eliminadas da análise deslocando-se a ROI para cima,

ajustando os valores dos parâmetros *y_desloca* e *altura* como realizado na Tabela 8 da Seção 4.4.

Na Tabela 27, tem-se que a acurácia média alcançada foi de 95,25%. Nos estudos de Paula e Jung [10], o valor de acurácia encontrado para esse vídeo foi de 85,42%. As Tabelas 28, 29, 30 e 31, mostram que os classificadores C1, C2, C3 e C4, obtiveram acurácia de 98,44%, 96,12%, 99,16% e 98,21, respectivamente.

A menor taxa de reconhecimento encontrada pelo sistema foi de 84,41% na identificação das linhas simples seccionadas. Dentre as 2720 linhas seccionadas presentes no pavimento, 424 delas foram classificadas erroneamente como contínuas. O motivo disso foi a ocorrência de linhas seccionadas de tamanhos menores e com intervalos muito próximos entre as mesmas. Essas linhas ocorreram do lado direito e indicam a possibilidade de entrada e saída de veículos à direita. A Figura 51 mostra o surgimento dessas linhas seccionadas curtas. Observa-se que as linhas seccionadas comuns localizadas à esquerda da imagem são maiores que aquelas localizadas à direita. Mesmo com essa dificuldade, o sistema conseguiu classificar corretamente as sinalizações desse quadro. Na mesma figura, a linha branca horizontal representa o limite inferior da ROI.

Figura 50: Experimento 5 - Exemplo de imagem de entrada. Percebe-se a exibição do capô e do painel do veículo principal.

Tabela 27: Matriz de Confusão do Experimento 5.

	c	s	cc	cs	sc	ss	(-)	Taxa de Reconhecimento (%)
c	8993	32	58	24	4	0	7	98,6291
s	424	2296	0	11	0	18	105	80,4485
cc	10	11	3201	23	22	12	13	97,2357
cs	75	1	116	5618	0	13	0	96,4795
sc	0	0	27	0	444	13	0	91,7355
ss	0	0	6	0	0	17	0	73,913
(-)	0	0	0	0	0	0	0	----------
								Acurácia Média = 95,25%.

Tabela 28: Experimento 5 - Matriz de Confusão do Classificador C1.

	Linha Simples	Linha Dupla	Sem Linha (-)	Taxa de Reconhecimento (%)
Linha Simples	11745	115	112	98,1039
Linha Dupla	97	9512	13	98,8568
Sem Linha (-)	0	0	0	----------
				Acurácia do Classificador C1 = 98,44%.

Tabela 29: Experimento 5 - Matriz de Confusão do Classificador C2.

	Contínua	Seccionada	Taxa de Reconhecimento (%)
Contínua	8993	32	99,6454
Seccionada	424	2296	84,4118
			Acurácia do Classificador C2 = 96,12%.

Tabela 30: Experimento 5 - Matriz de Confusão do Classificador C3.

	(Contínua) - (x)	(Seccionada) - (x)	Taxa de Reconhecimento (%)
(Contínua) - (x)	8958	47	99,4781
(Seccionada) - (x)	33	474	93,4911
			Acurácia do Classificador C3 = 99,16%.

Tabela 31: Experimento 5 - Matriz de Confusão do Classificador C4.

	(x) - (Contínua)	(x) - (Seccionada)	Taxa de Reconhecimento (%)
(x) - (Contínua)	3694	48	98,7173
(x) - (Seccionada)	122	5648	97,8856
			Acurácia do Classificador C4 = 98,21%.

Figura 51: Experimento 5 - Exemplo de imagem de saída. Observa-se a ocorrência de linhas seccionadas curtas à direita.

4.10 Experimento 6

Para o Experimento 6, selecionou-se o vídeo “japao_dia.mp4”, que é parte do vídeo original citado nos experimentos sobre de detecção de linha de Tan et al. [22] e obtido no seguinte endereço eletrônico: <http://youtu.be/qQHDvlsLu4c>. O trecho analisado possui 1000 quadros, com resolução de 640 x 360 *pixels* e taxa de exibição de 29 quadros/s. No total, tem-se 2000 classificações. O vídeo retrata uma rodovia japonesa de mão única, à luz do dia, composta por três faixas e sem curvas.

Nela ocorrem dois tipos de faixas, simples contínua e simples seccionada. O vídeo também exibe a presença de veículos trafegando ao lado e à frente do veículo principal. A Figura 52 ilustra um exemplo de imagem de entrada.

Figura 52: Experimento 6 - Exemplo de imagem de entrada.

Na Tabela 32, a acurácia média obtida foi de 98,55%. Das Tabelas 33 e 34, tem-se que as acurácias de C1 e de C2 são 99,7% e 98,85%, respectivamente. Por não existirem linhas duplas, as classificações C3 e C4 não se aplicam. Não há outros trabalhos sobre classificação de linhas discorrendo sobre esse vídeo que possam gerar comparações. A Figura 53 exibe um exemplo de imagem de saída do sistema.

Tabela 32: Matriz de Confusão do Experimento 6.

	c	**s**	**cc**	**cs**	**sc**	**ss**	**(-)**	**Taxa de Reconhecimento (%)**
c	978	19	0	0	0	0	3	97,8
s	4	993	0	0	0	0	3	99,3
cc	0	0	0	0	0	0	0	----------
cs	0	0	0	0	0	0	0	----------
sc	0	0	0	0	0	0	0	----------
ss	0	0	0	0	0	0	0	----------
(-)	0	0	0	0	0	0	0	----------

Acurácia Média = 98,55%.

Tabela 33: Experimento 6 - Matriz de Confusão do Classificador C1.

	Linha Simples	Linha Dupla	Sem Linha (-)	Taxa de Reconhecimento (%)
Linha Simples	1994	0	6	99,7
Linha Dupla	0	0	0	----------
Sem Linha (-)	0	0	0	----------
				Acurácia do Classificador C1 = 99,7%.

Tabela 34: Experimento 6 - Matriz de Confusão do Classificador C2.

	Contínua	Seccionada	Taxa de Reconhecimento (%)
Contínua	978	19	98,0943
Seccionada	4	993	99,5988
			Acurácia do Classificador C2 = 98,85%.

Figura 53: Experimento 6 - Exemplo de imagem de saída.

4.11 Experimento 7

Para concluir a fase de experimentos, escolheu-se um vídeo com características desafiadoras. O vídeo “japao_noite.mp4” traz imagens noturnas de uma rodovia japonesa com mão única e duas faixas. O veículo principal executa duas mudanças de faixa, ultrapassando e sendo ultrapassado por veículos. O vídeo possui 1185 quadros com resolução de 640 x 360 e taxa de exibição de 29 quadros/s, totalizando 2370 classificações. O mesmo é parte integrante do vídeo adquirido no endereço eletrônico <http://youtu.be/vkwJnOi1RZY>. O mesmo não foi utilizado em outros estudos até o momento. A Figura 54 exibe um dos quadros de entrada em que o veículo realiza uma mudança de faixa.

Figura 54: Experimento 7 - Exemplo de imagem de entrada. O veículo está realizando uma mudança de faixa.

Na Tabela 35, tem-se que a acurácia média alcançou 86,58%. As imagens com pouca luminosidade dificultaram a visualização das bordas, diminuindo a acurácia do sistema como um todo. A Tabela 36, traz a Matriz de Confusão do Classificador C1 com acurácia de 98,06%. Percebe-se que 21 linhas duplas foram

classificadas como linhas simples. Esse erros ocorreram porque a pouca luminosidade ocasionou a diminuição da quantidade de bordas detectáveis ao redor das sinalizações. Pelo mesmo motivo, 25 linhas simples foram classificadas como inexistentes. A Tabela 37 apresenta a Matriz de Confusão do Classificador C2 que teve sua acurácia diminuída para 87,93%, principalmente, pela baixa taxa de reconhecimento na classificação das linhas contínuas. Este também foi um problema causado pela pouca iluminação da cena, ocasionando a diminuição do tamanho de alguns segmentos de reta. As Tabelas 38 e 39 mostram que os classificadores C3 e C4 obtiveram 100% de acurácia. A Figura exibe um exemplo de imagem de saída. Percebe-se a presença de uma linha horizontal branca e fina. Ela indica a posição do limite inferior da ROI, excluindo da mesma a imagem do capô do veículo.

Tabela 35: Matriz de Confusão do Experimento 7.

	c	**s**	**cc**	**cs**	**sc**	**ss**	**(-)**	**Taxa de Reconhecimento (%)**
c	893	195	0	0	0	0	6	81,6271
s	77	1089	0	0	0	0	19	91,8987
cc	0	0	0	0	0	0	0	----------
cs	0	0	0	0	0	0	0	----------
sc	11	10	0	0	70	0	0	76,9231
ss	0	0	0	0	0	0	0	----------
(-)	0	0	0	0	0	0	0	----------
								Acurácia Média = 86,58%.

Tabela 36: Experimento 7 - Matriz de Confusão do Classificador C1.

	Linha Simples	**Linha Dupla**	**Sem Linha (-)**	**Taxa de Reconhecimento (%)**
Linha Simples	2254	0	25	98,903
Linha Dupla	21	70	0	76,9231
Sem Linha (-)	0	0	0	----------
				Acurácia do Classificador C1 = 98,06%.

Tabela 37: Experimento 7 - Matriz de Confusão do Classificador C2.

	Contínua	**Seccionada**	**Taxa de Reconhecimento (%)**
Contínua	893	195	82,0772
Seccionada	77	1089	93,3962
			Acurácia do Classificador C2 = 87,93%.

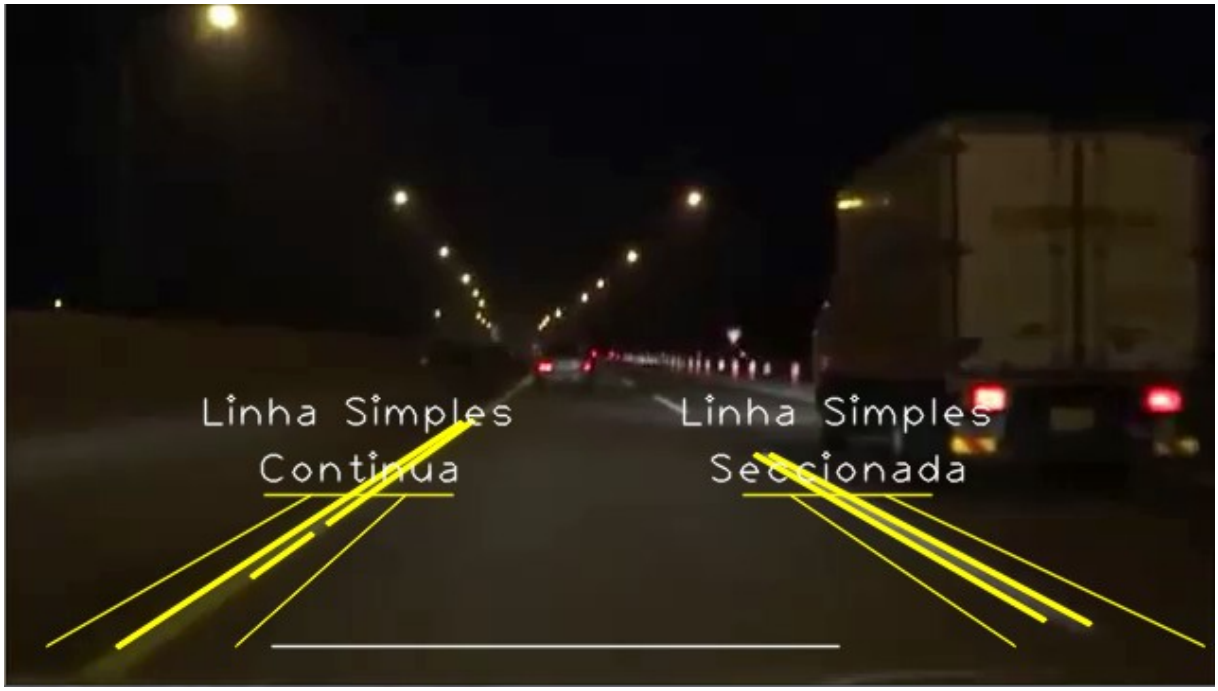
Tabela 38: Experimento 7 - Matriz de Confusão do Classificador C3.

	(Contínua) - (x)	**(Seccionada) - (x)**	**Taxa de Reconhecimento (%)**
(Contínua) - (x)	0	0	----------
(Seccionada) - (x)	0	70	100
			Acurácia do Classificador C3 = 100%.

Tabela 39: Experimento 7 - Matriz de Confusão do Classificador C4.

	(x) - (Contínua)	**(x) - (Seccionada)**	**Taxa de Reconhecimento (%)**
(x) - (Contínua)	70	0	100
(x) - (Seccionada)	0	0	----------
			Acurácia do Classificador C4 = 100%.

Figura 55: Experimento 7 - Exemplo de imagem de saída. A linha branca horizontal indica o limite inferior da ROI, excluindo o capô do veículo.



4.12 Resumo dos Resultados Experimentais

A Tabela 40 informa as acurácias alcançadas pelos experimentos numerados de 1 a 7 e faz uma comparação com as acurácias encontradas em outros estudos sobre classificação de sinalização horizontal. Assim como nos resultados de Paula e Jung [11], alcançou-se a máxima acurácia no Experimento 1. Também observou-se que, nos Experimentos 2 e 5, esse trabalho alcançou acurácias superiores àquelas encontradas por Paula e Jung [11]. Por outro lado, os Experimentos 3 e 4 obtiveram valores de acurácias 0,4% e 2,96% inferiores, quando também comparados com valores obtidos por Paula e Jung [11]. De forma geral, considerando todos os testes realizados, o sistema proposto obteve uma acurácia de 94,56%. Considerando, inclusive, o Experimento 7.

Tabela 40: Resumo das acurácias encontradas nesse estudo e comparação com outros trabalhos.

Nº	Vídeo	Resolução (*pixels*)	Sinalizações**	Acurácia desse estudo	Acurácia de outros trabalhos	Trabalhos relacionados
1	clip_85.avi	1100 x 380*	**s**	100%	100%	[11]
2	clip_94.avi	1100 x 380*	**s, c, sc, cs, -**	88,42%	82,39%	[11]
3	clip_96.avi	1100 x 380*	**cs, c, cc**	98,35%	98,75%	[11]
4	clip_104.avi	1100 x 380*	**sc, c, cc, cs**	95%	97,96%	[11]
5	rs-040.mp4	632 x 480	**cc, c, cs, s, sc**	95,25%	85,42%	[10]
6	japao_dia.mp4	640 x 360	**s, c**	98,55%	-	-
7	japao_noite.mp4	640 x 360	**s, sc, c**	86,58%	-	-

***** A resolução original desses vídeos era de 1920 x 1080 *pixels;*
****** Códigos que representam as sinalizações, como informado na Tabela 6.

Em relação ao Experimento 2, tem-se que a sua acurácia superou em 6,03% o resultado encontrado nos estudos de Paula e Jung [11]. Observando as caraterísticas do vídeo “clip_94.avi”, percebe-se que o mesmo apresenta trechos com interrupções de sinalizações e outros com sinalizações apagadas. Em Paula e Jung [11], classificam-se as sinalizações processando as densidades dos *pixels* próximos às linhas. Desta forma, levanta-se a hipótese na qual sequências de imagens com essas características são melhor classificadas pela análise das

bordas, tal como realizado pelo sistema proposto nesse trabalho. Além disso, consegue-se lidar melhor com as interrupções de sinalização, pois o Classificador C1 identifica sinalizações compostas por linhas simples, linhas duplas e ainda reconhece a falta de sinalizações.

4.12.1 Tempo de Processamento

A análise sobre o tempo médio de processamento foi realizada utilizando um computador com processador Intel(R) Core(TM) i5-4590, frequência de 3,30 GHz e 8 GB de memória RAM. A Tabela 41 relaciona os tempos médios encontrados para cada experimento. Da Tabela 7, tem-se que os vídeos de entrada possuem frequência de exibição de 29,97 quadros/s. Isto significa que, para haver um fluxo normal de imagens de saída, os quadros deverão ser exibidos a intervalos aproximados de 33,37 ms. Dentre as análises realizadas, o maior valor de tempo encontrado foi de 15,70 ms para o Experimento 3. Desta forma, percebe-se que o processamento ocorre em tempo real, não causando prejuízo à exibição das imagens capturados por câmeras de vídeo. Considerando o pior tempo médio encontrado nos experimentos, 15,70 ms, pode-se processar vídeos de até 63,69 quadros/s, sem ocorrerem atrasos.

Tabela 41: Média de tempo de processamento de um quadro para os experimentos de 1 a 7.

Nº	**Vídeo**	**Resolução (*pixels* x *pixels*)**	**Tempo médio gasto para o processamento de um quadro**
1	clip_85.avi	1100 x 380	15,26 ms
2	clip_94.avi	1100 x 380	15,68 ms
3	clip_96.avi	1100 x 380	15,70 ms
4	clip_104.avi	1100 x 380	15,68 ms
5	rs-040.mp4	632 x 480	15,60 ms
6	japao_dia.mp4	640 x 360	15,69 ms
7	japao_noite.mp4	640 x 360	15,68 ms

Comparando com outros trabalhos, o vídeo “rs-040.mp4” foi analisado por Paula e Jung [10], utilizando um *laptop* com processador Core 2 Duo, frequência de 2,53 GHz e 8 GB de memória RAM. O tempo médio gasto por quadro foi de 25,10 ms. Para os experimentos de 1 a 4, não foi possível comparar os tempos de processamento. Pois, como explicado no início desse capítulo, foi necessário modificar os tamanhos originais desses vídeos.

Nesse estudo, percebeu-se que a Transformada Probabilística de Hough requer boa parte do tempo de processamento de cada quadro. Por exemplo, para o Experimento 1, verificou-se que a TPH utilizou em média 38,3% do tempo total de processamento do sistema. Da mesma forma, para o Experimento 2, utilizou 39,8% do tempo de processamento. Nota-se que a TPH ainda é um processo demorado para se detectar segmentos de reta em imagens de vídeo. Nas seções 2.3 e 2.3.1, encontram-se maiores detalhes sobre a TPH. Em trabalhos futuros, propõe-se pesquisar ou desenvolver métodos mais rápidos para se detectar os segmentos de reta. Essa mudança poderia representar uma diminuição significativa no tempo médio de processamento de cada quadro.

4.12.2 Estabilidade do Sistema

Um valor alto para a acurácia é uma característica relevante para o programa. Porém, o trabalho proposto abre mão de um pouco da acurácia em prol da estabilidade do sistema. Tendo em vista que, para uma taxa de exibição de 29,97 quadros por segundo, a ocorrência de um erro dificilmente seria notada pelo usuário. Assim, o sistema privilegia a estabilidade, garantindo que as mudanças nos tipos de sinalizações ocorra de forma correta, sem ruído. Assim, utiliza-se o conhecimento a priori, isto é, uma combinação das 20 classificações anteriores com a atual. Desta forma, ocorrerá um atraso proposital na classificação quando houver uma mudança de sinalização que poderá chegar a 11 quadros. Assim, como já foi explicado na Seção 3.8.4, para uma taxa de exibição de 29,97 quadros/s o atraso poderia chegar a 11 x 33,37 ms, que corresponde a 0,367 segundos. Tempo muito curto para que o condutor perceba mudança na percepção de continuidade das imagens. No entanto, diminuirão as falhas causadas pela má conservação das sinalizações,

principalmente no que se refere às rodovias brasileiras. Diminuirá também a influência causada pelas condições insuficientes de iluminação e por eventos climáticos que possam diminuir a visibilidade da via, tal como a chuva.

Outra vantagem do sistema proposto é que a ROI possui tamanho reduzido e exibe o pavimento logo à frente do veículo. Assim, o programa é menos sensível às interferências causadas pela existência de automóveis próximos. Desta forma, como são tratadas apenas as bordas que convergem para o ponto de fuga, os veículos à frente do veículo principal raramente contribuem com segmentos de linha.

5 Considerações Finais

Os experimentos realizados mostraram que o sistema proposto consegue identificar e classificar corretamente as linhas de sinalização horizontal com acurácias que variam desde 86,58%, para sequências de imagens que apresentam condições desafiadoras, até 100%, para imagens de autovias bem iluminadas e com sinalização bem conservada. Considerando todos os 7 experimentos juntos, obteve-se uma acurácia média de 94,56%. Quanto melhor for a qualidade das imagens a serem processadas, quanto melhor forem as condições de conservação das autovias e quanto melhor forem as condições climáticas, melhores serão os resultados alcançados, podendo chegar a 100% de acurácia, como foi o caso do Experimento 1.

De acordo com os resultados obtidos, o sistema consegue trabalhar em tempo real sem causar atrasos. Isto é, o mesmo pode ser utilizado para analisar imagens obtidas por uma câmera fixada no para-brisa do veículo.

Como contribuições, esse trabalho apresenta métodos para a otimização/eliminação dos segmentos de reta duplicadas e/ou colineares, métodos para a localização do ponto de fuga de uma imagem utilizando boxplot, o desenvolvimento de métodos para a identificação do posicionamento das linhas de sinalização e métodos para a realização da classificação dessas linhas. Outra inovação foi a classificação da falta de sinalização (-), algo não verificado em outros estudos e que contribuiu para elevar as acurácias dos experimentos realizados.

Faz-se necessário elucidar uma questão sobre segurança. Não é recomendado que os condutores de veículos desviem seu olhar das estradas. Desta forma, a visualização das imagens exibidas pelo sistema serve apenas como apoio

didático. Representa apenas uma comprovação do seu funcionamento. Em situações reais, sinais sonoros devem ser emitidos para alertar sobre situações de saída indevida de faixa. Assim, a retirada de uma tela de vídeo do sistema, diminuiria o seu custo financeiro e o deixaria mais rápido. Pois, eliminaria os códigos necessários para gerar as imagens de saída. Para estudos futuros propõe-se as seguintes melhorias:

- Implementar a calibração automática do sistema para obtenção dos parâmetros necessários para o seu bom funcionamento, como descrito na Seção 4.4;
- Aperfeiçoamento do sistema ao passar por curvas fechadas. Curvas suaves já são detectadas. A utilização de duas ROIs, uma de altura maior focalizando o pavimento próximo ao veículo e outra com altura menor, exibindo a porção mais distante do pavimento proporcionaria o uso do sistema em curvas fechadas;
- Acrescentar filtros de textura para aprimorar a percepção do pavimento da autovia;
- Pesquisar ou desenvolver métodos mais rápidos que a Transformada Probabilística de Hough para identificar segmentos de reta, diminuindo o tempo médio de processamento para cada quadro do sistema;
- Com o objetivo de melhorar as taxas de reconhecimento e as acurácias desse sistema, que trabalha baseado em análise de bordas, pode-se adicionar algumas das técnicas desenvolvidas por Paula e Jung [11]. Dessa forma, um trabalho poderá suprir as deficiências do outro, formando um sistema mais robusto;
- Embarcar o sistema em *hardwares* que suportem a biblioteca OpenCV, tais como dispositivos Android, BeagleBone, Galileo, Edison, entre outros.

Esse trabalho pode ser adaptado para projetos de carros autônomos, visão de robô, para auxílio à locomoção de deficientes visuais e na automação de cadeiras de rodas motorizadas. Pode também ser utilizado na fiscalização de trânsito, servindo ou não como prova de infração por ultrapassagens em locais proibidos. Em suma, pode ajudar a salvar vidas ao diminuir as possibilidades de ocorrerem colisões frontais.

Referências Bibliográficas

[1] WHO, “Global status report on road safety 2013: supporting a decade of action.”, 2013.

[2] DetranRS, “OMS divulga Relatório Global de Segurança no Trânsito 2013”, 2013. [Online]. Disponível em: <http://www.detran.rs.gov.br/decadars/?p=1655>. [Acessado: 20-mar-2015].

[3] DPRF, “Principais Tipos de Acidentes Ocorridos no Primeiro Semestre de 2013”, 2013. [Online]. Disponível em: <http://www.dprf.info/Semestre/1/2013/Tipos>. [Acessado: 20-mar-2015].

[4] DPRF, “Principais Causas de Acidentes Ocorridos no Primeiro Semestre de 2013”, 2013. [Online]. Disponível em: <http://www.dprf.info/Semestre/1/2013/Causas>. [Acessado: 20-mar-2015].

[5] P. E. Ross, “Thus Spoke the Autobahn”, *IEEE Spectr.*, p. 50–53, jan. 2015.

[6] Wikipedia, “Advanced driver assistance systems”, *Wikipedia*, 2015. [Online]. Disponível em: <http://en.wikipedia.org/wiki/Advanced_driver_assistance_systems>. [Acessado: 28-mar-2015].

[7] “ADAS Advanced Driver Assistance Sytems – Definition AUTO Connected Car”, *Auto Conected Car News*, 2014. [Online]. Disponível em: <http://www.autoconnectedcar.com/?s=ADAS+definition>. [Acessado: 28-mar-2015].

[8] I. Riches, “Strategy Analytics”, 2014. [Online]. Disponível em: <http://standards.ieee.org/events/automotive/2014/00_Automotive_Ethernet_Market_Growth_Outlook.pdf>. [Acessado: 28-mar-2015].

[9] OpenCV.org, “OpenCV”. [Online]. Disponível em: <http://opencv.org/>. [Acessado: 28-mar-2015].

[10] M. B. De Paula e C. R. Jung, “Real-time detection and classification of road lane markings”, in *Brazilian Symposium of Computer Graphic and Image Processing*, 2013, p. 83–90.

[11] M. B. de Paula e C. R. Jung, “Automatic Detection and Classification of Road Lane Markings Using Onboard Vehicular Cameras”, *IEEE Trans. Intell. Transp. Syst.*, vol. 16, nº 6, p. 3160–3169, 2015.

[12] D. Ding, J. Yoo, J. Jung, S. Jin, e S. Kwon, “Various lane marking detection and classification for vision-based navigation system?”, in *IEEE International Conference on Consumer Electronics (ICCE)*, 2015, p. 491–492.

[13] W. Li, X. Gong, Y. Wang, e P. Liu, “A lane marking detection and tracking algorithm based on sub-regions”, in *International Conference on Informative and Cybernetics for Computational Social Systems (ICCSS)*, 2014, p. 68–73.

[14] Y. Dong, J. Xiong, L. Li, e J. Yang, “Robust lane detection and tracking for lane departure warning”, in *International Conference on Computational Problem-Solving (ICCP)*, 2012, p. 461–464.

[15] J.-H. Cho, E. Tsogtbaatar, S.-H. Kim, Y.-M. Jang, P.-M.-L. Nguyen, e S.-B. Cho, “Improved lane detection system using Hough transform with super-resolution reconstruction algorithm and multi-ROI”, in *International Conference on Electronics, Information and Communications (ICEIC)*, 2014, p. 1–4.

[16] C. Y. Low, H. Zamzuri, e S. A. Mazlan, “Simple robust road lane detection algorithm”, in *5th International Conference on Intelligent and Advanced Systems (ICIAS)*, 2014, p. 1–4.

[17] Y. Jun, T. Shuming, P. Xiuqin, e Z. Hong, “A robust vanishing point estimation method for lane detection”, in *33rd Chinese Control Conference (CCC)*, 2014, p. 4887–4892.

[18] D. Seo e K. Jo, “Inverse Perspective Mapping based road curvature estimation”, in *IEEE/SICE International Symposium on System Integration (SII)*, 2014, p. 480–483.

[19] V. Gaikwad e S. Lokhande, “Lane Departure Identification for Advanced Driver Assistance”, *IEEE Trans. Intell. Transp. Syst.*, vol. 16, nº 2, p. 910–918, 2015.

[20] G. Liu, S. Li, e W. Liu, “Lane detection algorithm based on local feature extraction”, in *Chinese Automation Congress (CAC)*, 2013, p. 59–64.

[21] J. Zhao, B. Xie, e X. Huang, “Real-time lane departure and front collision warning system on an FPGA”, in *IEEE High Performance Extreme Computing Conference (HPEC)*, 2014, p. 1–5.

[22] H. Tan, Y. Zhou, Y. Zhu, D. Yao, e K. Li, “A novel curve lane detection based on Improved River Flow and RANSAC”, in *IEEE 17th International Conference on Intelligent Transportation Systems (ITSC)*, 2014, p. 133–138.

[23] T. Tan, S. Yin, P. Ouyang, L. Liu, e S. Wei, “Efficient lane detection system based on monocular camera”, in *IEEE International Conference on Consumer Electronics (ICCE)*, 2015, p. 202–203.

[24] S.-Y. Jun e J.-H. Yoo, “Lane detection and curvature estimation based on motion template”, in *7th International Conference on Computing and Convergence Technology (ICCCT)*, 2012, p. 789–793.

[25] J. Han, Y. Dong, H. Kim, e S.-K. Park, “A new lane detection method based on vanishing point estimation with probabilistic voting”, in *IEEE International Conference on Consumer Electronics (ICCE)*, 2015, p. 204–205.

[26] CONTRAN, “Manual Brasileiro de Sinalização de Trânsito - Volume IV - Sinalização Horizontal”, 2007. [Online]. Disponível em: <http://www.denatran.gov.br/publicacoes/download/manual_horizontal_resolucao_236.pdf>. [Acessado: 28-mar-2015].

[27] R. C. Gonzalez e R. E. Woods, *Processamento Digital de Imagens*, 3. ed. São Paulo: Pearson Prentice Hall, 2010.

[28] OpenCV.org, “Canny Edge Detector — OpenCV 2.4.11.0 documentation”. [Online]. Disponível em: <http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html?highlight=canny>. [Acessado: 07-maio-2015].

[29] T. Jamundá, “Reconhecimento de Formas - A Transformada de Hough”, 2000. [Online]. Disponível em: <http://www.inf.ufsc.br/~visao/2000/Hough/>. [Acessado: 11-abr-2015].

[30] OpenCV.org, “Hough Line Transform”. [Online]. Disponível em: <http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html>. [Acessado: 08-jun-2016].

[31] “Licença BSD – Wikipédia, a enciclopédia livre”. [Online]. Disponível em: <http://pt.wikipedia.org/wiki/Licen%C3%A7a_BSD>. [Acessado: 07-maio-2015].

[32] “OpenCV – Wikipédia, a enciclopédia livre”. [Online]. Disponível em: <http://pt.wikipedia.org/wiki/OpenCV>. [Acessado: 07-maio-2015].

[33] N. Otsu, “A threshold selection method from gray-level histograms”, *IEEE Trans. Syst. Man. Cybern.*, vol. 9, nº 1, p. 62–66, 1979.

[34] R. Klette, “SET 10 - iROADS Dataset (Intercity Roads and Adverse Driving Conditions)”, 2013. [Online]. Disponível em: <http://ccv.wordpress.fos.auckland.ac.nz/eisats/set-10/>. [Acessado: 10-fev-2015].

[35] A. M. L. de Farias, “Boxplot”, *Uniersidade Federal Fluminense - Instituto de Matemática*. [Online]. Disponível em: <http://www.uff.br/cdme/conheceboxplot/conheceboxplot-html/boxplot.pdf>. [Acessado: 09-maio-2016].

[36] F. A. Oliveira, C. D. M. Regis, e S. E. N. Correia, “Aplicação de Métodos Geométricos para a Otimização da Quantidade de Segmentos de Reta na Identificação da Sinalização Horizontal em Autovias”, in *XII Simpósio Brasileiro de Automação Inteligente (SBAI)*, 2015.

[37] X. Deng, Q. Liu, Y. Deng, e S. Mahadevan, “An improved method to construct basic probability assignment based on the confusion matrix for classification problem”, *Inf. Sci. (Ny).*, vol. 340–341, p. 250–261, maio 2016.

[38] OpenCV.org, “Feature Detection - Canny”. [Online]. Disponível em: <http://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=canny#canny>. [Acessado: 02-fev-2016].

[39] OpenCV.org, “Feature Detection - HoughLinesP”. [Online]. Disponível em: <http://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=canny#houghlinesp>. [Acessado: 02-fev-2016].

APÊNDICE A – Função *Canny*()

A seguir, apresenta-se a função *Canny*() escrita na linguagem C++, contida na biblioteca OpenCV e utilizada no desenvolvimento do sistema proposto. Maiores detalhes podem ser vistos em opencv.org [38].

C++: void *Canny*(InputArray ***image***, OutputArray ***edges***, double ***threshold1***, double ***threshold2***, int ***apertureSize*** = 3, bool ***L2gradient*** = false)

Com os seguintes atributos:

- ***image*** – Imagem de entrada com 8 bits de canal simples;
- ***edges*** – Bordas de saída do mesmo tamanho e tipo da imagem de entrada;
- ***threshold1*** – Primeiro *threshold* para procedimentos de histerese;
- ***threshold2*** – Segundo *threshold* para procedimentos de histerese;
- ***apertureSize*** – Tamanho da abertura para o operador de “Sobel”;
- ***L2gradiente*** – um *flag* indicando se deverá ser utiliza uma norma mais precisa para se calcular a magnitude do gradiente da imagem.

No sistema, a linha de comando ficou configurada da seguinte forma:

Canny(*img_blur, dst*, 20, 60, 3);

Em que:

- ***img_blur*** - é o nome dado à imagem de entrada;
- ***dst*** – é o nome da imagem de destino;
- **20** – valor do limiar ***threshold1;***
- **60** – Valor do limiar ***threshold2;***
- **3 –** Tamanho da matriz para aplicação de Sobel;

OBS.: O atributo ***L2gradiente*** é opcional e não foi utilizado.

APÊNDICE B – Função *HoughLinesP*()

A função *HoughLinesP*() escrita na linguagem C++ que compõe a biblioteca OpenCV implementa a Transformada Probabilística de Hough para detecção de linhas retas e necessita dos seguintes parâmetros de entrada [39]:

C++: void *HoughLinesP*(InputArray ***image***, OutputArray ***lines***, double ***rho***, double ***theta***, int ***threshold***, double ***minLineLength*** = 0, double ***maxLineGap*** = 0)

Com os seguintes atributos:

- ***image*** – Imagem de entrada com 8 bits de canal simples. A imagem será modificada pela função;
- ***lines*** – Vetor de saída das linhas encontradas;
- ***rho*** – Distância de resolução do acumulador em *pixels*;
- ***theta*** – Ângulo de resolução do acumulador em radianos;
- ***threshold*** – Valor de limiar do acumulador. Só serão selecionadas linhas que possuírem quantidades de votos maiores que esse limiar;
- ***minLineLength*** – Tamanho mínimo para uma linha. Segmentos de linha menores que atributo serão rejeitados;
- ***maxLineGap*** – Distância máxima permitida entre pontos de uma mesma linha. Valores menores que esse, farão com que os pontos fiquem vinculados.

No sistema, a linha de comando ficou configurada da seguinte forma:

HoughLinesP(*dst*, *lines*, 1, CV_PI/180, 20, 30, 5);

Em que:

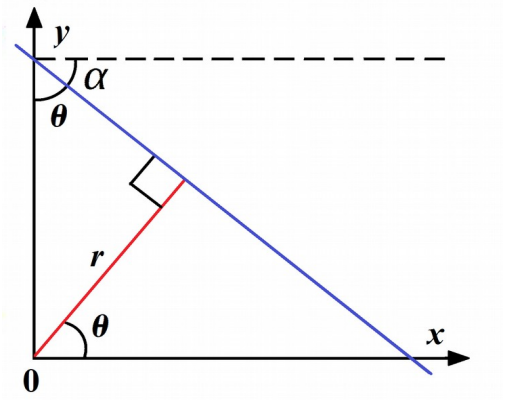
- ***dst*** - é o nome dado à imagem a ser modificada;
- ***lines*** – vetor de saída dos segmentos encontrados***;***
- **1** – valor de ***rho;***
- **CV_PI/180 –** ângulo de resolução ***theta***;
- **20** – Quantidade mínima de votos;
- **30** – Tamanho mínimo dos segmentos em *pixels*;
- **5** – Distância máxima entre pontos consecutivos.

APÊNDICE C – Demonstração da forma polar da Transformada de Hough

Nesse apêndice pretende-se demonstrar a equação que representa uma reta em coordenadas polares a partir de seus parâmetros retangulares. A equação da reta é dada pela Equação (42), em que m corresponde ao coeficiente angular da mesma e b o ponto de intersecção da reta com o eixo y. Uma reta na cor azul pode ser visualizada na Figura 56.

$$y = mx + b \tag{42}$$

Figura 56: Parâmetros de uma reta em coordenadas polares.



Na Figura 56, percebe-se que a inclinação da reta é negativa. Assim, o coeficiente angular m é dado por:

$$m = \tan(-\alpha) = \frac{sen(-\alpha)}{\cos(-\alpha)} = \frac{-sen(\alpha)}{\cos(\alpha)} = -\tan\alpha \tag{43}$$

Na Figura 56, também se percebe que os ângulos θ e α são complementares e a relação entre suas tangentes é dada pela identidade trigonométrica da Equação (45).

$$\theta = \frac{\pi}{2} - \alpha \tag{44}$$

$$\tan\left(\frac{\pi}{2} - \alpha\right) = \cot\alpha \tag{45}$$

$$\tan\theta = \cot\alpha = \frac{1}{\tan\alpha} = \frac{1}{-m} \tag{46}$$

Assim:

$$m = -\frac{1}{\tan\theta} = -\frac{\cos\theta}{sen\,\theta} \tag{47}$$

Desta forma, a Equação (42) pode ser escrita como:

$$y = -\frac{\cos\theta}{sen\,\theta}x + b \tag{48}$$

Da Figura 56, tem-se que:

$$sen\,\theta = \frac{\rho}{y} \tag{49}$$

$$y = \frac{\rho}{sen\,\theta} \tag{50}$$

Substituindo o valor de y na Equação (48), tem-se:

$$\frac{\rho}{sen\,\theta} = -\frac{\cos\theta}{sen\,\theta}x + b \tag{51}$$

O valor de b é encontrado fazendo x = 0 na Equação (51). Assim, encontra-se:

$$b = \frac{\rho}{sen\,\theta} \tag{52}$$

O valor de b da Equação (52) é, então substituído na Equação (51). Assim, encontra-se a Equação (53) que corresponde à representação de uma reta em função das coordenadas polares θ e ρ.

$$y = -\left(\frac{\cos\theta}{sen\,\theta}\right)x + \frac{\rho}{sen\,\theta} \tag{53}$$

Manipulando a Equação (53), tem-se:

$$y\,sen\,\theta = -x\cos\theta + \rho \tag{54}$$

$$\rho = y\,sen\,\theta + x\cos\theta \tag{55}$$

As Equações (53) e (55) são utilizadas nos cálculos da Transformada de Hough da Seção 2.3.

APÊNDICE D – Código do Sistema

A seguir tem-se o código do sistema proposto para identificação e classificação das linhas de sinalização em autovias, utilizando a biblioteca OpenCV. A linguagem de programação utilizada foi C++.

```
/*
Data de modificação: 09/05/2016
Autor: Francisco Alves de Oliveira Júnior

Este programa identifica e classificas as linhas de sinalização em autovias
utilizando a biblioteca OpenCV.

================================================================================================================================
*/

#include <stdlib.h>
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
#include <time.h>
#include <fstream>          // Para manipulação dos arquivos txt

using namespace std;
using namespace cv;

/// Variáveis globais ----------------------------------------------------------------------
int                 conta_frame = 0;                    // Sinaliza que é o primeiro frame

// Variáveis globais utilizadas para a Identificação
int                 erro = 0;                           // Conta a quantidade de frames errados consecutivos
bool        erro_frame_anterior = false;                // Indica que houve erro no frame anterior
float       Exb_anterior, Ext_anterior,                 // Utilizados para comparação
                    Dxb_anterior, Dxt_anterior;

// Variáveis globais para ratamento de erro na Classificação C1
string      C1_esquerda_memoria,
                    C1_direita_memoria;

int                 erro_Classificacao_esquerda = 0,
                    erro_Classificacao_direita = 0;

// Variáveis globais utilizadas para estabilização da Classificação 1
int                 conta_ausencia_E = 0,               // Conta as faltas da linha esquerda
                    conta_ausencia_D = 0;               // Conta as faltas de linha direita

int                 c1_E_t20 = 0, c1_E_t19 = 0, c1_E_t18 = 0, c1_E_t17 = 0, c1_E_t16 = 0, c1_E_t15 = 0,
                    c1_E_t14 = 0, c1_E_t13 = 0, c1_E_t12 = 0, c1_E_t11 = 0, c1_E_t10 = 0, c1_E_t9 = 0,
                    c1_E_t8 = 0, c1_E_t7 = 0, c1_E_t6 = 0, c1_E_t5 = 0, c1_E_t4 = 0, c1_E_t3 = 0,
                    c1_E_t2 = 0, c1_E_t1 = 0, c1_E_t0 = 0;

int                 c1_D_t20 = 0, c1_D_t19 = 0, c1_D_t18 = 0, c1_D_t17 = 0, c1_D_t16 = 0, c1_D_t15 = 0,
                    c1_D_t14 = 0, c1_D_t13 = 0, c1_D_t12 = 0, c1_D_t11 = 0, c1_D_t10 = 0, c1_D_t9 = 0,
                    c1_D_t8 = 0, c1_D_t7 = 0, c1_D_t6 = 0, c1_D_t5 = 0, c1_D_t4 = 0, c1_D_t3 = 0,
```

```
                    c1_D_t2 = 0, c1_D_t1 = 0, c1_D_t0 = 0;

// Variáveis globais utilizadas para estabilização da Classificação 2
int                 c2_E_t20 = 0, c2_E_t19 = 0, c2_E_t18 = 0, c2_E_t17 = 0, c2_E_t16 = 0, c2_E_t15 = 0,
                    c2_E_t14 = 0, c2_E_t13 = 0, c2_E_t12 = 0, c2_E_t11 = 0, c2_E_t10 = 0, c2_E_t9 = 0,
                    c2_E_t8 = 0, c2_E_t7 = 0, c2_E_t6 = 0, c2_E_t5 = 0, c2_E_t4 = 0, c2_E_t3 = 0,
                    c2_E_t2 = 0, c2_E_t1 = 0, c2_E_t0 = 0;

int                 c2_D_t20 = 0, c2_D_t19 = 0, c2_D_t18 = 0, c2_D_t17 = 0, c2_D_t16 = 0, c2_D_t15 = 0,
                    c2_D_t14 = 0, c2_D_t13 = 0, c2_D_t12 = 0, c2_D_t11 = 0, c2_D_t10 = 0, c2_D_t9 = 0,
                    c2_D_t8 = 0, c2_D_t7 = 0, c2_D_t6 = 0, c2_D_t5 = 0, c2_D_t4 = 0, c2_D_t3 = 0,
                    c2_D_t2 = 0, c2_D_t1 = 0, c2_D_t0 = 0;

// Variáveis globais utilizadas para estabilização da Classificação 3
int                 c3_E_t20 = 0, c3_E_t19 = 0, c3_E_t18 = 0, c3_E_t17 = 0, c3_E_t16 = 0, c3_E_t15 = 0,
                    c3_E_t14 = 0, c3_E_t13 = 0, c3_E_t12 = 0, c3_E_t11 = 0, c3_E_t10 = 0, c3_E_t9 = 0,
                    c3_E_t8 = 0, c3_E_t7 = 0, c3_E_t6 = 0, c3_E_t5 = 0, c3_E_t4 = 0, c3_E_t3 = 0,
                    c3_E_t2 = 0, c3_E_t1 = 0, c3_E_t0 = 0;

int                 c3_D_t20 = 0, c3_D_t19 = 0, c3_D_t18 = 0, c3_D_t17 = 0, c3_D_t16 = 0, c3_D_t15 = 0,
                    c3_D_t14 = 0, c3_D_t13 = 0, c3_D_t12 = 0, c3_D_t11 = 0, c3_D_t10 = 0, c3_D_t9 = 0,
                    c3_D_t8 = 0, c3_D_t7 = 0, c3_D_t6 = 0, c3_D_t5 = 0, c3_D_t4 = 0, c3_D_t3 = 0,
                    c3_D_t2 = 0,c3_D_t1 = 0, c3_D_t0 = 0;

// Variáveis globais utilizadas para estabilização da Classificação 4
int                 c4_E_t20 = 0, c4_E_t19 = 0, c4_E_t18 = 0, c4_E_t17 = 0, c4_E_t16 = 0, c4_E_t15 = 0,
                    c4_E_t14 = 0, c4_E_t13 = 0, c4_E_t12 = 0, c4_E_t11 = 0, c4_E_t10 = 0, c4_E_t9 = 0,
                    c4_E_t8 = 0, c4_E_t7 = 0, c4_E_t6 = 0, c4_E_t5 = 0, c4_E_t4 = 0, c4_E_t3 = 0,
                    c4_E_t2 = 0, c4_E_t1 = 0, c4_E_t0 = 0;

int                 c4_D_t20 = 0, c4_D_t19 = 0, c4_D_t18 = 0, c4_D_t17 = 0, c4_D_t16 = 0, c4_D_t15 = 0,
                    c4_D_t14 = 0, c4_D_t13 = 0, c4_D_t12 = 0, c4_D_t11 = 0, c4_D_t10 = 0, c4_D_t9 = 0,
                    c4_D_t8 = 0, c4_D_t7 = 0, c4_D_t6 = 0, c4_D_t5 = 0, c4_D_t4 = 0, c4_D_t3 = 0,
                    c4_D_t2 = 0, c4_D_t1 = 0, c4_D_t0 = 0;

// Variáveis globais utilizadas para estabilização da Classificação 2
int conta_seccionada_esq = 0;               // Conta frames seguidos com linhas seccionadas a esquerda
int conta_seccionada_dir = 0;               // Conta frames seguidos com linhas seccionadas a direita
int conta_continua_esq = 0;                 // Conta frames seguidos com linhas continuas a esquerda
int conta_continua_dir = 0;                 // Conta frames seguidos com linhas continuas a direita
// Variáveis globais utilizadas para estabilização da Classificação 3
int conta_primeira_seccionada_esq = 0;      // Conta frames seguidos das linhas seccionadas a esquerda da primeira linha de uma dupla
int conta_primeira_seccionada_dir = 0;      // Conta frames seguidos das linhas seccionadas a direita da primeira linha de uma dupla
int conta_primeira_continua_esq = 0;        // Conta frames seguidos das linhas continua a esquerda da primeira linha de uma dupla
int conta_primeira_continua_dir = 0;        // Conta frames seguidos das linhas continua a direita da primeira linha de uma dupla
// Variáveis globais utilizadas para estabilização da Classificação 4
int conta_segunda_seccionada_esq = 0;       // Conta frames seguidos das linhas seccionadas a esquerda da segunda linha de uma dupla
int conta_segunda_seccionada_dir = 0;       // Conta frames seguidos das linhas seccionadas a direita da segunda linha de uma dupla
int conta_segunda_continua_esq = 0;         // Conta frames seguidos das linhas continua a esquerda da segunda linha de uma dupla
int conta_segunda_continua_dir = 0;         // Conta frames seguidos das linhas continua a direita da segunda linha de uma dupla
int quant_frames = 1;                       // Quantidade geral de frames

/// Funções utilizadas ---------------------------------------------------------------------------

void fc_verifica_erro (void);
void fc_linha (char direcao, float B, float T, Mat& src, float ymax, float y_desloca);
void fc_texto (string text, int centro_x, int centro_y, Mat& imagem);
string intToString(int number);
void fc_segmentos_do_grupo (int E1, int grupo[], float x1[], float x2[], float y1[], float y2[], float xb[], float xt[], float novo_x1[], float novo_x2[], float novo_y1[], float novo_y2[],
float novo_xb[], float novo_xt[], int n, int& total);
void fc_forma_grupo (int total, float novo_xb[], float largura_subgrupo, int& conta_subgrupo, int subgrupo[]);
void fc_classifica1 (string direcao, int conta_grupo, string& texto);
void fc_classifica2 (string direcao, int ymax, int xmax, int total, float pos_x1[], float pos_x2[], float pos_y1[], float pos_y2[], string& texto);
void fc_classifica3 (string direcao, int ymax, int xmax, int total, float pos_x1[], float pos_x2[], float pos_y1[], float pos_y2[], int subgrupo[], string& texto);
void fc_classifica4 (string direcao, int ymax, int xmax, int total, float pos_x1[], float pos_x2[], float pos_y1[], float pos_y2[], int subgrupo[], int total_subgrupo_direcao,
```

```
string& texto);
int fc_maior_y (int ymax, int xmax, int total, float pos_x1[], float pos_x2[], float pos_y1[], float pos_y2[]);
void fc_estatistica (float valores[], float media, int n, float& sigma2, float& sigma, float& CV);
void fc_boxplot (float valores[], int n, float& lim_inf, float& lim_sup);

/// Programa Principal ---------------------------------------------------------------------------
int main(int, char**)
{
	VideoCapture cap;
	string	nome_arquivo_saida;		// = "teste.txt";
	string	nome_arquivo_padrao;		// = "padao.txt";
	string	caminho = "experimentos";
	string	pasta;

	int	y_desloca,			// Deslocamento da ROI para baixo
		altura,				// Altura da ROI
		largura_carro,
		largura_faixa,
		paralax = 10;

	float	Db = 10,			// Limiar da base da ROI
		Dt = 10,			// Limiar do topo da ROI
		Ds = 10,			// Limiar par junção de segmentos consecutivos
		largura_subgrupo;		// Limiar para unir grupos e segmentos muito próximos

	int	ultimo_frame_teste;


	// Menu de escolha dos experimentos ==========================================

	int	numero_experimento;

	cout<<"Experimentos disponiveis:"<<endl<<endl;

	cout<<"	1 - clip_85"<<endl;
	cout<<"	2 - clip_94"<<endl;
	cout<<"	3 - clip_96"<<endl;
	cout<<"	4 - clip_104"<<endl;
	cout<<"	5 - RS-040"<<endl;
	cout<<"	6 - japao_dia_reto"<<endl;
	cout<<"	7 - japao_noite_4"<<endl;
	cout<<"	8 - japao_noite_2"<<endl;

	cout<<endl<<"Digite o numero do experimento: ";
	cin>>numero_experimento;

	if(numero_experimento == 1)
	{
		VideoCapture cap_entrada("./experimentos/video_01/clip_i5s_0085-0302resultado.avi");	//  Dimensões: 1100 x 380.
		cap = cap_entrada;
		pasta = "video_01";
		y_desloca = 150; altura = 230; largura_carro = 900; largura_faixa = 1000; paralax = 0;
		Db = 5;		//10// 5
		Dt = 3;		//5// 3
		Ds = 5;		// Padrão = 5
		largura_subgrupo = 15; //15
		ultimo_frame_teste = cap.get(CV_CAP_PROP_FRAME_COUNT) – 1; // Obtem a quantidade total de frames do vídeo. 201 - 1 = 200.

	}
	else if(numero_experimento == 2)
	{
		VideoCapture cap_entrada("experimentos/video_02/clip_i5s_0094-0302resultado.avi"); //  Dimensões: 1100 x 380
		cap = cap_entrada;
		pasta = "video_02";
		y_desloca = 150; altura = 230; largura_carro = 900; largura_faixa = 1000; paralax = 0;
```

```
		Db = 15;	//10 // 5				// Proporção 19~7 => Db = 2,714*Dt
		Dt = 6;		//4  // 3				// Melhor Db = 15 e Dt = 6
		Ds = 5;		// Padrão = 5
		largura_subgrupo = 20;	//15 ==== melhor 20
		ultimo_frame_teste = cap.get(CV_CAP_PROP_FRAME_COUNT) – 1; // Obtem a quantidade total de frames do vídeo.  1201 - 1 = 1200
}
else if(numero_experimento == 3)
{
		VideoCapture cap_entrada("experimentos/video_03/clip_i5s_0096-0302resultado.avi"); //  Dimensões: 1100 x 380
		cap = cap_entrada;
		pasta = "video_03";
		y_desloca = 150; altura = 230; largura_carro = 900; largura_faixa = 1000; paralax = 0;
		Db = 15; //10 // 15 // 5				// Proporção 20~7 => Db = 2,86*Dt
		Dt = 5;  //4  // 5  // 3				// 5-3 foi melhor, mais o mais lógico é 15-5 que vou manter
		Ds = 5;		// Padrão = 5
		largura_subgrupo = 25; //15			// Melhor 25
		ultimo_frame_teste = cap.get(CV_CAP_PROP_FRAME_COUNT) – 1; // Obtém a quantidade total de frames do vídeo. 1001 - 1 = 1000
}
else if(numero_experimento == 4)
{
		VideoCapture cap_entrada("experimentos/video_04/clip_i5s_0104-0302resultado.avi"); //  Dimensões: 1100 x 380
		cap = cap_entrada;
		pasta = "video_04";
		y_desloca = 150; altura = 230; largura_carro = 900; largura_faixa = 1000; paralax = 0;
		Db = 15;	//10// 15// 5			// Proporção 20~7,4 => Db = 2,702*Dt
		Dt = 6;		//4// 6// 3			// 10-4 foi melhor, mas vou manter 15-6 pela coerência com minha teoria
		Ds = 5;		// Padrão = 5
		largura_subgrupo = 20; //15   // Melhor 20
		ultimo_frame_teste = cap.get(CV_CAP_PROP_FRAME_COUNT) – 1; // Obtem a quantidade total de frames do vídeo. 1101 - 1 = 1100
}
else if(numero_experimento == 5)
{
		VideoCapture cap_entrada("experimentos/video_05/rs-040.mp4"); // Video da BR no Rio Grande do Sul. Dimensões: 632 x 480
		cap = cap_entrada;
		pasta = "video_05";
		y_desloca = 210; altura = 120; largura_carro = 260; largura_faixa = 460; paralax = 50;
		Db = 5;		//9// 4 //5				// Prroporção 9,5-3,2 ==> Db = 2,97 * Dt
		Dt = 3;		//3// 2 //3				// Melhor 5-3
		Ds = 5;		// Padrão = 5
		largura_subgrupo = 7; // Inicial = 10
		//ultimo_frame_teste = cap.get(CV_CAP_PROP_FRAME_COUNT) – 1; // Obtém a quantidade total de frames do vídeo. (11360) Original
		ultimo_frame_teste = 10797;
}
else if(numero_experimento == 6)
{
		VideoCapture cap_entrada("experimentos/video_06/japao_dia_reto.mp4"); // Japão. Dimensões: 640 x 360
		cap = cap_entrada;
		pasta = "video_06";
		y_desloca = 200; altura = 160; largura_carro = 330; largura_faixa = 600; paralax = 0;
		Db = 10;	//10// 5					//Proporção: ==> Db = 6,28 * Dt
		Dt = 3;		//5// 3
		Ds = 5;		// Padrão = 5
		largura_subgrupo = 15;
		//ultimo_frame_teste = cap.get(CV_CAP_PROP_FRAME_COUNT) – 1; // Obtém a quantidade total de frames do vídeo. (17647) Original
		ultimo_frame_teste = 1000;
}
else if(numero_experimento == 7)
{
		VideoCapture cap_entrada("experimentos/video_07/japao_noite_4.mp4");		// Japão - Noite. Dimensões: 640 x 360
		cap = cap_entrada;
		pasta = "video_07";
		y_desloca = 220; altura = 120; largura_carro = 300; largura_faixa = 550; paralax = 30;
		Db = 5;		//10// 5				//Proporção: 12,6-2,8 ==> Db = 4,5 * Dt
		Dt = 3;		 //5// 3				// Melhor 5-3
		Ds = 5;		// Padrão = 5
```

```
		largura_subgrupo = 12;	// original = 15  // melhor 12
		ultimo_frame_teste = cap.get(CV_CAP_PROP_FRAME_COUNT) – 1; // Obtém a quantidade total de frames do vídeo. 1186 - 1 = 1185
}
else if(numero_experimento == 8)
{
		VideoCapture cap_entrada("experimentos/video_08/japao_noite_2.mp4"); // Japão - Noite. Dimensões: 640 x 360
		cap = cap_entrada;
		pasta = "video_08";
		y_desloca = 220; altura = 140; largura_carro = 300; largura_faixa = 550; paralax = 30;
		Db = 5;		//10// 5				// Proporção: 12,6-2,8 ==> Db = 4,5 * Dt
		Dt = 3;		//5// 3				// Melhor 5-3
		Ds = 5;		// Padrão = 5
		largura_subgrupo = 12;	// original = 15  // melhor 12
		ultimo_frame_teste = cap.get(CV_CAP_PROP_FRAME_COUNT) – 1; // Obtém a quantidade total de frames do vídeo. 6161 - 1 = 6160
}
else
{
}

// Montando os caminhos dos arquivos txt para gravação e para comparação ------------------------
caminho = caminho + "/" + pasta;
nome_arquivo_saida = caminho + "/teste.txt";
nome_arquivo_padrao = caminho + "/padrao.txt";

/// INFORMAÇÕES SOBRE O VIDEO ANALISADO %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//Comando utilizado para obter os parâmetros do video:   double VideoCapture::get(int propId)

int altura_video_original = cap.get(CV_CAP_PROP_FRAME_HEIGHT);		// Obtém a altura do vídeo.
int largura_video_original = cap.get(CV_CAP_PROP_FRAME_WIDTH);		// Obtém a largura do vídeo.
float taxa_de_exibicao = cap.get(CV_CAP_PROP_FPS);			// Obtém a taxa de exibição em quadros/s.

cout<<endl<<endl<<"====================== INFORMACOES SOBRE O VIDEO ====================="<<endl<<endl;
cout<<endl<<"Dimensoes do video (largura x altura): ("<<largura_video_original<<" x "<<altura_video_original<<")"<<endl;
cout<<endl<<"Taxa de exibicao = "<<taxa_de_exibicao<<" quadros/s"<<endl;
cout<<endl<<"Quantidade de quadros a serem analisados = "<<ultimo_frame_teste<<endl;

/// Cálculo dos Parâmetros ---------------------------------------------------------------------------------------------
float largura_grupo = 0.07*largura_faixa;		// Utilizado para selecionar os grupos de segmentos

/// Definindo a Região de Interesse -----------------------------------------------------------------------------------
int x = 0, y = y_desloca, h = altura;
//int w = src.cols;
int margem = 2; // Reduzo as margens para evitar surgimento de linhas perpendiculares nas laterais

/// Outras variáveis --------------------------------------------------------------------------------------------------------
float		ymax = altura - 1;		// Ordenada referente à altura da ROI em termos de linhas

// Definição de Matrizes para Imagens ---------------------------------------------------------------------------------
Mat		src;			// Imagem de entrada
Mat		roi_gray;		// Para a imagem ROI cinza
Mat		img_blur;		// Para a imagem com Blur
Mat		dst;			// Imagem de destino
Mat		A;

vector<Vec4i>		lines;		// Vetor utilizado por Hough
Vec4i			l;		// O vetor "l" armazena todas as linhas encontradas por Hough

// Variáveis utilizadas na IDENTIFICAÇÃO: Parâmetros para comparação ------------------------------------------------------------

float GAMA = 0.17*largura_faixa;			// Largura da faixa de transição		17%
float tolerancia_DELTA = 0.2*largura_faixa;		// Tolerância da largura da faixa		20%
float d = 0.08*largura_faixa;				// Tolerância da posição da linha		8%

//// Variáveis utilizadas para definir parâmetros dos segmentos de linha ----------------------------------------------
float		x1[300],			// Abcissa do ponto inicial
```

```
                y1[300],                    // Ordenada do ponto inocial
                x2[300],                    // Abcissa do ponto final
                y2[300],                    // Ordenada do ponto final
                delta_y[300],               // Variação vertical do segmento de reta
                delta_x[300],               // Variação horizontal do segmento de reta
                inclinacao[300],            // Inclinação
                xb[300],                    // Posição de x na base da ROI
                xt[300];                    // Posição de y no topo da ROI

int             ordenado[300],              // Se o segmento estiver ordenado, = 1, se não, = 0
                ativo[300];                 // Se 0 = desativado; Se 1 = ativado. No início, todos estarão ativados

int             numSegmentos;               // Número de segmentos encontrados por Hough

//************************************************************************************************************************************************************************

        ///------------------------------------------------------------------------------------------------------

        if(!cap.isOpened())  // check if we succeeded
        return -1;

        float e1_xb_total = 0;              // Usado para amortecer o e1_xb e e1_xt em todos os quadros
        float e1_xt_total = 0;

        float d1_xb_total = 0;              // Usado para amortecer o d1_xb e d1_xt em todos os quadros
        float d1_xt_total = 0;

        float peso;                         //4;

        float vertical_PF;


        float EXB, EXT, DXB, DXT;           // Coordenadas das linhas a serem impressas
        float largura_B = 0;                // Só para se ter ideia de início
        float largura_T = 0;                // Só para se ter ideia de início

        bool anterior_esq_ok = false;       // Inicialmente não há referencia confiável do frame anterior,.... lógico.
        bool anterior_dir_ok = false;       // Inicialmente não há referencia confiável do frame anterior,.... lógico.

        clock_t t0, tf;
        float tempo_gasto;
        float ciclos;

        t0 = clock();   // Grava momento atual

        // Matrizes para armazenar os resultados, temporariamente, antes de gravar no arquivo .txt.
        string          guarda_esquerda[15000],
                        guarda_direita[15000];

        for(;;)
        {
                /// Programa principal

                cap >> src;

                conta_frame++;

                /// ROI - REGIÃO DE INTERESSE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                int w = src.cols;
                Rect region_of_interest = Rect(x + margem, y, w - 2*margem, h);
                Mat src_roi = src(region_of_interest);

                /// CONVERTE PARA GRAYSCALE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                cvtColor(src_roi, roi_gray, CV_BGR2GRAY);
```

```
///APLICA BLUR %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
blur( roi_gray, img_blur, Size(3,3) );        // Reduz o ruído com um filtro 3x3

/// CANNY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Canny(img_blur, dst, 20, 60, 3);              // Valores fixos que funcionam muito bem em todos os videos testados até o momento

A = Mat::zeros(dst.rows, dst.cols, CV_AA);

/// IMPLEMENTANDO A TRANSFORMADA PROBABILÍSTICA DE HOUGH %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
HoughLinesP(dst, lines, 1, CV_PI/180, 20, 30, 5 );      // ... ,quantidade mínima de votos, tamanho mínimo, distânca máxima) CORRETO

numSegmentos = lines.size();

for( size_t i = 0; i < numSegmentos; i++ )
{
            l = lines[i];

            x1[i] = l[0];     // X1
            y1[i] = l[1];     // y1
            x2[i] = l[2];     // x2
            y2[i] = l[3];     // y2

            delta_y[i] = y2[i] - y1[i]; // Variação da altura y
            delta_x[i] = x2[i] - x1[i]; // Variação do comprimento y

            // Para evitar divisões por zero, faz-se uma aproximação:
            if(delta_x[i] == 0)
            {
                        delta_x[i] = 0.01;
            }

            /// Testando inclinação da reta ------

            inclinacao[i] = delta_y[i]/delta_x[i]; // Inlinação

            // Para evitar divisões por zero, faz-se uma aproximação:
            if(inclinacao[i] == 0)
            {
                        inclinacao[i] = 0.0000001;
            }

            /// Calculando Xb(Ymax) que é o valor de X no bottom da imagem
            //  Pela equação da reta, tem-se:

            xb[i] = x1[i] + (ymax - y1[i])/inclinacao[i];

            /// Calculando Xt(Y = 0) que é o valor de X no topo da imagem
            //  Pela equação da reta, tem-se:

            xt[i] = x1[i] - y1[i]/inclinacao[i];

            /// Indica se o segmento de reta foi ou não ordenado. No início, não estará ordenado.
            ordenado[i] = 0;             // 1 - Foi ordenado;
                                         // 0 - Não foi ordenado.
            /// Habilitando o segmento de reta
            ativo[i] = 1;                // 1 - Ativa o segmento;
                                         // 0 - Desativa o segmento.
}

///DESATIVA OS SEGMENTOS DE RETA HORIZONTAIS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for( int i = 0; i < numSegmentos; i++ )
{
            if(ativo[i] == 1 && abs(delta_y[i]) <= 4)    // São praticamente horizontais
```

```
			{
				ativo[i] = 0;	// Desativo o segmento
			}

		}

		///DESATIVA SEGMENTOS COM POSIÇÃO HORIZONTAL INFERIOR E/OU SUPERIOR MUITO AFASTADAS %%%%%%%%%%%%%
		for( int i = 0; i < numSegmentos; i++ )
		{
			// Posição Superior muito afastada
			if(
				ativo[i] == 1
				// Se ativo
				&& (xt[i] > 2*src_roi.cols || xt[i] < -1.5*src_roi.cols))	// Se a abcissa xt for muito distante para menos ou para mais
			{
				ativo[i] = 0;	// Desativo o segmento
				//cout<<"Desativei segmento com xt muito afastado"<<i<<endl;
			}

			// Posição Inferior muito afastada
			if(
				ativo[i] == 1
				// Se ativo
				&& (xb[i] > 2*src_roi.cols || xb[i] < -2*src_roi.cols))	// Se a abcissa xb for muito distante para menos ou para mais
			{
				ativo[i] = 0;	// Desativo o segmento
				//cout<<"Desativei segmento com xb muito afastado"<<i<<endl;
			}

			// Analisa a diferença em relação ao eixo x da Posição Inferior e Superior
			if(
				ativo[i] == 1
				// Se ativo
				&& abs(xb[i] - xt[i]) > 2*src_roi.cols)			// Se abs(xb - xt) for muito grande
			{
				ativo[i] = 0;	// Desativo o segmento
				//cout<<"Desativei pela grande diferença entre xb e xt: "<<i<<endl;
			}
		}

		// PONTO DE FUGA %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

		float Y_PF[300];
		float X_PF[300];

		float sigma2;		// Variância
		float sigma;		// Desvio Padrão
		float CV;		// Coeficiente de Variação


		// vertical_PF ===========================================================
		float PF, g, G;
		float soma_PF = 0;
		int quantidade_PF = 0;

		for( int i = 0; i < numSegmentos; i++ )
		{
			for( int j = i + 1; j < numSegmentos; j++ )
			{
				if(
					ativo[i] == 1				// Se i for válido
					&& ativo[j] == 1			// Se j for válido
					//&& inclinacao[i]*inclinacao[j] < 0	// Se tiverem inclinações opostas (para evitar erros grandes)
					&& abs(xt[i] - xt[j]) < src_roi.cols/3	// Se a parte de cima não for muito grande
```

```
                                        && abs(xt[i] - xt[j]) > src_roi.cols/6          // Se a parte de cima não for muito pequena
                                        && abs(xb[i] - xb[j]) < 2*src_roi.cols          // Se a parte de baixo não for muito grande
                                        && abs(xb[i] - xb[j]) > 0.5*src_roi.cols        // Se a parte de baixo não for muito pequena
                                        && abs(xt[i] - xt[j]) < 1.5*abs(xb[i] - xb[j]) // Se a parte de cima for menor que a de baixo
                                        //&& (xt[i] - xt[j])*(xb[i] - xb[j]) > 0          // Se não se cruzarem
                                        )
                                {
                                        g = abs(xt[i] - xt[j]);
                                        G = abs(xb[i] - xb[j]);
                                        PF = g*ymax/(G-g);
                                        soma_PF = soma_PF + PF;
                                        Y_PF[quantidade_PF] = PF;
                                        X_PF[quantidade_PF] = x1[i] + (- Y_PF[quantidade_PF] - y1[i])/inclinacao[i];
                                        quantidade_PF++;
                                }
                        }
}
// Limites horizontais que serão encontrados pela função Boxplot
                float           x_inferior,
                                x_superior;
// Limites verticais que serão encontrados pela função Boxplot
                float           y_inferior,
                                y_superior;
if(quantidade_PF > 0)
{
                vertical_PF = soma_PF/quantidade_PF;
                fc_estatistica (Y_PF, vertical_PF, quantidade_PF, sigma2, sigma, CV);
                // Seleção por BOXPLOT --------
                fc_boxplot(Y_PF, quantidade_PF, y_inferior, y_superior);
                fc_boxplot(X_PF, quantidade_PF, x_inferior, x_superior);
}
float           valor_x = 0,
                valor_y = 0;
int             quant_valida = 0;
for( int i = 0; i < quantidade_PF; i++ )
{
                if(x_inferior < X_PF[i] && X_PF[i] < x_superior && y_inferior < Y_PF[i] && Y_PF[i] < y_superior)     // Se X_PF e Y_PF
                                                                                                                    //estiverem dentro do limite
                {
                                valor_x = valor_x + X_PF[i];
                                valor_y = valor_y + Y_PF[i];
                                quant_valida++;
                }
}
float           media_x,
                media_y;
if(quant_valida > 0)            // Evita divisão por zero
{
                media_x = valor_x/quant_valida;
                media_y = valor_y/quant_valida;
}
// REFINAMENTO: Elimina os segmentos de reta que não apontam para o Ponto de Fuga calculado
float verifica_horizontal_PF;
for( int i = 0; i < numSegmentos; i++ )
{
                if(ativo[i] == 1)
                {
                                verifica_horizontal_PF = x1[i] + (- media_y - y1[i])/inclinacao[i];
                                if( verifica_horizontal_PF < x_inferior || verifica_horizontal_PF > x_superior) // Se estiver fora dos limites
                                {
                                                ativo[i] = 0;
                                                //cout<<"Eliminei segmento---------------------------------------: "<<i<<endl;
                                }

                }
```

```
                }
                // REFINAMENTO: Elimina os segmentos de reta que não apontam para o Ponto de Fuga calculado
                float verifica_vertical_PF;
                for( int i = 0; i < numSegmentos; i++ )
                {
                        if(ativo[i] == 1)
                        {
                                verifica_vertical_PF = (media_x - x1[i])*inclinacao[i] + y1[i];
                                if( verifica_vertical_PF < -y_superior || verifica_vertical_PF > -y_inferior)           // Se estiver fora dos limites
                                {
                                        ativo[i] = 0;
                                        //cout<<"Eliminei segmento ------------------------: "<<i<<endl;
                                }
                        }
                }

                /// ELIMINAR OS SEGMENTOS DE RETA COLINEARES, DUPLICADOS OU MUITO PRÓXIMOS %%%%%%%%%%%%%%%%%%%%

for( int mm = 0; mm < 2; mm++ )
{
        for( int i = 0; i < numSegmentos; i++ )
        {
                for( int j = i + 1; j < numSegmentos; j++ )             // Faço j = i + 1 para evitar comparar 2 vezes o mesmo par.
                {
                        if(j != i)
                        {

                                // Comparando as posições no topo e no bottom
                                float difb = abs(xb[i] - xb[j]);
                                float dift = abs(xt[i] - xt[j]);

                                // Só examino os segmentos ativos
                                if( difb < Db && dift < Dt && ativo[i] == 1 && ativo[j] == 1)
                                {
                                        int d, e;
                                        // Descobrir qual segmento está mais a direita.
                                        if(x2[i] >= x2[j])
                                        {
                                                d = i;
                                                e = j;
                                        }else
                                        {
                                                d = j;
                                                e = i;
                                        }
                                        // Executa comparações
                                        if(x1[d] > x2[e]) // Significa que são separados
                                        {
                                                if((x1[d] - x2[e]) <= Ds)
                                                {
                                                        // Vou unir em uma só reta
                                                        x1[d] = x1[e];          // Aumento o tamanho do segmento da direita
                                                        y1[d] = y1[e];
                                                        ativo[e] = 0;           // Desativo o segmento da esquerda

                                                        delta_y[d] = y2[d] - y1[d];
                                                        delta_x[d] = x2[d] - x1[d];

                                                        // Para evitar divisões por zero, faz-se uma aproximação:
                                                        if(delta_x[d] == 0)
                                                        {
                                                                delta_x[d] = 0.01;
                                                        }

                                                        inclinacao[d] = delta_y[d]/delta_x[d];
```

```
									// Para evitar divisões por zero, faz-se uma aproximação:
									if(inclinacao[d] == 0)
									{
										inclinacao[d] = 0.0000001;
									}
									xb[d] = x1[d] + (ymax - y1[d])/inclinacao[d];
									xt[d] = x1[d] - y1[d]/inclinacao[d];
									//cout<<"Achei xb1 = "<<xb[d]<<endl;
									}
								}
								else		// Se não estão separados
								{
									if(x1[d] > x1[e])		//Não está totalmente contido
									{
									// Vou unir em uma só reta
									x1[d] = x1[e]; // Aumento o tamanho do segmento da direita
									y1[d] = y1[e];
									ativo[e] = 0;   // Desativo o segmento da esquerda
									delta_y[d] = y2[d] - y1[d];
									delta_x[d] = x2[d] - x1[d];
									// Para evitar divisões por zero, faz-se uma aproximação:
									if(delta_x[d] == 0)
									{
										delta_x[d] = 0.01;
									}
									inclinacao[d] = delta_y[d]/delta_x[d];
									// Para evitar divisões por zero, faz-se uma aproximação:
									if(inclinacao[d] == 0)
									{
										inclinacao[d] = 0.0000001;
									}
									xb[d] = x1[d] + (ymax - y1[d])/inclinacao[d];
									xt[d] = x1[d] - y1[d]/inclinacao[d];
									//cout<<"Achei xb2 = "<<xb[d]<<endl;
								}
								else
								{
									ativo[e] = 0;   // Desativo o segmento da esquerda
									//cout<<"Caso 3: Continuei com "<<d<<" e desativei "<<e<<endl;
								}
							}
						}
					}
				}
		}
		}
				/// Visualizar os segmentos após eliminação de linhas colineares
				for( int i = 0; i < numSegmentos; i++ )
				{
					if(ativo[i] == 1) // Se o segmento for válido
					{
						line( A, Point(x1[i], y1[i]), Point(x2[i], y2[i]), Scalar(0,0,255), 1, CV_AA);
						line( src, Point((int)x1[i], (int)(y1[i])+y_desloca), Point((int)x2[i], (int)y2[i]+y_desloca), Scalar(0,255,255), 2, CV_AA);
					}

				//imshow("Linhas Filtradas", src);
				//imshow("Linhas Filtradas", A);
/// &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
//******** Ordenação **********
				/// ORDENAÇÃO DOS DADOS

				float		xbo[300],				//Vetor xb ordenado e simplificado
						xto[300],
						x1o[300],
						y1o[300],
```

```
                x2o[300],
                y2o[300];
int             conta_ordem = 0;
for (int i = 0; i < numSegmentos; i++)
{
                if(ativo[i] == 1)
                {
                                xbo[conta_ordem] = xb[i];
                                xto[conta_ordem] = xt[i];
                                x1o[conta_ordem] = x1[i];
                                y1o[conta_ordem] = y1[i];
                                x2o[conta_ordem] = x2[i];
                                y2o[conta_ordem] = y2[i];
                                conta_ordem++;
                }
}
/// ============================ novo ==================================
int             conta_grupo;
int             grupo[300];
float           media_xb[300];
float           media_xt[300];
bool            esquerda[100];
bool            direita[100];
int             quant_esquerda = 0,
                quant_direita = 0;
int             E1;
int             D1;
int             n = conta_ordem;            // Nova quantidade de segmentos

if(n == 0)      // Se não existe nenhum segmento ativo
{
                //cout<<"n = 0. Nao tem segmentos validos."<<endl;
                conta_grupo = 0;
}
else if(n == 1) // Não precisa ordenar, pois só tem 1
{
                conta_grupo = 1;
                grupo[0] = 1;

                xb[0] = xbo[0];
                xt[0] = xto[0];
                x1[0] = x1o[0];
                y1[0] = y1o[0];
                x2[0] = x2o[0];
                y2[0] = y2o[0];

                media_xb[0] = xb[0];
                media_xt[0] = xt[0];

                esquerda[0] = false;
                direita[0] = false;

                if(media_xb[0] - src.cols/2 <= 0)          // Se estiver do lado esquerdo
                {
                                esquerda[0] = true;
                                quant_esquerda++;
                                E1 = 0;
                }
                else
                {
                                direita[0] = true;
                                quant_direita++;
                                D1 = 0;
                }
```

```
                        }
else
{

/// ORDENAÇÃO DOS DADOS
//int           indice[300];   // Índice utilizado para ordenar os vetores
int             i, j;
float           xx, aa, bb, cc, dd, ee;                 // ff
for (j = 1; j < n; ++j)
{
                xx = xbo[j];

                aa = xto[j];
                bb = x1o[j];
                cc = y1o[j];
                dd = x2o[j];
                ee = y2o[j];
                for (i = j-1; i >= 0 && xbo[i] > xx; --i)
                {
                        xbo[i+1] = xbo[i];
                        xto[i+1] = xto[i];
                        x1o[i+1] = x1o[i];
                        y1o[i+1] = y1o[i];
                        x2o[i+1] = x2o[i];
                        y2o[i+1] = y2o[i];
                }
                xbo[i+1] = xx;
                xto[i+1] = aa;
                x1o[i+1] = bb;
                y1o[i+1] = cc;
                x2o[i+1] = dd;
                y2o[i+1] = ee;
}
/// Reorganiza os dados em seus vetores de origem
for (int i = 0; i < n; i++)
{
                        xb[i] = xbo[i];
                        xt[i] = xto[i];
                        x1[i] = x1o[i];
                        y1[i] = y1o[i];
                        x2[i] = x2o[i];
                        y2[i] = y2o[i];
}
/// SELECIONANDO GRUPOS ****************************
conta_grupo = 1;
grupo[0] = conta_grupo;
for(int i = 1; i < n ; i++)                             // O número 1 no início está correto
{
                if(xb[i] - xb[i - 1] > largura_grupo)           // Limite inferior para se juntar ao grupo
                {
                        conta_grupo++;
                }
                grupo[i] = conta_grupo;
}
/// CALCULANDO media_xb DE CADA GRUPO **********************
float soma;
float quant;
for(int i = 1; i < conta_grupo + 1; i++) // Valores dos grupos variam de 1 a conta_grupo + 1
{
                soma = 0;
                quant = 0;
                for(int j = 0; j < n; j++) // O índice dos grupos começa com 0 (zero)
                {
                        if(grupo[j] == i)
                        {
```

```
                                        soma = soma + xb[j];
                                        quant++;
                                }
                }
                media_xb[i] = soma/quant;
}
/// CALCULANDO media_xt DE CADA GRUPO **********************
for(int i = 1; i < conta_grupo + 1; i++)
{
                soma = 0;
                quant = 0;
                for(int j = 0; j < n; j++)
                {
                                if(grupo[j] == i)
                                {
                                                soma = soma + xt[j];
                                                quant++;
                                }
                }
                media_xt[i] = soma/quant;  // Índices começam com 1
}
/// DEFININDO O POSICIONAMENTO DAS RETAS ***********************************************
for(int i = 1; i < conta_grupo + 1; i++)
{
                esquerda[i] = false;
                direita[i] = false;
                if(media_xb[i] - src.cols/2 <= 0)                                                // Se estiver do lado esquerdo
                {
                                esquerda[i] = true;
                                quant_esquerda++;
                }
                else
                {
                                direita[i] = true;
                                quant_direita++;
                }
}
/// Encontrando E1 **********************
                if(quant_esquerda > 0)      // Se existe algum grupo à esquerda
                {
                                float maior = -10000;
                                for(int i = 1; i < conta_grupo + 1; i++)
                                {
                                                if(esquerda[i] == true)
                                                {
                                                                if(media_xb[i] > maior)
                                                                {
                                                                                maior = media_xb[i];
                                                                                E1 = i;
                                                                }
                                                }
                                }
                }
/// Encontrando D1 **********************
                if(quant_direita > 0)          // Se existe algum grupo à direita
                {
                                float menor = 10000;
                                for(int i = 1; i < conta_grupo + 1; i++)
                                {
                                                if(direita[i] == true)
                                                {
                                                                if(media_xb[i] < menor)
                                                                {
                                                                                menor = media_xb[i];
                                                                                D1 = i;
```

```
                    }
                }
            }
        }
}

/// IDENTIFICAÇÃO: PLOTANDO AS LINHAS
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

if(conta_grupo > 0)        // Se existe algum grupo
{
        if(conta_frame > 3 && largura_B != 0 && largura_T != 0)        // Se não é inicio do video nem ocorreram 5 erros seguidos
        {
                /// --  I  ------------------------------------------------------------------------
                if(quant_esquerda > 0 && quant_direita > 0)        // Se existe E1 e D1
                {
                        if(abs((media_xb[D1] - media_xb[E1]) - largura_B) < tolerancia_DELTA) // Se a largura da faixa estiver correta
                        {
                                if(abs(media_xb[E1] - Exb_anterior) < GAMA || abs(media_xb[D1] - Dxb_anterior) < GAMA)
                                {
                                        EXB = media_xb[E1];
                                        EXT = media_xt[E1];
                                        fc_linha('E', EXB, EXT, src, ymax, y_desloca);
                                        DXB = media_xb[D1];
                                        DXT = media_xt[D1];
                                        fc_linha('D', DXB, DXT, src, ymax, y_desloca);
                                        //#################################
                                        erro_frame_anterior = false;
                                        erro = 0;
                                        //#################################
                                }
                                else
                                {
                                        DXB = Dxb_anterior;
                                        DXT = Dxt_anterior;
                                        fc_linha('D', DXB, DXT, src, ymax, y_desloca);
                                        EXB = Exb_anterior;
                                        EXT = Ext_anterior;
                                        fc_linha('E', EXB, EXT, src, ymax, y_desloca);
                                        // Função para verificar erro -------
                                        fc_verifica_erro();
                                }
                        } // Fim do processo I ------------------------------------------------------------
                        /// --  II  -----------------------------------------------------------------------
                        else
                        {
                                if(abs(media_xb[E1] - Exb_anterior) < d)
                                {
                                        EXB = media_xb[E1];
                                        EXT = media_xt[E1];
                                        fc_linha('E', EXB, EXT, src, ymax, y_desloca);
                                        DXB = media_xb[E1] + largura_B;
                                        DXT = media_xt[E1] + largura_T;
                                        fc_linha('D', DXB, DXT, src, ymax, y_desloca);
                                }
                                else
                                {
                                        if(abs(media_xb[D1] - Dxb_anterior) < d)
                                        {
                                                DXB = media_xb[D1];
                                                DXT = media_xt[D1];
                                                fc_linha('D', DXB, DXT, src, ymax, y_desloca);

                                                EXB = media_xb[D1] - largura_B;
```

```
                                        EXT = media_xt[D1] - largura_T;
                                        fc_linha('E', EXB, EXT, src, ymax, y_desloca);
                                }
                                else
                                {
                                        DXB = Dxb_anterior;
                                        DXT = Dxt_anterior;
                                        fc_linha('D', DXB, DXT, src, ymax, y_desloca);
                                        EXB = Exb_anterior;
                                        EXT = Ext_anterior;
                                        fc_linha('E', EXB, EXT, src, ymax, y_desloca);
                                }
                        }
                        // Função para verificar erro -------
                        fc_verifica_erro();
                }               // Fim do processo II ------
}
/// --  III  ------------------------------------------------------------------------
if(quant_esquerda > 0 && quant_direita == 0)            // Se existe somente E1
{
                if(abs(media_xb[E1] - Exb_anterior) < d)
                {
                        EXB = media_xb[E1];
                        EXT = media_xt[E1];
                        fc_linha('E', EXB, EXT, src, ymax, y_desloca);
                        DXB = media_xb[E1] + largura_B;
                        DXT = media_xt[E1] + largura_T;
                        fc_linha('D', DXB, DXT, src, ymax, y_desloca);
                }
                else
                {
                        DXB = Dxb_anterior;
                        DXT = Dxt_anterior;
                        fc_linha('D', DXB, DXT, src, ymax, y_desloca);
                        EXB = Exb_anterior;
                        EXT = Ext_anterior;
                        fc_linha('E', EXB, EXT, src, ymax, y_desloca);
                }
                // Função para verificar erro -------
                fc_verifica_erro();
                }               // Fim do processo III ------------------------------------
                /// --  IV  ------------------------------------------------------------------------
                if(quant_esquerda == 0 && quant_direita > 0)            // Se existe somente D1
                {
                        if(abs(media_xb[D1] - Dxb_anterior) < d)
                        {
                                DXB = media_xb[D1];
                                DXT = media_xt[D1];
                                fc_linha('D', DXB, DXT, src, ymax, y_desloca);
                                EXB = media_xb[D1] - largura_B;
                                EXT = media_xt[D1] - largura_T;
                                fc_linha('E', EXB, EXT, src, ymax, y_desloca);
                        }
                        else
                        {
                                DXB = Dxb_anterior;
                                DXT = Dxt_anterior;
                                fc_linha('D', DXB, DXT, src, ymax, y_desloca);
                                EXB = Exb_anterior;
                                EXT = Ext_anterior;
                                fc_linha('E', EXB, EXT, src, ymax, y_desloca);
                        }

                        // Função para verificar erro -------
                        fc_verifica_erro();
```

```
                                        }               // Fim do processo IV ---------------------------------------
                        }
                        else // Se estiver no inicio ou ocorreram 5 erros seguidos
                        {
                        if(quant_esquerda > 0 && quant_direita == 0 && largura_B != 0)      // Se tem E1, não tem D1 e existe largura_B
                        {
                                EXB = media_xb[E1];
                                EXT = media_xt[E1];
                                fc_linha('E', EXB, EXT, src, ymax, y_desloca);
                                DXB = media_xb[E1] + largura_B;
                                DXT = media_xt[E1] + largura_T;
                                fc_linha('D', DXB, DXT, src, ymax, y_desloca);
                        }

                        if(quant_direita > 0 && quant_esquerda == 0 && largura_B != 0)      // Se tem D1, não tem E1 e existe largura_B)
                        {
                                DXB = media_xb[D1];
                                DXT = media_xt[D1];
                                fc_linha('D', DXB, DXT, src, ymax, y_desloca);
                                EXB = media_xb[D1] - largura_B;
                                EXT = media_xt[D1] - largura_T;
                                fc_linha('E', EXB, EXT, src, ymax, y_desloca);
                        }

                        if(quant_direita > 0 && quant_esquerda > 0)              // Se tem E1 e D1.
                        {
                                EXB = media_xb[E1];
                                EXT = media_xt[E1];
                                fc_linha('E', EXB, EXT, src, ymax, y_desloca);
                                DXB = media_xb[D1];
                                DXT = media_xt[D1];
                                fc_linha('D', DXB, DXT, src, ymax, y_desloca);
                        }

                        if(quant_direita == 0 && quant_esquerda == 0 && Exb_anterior != 0)                   // Se não tem E1 nem D1 e existe Exb_anterior.
                        {
                                DXB = Dxb_anterior;
                                DXT = Dxt_anterior;
                                fc_linha('D', DXB, DXT, src, ymax, y_desloca);
                                EXB = Exb_anterior;
                                EXT = Ext_anterior;
                                fc_linha('E', EXB, EXT, src, ymax, y_desloca);
                        }
                                erro = 0;
                        }
                        largura_B = Dxb_anterior - Exb_anterior;
                        largura_T = Dxt_anterior - Ext_anterior;
}
else
{
        /// -- Se conta_grupo == 0. --- Se não existe nenhuma linha, repete a anterior
        /// --  V  ------------------------------------------------------------------------

        if(largura_B != 0 && largura_T != 0) // Se não for o primeiro quadro, imprimo as linhas anteriores
        {
                DXB = Dxb_anterior;
                DXT = Dxt_anterior;
                fc_linha('D', DXB, DXT, src, ymax, y_desloca);
                EXB = Exb_anterior;
                EXT = Ext_anterior;
                fc_linha('E', EXB, EXT, src, ymax, y_desloca);
                // Função para verificar erro -------
                fc_verifica_erro();
        }
}
```

```
/// Mostra o posicionamento do carro principal ====================================================================
        line( src,
                Point((int)((src.cols - largura_carro)/2 - paralax), ymax + y_desloca),
                Point((int)((src.cols + largura_carro)/2 - paralax),ymax + y_desloca),
                Scalar(255,255,255),
                1, CV_AA);

/// CLASSIFICAÇÃO C1 - Linha simples, linha Dupla ou Inexistente
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
                int     xmax = dst.cols;
                ///Esquerda ==========================================================================================
                /// Posição da mensagem a ESQUERDA ----------------------------------------
                float   y_intermed_esq = ((ymax + y_desloca) + 2*(y_desloca))/3; // 2/3
                float   x_intermed_esq = ((EXB) + 2*(EXT))/3;
                string  C1_esquerda = "Erro";
                float   esq_x1[100],
                        esq_x2[100],
                        esq_y1[100],
                        esq_y2[100],
                        esq_xb[100],
                        esq_xt[100];
                int     total_esq = 0;
                int     conta_subgrupo_E1;
                int     subgrupo_E1[100];
                if(quant_esquerda > 0) // Se tem linha a esquerda -----------------
                {
                        // Seleciona os segmentos que pertençam ao grupo escolhido
                        fc_segmentos_do_grupo(E1, grupo, x1, x2, y1, y2, xb, xt, esq_x1, esq_x2, esq_y1, esq_y2, esq_xb, esq_xt, n, total_esq);
                        // Separa os subgrupos
                        fc_forma_grupo (total_esq, esq_xb, largura_subgrupo, conta_subgrupo_E1, subgrupo_E1);
                        // Realiza a classificação 1 - Linha Simples ou Dupla
                        fc_classifica1("E", conta_subgrupo_E1, C1_esquerda);
                        //-----------------------------------------------------------------------------------
                        // Escreve texto na imagem
                        fc_texto(C1_esquerda, x_intermed_esq, y_intermed_esq - 13 - 30, src);
                        // Se deu tudo certo, salvo a classificação e reseto o erro
                        C1_esquerda_memoria = C1_esquerda;
                        erro_Classificacao_esquerda = 0;
                }
                else
                {
                        // Realiza a classificação 1 - Linha Simples ou Dupla
                        fc_classifica1("E", 0, C1_esquerda);
                        // Escreve texto na imagem
                        fc_texto(C1_esquerda, x_intermed_esq, y_intermed_esq - 13 - 30, src);
                }
                /// Direita ==========================================================================================
                // Posição da mensagem DIREITA
                float   y_intermed_dir = ((ymax + y_desloca) + 2*(y_desloca))/3; // 2/3
                float   x_intermed_dir = ((DXB) + 2*(DXT))/3;
                string  C1_direita = "Erro";
                float   dir_x1[100],
                        dir_x2[100],
                        dir_y1[100],
                        dir_y2[100],
                        dir_xb[100],
                        dir_xt[100];
                int     total_dir = 0;
                int     conta_subgrupo_D1;
                int     subgrupo_D1[100];
                if(quant_direita > 0) // Se tem linha a direita -----------------
                {
                        // Seleciona os segmentos que pertençam ao grupo escolhido
                        fc_segmentos_do_grupo(D1, grupo, x1, x2, y1, y2, xb, xt, dir_x1, dir_x2, dir_y1, dir_y2, dir_xb, dir_xt, n, total_dir);
```

```
                    // Separa os subgrupos
                    fc_forma_grupo (total_dir, dir_xb, largura_subgrupo, conta_subgrupo_D1, subgrupo_D1);
                    // Realiza a classificação 1 - Linha Simples ou Dupla
                    fc_classifica1("D", conta_subgrupo_D1, C1_direita);
                    //--------------------------------------------------------------------------------
                    // Escreve texto na imagem
                    fc_texto(C1_direita, x_intermed_dir, y_intermed_dir - 13 - 30, src);
                    // Se deu tudo certo, salvo a classificação e zero o erro
                    C1_direita_memoria = C1_direita;
                    erro_Classificacao_direita = 0;
          }
          else
          {
                    // Realiza a classificação 1 - Linha Simples ou Dupla
                    fc_classifica1("D", 0, C1_direita);
                    // Escreve texto na imagem
                    fc_texto(C1_direita, x_intermed_dir, y_intermed_dir - 13 - 30, src);
          }

/// CLASSIFICAÇÃO C2 - Se for Linha Simples, classifica em Contínua ou Seccionada &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
          /// Esquerda ==========================================================================================================
          string    C2_esquerda = "Erro";
          if(C1_esquerda == "Linha Simples")
          {
                    // Realiza a classificação 2 - Quando for Linha Simples, classifica em Contínua ou Seccionada
                    fc_classifica2("E", ymax, xmax, total_esq, esq_x1, esq_x2, esq_y1, esq_y2, C2_esquerda);
                    // Escreve texto na imagem
                    fc_texto(C2_esquerda, x_intermed_esq, y_intermed_esq - 13, src);
          }
          /// Direita ===========================================================================================================
          string    C2_direita = "Erro";
          if(C1_direita == "Linha Simples")
          {
                    // Realiza a classificação 2 - Quando for Linha Simples, classifica em Contínua ou Seccionada
                    fc_classifica2("D", ymax, xmax, total_dir, dir_x1, dir_x2, dir_y1, dir_y2, C2_direita);
                    // Escreve texto na imagem
                    fc_texto(C2_direita, x_intermed_dir, y_intermed_dir - 13, src);
          }
/// CLASSIFICAÇÃO C3 - Se for Linha Dupla, classifica a primeira linha em Contínua ou Seccionada &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
          /// Esquerda ==========================================================================================================

          string    C3_esquerda = "Erro";

          if(C1_esquerda == "Linha Dupla")
          {
                    // Realiza a classificação 3 - Quando for Linha Dupla, classifica a primeira linha em Contínua ou Seccionada
                    fc_classifica3("E", ymax, xmax, total_esq, esq_x1, esq_x2, esq_y1, esq_y2, subgrupo_E1, C3_esquerda);
          }
          /// Direita ===========================================================================================================

          string    C3_direita = "Erro";

          if(C1_direita == "Linha Dupla")
          {
                    // Realiza a classificação 3 - Quando for Linha Dupla, classifica a primeira linha em Contínua ou Seccionada
                    fc_classifica3("D", ymax, xmax, total_dir, dir_x1, dir_x2, dir_y1, dir_y2, subgrupo_D1, C3_direita);
          }

/// CLASSIFICAÇÃO C4 - Se for Linha Dupla, classifica a segunda linha em Contínua ou Seccionada &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

          /// Esquerda ==========================================================================================================

          string    C4_esquerda = "Erro";

          if(C1_esquerda == "Linha Dupla")
```

```
{
	// Realiza a classificação 4 - Quando for Linha Dupla, classifica a segunda linha em Contínua ou Seccionada
	fc_classifica4("E", ymax, xmax, total_esq, esq_x1, esq_x2, esq_y1, esq_y2, subgrupo_E1, conta_subgrupo_E1, C4_esquerda);
}
/// Direita ====================================================================================================
string		C4_direita = "Erro";
if(C1_direita == "Linha Dupla")
{
	// Realiza a classificação 4 - Quando for Linha Dupla, classifica a segunda linha em Contínua ou Seccionada
	fc_classifica4("D", ymax, xmax, total_dir, dir_x1, dir_x2, dir_y1, dir_y2, subgrupo_D1, conta_subgrupo_D1, C4_direita);
}
if(C1_esquerda == "Linha Dupla")
{
	/// Escreve classificação da linha dupla a esquerda ---------------------------------
	fc_texto(C3_esquerda + " - " + C4_esquerda, x_intermed_esq, y_intermed_esq - 13, src);
}
if(C1_direita == "Linha Dupla")
{
	/// Escreve classificação da linha dupla a direita ---------------------------------
	fc_texto(C3_direita + " - " + C4_direita, x_intermed_dir, y_intermed_dir - 13, src);
}
/// VISUALIZAÇÃO %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

imshow("Plota Todas as Linhas", src);

/// PREENCHIMENTO DO ARQUIVO TXT &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
if(quant_frames <= ultimo_frame_teste)
{
	string	esquerda = "-",		// "-" Significa erro
		direita = "-";		// "-" Significa erro
	// Condições a esquerda ==========================================
	if(C1_esquerda == "Linha Simples")
	{
		if(C2_esquerda == "Continua")
		{
			esquerda = "c";
		}
		if(C2_esquerda == "Seccionada")
		{
			esquerda = "s";
		}
	}
	if(C1_esquerda == "Linha Dupla")
	{
		if(C3_esquerda == "Continua" && C4_esquerda == "Continua")
		{
			esquerda = "cc";
		}
		if(C3_esquerda == "Continua" && C4_esquerda == "Seccionada")
		{
			esquerda = "cs";
		}
		if(C3_esquerda == "Seccionada" && C4_esquerda == "Continua")
		{
			esquerda = "sc";
		}
		if(C3_esquerda == "Seccionada" && C4_esquerda == "Seccionada")
		{
			esquerda = "ss";
		}
	}
	// Condições a direita ==========================================
	if(C1_direita == "Linha Simples")
	{
		if(C2_direita == "Continua")
```

```
						{
							direita = "c";
						}
						if(C2_direita == "Seccionada")
						{
							direita = "s";
						}
					}
					if(C1_direita == "Linha Dupla")
					{
						if(C3_direita == "Continua" && C4_direita == "Continua")
						{
							direita = "cc";
						}
						if(C3_direita == "Continua" && C4_direita == "Seccionada")
						{
							direita = "cs";
						}
						if(C3_direita == "Seccionada" && C4_direita == "Continua")
						{
							direita = "sc";
						}
						if(C3_direita == "Seccionada" && C4_direita == "Seccionada")
						{
							direita = "ss";
						}
					}
					// Armazena os resultados para serem gravados no arquivo .txt posteriormente ****************************************
					guarda_esquerda[quant_frames] = esquerda;
					guarda_direita[quant_frames] = direita;
				}
				else
				{
					//printf("Fim :%s Hora:%s",__DATE__,__TIME__);
					tf = clock();    // Grava momento em que parou
					ciclos = (float) (tf - t0);
					tempo_gasto =  ciclos / (float)(CLOCKS_PER_SEC);
					cout<<endl<<endl
					<<"================ ANALISE DA VELOCIDADE DE PROCESSAMENTO ================"<<endl<<endl;
					cout<<endl<<"Ciclos por segundo = "<<CLOCKS_PER_SEC<<endl;
					cout<<endl<<"Ciclos = "<<ciclos<<" Tempo gasto = "<<tempo_gasto<<" segundos"<<endl;
					cout<<endl<<"Quantidade de quadros analisados = "<<ultimo_frame_teste<<endl;
					cout<<endl<<"Tempo de processamento por quadro = "<<tempo_gasto/ultimo_frame_teste<<"s ou "
						<< 1000*tempo_gasto/ultimo_frame_teste<<"ms"<<endl;
					break; //sair da iteração
				}
				quant_frames++;
				if(waitKey(1) >= 0) break;
		// Aqui é o fim do "for" que percorre todo o video -------------------------
}
		/// PREENCHIMENTO DO ARQTIVO TXT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
		// Inicializa um manipulador de arquivos txt ------------------------------------
		ofstream saidatxt;
		saidatxt.open(nome_arquivo_saida);      // Nome e caminho definido no menu inicial
		// Preenchimento -----------------------------------------------------------------
		for( int i = 1; i < ultimo_frame_teste; i++ )
		{
			// Escreve no arquivo txt
			saidatxt << i <<" "<<guarda_esquerda[i]<<" "<<guarda_direita[i]<<endl;
		}
		// Escreve no arquivo txt sem dar o último enter
		saidatxt << ultimo_frame_teste <<" "<<guarda_esquerda[ultimo_frame_teste]<<" "<<guarda_direita[ultimo_frame_teste];
		saidatxt.close();               // Fechando o arquivo .txt
		/// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
		/// COMPARANDO COM O ARQUIVO PADRÃO %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
/// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

// Inicializa um manipulador de arquivos txt -------------------------------------
ifstream     txt_padrao,
             txt_teste;
txt_padrao.open(nome_arquivo_padrao);          // Abre arquivo txt.
txt_teste.open(nome_arquivo_saida);            // Abre arquivo txt.
string       no_frame_padrao,
             no_frame_teste,
             esq_padrao,
             esq_teste,
             dir_padrao,
             dir_teste,
             esq_status,
             dir_status;
int          esq_acerto = 0,
             esq_erro = 0,
             dir_acerto = 0,
             dir_erro = 0;
//Variáveis para a Matriz Confusão

int          c_c  = 0, c_s  = 0, c_cc = 0, c_cs = 0, c_sc = 0, c_ss = 0, c_n  = 0, s_c  = 0, s_s  = 0, s_cc = 0, s_cs = 0, s_sc = 0, s_ss = 0, s_n  = 0,
             cc_c  = 0, cc_s  = 0, cc_cc = 0, cc_cs = 0, cc_sc = 0, cc_ss = 0, cc_n  = 0, cs_c  = 0, cs_s  = 0, cs_cc = 0, cs_cs = 0, cs_sc = 0,
             cs_ss = 0, cs_n  = 0, sc_c  = 0, sc_s  = 0, sc_cc = 0, sc_cs = 0, sc_sc = 0, sc_ss = 0, sc_n  = 0, ss_c  = 0, ss_s  = 0, ss_cc = 0,
             ss_cs = 0, ss_sc = 0, ss_ss = 0, ss_n  = 0, n_c  = 0, n_s  = 0, n_cc = 0, n_cs = 0, n_sc = 0, n_ss = 0, n_n  = 0;
double       soma_linha_c, percent_linha_c, soma_linha_s, percent_linha_s, soma_linha_cc, percent_linha_cc, soma_linha_cs, percent_linha_cs,
             soma_linha_sc, percent_linha_sc, soma_linha_ss, percent_linha_ss, soma_linha_n, percent_linha_n;
while(!txt_padrao.eof())
{
             // Carrega de três em três valores -----------------------------------
             txt_padrao >> no_frame_padrao >> esq_padrao >> dir_padrao;
             txt_teste >> no_frame_teste >> esq_teste >> dir_teste;
             if(no_frame_teste == no_frame_padrao) // Se forem do mesmo frame
             {
                          // Compara as classificações da esquerda
                          if(esq_teste == esq_padrao)
                          {
                                       esq_status = "OK";
                                       esq_acerto++;
                          }
                          else
                          {
                                       esq_status = "ERRO";
                                       esq_erro++;
                          }
                          // Compara as classificações da direita
                          if(dir_teste == dir_padrao)
                          {
                                       dir_status = "OK";
                                       dir_acerto++;
                          }
                          else
                          {
                                       dir_status = "ERRO";
                                       dir_erro++;
                          }
                          // Matriz Confusão--------
                          // Linha a Esquerda
                          if(esq_padrao == "c" && esq_teste == "c")          {c_c++;}
                          if(esq_padrao == "c" && esq_teste == "s")          {c_s++;}
                          if(esq_padrao == "c" && esq_teste == "cc")         {c_cc++;}
                          if(esq_padrao == "c" && esq_teste == "cs")         {c_cs++;}
                          if(esq_padrao == "c" && esq_teste == "sc")         {c_sc++;}
                          if(esq_padrao == "c" && esq_teste == "ss")         {c_ss++;}
                          if(esq_padrao == "c" && esq_teste == "-")          {c_n++;}
```

```
if(esq_padrao == "s" && esq_teste == "c")		{s_c++;}
if(esq_padrao == "s" && esq_teste == "s")		{s_s++;}
if(esq_padrao == "s" && esq_teste == "cc")		{s_cc++;}
if(esq_padrao == "s" && esq_teste == "cs")		{s_cs++;}
if(esq_padrao == "s" && esq_teste == "sc")		{s_sc++;}
if(esq_padrao == "s" && esq_teste == "ss")		{s_ss++;}
if(esq_padrao == "s" && esq_teste == "-")		{s_n++;}

if(esq_padrao == "cc" && esq_teste == "c")		{cc_c++;}
if(esq_padrao == "cc" && esq_teste == "s")		{cc_s++;}
if(esq_padrao == "cc" && esq_teste == "cc")		{cc_cc++;}
if(esq_padrao == "cc" && esq_teste == "cs")		{cc_cs++;}
if(esq_padrao == "cc" && esq_teste == "sc")		{cc_sc++;}
if(esq_padrao == "cc" && esq_teste == "ss")		{cc_ss++;}
if(esq_padrao == "cc" && esq_teste == "-")		{cc_n++;}

if(esq_padrao == "cs" && esq_teste == "c")		{cs_c++;}
if(esq_padrao == "cs" && esq_teste == "s")		{cs_s++;}
if(esq_padrao == "cs" && esq_teste == "cc")		{cs_cc++;}
if(esq_padrao == "cs" && esq_teste == "cs")		{cs_cs++;}
if(esq_padrao == "cs" && esq_teste == "sc")		{cs_sc++;}
if(esq_padrao == "cs" && esq_teste == "ss")		{cs_ss++;}
if(esq_padrao == "cs" && esq_teste == "-")		{cs_n++;}

if(esq_padrao == "sc" && esq_teste == "c")		{sc_c++;}
if(esq_padrao == "sc" && esq_teste == "s")		{sc_s++;}
if(esq_padrao == "sc" && esq_teste == "cc")		{sc_cc++;}
if(esq_padrao == "sc" && esq_teste == "cs")		{sc_cs++;}
if(esq_padrao == "sc" && esq_teste == "sc")		{sc_sc++;}
if(esq_padrao == "sc" && esq_teste == "ss")		{sc_ss++;}
if(esq_padrao == "sc" && esq_teste == "-")		{sc_n++;}

if(esq_padrao == "ss" && esq_teste == "c")		{ss_c++;}
if(esq_padrao == "ss" && esq_teste == "s")		{ss_s++;}
if(esq_padrao == "ss" && esq_teste == "cc")		{ss_cc++;}
if(esq_padrao == "ss" && esq_teste == "cs")		{ss_cs++;}
if(esq_padrao == "ss" && esq_teste == "sc")		{ss_sc++;}
if(esq_padrao == "ss" && esq_teste == "ss")		{ss_ss++;}
if(esq_padrao == "ss" && esq_teste == "-")		{ss_n++;}

if(esq_padrao == "-" && esq_teste == "c")		{n_c++;}
if(esq_padrao == "-" && esq_teste == "s")		{n_s++;}
if(esq_padrao == "-" && esq_teste == "cc")		{n_cc++;}
if(esq_padrao == "-" && esq_teste == "cs")		{n_cs++;}
if(esq_padrao == "-" && esq_teste == "sc")		{n_sc++;}
if(esq_padrao == "-" && esq_teste == "ss")		{n_ss++;}
if(esq_padrao == "-" && esq_teste == "-")		{n_n++;}

// Matriz Confusão--------

// Linha a Direita

if(dir_padrao == "c" && dir_teste == "c")		{c_c++;}
if(dir_padrao == "c" && dir_teste == "s")		{c_s++;}
if(dir_padrao == "c" && dir_teste == "cc")		{c_cc++;}
if(dir_padrao == "c" && dir_teste == "cs")		{c_cs++;}
if(dir_padrao == "c" && dir_teste == "sc")		{c_sc++;}
if(dir_padrao == "c" && dir_teste == "ss")		{c_ss++;}
if(dir_padrao == "c" && dir_teste == "-")		{c_n++;}

if(dir_padrao == "s" && dir_teste == "c")		{s_c++;}
if(dir_padrao == "s" && dir_teste == "s")		{s_s++;}
if(dir_padrao == "s" && dir_teste == "cc")		{s_cc++;}
if(dir_padrao == "s" && dir_teste == "cs")		{s_cs++;}
```

```
                    if(dir_padrao == "s" && dir_teste == "sc")          {s_sc++;}
                    if(dir_padrao == "s" && dir_teste == "ss")          {s_ss++;}
                    if(dir_padrao == "s" && dir_teste == "-")           {s_n++;}

                    if(dir_padrao == "cc" && dir_teste == "c")          {cc_c++;}
                    if(dir_padrao == "cc" && dir_teste == "s")          {cc_s++;}
                    if(dir_padrao == "cc" && dir_teste == "cc")         {cc_cc++;}
                    if(dir_padrao == "cc" && dir_teste == "cs")         {cc_cs++;}
                    if(dir_padrao == "cc" && dir_teste == "sc")         {cc_sc++;}
                    if(dir_padrao == "cc" && dir_teste == "ss")         {cc_ss++;}
                    if(dir_padrao == "cc" && dir_teste == "-")          {cc_n++;}

                    if(dir_padrao == "cs" && dir_teste == "c")          {cs_c++;}
                    if(dir_padrao == "cs" && dir_teste == "s")          {cs_s++;}
                    if(dir_padrao == "cs" && dir_teste == "cc")         {cs_cc++;}
                    if(dir_padrao == "cs" && dir_teste == "cs")         {cs_cs++;}
                    if(dir_padrao == "cs" && dir_teste == "sc")         {cs_sc++;}
                    if(dir_padrao == "cs" && dir_teste == "ss")         {cs_ss++;}
                    if(dir_padrao == "cs" && dir_teste == "-")          {cs_n++;}

                    if(dir_padrao == "sc" && dir_teste == "c")          {sc_c++;}
                    if(dir_padrao == "sc" && dir_teste == "s")          {sc_s++;}
                    if(dir_padrao == "sc" && dir_teste == "cc")         {sc_cc++;}
                    if(dir_padrao == "sc" && dir_teste == "cs")         {sc_cs++;}
                    if(dir_padrao == "sc" && dir_teste == "sc")         {sc_sc++;}
                    if(dir_padrao == "sc" && dir_teste == "ss")         {sc_ss++;}
                    if(dir_padrao == "sc" && dir_teste == "-")          {sc_n++;}

                    if(dir_padrao == "ss" && dir_teste == "c")          {ss_c++;}
                    if(dir_padrao == "ss" && dir_teste == "s")          {ss_s++;}
                    if(dir_padrao == "ss" && dir_teste == "cc")         {ss_cc++;}
                    if(dir_padrao == "ss" && dir_teste == "cs")         {ss_cs++;}
                    if(dir_padrao == "ss" && dir_teste == "sc")         {ss_sc++;}
                    if(dir_padrao == "ss" && dir_teste == "ss")         {ss_ss++;}
                    if(dir_padrao == "ss" && dir_teste == "-")          {ss_n++;}

                    if(dir_padrao == "-" && dir_teste == "c")           {n_c++;}
                    if(dir_padrao == "-" && dir_teste == "s")           {n_s++;}
                    if(dir_padrao == "-" && dir_teste == "cc")          {n_cc++;}
                    if(dir_padrao == "-" && dir_teste == "cs")          {n_cs++;}
                    if(dir_padrao == "-" && dir_teste == "sc")          {n_sc++;}
                    if(dir_padrao == "-" && dir_teste == "ss")          {n_ss++;}
                    if(dir_padrao == "-" && dir_teste == "-")           {n_n++;}
                }
}
double tot_acertos = esq_acerto + dir_acerto;
double tot_erros = esq_erro + dir_erro;
// Mostra resultados -------------------------
cout<<endl<<endl<<"=================== COMPARACAO COM O ARQUIVO PADRAO ========================="<<endl<<endl;
cout<<endl<<"Resultado"<<endl<<endl;
cout<<"Acertos              -> "<<esq_acerto<<"         "<<dir_acerto<<"              = "<<tot_acertos<<endl;
cout<<"Erros -> "<<esq_erro<<"          "<<dir_erro<<"                  = "<<tot_erros<<endl;
cout<<endl<<"Total de testes realizados = "<<tot_acertos + tot_erros<<endl;
cout<<endl<<"Percentuais Totais"<<endl<<endl;
cout<<"Acuracia =           "<<100*(tot_acertos)/(tot_acertos + tot_erros)<<"%"<<endl;
cout<<"Erros    =           "<<100*(tot_erros)/(tot_acertos + tot_erros)<<"%"<<endl;

// Matriz de Confusao ------------------------------------------------------------------------
soma_linha_c = c_c + c_s + c_cc + c_cs + c_sc + c_ss + c_n;
if(soma_linha_c > 0){percent_linha_c = 100*c_c/soma_linha_c;}else{percent_linha_c = 0;};
soma_linha_s = s_c + s_s + s_cc + s_cs + s_sc + s_ss + s_n;
if(soma_linha_s > 0){percent_linha_s = 100*s_s/soma_linha_s;}else{percent_linha_s = 0;};
soma_linha_cc = cc_c + cc_s + cc_cc + cc_cs + cc_sc + cc_ss + cc_n;
if(soma_linha_cc > 0){percent_linha_cc = 100*cc_cc/soma_linha_cc;}else{percent_linha_cc = 0;};
```

```
soma_linha_cs = cs_c + cs_s + cs_cc + cs_cs + cs_sc + cs_ss + cs_n;
if(soma_linha_cs > 0){percent_linha_cs = 100*cs_cs/soma_linha_cs;}else{percent_linha_cs = 0;};
soma_linha_sc = sc_c + sc_s + sc_cc + sc_cs + sc_sc + sc_ss + sc_n;
if(soma_linha_sc > 0){percent_linha_sc = 100*sc_sc/soma_linha_sc;}else{percent_linha_sc = 0;};
soma_linha_ss = ss_c + ss_s + ss_cc + ss_cs + ss_sc + ss_ss + ss_n;
if(soma_linha_ss > 0){percent_linha_ss = 100*ss_ss/soma_linha_ss;}else{percent_linha_ss = 0;};
soma_linha_n = n_c + n_s + n_cc + n_cs + n_sc + n_ss + n_n;
if(soma_linha_n > 0){percent_linha_n = 100*n_n/soma_linha_n;}else{percent_linha_n = 0;};
cout<<endl<<"========================== MATRIZ DE CONFUSAO ==============================="<<endl<<endl;
//cout<<"                                       Predicao"<<endl<<endl;
cout<<"      "<<"c"<<"    "<<"s"<<"    "<<"cc"<<"   "<<"cs"<<"   "<<"sc"<<"   "<<"ss"<<"   "<<"(-)"<<"   "<<"Taxa de Rec."<<endl<<endl;
cout<<"c"<<" "<<c_c<<"   "<<c_s<<"   "<<c_cc<<"  "<<c_cs<<"  "<<c_sc<<"  "<<c_ss<<"  "<<c_n<<"   ";
            if(soma_linha_c > 0){cout<<percent_linha_c<<"%";}else{cout<<"----------";}; cout<<endl;
cout<<"s"<<" "<<s_c<<"   "<<s_s<<"   "<<s_cc<<"  "<<s_cs<<"  "<<s_sc<<"  "<<s_ss<<"  "<<s_n<<"   ";
            if(soma_linha_s > 0){cout<<percent_linha_s<<"%";}else{cout<<"----------";}; cout<<endl;
cout<<"cc"<<"               "<<cc_c<<"  "<<cc_s<<"  "<<cc_cc<<" "<<cc_cs<<" "<<cc_sc<<" "<<cc_ss<<" "<<cc_n<<"  ";
            if(soma_linha_cc > 0){cout<<percent_linha_cc<<"%";}else{cout<<"----------";}; cout<<endl;
cout<<"cs"<<"               "<<cs_c<<"  "<<cs_s<<"  "<<cs_cc<<" "<<cs_cs<<" "<<cs_sc<<" "<<cs_ss<<" "<<cs_n<<"  ";
            if(soma_linha_cs > 0){cout<<percent_linha_cs<<"%";}else{cout<<"----------";}; cout<<endl;
cout<<"sc"<<"               "<<sc_c<<"  "<<sc_s<<"  "<<sc_cc<<" "<<sc_cs<<" "<<sc_sc<<" "<<sc_ss<<" "<<sc_n<<"  ";
            if(soma_linha_sc > 0){cout<<percent_linha_sc<<"%";}else{cout<<"----------";}; cout<<endl;
cout<<"ss"<<"               "<<ss_c<<"  "<<ss_s<<"  "<<ss_cc<<" "<<ss_cs<<" "<<ss_sc<<" "<<ss_ss<<" "<<ss_n<<"  ";
            if(soma_linha_ss > 0){cout<<percent_linha_ss<<"%";}else{cout<<"----------";}; cout<<endl;
cout<<"(-)"<<"              "<<n_c<<"   "<<n_s<<"   "<<n_cc<<"  "<<n_cs<<"  "<<n_sc<<"  "<<n_ss<<"  "<<n_n<<"   ";
            if(soma_linha_n > 0){cout<<percent_linha_n<<"%";}else{cout<<"----------";}; cout<<endl;

cout<<endl<<"======= CLASSIFICAO C1  ============================================================"<<endl<<endl;

double      C1_soma_linha_simples, C1_percent_linha_simples,
            C1_soma_linha_dupla, C1_percent_linha_dupla,
            C1_soma_linha_n, C1_percent_linha_n;
C1_soma_linha_simples = c_c + c_s + s_c + s_s + c_cc + c_cs + c_sc + c_ss + s_cc + s_cs + s_sc + s_ss + c_n + s_n;
if(C1_soma_linha_simples > 0){C1_percent_linha_simples = 100*(c_c + c_s + s_c + s_s)/C1_soma_linha_simples;}else{C1_percent_linha_simples = 0;};
C1_soma_linha_dupla = cc_c + cc_s + cc_cc + cc_cs + cc_sc + cc_ss + + cc_n + cs_c + cs_s + cs_cc + cs_cs + cs_sc + cs_ss + cs_n + sc_c
                    + sc_s + sc_cc + sc_cs + sc_sc + sc_ss + sc_n + ss_c + ss_s + ss_cc + ss_cs + ss_sc + ss_ss + ss_n;
if(C1_soma_linha_dupla > 0){C1_percent_linha_dupla = 100*(cc_cc + cc_cs + cc_sc + cc_ss + cs_cc + cs_cs + cs_sc + cs_ss + sc_cc + sc_cs
                    + sc_sc + sc_ss + ss_cc + ss_cs + ss_sc + ss_ss)/C1_soma_linha_dupla;}else{C1_percent_linha_dupla = 0;};
C1_soma_linha_n = n_c + n_s + n_cc + n_cs + n_sc + n_ss + n_n;
if(C1_soma_linha_n > 0){C1_percent_linha_n = 100*n_n/C1_soma_linha_n;}else{C1_percent_linha_n = 0;};
cout<<"      "<<"simples"<<"              "<<"dupla"<<"              "<<"(-)"<<"   "<<"Taxa de Rec."<<endl<<endl;
cout<<"simples              "<<c_c + c_s + s_c + s_s<<"              "<<c_cc + c_cs + c_sc + c_ss + s_cc + s_cs + s_sc + s_ss<<"       "<<c_n + s_n<<"
"; if(C1_soma_linha_simples > 0){cout<<C1_percent_linha_simples<<"%";}else{cout<<"----------";}; cout<<endl;
cout<<"dupla"<<cc_c + cc_s + cs_c + cs_s + sc_c + sc_s + ss_c + ss_s<<"       "
            <<cc_cc + cc_cs + cc_sc + cc_ss + cs_cc + cs_cs + cs_sc + cs_ss + sc_cc + sc_cs + sc_sc + sc_ss + ss_cc + ss_cs + ss_sc + ss_ss
            <<"              "<<cc_n + cs_n + sc_n + ss_n<<"          ";
            if(C1_soma_linha_dupla > 0){cout<<C1_percent_linha_dupla<<"%";}else{cout<<"----------";}; cout<<endl;
cout<<"(-)     "<<n_c + n_s<<"              "<<n_cc + n_cs + n_sc + n_ss<<"          "<<n_n<<"    ";
            if(C1_soma_linha_n > 0){cout<<C1_percent_linha_n<<"%";}else{cout<<"----------";}; cout<<"  ";
if((C1_soma_linha_simples + C1_soma_linha_dupla + C1_soma_linha_n) > 0)
{
            cout<<"Acuracia de C1 = "<<100.0*(c_c + c_s + s_c + s_s + cc_cc + cc_cs + cc_sc + cc_ss + cs_cc + cs_cs + cs_sc + cs_ss + sc_cc
                    + sc_cs + sc_sc + sc_ss + ss_cc + ss_cs + ss_sc + ss_ss + n_n)/(C1_soma_linha_simples
                    + C1_soma_linha_dupla + C1_soma_linha_n)<<"%"<<endl;
}
else
{
            cout<<"Acuracia de C1 = Nao se aplica!"<<endl;
}
cout<<endl<<"======= CLASSIFICAO C2  ============================================================"<<endl<<endl;

double      C2_soma_linha_c, C2_percent_linha_c,
            C2_soma_linha_s, C2_percent_linha_s;
            C2_soma_linha_c = c_c + c_s;
if(C2_soma_linha_c > 0){C2_percent_linha_c = 100*c_c/C2_soma_linha_c;}else{C2_percent_linha_c = 0;};
```

```
C2_soma_linha_s = s_c + s_s;
if(C2_soma_linha_s > 0){C2_percent_linha_s = 100*s_s/C2_soma_linha_s;}else{C2_percent_linha_s = 0;};
//cout<<"                Predicao"<<endl<<endl;
cout<<"      "<<"c"<<"      "<<"s"<<"      "<<"Taxa de Rec."<<endl<<endl;
cout<<"c"<<" "<<c_c<<"     "<<c_s<<"     "; if(C2_soma_linha_c > 0){cout<<C2_percent_linha_c<<"%";}else{cout<<"-------------";}; cout<<endl;
cout<<"s"<<" "<<s_c<<"     "<<s_s<<"     "; if(C2_soma_linha_s > 0){cout<<C2_percent_linha_s<<"%";}else{cout<<"-------------";}; cout<<"          ";
if((c_c + c_s + s_c + s_s) > 0)
{
        cout<<"     Acuracia de C2 = "<<100.0*(c_c + s_s)/(c_c + c_s + s_c + s_s)<<"%"<<endl;
}
else
{
        cout<<"Acuracia de C2 = Nao se aplica!"<<endl;
}
cout<<endl<<"======= CLASSIFICAO C3 - classifica a linha esquerda da dupla ==============="<<endl<<endl; // &&&&&&&&&&&&&&&&&&&&&&

double      soma_linha_cx, percent_linha_cx, soma_linha_sx, percent_linha_sx;
soma_linha_cx = cc_cc + cc_cs + cs_cc + cs_cs + cc_sc + cc_ss + cs_sc + cs_ss;
if(soma_linha_cx > 0){percent_linha_cx = 100*(cc_cc + cc_cs + cs_cc + cs_cs)/soma_linha_cx;}else{percent_linha_cx = 0;};
soma_linha_sx = sc_cc + sc_cs + ss_cc + ss_cs + sc_sc + sc_ss + ss_sc + ss_ss;
if(soma_linha_sx > 0){percent_linha_sx = 100*(sc_sc + sc_ss + ss_sc + ss_ss)/soma_linha_sx;}else{percent_linha_sx = 0;};
cout<<"      "<<"cx"<<"     "<<"sx"<<"     "<<"Taxa de Rec."<<endl<<endl;
cout<<"cx"<<"            "<<cc_cc + cc_cs + cs_cc + cs_cs<<"     "<<cc_sc + cc_ss + cs_sc + cs_ss<<"     ";
        if(soma_linha_cx > 0){cout<<percent_linha_cx<<"%";}else{cout<<"-------------";}; cout<<endl;
cout<<"sx"<<"            "<<sc_cc + sc_cs + ss_cc + ss_cs<<"     "<<sc_sc + sc_ss + ss_sc + ss_ss<<"     ";
        if(soma_linha_sx > 0){cout<<percent_linha_sx<<"%";}else{cout<<"-------------";}; cout<<"          ";
if((soma_linha_cx + soma_linha_sx) > 0)
{
        cout<<"Acuracia de C3 = "<<100.0*(cc_cc + cc_cs + cs_cc + cs_cs + sc_sc + sc_ss + ss_sc + ss_ss)/(soma_linha_cx
                                        + soma_linha_sx)<<"%"<<endl;
}
else
{
        cout<<"Acuracia de C3 = Nao se aplica!"<<endl;
}
cout<<endl<<"======= CLASSIFICAO C4 - classifica a linha direita da dupla ================"<<endl<<endl; // &&&&&&&&&&&&&&&&&&&&&&&

double      soma_linha_xc, percent_linha_xc, soma_linha_xs, percent_linha_xs;
soma_linha_xc = cc_cc + cc_sc + sc_cc + sc_sc + cc_cs + cc_ss + sc_cs + sc_ss;
if(soma_linha_xc > 0){percent_linha_xc = 100*(cc_cc + cc_sc + sc_cc + sc_sc)/soma_linha_xc;}else{percent_linha_xc = 0;};
soma_linha_xs = cs_cc + cs_sc + ss_cc + ss_sc + cs_cs + cs_ss + ss_cs + ss_ss;
if(soma_linha_xs > 0){percent_linha_xs = 100*(cs_cs + cs_ss + ss_cs + ss_ss)/soma_linha_xs;}else{percent_linha_xs = 0;};
//cout<<"                Predicao"<<endl<<endl;
cout<<"      "<<"xc"<<"     "<<"xs"<<"     "<<"Taxa de Rec."<<endl<<endl;
cout<<"xc"<<"            "<<cc_cc + cc_sc + sc_cc + sc_sc<<"     "<<cc_cs + cc_ss + sc_cs + sc_ss<<"     ";
        if(soma_linha_xc > 0){cout<<percent_linha_xc<<"%";}else{cout<<"-------------";}; cout<<endl;
cout<<"xs"<<"            "<<cs_cc + cs_sc + ss_cc + ss_sc<<"     "<<cs_cs + cs_ss + ss_cs + ss_ss<<"     ";
        if(soma_linha_xs > 0){cout<<percent_linha_xs<<"%";}else{cout<<"-------------";}; cout<<"          ";
if((soma_linha_xc + soma_linha_xs) > 0)
{
        cout<<"Acuracia de C4 = "<<100.0*(cc_cc + cc_sc + sc_cc + sc_sc + cs_cs + cs_ss + ss_cs + ss_ss)/(soma_linha_xc
                                                + soma_linha_xs)<<"%"<<endl;
}
else
{
        cout<<"Acuracia de C4 = Nao se aplica!"<<endl;
}

// Fechando os arquivos ----------------------
txt_padrao.close();
txt_teste.close();


/// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
/// FIM DA COMPARAÇÃO COM O ARQUIVO PADRÃO %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
            /// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            cout<<endl<<endl;
            system("pause");

    return 0;
}

///%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
///                         F U N Ç Õ E S
/// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

// Verifica erros nas linhas ------------------------------------------------------------------------------------------------
void fc_verifica_erro (void){

            if(erro_frame_anterior == true)
            {
                        erro++;
            }
            else
            {
                        erro = 0;
            }
            erro_frame_anterior = true;
            if(erro > 5)
            {
                        conta_frame = 0;
            }
}

// Imprime a linha Esquerda ou Direita e guarda os valores -------------------------------------------------------
void fc_linha (char direcao, float B, float T, Mat& src, float ymax, float y_desloca){

            // Tamanho da linha = 2/3 ------------------------------------------------------------
            float y_intermed = ((ymax + y_desloca) + 2*(y_desloca))/3;
            float x_intermed = ((B) + 2*(T))/3;

            ///Opção 1 de visualização -----------------------------------------------------------
            //line( src, Point((int)B, ymax + y_desloca), Point((int)T, 0 + y_desloca), Scalar(255,0,0), 8, CV_AA);

            ///Opção 2 de visualização
            //line( src, Point((int)B, ymax + (int)y_desloca), Point((int)x_intermed, (int)y_intermed), Scalar(255,0,0), 8, CV_AA);

            if(direcao == 'E') // Guarda os valores da Esquerda ou Direita
            {
                        Exb_anterior = B;
                        Ext_anterior = T;
            }
            else if(direcao == 'D')
            {
                        Dxb_anterior = B;
                        Dxt_anterior = T;
            }
            ///Opção 3 de visualização -----------------------------------------------------------
            // Polígono

            line(       src,
                        Point((int)(B - 50),                (int)(ymax + y_desloca)),
                        Point((int)(x_intermed - 25),       (int)(y_intermed)),
                        Scalar(0,255,255), 1, CV_AA);

            line(       src,
                        Point((int)(B + 50),                (int)(ymax + y_desloca)),
                        Point((int)(x_intermed + 25),       (int)(y_intermed)),
                        Scalar(0,255,255), 1, CV_AA);
```

```
		line(		src,
				Point((int)(x_intermed + 50),		(int)(y_intermed)),
				Point((int)(x_intermed - 50),		(int)(y_intermed)),
				Scalar(0,255,255), 1, CV_AA);
}
// Escreve texto na imagem ----------------------------------------------------------------
void fc_texto (string text, int centro_x, int centro_y, Mat& imagem){

		int fontFace =  CV_FONT_HERSHEY_PLAIN;		// Opções: FONT_HERSHEY_SIMPLEX, CV_FONT_HERSHEY_PLAIN,
								// CV_FONT_HERSHEY_DUPLEX, CV_FONT_HERSHEY_COMPLEX,
								// FONT_HERSHEY_COMPLEX_SMALL
		float fontScale = 1.5;
		int espessura = 1;
		int baseline=0;
		Size textSize = getTextSize(text, fontFace, fontScale, espessura, &baseline);
		// centraliza o texto
		Point textOrg(centro_x – (textSize.width)/2, centro_y + (textSize.height)/2);
		putText(	imagem,				// Imagem
				text,				// Texto
				textOrg,			// Coordenadas
				fontFace,			// Tipo de fonte
				fontScale,			// Tamanho da fonte
				Scalar(255,255,255),		// Cor da fonte
				espessura,			// Espessura da linha
				CV_AA);				// Tipo de linha:		4 - quatro conectado;
								//				8 - oito conectado;
								//				CV_AA - antialiased line.
}
// Transforma números inteiros em string --------------------------------------------------
string intToString(int number){
		std::stringstream ss;
		ss << number;
		return ss.str();
}
// Seleciona segmentos pertencentes a um mesmo grupo --------------------------------------
void fc_segmentos_do_grupo (int E1, int grupo[], float x1[], float x2[], float y1[], float y2[], float xb[], float xt[], float novo_x1[], float novo_x2[], float novo_y1[], float novo_y2[],
float		novo_xb[], float novo_xt[], int n, int& total){
		for(int i = 0; i < n; i++)
		{
			if(grupo[i] == E1) // Seleciona os segmentos que pertencem a E1,..., ou D1
			{
				novo_x1[total] = x1[i];
				novo_x2[total] = x2[i];
				novo_y1[total] = y1[i];
				novo_y2[total] = y2[i];
				novo_xb[total] = xb[i];
				novo_xt[total] = xt[i];
				total++;
			}
		}
}
// Função para formar grupos de segmentos dado um valor de intervalo que os separem ------------------------------
void fc_forma_grupo (int total, float novo_xb[], float largura_subgrupo, int& conta_subgrupo, int subgrupo[]){
		conta_subgrupo = 1;
		subgrupo[0] = conta_subgrupo;
		for(int i = 1; i < total; i++)
		{
			if(novo_xb[i] - novo_xb[i - 1] > largura_subgrupo)		// Distancia inferior para se juntar ao grupo
			{
				conta_subgrupo++;
			}
			subgrupo[i] = conta_subgrupo;

		}
```

```
}
// Classificação 1 - Identifica se as linhas são simples ou duplas --------------------------------
// C1-V3
void fc_classifica1 (string direcao, int conta_subgrupo, string& texto){
		int soma, soma_N;
		if(direcao == "E")
		{		c1_E_t20 = c1_E_t19; c1_E_t19 = c1_E_t18; c1_E_t18 = c1_E_t17; c1_E_t17 = c1_E_t16; c1_E_t16 = c1_E_t15; c1_E_t15 = c1_E_t14;
				c1_E_t14 = c1_E_t13; c1_E_t13 = c1_E_t12; c1_E_t12 = c1_E_t11; c1_E_t11 = c1_E_t10; c1_E_t10 = c1_E_t9; c1_E_t9 = c1_E_t8;
				c1_E_t8 = c1_E_t7; c1_E_t7 = c1_E_t6; c1_E_t6 = c1_E_t5; c1_E_t5 = c1_E_t4; c1_E_t4 = c1_E_t3; c1_E_t3 = c1_E_t2; c1_E_t2 = c1_E_t1;
				c1_E_t1 = c1_E_t0;
				if(conta_subgrupo == 1 || conta_subgrupo == 2)
				{
						c1_E_t0 = -1;// -1 significa linha simples
				}
				else if(conta_subgrupo >= 3)
				{
						c1_E_t0 = 1; // +1 significa linha dupla
				}
				soma =	c1_E_t20 + c1_E_t19 + c1_E_t18 + c1_E_t17 + c1_E_t16 + c1_E_t15 + c1_E_t14 + c1_E_t13 + c1_E_t12 + c1_E_t11
				+ c1_E_t10 + c1_E_t9 + c1_E_t8 + c1_E_t7 + c1_E_t6 + c1_E_t5 + c1_E_t4 + c1_E_t3 + c1_E_t2 + c1_E_t1 + c1_E_t0;
				if(soma > 0)
				{
						texto = "Linha Dupla";
				}
				else
				{
						texto = "Linha Simples";
				}
				// Controle da falta de linha ====================================================
				if(conta_subgrupo == 0)
				{
						conta_ausencia_E++;			// Incrementa contador de ausencia de linha.
				}
				else if(conta_subgrupo == 1)
				{
						conta_ausencia_E++;			// Possível falta de linha
				}
				else
				{
						conta_ausencia_E = 0;
				}
				if(conta_ausencia_E > 5)
				{
						texto = "Erro";
				}
		}
		else if(direcao == "D")
		{
				c1_D_t20 = c1_D_t19; c1_D_t19 = c1_D_t18; c1_D_t18 = c1_D_t17; c1_D_t17 = c1_D_t16; c1_D_t16 = c1_D_t15;
				c1_D_t15 = c1_D_t14; c1_D_t14 = c1_D_t13; c1_D_t13 = c1_D_t12; c1_D_t12 = c1_D_t11; c1_D_t11 = c1_D_t10;
				c1_D_t10 = c1_D_t9; c1_D_t9 = c1_D_t8; c1_D_t8 = c1_D_t7; c1_D_t7 = c1_D_t6; c1_D_t6 = c1_D_t5; c1_D_t5 = c1_D_t4;
				c1_D_t4 = c1_D_t3; c1_D_t3 = c1_D_t2; c1_D_t2 = c1_D_t1; c1_D_t1 = c1_D_t0;
				if(conta_subgrupo <= 3)
				{
						c1_D_t0 = -1;			// -1 significa linha simples
				}
				else if(conta_subgrupo > 3)
				{
						c1_D_t0 = 1; // +1 significa linha dupla
				}
				soma =	c1_D_t20 + c1_D_t19 + c1_D_t18 + c1_D_t17 + c1_D_t16 + c1_D_t15 + c1_D_t14
				+ c1_D_t13 + c1_D_t12 + c1_D_t11 + c1_D_t10 + c1_D_t9 + c1_D_t8 + c1_D_t7
				+ c1_D_t6 + c1_D_t5 + c1_D_t4 + c1_D_t3 + c1_D_t2 + c1_D_t1 + c1_D_t0;

				if(soma > 0)
```

```
                {
                        texto = "Linha Dupla";
                }
                else
                {
                        texto = "Linha Simples";
                }
                // Controle da falta de linha ======================================================
                if(conta_subgrupo == 0)
                {
                        conta_ausencia_D++;                     // Incrementa contador de ausencia de linha.
                }
                else if(conta_subgrupo == 1)
                {
                        conta_ausencia_D++;                     // Possível falta de linha
                }
                else
                {
                        conta_ausencia_D = 0;
                }
                if(conta_ausencia_D > 5)
                {
                        texto = "Erro";
                }
        }
}

// Classificação 2 - Quando a linha é Simples, identifica se a mesma é Contínua ou Seccionada -------------------------------
// c2-v5
void fc_classifica2 (string direcao, int ymax, int xmax, int total, float pos_x1[], float pos_x2[], float pos_y1[], float pos_y2[], string& texto){
        texto = "Continua";             // Por segurança, considero que o padrão é linha contínua.
        int             soma_C2;
        // "total" representa a quantidade de segmentos à esquerda ou a direita.
        // Preciso identificar o maior_y para as comparações
        int             maior_y;
        maior_y = fc_maior_y (ymax, xmax, total, pos_x1, pos_x2, pos_y1, pos_y2);
        int             divisoes = 10; // Valor padrão de divisões
        if(maior_y < ymax)
        {
                divisoes = 7; // Poderia ser proporcional!!!
        }
        float           intervalo, yref;
        intervalo = maior_y/divisoes;
        bool            encontrou;
        int             encontrados = 0;
        int             conta_pontos_horizontais;
        for(int j = 1; j < divisoes; j++)               // Percorrendo os intervalos de y
        {
                yref = maior_y - j*intervalo;
                conta_pontos_horizontais = 0;
                encontrou = false;
                for(int i = 0; i < total; i++)  // Para cada segmento do grupo
                {
                        if(pos_y2[i] > pos_y1[i])     // Isso geralmente ocorre para segmentos a direita, com essa inclinação: (\)
                        {
                                if(pos_y2[i] > yref && yref > pos_y1[i])    // Se yref pertence ao intervalo
                                {
                                        encontrou = true;
                                        break; //sair da iteração
                                }
                        }
                        else            // Isso geralmente ocorre para segmentos a esquerda
                        {
                                if(pos_y1[i] > yref && yref > pos_y2[i])    // Se yref pertence ao intervalo
                                {
```

```
                                                //cout<<"OK esquerda"<<endl;
                                                encontrou = true;
                                                break; //sair da iteração
                                        }
                                }
                        }
                        if(encontrou == true)
                        {
                                encontrados++;
                        }
                }
                if(direcao == "E")
                {
                        c2_E_t20 = c2_E_t19; c2_E_t19 = c2_E_t18; c2_E_t18 = c2_E_t17; c2_E_t17 = c2_E_t16; c2_E_t16 = c2_E_t15;
                        c2_E_t15 = c2_E_t14; c2_E_t14 = c2_E_t13; c2_E_t13 = c2_E_t12; c2_E_t12 = c2_E_t11; c2_E_t11 = c2_E_t10;
                        c2_E_t10 = c2_E_t9; c2_E_t9 = c2_E_t8; c2_E_t8 = c2_E_t7; c2_E_t7 = c2_E_t6; c2_E_t6 = c2_E_t5; c2_E_t5 = c2_E_t4;
                        c2_E_t4 = c2_E_t3; c2_E_t3 = c2_E_t2; c2_E_t2 = c2_E_t1; c2_E_t1 = c2_E_t0;
                        if(encontrados < 0.8*divisoes)          // Agora, quero detectar as linhas seccionadas
                        {
                                c2_E_t0 = 1; //          Seccionada
                        }
                        else
                        {
                                c2_E_t0 = -1;//          Contínua
                        }
                        // ESTABILIZAÇÃO ---------------------------------------------------------
                        soma_C2 = c2_E_t10 + c2_E_t9 + c2_E_t8 + c2_E_t7 + c2_E_t6 + c2_E_t5 + c2_E_t4 + c2_E_t3 + c2_E_t2 + c2_E_t1 + c2_E_t0;
                        if(soma_C2 > 0)
                        {
                                texto = "Seccionada";
                        }
                        if(c2_E_t1 == -1 && c2_E_t0 == -1)      //
                        {
                                texto = "Continua";
                        }
                        if((c2_E_t3 + c2_E_t2 + c2_E_t1 + c2_E_t0) <= -2)     //-2
                        {
                                texto = "Continua";
                        }
                }
                else if(direcao == "D")
                {
                        c2_D_t20 = c2_D_t19; c2_D_t19 = c2_D_t18; c2_D_t18 = c2_D_t17; c2_D_t17 = c2_D_t16; c2_D_t16 = c2_D_t15;
                        c2_D_t15 = c2_D_t14; c2_D_t14 = c2_D_t13; c2_D_t13 = c2_D_t12; c2_D_t12 = c2_D_t11; c2_D_t11 = c2_D_t10;
                        c2_D_t10 = c2_D_t9; c2_D_t9 = c2_D_t8; c2_D_t8 = c2_D_t7; c2_D_t7 = c2_D_t6; c2_D_t6 = c2_D_t5;
                        c2_D_t5 = c2_D_t4; c2_D_t4 = c2_D_t3; c2_D_t3 = c2_D_t2; c2_D_t2 = c2_D_t1; c2_D_t1 = c2_D_t0;
                        if(encontrados <= 0.8*divisoes)
                        {
                                c2_D_t0 = 1; //          Seccionada
                        }
                        else
                        {
                                c2_D_t0 = -1;           //          Contínua
                        }
                        // ESTABILIZAÇÃO ---------------------------------------------------------
                        soma_C2 =   c2_D_t10 + c2_D_t9 + c2_D_t8 + c2_D_t7 + c2_D_t6 + c2_D_t5 + c2_D_t4
                                + c2_D_t3 + c2_D_t2 + c2_D_t1 + c2_D_t0;

                        if(soma_C2 > 0)
                        {
                                texto = "Seccionada";
                        }
                        if(c2_D_t1 == -1 && c2_D_t0 == -1)
                        {
                                texto = "Continua";
```

```
                }
                if((c2_D_t3 + c2_D_t2 + c2_D_t1 + c2_D_t0) <= -2)
                {
                        texto = "Continua";
                }
        }
}

// Classificação 3 - Quando a linha for Dupla, identifica se a primeira linha é Contínua ou Seccionada ------------------------------
/// Versão: c3_v1
void fc_classifica3 (string direcao, int ymax, int xmax, int total, float pos_x1[], float pos_x2[], float pos_y1[], float pos_y2[], int subgrupo[], string& texto){
        int     soma_C3;
        // Preciso identificar o maior_y para as comparações
        int     maior_y;
        maior_y = fc_maior_y (ymax, xmax, total, pos_x1, pos_x2, pos_y1, pos_y2);
        int     divisoes = 10; // Valor padrão de divisões
        if(maior_y < ymax)
        {
                divisoes = 7;
        }
        float   intervalo, yref;
        intervalo = maior_y/divisoes;
        bool    encontrou;
        int     encontrados = 0;
        for(int j = 1; j < divisoes; j++)       // Percorrendo os intervalos de y
        {
                yref = maior_y - j*intervalo;
                encontrou = false;
                for(int i = 0; i < total; i++)  // Para cada segmento do grupo
                {
                        //Se for o primeiro sugrupo --------
                        if(subgrupo[i] == 1)
                        {
                                if(pos_y2[i] > pos_y1[i])       // Isso geralmente ocorre para segmentos a direita (\)
                                {
                                        if(pos_y2[i] > yref && yref > pos_y1[i])    // Se yref pertence ao intervalo
                                        {
                                                encontrou = true;
                                                break; //sair da iteração
                                        }
                                }
                                else            // Isso geralmente ocorre para segmentos a esquerda (/)
                                {
                                        if(pos_y1[i] > yref && yref > pos_y2[i])    // Se yref pertence ao intervalo
                                        {
                                                encontrou = true;
                                                break; //sair da iteração
                                        }
                                }
                        }
                }
                if(encontrou == true)
                {
                        encontrados++;
                }
        }
        if(direcao == "E")
        {
                c3_E_t20 = c3_E_t19; c3_E_t19 = c3_E_t18; c3_E_t18 = c3_E_t17; c3_E_t17 = c3_E_t16; c3_E_t16 = c3_E_t15;
                c3_E_t15 = c3_E_t14; c3_E_t14 = c3_E_t13; c3_E_t13 = c3_E_t12; c3_E_t12 = c3_E_t11; c3_E_t11 = c3_E_t10;
                c3_E_t10 = c3_E_t9; c3_E_t9 = c3_E_t8; c3_E_t8 = c3_E_t7; c3_E_t7 = c3_E_t6; c3_E_t6 = c3_E_t5;
                c3_E_t5 = c3_E_t4; c3_E_t4 = c3_E_t3; c3_E_t3 = c3_E_t2; c3_E_t2 = c3_E_t1; c3_E_t1 = c3_E_t0;
                if(encontrados > 0.6*divisoes)
                {
                        conta_primeira_continua_esq++;
```

```
                        conta_primeira_seccionada_esq = 0;
                        c3_E_t0 = 1;                    // Contínua
                }
                else
                {
                        conta_primeira_seccionada_esq++;
                        if(conta_primeira_seccionada_esq > 5) // Cinco seccionadas seguidas
                        {
                                conta_primeira_continua_esq = 0;
                        }
                        c3_E_t0 = -1;                   // Seccionada
                }
                // ESTABILIZAÇÃO ------------------------------------------------------------
                //Novo ==========================
                soma_C3 =   c3_E_t20 + c3_E_t19 + c3_E_t18 + c3_E_t17 + c3_E_t16 + c3_E_t15 + c3_E_t14
                        + c3_E_t13 + c3_E_t12 + c3_E_t11 + c3_E_t10 + c3_E_t9 + c3_E_t8 + c3_E_t7
                        + c3_E_t6 + c3_E_t5 + c3_E_t4 + c3_E_t3 + c3_E_t2 + c3_E_t1 + c3_E_t0;
                if(soma_C3 > 0)
                {
                        texto = "Continua";
                        //cout<<"Continua"<<endl;
                }
                else
                {
                        texto = "Seccionada";
                        //cout<<"Seccionada"<<endl;
                }
        }
        else if(direcao == "D")
        {
                c3_D_t20 = c3_D_t19; c3_D_t19 = c3_D_t18; c3_D_t18 = c3_D_t17; c3_D_t17 = c3_D_t16; c3_D_t16 = c3_D_t15;
                c3_D_t15 = c3_D_t14; c3_D_t14 = c3_D_t13; c3_D_t13 = c3_D_t12; c3_D_t12 = c3_D_t11; c3_D_t11 = c3_D_t10;
                c3_D_t10 = c3_D_t9; c3_D_t9 = c3_D_t8; c3_D_t8 = c3_D_t7; c3_D_t7 = c3_D_t6; c3_D_t6 = c3_D_t5;
                c3_D_t5 = c3_D_t4; c3_D_t4 = c3_D_t3; c3_D_t3 = c3_D_t2; c3_D_t2 = c3_D_t1; c3_D_t1 = c3_D_t0;
                if(encontrados > 0.6*divisoes)
                {
                        conta_primeira_continua_dir++;
                        conta_primeira_seccionada_dir = 0;
                        c3_D_t0 = 1;                    // Contínua
                }
                else
                {
                        conta_primeira_seccionada_dir++;
                        if(conta_primeira_seccionada_dir > 5) // Cinco seccionadas seguidas
                        {
                                conta_primeira_continua_dir = 0;
                        }
                        c3_D_t0 = -1;                   // Seccionada
                }
                // ESTABILIZAÇÃO ------------------------------------------------------------
                soma_C3 =   c3_D_t20 + c3_D_t19 + c3_D_t18 + c3_D_t17 + c3_D_t16 + c3_D_t15 + c3_D_t14
                        + c3_D_t13 + c3_D_t12 + c3_D_t11 + c3_D_t10 + c3_D_t9 + c3_D_t8 + c3_D_t7
                        + c3_D_t6 + c3_D_t5 + c3_D_t4 + c3_D_t3 + c3_D_t2 + c3_D_t1 + c3_D_t0;
                if(soma_C3 > 0)
                {
                        texto = "Continua";
                }
                else
                {
                        texto = "Seccionada";
                }
        }
}

// Classificação 4 - Quando a linha for Dupla, identifica se a segunda linha é Contínua ou Seccionada ------------------------------
```

```
// c4-v2 - Melhor
void fc_classifica4 (string direcao, int ymax, int xmax, int total, float pos_x1[], float pos_x2[], float pos_y1[], float pos_y2[], int subgrupo[]
                        , int total_subgrupo_direcao, string& texto){
                int             soma_C4;
                // Preciso identificar o maior_y para as comparações
                int             maior_y;
                maior_y = fc_maior_y (ymax, xmax, total, pos_x1, pos_x2, pos_y1, pos_y2);
                int             divisoes = 10; // Valor padrão de divisões
                if(maior_y < ymax)
                {
                        divisoes = 7;
                }
                float           intervalo, yref;
                intervalo = maior_y/divisoes;
                bool            encontrou;
                int             encontrados = 0;
                for(int j = 1; j < divisoes; j++)               // Percorrendo os intervalos de y
                {
                        yref = maior_y - j*intervalo;
                        encontrou = false;
                        for(int i = 0; i < total; i++)  // Para cada segmento do grupo
                        {
                                //Se for o último sugrupo --------
                                if(subgrupo[i] == (total_subgrupo_direcao))
                                {
                                        if(pos_y2[i] > pos_y1[i])     // Isso geralmente ocorre parasegmentos a direita
                                        {
                                                if(pos_y2[i] > yref && yref > pos_y1[i])    // Se yref pertence ao intervalo
                                                {
                                                        encontrou = true;
                                                        break; //sair da iteração
                                                }
                                        }
                                        else            // Isso geralmente ocorre parasegmentos a esquerda
                                        {
                                                if(pos_y1[i] > yref && yref > pos_y2[i])    // Se yref pertence ao intervalo
                                                {
                                                        encontrou = true;
                                                        break; //sair da iteração
                                                }
                                        }
                                }
                        }
                        if(encontrou == true)
                        {
                                encontrados++;
                        }
                }
                if(direcao == "E")
                {
                        c4_E_t20 = c4_E_t19; c4_E_t19 = c4_E_t18; c4_E_t18 = c4_E_t17; c4_E_t17 = c4_E_t16; c4_E_t16 = c4_E_t15;
                        c4_E_t15 = c4_E_t14; c4_E_t14 = c4_E_t13; c4_E_t13 = c4_E_t12; c4_E_t12 = c4_E_t11; c4_E_t11 = c4_E_t10;
                        c4_E_t10 = c4_E_t9; c4_E_t9 = c4_E_t8; c4_E_t8 = c4_E_t7; c4_E_t7 = c4_E_t6; c4_E_t6 = c4_E_t5;
                        c4_E_t5 = c4_E_t4; c4_E_t4 = c4_E_t3; c4_E_t3 = c4_E_t2; c4_E_t2 = c4_E_t1; c4_E_t1 = c4_E_t0;
                        if(encontrados > 0.6*divisoes)
                        {
                                conta_segunda_continua_esq++;
                                conta_segunda_seccionada_esq = 0;
                                c4_E_t0 = 1;
                        }
                        else
                        {
                                conta_segunda_seccionada_esq++;
                                if(conta_segunda_seccionada_esq > 5) // Cinco seccionadas seguidas
                                {
```

```
                                        conta_segunda_continua_esq = 0;
                                }
                                c4_E_t0 = -1;
                        }
                        // ESTABILIZAÇÃO ------------------------------------------------------------
                        soma_C4 =   c4_E_t20 + c4_E_t19 + c4_E_t18 + c4_E_t17 + c4_E_t16 + c4_E_t15 + c4_E_t14
                                + c4_E_t13 + c4_E_t12 + c4_E_t11 + c4_E_t10 + c4_E_t9 + c4_E_t8 + c4_E_t7
                                + c4_E_t6 + c4_E_t5 + c4_E_t4 + c4_E_t3 + c4_E_t2 + c4_E_t1 + c4_E_t0;
                        if(soma_C4 > 0)
                        {
                                texto = "Continua";
                        }
                        else
                        {
                                texto = "Seccionada";
                        }
                }
                else if(direcao == "D")
                {
                        c4_D_t20 = c4_D_t19; c4_D_t19 = c4_D_t18; c4_D_t18 = c4_D_t17; c4_D_t17 = c4_D_t16; c4_D_t16 = c4_D_t15;
                        c4_D_t15 = c4_D_t14; c4_D_t14 = c4_D_t13; c4_D_t13 = c4_D_t12; c4_D_t12 = c4_D_t11; c4_D_t11 = c4_D_t10;
                        c4_D_t10 = c4_D_t9; c4_D_t9 = c4_D_t8; c4_D_t8 = c4_D_t7; c4_D_t7 = c4_D_t6; c4_D_t6 = c4_D_t5;
                        c4_D_t5 = c4_D_t4; c4_D_t4 = c4_D_t3; c4_D_t3 = c4_D_t2; c4_D_t2 = c4_D_t1; c4_D_t1 = c4_D_t0;
                        if(encontrados > 0.6*divisoes)                  // Melhorou
                        {
                                conta_segunda_continua_dir++;
                                conta_segunda_seccionada_dir = 0;
                                c4_D_t0 = 1;
                        }
                        else
                        {
                                conta_segunda_seccionada_dir++;
                                if(conta_segunda_seccionada_dir > 5) // Cinco seccionadas seguidas
                                {
                                        conta_segunda_continua_dir = 0;
                                }
                                c4_D_t0 = -1;
                        }
                        // ESTABILIZAÇÃO ------------------------------------------------------------

                        soma_C4 =   c4_D_t20 + c4_D_t19 + c4_D_t18 + c4_D_t17 + c4_D_t16 + c4_D_t15 + c4_D_t14
                                + c4_D_t13 + c4_D_t12 + c4_D_t11 + c4_D_t10 + c4_D_t9 + c4_D_t8 + c4_D_t7
                                + c4_D_t6 + c4_D_t5 + c4_D_t4 + c4_D_t3 + c4_D_t2 + c4_D_t1 + c4_D_t0;
                        if(soma_C4 > 0)
                        {
                                texto = "Continua";
                        }
                        else
                        {
                                texto = "Seccionada";
                        }
                }
}

// fc_maior_y() - Encontra o maior valor de y para utilizar nas classificações ------------------------------
int fc_maior_y (int ymax, int xmax, int total, float pos_x1[], float pos_x2[], float pos_y1[], float pos_y2[]){
                int maior_y = ymax;
                for(int i = 0; i < total; i++)    // Para cada segmento do grupo
                {
                        if(pos_x1[i] <= 3 || pos_x1[i] >= xmax - 3) // Se o ponto x1 estiver na borda
                        {
                                if(pos_y1[i] < maior_y) // Comparo com o valor guardado
                                {
                                        maior_y = pos_y1[i];
                                }
```

```
                        }
                        if(pos_x2[i] <= 3 || pos_x2[i] >= xmax - 3) // Se o ponto x2 estiver na borda
                        {
                                if(pos_y2[i] < maior_y) // Comparo com o valor guardado
                                {
                                        maior_y = pos_y2[i];
                                }
                        }
                }
                return maior_y; //valor que será retornado
}

// fc_estatistica() - Calcula a Variância, o Desvio Padrão e o Coeficiente de Variação ------------------------------
void fc_estatistica (float valores[], float media, int n, float& sigma2, float& sigma, float& CV){
                // Variáveis de entrada:
                //                              - valores;
                //                              - media;
                //                              - quantidade de valores;
                // Variáveis modificadas por referência:
                //                              - Variância;
                //                              - Desvio padrão;
                //                              - Coeficiente de Variação;
                // ---------------------------------------------------------------------
                float           somatorio = 0;
                float           dispersao;
                for( int i = 0; i < n; i++ )
                {
                        dispersao = (valores[i] - media);
                        somatorio = somatorio + dispersao*dispersao;
                }
                sigma2 = somatorio/n;
                sigma = sqrt (sigma2);
                CV = 100*sigma/media;
}

// fc_boxplot() - Calcula os valores válidos a partir da perspectiva do Boxplot --------------------------------
void fc_boxplot (float valores[], int n, float& lim_inf, float& lim_sup){
                // Variáveis de entrada:
                //              - valores;
                //              - media;
                //              - quantidade de valores;
                // Variáveis modificadas por referência:
                // ---------------------------------------------------------------------
                // Ordenação dos valores ------------------------------
                int             i, j, min;
                float           aux;
                for (i=0; i<n-1; i++) {
                        min = i;
                        for (j=i+1; j<n; j++) {
                                if (valores[j] < valores[min]) min = j;
                                }
                                aux = valores[i];
                                valores[i] = valores[min];
                        valores[min] = aux;
                }
                float           Q1, Q2, Q3;
                int             iQ1a, iQ1b, iQ2a, iQ2b, iQ3a, iQ3b;       // Índices dos quartis.
                // Encontrando o Segundo Quartil (Mediana) ------------------------------
                if(n % 2 == 0)                  // Par ------------------------
                {
                        iQ2a = n/2 - 1;
                        iQ2b = n/2;
                        Q2 = (valores[iQ2a] + valores[iQ2b])/2;
                        // Encontrando o Primeiro e o Terceiro Quartis (Mediana) ---------------
                        if((n/2) % 2 == 0)              // Par
```

```
                {
                        iQ1a = n/4;
                        iQ1b = n/4 - 1;
                        Q1 = (valores[iQ1a] + valores[iQ1b])/2;
                        iQ3a = 3*n/4;
                        iQ3b = 3*n/4 - 1;
                        Q3 = (valores[iQ3a] + valores[iQ3b])/2;
                }
                else            // Ímpar
                {
                        iQ1a = (n - 2)/4;
                        Q1 = valores[iQ1a];
                        iQ3a = (3*n - 2)/4;
                        Q3 = valores[iQ3a];
                }
        }
        else            // Ímpar ------------------------
        {
                iQ2a = (n - 1)/2;
                Q2 = valores[iQ2a];
                // Encontrando o Primeiro e o Terceiro Quartis (Mediana) ----------
                if(((n - 1)/2) % 2 == 0)        // Par
                {
                        iQ1a = (n - 1)/4;
                        iQ1b = (n - 1)/4 - 1;
                        Q1 = (valores[iQ1a] + valores[iQ1b])/2;
                        iQ3a = (3*n + 1)/4;
                        iQ3b = (3*n + 1)/4 - 1;
                        Q3 = (valores[iQ3a] + valores[iQ3b])/2;
                }
                else            // Ímpar
                {
                        iQ1a = (n - 3)/4;
                        Q1 = valores[iQ1a];
                        iQ3a = (3*n - 1)/4;
                        Q3 = valores[iQ3a];
                }
        }
        // Cálculo da Amplitude Interquartil AIQ ------------------------------------------
        float           AIQ = Q3 - Q1;
        // Calculando o Limite Inferior e Superior do Boxplot -----------------------------
        // Valores fora desses limites são considerados Valores Atípicos
        lim_inf = Q1 - 1.5*AIQ;
        lim_sup = Q3 + 1.5*AIQ;
}
```