UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
DEPARTAMENTO DE ENGENHARIA DE TRANSPORTES

LEANDRO ARAB MARCOMINI

Identificação Automática do Comportamento do Tráfego
a Partir de Imagens de Vídeo

São Carlos

2018

LEANDRO ARAB MARCOMINI

**IDENTIFICAÇÃO AUTOMÁTICA DO COMPORTAMENTO DO TRÁFEGO A PARTIR
DE IMAGENS DE VÍDEO**

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Transportes da Escola de Engenharia de São Carlos da Universidade de São Paulo como parte dos requisitos para obtenção do título de “Mestre em Ciências”.

Programa: Engenharia de Transportes.

Área de concentração: Planejamento e Operação dos Sistemas de Transportes.

Orientador:

Prof. Dr. André Luiz Barbosa Nunes da Cunha

São Carlos

2018

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

M321i

Marcomini, Leandro Arab

Identificação automática do comportamento do tráfego a partir de imagens de vídeo / Leandro Arab Marcomini; orientador André Luiz Barbosa Nunes da Cunha. São Carlos, 2018.

Dissertação (Mestrado) - Programa de Pós-Graduação em Engenharia de Transportes e Área de Concentração em Planejamento e Operação de Sistemas de Transporte --
Escola de Engenharia de São Carlos da Universidade de São Paulo, 2018.

1. segmentação de veículos. 2. rastreamento de features. 3. remoção de background. 4. modelagem de background. 5. MOG2. 6. OpenCV. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE JULGAMENTO

Candidato: Bacharel **LEANDRO ARAB MARCOMINI**.

Título da dissertação: "Identificação automática do comportamento do tráfego a partir de imagens de vídeo".

Data da defesa: 10/08/2018.

Comissão Julgadora:

Resultado:

Prof. Dr. **André Luiz Barbosa Nunes da Cunha**
(Orientador)
(Escola de Engenharia de São Carlos/EESC)

APROVADO

Prof. Associado **Adilson Gonzaga**
(Escola de Engenharia de São Carlos/EESC)

APROVADO

Dr. **Flávio José Craveiro Cunto**
(Universidade Federal do Ceará/UFC)

APROVADO

Coordenadora do Programa de Pós-Graduação em Engenharia de Transportes:
Prof^a. Associada **Ana Paula Camargo Larocca**

Presidente da Comissão de Pós-Graduação:
Prof. Associado **Luís Fernando Costa Alberto**

AGRADECIMENTOS

Agradeço à minha mãe por todos os esforços e os auxílios em me guiar ao longo dessa jornada, por todos os sonhos que teve de abrir mão para que eu pudesse alcançar os meus próprios e por todas as comidas incríveis inventadas nessas últimas décadas de convivência.

Agradeço à Jéssica, minha irmã, parceira de ideias mirabolantes e de dominação mundial, que um dia há de se realizar! Também agradeço por voltar para casa perto do Natal, porque senão não teria Chocotone aqui nunca...

Também agradeço à minha namorada Mariana Shu Yu Chang, que de alguma forma olhou para mim e gostou do que viu (precisa de oftalmologista, com certeza...), que consegue aceitar minhas estranhezas e consegue ser exatamente o encaixe perfeito nessa vida para mim. Espero que eu consiga ser para você uma fração do que você é para mim. Agradeço, também, à sua irmã, Marquesa, pelos milhares de planos bem-sucedidos de incluir batata frita em nosso cardápio.

A todo o restante da minha família, avós, tios e primos, que de várias maneiras únicas e especiais contribuíram para que eu me tornasse o que sou hoje, meu muito obrigado.

Agradeço ao Xico, que me ajudou na lógica de programação usada em parte essencial deste trabalho e que, de 20 em 20 minutos, tinha o cuidado e carinho em dar ALT-TAB no que estava fazendo, só para me mandar trabalhar. Aproveito esse espaço para mandá-lo ler a quest.

Agradeço aos meus amigos de longa (muito longa) data, que sempre fizeram questão de exaltar como minha situação era desesperadora, que sempre tiveram do meu lado para lembrar e apontar os erros, que sempre tiveram gosto em me matar virtualmente e fazer com que descobrisse meus limites em questões de paciência e nervosismo. Obrigado.

A todos amigos do departamento que conviveram comigo e me auxiliaram em meu desenvolvimento pessoal, que me viram por mais do que quarenta horas semanais durante TODO o tempo do Mestrado, muito obrigado.

Agradeço ao meu orientador Prof. Dr. André Luiz Cunha, pela amizade, pelas palavras de incentivo, pelas oportunidades que me concedeu, e também pela paciência dispendida ao nosso trabalho.

A Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES, pela bolsa de estudo concedida.

Ao Departamento de Engenharia de Transportes e a Escola de Engenharia de São Carlos – STT/EESC/USP, pela oportunidade de desenvolver este trabalho.

Aos professores e funcionários do Departamento de Engenharia de Transportes, pela amizade, pelo auxílio e pelos conhecimentos fornecidos.

Em especial ao Instituto de Ciências Matemáticas e de Computação – ICMC/USP, por ser minha eterna *Alma mater*, o local onde me formei em Ciências de Computação.

Agradeço, do fundo do meu coração, a tecnologia, pois sem ela, nada deste trabalho poderia ter sido feito.

LISTA DE FIGURAS

Figura 1 – Imagem digital: (a) matriz de pixels (b) padrão de representação dos eixos	26
Figura 2 – Operações básicas (a) Subtração de 100 (b) Imagem original (c) Adição de 100 ..	27
Figura 3 – Vídeos em perspectiva	31
Figura 4 – Vídeos frontais	31
Figura 5 – Vídeo com posição lateral	32
Figura 6 – (a) <i>Frame</i> 1 (b) <i>Frame</i> 10 (c) Subtração entre <i>Frames</i>	35
Figura 7 – (a) <i>Frame</i> 10 (b) <i>Frame</i> de <i>background</i> (c) Subtração de fundo.....	35
Figura 8 – Fluxograma do módulo de avaliação de desempenho.....	36
Figura 9 – <i>Frames</i> originais dos vídeos em perspectiva e <i>ground-truth</i>	37
Figura 10 – <i>Frames</i> originais dos vídeos frontais e <i>ground-truth</i>	38
Figura 11 – <i>Frame</i> original do vídeo lateral e <i>ground-truth</i>	39
Figura 12 – Construtor do algoritmo GMG, com valores padrão	40
Figura 13 – Construtor do algoritmo MOG, com valores padrão	41
Figura 14 – Construtor do algoritmo MOG2, com valores padrão	42
Figura 15 – <i>Frames</i> resultantes de cada fase do processo de avaliação	43
Figura 16 – Exemplo de imagens (a) Gerada pelo modelo (b) <i>Ground-truth</i>	44
Figura 17 – <i>Frames</i> resultantes dos métodos de remoção de <i>background</i>	46
Figura 18 – Histogramas das Taxas de Exatidão.....	47
Figura 19 – Histogramas das Taxas de Precisão	49
Figura 20 – Tempo de processamento dos três algoritmos	51
Figura 21 – Função do detector de Harris	59
Figura 22 – Função que implementa o detector de Shi-Tomasi	60
Figura 23 – Construtor do algoritmo SIFT, com valores de parâmetros padrão	61
Figura 24 – Construtor do algoritmo SURF, com valores padrão.....	62
Figura 25 – Funcionamento do algoritmo (FAST ALGORITHM FOR DETECTION, 2014)	63
Figura 26 – Construtor do algoritmo FAST, com valores padrão	64
Figura 27 – Parâmetros do algoritmo de fluxo ótico de Lucas-Kanade	65
Figura 28 – Parâmetros da primeira função da transformada.....	68
Figura 29 – Parâmetros da segunda função usada na transformação de perspectiva	68
Figura 30 – Fluxograma da detecção e <i>tracking</i> de <i>features</i>	69
Figura 31 – (a) ROI no vídeo da SP-310 (b) ROI no vídeo da BR-116	70

Figura 32 – ROI e Transformadas (a) SP-310 (b) BR-116	71
Figura 33 – Função de <i>threshold</i> e seus parâmetros	72
Figura 34 – Resultado da variação do valor do parâmetro <i>thresh</i>	72
Figura 35 – Remoção de <i>background</i> e <i>threshold</i> (1) ROI no <i>frame</i> original (2) Resultado ..	73
Figura 36 – Divisão da ROI (I) Detecção (II) <i>Tracking</i> (III) Salvamento	74
Figura 37 – Resultado do <i>tracking</i> nos vídeos em perspectiva	76
Figura 38 – Resultado do <i>tracking</i> nos vídeos frontais, na Av. Colturato	77
Figura 39 – Resultado do <i>tracking</i> nos vídeos frontais, na SP-310	78
Figura 40 – Resultado do <i>tracking</i> no vídeo lateral, na SP-310	78
Figura 41 – <i>Frames</i> de exemplo do desempenho do MOG2 com aplicação de <i>threshold</i>	80
Figura 42 – Veículo com falha de detecção por falta de <i>features</i>	81
Figura 43 – Detecções múltiplas de um mesmo veículo pesado.....	81
Figura 44 – Detecções múltiplas em veículos pesados	82
Figura 45 – Agrupamento de vários veículos em um só.....	83
Figura 46 – Diagrama Espaço-Tempo	86
Figura 47 – Fluxograma do método de extração de informações	88
Figura 48 – Exemplo de arquivo de saída, com informações resumidas	89
Figura 49 – Arquivo de saída detalhado	90
Figura 50 – <i>Frame</i> original, com ROI destacada, e diagrama espaço-tempo gerado	92
Figura 51 – Distribuições das duas amostras de velocidade: do radar-pistola e do algoritmo	94
Figura 52 – Distribuição dos erros relativos ocorridos durante as detecções	95
Figura 53 – Distribuição dos erros ocorridos durante as detecções, sem veículos pesados	96
Figura 54 – Primeiros 27 minutos do vídeo da BR-116 e suas contagens.....	97
Figura 55 – Últimos 28 minutos do vídeo da BR-116 e suas contagens	97
Figura 56 – Primeiros 27 minutos da BR-116 (#1) e suas contagens, sem veículos pesados..	98
Figura 57 – Últimos 28 minutos da BR-116 (#1) e suas contagens, sem veículos pesados	98
Figura 58 – Comparação das contagens no vídeo #2, da rodovia SP-310	99
Figura 59 – Comparação das contagens do vídeo #2, da SP-310, sem veículos pesados.....	100
Figura 60 – Comparação das contagens do vídeo #3, Av. Colturato	100
Figura 61 – Comparação das contagens do vídeo #3, Av. Colturato, sem veículos pesados	101
Figura 62 – Comparação das contagens do vídeo #5, da Av. Colturato	102
Figura 63 – Comparação das contagens do vídeo #5, Av. Colturato, sem veículos pesados	102
Figura 64 – Comparação das contagens no vídeo #4, SP-310	103
Figura 65 – Comparação das contagens no vídeo #4, SP-310, sem veículos pesados.....	104

Figura 66 – Comparação das contagens do vídeo #6, da SP-310.....	104
Figura 67 – Comparação das contagens no vídeo #6, SP-310, sem veículos pesados	105
Figura 68 – Comparação das contagens no vídeo #7 SP-310.....	106
Figura 69 – Comparação das contagens no vídeo #7, SP-310, sem veículos pesados	106
Figura 70 – Dimensão maior que a ROI e oclusão de veículo	109

LISTA DE TABELAS

Tabela 1 – Características dos vídeos utilizados	32
Tabela 2 – Matriz de Contingência e métricas comumente calculadas	44
Tabela 3 – Taxas de FPS para cada teste	51
Tabela 4 – Velocidades medidas com radar-pistola e geradas pelo algoritmo.....	93
Tabela 5 – Resumo dos resultados das contagens	107

SUMÁRIO

AGRADECIMENTOS	9
LISTA DE FIGURAS.....	11
LISTA DE TABELAS	15
SUMÁRIO	17
RESUMO.....	17
ABSTRACT	19
1 INTRODUÇÃO	21
1.1 CONTEXTO	22
1.2 OBJETIVO	23
1.3 JUSTIFICATIVA	23
1.4 ESTRUTURA DA DISSERTAÇÃO	24
2 CONCEITOS GERAIS E FERRAMENTAS	25
2.1 PROCESSAMENTO DE IMAGENS	25
2.2 IMAGEM E VÍDEO DIGITAIS	26
2.3 OPERAÇÃO EM IMAGENS.....	27
2.4 FERRAMENTAS COMPUTACIONAIS	27
2.4.1 PYTHON	28
2.4.2 OPENCV	29
2.5 COLETA DE DADOS.....	30
3 DETECÇÃO DE VEÍCULOS	33
3.1 SUBTRAÇÃO ENTRE <i>FRAMES</i>	34
3.2 MÉTODO PROPOSTO	36
3.2.1 <i>GROUND-TRUTH</i>	37
3.2.2 AVALIAÇÃO DE ALGORITMOS DE MODELAGEM DE <i>BACKGROUND</i>	39
3.2.2.1 GMG	39

3.2.2.2 <i>MIXTURE OF GAUSSIANS</i>	40
3.2.2.3 <i>MIXTURES OF GAUSSIANS 2</i>	41
3.2.2.4 <i>AVALIAÇÃO DO DESEMPENHO</i>	42
3.3 <i>RESULTADOS OBTIDOS</i>	45
3.3.1 <i>TAXA DE EXATIDÃO</i>	46
3.3.2 <i>TAXA DE PRECISÃO</i>	48
3.3.3 <i>TEMPO DE PROCESSAMENTO</i>	50
3.4 <i>CONCLUSÕES</i>	52
4 <i>TRACKING DE VEÍCULOS EM IMAGENS</i>	55
4.1 <i>ALGORITMOS DE DETECÇÃO DE FEATURES</i>	58
4.1.1 <i>HARRIS CORNER DETECTION</i>	58
4.1.2 <i>SHI-TOMASI CORNER DETECTION</i>	59
4.1.3 <i>SIFT</i>	61
4.1.4 <i>SURF</i>	62
4.1.5 <i>FAST</i>	63
4.2 <i>TRACKING DE FEATURES</i>	64
4.3 <i>AGRUPAMENTO DE FEATURES</i>	66
4.4 <i>TRANSFORMAÇÃO DE PERSPECTIVA</i>	68
4.5 <i>MÉTODO PROPOSTO</i>	69
4.5.1 <i>ROI E TRANSFORMAÇÃO DE PERSPECTIVA</i>	70
4.5.2 <i>REMOÇÃO DE BACKGROUND</i>	71
4.5.3 <i>TRACKING</i>	73
4.6 <i>RESULTADOS OBTIDOS</i>	74
4.7 <i>CONCLUSÕES</i>	79
5 <i>IDENTIFICAÇÃO DO COMPORTAMENTO</i>	85
5.1 <i>DIAGRAMA ESPAÇO-TEMPO</i>	86
5.2 <i>MÉTODO PROPOSTO</i>	88

5.2.2 RELATÓRIO AGREGADO	89
5.2.3 RELATÓRIO DETALHADO	90
5.2.4 DIAGRAMA ESPAÇO-TEMPO	91
5.3 RESULTADOS	91
5.3.1 DIAGRAMA ESPAÇO-TEMPO	91
5.3.2 VELOCIDADE	93
5.3.3 CONTAGEM	96
5.4 CONCLUSÕES.....	107
6 CONCLUSÕES E SUGESTÕES	111
6.1 CONCLUSÕES.....	111
6.2 SUGESTÕES PARA TRABALHOS FUTUROS	113
REFERÊNCIAS BIBLIOGRÁFICAS	115

MARCOMINI, L. A. Identificação Automática do Comportamento do Tráfego a Partir de Imagens de Vídeo. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2018.

RESUMO

Este trabalho tem por objetivo propor um sistema computacional automático capaz de identificar, a partir de imagens de vídeos, o comportamento do tráfego veicular rodoviário. Todos os códigos gerados foram escritos em Python, com o uso da biblioteca OpenCV. O primeiro passo do sistema proposto foi remover o *background* do *frame* do vídeo. Para isso, foram testados três métodos disponíveis no OpenCV, com métricas baseadas em uma Matriz de Contingência. O MOG2 foi escolhido como melhor método, processando 64 FPS, com mais de 95% de taxa de exatidão. O segundo passo do sistema envolveu detectar, rastrear e agrupar *features* dos veículos em movimento. Para isso, foi usado o algoritmo de Shi-Tomasi, junto com funções de fluxo ótico para o rastreamento. No agrupamento, usou-se a distância entre os pixels e as velocidades relativas de cada *feature*. No passo final, foram extraídos tanto as informações microscópicas quanto as informações macroscópicas em arquivos de relatório. Os arquivos têm padrões definidos, salvos em CSV. Também foi gerado, em tempo de execução, um diagrama espaço-tempo. Desse diagrama, é possível extrair informações importantes para as operações de sistemas de transportes. A contagem e a velocidade dos veículos foram usadas para validar as informações extraídas, comparadas a métodos tradicionais de coletas. Na contagem, o erro médio em todos os vídeos foi de 12,8%. Na velocidade, o erro ficou em torno de 9,9%.

Palavras chaves: segmentação de veículos; rastreamento de *features*; remoção de *background*; modelagem de *background*; MOG2; OpenCV.

MARCOMINI, L. A. Automatic Identification of Traffic Behavior Using Video Images. Thesis (Master Degree) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2018.

ABSTRACT

The objective of this research is to propose an automatic computational system capable to identify, based on video images, traffic behavior on highways. All written code was made using Python, with the OpenCV library. The first step of the proposed system is to subtract the background from the frame. We tested three different background subtraction methods, using a contingency table to extract performance metrics. We decided that MOG2 was the best method for this research, processing frames at 64 FPS and scoring more than 95% on accuracy rate. The second step of the system was to detect, track and group all moving vehicle features. We used Shi-Tomasi detection method with optical flow to track features. We grouped features with a mixture of distance between pixels and relative velocity. For the last step, the algorithm exported microscopic and macroscopic information on CSV files. The system also produced a space-time diagram at runtime, in which it was possible to extract important information to transportation system operators. To validate the information extracted, we compared vehicle counting and velocities with traditional extraction methods. The algorithm had a mean error rate of 12.8% on counting vehicles, while achieving 9.9% error rate in velocity.

Keywords: vehicle segmentation; feature tracking; background subtraction; background modeling; MOG2; OpenCV.

1 INTRODUÇÃO

A geração e a análise de dados estão na base da evolução de qualquer ramo da ciência. Dados são importantes pois embasam a tomada de decisões não só de cientistas, mas também de operadores e administradores de sistemas em todo mundo. É esperado que, até o ano de 2020, a quantidade de dados gerados pela humanidade atinja a marca dos 40.000 *Exabytes* (1 *Exabyte* = 10^{18} Bytes). Desta quantidade, somente uma porção pequena será analisada (GANTZ; REINSEL, 2012).

No campo da Engenharia de Transportes não é diferente. De fato, dados ajudam no planejamento, no desenvolvimento e na operação de projetos, refletindo em sistemas de transportes mais eficientes economicamente e energeticamente, impactando de modo positivo a vida de milhões de pessoas.

Para a aquisição de dados de tráfego são necessárias coletas em campo, podendo ser de modo manual ou automático. Nas coletas manuais, os eventos são observados em tempo real e marcados por meio de contadores ou anotados para análise posterior. Esse tipo de coleta está sujeito a várias interferências intrínsecas aos seres humanos, como o tempo que um ser humano consegue manter sua atenção a uma tarefa qualquer. Esse tempo costuma ser muito variável, indo de segundos a até horas, dependendo da motivação da pessoa (CORNISH; DUKETTE, 2009). De acordo com o DNIT (2006), um observador é capaz de contar e classificar veículos em um volume de até 350 veic/hora.sentido.

As coletas automáticas utilizam dispositivos com algum tipo de sensor. Segundo Leduc (2008), existem basicamente três tipos de sensores: (1) os instalados no pavimento e, portanto, necessitam de cortes para sua instalação ou manutenção – piezelétricos, pneumáticos ou magnéticos; (2) os remotos, instalados na lateral ou acima das faixas de rodagem, baseados em luz infravermelha, ondas de radar ou sonoras, campo magnético, ou imagens; e (3) instalados nos próprios veículos, baseados em GPS.

Sensores instalados no pavimento possuem manutenção alta, devido ao desgaste dos veículos que passam por cima deles, danificando o material. Dessa maneira, são particularmente sensíveis a veículos pesados.

Sensores remotos exigem menos manutenção em geral e, nesta categoria, o sistema de monitoramento por circuito fechado de televisão (CFTV) é amplamente usado em diversas

localidades para a segurança dos usuários das vias. Porém, o monitoramento ainda depende de operadores assistindo às transmissões das imagens e reportando situações perigosas. Poucos sistemas possuem automatização, seja na emissão de alertas, detecção de anomalias ou nas contagens de veículos.

Da automatização do sistema de transporte surge a definição de ITS (*Intelligent Transportation Systems*), englobando aplicações avançadas que fornecem serviços em sistemas de transporte, seja na informação ao usuário ou aos gerenciadores do sistema (PARLIAMENT; UNION, 2010). Técnicas de visão computacional e processamento de imagens também são ferramentas importantes na área de ITS e que vêm sendo usadas cada vez com mais frequência (CYGANNEK; GRUSZCZYŃSKI, 2014; FAGNANT; KOCKELMAN, 2015; KOCH, 2015; LIMA, 2017).

A combinação dessas ferramentas é capaz de desenvolver sistemas automáticos para analisar vídeos de tráfego veicular e identificar comportamentos. Assim, é possível também extrair dados automaticamente somente com o uso de câmeras instaladas nas rodovias, sem a necessidade de instalação de sensores no pavimento. Com o baixo custo de câmeras e equipamentos computacionais (NIXON, 2008), é possível montar sistemas para obter esses dados e analisá-los também em tempo real.

Por ser um campo que envolve computação, as técnicas de Processamento de Imagens estão em constante adaptação. Os desafios atuais na área de Transportes se concentram em obter dados com qualidade, minimizando os erros em várias situações distintas de luminosidade e de clima, assim como a aplicação em tempo real.

1.1 CONTEXTO

Este trabalho faz parte de uma linha de pesquisa coordenada pelo Professor Doutor André Luiz Barbosa Nunes da Cunha, no Departamento de Transportes da Escola de Engenharia de São Carlos, na USP, em Sistemas Inteligentes de Transportes (ITS). Essa linha de pesquisa tem como objetivo extrair características e comportamentos do tráfego veicular automaticamente a partir de imagens de vídeos, de forma a complementar coletas de dados feitas, atualmente, de forma manual. Este trabalho é complementar ao trabalho de Cunha (2013), com a investigação de métodos e algoritmos alternativos, além da implementação de código em uma nova linguagem de programação.

1.2 OBJETIVO

O objetivo principal deste trabalho é propor um sistema computacional automático capaz de identificar, a partir de imagens de vídeo, o comportamento do tráfego veicular rodoviário, com regime de fluxo ininterrupto. Para atingir tal objetivo, são necessários os seguintes objetivos específicos:

- (1) Detectar veículos nas imagens fornecidas pelas câmeras, analisando comparativamente o desempenho das principais estratégias de detecção para fluxo ininterrupto;
- (2) Rastrear os veículos durante toda sua trajetória dentro da área de interesse na imagem;
- (3) Extrair o comportamento e informações do tráfego, tais como velocidade dos veículos, contagem veicular e o diagrama espaço-tempo;
- (4) Validar as informações geradas pelo algoritmo, comparando-as com valores obtidos em campo.

Como produto final, espera-se criar um método que seja capaz de extrair as informações necessárias e implementá-lo.

1.3 JUSTIFICATIVA

A extração de dados a partir de imagens de vídeo é importante pois subsidia o trabalho de planejamento e de operação de qualquer sistema de transporte. Mais importante ainda, é a automatização da coleta desses mesmos dados.

A disponibilidade de equipamentos de monitoramento nas rodovias é uma oportunidade que não pode ser ignorada. É necessária a criação de algoritmos que consigam extrair dados automaticamente, pois é fato conhecido que faltam dados, mas não pessoas dispostas a analisá-los. Em trabalhos anteriores, a coleta de vídeos e a subsequente análise para extração de dados foi um processo demorado e árduo, como pode ser visto nos trabalhos de Araújo (2007) e Cunha (2007). Esse processo poderia ser delegado à um sistema automático.

Além da facilidade e mobilidade de se usar câmeras, em monitoramento rodoviário algumas já estão instaladas, todas as ferramentas computacionais necessárias estão disponíveis, sem custos. São bibliotecas computacionais *open source*, com o uso de linguagens de programação livres. Essas ferramentas são extremamente poderosas pois, além de contar com

atualizações constantes, possuem uma comunidade ativa de desenvolvedores dispostos a ajudar uns aos outros.

Portanto, um sistema construído sobre essa base de conhecimento, com baixo custo de implementação, móvel para ser instalado em qualquer local, que consiga identificar o comportamento do tráfego em situações distintas, capaz de funcionar em computadores até mesmo de baixo desempenho (computadores pessoais), é necessário e importante para o avanço da ciência na área de Engenharia de Transportes.

1.4 ESTRUTURA DA DISSERTAÇÃO

Esta dissertação está dividida em 6 capítulos, com o primeiro capítulo tendo a função de introduzir o assunto, além de mostrar quais são os objetivos a serem alcançados, e uma justificativa da necessidade de se fazer pesquisas na área na qual se está inserida.

No segundo capítulo, conceitos básicos utilizados neste trabalho sobre Visão Computacional e Operações em Imagens são apresentados, além de uma explicação sobre as ferramentas computacionais que tornaram este trabalho possível, OpenCV e Python.

O capítulo três traz informações sobre segmentação de veículos em vídeos. Apresenta também todos os testes que foram realizados para a escolha do melhor modelo dentre os disponíveis na biblioteca OpenCV, em Python, junto com uma conclusão sobre os resultados.

Seguindo a segmentação, o capítulo quatro discorre sobre detecção e *tracking* de *features*, usando o método escolhido. Também mostra resultados da aplicação do método e traz conclusões sobre o assunto.

No capítulo 5, este trabalho se concentra na identificação e extração do comportamento dos veículos, com uma comparação das informações extraídas pelo método proposto e das informações medidas por métodos mais tradicionais.

No sexto e último capítulo são apresentadas as conclusões gerais deste trabalho e sugestões de pesquisas futuras.

2 CONCEITOS GERAIS E FERRAMENTAS

Neste capítulo são apresentadas definições básicas sobre Visão Computacional e Processamento de Imagens, temas que fazem parte da fundamentação deste trabalho. Como se trata de áreas muito amplas, somente conceitos-chave para este trabalho são explicados. Ao final deste capítulo, são apresentadas as ferramentas utilizadas nesta pesquisa para a implementação de todos os códigos computacionais.

2.1 PROCESSAMENTO DE IMAGENS

As definições da área de Processamento de Imagens vêm mudando nos últimos anos. Jain (1989) disse que a área se resume a todo processamento que envolva imagens bidimensionais feitos por um computador digital. Também afirma que, numa definição mais ampla, pode-se considerar todo processamento de quaisquer tipos de dados bidimensionais. Já Vernon (1991) afirma que Processamento de Imagem Digital engloba qualquer técnica que faça alguma transformação, codificação e transmissão de uma imagem digital. Por outro lado, Jähne (2005) limita a área a ferramentas e processos que manipulam e analisam imagens digitais. Velho, Frery e Gomes (2008) descreveram que, no campo de Processamento de Imagem Digital, a entrada do processo é composta de uma imagem já formada, a qual é processada para gerar outra imagem como saída. Enquanto, Gonzalez e Woods (2008) afirmam que qualquer processamento de imagens digitais através de um computador digital é considerado como uma técnica dentro dessa área. Recentemente, Sonka, Hlavac e Boyle (2014) dividiram a área em: (1) processamento de baixo-nível, na qual técnicas que usam pouco ou nenhum conhecimento sobre o conteúdo da imagem são usadas; e (2) processamento de alto-nível, baseadas em conhecimento, onde técnicas de inteligência artificial são aplicadas. Russ e Russ, em 2017, voltam à definição de Velho, Frery e Gomes de 2008, de que processamento digital envolve qualquer método que processe imagens e gere uma imagem como resultado.

Para este trabalho, a definição mais geral de processamento de imagens usada por Gonzalez e Woods (2008) em seu livro é mais do que suficiente, pois engloba todos os processos aqui utilizados. Outras técnicas de processamento de imagens existem, como técnicas ópticas e analógicas, mas estão fora do escopo deste trabalho.

2.2 IMAGEM E VÍDEO DIGITAIS

Uma imagem digital pode ser definida como sendo um mapa de *bits* (*bitmap*, em inglês), uma representação da imagem real na forma de uma matriz (VELHO; FRERY; GOMES, 2008). Cada par de coordenadas (x,y) denota um ponto na imagem, seguindo o padrão de representação de eixos mostrado na Figura 1, e corresponde à cada um dos elementos a_{ij} de uma matriz. Cada elemento desta matriz é chamado de pixel, a menor unidade possível de uma imagem digital.

Os pixels da imagem dada no exemplo na Figura 1 assumem valores dentro de um intervalo discreto, de números inteiros. Os valores que se aproximam mais do limite inferior do intervalo representam cores mais escuras, como o preto ou o cinza escuro, enquanto valores mais próximos do limite superior representam cores mais claras, como o branco ou o cinza claro. Por ser uma imagem que só possui informações sobre variações de cinzas, ela é classificada como uma imagem em escala de cinzas.

Figura 1 – Imagem digital: (a) matriz de pixels (b) padrão de representação dos eixos

Um arquivo de vídeo é composto por uma sequência de imagens digitais, denominadas *frames*. Para criar a ilusão de um movimento contínuo, os vídeos mostram vários *frames* sucessivos em taxas denominadas de *Frames Per Second* (FPS). Essa taxa simboliza a quantidade de imagens que são mostradas a cada segundo de vídeo. Cada vídeo pode possuir uma taxa de FPS diferente, podendo ir de 16 a 24 FPS em filmes mais antigos (NEUMEYER, 2013) a até 300 FPS em transmissões de televisão ao vivo mais recentes, testadas pela BBC (ARMSTRONG *et al.*, 2009). Em dispositivos de gravação de câmera lenta, a captura do vídeo pode chegar a até 4,4 trilhões de FPS (NAKAGAWA *et al.*, 2014). Os *frames* de um vídeo também podem ser processados pelas mesmas técnicas que se aplicam a imagens digitais.

2.3 OPERAÇÃO EM IMAGENS

As operações matemáticas que podem ser feitas em matrizes também podem ser aplicadas às imagens. No caso, as operações feitas em imagens não devem ultrapassar alguns valores limitantes. Por exemplo, em uma imagem com 256 níveis de cores em escala de cinza, o resultado de uma subtração ou de uma adição não pode ser menor do que zero nem maior do que 255. Esses são os valores limites que cada pixel pode representar, com a quantidade de bits que lhe foi atribuído. Quando são usadas ferramentas próprias de processamento de imagens, esses limites já são levados em conta automaticamente.

Os efeitos de operações com um valor constante em uma imagem podem ser vistos na Figura 2: a Figura 2(b) representa a imagem original, com 704x480 pixels, em escala de cinza; a Figura 2(a) é resultado da subtração de 100 do valor de todos os pixels; e a Figura 2(c) é o resultado da adição de 100 em todos os pixels. Os resultados dessas operações não ultrapassam os limites 0 e 255, pois as cores da imagem original estão dentro de uma escala de cinzas com 256 níveis distintos. Para que isso não aconteça, usou-se uma função específica da biblioteca de processamento de imagens para que os limites de valores de cada pixel fossem respeitados.

Figura 2 – Operações básicas (a) Subtração de 100 (b) Imagem original (c) Adição de 100

Além de operações com constantes, também é possível somar e subtrair uma imagem da outra. Esta operação está na base das técnicas de detecção de objetos. Um exemplo de subtração entre imagens pode ser visto na seção 3.1, no capítulo 3.

2.4 FERRAMENTAS COMPUTACIONAIS

Um sistema automático de identificação de comportamento deve ser implementado com auxílio de ferramentas computacionais. Algumas opções disponíveis possuem licença comercial, sendo necessária a compra de pacotes especiais para o processamento de imagens. Como exemplo, é possível citar o MATLAB, com o uso do pacote IPT (*Image Processing*

Toolbox – MATLAB). Porém, também estão disponíveis ferramentas gratuitas e de código aberto. Dentre elas, são apresentadas duas dessas ferramentas utilizadas nesta pesquisa.

2.4.1 PYTHON

Python é uma linguagem de programação de alto nível, de uso geral, interpretada e interativa, disponível sem custos no site oficial da linguagem (PYTHON SOFTWARE FOUNDATION). Por ser de alto-nível, os comandos da linguagem estão mais afastados da linguagem de máquina e mais próximos da linguagem humana e, por causa de sua flexibilidade para usos em geral, a disponibilidade de bibliotecas da linguagem é enorme, com mais de 88000 pacotes disponíveis no repositório oficial da linguagem, PyPI (PYTHON SOFTWARE FOUNDATION - PYPI). Python possui uma hierarquia estrutural forte, com blocos de código aninhados, funções, classes, módulos e pacotes. Por isso, a linguagem permite a escrita de códigos claros e lógicos (DUBOIS, 2007; KUHLMAN, 2009).

Os desenvolvedores se referem ao Python como uma linguagem que já vem com as baterias inclusas (*batteries included*, em inglês) pois, mesmo sem pacotes adicionais instalados, já apresenta muitas funções. A linguagem é indicada para uso acadêmico, pois possui bibliotecas como NumPy (NUMPY, 2016) e SciPy (SCIPY, 2016), com funções implementadas de vários métodos matemáticos, além de ferramentas para gráficos, como Matplotlib (HUNTER *et al.*, 2007), entre outras (OLIPHANT, 2007).

Python é usado por empresas e instituições em todo mundo, dentre as quais estão presentes nomes como Firaxis Games, EVE Online, YouTube, Google, LucasFilms, CERN e NASA (PYTHON SOFTWARE FOUNDATION – QUOTES; LARA, 2013).

Embora Python não seja uma linguagem que precise de um ambiente de desenvolvimento (*IDE*), podendo ser codificada diretamente no console, vários IDEs estão disponíveis. Existem opções pagas, como Komodo e Visual Studio, e opções gratuitas, como Spyder, Idle, iPython e Eclipse (IDES PYTHON).

Atualmente, duas versões distintas e oficiais de Python estão em produção e em constante atualização: a família Python 3 e a família Python 2. Os pacotes da versão 3 contam com sintaxe alterada, com mudanças importantes em algumas funções comuns. A mudança de versão teve o objetivo de padronizar algumas das inconsistências contidas na linguagem anteriormente e retirar algumas das funções mais antigas e não usadas. A documentação desta versão pode ser vista no site oficial (PYTHON SOFTWARE FOUNDATION – OVERVIEW

v3). Já para os pacotes da versão 2, as atualizações atuais contam com adições de funções, sem mudar a estrutura difundida anteriormente, e com correções de problemas. A documentação da versão 2 pode ser encontrada também no site oficial (PYTHON SOFTWARE FOUNDATION – OVERVIEW v2).

Todos os códigos deste trabalho foram feitos com o uso da versão 2 do Python (v2.7), pois está consolidada há mais tempo e apresenta melhor suporte para operações em visão computacional.

2.4.2 OPENCV

OpenCV (*Open source Computer Vision library*) é uma biblioteca nativa em C++, disponível em Windows, Linux, Mac OS, iOS e Android, usada para visão computacional, aprendizado de máquina e processamento de imagem. É uma biblioteca que faz parte da iniciativa *open source* (OPEN SOURCE INITIATIVE), gratuita em seu site oficial (OPENCV). Vem sendo cada vez mais adotada para uso com Python, porém também está disponível para uso em outras linguagens, tais como Java, C, Matlab, entre outras (BEYELER, 2015).

Por ser uma biblioteca com licenciamento *open source*, existe uma ampla comunidade de pessoas usando e dando suporte ao OpenCV, incluindo grandes empresas no mundo, como Google, Microsoft, Intel e Siemens, além de instituições como USP, Stanford, MIT e Cambridge (BRADSKI, KAEHLER, 2008).

Desde seu lançamento em 1999, pesquisadores adotaram a biblioteca para trabalhos na área de visão computacional como sua ferramenta principal. O OpenCV foi desenvolvido na Intel por um time liderado por Gary Bradski, em uma tentativa de avançar os estudos em visão e promover o desenvolvimento de aplicações na área. Sempre foi uma biblioteca *open source*, desde suas versões iniciais (LAGANIÈRE, 2011).

Um dos objetivos da biblioteca é disponibilizar uma infraestrutura de visão computacional que permita ao programador construir aplicações sofisticadas e com funcionalidades complexas rapidamente. Foi projetada com foco em aplicações em tempo real e tenta ser o mais eficiente possível. A biblioteca contém mais de 500 funções dentro da área de visão computacional, incluindo imagens médicas, segurança, calibração de câmeras, visão estérea e robótica. Inclui funções para aprendizado de máquina, com foco em agrupamento e reconhecimento estatístico de padrões (BRADSKI, KAEHLER, 2008).

A escolha por usar a versão em Python das funções da biblioteca do OpenCV se deu pela facilidade de programação da linguagem. O tempo gasto para implementar um programa em Python é, em média, metade do tempo necessário para se programar em C/C++, além de resultar em um programa com a metade do tamanho. Embora Python possua requerimentos maiores com relação à memória e a tempo de processamento, é uma alternativa viável para programas que tenham que lidar com quantidades consideráveis de cálculos e dados. As desvantagens se tornam aceitáveis dada à facilidade na programação e ao ganho de produtividade de programadores que escolhem Python no lugar de linguagens mais convencionais, como C/C++ (PRECHELT, 2000; PRECHELT, 2003).

Neste trabalho, todos os códigos foram escritos com a biblioteca OpenCV, em sua versão 3.2, compilada com todas as funções extras presentes no repositório *opencv_contrib*¹.

2.5 COLETA DE DADOS

Os vídeos processados em todas as fases deste trabalho foram obtidos de várias fontes distintas:

- (1) do trabalho de Cunha (2013) em coletas realizadas em 2010 na SP-310, rodovia Washington Luís, km 235 – entrada de São Carlos, SP;
- (2) de vídeos fornecidos pela concessionária CCR – NovaDutra, na BR-116, rodovia Presidente Dutra, km 180, em 2016;
- (3) de coletas feitas na SP-310, km 233, em São Carlos;
- (4) de coletas feitas em Araraquara, na Avenida Padre Francisco Sales Colturato, em 2017.

Ao todo, a amostra é composta de sete vídeos, totalizando aproximadamente 2 horas de filmagens, em condições distintas de luminosidade e visibilidade, com tempo ensolarado e nublado, e também três ângulos de câmera distintos.

Dos sete vídeos, dois foram gravados em perspectiva, mostrados na Figura 3. O vídeo #1, fornecido pela CCR–NovaDutra, foi gravado em um dia nublado, com visibilidade reduzida, sem presença de sombras. O vídeo #2 foi obtido na rodovia SP-310 com sol próximo ao meio dia, e com várias nuvens passageiras, oriundo do trabalho de Cunha (2013).

¹ https://github.com/opencv/opencv_contrib

vídeo #1 : BR-116

vídeo #2 : SP-310

Figura 3 – Vídeos em perspectiva

Quatro vídeos foram feitos com a câmera posicionada em frente ao tráfego, todos em cima de viadutos, como pode ser visto na Figura 4. Os vídeos #3 e #5 foram gravados na Av. P. Francisco S. Colturato, em Araraquara, de manhã, sem presença de nuvens, a favor e contra o fluxo do tráfego. Os outros dois vídeos foram feitos na SP-310, contra o fluxo, com sol ao meio dia e sombras curtas – vídeo #4 – e sol ao entardecer, com sombras alongadas – vídeo #6.

vídeo #3 : Av. Colturato

vídeo #4 : SP-310

vídeo #5 : Av. Colturato

vídeo #6 : SP-310

Figura 4 – Vídeos frontais

O vídeo #7 também foi gravado na SP-310, capturado lateralmente ao tráfego, como pode ser visto na Figura 5. Foi gravado no meio da tarde, com sol forte, e com os veículos da pista sul.

vídeo #7 : SP-310

Figura 5 – Vídeo com posição lateral

Um resumo das características dos vídeos utilizados, com informações sobre duração individual, resolução e taxa de *frames* por segundo, pode ser visto na Tabela 1.

Tabela 1 – Características dos vídeos utilizados

Vídeo	Local	Visão	Tempo	Duração	Resolução	FPS
#1	BR-116	Perspectiva	Nublado	00:55:21	704x480	24
#2	SP-310	Perspectiva	Sol ao meio dia	00:18:16	320x240	30
#3	Av. Colturato	Frontal	Sol de manhã	00:13:14	1920x1080	30
#4	SP-310	Frontal	Sol ao meio dia	00:06:44	320x240	30
#5	Av. Colturato	Frontal	Sol de manhã	00:12:42	1920x1080	30
#6	SP-310	Frontal	Sol ao entardecer	00:07:40	320x240	30
#7	SP-310	Lateral	Sol ao entardecer	00:13:26	1920x1080	60

3 DETECÇÃO DE VEÍCULOS

Todo *frame* de um vídeo pode ser dividido em dois planos: *foreground*, composto por todos os objetos de interesse, e *background*, composto de todos os *pixels* que formam o fundo estático (ELHABIAN; EL SAYED; AHMED, 2008). Portanto, qualquer sistema automático de detecção de objetos deve ser capaz de distinguir claramente entre esses dois planos (SENCHING; KAMATH, 2004).

Vários métodos para a detecção de veículos em um vídeo já foram propostos, como descritos por McIvor (2000) e Benezeth *et al.* (2008), os quais detalham vantagens e desvantagens de cada método.

Wren *et al.* (1997) usaram modelos estocásticos para modelar o *background* e o *foreground*, baseado em regiões, que se adaptam às mudanças em tempo real, na segmentação de objetos. A limitação desse método está na dependência do objeto que se deseja rastrear, necessitando assim assumir características específicas. Já Coifman *et al.* (1998) propuseram um sistema de rastreamento de veículos em imagens que, no lugar de detectar regiões de veículos inteiros, detectaram *features* em movimento. *Features* são conjuntos de *pixels* que formam características particulares dos veículos, como cantos de para-choques e lanternas, que se destacam do *background*. Desse modo, tenta-se evitar o problema da oclusão, que ocorre quando partes de um veículo perdem a linha de visão com a câmera e são obstruídas por outro veículo. A desvantagem do método de Coifman *et al.* (1998) está na calibração dos parâmetros para agrupamento das *features* detectadas.

Connell *et al.* (2004) usaram padrões de movimento aparente (*optical flow*) para rastrear objetos em movimento e ligar suas trajetórias por todos os *frames* em que aparecem. O *optical flow* procura a ocorrência de uma *feature* em uma sequência de *frames*, buscando o mesmo padrão de *pixels*. Para detectar o *foreground*, o algoritmo subtrai todos os *frames* de cada vídeo de um modelo de *background* gerado com o uso de Modelos de Misturas Gaussianas (*GMM*, em inglês). O conjunto de técnicas aplicadas por Connell *et al.* (2004) funciona tanto em ambientes internos quanto externos. Petrovskaya e Thrun (2009) usam filtros Bayesianos para segmentar veículos, selecionando porções da imagem que se encaixam em modelos previamente criados com informações sobre a geometria dos veículos. A desvantagem desse método está no alto custo computacional da detecção de novos veículos, pois os modelos são gerados dinamicamente, conforme os veículos entram em cena.

Cheng, Weng e Chen (2012) detectaram veículos em imagens aéreas com classificação pixel-a-pixel, com uso de uma Rede Bayesiana Dinâmica (*DBN*, em inglês). Uma *DBN* usa grafos para representar a relação entre cada nó analisado – os pixels – e funções de probabilidade para classificar cada um desses nós. Desse modo, o algoritmo proposto analisa cada elemento da imagem e o classifica como *background* ou *foreground*. A desvantagem do método está no ajuste dos parâmetros de detecção, que deve ser feito baseado em cada vídeo processado. Já Chen e Ellis (2014) segmentam veículos com um algoritmo de modelagem de *background* baseado em um *GMM* que se adapta conforme as mudanças das condições de luz. O algoritmo faz isso usando *clusters* com pesos diferentes que se atualizam *frame-a-frame*. Novos veículos que entram no *frame* geram vários *clusters* pequenos e com pesos menores. A modelagem do *background* acontece na escolha dos *clusters* com pesos maiores. A desvantagem é que o método só pode ser usado com câmeras estáticas. Sivanantham, Paul e Iyer (2016) propõem a construção de um *hardware* específico para o *tracking* de objetos, usando a diferença entre *frames* sucessivos para detectar movimentos. O sistema é capaz somente de detectar o movimento dos objetos, sem informações de rastreamento. Por fim, Hou, Chiou e Lin (2017) testam diferentes maneiras de se detectar e rastrear objetos, desde análise de cores até análise de continuidade de bordas, passando por técnicas de erosão e dilatação.

Em resumo, todos esses trabalhos apresentam em comum, em seus métodos propostos, alguma forma de modelagem ou de remoção do *background*. Essas técnicas de remoção de *background* se baseiam em uma operação matemática simples feita em dois ou mais *frames* do vídeo: a subtração.

3.1 SUBTRAÇÃO ENTRE *FRAMES*

Existe uma grande variedade de técnicas de subtração entre imagens com o objetivo de detectar objetos, com capacidades e usos diferentes. Na literatura, é possível identificar duas frentes de subtração de imagens: subtração entre *frames* sucessivos (*frame difference*) e subtração de fundo (*background subtraction*).

Técnicas de subtração entre *frames* sucessivos usam da capacidade de aplicação das operações matemáticas básicas sobre matrizes para determinar quais pixels estão sofrendo alguma alteração. Pode-se usar somente a diferença entre dois *frames* sucessivos para observar mudanças repentinas, como também é possível usar mais *frames* para detectar mudanças mais sutis. O resultado é o destaque dos pixels que tiveram seus valores alterados entre os *frames*

utilizados. É possível ver um exemplo dessa técnica sendo aplicada na Figura 6, na qual os pixels dos veículos em movimento são destacados em relação ao *background*, que fica preto.

Figura 6 – (a) *Frame 1* (b) *Frame 10* (c) Subtração entre *Frames*

O método de subtração entre *frames* sucessivos apresenta falhas em situações em que os objetos se movem a baixas velocidades. Como a técnica usa uma quantidade de *frames* pré-definida para calcular as diferenças entre os pixels, um objeto que demore mais do que esta quantidade de *frames* para se mover não será detectado. Por isso, alguns autores apresentam modificações na técnica original e a combinam com outras técnicas de detecção de movimento, para tentar mitigar esse problema (CUCCHIARA; PICCARDI, 1999; KIM *et al.*, 2001; ZHAN *et al.*, 2007; ALEX; WAHI, 2014; HAN, 2015).

Por sua vez, as técnicas de subtração de fundo comparam o *frame* atual do vídeo a um *frame* de *background* ou de referência, gerado previamente. Pixels no *frame* atual que desviam em suas características dos pixels no *frame* de referência são considerados como objetos em movimento (BRUTZER; HOFERLIN; HEIDEMANN, 2011). O *frame* usado como referência deve ser uma representação do *background* da imagem, sem nenhum objeto em movimento, e deve ser atualizado constantemente, devido às alterações de luminosidade, força do vento e clima que afetam diretamente as características dos pixels (PICCARDI, 2004; CHEN, 2017).

Na Figura 7 é possível notar o resultado de uma operação de subtração de fundo, com o destacamento de todos os pixels diferentes entre um *frame* e outro.

Figura 7 – (a) *Frame 10* (b) *Frame de background* (c) Subtração de fundo

Justamente por necessitar de um *frame* que possa representar o *background* das imagens, os métodos de subtração de fundo devem ser capazes de gerar tal imagem. É bem improvável que uma via movimentada apresente condições ideais para se construir o *background* perfeito, sem nenhum objeto em movimento por tempo suficiente, e que seja capaz de mantê-lo atualizado (ZHENG *et al.*, 2006).

3.2 MÉTODO PROPOSTO

O objetivo do método proposto neste capítulo é detectar veículos em imagens de vídeo em tráfego rodoviário. Para tanto, é necessário avaliar métodos que possam gerar um modelo do *background*, para que os pixels que fazem parte dos veículos seja segmentado.

O processo de avaliação foi feito seguindo os passos do fluxograma da Figura 8.

Figura 8 – Fluxograma do módulo de avaliação de desempenho

No primeiro passo na avaliação de desempenho, o vídeo é passado como parâmetro e é usado para criar o modelo de *background* do algoritmo. Esse modelo gerado é subtraído do *frame* atual do vídeo, resultando um *frame* de detecção, no qual os pixels já estão segmentados. Durante esse processo é computado o tempo de processamento para gerar a imagem de fundo, para fins de comparação entre os métodos avaliados. No segundo passo, o *frame* de detecção

tem sua classificação comparada à classificação feita manualmente (*ground-truth*) e os valores das métricas – taxa de exatidão e taxa de precisão – são extraídas. Essas informações podem ser analisadas para a decisão do melhor algoritmo de modelagem de *background*.

Os vídeos processados por essa fase do algoritmo, e suas características, podem ser vistos na Tabela 1, no item 2.5.

3.2.1 GROUND-TRUTH

Para que seja possível fazer a comparação dos métodos testados, foi necessária a criação de um banco de imagens contendo *frames ground-truth*. Um *frame* que tenha sido segmentado em *background* e *foreground* perfeitamente, ou seja, com todos os pixels pertencentes ao objeto de interesse corretamente classificados como pertencentes ao *foreground*, e todos os pixels pertencentes ao *background* classificados como *background*, é chamado de *ground-truth*.

Neste trabalho, os objetos correspondem aos veículos em movimento na rodovia, dentro da área de interesse. Para gerar o *ground-truth*, cinco *frames* aleatórios de cada um dos vídeos foram selecionados e segmentados manualmente, um por um. Os *frames* de exemplo dos vídeos gravados em perspectiva podem ser vistos na Figura 9. Todos os *frames* já estão cortados, exibindo somente a região de interesse.

Perspectiva

Figura 9 – *Frames* originais dos vídeos em perspectiva e *ground-truth*

Os *frames* de exemplo dos vídeos gravados frontalmente ao tráfego veicular podem ser vistos na Figura 10.

Frontal

Frame Original

Ground-Truth

Figura 10 – *Frames* originais dos vídeos frontais e *ground-truth*

O *frame* de exemplo do vídeo gravado com ângulo lateral ao tráfego veicular pode ser visto na Figura 11.

Lateral

Figura 11 – *Frame* original do vídeo lateral e *ground-truth*

Estão disponíveis na *internet* repositórios de imagens *ground-truth*, já devidamente classificadas. Trata-se de uma ferramenta tradicional na avaliação de técnicas de visão computacional. Porém, para a área de Engenharia de Transportes, essas imagens ainda são pouco utilizadas.

3.2.2 AVALIAÇÃO DE ALGORITMOS DE MODELAGEM DE *BACKGROUND*

Estão disponíveis na biblioteca *OpenCV*, na sua versão 3.2 em Python, três algoritmos de modelagem de *background*. Neste trabalho, métricas objetivas foram usadas para avaliar o desempenho desses algoritmos em situações comuns em vídeos de tráfego rodoviário.

Todas as funções usadas nos testes possuem parâmetros que podem ser alterados para calibrar o algoritmo. É importante conhecer o comportamento de cada um desses parâmetros e, para isso, suas funcionalidades podem ser vistas na documentação do *bgsegm*² e do *MOG2*³. Um resumo desses comportamentos pode ser visto na descrição de cada algoritmo avaliado na seção 3.2.2.

3.2.2.1 GMG

O algoritmo GMG (GODBEHERE; MATSUKAWA; GOLDBERG, 2012), cujo nome tem origem na abreviação dos sobrenomes dos autores, modela o *background* com o uso de Inferência Bayesiana e Filtros de Kalman. Usa necessariamente os primeiros *frames* do vídeo para modelar o *background*, apresentando *frames* completamente pretos durante toda sua inicialização. É dividido em duas etapas principais. A primeira etapa é uma segmentação probabilística do *foreground*, que identifica possíveis objetos usando Inferência Bayesiana, e a criação de um modelo estimado de *background*, que varia com o tempo. O modelo de *background* consiste em distribuições não-paramétricas em RGB para cada pixel da imagem,

² https://docs.opencv.org/3.2.0/df/d5d/namespacecv_1_1bgsegm.html

³ https://docs.opencv.org/3.2.0/d7/d7b/classcv_1_1BackgroundSubtractorMOG2.html

que recebem pesos diferentes de acordo com o tempo em que aparecem. Observações novas no *background* ganham mais peso que observações mais antigas, permitindo que o modelo se adapte a mudanças de luminosidade. A segunda etapa do algoritmo filtra os objetos detectados no *foreground*, com o uso de filtros de Kalman. Filtro de Kalman é normalmente usado para medir grandezas com ruídos e aproximá-las dos valores reais, sem ruídos. Neste estágio, também é usada a distância de Mahalanobis para decidir se o pixel faz parte do *foreground* ou não.

O algoritmo GMG, em sua implementação no OpenCV 3.2, possui parâmetros que podem ser alterados. Também pode ser chamado sem nenhum parâmetro, fazendo com que seja executado com valores padrão, como visto na Figura 12.

```
cv2.bgsegm.createBackgroundSubtractorGMG(  
                                     initializationFrames=120,  
                                     decisionThreshold=0.8  
                                     )
```

Figura 12 – Construtor do algoritmo GMG, com valores padrão

O parâmetro `initializationFrames` indica quantos *frames* do começo do vídeo devem ser utilizados para inicializar o algoritmo. Durante toda essa quantidade especificada nesse parâmetro, o resultado do algoritmo será um *frame* completamente preto. Quanto mais *frames* utilizados, melhor o modelo do *background* inicial. O parâmetro `decisionThreshold` determina o valor de corte para a probabilidade usada na geração dos objetos do *foreground*. Pixels com valores menores que o valor de corte serão atribuídos ao *background*, enquanto que pixels com valores maiores serão classificados como *foreground*. Valores muito baixos nesse parâmetro podem criar vários objetos falsos no *foreground*, ao passo que valores muito altos podem acabar classificando pixels de objetos como sendo pixels do *background*.

3.2.2.2 MIXTURE OF GAUSSIANS

Baseado em trabalho anterior (STAUFFER; GRIMSON, 1999; STAUFFER; GRIMSON, 2000), o algoritmo MOG (*Mixture of Gaussians*) implementado no OpenCV foi proposto inicialmente por Kaewtrakulpong e Bowden (2002). Nele, cada pixel do *background* é modelado a partir de uma mistura de K distribuições Gaussianas, com K assumindo um valor sempre pequeno, fixo, de três a cinco. Assume-se que distribuições Gaussianas diferentes representem cores diferentes no *background*. Os parâmetros de peso associados a cada mistura de distribuições representam a proporção de tempo na qual uma cor permanece no vídeo. Como

as prováveis cores do *background* tendem a ficar estáticas e a permanecer por mais tempo no vídeo, os pixels do *background* são identificados ao verificar-se quais misturas de distribuições possuem pesos maiores.

No OpenCV 3.2, o algoritmo MOG possui alguns parâmetros, para ajudar no ajuste fino a cada vídeo diferente. Para se usar o algoritmo, basta chamar o construtor presente na Figura 13. Caso não sejam passados parâmetros para a função, os valores padrão serão adotados.

```
cv2.bgsegm.createBackgroundSubtractorMOG(
    history=200,
    nmixtures=5,
    backgroundRatio=0.7,
    noiseSigma=0
)
```

Figura 13 – Construtor do algoritmo MOG, com valores padrão

O parâmetro *history* altera a quantidade de *frames* anteriores que serão usados para se gerar o *background*. Valores pequenos se tornam muito sensíveis a mudanças rápidas na cena e se comportam melhor com mudanças bruscas de luminosidade. Valores altos fazem com que mudanças no *background* sejam calculadas de maneira mais suave, porém requerem muito mais tempo de vídeo para estabilizar o modelo. O parâmetro *nmixtures* indica a quantidade de misturas Gaussianas que se deseja usar. Quanto mais misturas para cada pixel, maior o processamento necessário. O parâmetro *backgroundRatio* define qual o peso necessário para que uma mistura seja considerada e incluída no modelo do *background*. Valores altos podem influenciar na atualização do *background* devido a mudanças de luminosidade. Por fim, o parâmetro *noiseSigma* indica o nível de ruídos aceitos pelo modelo, pois é uma medida do desvio padrão do brilho de cada canal de cor dos pixels. Valores muito altos podem prejudicar a detecção de objetos em movimento. Valores muito baixos podem criar objetos falsos no *foreground*.

3.2.2.3 MIXTURES OF GAUSSIANS 2

No algoritmo MOG anterior, as diferentes cores do *background* são associadas a misturas de distribuições Gaussianas, com cada cor sendo relacionada com uma mistura. Porém, os valores das cores dos pixels normalmente apresentam distribuições mais complexas, criando-se a necessidade de modelos mais elaborados. Com o propósito de resolver esse problema foi desenvolvido o algoritmo *MOG2* (em inglês, *Mixture of Gaussians 2*). A diferença básica entre

o *MOG2* e o *MOG* está justamente no uso das distribuições Gaussianas. Enquanto o algoritmo *MOG* usa somente um valor de *K* para todos os pixels do *background*, o algoritmo *MOG2* usa uma quantidade de distribuições diferente para cada pixel, que varia durante a execução. Essa alteração permite ao algoritmo se adaptar melhor às mudanças na luminosidade da imagem (ZIVKOVIC, 2004; ZIVKOVIC; VAN DER HEIJDEN, 2006).

Assim como no caso do *MOG*, o algoritmo *MOG2* também possui parâmetros que podem ser alterados em sua implementação no OpenCV 3.2. Seu construtor também pode ser chamado sem nenhum parâmetro, fazendo com que seja executado com valores padrão. As especificações podem ser vistas na Figura 14.

```
cv2.createBackgroundSubtractorMOG2(  
                                     history=200,  
                                     varThreshold=16,  
                                     detectShadows=True  
                                   )
```

Figura 14 – Construtor do algoritmo *MOG2*, com valores padrão

O parâmetro *history*, assim como no algoritmo *MOG*, altera a quantidade de *frames* anteriores que serão usados para se gerar o *background*. Valores pequenos também se tornam muito sensíveis a mudanças rápidas na cena e se comportam melhor com mudanças bruscas de luminosidade. Valores altos também podem fazer com que mudanças no *background* sejam calculadas de maneira mais suave, porém requerem muito mais tempo de vídeo para estabilizar o modelo. O parâmetro *varThreshold* é o quadrado da Distância de Mahalanobis entre o pixel e o modelo, para decidir se o pixel atual é bem descrito pelo modelo de *background*. A Distância de Mahalanobis é uma medida de distância baseada nas correlações entre variáveis, usada para identificar a similaridade entre amostras, como no caso dos pixels do *frame* atual e do modelo. Valores baixos tendem a gerar objetos falsos no *foreground*, enquanto valores mais altos podem gerar objetos falsos no *background*. Por último, *detectShadows* diz ao algoritmo para detectar sombras e ignorá-las. Por ser um passo a mais, tem um impacto negativo no desempenho do algoritmo.

3.2.2.4 AVALIAÇÃO DO DESEMPENHO

O método de avaliação de desempenho dos algoritmos testados neste trabalho segue o método adotado no trabalho de Cunha (2013). Os algoritmos foram avaliados de acordo com

as taxas de precisão e de exatidão com que segmentam objetos do *foreground* e o tempo de se processar um *frame*.

Para se descobrir quais objetos foram identificados por cada método, os *frames* originais em escala de cinza são subtraídos dos *frames* de *background* gerados por cada um dos modelos de geração de *background*. No *frame* resultante dessa subtração é aplicado um algoritmo de limiarização binária, para que todos os pixels assumam ou valores brancos ou pretos. A limiarização é feita com a aplicação de uma função de *threshold*, onde todos os pixels com valores menores que 60 são alterados para 0 (preto) e valores maiores, para 255 (branco). O valor de 60 para o *threshold* foi definido através de análise visual do *frame* resultante em cada um dos vídeos. Com esse processo, o resultado é um *frame* muito parecido com o *frame* do *ground-truth*, mas criado de maneira automática, com objetos do *foreground* marcados com pixels brancos e o *background* marcado com pixels pretos. Os *frames* intermediários resultantes do processo podem ser visualizados na Figura 15.

Figura 15 – *Frames* resultantes de cada fase do processo de avaliação

Para calcular as métricas de avaliação dos modelos usados, adotou-se o método proposto por Faucett, em 2006 (FAUCETT, 2006). Todos os valores das métricas utilizados têm origem na construção de uma Matriz de Contingência (Tabela 2), junto com métricas que podem ser geradas a partir dela.

Tabela 2 – Matriz de Contingência e métricas comumente calculadas

Modelo	Ground-Truth	
	Foreground (Positivo)	Background (Negativo)
Foreground (Positivo)	VP	FP
Background (Negativo)	FN	VN
Total	P	N

$$\text{Exatidão} = \frac{VP+VN}{P+N}$$

$$\text{Precisão} = \frac{VP}{VP+FP}$$

A Matriz de Contingência é preenchida com valores vindos da comparação e contagem de pixels. É possível notar que sempre são comparados os valores dos pixels do *ground-truth*, nas colunas, com os valores dos pixels de cada um dos modelos gerados, nas linhas.

Quatro valores distintos são necessários para o preenchimento da matriz:

- Verdadeiros Positivos (VP): ocorrem quando o modelo e o *ground-truth* classificam o mesmo pixel como sendo do *foreground*;
- Falsos Positivos (FP): ocorrem quando o modelo classifica o pixel como *foreground*, mas o *ground-truth* classifica como *background*;
- Falsos Negativos (FN): ocorrem quando o modelo classifica o pixel como *background*, mas o *ground-truth* classifica como *foreground*;
- Verdadeiros Negativos (VN): ocorrem quando tanto o modelo quanto o *ground-truth* classificam o pixel como *background*.

Na Figura 16 é possível ver duas imagens de exemplo, de 5x5 pixels: uma gerada pelo algoritmo de modelagem de *background* (1); e uma imagem de *ground-truth* (2). É possível notar, no *ground-truth*, dois objetos com um pixel de tamanho cada, nas coordenadas (2,2) e (4,2).

Figura 16 – Exemplo de imagens (a) Gerada pelo modelo (b) *Ground-truth*

Na comparação dos pixels das imagens de exemplo, o pixel da coordenada (2,2) é contado como VP, pois está classificado como *foreground* tanto pelo modelo quanto pelo *ground-truth*. O pixel da coordenada (2,4) é contado como FP, pois está classificado como *foreground* pelo modelo, mas classificado como *background* pelo *ground-truth*. O pixel (4,2) é contado como FN, pois é classificado como *background* pelo modelo, mas como *foreground* pelo *ground-truth*. Por fim, o pixel (4,4) é contado como VN, pois está classificado como *background* tanto pelo modelo quanto pelo *ground-truth*.

Duas métricas importantes são extraídas da matriz de contingência e podem ser vistas na Tabela 2: a taxa de exatidão e a taxa de precisão. A taxa de exatidão mostra quantos pixels foram classificados corretamente no total, levando em conta tanto as classificações no *foreground* como no *background*. A taxa de precisão, por sua vez, só considera acertos no *foreground*, sendo uma relação entre quantos pixels do objeto o modelo acertou pelo total de pixels classificados como objeto.

O tempo de execução de cada um dos modelos é medido separadamente, com o auxílio do módulo `timeit`, do Python. Esse módulo aceita, como parâmetro, a quantidade de iterações para teste, gerando resultados mais confiáveis. É interessante usar valores altos de repetições para que fatores externos ao Python, como escalonamento de processos no processador, não tenham tanta influência sobre o teste. Como saída, a somatória do tempo levado em todas as iterações executadas é mostrada.

3.3 RESULTADOS OBTIDOS

É possível ver alguns dos *frames* gerados por cada método de remoção de *background* na Figura 17, junto com o *frame* original e o *Ground-truth*, para comparação visual. Nota-se que os três métodos testados conseguiram identificar os objetos presentes no *foreground*, pois os *backgrounds* foram bem modelados. O algoritmo GMG acabou gerando mais ruído no *background*, com pixels sendo classificados como *foreground* em várias regiões em que não deveriam existir nenhum. Esse excesso de ruído é melhor visto na terceira imagem da coluna central da Figura 17. Também é possível notar ruídos no algoritmo MOG, mas com bem menos intensidade, enquanto no MOG2 não existe quase ruído algum.

Figura 17 – *Frames* resultantes dos métodos de remoção de *background*

3.3.1 TAXA DE EXATIDÃO

Para calcular a Taxa de Exatidão, foram usados os cinco *frames* gerados para cada um dos vídeos, resultando em um conjunto com 135 imagens. As distribuições relativa e acumulada dos erros de cada um dos algoritmos testados estão apresentadas na Figura 18.

Figura 18 – Histogramas das Taxas de Exatidão

Os três métodos apresentaram alta concentração de casos nas faixas superiores a 98% de exatidão. Esse resultado na taxa de exatidão indica que os algoritmos possuem alto desempenho em classificar o *foreground* e o *background* corretamente, errando somente 2% dos pixels. Nos três algoritmos, as instâncias com menor taxa de exatidão aconteceram no vídeo com perspectiva lateral: GMG, com 80%; MOG, com 89%; e MOG2, com 81%. Esse baixo desempenho no vídeo lateral pode ser explicado pelo ângulo de gravação e pelo fato de favorecer a captura de objetos dinâmicos do *background*. No vídeo lateral, árvores são gravadas ao fundo, junto com os veículos. Quando elas se mexem por causa do vento, o algoritmo considera que essas árvores não fazem mais parte do *background*, retirando seus pixels do modelo de fundo. Com isso, alguns pixels são classificados erroneamente como pertencentes aos veículos, reduzindo as taxas de acertos. Para os outros vídeos, tanto em perspectiva quanto frontal, o desempenho dos três métodos testados se mostrou equivalente, pois apresentam *background* mais estático.

Dados os desempenhos dos algoritmos na taxa de exatidão, não é possível escolher o melhor candidato a ser usado neste trabalho. Outras métricas devem ser analisadas e levadas em consideração.

3.3.2 TAXA DE PRECISÃO

Assim como para a taxa de exatidão, a taxa de precisão também foi calculada com os valores vindos da comparação de todas as imagens geradas, tanto manualmente quanto automaticamente. Seus valores também foram organizados em distribuições de frequências, na Figura 19.

Figura 19 – Histogramas das Taxas de Precisão

Ao contrário do que ocorreu na taxa de exatidão, o desempenho na taxa de precisão dos métodos testados apresentou diferenças. O algoritmo GMG teve resultados dispersos por todas as categorias do histograma. A maior concentração dos casos ficou nas faixas entre 60 e 80%. Para o MOG e o MOG2, os resultados ficaram todos concentrados nos dois intervalos superiores do histograma, de 95% a 100%. A diferença dos resultados vem do fato de que o algoritmo GMG simplifica a composição do *background*, assumindo que todos os pixels podem ser modelados por somente uma distribuição não-paramétrica. Isso ocasiona ruídos na detecção dos objetos, como é possível notar na Figura 17. Como a taxa de precisão considera somente os pixels do *foreground*, o resultado do GMG é mais prejudicado do que os resultados do MOG e do MOG2.

Com os resultados obtidos na taxa de precisão, pode-se excluir o algoritmo GMG da escolha do melhor candidato ao uso neste trabalho. Porém, ainda existem outros dois algoritmos com desempenho similar entre si, o MOG e o MOG2.

Na detecção de objetos, os resultados obtidos neste trabalho confirmam os resultados obtidos por Trnovszký, Sýkora e Hudec (2017). Os autores usaram 20 vídeos gravados com infravermelho para detectar o movimento de animais em câmeras estacionárias, em florestas, e chegam a conclusão de que os algoritmos MOG e MOG2 apresentam desempenho equivalente.

3.3.3 TEMPO DE PROCESSAMENTO

Para se testar o tempo de processamento, foi escolhido um *frame* intermediário de um dos vídeos. Como cada algoritmo precisa de uma quantidade de *frames* iniciais para estabilizar seus modelos de *background*, o processamento se inicia 1.000 *frames* antes do *frame* de teste. Para calcular o tempo de processamento neste trabalho, o módulo `timeit` iterou sobre a instrução do algoritmo testado de modo incremental: o primeiro teste repetiu a instrução por 100 vezes; o segundo teste repetiu por 1.000 vezes; e o terceiro teste repetiu por 10.000 vezes. Dessa maneira, qualquer influência externa no cálculo do tempo, seja por escalonamento de processos do Sistema Operacional ou por *overhead* de funções do Python, é mitigada e seu efeito dissolvido nas várias iterações do teste. Os resultados podem ser observados na Figura 20.

Figura 20 – Tempo de processamento dos três algoritmos

Em cada um dos cenários, de 100, 1.000 e 10.000 repetições, o tempo resultante mostrado na Figura 20 é um somatório do tempo de processamento de todas as iterações. Por exemplo, o GMG levou quase 1.100 segundos para processar todas as 10.000 repetições do teste. O MOG, no mesmo teste, levou 550 segundos e o MOG2, 150. Na Tabela 3 é possível observar os valores dos testes em outra unidade de medida, em *frames* por segundo.

Tabela 3 – Taxas de FPS para cada teste

Taxa de <i>Frames</i> por Segundo				
	100	1.000	10.000	Média
GMG	8	13	9	10
MOG	18	18	18	18
MOG2	59	67	66	64

Neste teste de processamento, os algoritmos apresentaram desempenhos diferentes entre si. O algoritmo GMG alcançou processamento médio de 10 *frames* por segundo, sendo o pior entre os testados. Já o MOG2 processou os *frames* a 64 *frames* por segundo, com o melhor desempenho entre os métodos testados.

Portanto, o teste de tempo de processamento conclui a escolha do melhor método para este trabalho, pois consolida o algoritmo MOG2 como o melhor dentre os avaliados. Os resultados obtidos nesta fase do trabalho corroboram os resultados de Sobral e Vacavant (2014),

no tempo de processamento. Sobral e Vacavant (2014) testaram cenários tanto gerados sinteticamente, com simuladores, quanto cenários reais, e aplicaram os modelos de *background* disponíveis, avaliando seus desempenhos na detecção de veículos. Seus cenários incluíram situações simuladas de chuva e vento, para dificultar a modelagem do *background*. Os resultados apresentados também colocam o MOG2 na frente dos algoritmos MOG e GMG.

3.4 CONCLUSÕES

Esse capítulo foi dedicado ao estudo de técnicas disponíveis no processo de modelagem de *background*, com especial atenção a técnicas de subtração de fundo.

Os vídeos usados para o teste apresentam variações de condições de luminosidade, de visibilidade e de angulação da câmera, tornando-os suficientes para se testar os algoritmos presentes no OpenCV 3.2, implementados em Python.

É importante lembrar que todos os *frames* resultantes dos modelos analisados passaram por uma função de *threshold*, mudando pixels com valores menores que 60 para 0, e pixels com valores maiores para 255, para transformar a imagem em binária. Testes mais extensivos no valor do *threshold* podem gerar resultados ainda melhores na segmentação dos veículos, com valores sendo ajustados para cada um dos métodos analisados.

Na questão da exatidão, os algoritmos apresentaram um alto desempenho, com grande concentração de resultados acima de 98%, mostrando resultados equivalentes entre si. Para este trabalho, a contribuição dessa métrica é mostrar que todos os algoritmos testados conseguem identificar pixels de objetos e *background* com alta exatidão, porém não ajuda na decisão sobre o mais apropriado. Portanto, para a determinação da escolha do melhor algoritmo a ser utilizado, a taxa de exatidão, nos vídeos usados neste trabalho, precisa de complementação por outras métricas.

Nas taxas de precisão, o algoritmo GMG obteve desempenho inferior aos outros, com resultados espalhados por todo o intervalo do histograma e concentração nas faixas de 60 a 80%. Já os algoritmos MOG e MOG2 apresentaram resultados parecidos, com concentração de ocorrências nas faixas de 95 a 100% de precisão. Portanto, para este trabalho, essa métrica ajuda a descartar o algoritmo GMG, mas ainda não resolve a questão do melhor algoritmo.

Por último, no tempo de processamento, cada algoritmo atingiu um desempenho diferente. O GMG processou os *frames* a uma taxa de 10 FPS, sendo o algoritmo com o pior

desempenho. O MOG conseguiu processar seus *frames* a uma taxa de 18 *frames* por segundo, resultado aproximadamente 50% superior ao do GMG. O algoritmo MOG2 processou a 64 *frames* por segundo, três vezes mais rápido que o MOG. Portanto, essa métrica completa os testes dos algoritmos, definindo o MOG2 como sendo a técnica mais adequada para este trabalho.

4 TRACKING DE VEÍCULOS EM IMAGENS

O *tracking*, ou rastreamento, de um objeto qualquer se dá quando um sistema computacional consegue relacionar o mesmo objeto em diferentes *frames* do vídeo. Com essa relação entre os *frames*, é possível extrair informações como histórico da trajetória, velocidade e aceleração do movimento e quantidade de objetos na cena (BRAKATSOULAS, 2005; YILMAZ; JAVED; SHAH, 2006; STURARI *et al.*, 2016).

As técnicas para *tracking* de veículos existentes na literatura podem ser classificadas em quatro áreas distintas, que simbolizam o modo como o problema é resolvido, segundo Cavallaro, Steiger e Ebrahimi (2005).

Técnicas que usam modelos em 3D dos objetos na cena são classificados como o primeiro grupo. Essas técnicas devem ter um conhecimento *a priori* das características dos veículos presentes na via para se criar um modelo típico representativo. Com esse conhecimento é possível fazer uma comparação entre os objetos em cada *frame* e os modelos definidos. Este método permite recuperar trajetórias, pois o modelo possui uma orientação conhecida, além de resolver parcialmente problemas de oclusão – dois ou mais veículos trafegando próximos, sendo que um acaba escondendo as características dos demais. No entanto, justamente por depender de modelos previamente gerados, é necessário ter uma modelagem feita para cada tipo de veículo possível de aparecer na via, o que, na prática, é bem difícil de acontecer (SAUNIER; SAYED, 2006).

Kaestner *et al.* (2012) criaram o modelo 3D usando clusterização para agrupar os pixels que fazem parte do objeto para rastreá-los. Porém, o algoritmo ignora informações sobre formato e cor, rastreando-os somente pelo centroide. Justamente por isso, o algoritmo proposto consegue rastrear objetos com modelos 3D, mas não consegue classificar ou identificar o objeto rastreado.

Held *et al.* (2016) utilizaram o formato, a cor e o movimento para gerar um modelo completo em 3D do objeto. O método é composto de uma parte de pré-processamento, na qual os objetos são segmentados e associados através de vários *frames*, seguida de uma fase de modelagem probabilística. Essa modelagem leva em consideração que os objetos de interesse, que são pessoas, bicicletas e veículos, sempre se mantêm no chão, com pouco movimento vertical. Uma Rede Bayesiana dinâmica é usada para associar informações do modelo 3D com cor e movimento, permitindo que o algoritmo rastreie o objeto.

As técnicas de *tracking* que usam regiões (*blobs*, em inglês) para reconhecer objetos em movimento nos *frames* fazem parte de outra categoria de algoritmos. Esses métodos identificam regiões conectadas e em movimento com o uso de subtração de fundo, e tentam separá-las em hipóteses de veículos independentes. Para fazer isto, usam informações das próprias regiões identificadas, como cor dos pixels, movimento relativo, tamanho, formato, textura e centroide (MEIER; NGAN, 1998; TAO; SAWHNEY; KUMAR, 2002). Este método funciona bem em situações de fluxo livre e é computacionalmente eficiente, mas apresenta dificuldades em tráfego intenso. Veículos próximos podem ser classificados como uma grande região em movimento, dificultando a extração de parâmetros do tráfego (SAUNIER; SAYED, 2006).

Song e Nevatia (2007) propuseram um algoritmo que tenta resolver a questão de uma região detectada contendo vários veículos. O algoritmo usa caixas de tamanho fixo como aproximação do formato dos veículos e cria várias configurações possíveis dessas caixas dentro das regiões. Baseado nos movimentos das regiões através do tempo, o algoritmo usa o método estatístico MCMC (*Markov Chain Monte Carlo*) para calcular qual a configuração mais provável. O problema do método está justamente no tamanho fixo das caixas, pois nem todos os veículos possuem tamanhos ou formatos parecidos.

Uzkent, Rangnekar e Hoffman (2017) propuseram o uso de *blobs* como parte de seu método de *tracking* de veículos por imagens aéreas. O algoritmo consegue rastrear veículos analisando o histograma dos *blobs* e seu movimento através dos *frames*. O problema desse método de rastreamento é que objetos ou até mesmo pedaços do *background* que tenham a mesma composição de histograma, com esquema de cores parecidas, por exemplo, podem confundir o rastreador, perdendo o veículo na cena.

A terceira categoria de técnicas engloba modelos que usam apenas o contorno dos objetos para manter o *tracking* entre *frames*. Estes métodos se baseiam no movimento para projetar o contorno e então adaptá-lo ao veículo nos próximos *frames* (GU; LEE, 1998). O contorno pode ser atualizado ativamente (quando é chamado de *snake*, em inglês) ou *meshes*, que se adaptam a objetos que podem mudar de forma mais facilmente (PETERFREUND, 1999; ZHAO; WANG; LIU, 2002). Estas técnicas apresentam complexidade computacional menor do que técnicas baseadas em região, mas também podem ter dificuldades quando confrontadas com muitos veículos lado a lado, com várias situações de oclusão. Se um contorno independente puder ser gerado para cada veículo que entra na cena, essas dificuldades podem ser reduzidas. Porém, a fase de inicialização dos contornos é um processo dispendioso.

Sun, Haynor e Kim (2003) propuseram um algoritmo que usa *snakes* para rastrear o objeto. O algoritmo inicializa um contorno para cada objeto e depois atualiza dinamicamente o formato, com passos extra para simplificar o contorno e diminuir a carga de processamento.

No Brasil, Silva *et al.* (2016) juntaram detecção de cores ao tradicional método de segmentação por contornos, resultando em um algoritmo com boa performance. Foi implementado na linguagem C++, com a biblioteca OpenCV, na plataforma Arduino, e usava dois servo-motores para seguir o objeto enquanto se movia. No entanto, a técnica desenvolvida não foi testada especificamente para *tracking* de veículos.

O quarto grupo de técnicas baseia o *tracking* em características particulares dos objetos no *frame* (do inglês, *features*). Estas técnicas abandonam a ideia de seguir objetos como um todo e tentam rastreá-los através de características discerníveis, como cantos, linhas, ou pontos de interesse. Esta estratégia ajuda a identificar veículos mesmo em situações de oclusão parcial, além de permitir ao algoritmo trabalhar em situações de luminosidade diferentes. Para o caso de veículos, este método consegue, em boas condições de iluminação, selecionar *features* relativas aos cantos de para-brisas ou para-choques e, em vídeos no período noturno, identificar as lanternas dos veículos como *features* (SAUNIER; SAYED, 2006). Após a detecção das *features*, é necessário agrupar essas características que compõe um único veículo, para que seja possível extrair as informações dos veículos. O agrupamento das características é o ponto crítico deste método (CAVALLARO; STEIGER; EBRAHIMI, 2005).

Vários autores vêm sugerindo melhorias e soluções para o problema do agrupamento, mas nenhuma técnica conseguiu bons resultados para todas as situações. Beymer *et al.* (1997) atingiram, em rodovias norte-americanas, uma taxa de acerto de 75,2%. Coifman *et al.* (1998), conseguiram taxas de acerto variando de 75% a 97,5%, dependendo de condições de tráfego e luminosidade. Collins, Liu e Leordeanu (2005) conseguiram bons resultados na detecção, porém o *tracking* das *features* deve ser inicializado manualmente, com o usuário selecionando os objetos.

Jazayeri *et al.* (2011) propuseram um método que usa detectores de Harris – mostrados na seção 4.2.1 – para detectar *features* dos veículos nos *frames*, junto com um detector de linhas e um detector de picos de intensidade de luz, para ajudar na segmentação e no *tracking* de veículos. Seu método atingiu 86,6% de identificações corretas de veículos em seus testes.

Lin *et al.* (2012) sugeriram o uso de dois tipos distintos de *features* para identificar e rastrear veículos. O primeiro tipo detecta *features* baseadas na aparência, com o uso de um

banco de dados de treinamento para ensinar ao algoritmo quais características são mais comuns em veículos. O segundo tipo de *feature* detectada é baseada em bordas, detectando *features* nas regiões limítrofes dos veículos. O algoritmo obteve 90% de acertos em reconhecer e rastrear veículos.

Do e Woo (2016) propuseram um método para rastrear veículos que detecta *features* usando uma combinação do algoritmo de Shi-Tomasi (SHI *et al.*, 1994) com o algoritmo SIFT. A combinação se mostrou eficiente, mas somente avaliou se o algoritmo conseguia rastrear a trajetória das *features*, e não do veículo inteiro.

Shih e Zhong (2017) também utilizaram detecção de *features* usando o método de Shi-Tomasi para detectar bordas e cantos. O agrupamento das *features* foi feito através de uma técnica de clusterização, usando a distância entre as *features* detectadas. O algoritmo possui uma taxa de acerto de 90% na contagem de veículos.

Baseado na análise dos resultados dos trabalhos mencionados, e em trabalhos recentes (WOO; HONG, 2014; JAIN *et al.*, 2015; DO; WOO, 2016; DOOLEY *et al.*, 2016; SOUZA CASTRO *et al.*, 2016; KE *et al.*, 2017; SHIH; ZHONG, 2017), foi decidido usar, para este trabalho, a técnica de detecção de *features* para o *tracking* dos veículos.

4.1 ALGORITMOS DE DETECÇÃO DE *FEATURES*

Com a decisão de usar detectores de *features* tomada, o passo seguinte foi analisar quais algoritmos estavam disponíveis no OpenCV 3.2, em Python, e que poderiam ajudar a resolver a questão do *tracking*. A documentação⁴ do OpenCV lista esses algoritmos de detecção de *features*, com explicação da funcionalidade de todos os parâmetros e seus efeitos na detecção.

4.1.1 HARRIS *CORNER DETECTION*

Corners, ou cantos, em português, são regiões de uma imagem que apresentam grande variação de intensidade em todas as direções. O detector de *corners* proposto por Harris e Stephens (1988) converteu essa ideia em uma fórmula matemática que consegue, ao alocar pesos diferentes para cada porção da imagem, detectar se existe um canto em determinada posição. A equação que determina o peso das janelas pode ser vista na equação (4.1).

⁴ https://docs.opencv.org/3.2.0/db/d27/tutorial_py_table_of_contents_feature2d.html

$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 \quad (4.1)$$

onde λ_1 e λ_2 são autovalores que trazem informações sobre as mudanças de intensidade da região observada.

O valor de R é então avaliado de acordo com três regras:

- Se $|R|$ é pequeno, o que acontece quando λ_1 e λ_2 são pequenos, a região é plana, ou seja, não contém cantos nem bordas;
- Se $R < 0$, o que acontece quando $\lambda_1 \gg \lambda_2$ ou $\lambda_2 \gg \lambda_1$, a região contém uma borda;
- Se R é grande, o que acontece quando λ_1 e λ_2 são grandes ou quando $\lambda_1 \sim \lambda_2$, a região é um canto.

Dessa forma, o método proposto por Harris e Stephens (1988) consegue classificar regiões das imagens. No OpenCV, o método está implementado e possui parâmetros que podem ser ajustados, como visto na Figura 21.

```
cv2.cornerHarris(  
    img,  
    blockSize,  
    kSize=3,  
    k=0.04  
)
```

Figura 21 – Função do detector de Harris

O parâmetro `img` determina o *frame* a ser analisado. O parâmetro `blockSize` indica o tamanho da região a ser analisada, em volta de um pixel. Valores muito baixos podem fazer com que bordas e cantos sejam considerados como regiões planas. Valores muito altos podem acabar agrupando mais de um canto. O parâmetro `kSize` influencia na derivada de Sobel usada dentro do algoritmo para derivar os valores da imagem. Valores baixos podem deixar de detectar cantos ou bordas. Valores muito altos agrupam vários pixels distintos e podem agrupar mais de um canto. O parâmetro `k` aparece na função de cálculo de peso e atua na classificação do que pode ser considerado um canto ou uma borda. Valores baixos classificam mais regiões como sendo planas. Valores altos classificam mais bordas e cantos.

4.1.2 SHI-TOMASI CORNER DETECTION

O detector de cantos de Shi-Tomasi (SHI *et al.*, 1994) apresentou uma pequena modificação no detector de Harris. A alteração proposta foi feita no cálculo da parte final do

algoritmo, na determinação do peso, o que acabou proporcionando melhores detecções, além de envolver menos cálculos. O peso dado para cada região analisada passou a ser o valor mínimo entre λ_1 e λ_2 . O método de Shi-Tomasi detecta cantos somente quando os valores de λ_1 e λ_2 são maiores que um λ_{min} determinado.

No OpenCV, o detector de Shi-Tomasi possui vários parâmetros de entrada. Eles podem ser vistos na Figura 22.

```
cv2.goodFeaturesToTrack(  
    image,  
    corners,  
    maxCorners,  
    qualityLevel,  
    minDistance,  
    mask=noArray(),  
    blockSize=3,  
    useHarrisDetector=False,  
    k=0.04  
)
```

Figura 22 – Função que implementa o detector de Shi-Tomasi

O parâmetro `image` recebe a imagem de entrada, a ser processada. O parâmetro `corners` recebe o vetor de saída com os cantos detectados pelo algoritmo. O parâmetro `maxCorners` indica a quantidade máxima de cantos que o algoritmo retorna. Quanto mais cantos, maior o custo de processamento. O parâmetro `qualityLevel` indica qual o valor mínimo que um canto pode ter como peso. Esse parâmetro é multiplicado pelo valor do melhor canto detectado pelo algoritmo, ou seja, o menor autovalor λ . Por exemplo, se a melhor qualidade encontrada for 2.000 e o `qualityLevel` for 0,1, cantos com valor de qualidade menor que 200 serão rejeitados. Valores altos podem gerar muitos cantos, diminuindo a performance do algoritmo. O parâmetro `minDistance` indica qual a distância mínima que deve existir entre dois cantos. Valores muito baixos também podem aumentar demais a quantidade de cantos, causando baixa performance. O parâmetro `mask` serve para diminuir a região na qual se procurará cantos, ou para ocultar regiões em que não exista interesse em encontrar objetos. O parâmetro `blockSize` é usado para indicar o tamanho da vizinhança centrada em um pixel na qual se procurará cantos. Valores muito altos faz o algoritmo processar muitas combinações de pixels, causando queda no desempenho. O parâmetro `useHarrisDetector` substitui o detector de Shi-Tomasi pelo detector de Harris e faz com que o parâmetro `k`, o mesmo parâmetro da função anterior, de Harris, tenha utilidade.

4.1.3 SIFT

O algoritmo SIFT (*Scale-Invariant Feature Transform*) para detecção de *features* foi desenvolvido por Lowe (2004). Sua vantagem com relação aos detectores de cantos tradicionais, de Shi-Tomasi e Harris, é que pode funcionar em imagens na qual a escala dos objetos muda. Por exemplo, um objeto que ocupe 50% da área de uma imagem tem seus cantos representados por uma quantidade de pixels bem maior do que um objeto que ocupe somente 10% da área da imagem. Para isso, o algoritmo SIFT usa Diferença de Gaussianas, que varia a escala das imagens em busca de pontos extremos, e usa dois limites (*thresholds*) diferentes, baseados na intensidade do ponto e na presença de bordas, para refinar a busca por pontos extremos. Após esses dois passos, os pontos têm sua orientação calculada, para que o algoritmo não confunda quando um ponto extremo girar e, por fim, um descritor é criado. Nesse descritor, um vetor com 128 posições diferentes é criado, com informações sobre o histograma de todos os vizinhos do pixel, agrupados em regiões de 4x4, numa janela de 16x16 pixels.

No OpenCV, o construtor do algoritmo SIFT possui alguns parâmetros que podem ser modificados. Na Figura 23 é possível ver tais parâmetros.

```
cv2.xfeatures2d.SIFT_create(
                                nFeatures,
                                nOctaveLayers=3,
                                contrastThreshold=0.04,
                                edgeThreshold=10,
                                sigma=1.6
                                )
```

Figura 23 – Construtor do algoritmo SIFT, com valores de parâmetros padrão

O parâmetro *nfeatures* indica quantas *features* se quer detectar. Valores altos impactam no desempenho do algoritmo. O parâmetro *nOctaveLayers* diz ao algoritmo o número de camadas que deve ser usada na Diferença de Gaussianas. Valores mais altos aumentam a precisão e a carga de processamento. O parâmetro *contrastThreshold* é usado como limite para descartar *features* com valores inferiores ao do parâmetro. Quanto maior seu valor, menos *features* são produzidas pelo detector. O parâmetro *edgeThreshold* é usado para indicar ao algoritmo qual peso é necessário para uma *feature* ser considerada como canto. Ao contrário do outro limite, valores altos produzem mais *features*. Por fim, o parâmetro *sigma* é usado pela Diferença de Gaussianas, na camada inicial. Se a imagem está um pouco desfocada, valores mais baixos podem ajudar a corrigir esse problema.

4.1.4 SURF

SURF (BAY; TUYTELAARS; VAN GOOL, 2006), do inglês *Speeded-Up Robust Features*, foi desenvolvido com a intenção de ser uma versão mais rápida do algoritmo SIFT. Em todos os passos, alguma técnica diferente foi aplicada para aumentar a rapidez com que as *features* são detectadas. Ao passo que o algoritmo SIFT usa Diferença de Gaussianas, o algoritmo SURF usa filtros lineares, que são mais rápidos para serem processados. Para detectar objetos com tamanhos variados e suas posições, o SURF usa determinante de matrizes Hessianas. Matrizes Hessianas são matrizes quadradas compostas a partir das derivadas parciais de segunda ordem de uma função. São usadas para encontrar pontos críticos e em problemas de otimização. Para calcular as orientações das *features*, o SURF aplica *wavelets*. O algoritmo SURF tem limitações, no entanto. Por exemplo, mudanças de luminosidade não são bem tratadas e podem causar perda de *features* e de rastreamento.

No OpenCV, o algoritmo SURF possui cinco parâmetros em seu construtor, como visto na Figura 24.

```
cv2.xfeatures2d.SURF_create(  
                                hessianThreshold,  
                                nOctaves=4,  
                                nOctaveLayers=2,  
                                extended=True,  
                                upright=False  
                                )
```

Figura 24 – Construtor do algoritmo SURF, com valores padrão

O parâmetro `hessianThreshold` tem influência nas *features* que serão detectadas, pois o algoritmo usa esse valor como limite. Somente as *features* com determinante da matriz Hessiana maior que o valor desse parâmetro serão salvas. Valores muito altos limitam demais a quantidade de *features* detectadas. O intervalo de valores recomendados pela documentação do OpenCV é entre 300 e 500. O parâmetro `nOctaves` determina quantos níveis de escala serão usados para detectar *features*. Valores altos fazem o algoritmo detectar *features* de objetos maiores. O valor `nOctaveLayers` faz cada nível de detecção ter mais do que uma camada, refinando as *features* encontradas. Valores baixos aumentam o desempenho do algoritmo. O parâmetro `extended` determina se será usado o valor padrão de elementos para o descritor das *features*, de 128. Caso não se queira usar o padrão, o detector usa 64 elementos, aumentando a velocidade do algoritmo. O parâmetro `upright` diz ao SURF se é necessário calcular as

orientações para cada uma das *features*. O valor True não calcula as orientações, o que deixa o algoritmo ainda mais rápido.

4.1.5 FAST

O algoritmo FAST (*Features from Accelerated Segment Test*) foi proposto por Rosten e Drummond (2006) e tem como objetivo detectar *features* de maneira eficiente, com alto desempenho. Embora consiga essa detecção, o algoritmo não é robusto em situações de ruídos na imagem e depende de um valor de *threshold* que deve ser introduzido manualmente.

O funcionamento do FAST obedece a regras simples e rápidas de serem executadas. Na Figura 25 é possível ver a ilustração do método em funcionamento e no modo como analisa os pixels.

Figura 25 – Funcionamento do algoritmo (FAST ALGORITHM FOR DETECTION, 2014)

Primeiro, são determinados os 16 pixels em um círculo em volta do pixel central p . Analisam-se os quatro pixels nos quatro pontos cardeais principais: norte (1), sul (9), leste (5) e oeste (13). Se pelo menos três desses pixels possuírem intensidade maior ou menor (com a soma ou subtração de um valor de *threshold*) que do pixel central ao mesmo tempo, então o pixel central é considerado como uma *feature* em potencial. No exemplo da Figura 25, os pixels nas direções norte, leste e oeste possuem intensidade menor. Se essas condições não existirem, o pixel é descartado como *feature*. Caso as condições aconteçam, são analisados os 16 pontos em volta do pixel central: se 12 desses pontos possuírem intensidade maior ou menor, também levando em consideração o valor do *threshold*, e se forem contíguos uns aos outros, então o ponto é definido como *feature*. No exemplo da Figura 25, os pixels 11, 12, 13, 14, 15, 16, 1, 2, 3, 4, 5 e 6 possuem todos uma intensidade menor e são contíguos. Portanto, o pixel p é considerado uma *feature*. Para aumentar a eficiência na detecção, é usado um método de

aprendizado de máquina, com treinamento anterior feito em imagens parecidas com a qual se deseja analisar. Também é usado um método para suprimir *features* que tenham sido detectadas muito próximas umas às outras.

O algoritmo FAST foi implementado na biblioteca OpenCV e possui alguns parâmetros para se ajustar. É possível ver os parâmetros na Figura 26.

```
cv2.FastFeatureDetector_create(  
                                threshold=10,  
                                nomaxSuppression=True,  
                                neighborhood=2,  
                                )
```

Figura 26 – Construtor do algoritmo FAST, com valores padrão

O parâmetro *threshold* define o intervalo de luminosidade que será aceito na detecção das *features*. Valores altos fazem com que muitos pixels sejam considerados como cantos. O parâmetro *nomaxSuppression* indica se deseja usar o passo final do algoritmo, para diminuir a quantidade de *features* próximas umas às outras. O parâmetro *neighborhood* indica qual o tamanho do círculo que deve ser analisado em volta do pixel central. Os valores possíveis são: zero, para um círculo com tamanho de 8 pixels; um, para um círculo com tamanho de 12 pixels; dois, para um círculo de tamanho de 16 pixels.

Analisando a literatura disponível, decidiu-se por adotar o método de detecção de *features* proposto por Shi-Tomasi. Jain *et al.* (2015) obtiveram 80% de precisão na detecção usando Shi-Tomasi em seu método. Também em 2015, Mendes, Bianchi e Junior (2015) usaram o método de Shi-Tomasi e obtiveram 75% de precisão em suas detecções de veículos em cruzamentos urbanos. Li *et al.* (2016) aplicam Shi-Tomasi por causa de sua eficiência. Seu método proposto detecta 86% dos objetos. Shih e Zhong (2017), também escolheram o método de Shi-Tomasi por causa de sua eficiência computacional. O método proposto rastreou corretamente mais de 90% dos veículos.

4.2 TRACKING DE FEATURES

A detecção das *features* é o primeiro passo de um sistema de rastreamento. Para conseguir extrair as informações, é necessário um algoritmo capaz de ligar o movimento das *features* detectadas por todos os *frames*. Métodos recentes de rastreamento de veículos baseados na detecção de *features* usam técnicas de fluxo ótico para manter a relação entre *features* através do tempo, na mudança entre *frames*. Lin *et al.* (2012) obtiveram 85% de precisão na detecção

e no rastreamento de veículos em pontos cego. Mendes, Bianchi e Junior (2015) usaram fluxo ótico para rastrear as *features* detectadas, alcançaram 75% de precisão nas detecções. Souza Castro *et al.* (2016), no Brasil, usaram fluxo ótico na detecção e rastreamento de veículos, detectando 43 dos 45 veículos gravados em seus testes em rodovias. Li *et al.* (2016) obtiveram 86% de taxa de exatidão na detecção e *tracking* de veículos, com imagens geradas a partir de veículos aéreos não-tripulados (VANT). Shih e Zhong (2017) avaliaram a detecção e o *tracking* de *features* com fluxo ótico em situações com alta ocorrência de oclusões de veículos. Obtiveram 70% de precisão nas detecções. Ruimin *et al.* (2017) conseguiram detectar 85% dos veículos filmados, também por VANTs. Usaram o algoritmo Shi-Tomasi para detecção de *features* e fluxo ótico para o *tracking*. Portanto, a análise dos resultados desses trabalhos ajudou na escolha do fluxo ótico como o algoritmo de *tracking* deste trabalho.

No OpenCV 3.2, está disponível o algoritmo de Lucas-Kanade (1981), que implementa rastreamento por fluxo ótico. A ideia do método é simples: procurar, no *frame* seguinte, por uma combinação de pixels similar a *feature* atual, usando equações de fluxo ótico, em uma vizinhança pré-definida. A combinação de pixels é dada por uma região de 3x3 pixels em volta da *feature* selecionada e a busca ocorre em uma área definida pelos parâmetros de entrada da função no OpenCV. O algoritmo, então, retorna um vetor com todas as novas coordenadas das posições das *features* detectadas.

Na biblioteca OpenCV, a função possui parâmetros que podem ser ajustados para se refinar a funcionalidade do algoritmo, como é possível observar na Figura 27.

```
cv2.calcOpticalFlowPyrLK(
    prevImg,
    nextImg,
    prevPts,
    nextPts,
    winSize=(21,21),
    maxLevel=3,
    criteria=(cv2.TERM_CRITERIA_EPS
| cv2.TERM_CRITERIA_COUNT, 30,
0.01)
)
```

Figura 27 – Parâmetros do algoritmo de fluxo ótico de Lucas-Kanade

O parâmetro *prevImg* recebe o *frame* anterior, em escala de cinzas, do vídeo processado. O parâmetro *nextImg* recebe o *frame* atual do vídeo, também em escalas de cinza. Os dois *frames* são importantes pois o algoritmo precisa saber a combinação de pixels em volta

das *features* no *frame* anterior para buscar a mesma combinação no *frame* atual. O parâmetro *prevPts* recebe as coordenadas das *features* selecionadas. O parâmetro *nextPts* recebe a previsão da localização das *features* no *frame* atual, para facilitar no rastreamento. Esse parâmetro não é obrigatório e pode receber um valor nulo. O parâmetro *winSize* denota o tamanho da janela em volta da *feature* na qual a combinação de pixels deve ser procurada. Valores altos afetam o desempenho do algoritmo, pois obrigam a procura em regiões muito grandes. O parâmetro *criteria* define qual o critério de parada na busca pela combinação de pixels. Duas opções estão disponíveis: parar depois de uma determinada quantidade de tentativas, usando a *flag* `cv2.TERM_CRITERIA_COUNT`; e parar depois que a janela de busca se move menos que um valor determinado, com a *flag* `cv2.TERM_CRITERIA_EPS`. As duas *flags* podem ser usadas em conjunto, o que, inclusive, é o comportamento padrão do algoritmo.

4.3 AGRUPAMENTO DE *FEATURES*

Após a detecção e o *tracking* das *features*, um método de similaridades deve ser capaz de agrupá-las em hipóteses de veículos. *Features* mais próximas umas às outras tendem a se mover da mesma forma, pois estão todas sob influência da mesma fonte de movimento: um mesmo veículo. Por isso, Coifman *et al.* (1998) afirmaram, em seu tradicional trabalho na área de Engenharia de Transportes, que as *features* pertencentes à um mesmo veículo possuem velocidades longitudinais e laterais parecidas umas às outras e que pequenas variações nessas velocidades são suficientes para diferenciá-las. Também afirmaram que, em situações de congestionamentos, os veículos sempre estão acelerando e diminuindo a velocidade, fazendo com que cada veículo apresente um padrão diferente de movimento. Em situações de fluxo livre, os veículos tendem a se posicionar de forma a ter mais espaçamento entre eles. Portanto, para Coifman *et al.* (1998), duas características dos movimentos podem ser usadas para agrupar *features* de veículos: a velocidade do movimento, e a distância entre as *features*.

Saunier e Sayed (2006) usaram duas distâncias para agrupar *features* de um mesmo veículo. Calculam tanto a distância mínima quanto a distância máxima que cada par de *feature* já teve em suas trajetórias. Esses valores são atualizados a cada *frame* do vídeo e somente são armazenados se forem maior ou menor que o máximo ou o mínimo anterior. Se a diferença entre essas duas distâncias é menor que um valor pré-determinado, as *features* são agrupadas e classificadas como pertencendo ao mesmo veículo. Mandellos, Keramitsoglou e Kiranoudis (2011) agrupam as *features* dos veículos baseados em seus movimentos relativos e em suas distâncias. A cada nova *feature* conectada, criam uma região preenchida, limitada pelas

features, para verificar dimensões e detectar oclusão. Caso uma dessas regiões seja maior que um valor pré-definido, são divididas, baseando-se no histórico de movimento das *features*. Mais recentemente no Brasil, Souza Castro *et al.* (2016) fazem o agrupamento baseado nas distâncias Euclidianas entre as *features*. Para descartar *features* que não pertencem a veículo algum, usam a velocidade de movimento em vários *frames*. Caso as *features* se movimentem em direções contrárias ou em ângulos diferentes do esperado, são descartadas.

Neste trabalho, o agrupamento das *features* em possíveis veículos é feito com a aplicação de duas regras, que se complementam. Primeiro, calcula-se a distância Euclidiana entre todas as *features* detectadas. Para calcular a distância D entre dois pontos $P = (p_x, p_y)$ e $Q = (q_x, q_y)$, usa-se a equação (4.2).

$$D = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}. \quad (4.2)$$

A distância Euclidiana tem como função juntar *features* que estão perto uma das outras e que pertençam a um mesmo veículo e a desconectar as que estejam muito distantes uma das outras. Porém, para veículos grandes, nem todas as *features* detectadas estão próximas umas às outras e, em situações de oclusão, nem todas pertencem a um mesmo veículo. Para resolver esses problemas, é usada a velocidade de movimento de cada *feature*. A velocidade pode ser calculada de duas maneiras distintas. É possível ignorar a componente do movimento no eixo x , caso se tenha certeza que os veículos possuam movimento predominante no eixo y . Ou pode-se calcular a velocidade vetorial, usando o movimento das duas componentes (x e y). As duas maneiras foram avaliadas neste trabalho. Porém, a velocidade vetorial foi usada para todos os vídeos, por apresentar melhor desempenho no agrupamento em testes realizados nos próprios vídeos disponíveis.

Contudo, vídeos de tráfego veicular apresentam distorções nas imagens criadas pela perspectiva de filmagem. Objetos mais próximos à câmera são maiores, ao passo que objetos distantes, são menores. Um algoritmo que use distâncias para agrupar *features* não vai ser capaz de realizar o agrupamento caso as dimensões dos mesmos objetos sejam representadas usando quantidades diferentes de pixels. Esse problema pode ser resolvido aplicando-se uma transformação de perspectiva.

4.4 TRANSFORMAÇÃO DE PERSPECTIVA

A transformação de perspectiva é uma função que relaciona cada ponto 3D do mundo real a um ponto 2D em uma imagem (HEIKKILA; SILVEN, 1997). Essa relação entre os pontos permite o cálculo de medidas reais, como distâncias e velocidades, a partir das medidas abstratas de uma imagem digital, como número de pixels e quantidade de pixels por *frame* (KANHERE, 2008).

Especificamente em imagens de vídeo de monitoramento de tráfego, os veículos possuem movimento predominante somente no plano da superfície do pavimento. Essa característica simplifica o processo de transformação da perspectiva, pois limita as informações necessárias para se calcular dimensões reais (Worrall *et al.*, 1994).

No OpenCV 3.2, em Python, duas funções são utilizadas em conjunto para fazer a transformação de perspectiva. A primeira das funções pode ser vista na Figura 28 junto com seus parâmetros.

```
M = cv2.getPerspectiveTransform(  
                                ptsImage,  
                                ptsTransform  
                                )
```

Figura 28 – Parâmetros da primeira função da transformada

Essa primeira função tem o propósito de pegar os pontos selecionados na imagem do mundo real, `ptsImage`, e gerar a matriz `M` que mapeia os pontos na imagem transformada. O parâmetro `ptsTransform` indica a ordem no qual os pontos serão mapeados e suas distâncias máximas, em pixels. A ordem desse parâmetro de entrada deve ser a mesma do parâmetro anterior, `ptsImage`. Desse modo, qualquer pixel que estiver dentro da região selecionada em `ptsImage` vai ter um correspondente na imagem transformada, mapeado pela matriz gerada.

A segunda função do OpenCV pode ser vista na Figura 29, junto com seus parâmetros.

```
frame = cv2.warpPerspective(  
                                image,  
                                M,  
                                (maxWidth,maxHeight)  
                                )
```

Figura 29 – Parâmetros da segunda função usada na transformação de perspectiva

Essa função tem o propósito de transformar a imagem original, passada no parâmetro *image*, junto com a matriz gerada no passo anterior, *M*, limitada pelos tamanhos máximos definidos em (*maxWidth,maxHeight*), para criar a imagem com a transformação de perspectiva, retornada em *frame*. Os tamanhos máximos passados aqui devem ser iguais aos usados na função anterior e representados pelo parâmetro *ptsTransform*.

4.5 MÉTODO PROPOSTO

O objetivo do método proposto nesse capítulo é detectar as *features* dos veículos, agrupá-las, rastrear seus movimentos, e passar os dados extraídos para a próxima fase do algoritmo. O fluxograma do método pode ser visto na Figura 30.

Figura 30 – Fluxograma da detecção e *tracking* de *features*

O frame do vídeo entra como parâmetro para que o recorte da região de interesse (ROI) possa ser feito, usando as configurações definidas pelo usuário. A ROI tem a função de limitar, no frame do vídeo, a região na qual os veículos se movimentam, para aumentar o desempenho. Com a região recortada, o algoritmo faz a transformação de perspectiva e passa o *frame* resultante para o método de modelagem de *background* escolhido (MOG2), que gera um *frame* de detecção, com os veículos já segmentados. Esse *frame* entra no método de detecção de *features* e gera uma lista de *corners*. Essa lista é passada para o método de rastreamento e agrupamento, que detecta, no próximo *frame* do vídeo, o local para qual os *corners* se moveram.

As coordenadas do movimento são então agrupadas em veículos independentes e os dados são enviados para a próxima fase do algoritmo.

Todos os vídeos do item 2.5, com suas características presentes na Tabela 1, foram processados e analisados nessa fase do trabalho.

4.5.1 ROI E TRANSFORMAÇÃO DE PERSPECTIVA

Em tráfego de rodovias, a entrada de veículos novos na cena acontece normalmente em áreas bem definidas. Como as câmeras foram posicionadas em lugares fixos, apontando para a rodovia, uma grande área da imagem é composta por elementos que não são importantes para o rastreamento dos veículos. Por isso, o primeiro passo do algoritmo proposto foi a definição da região de interesse. Essa região deve englobar as faixas de tráfego de um dos sentidos, ignorando regiões adjacentes que não possam conter veículos, como por exemplo, o canteiro central, vegetação, céu, ou regiões lindeiras a rodovia. As regiões podem conter pedaços do acostamento, para acomodar a perspectiva da câmera. Exemplos de ROIs usadas neste trabalho podem ser vistas na Figura 31.

Figura 31 – (a) ROI no vídeo da SP-310 (b) ROI no vídeo da BR-116

A ROI específica para cada vídeo deve ser definida pelo usuário antes da execução do algoritmo e, então, o retângulo selecionado é inserido no código em Python, nas variáveis corretas. As coordenadas devem ser baseadas no *frame* original do vídeo analisado e não possuem ordem de colocação específica.

Com as informações da ROI, é necessário corrigir a perspectiva gerada pelo posicionamento da câmera. Para que o algoritmo proposto tenha um desempenho melhor, é interessante que as faixas de tráfego estejam orientadas de forma a ficar na vertical. Esse tipo

de orientação é necessário pois a ROI é processada em áreas divididas e fixas nos próximos passos do algoritmo (seção 4.5.4).

No caso deste trabalho, as ROIs das imagens originais são transformadas em um *frame* quadrado, com 300x300 pixels, com dimensões conhecidas no mundo real. Desse modo, é possível, a partir da imagem gerada pela nova perspectiva, calcular a distância percorrida por cada veículo e sua velocidade. O tamanho fixo do *frame* da transformada ajuda a padronizar as resoluções dos vídeos de entrada, padronizando também as distâncias entre as *features* detectadas.

Na Figura 32, a imagem transformada pela função pode ser vista, junto com o *frame* original.

Figura 32 – ROI e Transformadas (a) SP-310 (b) BR-116

4.5.2 REMOÇÃO DE *BACKGROUND*

Como conclusão do capítulo anterior, foi proposto o uso do algoritmo MOG2 para executar a remoção do *background* e a segmentação dos veículos. A remoção do *background* é aplicada somente na região de interesse, para diminuir o impacto da técnica na performance do algoritmo como um todo. Após a aplicação do MOG2, uma função de *threshold* é usada para

binarizar a imagem, fazendo com que só existam pixels com valor 0, preto, ou 255, branco. A função de *threshold* usada e seus parâmetros podem ser vistos na Figura 33.

```
cv2.threshold(  
    img,  
    thresh,  
    maxVal,  
    type  
)
```

Figura 33 – Função de *threshold* e seus parâmetros

O parâmetro *img* recebe o *frame* que se deseja processar. O parâmetro *thresh* indica qual o valor limite para a operação. O parâmetro *maxVal*, na operação de *threshold* binário, indica para qual valor todos os pixels que tiverem valor maior ou menor que o parâmetro *thresh* irá receber. Neste trabalho, os parâmetros usados foram *thresh=60* e *maxval=255* para todos os vídeos. Portanto, todos os pixels que tiverem valor maior que 60 receberão valor 255, assumindo a cor branca, destacando o veículo. O valor do parâmetro de *threshold* foi definido através de testes com todos os vídeos, a partir de análises visuais dos resultados da remoção de *background*, como visto na Figura 34. Com *thresh=60*, pixels fora dos objetos são pintados de preto. O parâmetro *type* indica qual tipo de *threshold* se deseja fazer. Neste trabalho, foi usado o valor *cv2.THRESH_BINARY*, transformando a imagem em puramente preto e branco.

Figura 34 – Resultado da variação do valor do parâmetro *thresh*

O resultado da remoção de *background* e do *threshold* aplicados em na ROI do *frame* do vídeo da BR-116 pode ser visto na Figura 35.

Figura 35 – Remoção de *background* e *threshold* (1) ROI no *frame* original (2) Resultado

O algoritmo MOG2 conseguiu segmentar a cena, sem ruídos, destacando os veículos. O *threshold* ajudou a destacar os pixels classificados como *foreground* e a limpar qualquer ruído que possa ter passado pela remoção de *background*. A transformação da perspectiva fez com que os veículos ficassem orientados no sentido norte-sul, na vertical, e normaliza as distâncias entre os pixels, corrigindo a perspectiva. Embora os veículos não estejam completamente delineados, com espaços vazios dentro da detecção, o algoritmo de *tracking* detecta somente *features*, sem necessidade de regiões preenchidas.

4.5.3 TRACKING

Com os objetos segmentados, foi necessário aplicar o algoritmo de detecção de *features* e o algoritmo responsável por rastrear esses pontos através do tempo. Neste trabalho, o método de Shi-Tomasi foi o escolhido para a detecção das *features* depois de análise da literatura atual. Mendes, Bianchi e Junior (2015) obtiveram resultados de 75% de precisão em suas detecções. No Brasil, Souza Castro *et al.* (2016), afirmam que o algoritmo Shi-Tomasi, quando usado em conjunto com o algoritmo Lucas-Kanade, é capaz de detectar *features* rapidamente, desde que o movimento não seja grande de um *frame* para o outro. Com ROIs grandes o suficiente, o movimento entre os *frames* é pequeno. Shih e Zhong (2017) obtiveram 90% de precisão na contagem de veículos usando Shi-Tomasi.

Para realizar o *tracking*, a ROI foi dividida em três áreas que denotam a fase do movimento dos veículos (Figura 36). A área I é denominada detecção de *features* e foi definida com um tamanho de 1/3 da ROI (100 pixels). Na área I, os veículos entram em cena e têm suas *features* detectadas pelo algoritmo de Shi-Tomasi. A restrição da detecção de *features* na área I fez com que o tempo de processamento fosse menor do que seria se a detecção fosse feita a

cada novo *frame* e em toda a ROI. Em seguida, a área II (Figura 36), denominada área de rastreamento, foi definida compreendendo quase o restante da ROI (185 pixels). Nesta área, o algoritmo de *tracking* de fato realiza o rastreamento das *features* e aplica o método de agrupamento, atualizando dinamicamente quais *features* fazem parte de quais veículos. Nesta área II, também são calculadas as velocidades, com a trajetória do centroide do veículo sendo salva *frame a frame*. Por fim, a área III (Figura 36), de aproximadamente 5% do total (15 pixels), foi definida com o propósito de representar uma linha de chegada, finalizando o percurso dos veículos. Assim que os veículos cruzam essa linha, as informações são consolidadas e salvas em arquivos de relatório, para análise posterior.

Figura 36 – Divisão da ROI (I) Detecção (II) *Tracking* (III) Salvamento

Veículos que atingem a linha da área III têm todos os seus dados de rastreamento apagados dos vetores do algoritmo, para evitar o uso excessivo de memória computacional. As informações de cada uma das *features* que fazem parte de um único veículo foram agrupadas e salvas como um único valor de centroide para cada *frame*. Dessa forma, cada veículo possuía uma única trajetória.

4.6 RESULTADOS OBTIDOS

Todas as funções usadas para completar o passo de rastreamento possuem seus parâmetros. Como o objetivo deste trabalho é a construção de um método que seja capaz de extrair dados e informações de vídeos, e a demanda de tempo para fazer análises de

sensibilidade dos parâmetros é alta, as funções de detecção e de *tracking* das *features* foram usadas com seus valores padrão, sem alteração.

Como a transformada sempre faz com que o *frame* analisado tenha 300x300 pixels, as distâncias, independentemente dos vídeos, são sempre muito parecidas. Justamente por isso, e analisando o tamanho em pixel que a maioria dos veículos ocupa no *frame* da transformada para todos os vídeos, o valor da distância de agrupamento foi definido como 60 pixels. Essa quantidade de pixel não representa medidas constantes no mundo real, pois esses valores são dependentes do tamanho da região de interesse, mas é suficiente para que um veículo de passeio seja agrupado corretamente.

A velocidade das *features* é usada no agrupamento como um intervalo. *Features* que estejam se movimentando com velocidades similares são agrupadas. Neste trabalho, o intervalo adotado foi de 10%, pois os vídeos analisados são de tráfego veicular rodoviário, em trechos retos. Um intervalo maior causaria *features* de veículos com velocidades muito diferentes se agrupando. Já um intervalo menor gera veículos fantasmas, dividindo as *features* de um só veículo em dois grupos.

Portanto, com a aplicação dessas duas regras, se uma *feature* está dentro da distância de agrupamento e se movimenta com velocidade entre 90% e 110% da sua vizinha, elas são agrupadas em um mesmo veículo.

Os resultados do *tracking* em todos os vídeos analisados podem ser vistos nas próximas figuras, organizados pelo ângulo da câmera no qual foram filmados. Para todos os vídeos, os pontos das trajetórias de cada uma das *features* é marcada com um traço verde. O resultado do algoritmo de agrupamento é mostrado como um quadrado azul. Em teoria, cada quadrado azul deve incluir toda a dimensão do veículo sendo rastreado. Também são mostrados os centroides de cada veículo, denotados com um círculo azul.

O resultado do *tracking* nos vídeos em perspectiva pode ser visto na Figura 37.

Perspectiva

Figura 37 – Resultado do *tracking* nos vídeos em perspectiva

Na rodovia BR-116 foram detectados quatro veículos dentro da região de interesse, delimitados pelo quadrado azul. Na rodovia SP-310, somente um veículo se encontra no *frame*. Não foram detectados nenhum veículo a mais, porém é possível ver que alguns veículos escapam da delimitação do quadrado azul, indicando a falta de *features* detectadas na posição. É possível notar também que o primeiro veículo detectado na rodovia BR-116 possui movimento lateral em sua trajetória, indicando uma provável mudança de faixa antes de entrar na região de interesse.

O resultado do *tracking* dos vídeos capturados na Avenida Padre Francisco Sales Colturato, tanto no contra fluxo quanto a favor do fluxo veicular, podem ser vistos na Figura 38.

Frontal

Figura 38 – Resultado do *tracking* nos vídeos frontais, na Av. Colturato

Nos *frames* da Figura 38, os veículos estão bem espaçados entre si, facilitando a detecção e o *tracking* dos dados. É possível notar, no *frame* a favor do fluxo, que existe um veículo no topo da imagem, com *features* detectadas somente de sua porção traseira. Analisando o *frame* original, nota-se um veículo pesado mais a frente, indicando um possível agrupamento com falha. Porém, nesse caso, o algoritmo de agrupamento reconheceu a falha, conseguindo dividir as *features* em dois veículos logo antes das informações serem consolidadas, corrigindo o erro a tempo de registrar dois veículos.

Os vídeos capturados na rodovia SP-310, ao meio dia e ao entardecer, podem ser vistos na Figura 39. Em ambos exemplos, os veículos aparecem devidamente segmentados, sem agrupamentos errados.

Frontal

Figura 39 – Resultado do *tracking* nos vídeos frontais, na SP-310

O vídeo com ângulo de captura lateral, na rodovia SP-310, ao entardecer e com sol forte, pode ser visto na Figura 40.

Lateral

Figura 40 – Resultado do *tracking* no vídeo lateral, na SP-310

É possível notar que, independentemente do ângulo de filmagem da câmera e da direção do tráfego, os veículos sempre aparecem na parte inferior da imagem e saem pela parte superior. Isso ocorre pois, a região de interesse foi programada com regiões fixas de análise, como visto na seção 4.5.4, padronizando o funcionamento do algoritmo independentemente da posição da câmera. Essa consideração evitou a necessidade de criar uma configuração de ROI diferente para cada direção de entrada de veículos na imagem, além de exigir que o usuário entrasse com um parâmetro a mais no programa, com a direção do tráfego.

Essa padronização da ROI também fez com que os dados de saída se comportem da mesma maneira, com as trajetórias começando com valores grandes no eixo Y e terminando com valores pequenos. É justamente essa característica que facilita o processamento de uma das saídas de dados, discutidas no capítulo 5: o diagrama espaço-tempo. Com dados sempre com as mesmas características, a direção do eixo Y sempre fica igual, sem necessidade de inversão.

4.7 CONCLUSÕES

Neste capítulo, foi exposto o método usado para se rastrear e agrupar as *features* detectadas pelos algoritmos usados. A combinação das técnicas de remoção de *background*, de detecção de *features*, de *tracking* e de agrupamento formam o cerne deste trabalho, pois determinam a qualidade dos dados que serão identificados. Todas as informações retiradas automaticamente através dos vídeos dependem do bom desempenho dessas técnicas. Um método que não consegue manter a sequência das *features* detectadas não consegue calcular as informações que se desejam extrair dos veículos, como trajetória e velocidade. Além disso, não agrupar os veículos de forma correta pode transformar todas as informações retiradas em informações sem utilidade, pois passam a não possuir significado no mundo real.

A combinação dos métodos utilizados neste trabalho conseguiu extrair dados do tráfego veicular, gerando arquivos de relatório, explicados no capítulo 5, que foram usados para validar as informações geradas nesse passo do algoritmo.

A segmentação, feita pelo algoritmo MOG2, seguiu os resultados obtidos no capítulo 3, gerando imagens com os veículos destacados em branco. O valor fixo escolhido para o *threshold*, usado para todos os vídeos, teve um bom desempenho médio, ajudando a destacar o *foreground*. O resultado da combinação do MOG2 e do *threshold* pode ser visto na Figura 41, com *frames* de alguns dos vídeos.

Figura 41 – *Frames* de exemplo do desempenho do MOG2 com aplicação de *threshold*

É importante notar que o valor do *threshold* foi definido através de observações feitas em todos os vídeos utilizados. Uma comparação mais rigorosa, com testes mais extensivos, pode gerar resultados ainda melhores na segmentação dos veículos. Esses testes possuem uma alta demanda de tempo e podem ser mais bem explorados em trabalhos futuros.

As etapas de detecção e de *tracking* das *features* acontecem em sucessão direta, pois o produto gerado pelo algoritmo Shi-Tomasi de detecção de *features* é usado, sem modificações ou filtragens, no algoritmo de fluxo ótico, de *tracking*. Por ser a fase que mais consome processamento, o fato do algoritmo proposto aplicar as funções somente na região de interesse melhora o desempenho. Porém, mesmo com essas alterações, o algoritmo não consegue processar vídeos em tempo real. Nos vídeos utilizados, o algoritmo não teve dificuldades em encontrar as *features* dos veículos de passeio que entravam na ROI. A dificuldade de detecção ocorria em veículos pesados, pois como o algoritmo leva em conta alterações bruscas de intensidade dos pixels, veículos com cores constantes e que ocupam grande parte da ROI não têm nenhuma *feature* intermediária detectada, como pode ser visto na Figura 42.

Figura 42 – Veículo com falha de detecção por falta de *features*

Esse problema na detecção pode ser mitigado se a detecção das *features* for liberada para ocorrer em toda a ROI. Porém, o desempenho do algoritmo piora notavelmente, pois mais informações são geradas a cada *frame* e devem ser interpretadas pelo algoritmo de rastreamento e agrupamento. Outra alternativa seria detectar novas *features* em intervalos de *frames*, como de 5 em 5 *frames*, e liberar a área de detecção. Porém, o desempenho dessa alternativa também foi bem pior que o implementado.

As maiores falhas aconteceram na lógica de agrupamento das *features*. Veículos pesados que não entram inteiramente na ROI antes de serem consolidados, geram detecções falsas de múltiplos veículos. Alguns desses casos podem ser vistos na Figura 43 e na Figura 44, com a apresentação de *frames* de exemplo dessas situações.

Figura 43 – Detecções múltiplas de um mesmo veículo pesado

O algoritmo consegue agrupar corretamente os veículos diferentes que aparecem no *frame*, mas falha quando o veículo pesado ainda não saiu completamente da área de captura de

novas *features* (área I), quando atinge o ponto de consolidação das informações extraídas (área III).

Figura 44 – Detecções múltiplas em veículos pesados

Na Figura 44 o mesmo ocorre, mas em outros dos vídeos testados. O veículo pesado chega a ser detectado como três veículos diferentes.

Esses erros ocorreram quando os veículos chegaram na porção superior do *frame*, e suas *features* foram apagadas dos vetores de *tracking*. Porém, a porção traseira do veículo ainda estava dentro da área de detecção de novas *features* (área I). Com isso, *features* são detectadas na parte traseira dos veículos e o algoritmo cria um novo agrupamento, associando-o a um novo identificador veicular na cena. Esse tipo de falha é o que mais afeta o desempenho do algoritmo, indicando que veículos longos devem ser tratados de forma diferente dos demais veículos.

Outro tipo de falha de agrupamento ocorre quando veículos estão muito próximos uns aos outros, ou em situações de oclusão. Exemplos podem ser vistos na Figura 45.

Figura 45 – Agrupamento de vários veículos em um só

A maioria dos erros de agrupamento das *features* de mais de um veículo acaba sendo resolvido pela diferença de velocidade. Uma ROI maior permite ao algoritmo rastrear as velocidades das *features* por mais tempo, aumentando a probabilidade de detectar mudanças de velocidade.

Mesmo com essas falhas, o algoritmo foi capaz de detectar e rastrear os veículos nos vídeos. Os resultados desse rastreo são apresentados no capítulo 5.

5 IDENTIFICAÇÃO DO COMPORTAMENTO

Projetos na área de Engenharia de Transporte são custosos e podem ter um impacto muito grande na vida dos habitantes locais. Por isso, extrair informações e comportamentos do fluxo de tráfego é necessário pois permite aos administradores e operadores de sistemas de tráfego a tomada de decisões de modo eficiente.

Uma discussão bem antiga no campo da computação e do conhecimento é sobre a diferença entre dados, informações e conhecimento. Teskey (1989) já propunha, em sua época, alterações nas definições criadas anteriormente, por Cooley (1987). Para Teskey (1989), dados são o resultado direto de observação de um evento, por exemplo, os valores do atributo de um objeto. Informação é uma coleção estruturada de dados, enquanto conhecimento é um modelo do mundo, que pode ser criado ou modificado com o uso de novas informações. Outros autores vêm propondo definições diferentes ao longo dos anos, com Aamodt e Nygård (1995), propondo alterações principalmente na definição de informação (dados com significado). Chen *et al.* (2009) propuseram que dados são representações computadorizadas de modelos e atributos de objetos reais ou simulados; informações são dados vindos de resultados de análises estatísticas ou de outros processos computacionais; e que conhecimento representa o resultado de um processo cognitivo, como percepção ou aprendizado. Mais recentemente, Cooper (2017) disse que dados são valores, informações são dados com contexto, e que conhecimento é uma informação estruturada por um processo cognitivo e por validação.

No campo de Engenharia de Transportes, é possível fazer uma associação entre essas definições e termos comumente usados na área. Velocidades e contagens podem ser classificados como dados pois precisam de contexto para completar seus significados. Isolados, sem outras informações pré-concebidas, não possuem significado. O dado 120 km/h, sem contexto sobre veículo, pavimento ou situação, não tem significado. Para que esse dado possa começar a fazer sentido, é necessário que seja agregado o contexto no qual ele está inserido. Por exemplo, 700 veículos de passeio por hora, com velocidade média de 120 km/h, em uma rodovia pavimentada, na pista norte. Desse modo, o conjunto desses dados em específico passa a ser informação.

Portanto, embora todos os passos anteriores do algoritmo sejam importantes, a organização dos dados e a extração de informação e conhecimento é mais importante ainda. Em

Engenharia de Transportes, o método mais tradicional de se organizar os dados é através da criação de um diagrama Espaço-Tempo.

5.1 DIAGRAMA ESPAÇO-TEMPO

O diagrama espaço-tempo é uma ferramenta usada em questões de planejamento de sistemas de transportes, desde a área da aviação, em organização do tráfego da pista de pouso para determinar o limite de fluxo de aviões (REYNOLDS *et al.*, 2003), até na área de planejamento de semáforos (KOONCE *et al.*, 2008). O diagrama espaço-tempo descreve o movimento dos veículos em escala do tempo e do espaço (ANWAR; ZENG; ARISONA, 2014).

O modo com que o diagrama é construído faz com que informações importantes para o planejamento sejam vistas e reconhecidas de maneira simples. Em um eixo, são plotadas informações sobre o tempo decorrido, a partir de um referencial. No outro eixo, são plotadas informações sobre o posicionamento dos veículos, também com um determinado referencial. O conjunto dessas duas informações, plotadas dessa maneira, faz com que várias informações importantes possam ser calculadas facilmente. Um exemplo de diagrama pode ser visto na Figura 46, com 6 veículos sendo registrados. É possível observar como as principais informações podem ser calculadas.

Figura 46 – Diagrama Espaço-Tempo

O espaçamento é a distância no eixo Y que separa duas curvas. Indica, na unidade de medida do eixo, qual o espaço físico entre dois veículos em um determinado instante de tempo.

O tempo de viagem de um veículo pode ser calculado com a subtração do tempo de saída da área de medição do diagrama, t_f , pelo tempo no qual o veículo entra na área de medição, t_i . É basicamente uma medida do tempo que um veículo leva para percorrer a distância que o diagrama cobre.

O *headway* representa a quantidade de tempo que separa dois veículos. Normalmente é medida a partir de pontos em comum dos dois veículos, como por exemplo o para-choque frontal. É dada pela distância que separa duas curvas no eixo X .

Além das informações microscópicas do tráfego, é possível também extrair informações macroscópicas do diagrama espaço-tempo. A densidade do tráfego (k), é calculada com base no espaçamento, a partir da quantidade de veículos (n) que se encontra em um determinado trecho de tamanho fixo (L). Portanto, a densidade é obtida pela equação (5.1).

$$k = \frac{n}{L} \quad (5.1)$$

Outra informação macroscópica que pode ser extraída do diagrama espaço-tempo é o fluxo de veículos (q). O fluxo é calculado com base no headway entre os veículos, a partir da divisão do número de veículos (n) pelo intervalo de tempo (Δt), como na equação (5.2).

$$q = \frac{n}{\Delta t} \quad (5.2)$$

Com essas duas informações e com o auxílio da lei fundamental do tráfego, é possível calcular a velocidade do fluxo com a equação (5.3).

$$q = k \cdot u \quad (5.3)$$

A velocidade média pode ser calculada dividindo-se a variação do espaço percorrido, $d_b - d_a$, pelo tempo decorrido $t_b - t_a$. A velocidade instantânea de um veículo também pode ser calculada e é dada pela declividade da curva em algum instante de tempo.

Portanto, somente com as informações presentes no diagrama espaço-tempo, vários dos parâmetros usados em planejamento de sistemas de transportes foram extraídos. Então, é necessário que o algoritmo consiga fornecer os dados necessários para que se possa construir essa ferramenta importante.

5.2 MÉTODO PROPOSTO

O objetivo do método proposto nesse capítulo é extrair as informações do fluxo de tráfego a partir dos dados brutos gerados nas fases anteriores desse trabalho. O fluxograma do método pode ser visto na Figura 47.

Figura 47 – Fluxograma do método de extração de informações

Os dados brutos gerados pelo algoritmo vêm na forma de pares de coordenadas (x, y) do centroide de cada veículo, acompanhados do número do *frame* no qual os centroides foram detectados e de um identificador numérico único, alocado no primeiro momento no qual o veículo entra na ROI e consolidado quando esse mesmo veículo sai da ROI. Esses dados são então interpretados no módulo de interpretação e convertidos para unidades do mundo real, como velocidades instantâneas e médias. Depois de sair do interpretador, os dados são enviados para o módulo responsável pelo salvamento em arquivos e pela exibição do diagrama. Os dados são então salvos em arquivos de relatório, tanto detalhado como resumido, e o diagrama espaço-tempo é atualizado com as trajetórias já completadas.

Para os arquivos de relatório, foi escolhido salvar os dados em padrão CSV (*Comma Separated Value*). Esses arquivos possuem a vantagem de ser facilmente importados para ferramentas de análise de dados, como o Excel, da Microsoft, ou o SPSS, da IBM, e são facilmente escritos em Python. Não é necessário usar nenhuma biblioteca adicional, diminuindo o *overhead* do algoritmo, bastando separar os valores escritos com vírgulas.

Os dados salvos pelo algoritmo são todos microscópicos. Portanto, são dados sobre velocidade média, velocidade instantânea, posição no eixo X (importante para descobrir sobre mudanças de faixa), posição no eixo Y (sentido do tráfego) e *frame* relacionado a cada posição.

Para gerar todas as informações dos relatórios e o diagrama espaço-tempo, foram utilizados todos os vídeos do item 2.5. As características dos vídeos podem ser vistas na Tabela 1.

5.2.2 RELATÓRIO AGREGADO

Durante o *tracking* dos veículos, uma quantidade grande de informações é gerada. Caso se queira fazer uma verificação rápida sobre, por exemplo, qual a velocidade média de um dos veículos, olhar um relatório completo em busca da informação necessária toma muito tempo.

Por isso, para facilitar a visualização dos dados, é gerado um arquivo resumido, com seu padrão mostrado na Figura 48.

```

1 ##### Dados Veiculares Resumidos #####
2 Padrão do arquivo:
3 ID, Velocidade Média, Frame Final, Faixa Final
4 #####
5
6 34,103,4324,2
7 35,119,4443,2
8 36,139,4474,2
9 37,101,4567,2
10 38,108,4757,1
11 39,112,4963,1
12 40,103,5128,1
13 41,114,5163,2
14 42,101,5411,1
15 43,83,6024,1
16 44,106,6128,2

```

Figura 48 – Exemplo de arquivo de saída, com informações resumidas

É possível ver as informações de 10 veículos, do 34 ao 44, com suas velocidades médias, os *frames* finais com que passaram pela ROI e em qual faixa de tráfego terminaram sua trajetória. Todas essas informações ficam salvas num arquivo nomeado `nome-do-video_veiculos_resumo.csv`.

Neste arquivo, são salvos dados sobre o número único de identificação do veículo, sobre sua velocidade média, sobre o *frame* no qual o veículo entra e sai da região de interesse e sobre qual a faixa de tráfego que o veículo encerrou sua trajetória. As informações são salvas sempre que os veículos encerram sua trajetória pela ROI. Após o processamento de todo o vídeo, para saber a quantidade de veículos, basta olhar a quantidade de linhas do arquivo que contém informações sobre veículos.

5.2.4 DIAGRAMA ESPAÇO-TEMPO

Com todas as informações salvas no arquivo detalhado, é possível construir o diagrama espaço-tempo usando qualquer ferramenta de plotagem de gráficos. Porém, em Python também é possível plotar gráficos enquanto o processamento do vídeo é feito. Desse modo, sempre que um veículo encerra sua trajetória, isto é, atinge a linha de chegada no topo da área de interesse, seus dados são adicionados em um gráfico atualizado dinamicamente. Por questões de desempenho, optou-se por adicionar os dados somente no fim da trajetória, e não durante.

A biblioteca usada para a criação do diagrama em Python foi a `matplotlib`, disponível no site <https://matplotlib.org/>. A documentação com as funções e as capacidades dessa biblioteca estão disponíveis no próprio site principal, com detalhes sobre as diversas maneiras de se plotar dados.

5.3 RESULTADOS

O diagrama Espaço-Tempo pode ser gerado ao mesmo tempo em que o algoritmo processa o vídeo, através de funções de bibliotecas do Python. Esse diagrama, junto com informações sobre a velocidade e a contagem de veículos dos vídeos, formam o resultado final do método proposto.

Tanto a velocidade quanto a contagem dos veículos foram usadas para verificar a integridade das informações geradas pelo algoritmo. Nos dois casos, os valores obtidos pelo sistema proposto foram comparados à valores obtidos de forma manual ou com o auxílio de dispositivos eletrônicos.

5.3.1 DIAGRAMA ESPAÇO-TEMPO

Com os dados gerados pelo algoritmo, mais especificamente as posições no eixo Y e os *frames* de cada uma das posições, é possível gerar o diagrama, como visto na Figura 50.

Figura 50 – *Frame* original, com ROI destacada, e diagrama espaço-tempo gerado

Observa-se que foram detectadas e computadas as trajetórias de 9 veículos no diagrama. As posições plotadas foram geradas a partir do centroide de cada veículo. Caso alguma *feature* seja perdida durante o *tracking*, ocorre o fenômeno apresentado nos veículos 1, 2 e 4 – os veículos voltam posições no *tracking* com o avanço dos *frames*. Com menos *features*, o centroide é recalculado, fazendo com que sua posição sofra uma variação brusca. Também é possível observar, nos veículos 4-5 e 6-7, que o algoritmo detectou, em um primeiro momento, somente um veículo. Com o passar dos *frames*, o algoritmo reagrupou as *features* e determinou que, na verdade, eram dois veículos distintos, dividindo sua trajetória em duas. O último veículo detectado, o 9, é o veículo que, no *frame* original, está atravessando a fronteira da ROI.

Como exemplo de uso do diagrama, pode-se calcular o *headway* médio entre dois veículos. Entre o veículo 8 e 9, é possível notar uma diferença relativamente constante de 50 *frames*. O vídeo usado nesse exemplo foi gravado a 25 *frames* por segundo. Portanto, durante o tempo no qual os veículos percorreram a ROI, o *headway* médio entre eles foi de 2 segundos.

5.3.2 VELOCIDADE

Um dos vídeos da rodovia SP-310 – vídeo #1, na Tabela 1 – foi gravado juntamente com captura de velocidade usando um dispositivo do tipo radar-pistola. Por esse motivo, somente nesse vídeo as velocidades foram analisadas e validadas.

No total, as velocidades de 126 veículos foram medidas durante os 18 minutos do vídeo. Desses 126 veículos, o algoritmo foi capaz de detectar e calcular a velocidade de 125, o que significa uma taxa de detecção de 99,2%. Com relação aos erros, o erro médio das velocidades foi de 9,9%. O maior erro foi de 32%, na medida da velocidade de um caminhão. O algoritmo calculou sua velocidade em 99 km/h, ao passo que o radar-pistola mediu em 75 km/h. Os menores erros calculados foram de 0%, acontecendo em seis ocasiões, tanto em veículos leves como em veículos pesados.

Os primeiros 23 veículos que tiveram suas velocidades medidas e calculadas pelo algoritmo são mostrados na Tabela 4. É possível notar que, para a maior parte dos erros mais altos, os veículos eram de porte pesado. Também é possível notar o único veículo não detectado pelo algoritmo, de número 19.

Tabela 4 – Velocidades medidas com radar-pistola e geradas pelo algoritmo

ID	Veículo	Faixa	Frame	Velocidade [km/h]		Diferença (%)
				Pistola	Algoritmo	
1	Del Rey Branco	1	450	76	78	2,6%
2	Mondeo Preto	2	910	117	120	2,6%
3	Vectra Preto	1	1304	90	86	4,4%
4	Caminhão MB 1113 Verde Claro	1	1480	92	116	26,1%
5	Fit Preto	2	2196	109	114	4,6%
6	Caminhão MB 1113 Laranja	1	2519	91	111	22,0%
7	Golf Vinho	2	2713	85	95	11,8%
8	Caminhão MB 1113 Azul Claro	1	2760	80	96	20,0%
9	Palio Preto	1	2880	86	66	23,3%
10	Fiorino Azul Escuro	1	2970	81	83	2,5%
11	S-10 Preto	2	3300	97	106	9,3%
12	Caminhão MB Azul	2	3460	102	120	17,6%
13	Caminhão MB 1113 Azul Claro	1	3580	78	91	16,7%
14	Strada Preto	2	3815	91	98	7,7%
15	Gol Branco	2	3845	94	105	11,7%
16	Caminhão VW Branco	2	3994	97	115	18,6%
17	Caminhão MB Branco	1	4410	89	119	33,7%
18	Ipanema Preto	2	4535	93	101	8,6%
19	Caminhão Vermelho	1	4535	93	-	-
20	Eco Esport Preto	1	4742	108	108	0,0%
21	C3 Preto	1	4960	90	112	24,4%
22	Caminhão MB Branco	1	5071	88	103	17,0%
23	Caminhão GMC Branco	1	5366	92	101	9,8%

Como primeiro passo para a análise, é necessário fazer um teste de aderência entre as distribuições das amostras capturadas pelo radar-pistola e calculadas pelo algoritmo. Para isso, aplica-se um teste Qui-Quadrado de Aderência. As distribuições das duas amostras podem ser vistas na Figura 51.

Figura 51 – Distribuições das duas amostras de velocidade: do radar-pistola e do algoritmo

Para começar o teste, devem ser definidas duas hipóteses: a hipótese nula, que afirma que não existe diferença entre as distribuições do radar-pistola e do algoritmo; e a hipótese alternativa, que afirma que existe diferença entre as distribuições.

H_0 = não existe diferença entre as distribuições das amostras medidas pelo radar-pistola e calculadas pelo algoritmo;

H_1 = existe diferença entre as distribuições.

Calculando-se o valor χ^2 para o teste, obtêm-se $\chi^2 = 0,445$. Com esse valor, deve-se procurar o valor crítico na tabela do teste, conferindo o grau de liberdade e o nível de confiança. Para este trabalho, foi usado grau de liberdade de 12 e analisados níveis de confiança de 95% e de 99%. Para esses valores, na tabela, o valor crítico para 95% é de $\chi^2_{crítico} = 5,226$. Para 99%, o valor crítico é de $\chi^2_{crítico} = 3,571$. Portanto, o teste passa nos dois níveis de confiança e é possível concluir que não é possível rejeitar a hipótese nula. Conclui-se então que os valores calculados pelo algoritmo são representativos dos valores medidos pelo radar-pistola.

Para visualização das informações, foi feito um histograma categorizado em taxas de erro, divididos em intervalos de 2 em 2%. Dessa maneira, é possível ver em qual faixa se concentram os erros calculados pelo algoritmo. O histograma pode ser visto na Figura 52.

Figura 52 – Distribuição dos erros relativos ocorridos durante as detecções

É possível perceber que 60% dos erros se concentram em intervalos menores do que 10%, ou seja, para mais da metade das velocidades calculadas, o erro entre a velocidade gerada pelo algoritmo e a velocidade medida pela radar-pistola não passou de 10%.

Como o agrupamento de *features* de veículos pesados é sempre feito com grande dificuldade, é também mostrado um histograma com o erro das velocidades sem os veículos pesados, para comparação. O algoritmo proposto tem dificuldades quando veículos ocupam grande parte da região de interesse ou quando não caibam na região. Quando isso acontece, são

detectados e contados como múltiplos veículos menores. A distribuição dos erros sem veículos pesados pode ser vista na Figura 53.

Figura 53 – Distribuição dos erros ocorridos durante as detecções, sem veículos pesados

Sem os veículos pesados, é possível notar que os erros se concentram nas categorias inferiores do histograma. Quase 80% dos erros calculados acontecem abaixo de 10%, significando que 80% dos veículos tiveram suas velocidades calculadas pelo algoritmo com menos de 10% de erro em comparação à medição do radar-pistola.

É possível comparar os resultados deste trabalho, na medição da velocidade, com os resultados de Cunha (2013), pois trata-se da mesma fonte de dados. No trabalho de Cunha (2013), as técnicas empregadas foram a *Scoreboard*, para modelagem do *background* e segmentação dos veículos, seguida de um algoritmo de busca em vizinhança para o *tracking* dos veículos. Neste trabalho, o algoritmo para modelagem do *background* e segmentação foi o MOG2, seguido de Shi-Tomasi e Lucas-Kanade para detecção e *tracking* de *features*. Cunha (2013) obteve 95,2% de sucesso na detecção dos veículos, com erro relativo no cálculo das velocidades em 7,3%. Este trabalho obteve detecção de 99,2% dos veículos, com erro relativo médio de 9,9% no cálculo das velocidades.

5.3.3 CONTAGEM

Para verificar o desempenho da contagem do algoritmo proposto, foram feitas contagens manuais em todos os vídeos utilizados no teste, para comparação. Os resultados são apresentados de acordo com o ângulo de captura da câmera, divididos nas três categorias

distintas já introduzidas anteriormente. Junto com o histograma da contagem é exibida uma reta indicando o erro da contagem automática em relação à contagem manual.

- **PERSPECTIVA: BR-116**

Com o intuito de melhorar a visualização dos dados, os resultados do vídeo da rodovia BR-116 – vídeo #1 – são mostrados em duas partes. Na Figura 54, é possível ver o resultado da contagem manual e automática dos primeiros 27 minutos de gravação. Na Figura 55, são apresentados os resultados dos últimos 28 minutos, totalizando os 55 minutos de gravação.

Figura 54 – Primeiros 27 minutos do vídeo da BR-116 e suas contagens

Figura 55 – Últimos 28 minutos do vídeo da BR-116 e suas contagens

O erro médio da contagem automática foi de 12,3%. O melhor desempenho se deu na faixa do minuto 20, com a passagem de 25 veículos e nenhum erro de contagem. Outros intervalos também não apresentaram erro na contagem. O pior desempenho ocorreu no minuto 37, no qual o algoritmo contou 20 veículos na passagem de 13, errando em pouco mais de 50%.

Sem considerar os veículos pesados, o algoritmo tem um desempenho um pouco melhor. Os resultados também serão divididos em duas partes, na Figura 56 e Figura 57, para melhorar a visualização, seguindo o mesmo padrão anterior.

Figura 56 – Primeiros 27 minutos da BR-116 (#1) e suas contagens, sem veículos pesados

Figura 57 – Últimos 28 minutos da BR-116 (#1) e suas contagens, sem veículos pesados

É possível notar que, com a eliminação dos veículos pesados, não existem mais erros acima de 25%. O erro médio baixou para 10,3%, com maior valor na faixa dos 54 minutos. Também houve um aumento em faixas de tempo sem erros de detecção, com 13 instâncias com 0%.

A pouca variação do resultado das contagens com e sem veículos pesados é explicada pela posição da câmera. No vídeo da BR-116, a câmera está posicionada em cima de um poste com o intuito de filmar grandes porções da rodovia. Esse fato possibilita que a área de interesse selecionada seja grande o suficiente para que veículos pesados consigam ser capturados inteiramente.

- **PERSPECTIVA: SP-310**

Para a rodovia SP-310 – vídeo #2 – o resultado da comparação da contagem manual e automática pode ser visto na Figura 58.

Figura 58 – Comparação das contagens no vídeo #2, da rodovia SP-310

O erro médio da contagem automática foi de 17%. O melhor desempenho se deu na faixa do minuto 15, com a passagem de 20 veículos e nenhum erro de contagem. Seis intervalos não apresentaram nenhum erro na contagem. O pior desempenho ocorreu no minuto final, 18, no qual o algoritmo contou 7 veículos a mais que os 12 que realmente passaram, errando em quase 60%.

Retirando os veículos pesados, o resultado melhora, como pode ser visto na Figura 59.

Contagem da SP-310 (#2), sem Veículos Pesados

Figura 59 – Comparação das contagens do vídeo #2, da SP-310, sem veículos pesados

Sem veículos pesados, o algoritmo proposto errou, em média 5%. Em 11 dos 18 intervalos, o erro na contagem foi de 0%. O maior erro aconteceu no minuto 10, com 25%, no qual o algoritmo contou 3 veículos onde passaram 4.

A diferença dos erros de contagem diminui quando veículos pesados são retirados, com melhora significativa no vídeo #2, do SP-310. Portanto, o ângulo da câmera mais reduzido em comparação ao vídeo da BR-116 não permite que veículos pesados entrem completamente na área de interesse, causando contagens erradas e superiores à real.

- **FRONTAL: Av. P. Francisco S. Colturato – Contra Fluxo**

O resultado da contagem do vídeo frontal, no sentido do contra fluxo, gravado na Avenida Padre Francisco Sales Colturato – vídeo #3 – pode ser visto na Figura 60.

Contagem da Av. Colturato - Contra Fluxo (#3)

Figura 60 – Comparação das contagens do vídeo #3, Av. Colturato

O erro médio da contagem automática foi de 9,4%. O melhor desempenho se deu na faixa do minuto 11, com a passagem de 13 veículos e nenhum erro de contagem. Seis intervalos não apresentaram nenhum erro na contagem. O pior desempenho ocorreu no minuto 4, no qual o algoritmo contou 17 veículos, quando somente 12 passaram, errando em quase 42%.

Na Figura 61 é possível ver o resultado das contagens sem os veículos pesados.

Figura 61 – Comparação das contagens do vídeo #3, Av. Colturato, sem veículos pesados

Sem veículos pesados, o erro médio caiu para 6%. Em 7 dos 13 intervalos de tempo, o erro medido foi de 0%, com nenhum veículo contado erroneamente. O maior erro aconteceu no minuto final, 13, com 17,6%, no qual o algoritmo contou 14 dos 17 veículos que passaram nessa faixa de tempo.

- **FRONTAL: Av. P. Francisco S. Colturato – Fluxo**

O resultado das contagens para o vídeo gravado no sentido do fluxo do tráfego na Avenida Padre Francisco Sales Colturato – vídeo #5 – pode ser visto na Figura 62.

Figura 62 – Comparação das contagens do vídeo #5, da Av. Colturato

O erro médio da contagem automática foi de 11%. O melhor desempenho se deu na faixa do minuto 6, com a passagem de 10 veículos e nenhum erro de contagem. Três intervalos não apresentaram nenhum erro na contagem. O pior desempenho ocorreu no minuto inicial e no minuto final, com 25% de taxa de erro na contagem.

Na Figura 63 é possível ver o resultado das contagens sem os veículos pesados.

Figura 63 – Comparação das contagens do vídeo #5, Av. Colturato, sem veículos pesados

Sem veículos pesados, o erro médio caiu para 9%. Em 4 intervalos de tempo, o erro medido foi de 0%, com nenhum veículo contado erroneamente. Os piores desempenhos ainda se mantiveram nos minutos iniciais e finais, com 25% de erro.

Os erros médios, tanto com veículos pesados quanto sem veículos pesados, foram menores no vídeo contra o fluxo de tráfego. Isso indica que o algoritmo pode se comportar melhor, com relação à contagem, quando os veículos se aproximam da câmera. Nessa situação, o espaçamento lateral em pixel entre os veículos é maior, auxiliando no agrupamento das *features*.

- **FRONTAL: SP-310 – meio dia**

Os resultados das contagens da rodovia SP-310 – vídeo #4 – pode ser visto na Figura 64.

Figura 64 – Comparação das contagens no vídeo #4, SP-310

O erro médio da contagem automática foi de 17%. O melhor desempenho se deu na faixa do minuto 5, com erro igual a 8%. Em nenhum dos intervalos o erro observado foi nulo. O pior desempenho ocorreu no minuto inicial, com 20 veículos contados na passagem de somente 15, resultando em 33% de erro.

O resultado das contagens sem veículos pesados pode ser observado na Figura 65.

Figura 65 – Comparação das contagens no vídeo #4, SP-310, sem veículos pesados

Sem considerar veículos pesados, o erro médio caiu para 3%. Em 5 dos intervalos de tempo, o erro medido foi de 0%, com nenhum veículo contado erroneamente. O pior desempenho se deu na faixa de tempo de 3 minutos, com 9 veículos contados, de 8, e erro de 13%.

O algoritmo, como nos vídeos anteriores, também teve dificuldades em agrupar veículos pesados, por causa da diferença na dimensão dos veículos e no modo como as *features* são agrupadas.

- **FRONTAL: SP-310 – entardecer**

As contagens do vídeo da rodovia SP-310 – vídeo #6 – podem ser vistas na Figura 66.

Figura 66 – Comparação das contagens do vídeo #6, da SP-310

O erro médio da contagem automática foi de 16,3%. O melhor desempenho se deu na faixa do minuto 7, sem erro na contagem dos veículos. O pior desempenho ocorreu no minuto 6, com 34 veículos contados na passagem de 24, resultando em 42% de erro.

Os resultados, sem considerar veículos pesados, podem ser vistos na Figura 67.

Figura 67 – Comparação das contagens no vídeo #6, SP-310, sem veículos pesados

Sem considerar veículos pesados, o erro médio caiu para 4,7%. Em 3 dos intervalos de tempo, o erro medido foi de 0%, com nenhum veículo contado erroneamente. O pior desempenho se deu na faixa de tempo de 4 minutos, com 10 veículos contados, de 9, e erro de 11%.

- **LATERAL: SP-310**

A comparação das contagens do último vídeo da SP-310 – (vídeo #7) – com visão lateral da via, pode ser vista na Figura 68.

Figura 68 – Comparação das contagens no vídeo #7 SP-310

O erro médio da contagem automática foi de 6,6%. O melhor desempenho se deu na faixa do minuto 3, sem erro na contagem dos 29 veículos que passam na via. O pior desempenho ocorreu no minuto final, com 33% de erro, contando 2 veículos na passagem de 3.

Na Figura 69 é possível ver o resultado das contagens sem os veículos pesados.

Figura 69 – Comparação das contagens no vídeo #7, SP-310, sem veículos pesados

Contrariando os demais resultados sem veículos pesados, o erro médio subiu para 13,4%. O melhor desempenho foi observado na faixa dos 9 minutos, com nenhum veículo contado erroneamente. O pior desempenho se manteve no minuto final, com 33% de erro.

Vídeos com perspectiva lateral são os mais propensos a apresentar oclusão, pois uma faixa de tráfego oculta a outra. Nesse vídeo, o desempenho melhor do algoritmo com os veículos pesados se explica justamente por esse fato. Quando ocorre oclusão, o algoritmo tende a contar uma quantidade menor de veículos. Porém, quando veículos pesados estão presentes, o algoritmo tende a contar veículos a mais. Portanto, um erro anula o outro, gerando resultados aparentemente melhores.

5.4 CONCLUSÕES

Neste capítulo, foram relatados os dados e informações que podem ser gerados pelo sistema proposto. Foram gerados dados em dois arquivos diferentes: um arquivo resumido, com informações simples como velocidade média, faixa de tráfego e *frame* final dos veículos que percorrem a área de interesse; e um arquivo completo, com informações sobre cada velocidade, posição e faixa em cada *frame* do vídeo. Também foi gerado o diagrama espaço-tempo, que habilita ao administrador de um sistema de transportes a extração de informações importantes para o gerenciamento.

Foi feita a comparação dos dados gerados pelo sistema com os dados medidos por dispositivos mais tradicionais. As contagens foram divididas em três categorias, baseadas no ângulo de filmagem dos vídeos. Um resumo dos resultados pode ser visto na Tabela 5 – Resumo dos resultados das contagens.

Tabela 5 – Resumo dos resultados das contagens

Visão	Vídeo	Erro	Erro Sem Veículos Pesados
Perspectiva	(#1) BR-116	12,3%	10,3%
Perspectiva	(#2) SP-310	17,0%	5,0%
Frontal	(#3) Av. Colturato (contra fluxo)	9,4%	6,0%
Frontal	(#5) Av. Colturato (fluxo)	11,0%	9,0%
Frontal	(#4) SP-310 (meio dia)	17,0%	3,0%
Frontal	(#6) SP-310 (tarde)	16,3%	4,7%
Lateral	(#7) SP-310	6,6%	13,4%

Para os vídeos em perspectiva, o erro foi de 12,3% nas contagens do vídeo #1, da BR-116, diminuindo para 10,3% quando excluídos os veículos pesados, e 17% para o vídeo #2, da SP-310, diminuindo para 5% na contagem sem veículos pesados. Essa variação do erro pode ser explicada pela mudança da posição da câmera. No vídeo #1, da BR-116, a câmera foi posicionada bem mais alta que a câmera do vídeo #2, da SP-310. Esse detalhe possibilita que a

região de interesse possa ser maior, permitindo que o algoritmo rastreie e agrupe as *features* de maneira mais adequada e constante.

Nos vídeos gravados com ângulo frontal ao tráfego, os resultados foram de 9,4%, no contra fluxo e 11% no fluxo, para os vídeos gravados na Av. Padre Francisco Sales Colturato. O erro menor no contra fluxo indica que câmeras posicionadas contra o fluxo de tráfego contribuem para um desempenho melhor para o algoritmo proposto, pois a consolidação dos dados se dá quando os veículos estão mais pertos da câmera e, conseqüentemente, maiores. O espaçamento lateral em pixels dos veículos também aumenta, facilitando a lógica de agrupamento de *features* utilizada. Ainda nos vídeos da Av. Colturato, sem considerar veículos pesados, os erros resultantes foram de 6% no contra fluxo e 9% a favor do fluxo. A variação dos dois vídeos é parecida e consistente com o comportamento dos resultados com veículos pesados.

Ainda nas situações frontais ao tráfego, os vídeos da SP-310 apresentaram erro de 17% no vídeo #4, gravado ao meio dia, e 16,3% no vídeo #6, gravado ao entardecer. Sem considerar veículos pesados, os erros foram para 3% no vídeo #4 e 4,7% no vídeo #6. É possível observar que os erros das gravações da rodovia SP-310 são próximos, indicando uma boa consistência no resultado do algoritmo, tanto com veículos pesados e sem veículos pesados. Isso indica que a presença de sombras mais alongadas no vídeo ao entardecer não influenciou o resultado das contagens. O efeito das sombras acontece sempre na fase de remoção do *background*, pois afeta o modelo de *background* gerado pelo algoritmo. No algoritmo escolhido para essa tarefa, o MOG2, há um parâmetro que, quando habilitado, remove as sombras dos objetos em movimento, quando detectadas. Os resultados dos testes nos vídeos da SP-310 confirmam que o efeito das sombras foi mitigado pelo MOG2. Também é possível notar que os erros nos dois vídeos da SP-310, com ângulo frontal, são similares ao da gravação feita em perspectiva, na mesma rodovia. Isso indica que o ângulo da câmera, quando posicionada em cima de viadutos, não tem influência no resultado.

Finalmente, no vídeo lateral, o erro médio foi de 6,6% e, sem veículos pesados, de 13,4%. O vídeo lateral é o que mais favorece situações de oclusão parcial e total de veículos, causando contagens de um único veículo em *features* que deveriam indicar dois ou mais. Além disso, em vídeos laterais, as dimensões dos veículos pesados não se encaixam no ângulo de visão de câmeras normais, sendo detectados como múltiplos veículos. No caso excepcional do vídeo usado neste trabalho, as ocorrências de oclusão se anulam com as ocorrências de veículos maiores que a região de interesse. Por esse motivo, a contagem completa apresenta um índice

melhor que a contagem sem veículos pesados. Um exemplo em que ambos os casos acontecem ao mesmo tempo pode ser visto na Figura 70.

Figura 70 – Dimensão maior que a ROI e oclusão de veículo

Na medição da velocidade, foram registrados os valores de 126 veículos com um dispositivo do tipo radar-pistola. O algoritmo mediu e calculou as velocidades de 125 desses veículos, resultando em 99,2% dos veículos detectados. O erro médio dos cálculos foi de 9,9%, com seis casos de erro nulo. Na análise da distribuição dos erros relativos, 60% das velocidades geradas pelo algoritmo ficaram abaixo de 10% de erro. Em um teste sem veículos pesados, foi possível observar que quase 80% dos erros se concentrou abaixo de 10%.

Para todos os erros gerados pelo algoritmo, a calibração dos parâmetros de cada método utilizado em cada um dos passos tem o potencial de melhorar o resultado, diminuindo o erro tanto nas contagens quanto no cálculo das velocidades.

6 CONCLUSÕES E SUGESTÕES

No campo da Engenharia de Transportes, dados ajudam no planejamento, no desenvolvimento e na operação de projetos, refletindo em sistemas de transportes mais eficientes economicamente e energeticamente, impactando de modo positivo a vida de milhões de pessoas.

O objetivo deste trabalho foi propor um método computacional automático capaz de identificar, a partir de imagens de vídeo, o comportamento do tráfego veicular rodoviário. Nesse quesito, este trabalho cumpriu seu objetivo e conseguiu extrair informações importantes para um operador de um sistema de transportes.

6.1 CONCLUSÕES

Para se atingir o objetivo principal deste trabalho, quatro objetivos específicos foram propostos:

- (1) Detectar veículos nas imagens fornecidas pelas câmeras, analisando comparativamente o desempenho das principais estratégias de detecção para fluxo ininterrupto;
- (2) Rastrear os veículos durante toda sua trajetória dentro da área de interesse na imagem;
- (3) Extrair o comportamento e informações do tráfego, como velocidade média de cada veículo, faixa de rodagem, contagem de veículos e o diagrama espaço-tempo;
- (4) Validar as informações geradas pelo algoritmo, comparando-as com valores obtidos em campo.

Antes de começar a codificar os algoritmos envolvidos neste trabalho, foi necessária uma intensa busca de referências bibliográficas, para aumentar o entendimento sobre quais informações são importantes para um operador de sistemas de transportes. Além disso, a quantidade de técnicas disponíveis em várias linguagens de programação diferentes, junto com a grande disponibilidade de bibliotecas com funções para o processamento de imagens, faz o trabalho de decisão parecer impossível. Portanto, antes do uso e da escolha de cada um dos algoritmos, o entendimento sobre seu funcionamento e sobre suas qualidades e defeitos é imprescindível.

Para o primeiro objetivo específico, foram testados algoritmos que estão disponíveis no OpenCV, em Python. O algoritmo de segmentação que mais se destacou foi o MOG2, com taxa de exatidão acima de 98%, taxa de precisão com valores acima de 95% e tempo de processamento de 64 *frames* por segundo. Embora o desempenho do segundo colocado, o MOG, nas taxas de exatidão e de precisão tenham sido equivalentes, o tempo de processamento do MOG2 foi mais de três vezes melhor que o MOG, que apresentou taxas de somente 18 FPS. É importante que um algoritmo que se disponha a segmentar veículos consiga executar essa primeira fase do processamento da maneira mais completa possível. Todas as outras fases do processamento se apoiam nesse primeiro passo. De fato, o algoritmo MOG2 conseguiu cumprir sua função e segmentou os pixels em *foreground* e *background*.

No segundo objetivo específico, rastreamento dos veículos, testou-se o algoritmo Shi-Tomasi de detecção de *features*, junto com o algoritmo de fluxo ótico de Lucas-Kanade. Esses métodos, quando usados em conjunto com o MOG2 para remoção de *background* e com um algoritmo de agrupamento baseado em distâncias e velocidades, conseguiu rastrear as *features* dos veículos por toda a área de interesse. Essa fase do algoritmo é crítica para a obtenção das informações veiculares, pois é a fase que gera as informações dos pontos de trajetórias do centroide de cada veículo. A limitação do método está no agrupamento das *features*, como mencionado na literatura deste trabalho. Para minimizar o efeito do uso somente da distância entre as *features*, foi implementado o uso das velocidades nas condições de agrupamento. Embora ainda apresente erros, as detecções do algoritmo proposto se mantiveram, na média entre todos os vídeos, em 88,2% de acerto. Portanto, para segmentar, rastrear e agrupar as *features* dos veículos, o uso combinado dos algoritmos propostos teve êxito.

Os terceiro e quarto objetivos específicos foram realizados em conjunto. Nessa fase, foram extraídas informações relacionadas ao tráfego veicular a partir de dados gerados pelo próprio algoritmo. Os dados foram salvos em arquivos diferentes, com versões resumidas (contendo o identificador único dos veículos, as velocidades médias de sua trajetória, o frame final do movimento e a faixa de encerramento da trajetória) e completas (contendo todos os pares de coordenadas da trajetória do centroide do veículo, os *frames* no qual foram registrados, as velocidades em cada frame e a faixa de tráfego de cada ponto do centroide). Também foi gerada uma ferramenta importante para operadores de sistemas, o diagrama espaço-tempo. Nos dois casos, informações com medidas no mundo real, como velocidade e contagens de veículos, foram extraídas a partir de dados virtuais, como quantidade de pixels por frame.

Para validar os dados extraídos, métodos tradicionais no campo da Engenharia de Transportes foram usados para aquisição dos mesmos parâmetros. A contagem foi feita de modo manual em cada um dos vídeos e a velocidade foi medida com o uso de radar-pistola. Foi feito um teste Qui-Quadrado para avaliar a aderência das amostras das velocidades do radar-pistola com os valores gerados pelo algoritmo. O teste mostrou boa aderência, não rejeitando a hipótese nula, que afirmava que as distribuições das duas fontes de dados não tinham diferenças. No cálculo da velocidade, o erro médio para todos os veículos foi de 9,9%, com pouco mais de 50% dos erros abaixo de 8%. Somente um dos veículos não teve sua velocidade medida por causa de falhas na detecção, dentre os 126 veículos totais. Nas contagens, o algoritmo não teve taxas de erro superiores a 17%, com média de erro de detecção entre todos os vídeos de 12,8%. A maioria dos erros se deu na contagem de veículos pesados pois, removendo-os dos resultados, os erros caem para 7%, em média. Portanto, conclui-se que, embora o conjunto de algoritmos implementados tenha tido algumas dificuldades, ainda assim apresentou desempenho satisfatório na detecção de veículos e na medição de velocidades.

6.2 SUGESTÕES PARA TRABALHOS FUTUROS

A maioria dos problemas de detecção ocorreram em veículos pesados. A região de interesse do processamento tem que ser capaz de conter todas as *features* do veículo pesado, mas também ter espaço suficiente para que todas essas *features* percorram uma trajetória conjuntamente. No caso dos vídeos testados, essa condição nem sempre pode ser observada, pois o ângulo da câmera e a resolução não permitiam que se estendesse a ROI.

Portanto, uma sugestão para trabalhos futuros seria usar uma distância de agrupamento que seja ajustada dinamicamente, durante o processamento do vídeo, para mitigar os problemas encontrados e melhorar a detecção na categoria dos veículos pesados.

No quesito dos ângulos das câmeras, sugere-se que sejam testados mais vídeos que combinem gravações no contra fluxo com gravações a favor fluxo do tráfego, para investigar se alguma das duas situações é favorecida pelo algoritmo. Espera-se que, para vias que não possuam declives ou aclives, o desempenho seja melhor em vídeos gravados a favor do fluxo, pois as *features* dos veículos são detectadas na parte mais próxima à câmera, onde mais detalhes estão disponíveis.

Outra sugestão seria testar mais algoritmos de *tracking* disponíveis no OpenCV. Como visto neste trabalho, a biblioteca oferece várias funções com esse objetivo. Um teste com todas,

verificando qual conseguiria um desempenho melhor nas condições usuais de vídeos de tráfego veicular em rodovias, seria importante.

Para que o algoritmo possa ser utilizado em tempo real, modificações de desempenho também são recomendadas. Pode-se adotar uma estratégia para detectar novas *features* em uma porcentagem da taxa de FPS. Por exemplo, para detectar *features* em 50% dos *frames*, basta pular *frames* pares ou ímpares. Essas alterações diminuem o grande impacto da atividade de detecção no tempo de processamento e não influenciam na detecção dos veículos.

Por último, para todas as funções utilizadas do Python, seria interessante algum tipo de programa que ajude no ajuste dos vários parâmetros possíveis. Neste trabalho, os valores padrão foram usados em boa parte das funções, mas testes com outras combinações de parâmetros seriam importantes, pois podem ajudar no resultado da detecção e no rastreamento dos veículos. O trabalho de ajuste dos parâmetros é complexo e demanda muito tempo, pois deve ser realizado vídeo a vídeo.

REFERÊNCIAS BIBLIOGRÁFICAS

AAMODT, Agnar; NYGÅRD, Mads. Different roles and mutual dependencies of data, information, and knowledge—an AI perspective on their integration. **Data & Knowledge Engineering**, v. 16, n. 3, p. 191-222, 1995.

ALEX, D. Stalin; WAHI, Amitabh. BSFD: Background Subtraction Frame Difference Algorithm for Moving Object Detection and Extraction. **Journal of Theoretical & Applied Information Technology**, v. 60, n. 3, 2014.

ANWAR, Afian; ZENG, Wei; ARISONA, Stefan. Time-Space Diagram Revisited. **Transportation Research Record: Journal of the Transportation Research Board**, n. 2442, p. 1-7, 2014.

ARAÚJO, Juliana Jerônimo. **Estudo do impacto de veículos pesados sobre a infra-estrutura rodoviária através de simulação microscópica de tráfego**. 2007. Tese de Doutorado. Universidade de São Paulo.

ARMSTRONG, M. G. *et al.* High Frame-rate television. **SMPTE motion imaging journal**, v. 118, n. 7, p. 54-59, 2009.

BAY, Herbert; TUYTELAARS, Tinne; VAN GOOL, Luc. Surf: Speeded up robust features. Em: **European conference on computer vision**. Springer Berlin Heidelberg, 2006. p. 404-417.

BENEZETH, Yannick *et al.* Review and evaluation of commonly-implemented background subtraction algorithms. Em: **Pattern Recognition**, 2008. ICPR 2008. 19th International Conference on. IEEE, 2008. p. 1-4.

BEYELER, Michael. **OpenCV with Python Blueprints**. Packt Publishing Ltd, 2015.

BEYMER, David *et al.* A real-time computer vision system for measuring traffic parameters. Em: **Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on**. IEEE, 1997. p. 495-501.

BRADSKI, Gary; KAEHLER, Adrian. **Learning OpenCV: Computer vision with the OpenCV library**. " O'Reilly Media, Inc.", 2008.

BRAKATSOULAS, Sotiris *et al.* On map-matching vehicle tracking data. Em: **Proceedings of the 31st international conference on Very large data bases**. VLDB Endowment, 2005. p. 853-864.

BRUTZER, Sebastian; HÖFERLIN, Benjamin; HEIDEMANN, Gunther. Evaluation of background subtraction techniques for video surveillance. Em: **Computer Vision and Pattern Recognition (CVPR)**, 2011 IEEE Conference on. IEEE, 2011. p. 1937-1944.

CAVALLARO, Andrea; STEIGER, Olivier; EBRAHIMI, Touradj. Tracking video objects in cluttered background. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 15, n. 4, p. 575-584, 2005.

CHEN, Min *et al.* Data, information, and knowledge in visualization. **IEEE Computer Graphics and Applications**, v. 29, n. 1, 2009.

CHEN, Mingliang *et al.* Spatiotemporal GMM for background subtraction with superpixel hierarchy. **IEEE transactions on pattern analysis and machine intelligence**, 2017.

CHEN, Zezhi; ELLIS, Tim. A self-adaptive Gaussian mixture model. **Computer Vision and Image Understanding**, v. 122, p. 35-46, 2014.

CHENG, Hsu-Yung; WENG, Chih-Chia; CHEN, Yi-Ying. Vehicle detection in aerial surveillance using dynamic Bayesian networks. **IEEE Transactions on Image Processing**, v. 21, n. 4, p. 2152-2159, 2012.

COIFMAN, Benjamin *et al.* A real-time computer vision system for vehicle tracking and traffic surveillance. **Transportation Research Part C: Emerging Technologies**, v. 6, n. 4, p. 271-288, 1998.

COLLINS, Robert T.; LIU, Yanxi; LEORDEANU, Marius. Online selection of discriminative tracking features. **IEEE transactions on pattern analysis and machine intelligence**, v. 27, n. 10, p. 1631-1643, 2005.

CONNELL, J. *et al.* Detection and tracking in the IBM PeopleVision system. Em: **Multimedia and Expo, 2004. ICME'04. 2004 IEEE International Conference on**. IEEE, 2004. p. 1403-1406.

COOLEY, M. Architecture or bee. 1987.

COOPER, Paul. Data, information, knowledge and wisdom. **Anaesthesia & Intensive Care Medicine**, v. 18, n. 1, p. 55-56, 2017.

CORNISH, David; DUKETTE, Diane. **The essential 20: Twenty components of an excellent health care team**. Dorrance Publishing, 2009.

CUCCHIARA, Rita; PICCARDI, Massimo. Vehicle Detection under Day and Night Illumination. Em: **IIA/SOCO**. 1999.

CUNHA, ANDRÉ LUIZ BARBOSA NUNES DA. (2007) **Avaliação do impacto da medida de desempenho no equivalente veicular de caminhões**. Dissertação (mestrado) EESC-USP, São Carlos, 142 p.

CUNHA, ANDRÉ LUIZ BARBOSA NUNES DA. **Sistema automático para obtenção de parâmetros do tráfego veicular a partir de imagens de vídeo usando OpenCV**. 2013. Tese de Doutorado. Universidade de São Paulo.

CYGANEK, Bogusław; GRUSZCZYŃSKI, Sławomir. Hybrid computer vision system for drivers' eye recognition and fatigue monitoring. **Neurocomputing**, v. 126, p. 78-94, 2014.

DNIT. **Manual de estudos de tráfego**. Departamento Nacional de Infra-estrutura de Transportes. Instituto de Pesquisas Rodoviárias. Rio de Janeiro, 2006.

DO, Viet Dung; WOO, Dong-Min. Multi-Resolution Estimation of Optical Flow for Vehicle Tracking. 2016.

DOOLEY, Damien *et al.* A blind-zone detection method using a rear-mounted fisheye camera with combination of vehicle detection methods. **IEEE Transactions on Intelligent Transportation Systems**, v. 17, n. 1, p. 264-278, 2016.

DUBOIS, Paul F. Guest editor's introduction: Python: batteries included. **Computing in Science & Engineering**, v. 9, n. 3, p. 7-9, 2007.

ELHABIAN, Shireen Y.; EL-SAYED, Khaled M.; AHMED, Sumaya H. Moving object detection in spatial domain using background removal techniques-state-of-art. **Recent patents on computer science**, v. 1, n. 1, p. 32-54, 2008.

FAGNANT, Daniel J.; KOCKELMAN, Kara. Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. **Transportation Research Part A: Policy and Practice**, v. 77, p. 167-181, 2015.

FAST ALGORITHM FOR DETECTION. OpenCV – FAST Algorithm for Corner Detection. Disponível em: <https://docs.opencv.org/3.2.0/df/d0c/tutorial_py_fast.html>. Acesso em 16 jun. 2018.

FAWCETT, Tom. An introduction to ROC analysis. **Pattern recognition letters**, v. 27, n. 8, p. 861-874, 2006.

GANTZ, John; REINSEL, David. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. **IDC iView: IDC Analyze the future**, v. 2007, p. 1-16, 2012.

GODBEHERE, Andrew B.; MATSUKAWA, Akihiro; GOLDBERG, Ken. Visual tracking of human visitors under variable-lighting conditions for a responsive audio art installation. Em: **2012 American Control Conference (ACC)**. IEEE, 2012. p. 4305-4312.

GONZALEZ, Rafael C.; WOODS, Richard E. Digital image processing. **Nueva Jersey**, 2008.

GU, Chuang; LEE, Ming-Chieh. Semiautomatic segmentation and tracking of semantic video objects. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 8, n. 5, p. 572-584, 1998.

HAN, Xiaowei *et al.* Research on moving object detection algorithm based on improved three Frame difference method and optical flow. Em: **Instrumentation and Measurement, Computer, Communication and Control (IMCCC), 2015 Fifth International Conference on**. IEEE, 2015. p. 580-584.

HARRIS, Chris; STEPHENS, Mike. A combined corner and edge detector. Em: **Alvey vision conference**. 1988. p. 50.

HEIKKILA, Janne; SILVEN, Olli. A four-step camera calibration procedure with implicit image correction. In: **Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on**. IEEE, 1997. p. 1106-1112.

HELD, David *et al.* Robust real-time tracking combining 3D shape, color, and motion. **The International Journal of Robotics Research**, v. 35, n. 1-3, p. 30-49, 2016.

HOU, Yi-You; CHIOU, Sz-Yu; LIN, Ming-Hung. Real-time detection and tracking for moving objects based on computer vision method. Em: **Control and Robotics Engineering (ICCARE), 2017 2nd International Conference on**. IEEE, 2017. p. 213-217.

HUNTER, John D. *et al.* Matplotlib: A 2D graphics environment. **Computing in science and engineering**, v. 9, n. 3, p. 90-95, 2007.

IDES PYTHON. Disponível em: <<http://wiki.python.org.br/IdesPython>>. Acesso em 29 set. 2016

IMAGE PROCESSING TOOLBOX - MATLAB. The Mathworks Inc. Disponível em: <<https://www.mathworks.com/products/image>>. Acesso em 29 set. 2016

JÄHNE, Bernd. **Digital image processing**. Berlin etc.: Springer, 2005.

JAIN, Anil K. **Fundamentals of digital image processing**. Prentice-Hall, Inc., 1989.

JAIN, Ashesh *et al.* Car that knows before you do: Anticipating maneuvers via learning temporal driving models. Em: **Proceedings of the IEEE International Conference on Computer Vision**. 2015. p. 3182-3190.

JAZAYERI, Amirali *et al.* Vehicle detection and tracking in car video based on motion model. **IEEE Transactions on Intelligent Transportation Systems**, v. 12, n. 2, p. 583-595, 2011.

KAESTNER, Ralf *et al.* Generative object detection and tracking in 3d range data. Em: **Robotics and Automation (ICRA), 2012 IEEE International Conference on**. IEEE, 2012. p. 3075-3081.

KAEWTRAKULPONG, Pakorn; BOWDEN, Richard. An improved adaptive background mixture model for real-time tracking with shadow detection. Em: **Video-based surveillance systems**. Springer US, 2002. p. 135-144.

KANHERE, Neeraj Krantiveer. **Vision-based detection, tracking and classification of vehicles using stable features with automatic camera calibration**. 2008. Tese de Doutorado. Clemson University.

KE, Ruimin *et al.* Real-time bidirectional traffic flow parameter estimation from aerial videos. **IEEE Transactions on Intelligent Transportation Systems**, v. 18, n. 4, p. 890-901, 2017.

KIM, J. B. *et al.* Wavelet-based vehicle tracking for automatic traffic surveillance. Em: **TENCON 2001. Proceedings of IEEE Region 10 International Conference on Electrical and Electronic Technology**. IEEE, 2001. p. 313-316.

KOCH, Christian *et al.* A review on computer vision based defect detection and condition assessment of concrete and asphalt civil infrastructure. **Advanced Engineering Informatics**, v. 29, n. 2, p. 196-210, 2015.

KOONCE, Peter *et al.* **Traffic signal timing manual**. United States. Federal Highway Administration, 2008.

KUHLMAN, Dave. **A python book: Beginning python, advanced python, and python exercises**. Dave Kuhlman, 2009.

LAGANIÈRE, Robert. **OpenCV 2 Computer Vision Application Programming Cookbook: Over 50 recipes to master this library of programming functions for real-time computer vision**. Packt Publishing Ltd, 2011.

LEDUC, Guillaume. Road traffic data: Collection methods and applications. **Working Papers on Energy, Transport and Climate Change**, v. 1, n. 55, 2008.

LI, Jing *et al.* Multi-target detection and tracking from a single camera in Unmanned Aerial Vehicles (UAVs). Em: **Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on**. IEEE, 2016. p. 4992-4997.

LIMA, S. F. *et al.* Systematic Review: Techniques and Methods of Urban Monitoring in Intelligent Transport Systems. **ICWMC 2017**, v. 17, p. 9, 2017.

LIN, Bin-Feng *et al.* Integrating appearance and edge features for sedan vehicle detection in the blind-spot area. **IEEE Transactions on Intelligent Transportation Systems**, v. 13, n. 2, p. 737-747, 2012.

LOWE, David G. Distinctive image features from scale-invariant keypoints. **International journal of computer vision**, v. 60, n. 2, p. 91-110, 2004.

LUCAS, Bruce D.; KANADE, Takeo. An iterative image registration technique with an application to stereo vision. 1981.

MANDELLOS, Nicholas A.; KERAMITSOGLOU, Iphigenia; KIRANOUDIS, Chris T. A background subtraction algorithm for detecting and tracking vehicles. **Expert Systems with Applications**, v. 38, n. 3, p. 1619-1631, 2011.

MCIVOR, Alan M. Background subtraction techniques. **Proc. of Image and Vision Computing**, v. 4, p. 3099-3104, 2000.

MEIER, Thomas; NGAN, King Ngi. Automatic segmentation of moving objects for video object plane generation. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 8, n. 5, p. 525-538, 1998.

MENDES, Jean Carlo; BIANCHI, Andrea Gomes Campos; JÚNIOR, Álvaro Rodrigues Pereira. Vehicle Tracking and Origin-destination Counting System for Urban Environment. Em: **VISAPP (3)**. 2015. p. 600-607.

NAKAGAWA, Keiichi *et al.* Sequentially timed all-optical mapping photography (STAMP). **Nature Photonics**, v. 8, n. 9, p. 695-700, 2014.

NEUMEYER, David. **The Oxford handbook of film music studies**. Oxford University Press, 2013.

NIXON, Mark. **Feature extraction & image processing**. Academic Press, 2008.

NUMPY. Disponível em: <<http://www.numpy.org/>>. Acesso em 15 set. 2016.

OLIPHANT, Travis E. Python for scientific computing. **Computing in Science & Engineering**, v. 9, n. 3, p. 10-20, 2007.

OPEN SOURCE INITIATIVE. Disponível em: <<https://opensource.org/>>. Acesso em 15 set. 2016

OPENCV. Open Source Computer Vision. Disponível em: <<http://opencv.org/>>. Acesso em 15 set. 2016.

PARLIAMENT, T. E.; UNION, TCOTE. Directive 2010/40/eu of the european parliament and of the council. **Official Journal of the European Union**, v. 50, p. 207, 2010.

PETERFREUND, Natan. Robust tracking of position and velocity with Kalman snakes. **IEEE transactions on pattern analysis and machine intelligence**, v. 21, n. 6, p. 564-569, 1999.

PETROVSKAYA, Anna; THRUN, Sebastian. Model based vehicle detection and tracking for autonomous urban driving. **Autonomous Robots**, v. 26, n. 2-3, p. 123-139, 2009.

PICCARDI, Massimo. Background subtraction techniques: a review. Em: **Systems, man and cybernetics, 2004 IEEE international conference on**. IEEE, 2004. p. 3099-3104.

PRECHELT, Lutz. An empirical comparison of C, C++, Java, Perl, Python, Rexx and Tcl. **IEEE Computer**, v. 33, n. 10, p. 23-29, 2000.

PRECHELT, Lutz. Are scripting languages any good? A validation of Perl, Python, Rexx, and Tcl against C, C++, and Java. **Advances in Computers**, v. 57, p. 205-270, 2003.

PYTHON SOFTWARE FOUNDATION – OVERVIEW v2. Overview – Python v2.7.12 documentation. Disponível em: <<https://docs.python.org/2/>>. Acesso em 15 set. 2016.

PYTHON SOFTWARE FOUNDATION – OVERVIEW v3. Overview – Python v3.0.1 documentation. Disponível em: <<https://docs.python.org/3.0/>>. Acesso em 15 set. 2016.

PYTHON SOFTWARE FOUNDATION - PYPI. PyPI – the Python Package Index. Disponível em: <<https://pypi.python.org/pypi>>. Acesso em: 14 set. 2016.

PYTHON SOFTWARE FOUNDATION – QUOTES. Quotes about Python. Disponível em: <<https://www.python.org/about/quotes/>>. Acesso em: 15 set. 2016.

PYTHON SOFTWARE FOUNDATION. Disponível em: < <https://www.python.org/>>. Acesso em 14 set. 2016.

REYNOLDS, T. *et al.* Airport Systems, Planning, Design and Management. 2013.

ROSTEN, Edward; DRUMMOND, Tom. Machine learning for high-speed corner detection. **Computer Vision–ECCV 2006**, p. 430-443, 2006.

RUSS, John C.; RUSS, J. Christian. **Introduction to image processing and analysis**. CRC press, 2017

SAUNIER, Nicolas; SAYED, Tarek. A feature-based tracking algorithm for vehicles in intersections. Em: **The 3rd Canadian Conference on Computer and Robot Vision (CRV'06)**. IEEE, 2006. p. 59-59.

SCIPY. Disponível em: <<http://www.scipy.org/>>. Acesso em 15 set. 2016.

SEN-CHING, S. Cheung; KAMATH, Chandrika. Robust techniques for background subtraction in urban traffic video. Em: **Electronic Imaging 2004**. International Society for Optics and Photonics, 2004. p. 881-892.

SHI, Jianbo; TOMASI, Carlo. Good features to track. Em: **Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on**. IEEE, 1994. p. 593-600.

SHIH, Frank Y.; ZHONG, Xin. Automated Counting and Tracking of Vehicles. **International Journal of Pattern Recognition and Artificial Intelligence**, p. 1750038, 2017.

SILVA, Andrei Santos *et al.* Object Tracking by Color and Active Contour Models Segmentation. **IEEE Latin America Transactions**, v. 14, n. 3, p. 1488-1493, 2016.

SIVANANTHAM, S.; PAUL, N. Nitin; IYER, R. Suraj. Object tracking algorithm implementation for security applications. **Far East Journal of Electronics and Communications**, v. 16, n. 1, p. 1, 2016.

SOBRAL, Andrews; VACAVANT, Antoine. A comprehensive review of background subtraction algorithms evaluated with synthetic and real videos. **Computer Vision and Image Understanding**, v. 122, p. 4-21, 2014.

SONG, Xuefeng; NEVATIA, Ram. Detection and tracking of moving vehicles in crowded scenes. Em: **Motion and Video Computing, 2007. WMVC'07. IEEE Workshop on**. IEEE, 2007. p. 4-4.

SONKA, Milan; HLAVAC, Vaclav; BOYLE, Roger. **Image processing, analysis, and machine vision**. Cengage Learning, 2014.

SOUZA CASTRO, Fernando *et al.* IDENTIFICAÇÃO E RASTREAMENTO DE VEÍCULOS UTILIZANDO FLUXO ÓPTICO. Em: **Colloquium Exactarum**. 2016. p. 73-88.

STAUFFER, Chris; GRIMSON, W. Eric L. Learning patterns of activity using real-time tracking. **IEEE Transactions on pattern analysis and machine intelligence**, v. 22, n. 8, p. 747-757, 2000.

STURARI, Mirco *et al.* An integrated mobility system using real-time data for traffic simulation. Em: **Mechatronic and Embedded Systems and Applications (MESA), 2016 12th IEEE/ASME International Conference on**. IEEE, 2016. p. 1-6.

SUN, Shijun; HAYNOR, David R.; KIM, Yongmin. Semiautomatic video object segmentation using VSnares. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 13, n. 1, p. 75-82, 2003.

TAO, Hai; SAWHNEY, Harpreet S.; KUMAR, Rakesh. Object tracking with bayesian estimation of dynamic layer representations. **IEEE transactions on pattern analysis and machine intelligence**, v. 24, n. 1, p. 75-89, 2002.

TESKEY, Francis N. User models and world models for data, information, and knowledge. **Information processing & management**, v. 25, n. 1, p. 7-14, 1989.

TRNOVSZKÝ, Tibor; SÝKORA, Peter; HUDEC, Róbert. Comparison of background subtraction methods on near Infra-Red spectrum video sequences. **Procedia engineering**, v. 192, p. 887-892, 2017.

UZKENT, Burak; RANGNEKAR, Aneesh; HOFFMAN, Matthew J. Aerial vehicle tracking by adaptive fusion of hyperspectral likelihood maps. Em: **Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on**. IEEE, 2017. p. 233-242.

VELHO, Luiz; FRERY, Alejandro C.; GOMES, Jonas. **Image processing for computer graphics and vision**. Springer Science & Business Media, 2009.

VERNON, David. Machine vision - Automated visual inspection and robot vision. **NASA STI/Recon Technical Report A**, v. 92, p. 40499, 1991.

WOO, Dong-Min; HONG, An Nguyen. An Integrated Solution to Motion Tracking of Moving Vehicles Using Optical Flow. **Int. Journal of Computing, Communication and Instrumentation Eng.**, 1 (1), p. 60-63, 2014.

WORRALL, Anthony D. *et al.* A simple, intuitive camera calibration tool for natural images. In: **BMVC**. 1994. p. 1-10.

WREN, Christopher Richard *et al.* Pfinder: Real-time tracking of the human body. **IEEE Transactions on pattern analysis and machine intelligence**, v. 19, n. 7, p. 780-785, 1997.

YILMAZ, Alper; JAVED, Omar; SHAH, Mubarak. Object tracking: A survey. **Acm computing surveys (CSUR)**, v. 38, n. 4, p. 13, 2006.

ZHAN, Chaohui *et al.* An improved moving object detection algorithm based on Frame difference and edge detection. Em: **Image and Graphics, 2007. ICIG 2007. Fourth International Conference on**. IEEE, 2007. p. 519-523.

ZHAO, Jian-Wei; WANG, Peng; LIU, Chong-Qing. An object tracking algorithm based on occlusion mesh model. Em: **Machine Learning and Cybernetics, 2002. Proceedings. 2002 International Conference on**. IEEE, 2002. p. 288-292.

ZHENG, Jianyang *et al.* Extracting roadway background image: Mode-based approach. **Transportation Research Record: Journal of the Transportation Research Board**, n. 1944, p. 82-88, 2006.

ZIVKOVIC, Zoran. Improved adaptive Gaussian mixture model for background subtraction. Em: **Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on**. IEEE, 2004. p. 28-31.

ZIVKOVIC, Zoran; VAN DER HEIJDEN, Ferdinand. Efficient adaptive density estimation per image pixel for the task of background subtraction. **Pattern recognition letters**, v. 27, n. 7, p. 773-780, 2006.