UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

THIAGO WASZAK ALVES

NOVO MÉTODO ITERATIVO DE LOCALIZAÇÃO DA CÂMERA BASEADO NO CONCEITO DE RESECTION-INTERSECTION

Porto Alegre
2021

THIAGO WASZAK ALVES

NOVO MÉTODO ITERATIVO DE LOCALIZAÇÃO DA CÂMERA BASEADO NO CONCEITO DE RESECTION-INTERSECTION

Tese de doutorado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de Doutor em Engenharia Elétrica.

Área de concentração: Engenharia da Computação

ORIENTADOR: Prof. Dr. Altamiro A. Susin

Porto Alegre
2021

THIAGO WASZAK ALVES

NOVO MÉTODO ITERATIVO DE LOCALIZAÇÃO DA CÂMERA BASEADO NO CONCEITO DE RESECTION-INTERSECTION

Esta tese foi julgada adequada para a obtenção do título de Doutor em Engenharia Elétrica e aprovada em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____
Prof. Dr. Altamiro A. Susin, UFRGS
Doutor pelo Instituto Nacional Politécnico de Grenoble – Grenoble, França

Banca Examinadora:

Prof. Dr. César Augusto Missio Marcon, PUCRS
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Prof. Dr. Cláudio Rosito Jung, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Prof. Dr. Ivan Müller, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Coordenador do PPGEE: _____
Prof. Dr. Sérgio Luís Haffner

Porto Alegre, dezembro de 2021.

AGRADECIMENTOS

A minha mãe, Zorane, que sempre me incentivou e apoiou ao longo dessa caminhada. Quero dizer que essa conquista não é só minha, mas nossa. Ao meu tio, Rui, que meu ajudou a chegar a onde eu precisava e pelas palavras de incentivo, muito obrigado. A minha prima, Rosana, que sempre torceu por mim e me ajudou em diversas ocasiões, muito obrigado. Tudo que consegui só foi possível graças ao amor, apoio e dedicação que vocês sempre tiveram por mim.

Aos professores, pesquisadores, doutorandos, mestrandos e graduandos do LaPSI pelas contribuições e debates nos diversos trabalhos que desenvolvemos juntos e aos colegas e professores do PPGEE pelos conhecimentos e experiências e todos que de alguma maneira foram importantes para a realização desse trabalho.

Ao meu orientador e amigo Prof. Dr. Altamiro Amadeu Susin, pela acolhida, confiança, orientação e incentivo dados durante todos esses anos de trabalho, dos quais este trabalho não teria sido possível.

Ao Programa de Pós-Graduação em Engenharia Elétrica, PPGEE, pela oportunidade de realização do trabalho de pesquisa na área de Processamento de Sinais.

“Sic Parvis Magna”
(Sir Francis Drake)

RESUMO

A Odometria Visual é o processo de estimar o movimento de um ente a partir de duas ou mais imagens fornecidas por uma ou mais câmeras. É uma técnica de grande importância na visão computacional, com aplicações em diversas áreas tais como assistência ao motorista e navegação de veículos autônomos, sistemas de realidade aumentada, veículos autônomos não-tripulados (VANTs) e até mesmo na exploração interplanetária. Os métodos mais comuns de Odometria Visual utilizam câmeras com visão estéreo, através das quais é possível calcular diretamente as informações de profundidade de detalhes de uma cena, o que permite estimar as posições sucessivas das câmeras. A Odometria Visual Monocular estima o deslocamento de um objeto com base nas imagens fornecidas por uma única câmera, o que oferece vantagens construtivas e operacionais embora exija processamento mais complexo. Os sistemas de Odometria Visual Monocular do tipo esparsos estimam a pose da câmera a partir de singularidades detectadas nas imagens, o que reduz significativamente o poder de processamento necessário, sendo assim ideal para aplicações de tempo real. Nessa óptica, este trabalho apresenta um novo sistema de Odometria Visual Monocular esparsa para tempo real, validado em veículo instrumentado. O novo sistema é baseado no conceito de *Resection-Intersection*, combinado com um novo teste de convergência, e um método de refinamento iterativo para minimizar os erros de reprojeção. O sistema foi projetado para ser capaz de utilizar diferentes algoritmos de otimização não linear, tais como Gauss-Newton, Levenberg-Marquardt, Davidon-Fletcher-Powell ou Broyden–Fletcher–Goldfarb–Shannon. Utilizando o *benchmark* KITTI, o sistema proposto obteve um erro de translação em relação à distância média percorrida de $0,86\%$ e erro médio de rotação em relação à distância média percorrida de $0.0024°/m$. O sistema foi desenvolvido em Python em uma única *thread*, foi embarcado em uma placa Raspberry Pi 4B e obteve um tempo médio de processamento de $775ms$ por imagem para os onze primeiros cenários do *benchmark*. O desempenho obtido neste trabalho supera os resultados de outros sistemas de Odometria Visual Monocular baseados no conceito de *Resection-Intersection* até o momento submetidos na classificação do *benchmark* KITTI.

Palavras-chave: Odometria Visual Monocular, Resection-Intersection, Visão Computacional, SLAM, Lucas-Kanade, KITTI Dataset, Processamento de Imagens.

ABSTRACT

Visual Odometry is the process of estimating the movement of an entity from two or more images provided by one or more cameras. It is a technique ofmain concern in computer vision, with applications in several areas such as driver assistance and autonomous vehicle navigation, augmented reality systems, Unmanned Aerial Vehicle (UAV) and even in interplanetary exploration. Most common methods of Visual Odometry use stereo cameras, through which it is possible to directly calculate the depth information of details of a scene, which allows to estimate the successive positions of the cameras. Monocular Visual Odometry estimates the displacement of an object based on images provided by a single camera, which offers constructive and operational advantages although it requires more complex processing. Sparse-type Monocular Visual Odometry systems estimate the camera pose from singularities detected in the images, which significantly reduces the processing power required, thus making it ideal for real-time applications. In this perspective, this work presents a new Sparse Monocular visual Odometry system for real-time, validated on a instrumented vehicle. The new system is based on the Resection-Intersection concept, combined with an expanded convergence test, and an iterative refinement method to minimize reprojection errors. It was designed to be able to use different non-linear optimization algorithms, such as Gauss-Newton, Levenberg-Marquardt, Davidon-Fletcher-Powell or Broyden–Fletcher–Goldfarb–Shannon. Using the benchmark KITTI, the proposed system obtained a translation error in relation to the average distance traveled of 0.86% and an average rotation error in relation to the average distance covered of $0.0024^{\circ}/m$. The system was developed in Python on a single thread, was embedded on a Raspberry Pi 4B board and an average processing time of $775ms$ per image for the first eleven scenarios of the benchmark. The results obtained in this work surpass the results obtained by other visual odometry systems based on the concept of Resection-Intersection so far submitted to the KITTI benchmark ranking.

Keywords: Monocular Visual Odometry, Resection-Intersection, Computer Vision, SLAM, Lucas-Kanade, KITTI Dataset, Image Processing.

LISTA DE ILUSTRAÇÕES

Figura 1 – Conceito de sistema de assistência ao motorista. 22
Figura 2 – Odometria Visual Densa e Semi-densa. 23
Figura 3 – Ilustração do conceito de discretização de imagens por câmeras digitais. 30
Figura 4 – Ilustração do conceito de pixel e da convenção do sistema de coordenadas para representação de imagens digitais 30
Figura 5 – Ilustração do processo de aquisição de imagens digitais 32
Figura 6 – Ilustração do funcionamento e modelo de uma câmera Pinhole 33
Figura 7 – Ilustração do sistema de coordenadas utilizado neste trabalho. 34
Figura 8 – Exemplo de erro de reprojeção. 36
Figura 9 – Detecção e Rastreamento de *features* 37
Figura 10 – Ilustração do efeito da Equalização de Histograma. 40
Figura 11 – Geometria Epipolar 42
Figura 12 – Exemplo de linhas epipolares. 42
Figura 13 – Exemplo de situação de triangulação sem erros. 43
Figura 14 – Segmento de interseção perpendicular de triangulação. 44
Figura 15 – Triangulação ideal. 44
Figura 16 – Ilustração da aplicabilidade e limitações da matriz fundamental. . . . 48
Figura 17 – Ciclo de funcionamento do *Resection-Intersection*. 49
Figura 18 – Exemplo clássico de ambiguidade de similaridade. 50
Figura 19 – Fluxograma genérico de OVM. 51
Figura 20 – Arquitetura do sistema RI_MVO. 57
Figura 21 – Fluxograma de operação do sistema OVM esparso proposto neste trabalho. 61
Figura 22 – Ilustração do resultado da equalização adaptativa de histograma. . . . 62
Figura 23 – Ilustração dos resultados obtidos com cada uma das técnicas de extração de *features* propostas. 64
Figura 24 – Exemplificação do procedimento realizado na etapa Geometrica . . . 65
Figura 25 – Ilustração das formas de região testadas. 66
Figura 26 – Resultados obtidos utilizando diferentes áreas de separação. 67
Figura 27 – Exemplo da área de classificação da *features* presentes no solo. 68
Figura 28 – Estrutura de triângulos utilizadas para estimar a escala real da cena. . 69
Figura 29 – Ilustração do ciclo de operação do método de refinamento cíclico. . . 80
Figura 30 – Tipos de Odometria Visual e exemplo de estimação de trajetória. . . . 84
Figura 31 – Ilustração da Raspberry Pi 4 modelo B. 85
Figura 32 – Resultados obtidos utilizando o sistema GN_RI. 90
Figura 33 – Resultados obtidos utilizando o sistema LM_RI. 91
Figura 34 – Resultados obtidos utilizando o sistema DFP_RI e BFGS_RI. 92

Figura 35 – Resultados obtidos utilizando o sistema CY_RI. 93
Figura 36 – Ilustração dos resultados de todos os métodos. 93
Figura 37 – Exemplo de trechos desafiadores. 94
Figura 38 – Ilustração dos resultados dos trabalhos RI_MVO, DVSO e D3VO e pelo sistema CY_RI implementado neste trabalho. 98
Figura 39 – Trajetórias de 00 até 07 estimadas pelo método GN_RI. 114
Figura 40 – Trajetórias de 00 até 07 estimadas pelo método LM_RI. 115
Figura 41 – Trajetórias de 00 até 07 estimadas pelo método DFP_RI. 116
Figura 42 – Trajetórias de 00 até 07 estimadas pelo método BFGS_RI. 117
Figura 43 – Trajetórias de 00 até 07 estimadas pelo método CY_RI. 118

LISTA DE TABELAS

Tabela 1 – Tabela de comparação entre trabalhos 60
Tabela 2 – Desempenho das principais técnicas de detecção de *features*. 64
Tabela 3 – Tabela de características da Raspberry Pi 4B 86
Tabela 4 – Resultados utilizando os onze primeiros cenários do KITTI. 89
Tabela 5 – Tempo de execução médio obtido nos onze primeiros cenários do KITTI. 95
Tabela 6 – Tempo de execução médio das principais etapas do sistema. 95
Tabela 7 – Número de Iterações vs. Erro de Reprojeção. 96
Tabela 8 – Tabela de comparação dos resultados 97

LISTA DE ABREVIATURAS

2B	Bidimensional
3D	Tridimensional
ADAS	*Advanced Driver Assistance Systems*
A/D	Analógico/Digital
BA	*Bundle Adjustment*
BFGS	Broyden–Fletcher–Goldfarb–Shanno.
BRIEF	*Binary Robust Independent Elementary Features*
CLAHE	Contraste Limitado Equalização Adaptativa do Histograma
CMOS	*Complementary Metal Oxide Semiconductor*
CPU	*Central Processing Unit*
DFP	Davidon Fletcher Powell
DVSO	*Deep Virtual Stereo Odometry*
EAH	Equalização Adaptativa do Histograma
EPnP	*Efficient Perspective-n-Point Camera Pose Estimation*
FAST	*Features Accelerated Segment Test*
FPGA	*Field Programmable Gate Array*
GPS	*Global Positioning System*
GPU	*Graphics Processing Unit*
IOT	*Internet of Things*
KF	Filtro de Kalman
LE	Linhas Epipolares
LK	Lucas-Kanade
NoC	*Network on Chip*
OpenCV	*Open Source Computer Vision Library*
ORB	*Oriented FAST and rotated BRIEF*
OV	Odometria Visual

OVE Odometria Visual Estéreo

OVM Odometria Visual Monocular

P3P Perspectiva-3-Pontos

P5P Perspectiva-5-Pontos

PC *Personal Computer*

PnP Perspectiva-n-Pontos

PTAM *Parallel Tracking and Mapping*

PVS *Personal Vehicle System*

RAM *Random Access Memory*

RCNN *Region Based Convolutional Neural Networks*

RGB *Red-Green-Blue*

RI *Resection-Intersection*

SFM *Structure From Motion*

SIFT *Scale-invariant Feature Transform*

SLAM *Simultaneous Localization and Mapping*

SNA Sistemas de Navegação Autônomos

SoC *System On a Chip*

SURF *Speeded-Up Robust Features*

TV Televisão

VA Veículos Autônomos

VANT Veículo Aéreo Não-Tripulado

VSLAM Visual Simultaneous Localization and Mapping

SUMÁRIO

1 INTRODUÇÃO 21
1.1 Odometria Visual 23
1.2 Motivação 24
1.3 Objetivo 25
1.4 Contribuições 25
1.5 Notação 26
1.6 Organização 26

2 FUNDAMENTOS DE ODOMETRIA VISUAL 29
2.1 Representação Digital de uma Imagem 29
2.2 Aquisição das Imagens 31
2.3 Modelo de Projeção e Sistema de Coordenadas 32
2.4 Conversão de Sistemas de Coordenadas 35
2.5 Erro de Reprojeção 35
2.6 Detecção e Rastreamento de Features 37
2.6.1 Detector de Canto Harris 38
2.6.2 Rastreador Lucas-Kanade 39
2.7 Equalização de Histograma 39
2.8 Geometria Epipolar 41
2.9 Triangulação 43
2.10 Otimização Numérica de Gauss-Newton 44
2.11 Otimização Numérica de Levenberg-Marquardt 46
2.12 Otimização Numérica de Davidon-Fletcher-Powell 46
2.13 Otimização Numérica de Broyden-Fletcher-Goldfarb-Shanno 47
2.14 Bundle Adjustment 48
2.15 Resection-Intersection 49
2.16 Ambiguidade da Similaridade da Cena 50
2.17 Configuração Básica de Odometria Visual Monocular Esparsa 51

3 REVISÃO DA LITERATURA 53
3.1 MPVue 55
3.2 MLM-SFM 56
3.3 RI_MVO 56
3.4 DVSO 58
3.5 SGANVO 58
3.6 D3VO 59
3.7 OV2SLAM 59

3.8 Comparação de Trabalhos 60

4 ARQUITETURA DO SISTEMA 61
4.1 Processamento de Imagem 62
4.2 Detecção e Rastreamento de Features 63
4.3 Estimação da Pose da Câmera 65
4.4 Recuperação da Escala da Cena 66

5 MÉTODOS DE ESTIMAÇÃO DA POSE DA CÂMERA 71
5.1 Resection-Intersection Baseado na Otimização de GN 71
5.2 Resection-Intersection Baseado na Otimização de LM 73
5.3 Teste de Convergência Expandido 75
5.4 Resection-Intersection Baseado na Otimização de DFP 77
5.5 Resection-Intersection Baseado na Otimização BFGS 78
5.6 Resection-Intersection com Refinamento Cíclico 79

6 RESULTADOS 83
6.1 Banco de Dados KITTI 84
6.2 Raspberry Pi 85
6.3 Ambiente de Desenvolvimento 87
6.4 Calibração Inicial do Sistema 87
6.5 Validação do Sistema Proposto 88
6.6 Avaliação do Esforço Computacional 94
6.7 Discussão e Comparação 97

7 CONCLUSÃO 101

REFERÊNCIAS 103

8 APÊNDICE 113

1 INTRODUÇÃO

O desenvolvimento tecnológico foi o motor das revoluções industriais, e trouxe, sem sombra de dúvida, melhores condições de vida para a humanidade. No século passado, a eletricidade e a eletrônica foram responsáveis pela onda das comunicações, seguida na segunda metade do século, pelo surgimento da sociedade da informação com a evolução do mundo digital. A flexibilidade dos sistemas digitais permitiu a simbiose das atividades de instrumentação, controle e processamento de informações com praticamente todas as áreas do conhecimento ligadas à tecnologia: da medicina à agricultura, da geração de energia, ao comércio e às finanças, das artes às interações entre pessoas e grupos e, principalmente, na automatização de processos. Embora a área da robótica tenha focado principalmente em contextos industriais, o campo da robótica móvel expandiu-se rapidamente nos últimos vinte anos, impulsionado por uma combinação de fatores, incluindo avanços em sensores pequenos e leves, que tornaram viáveis a resolução de muitos problemas na área de veículos de transporte.

Impulsionado pela evolução da eletrônica, dispositivos equipados com câmeras experimentaram um crescimento exponencial na última década devido a popularização de dispositivos como smartphones. Este cenário motivou muitos eventos e projetos da comunidade científica, aplicando otimizações em algoritmos e arquiteturas de hardware para lidar com a crescente demanda de potência de processamento de aplicações na área de Visão Computacional (VC).

A Visão Computacional é um campo multidisciplinar que estuda métodos para compreensão e interpretação automática de imagens (RADKE, 2012). Sistemas de visão podem oferecer uma grande quantidade de informações sobre a disposição de objetos em um cenário, ao contrário de outros tipos de sistemas de odometria que necessitam um conjunto de sensores para obter todas as informações necessárias do cenário. Outro grande vantagem é que, além das câmeras monoculares serem de baixo custo, não geram ou sofrem interferência na presença de sistema similares, pois atividade dos sensores não influencia o estado do ambiente, diferentemente do que acontece com sensores que emitem luz infravermelha, laser ou pulsos de ultrassom.

Veículos Autônomos (VA) são aqueles capazes de se movimentar no ambiente seguindo um trajeto planejado, de forma rápida, eficiente e segura, proporcionando uma melhorar significativa na áreas de segurança, economia e ambiental. O primeiro veículo inteligente que empregou o sistema de visão para detecção de obstáculos foi o Veículo Inteligente (DAVISON *et al.*, 2007) que foi desenvolvido em meados dos anos 1970. Hoje em dia, muitos dos veículos mais modernos estão equipados com sistemas de assistência de direção, que auxiliam o motorista em várias situações para tornar a direção mais segura e fácil (MENG; HONG; CHEN, 2009; KLEIN; MURRAY, 2007).

Figura 1 – Conceito de sistema de assistência ao motorista.

Fonte: (MENG; HONG; CHEN, 2009)

Sistemas de assistência de direção baseados em visão, como o ilustrado na Figura 1, podem determinar as posições das faixas, sinais de trânsito, rastrear as posições de outros carros na rua, e usar essas informações para ajudar o motorista a guiar o veículo e aumentar a segurança na locomoção (ALVES, 2017; HOELSCHER, 2017).

Nos dias de hoje, veículos autônomos são o grande foco do desenvolvimento de sistemas automobilísticos inteligentes da indústria. Tanto a Google como a Tesla acreditam que o futuro da condução será predominantemente autônoma (BERGEN, 2015; LEVIN, 2019). Todos os veículos que trafegam por uma cidade poderão se comunicar entre si, com o sistema de tráfego interconectado e compartilhar informações de forma a possibilitar um controle e dinâmica de trânsito nunca antes visto (PIPER, 2020). Conforme a função de condução tornar-se cada vez mais automatizada, há também uma mudança na responsabilidade do motorista humano ao próprio veículo, e consequentemente, um aumento no número de normas técnicas de segurança de forma a estender os requisitos mínimos de segurança dos veículos (LDSV, 2021). Esse aumento será um dos catalisadores da demanda futura no desenvolvimento de tecnologia na área de VA.

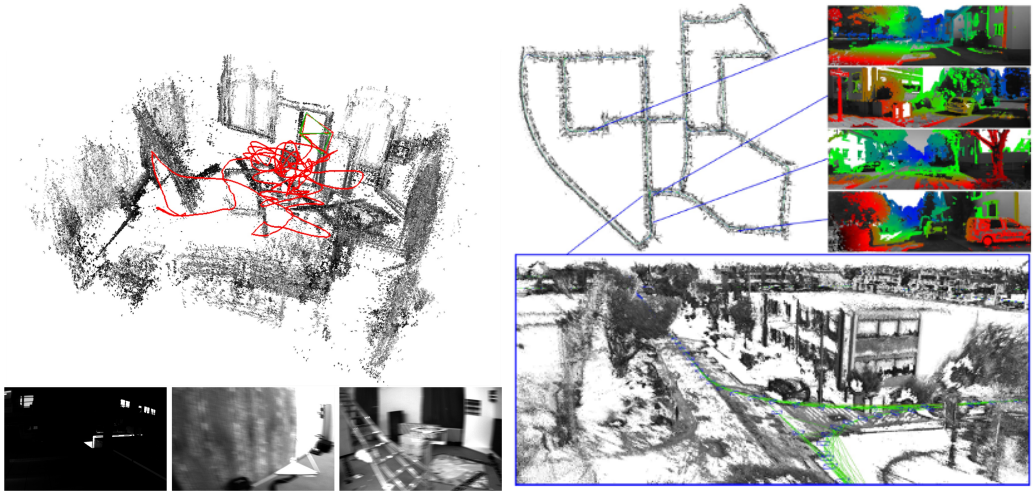
1.1 Odometria Visual

A Odometria Visual é o processo de estimar o movimento de um agente (por exemplo, veículo, pessoa e robô) utilizando duas ou mais imagens fornecidas por uma ou mais câmeras. É uma área de grande importância na visão computacional, com aplicações em diversas áreas, tais como assistência ao motorista e navegação de veículos autônomos, sistemas de realidade aumentada, veículos autônomos não-tripulados (VANTs) e até mesmo na exploração interplanetária. As técnicas mais comuns de Odometria Visual utilizam câmeras com visão estéreo, através das quais é possível calcular diretamente as informações de profundidade da cena, que são utilizadas para estimar a posição relativa da câmera.

A visão estéreo geralmente proporciona uma estimativa da câmera mais precisa. No entanto, os sistemas visão estéreo podem exigir auto-calibração após operação de longo prazo, uma vez que a vibração mecânica pode afetar gradativamente a posição das lentes (ROSTEN; DRUMMOND, 2006; YVES BOUGUET, 2000).

Já a Odometria Visual Monocular (OVM) estima a posição e a orientação da câmera, a partir de imagens sucessivas de uma única câmera, sem a necessidade de auto-calibração. É caracterizado como um problema computacional mais complexo, uma vez a estimação da profundidade da cena e da escala real da cena são mais complexos, porém, mais atraente devido à ampla popularidade das câmera monoculares.

Figura 2 – Odometria Visual Densa e Semi-densa.



(a) Exemplo de Odometria Visual Semi-Densa (b) Exemplo de Odometria Visual Densa

Fonte: (ENGEL, 2021).

Os sistemas OVM que extraem grandes quantidades de informações das imagens são chamados de soluções Semi-Densas ou Densas (GAO *et al.*, 2003; ENGEL; STURM; CREMERS, 2013). Essas soluções tentam ajustar a pose da câmera e a estrutura da cena combinando todos os píxeis com gradiente de imagem significativo. São bastante utilizadas em aplicações onde se deseja mapear tridimensionalmente um ambiente, e representa

uma alternativa de baixo custo em relação aos sistemas de mapeamento a laser dos veículos autônomos atuais, sem causar ou sofrer interferência de outros sistemas similares. Porém, não fornecem uma estimativa da pose da câmera muito precisa, visto que devido à grande quantidade de informação utilizada, são bastante afetados por *outliers*.

Um exemplo de Odometria Semi-Densa é mostrado na Figura 2 (a) onde técnicas de Odometria Visual foram utilizadas para mapear um ambiente fechado, utilizando milhares de *features*. A Figura 2 (b) ilustra um exemplo de Odometria Densa, onde um sistema OVM foi utilizado para criar um mapa 3D de partes de uma cidade, utilizando um número de *features* da ordem de centena de milhares (GEIGER; LENZ; URTASUN, 2012).

Por outro lado, sistemas OVM do tipo esparsos (SONG; CHANDRAKER; GUEST, 2016) selecionam regiões na imagem de fácil identificação, e extraem da imagem um número de *features* da ordem de centenas. Essas *features* são localizadas nas imagens subsequentes. A movimentação da câmera pela cena resulta em um deslocamento relativo dessas *features* rastreadas, que é utilizado para estimar a pose da câmera. Uma vez que, sistemas OVM esparsos utilizam uma quantidade de informação menor o poder computacional demandado diminui significativamente, sendo dessa forma, ideais para sistemas que são executados em tempo real (KEJARIWAL; ORSINI, 2016).

1.2 Motivação

A localização é parte essencial da navegação autônoma e de outros sistemas de reconstrução 3D. O problema de localização consiste em estimar a posição do robô ou veículo autônomo com relação a algum referencial externo baseado na informação recuperada de uma sequência de imagens.

Embora câmeras sejam sensores com grande riqueza de informação, técnicas de Odometria Visual são pouco usadas pelo fato de muitas delas necessitarem de um alto poder computacional ou não atingirem requisitos de tempo real ou mesmo pela alta complexidade dos algoritmos envolvidos. Além disso, um veículo autônomo depende de um sistema de navegação preciso e robusto, capaz de lidar com a natureza imprevisível do trânsito em ruas e rodovias de forma eficiente. Para tanto, o sistema também precisa conhecer sua localização em relação à estrutura da rua, aos outros veículos e aos demais elementos (por exemplo: ciclistas, placas de sinalização) que descrevem o ambiente urbano complexo.

Embora sistemas de Odometria Visual apresentem erros em suas estimativas para a localização, é uma escolha óbvia para sistemas de controle veicular em ambiente urbano, uma vez que a aquisição de imagens do ambiente já é realizada para a extração de informação semântica contida em placas e outras sinalizações, utilizadas para ajustar o controle do veículo.

1.3 Objetivo

O objetivo da presente tese é elaborar um novo sistema de Odometria Visual Monocular, baseado no conceito *Resection-Intersection*. O novo sistema deve ser capaz de atingir alta precisão na estimativa da pose da câmera com um baixo custo computacional, permitindo a implementação de um sistema de navegação que opere em tempo real, utilizando hardware de capacidade moderada e de fácil acesso.

A Odometria Visual para veículos equipados com câmera foi escolhida como estudo de caso: um veículo equipado com câmera monocular fornece uma sequência de imagens, e a posição e orientação da câmera devem ser determinadas para cada imagem fornecida. Essa posição pode ser utilizada para realizar a construção de uma nuvem de pontos que permita extração de informações relevantes para seu objetivo, seja de simples deslocamento evitando choques, seja a identificação da presença de objetos e suas distâncias. Sistemas OVM do tipo esparso são mais afetados por *outliers* e por problemas de estabilidade (PERSSON *et al.*, 2015), o que impõem um nível adicional de dificuldade quando comparado com outros trabalhos com múltiplas configurações de câmera, resultando em um problema mais complexo e desafiador.

1.4 Contribuições

Durante o desenvolvimento da tese, diversas soluções baseadas no conceito de RI foram implementadas, testadas e comparadas, de forma, a contribuir com o estado da arte através da busca pela otimização dos métodos de estimativa da pose da câmera de OVM do tipo esparsa.

Nesse contexto, este trabalho apresenta uma nova técnica para melhoramento da precisão da Odometria Visual Monocular esparsa direcionada a veículos autônomos e sistemas de assistente de direção. Com base nessa nova técnica foi desenvolvido um sistema OVM do tipo esparso que foi avaliado utilizando o popular banco de dados KITTI. Suas principais contribuições são as seguintes:

- O novo sistema OVM esparso, baseado no conceito de *Resection-Intersection*, combinado com um teste de convergência com região de busca expandida e Equalização de Histograma.
- Um novo método de *Resection-Intersection* baseado em refinamento cíclico.
- Utilização e comparação de diferentes algoritmos de otimização não linear: Gauss-Newton, Levenberg-Marquardt, Davidon-Fletcher-Powell e Broyden–Fletcher– Goldfarb–Shanno.

O sistema OVM proposto associa simplicidade, precisão e baixo custo computacional, somado a uma rápida taxa de convergência e robustez a pequenos erros na estimativa inicial, conforme será mostrado no capítulo de resultados. A avaliação de desempenho com o conjunto de dados KITTI (GEIGER; LENZ; URTASUN, 2012) indica que a solução proposta neste trabalho supera os resultados de outros trabalhos baseados no conceito de *Resection-Intersection* anteriormente publicados (PEREIRA *et al.*, 2018).

1.5 Notação

Esta seção apresenta a notação usada ao longo deste trabalho. A notação adotada é a similar à adotada por Forster et al. (FORSTER; PIZZOLI; SCARAMUZZA, 2014; RADKE, 2012). Os escalares são ilustrados no formato itálico *(s, C)*, os vetores são em negrito (**v**) e salvo indicação em contrário, são colunas. Matrizes são em negrito e em letras maiúsculas (**M**). A localização da referência de uma *feature* na k-ésima imagem é denotada por $\mathbf{u}_k = (u_k, v_k)^T$, e o conjunto de todas as origens das *feature* na imagem k é agrupado na matriz $n \times 2$ ($\mathbf{U}_k$) onde cada linha armazena um par de coordenadas. A matriz **P** representa um conjunto de pontos no espaço 3D, onde cada linha representa um ponto $\mathbf{p} = (x, y, z)$ que pode ser mapeado para uma coordenada de imagem pelo modelo de projeção da câmera $\hat{\mathbf{u}} = (\hat{u}, \hat{v})^T$.

O símbolo $\prime$ (linha) é usado para denotar as coordenadas de imagem que foram mapeadas no espaço de coordenadas da câmera ou dimensionamento em relação ao comprimento focal da câmera, em coordenadas homogêneas: $\mathbf{u}' = \mathbf{K}^{-1}\mathbf{u}$ onde **K** representa os parâmetros intrínsecos da câmera. A pose da câmera é representada pelo vetor $\mathbf{w} = (\theta_x, \theta_y, \theta_z, t_x, t_y, t_z)$ contendo as três rotações e translações em relação à pose anterior da câmera ou por uma matriz 3×4, caracterizando a transformação do corpo rígido (também chamada de matriz fundamental) $\mathbf{T} = [\mathbf{R} \mid \mathbf{t}]$. No contexto da presente tese, o conceito de tempo real é compreendido como a execução do algoritmo proposto dentro do tempo de amostragem da sequência de imagens de entrada.

1.6 Organização

Este documento está organizado da seguinte maneira: o Capítulo 2 apresenta os fundamentos básicos e algoritmos clássicos relacionados à estimativa de posição 3D a partir de imagens. O Capítulo 3 traz uma pesquisa sobre desenvolvimentos recentes na área de Odometria Visual.

Os novos sistema OVM e os métodos de estimação da posição da câmera propostos neste trabalho são computados nos Capítulos 4 e 5. Os resultados obtidos são apresen-

tados e discutidos no Capítulo 6. Finalmente, uma conclusão e uma menção sobre os trabalhos futuros são feitos no Capítulo 7.

2 FUNDAMENTOS DE ODOMETRIA VISUAL

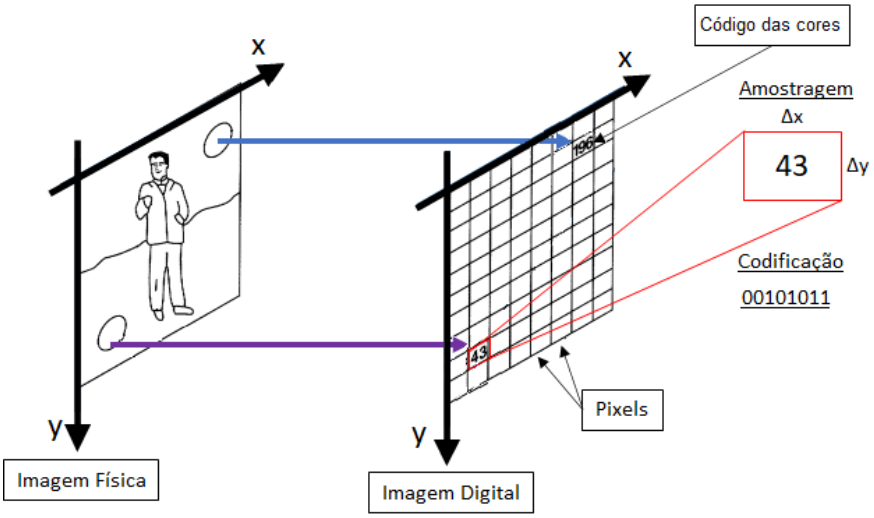
Este Capítulo descreve os conceitos e algoritmos fundamentais relacionados à Odometria Visual. O Capítulo começa com uma revisão dos conceitos básicos de Visão Computacional e dos métodos de processamento de imagem mais relevantes utilizados ao longo do trabalho. O Capítulo continua descrevendo o modelo básico que explica como uma estrutura de cena 3D é projetada em um plano de imagem 2D, seguido pela definição de erro de reprojeção e técnicas de detecção e rastreamento de *features*. As restrições de projeção da geometria epipolar, triangulação, métodos de otimização numérica utilizados e as principais ideias sobre a estimativa da posição da câmera também são abordados. Na sequência, os conceitos dos métodos *Bundle Adjustment* e *Resection-Intersection* são apresentados. Por fim, a típica estruturação de operação OVM esparsa é discutida e o problema de ambiguidade da similaridade da cena é contextualizado.

2.1 Representação Digital de uma Imagem

Uma imagem pode ser definida como uma função bidimensional de intensidade luminosa $f(x, y)$, onde x e y são as coordenadas espaciais de linha e coluna, e o valor de f representa a informação de cor ou intensidade luminosa de uma imagem. O termo imagem digital refere-se a uma imagem que pode ser discretizada quanto às suas coordenadas espaciais e quanto à intensidade luminosa de cada ponto (pixel).

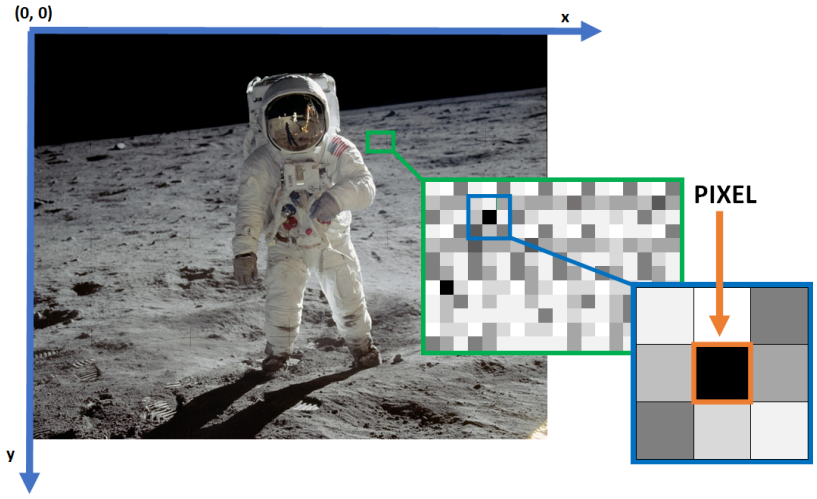
Como os computadores não manipulam dados analógicos, é necessário converter uma imagem contínua em sua forma digital, ou seja, a função $f(x, y)$ precisa ser convertida para uma descrição discreta, assim a imagem digital é obtida por um processo de discretização, envolvendo dois passos, a amostragem e a quantização, conforme ilustra a Figura 3. A amostragem consiste em discretizar o domínio de definição da imagem em um valor de intensidade de cor dentro de uma escala conhecida, geralmente entre 0 a 255. Nas direções dos eixos y e x da imagem, gerando uma matriz de $M \times N$ amostras, onde M representa o eixo vertical e N o eixo horizontal da imagem.

Figura 3 – Ilustração do conceito de discretização de imagens por câmeras digitais.



Fonte: Adaptado de (CASTLEMAN, 1996).

Figura 4 – Ilustração do conceito de pixel e da convenção do sistema de coordenadas para representação de imagens digitais



Fonte: Adaptado de (GALILEU, 2021).

Os elementos desta matriz digital são chamados de elementos da figura ou de píxeis (*picture elements*) (GONZALEZ; WOODS, 2006). O conceito de pixel e a convenção espacial utilizados neste trabalho estão ilustrados na Figura 4, onde a origem da imagem está localizada no canto superior esquerdo da mesma. Segundo Pedrini (2007), uma imagem colorida é uma imagem multi-espectral, onde a cor em cada ponto (x,y) é definida por meio de três grandezas: luminância, matiz e saturação. A luminância está associada

com a intensidade luminosa, o matiz com o comprimento de onda dominante e a saturação com o grau de pureza do matiz. De acordo com Castleman (CASTLEMAN, 1996), para algumas aplicações pode ser mais conveniente tratar a imagem colorida como uma função bidimensional que possui duas coordenadas espaciais e uma de cor. Em algumas aplicações a informação de cor pode não ser necessária para realizar determinadas tarefas, como extrair informações de contorno e estruturas na imagem. Neste caso, utilizamos a imagem na escala de cinza, que é uma representação de uma imagem colorida em uma única componente, regida por uma função bidimensional que possui duas coordenadas espaciais e uma de intensidade. Essa escala usualmente é definida com valores de 0 até 255, e representa 255 tons de cinza. É usual utilizar 8 bits para representar cada um dos componentes de cor da imagem a cores e também para o componente único da imagem em tons de cinza. Há padrões para aplicações que demandam maior resolução que utilizam 10 bits para representar os componentes de uma imagem. A conversão para escala de cinza de uma imagem no formato RGB pode ser obtida através de várias equações. A Equação (1), descrita por Hoffmann (2001), descreve uma delas.

$$Y = 0,2126R + 0,7152G + 0,0722B. \tag{1}$$

Onde R é o componente vermelho da imagem, G é o componente verde, B é o componente azul e Y é a intensidade na escala de cinza.

2.2 Aquisição das Imagens

De acordo com Gonzalez (GONZALEZ; WOODS, 2006), o processo de aquisição de imagens digitais consiste em converter cenas tridimensionais em imagens digitais, por meio de um dispositivo ou sensor, como por exemplo, uma câmera de vídeo, e a conversão em uma representação adequada para o processamento digital subsequente. Estes dispositivos devem ser escolhidos levando em consideração o tipo de sensor, resolução e o número de níveis de cinza ou cores da imagem digitalizada, taxa de amostragem e codificação.

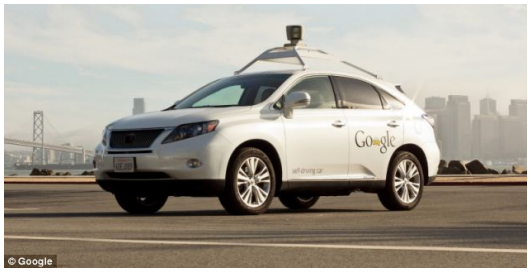
O processo de aquisição de uma cena tridimensional em uma imagem eletrônica é um processo óptico de redução de dimensionalidade, ou seja, uma câmera digital converterá a cena 3D do ambiente em uma representação 2D eliminado a informação da profundidade de cada ponto adquirido. Assim como as câmeras convencionais, as digitais possuem uma variedade de lentes, que servem para direcionar a luz do ambiente para o sensor. Porém, ao invés de utilizar um filme fotográfico, é utilizado um sensor semicondutor, que registra eletricamente a luz incidente sobre as lentes.

O sensor utilizado para fazer a captura da imagem é um dispositivo capaz de converter a energia luminosa de um ponto da imagem em carga elétrica, processando-a através de um circuito eletrônico especifico, formando, assim, a imagem digital. Hoje em dia, o

dispositivo de conversão mais utilizado usa a tecnologia CMOS (*Complementary Metal Oxide Semiconductor*).

No processo de aquisição de imagens a partir de veículos a posição desse dispositivo deve ser escolhida com cuidado, para se obter uma vista vantajosa da pista e permitir a detecção de sinalização horizontal de forma mais eficiente. As posições mais proveitosas são na parte superior-frontal do veículo e na parte frontal-interna, perto da posição do retrovisor do veículo, como ilustrado nas Figuras 5 (a) e (b).

Figura 5 – Ilustração do processo de aquisição de imagens digitais



(a) Ilustração da posição da câmera na parte externa do veículo.

(b) Ilustração da posição da câmera na parte interna do veículo.

Fonte: (THOME, 2004) e (INAVI, 2020)

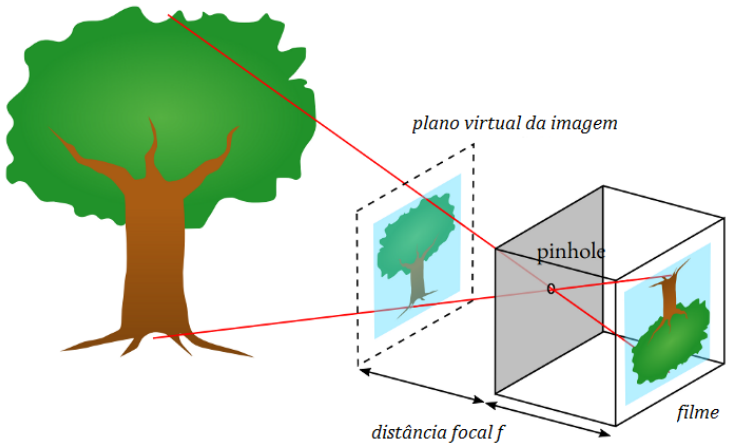
Os veículos autônomos geralmente utilizam câmeras com visão estéreo para fazer a captura da imagem da pista e do cenário no qual o veículo está trafegando, e ao mesmo tempo, para capturar as informações de profundidade dos objetos presentes no caminho do veículo no ambiente. Esse tipo de aquisição permite que seja levada em consideração a posição espacial relativa dos objetos presentes no pista na hora de se determinar a trajetória do veículo na pista. Câmeras com visão estéreo (ou câmeras estereoscópicas) é um tipo de câmera com dois ou mais sensores óticos, geralmente espaçadas igualmente e que realizam a captura da imagem individualmente. Permitindo que a combinação das imagens de cada câmera seja utilizada para para se obter uma visão espacial do ambiente.

2.3 Modelo de Projeção e Sistema de Coordenadas

Pinhole é um processo rudimentar de se fazer fotografia sem a necessidade do uso de equipamentos convencionais. É uma câmera artesanal que pode ser construída facilmente utilizando-se materiais simples e de poucos elementos. Também conhecida como câmera estenopeica, a Pinhole é basicamente uma câmara escura com um pequeno orifício que projeta uma imagem invertida em um plano, como mostrado na Figura 6.

O processo de formação de uma imagem advém dos parâmetros intrínsecos da câmera, que por sua vez, são diretamente dependentes da forma de montagem da câmera, do

Figura 6 – Ilustração do funcionamento e modelo de uma câmera Pinhole



Fonte: Adaptado de (HOÀNG-ÂN, 2018).

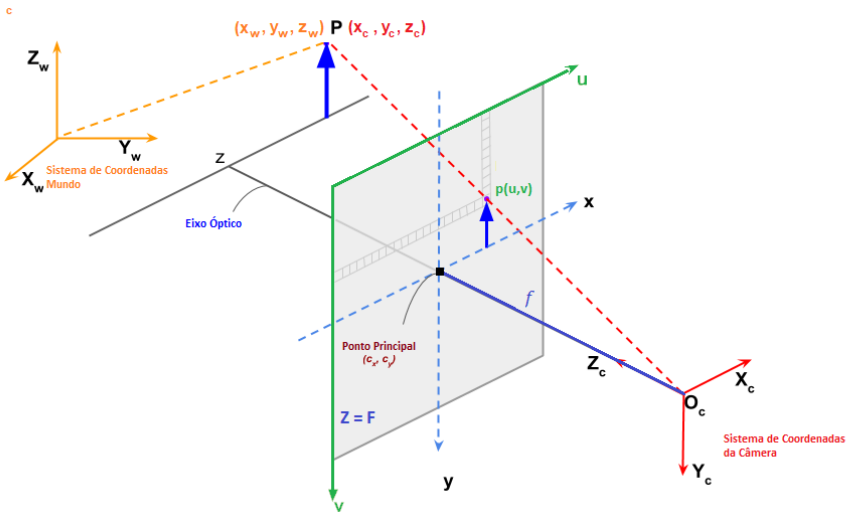
tipo de lente e do sensor utilizado para amostrar o sinal. Os parâmetros intrínsecos são considerados dados de entrada pré-definidos e são representados pela matriz **K**, e descrita pela Equação 2 (HARTLEY; ZISSERMAN, 2004a).

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2}$$

A matriz **K** é composta pelo ponto focal $f = (f_x, f_y)$ e pelo ponto central da imagem $c = (c_x, c_y)$, ambos representados em relação ao eixo x e ao eixo y do sistema de coordenadas da câmera. Os parâmetros intrínsecos e a distorção da câmera podem ser estimados durante o processo de calibração da câmera (WÖHLER, 2013). O fator de distorção da lente também deve ser levado em consideração, pois pode causar resultados indesejados nas coordenadas dos pontos da cena projetados no plano da imagem. Nesse trabalho, as imagens são utilizadas sempre após o processo de compensação da distorção.

Através da definição de um modelo de câmera, é possível definir o sistema de coordenadas relativo à câmera. O sistema de coordenadas mundo utilizado neste trabalho está ilustrado na Figura 7. Na figura, um plano virtual da imagem, em frente ao ponto principal é ilustrado e uma projeção não invertida é formada. Nesse sistema é mostrado um ponto $P(x, y, z)$ no espaço de coordenadas mundo e sua projetação correspondente no sistema de coordenadas da imagem $p(u, v)$, localizado no plano perpendicular ao eixo Z e distante f da origem (O_c). Os eixos X e Y estão alinhados ao plano perpendicular da imagem ao eixo Z. Os objetos são projetados no plano da imagem descartando a informação de distância no eixo Z, implicando que pontos que estão alinhados no espaço entre si e o centro óptico são projetados no plano da imagem em um único ponto.

Figura 7 – Ilustração do sistema de coordenadas utilizado neste trabalho.



Fonte: Adaptado de (SADEKAR, 2019).

Para obter-se as coordenadas no plano da imagem de um ponto no espaço 3D, primeiramente se converte o ponto para o sistema de coordenadas da câmera, centralizado no ponto focal da lente. A conversão do espaço 3D para o sistema de coordenadas do plano da imagem pode ser simplificado conforme mostra a Equação 3 (HARTLEY; ZISSERMAN, 2004a).

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}. \tag{3}$$

Onde $(u, v, 1)^T$ representa as coordenadas de um pixel na imagem, (f_x, f_y) é a distância focal da câmera dada em píxeis, λ é o fator de escala, (c_x, c_y) são as coordenadas do centro da câmera também definidas em píxeis e o vetor $(x, y, z)^T$ é a coordenada de ponto P dado em metros. Portanto, um ponto no espaço de coordenadas mundo $P(x, y, z)^T$ pode ser projetado diretamente no plano da imagem $(u, v)^T$. A Equação 3 pode ser reescrita na forma das Equações 4 e 5 para se obter as coordenadas u e v diretamente.

$$u = \frac{x}{y} f_x + c_x. \tag{4}$$

$$v = \frac{y}{z} f_y + c_y. \tag{5}$$

O conjunto de parâmetros (f_x, f_y, c_x, c_y) define as principais características do modelo da câmera Pinhole e são chamados de parâmetros intrínsecos da câmera. O ponto central é definido como $u = v = 0$.

2.4 Conversão de Sistemas de Coordenadas

Considerando um sistema de coordenadas arbitrário **W**, a transformação de um ponto representado nesse sistema $\mathbf{p} = [x, y, z]^T$ para outro sistema de coordenadas **C** é representado por uma matriz de transformação **M**. Essa matriz é composta por uma parte rotacional **R**, que corresponde a uma matriz 3×3 ortonormal, determinada por três parâmetros independentes e uma parte translacional **t** representando a diferença entre os sistemas de coordenadas. De forma análoga, a conversão de um ponto do sistema de coordenadas da câmera para um sistema de coordenadas mundo pode ser feito através da inversa da matriz **M**.

Quando a posição dos pontos usa uma referência de coordenada diferente dos da câmera (por exemplo sistema de coordenadas mundo), as coordenadas desses pontos devem ser convertidas para o sistema de coordenadas da câmera, antes de serem projetadas no plano da imagem. A Equação 6 descreve como essa conversão pode ser alcançada.

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}. \tag{6}$$

Substituindo a matriz com os parâmetros intrínsecos da câmera, **K**, a matriz de rotação por **R** e o vetor de translação por **t**, e usando coordenadas homogêneas. A notação matricial da Equação 6 está descrita pela Equação 7.

$$\mathbf{u} = \mathbf{K}\,[\mathbf{R}|\,\mathbf{t}]\mathbf{p}. \tag{7}$$

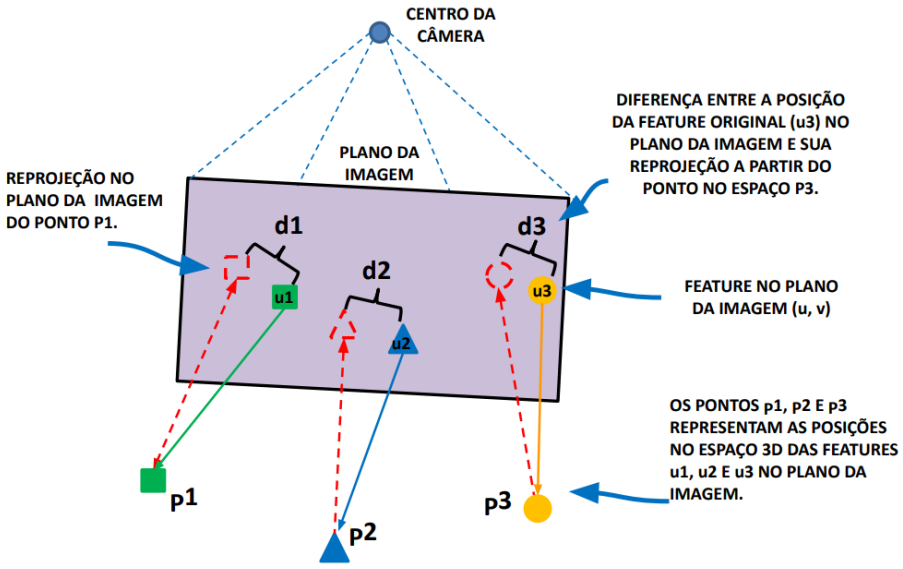
A rotação e a translação são chamadas parâmetros extrínsecos da câmera ou pose da câmera.

2.5 Erro de Reprojeção

O modelo de projeção e a definição do problema de reconstrução 3D é largamente discutido no trabalho de Triggs et al. (TRIGGS, 1996). Para definir o erro de reprojeção de um ponto (ou o erro de predição), Triggs considera um modelo simples de cena, contendo pontos 3D estáticos e individuais capturados por câmeras. Nesse modelo, esses pontos são medidos com ruído em uma imagem que os contenha. Dessa forma, um modelo preditivo dessa medida pode ser estabelecido.

Dessa forma, é possível definir que essa medida obedece a um modelo preditivo que pode ser estabelecido.

Figura 8 – Exemplo de erro de reprojeção.



Fonte: do autor.

Na Figura 8, temos três *features* ilustradas na forma de um quadrado, um triangulo e um círculo no plano da imagem. Os vetores **u1**, **u2** e **u3** representam a localização da referência dessas *features* no plano da imagem. Os pontos **p1**, **p2** e **p3** representam a localização dessas três *features* no espaço tridimensional localizados por um método de triangulação. Contudo, devido a fontes de incerteza no processo de triangulação os pontos estimados no espaço 3D não são precisos. Devido a essa imprecisão, quando os pontos forem reprojetados na imagem, a sua posição acaba sendo diferente da sua posição original no plano da imagem (indicado na figura pelas formas em vermelho tracejado). Dessa forma, as distâncias $d1$, $d2$ e $d3$ representam a diferença entre a posição reprojetada e a posição original no plano da imagem. Essa diferença é chamada de erro de reprojeção e representado matematicamente pela Equação 8 (TRIGGS *et al.*, 2000).

$$\begin{bmatrix} \delta_u \\ \delta_v \end{bmatrix} = \begin{bmatrix} \hat{u} \\ \hat{v} \end{bmatrix} - \begin{bmatrix} u \\ v \end{bmatrix}. \tag{8}$$

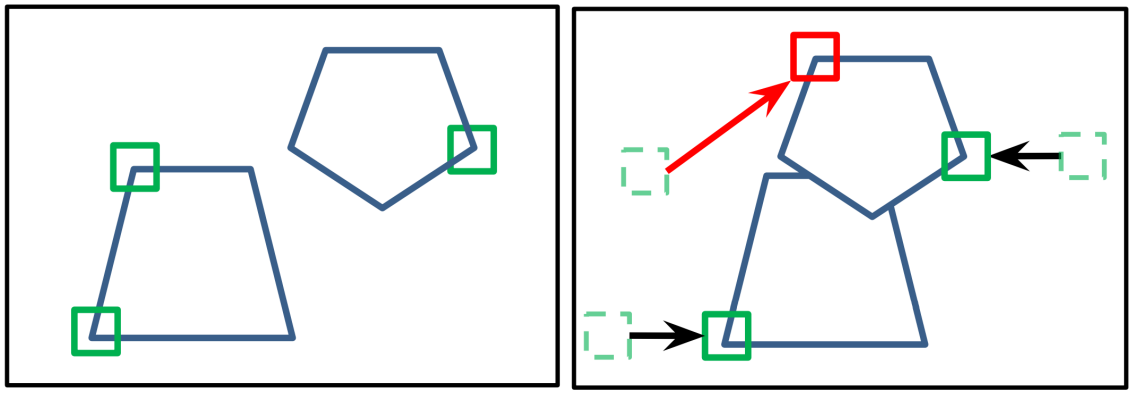
Essa Equação pode ser utilizada para medir o erro entre o valor estimado de um ponto na imagem e o efetivamente medido. Dessa forma, o processo de minimização do erro de reprojeção da pose visa encontrar uma transformação $\mathbf{T}_{(k,w)}$ que minimize a diferença entre as coordenadas dos pontos extraídos da imagem (*features*) u_i e a projeção calculada no plano da imagem, $\tilde{u_i}$, conforme mostrado na Equação 9.

$$\mathbf{T}_{k,w} = \underset{\mathbf{T}_{k,w}}{\operatorname{argmin}} \sum_{i \epsilon n}^{n-1} \left\| u_i - \tilde{u}_i \right\|^2. \tag{9}$$

2.6 Detecção e Rastreamento de Features

A Odometria Visual esparsa é baseada na detecção e rastreamento de *features* em uma sequência de imagens; isto é, gerar padrões de informação a partir da imagem para encontrar regiões com características singulares, que possibilitem a sua localização em outras imagens, e que posteriormente, são utilizadas para estimação da pose da câmera. Ao longo deste trabalho, essas regiões são chamados de *features* e suas respectivas projeções no espaço 3D são chamadas de pontos. A detecção e o rastreamento de *features* são referidos como o processo de selecionar regiões de imagem rastreáveis e localizar essas regiões em imagens subsequentes.

Figura 9 – Detecção e Rastreamento de *features*



(a) Exemplo de detecção de *features* (b) Rastreamento e movimentos de *features*

Fonte: Adaptado de (PEREIRA, 2018).

O processo de rastreamento de *features* refere-se a encontrar a correspondência de uma certa característica (*features*), descrita em pares de coordenadas de píxeis, em uma outra imagem. Na Figura 9 (a), os quadrados verdes marcam três *features* detectadas. As *features* correspondentes encontradas na Figura 9 (b) são indicadas pelos quadrados verdes, e o quadrado vermelho marca uma falha correspondente devido à oclusão do topo do trapézio resultante do movimento realizado pelas estruturas. As setas representam vetores de movimento, conectando a posição das *features* na imagem original até a sua nova posição rastreada.

Ao longo dos últimos anos, diversos algoritmos para a detecção e rastreamento de *features* vêm sendo utilizados em trabalhos relacionados à odometria, como o algoritmo detector de cantos de Harris (KITT; GEIGER; LATEGAHN, 2010), ou ainda em algoritmos mais recentes como SIFT (TARDIF; PAVLIDIS; DANIILIDIS, 2008), SURF (LEE; FAUNDORFER; POLLEFEYS, 2013), BRIEF (PERSSON *et al.*, 2015), SURF (DESAI; LEE, 2016) e ORB (GOMEZ-OJEDA; GONZALEZ-JIMENEZ, 2016), entre ou-

tros. Dada essa ampla gama de técnicas de detecção de *features* disponíveis, selecionar uma técnica que se adapte à velocidade e exigência de precisão não é uma tarefa trivial e uma avaliação prática é necessária.

2.6.1 Detector de Canto Harris

Harris e Stephens (HARRIS; STEPHENS, 1988) apresentaram um trabalho apoiado nas ideias de Moravec (MORAVEC, 1980), que tinha o objetivo de detectar cantos e arestas em imagens, com base na similaridade de uma região de imagem com regiões sobrepostas em vizinhança horizontal, vertical e diagonal e que usava gradientes de imagem em vez de blocos de imagem deslocados. Neste trabalho, os autores definiram a matriz de Harris, **H**, como mostrado na Equação 10.

$$\mathbf{H}_a = \begin{bmatrix} \sum_{(u,v)\epsilon\omega}(\frac{\partial I}{\partial u}(u,v))^2 & \sum_{(u,v)\epsilon\omega}\frac{\partial I}{\partial u}(u,v)\frac{\partial I}{\partial v}(u,v) \\ \sum_{(u,v)\epsilon\omega}\frac{\partial I}{\partial u}(u,v)\frac{\partial I}{\partial v}(u,v) & \sum_{(u,v)\epsilon\omega}(\frac{\partial I}{\partial v}(u,v))^2 \end{bmatrix}. \tag{10}$$

Onde $I(u,v)$ é a intensidade do pixel nas coordenadas (u,v), $\frac{\partial I}{\partial u}$ é a derivada parcial da intensidade (gradiente de imagem) ao longo da direção u e sobre a região de imagem atualmente testada ω. Como base na Equação 10, os autores propuseram um ajuste de qualidade aproximada baseada no traço (tr) da matriz $(\mathbf{H}_a)$ como mostrado na Equação 11, onde κ é um parâmetro de sensibilidade ajustável:

$$Q_l = det(\mathbf{H}_a) - \kappa \cdot tr(\mathbf{H}_a)^2. \tag{11}$$

Através dos autovalores (λ_1, λ_2) extraídos da matriz $\mathbf{H}_a$, pode-se obter uma melhor aproximação da qualidade de um canto ou aresta presente em uma imagem. Se as intensidades dos píxeis da região de análise forem semelhantes (área de imagem homogênea), tanto $\frac{\partial I}{\partial u}$ quanto $\frac{\partial I}{\partial v}$ apresentarão valores pequenos. Porém, na situação em que a região de análise da imagem apresentar uma borda linear e na direção de um dos eixos, um dos autovalores apresentará um valor grande com o autovetor associado perpendicular à borda, enquanto o outro será próximo de zero. Os autores J. Shi e C. (SHI; TOMASI, 1994) propuseram uma pequena modificação no trabalho de Harris e Stephens, descrita pela Equação 12.

$$Q_l = min(\lambda_1, \lambda_2). \tag{12}$$

Nesse trabalho, os autores averiguaram que, quando um canto estiver presente na região de análise da imagem, tanto λ_1 como λ_2 apresentarão valores diferentes de zero e através da análise do menor autovalor da matriz de Harris, a qualidade do canto pode ser definida.

Todos os cantos presentes na imagem são analisados e o algoritmo somente retorna as regiões da imagem que apresentarem uma medida de qualidade acima de um limiar,

que é definido como uma fração da qualidade máxima encontrada na imagem. Para evitar que vários resultados sejam detectados em uma mesma região, geralmente é aplicada a supressão não máxima, em que apenas o maior valor em um raio definido pelo usuário é considerado.

2.6.2 Rastreador Lucas-Kanade

O método de Lucas-Kanade (LUCAS; KANADE, 1981) é um método de pesquisa utilizado para localizar uma região presente em uma imagem em outras imagens semelhantes. Considerando uma série de imagens registradas por uma câmera e indexadas no tempo por $I(u, v, t)$, é possível aproximar um bloco ω de píxeis no tempo $t+1$ pelo bloco no tempo t, conforme descreve a Equação 13.

$$I(x + u, y + v, t + 1) \approx I(u, v, t) \forall (u, v) \epsilon \omega. \tag{13}$$

Aplicando a série de Taylor de primeira ordem, é possível ir da Equação 13 para a Equação 14 (LUCAS; KANADE, 1981).

$$\mathbf{H}_a \begin{bmatrix} x \\ y \end{bmatrix} = - \begin{bmatrix} \sum_{(u,v)\epsilon\omega} (\frac{\partial I}{\partial u}(u, v, t) \frac{\partial I}{\partial t}(u, v, t)) \\ \sum_{(u,v)\epsilon\omega} (\frac{\partial I}{\partial v}(u, v, t) \frac{\partial I}{\partial t}(u, v, t)) \end{bmatrix}. \tag{14}$$

Na Equação acima, $\mathbf{H}_a$ é chamada de matriz de Harris. Multiplicando ambos os lados da Equação 14 pelo inverso da matriz de Harris, o vetor de deslocamento pode ser aproximado iterativamente. Como o algoritmo só funciona para pequenos deslocamentos onde as expansões da série de Taylor são significativas, uma implementação piramidal do algoritmo é geralmente usada como proposto por Bouguet (YVES BOUGUET, 2000).

Nessa abordagem, são geradas pirâmides de imagem com diferentes resoluções, e as *features* são localizadas na imagem de menor resolução. As posições encontradas são subsequentemente refinadas em imagens de resolução mais alta até que a imagem de resolução total original seja processada.

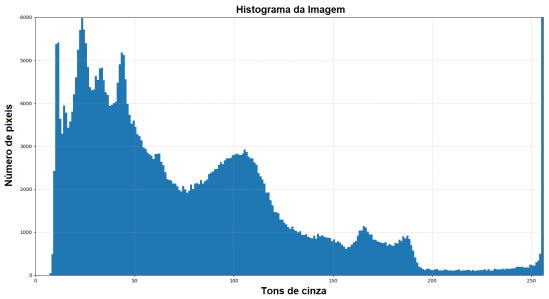
2.7 Equalização de Histograma

Conforme Gonzales (GONZALEZ; WOODS, 2006), o histograma é uma das formas mais comuns de se representar a distribuição dos tons de cinza de uma imagem, e possivelmente o procedimento mais comum em processamento digital de imagens. O histograma de uma imagem descreve a distribuição dos níveis de cinza em termos do número de amostras (píxeis) com cada nível. A distribuição pode também ser dada em termos da percentagem do número total de píxeis na imagem. Pode ser estabelecida uma analogia entre o histograma de uma imagem e a função densidade de probabilidade, que é um

modelo matemático da distribuição de tons de cinza de uma classe de imagens.

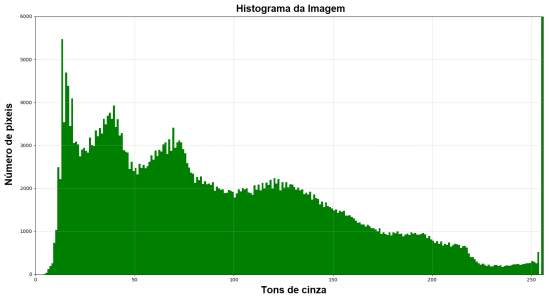
Figura 10 – Ilustração do efeito da Equalização de Histograma.

(a) Ilustração da imagem original.



(b) Ilustração do histograma da imagem original.

(c) Ilustração da imagem ajustada.



(d) Ilustração do histograma equalizado da imagem.

Fonte: Adaptado de (GEIGER ANDREAS LENZ PHILIP, 2020).

A técnica de Equalização de Histograma tem por objetivo melhorar o balanço de iluminação das imagens. É normalmente utilizada como uma etapa de pré-processamento para sistemas de reconhecimento de padrões. O contraste entre dois objetos pode ser definido como a razão entre os seus níveis de cinza médios. A manipulação do contraste consiste numa transferência radiométrica em cada pixel, com o objetivo de aumentar a discriminação visual entre os objetos presentes na imagem. Realiza-se a operação ponto a ponto, independentemente da vizinhança (GONZALEZ; WOODS, 2006).

Contudo, a equalização global do histograma não gera bons resultados em ambientes de alta complexidade luminosa. Por sua vez, a Equalização Adaptativa do Histograma (EAH), é uma técnica usada para melhorar contraste em imagens em ambientes de iluminação complexa. Ao contrário da Equalização Global, o EAH divide a imagem em N partes e calcula o histograma para cada uma delas, equalizando os valores de luminosidade da imagem. No entanto, o EAH tende a super-amplificar o ruído em regiões

relativamente homogêneas de uma imagem.

Neste trabalho, o algoritmo CLAHE (PIZER *et al.*, 1987) foi utilizado para realizar a equalização adaptativa do histograma. Equalização de histograma adaptativo limitada por contraste (CLAHE) é uma variante da equalização adaptativa do histograma (EAH) que possui uma maior flexibilidade na escolha da função de mapeamento de histograma local, através da escolha de um valor de limiar para reduzir os ruídos indesejados.

O resultado é uma imagem com alto contraste, e com detalhes mais visíveis, conforme ilustra a Figura 10. Na figura (a) temos uma imagem retirada do banco de dados KITTI e seu respectivo histograma em (b). Em (c) temos a imagem (a) após ser equalizada pelo algoritmo CLAHE e seu respectivo histograma em (d). Por inspeção, é possível verificar o efeito da equalização do histograma ao se observar a redistribuição mais homogênea do histograma em (d) em relação ao mostrado em (b).

A função CLAHE disponível na biblioteca OpenCV (BRADSKI, 2000) é a utilizada no sistema proposto neste trabalho. Os parâmetros de configuração aplicados foram 1.5 como valor de limiar para limitação de contraste e 8 como tamanho da grade para equalização do histograma.

2.8 Geometria Epipolar

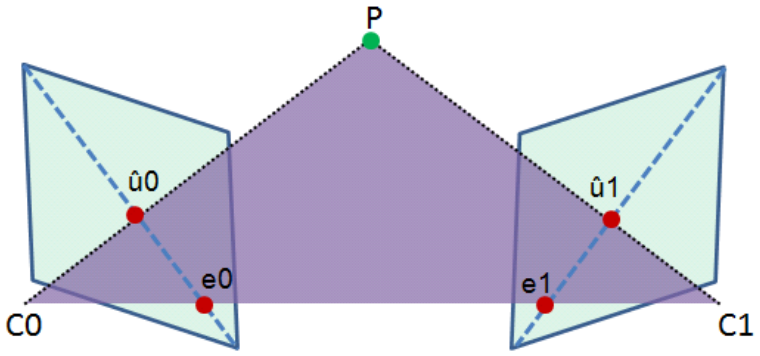
Esta seção apresenta as restrições de projeção de linhas epipolares para uma cena vista de duas perspectivas diferentes. Na visão estereoscópica, duas câmeras são usadas, geralmente lado a lado, para adquirir simultaneamente o que é chamado de par de imagem estereoscópicas. A diferença geométrica das configurações estereoscópicas para monocular é que, no caso estereoscópicas, os parâmetros intrínsecos da câmera (distância focal, centro da câmera e coeficientes de distorção) podem ser diferentes, mas a posição relativa entre as câmeras é geralmente conhecida.

No caso da monocular, os parâmetros intrínsecos da câmera são os mesmos, mas a posição relativa da câmera deve ser estimada para cada imagem. No entanto, os parâmetros extrínsecos (posição e orientação da câmera) têm propriedades equivalentes e os tópicos abordados nesta seção são igualmente válidos para ambas as situações (PEREIRA, 2018).

Conforme ilustra a Figura 11, considere-se um ponto no espaço tridimensional **P** visto por duas câmeras com parâmetros intrínsecos conhecidos. O ponto e os dois centros das câmeras formam um plano no espaço, e a intersecção desse plano com os planos da imagem formam uma linha em cada imagem, essas linhas são chamadas de linhas epipolares (e).

Os centros das câmeras são definidos por **C0** e **C1**, as projeções dos pontos nos planos das imagens são descritos por $\hat{\mathbf{u}}_0$ e $\hat{\mathbf{u}}_1$, os epipólos por $\mathbf{e}_0$ e $\mathbf{e}_1$ e as linhas azuis tracejadas são as linhas epipolares. Uma vez que a posição relativa das câmeras é conhecida, a

Figura 11 – Geometria Epipolar



Fonte: Adaptado de (PEREIRA, 2018).

projeção de ponto em uma imagem ($\hat{\mathbf{u}}_0$ por exemplo), restringe a região da imagem onde a projeção de ponto na outra imagem ($\hat{\mathbf{u}}_1$) pode ser localizada na linha epipolar. A posição ao longo da LE, onde a segunda projeção está localizada, define a distância do ponto ($\hat{\mathbf{u}}_1$) ao centro da câmera (PEREIRA, 2018).

Figura 12 – Exemplo de linhas epipolares.

(a) (b)

Fonte: Adaptado de (RADKE, 2012).

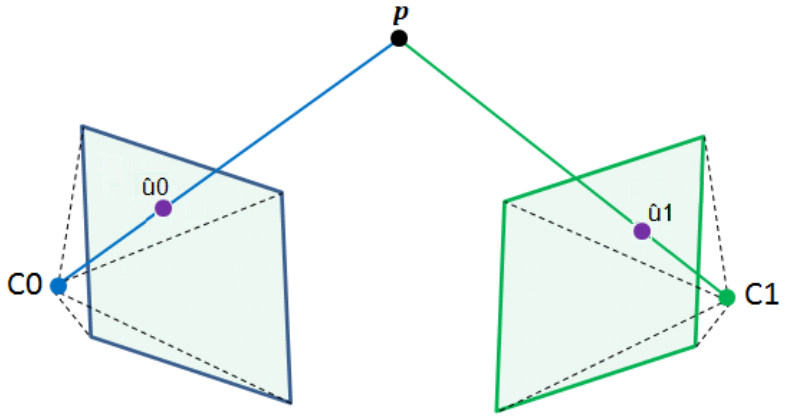
Um exemplo real é demonstrado na Figura 12, onde um templo é fotografado de diferentes perspectivas. As linhas vermelhas são linhas epipolares (PEREIRA, 2018). É possível notar que a linha superior da figura da esquerda cruza o telhado superior e intercepta uma ponta do segundo teto. Os mesmos pontos também estão localizados na linha superior da imagem à direita (PEREIRA, 2018).

O mesmo acontece com todas as linhas em ambas as imagens. Qualquer ponto da imagem em uma linha particular, será encontrado ao longo da linha correspondente na outra imagem (desde que seja visível em ambas as imagens).

2.9 Triangulação

Triangulação é um algoritmo utilizado para estimar a posição tridimensional de um ponto a partir de sua projeção em duas ou mais imagens. Para realizar o processo de triangulação é necessário conhecer os parâmetros intrínsecos da câmera e sua posição e orientação relativas. Na ausência de erros, a triangulação exata pode ser calculada diretamente encontrando-se a interseção das linhas que conectam os centros da câmera à posição da *feature* na imagem, conforme representa a Figura 13.

Figura 13 – Exemplo de situação de triangulação sem erros.

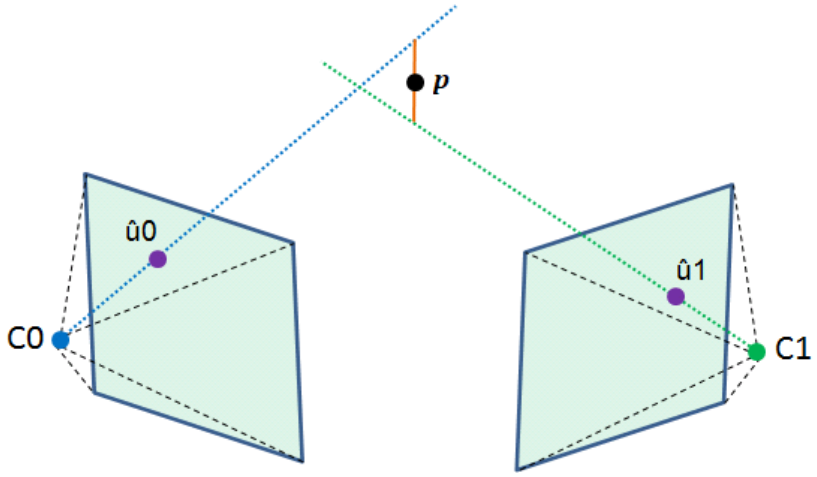


Fonte: Adaptado de (PEREIRA, 2018).

No entanto, na prática, o cálculo da triangulação de dois pontos estará sujeito a fontes de imprecisão geradas pela limitação de resolução da imagem e também diretamente relacionado ao dinamismo do cenário de aplicação, que pode gerar diversas fontes de erro tais como: oclusões, falsos positivos no rastreamento, objetos em movimento na cena e desfoque de movimento.

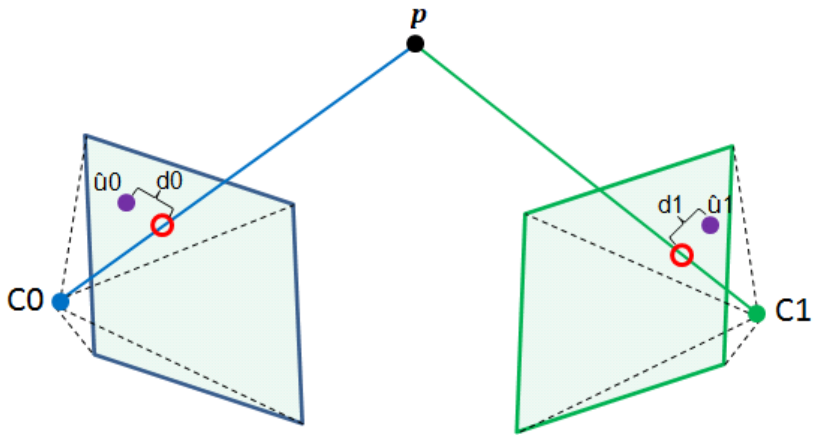
Conforme demonstra a Figura 14, na presença de erros, as linhas que conectam os centros da câmera à posição da *feature* não se cruzam mais no espaço. Nesta situação, algoritmos de triangulação inexatos devem ser usados. Nesse tipo de abordagem, a distância perpendicular entre as linhas que não se cruzam no espaço devem ser encontradas para que as posições dos pontos possam ser ajustada. No trabalho de Hartley e Sturm (HARTLEY; STURM, 1997), foi proposto um método teórico que encontra o ponto tridimensional que minimiza a distância do ponto **p** projetado nas imagens e a posição original da *feature* em ambas as imagens. Por exemplo, a Figura 15 ilustra um cenário onde as distâncias $d0$ e $d1$ (medidas em píxeis) foram minimamente reduzidas, através da solução de um polinômio de sexto grau, que encontra a média harmônica (com o peso inversamente proporcional à distância do epipolo) do segmento de linha perpendicular (ilustrado na Figura 14).

Figura 14 – Segmento de interseção perpendicular de triangulação.



Fonte: Adaptado de (PEREIRA, 2018).

Figura 15 – Triangulação ideal.



Fonte: Adaptado de (PEREIRA, 2018).

Neste trabalho, um método de triangulação rápida baseado na minimização da distância euclidiana do erro de projeção utilizado foi o proposto em (PEREIRA *et al.*, 2018).

2.10 Otimização Numérica de Gauss-Newton

No contexto científico, a otimização é a seleção da melhor alternativa em relação a um critério especificado e representa um campo de grande importância na pesquisa de tomada de decisão e na análise de sistemas físicos, econômicos, sociais, etc. Esse critério

estipulado deve ser definido objetivamente como uma medida do desempenho do sistema (como tempo, lucro, custo, etc.) e depende de variáveis do sistema.

A otimização de sistema é o processo de identificação da relação entre a função objetivo e as variáveis do sistema. Uma vez definido o modelo, o algoritmo de otimização visa ajustar essas variáveis para otimizar o objetivo. Nos problemas óptico-geométricos estudados neste trabalho, os dados disponíveis são a posição da *feature* rastreada e de sua projeção tridimensional calculada, a partir do modelo da câmera. O objetivo da otimização minimizar a distância entre as coordenadas rastreadas e projetadas, de forma a estimar a Matriz Fundamental da câmera (GONZALEZ; WOODS, 2006) com o menor erro possível.

A Matriz de Projeção ou Pose da Câmera é uma relação entre planos de duas imagens da mesma cena que restringe onde a projeção de pontos da cena pode ocorrer em ambas as imagens. Neste trabalho ela é definida por $\mathbf{T} = [\mathbf{R}|\mathbf{t}]$ (de dimensão 3×4), podendo ser representada pelo vetor $\mathbf{w} = (\theta_x, \theta_y, \theta_z, t_x, t_y, t_z)$ obtido através da fórmula de Rodrigues (BRADSKI, 2000) que traduz a matriz fundamental em um vetor contendo apenas os três ângulos de rotação e as translações nos eixos X, Y e Z. O objetivo da função s que será minimizada para um conjunto de m características é a soma quadrática dos resíduos $\mathbf{r} = (r_0, ..., r_{2m} - 1)$, conforme mostrado a seguir:

$$s(\mathbf{w}) = \sum_{i=0}^{2m-1} r_i^2(\mathbf{w}). \tag{15}$$

Onde os resíduos são definidos como a diferença entre as coordenadas horizontal e vertical das *features* rastreadas e as projetadas:

$$r_{2n} = \hat{u}_n - u\prime_n; r_{2n+1} = \hat{v}_n - v\prime_n; |\forall n \epsilon [0, m-1]]. \tag{16}$$

O algoritmo de Gauss-Newton (GN) é uma aproximação do método de Newton, que tem a finalidade de minimizar a soma dos valores de uma função quadrática, com o diferencial de não necessitar do cálculo de derivadas de segunda ordem. O passo de atualização do algoritmo GN utilizado neste trabalho é o mesmo definido no trabalho de Silva (SILVA, 1998) e está descrito na Equação a seguir:

$$\mathbf{w}^{(s+1)} = \mathbf{w}^{(s)} - (\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\mathbf{r(w)}. \tag{17}$$

Onde $\mathbf{J}$ é a matriz Jacobiana:

$$\mathbf{J}_{(i,j)} = \frac{\partial r_i(\mathbf{w})}{\partial \mathbf{w}_j}. \tag{18}$$

O algoritmo de Gauss-Newton apresenta uma taxa de conversão rápida, porém, é mais propenso a problemas de estabilidade.

2.11 Otimização Numérica de Levenberg-Marquardt

O algoritmo de Levenberg-Marquardt (LM) (TRIGGS *et al.*, 2000) é uma versão aperfeiçoada do algoritmo GN, apresentando uma estabilidade mais alta, devido ao elemento de amortecimento $\lambda_d\mathbf{I}$, conforme descreve a Equação 19, definida no trabalho de Silva (SILVA, 1998):

$$\mathbf{b}^{(n+1)} = \mathbf{b}^{(n)} - (\mathbf{J}^T\mathbf{J} + \lambda_d\mathbf{I})^{-1}\mathbf{J}^T\mathbf{r}. \tag{19}$$

Na Equação acima, λ_d é o parâmetro de amortecimento ajustável. Valores próximos de zero aproximam LM a GN, e aumentar o valor do parâmetro muda a solução para mais perto de gradiente-descendente. O ajuste $\lambda_d\mathbf{I}$ atua como um elemento de ajuste em situações onde o Hessiano inverso aproximado $(\mathbf{J}^T\mathbf{J})^{-1}$ é mal condicionado. Nessa situação, o termo $\lambda_d\mathbf{I}$ irá atuar como um fator de amortecimento.

2.12 Otimização Numérica de Davidon-Fletcher-Powell

O método de otimização de Davidon-Fletcher-Powell (DFP) é um método classificado como Quase-Newton. Foi o primeiro método Quase-Newton a generalizar o método secante para um problema multidimensional e foi nomeado em homenagem a William C. Davidon, Roger Fletcher e Michael J. D. Powell (MAMAT *et al.*, 2018). A ideia por trás do método Quase-Newton é fazer uma aproximação iterativa da inversa da matriz Hessiana, de forma que:

$$\lim_{i\to\infty} \mathbf{H}_i = [\frac{\partial^2 r_i(\mathbf{w})}{\partial^2 \mathbf{w}_j}]^{-1} = \nabla^2 J(\mathbf{w})^{-1}. \tag{20}$$

O objetivo é construir uma aproximação da inversa da matriz Hessiana, utilizando informações de primeira ordem obtidas durante o processo iterativo de otimização. A aproximação atual $\mathbf{H}_i$ é utilizada a cada iteração para definir a próxima direção descendente do método. Idealmente, as aproximações convergem para a inversa da matriz Hessiana (BURDEN; FAIRES, 2004).

Suponha-se que o funcional de erro $J(\mathbf{w})$ tenha derivada parcial contínua até segunda ordem. Tomando dois pontos $\mathbf{w}_i$ e $\mathbf{w}_{i+1}$, defina-se $\mathbf{g}_i = -\nabla J(\mathbf{w}_i)^T$ e $\mathbf{g}_{i+1} = -\nabla J(\mathbf{w}_{i+1})^T$. Se a Hessiana, $\nabla^2 J(\mathbf{w})$, é constante, então tem-se:

$$\mathbf{q}_i = \mathbf{g}_{i+1} - \mathbf{g}_i = \nabla^2 J(\mathbf{w})\mathbf{p}_i. \tag{21}$$

$$\mathbf{p}_i = \alpha_i\mathbf{d}_i = \alpha_i\mathbf{H}_i\mathbf{g}_i. \tag{22}$$

Vê-se, então, que a avaliação do gradiente em dois pontos fornece informações sobre a matriz Hessiana $\nabla^2 J(\mathbf{w})$. Como $\mathbf{w} \in \mathbb{R}^P$, tomando P direções linearmente independentes

$\mathbf{p}_0, \mathbf{p}_1, ..., \mathbf{p}_{P-1}$, é possível determinar $\nabla^2 J(\mathbf{w})$, caso se conheça $\mathbf{q}_i, i = 0, 1, ..., P-1$. Para tanto, basta aplicar iterativamente a Equação 23 a seguir, com $\mathbf{H}_0 = \mathbf{I}_P$ (matriz identidade de dimensão P).

$$\mathbf{H}_{i+1} = \mathbf{H}_i + \mathbf{H}_{i+1} = \mathbf{H}_i + \frac{\mathbf{p}_i \mathbf{p}_i^T}{\mathbf{p}_i^T \mathbf{q}_i} - \frac{\mathbf{H}_i \mathbf{q}_i \mathbf{q}_i^T \mathbf{H}_i}{\mathbf{q}_i^T \mathbf{H}_i \mathbf{q}_i}, i = 0, 1, ..., P-1. \tag{23}$$

Após P iterações sucessivas, se $J(\mathbf{w})$ for uma função quadrática, então $\mathbf{H}_P = \nabla^2 J(\mathbf{w})^{-1}$. Como geralmente não estamos tratando problemas quadráticos, a cada P iterações, é recomendável que se faça uma reinicialização do algoritmo, ou seja, toma-se a direção de minimização como a direção oposta àquela dada pelo vetor gradiente e $\mathbf{H} = \mathbf{I}_P$ (GRAMFORT, 2021).

2.13 Otimização Numérica de Broyden-Fletcher-Goldfarb-Shanno

O BFGS foi nomeado em homenagem às quatro pessoas que o desenvolveram independentemente em 1970: Broyden, Fletcher, Goldfarb e Shanno. O BFGS é um método de otimização derivado da método DFP. Neste método, uma inversão de papéis do vetor $\mathbf{s}_k$ com o $\mathbf{y}_k$ é realizada.

O algoritmo BFGS tem a mesma propriedade do método DFP: no caso quadrático, produz direções conjugadas e converge em menos de n iterações (GRAMFORT, 2021). O diferencial do BFGS em relação ao DFP reside na sua velocidade de convergência que é muito menos sensível ao tamanho do passo utilizado. É, portanto, considerado um ótimo algoritmo de otimização numérica quando combinado com a regra de Wolfe e Powell ou Goldstein (GRAMFORT, 2021), que permite manter uma aproximação do Hessiano $\mathbf{H}_k$ que satisfaz as restrições: $\mathbf{H}_{k+1} > 0$ se $\mathbf{H}_k > 0$, satisfazendo a relação Quase-Newton:

$$\mathbf{y}_k = \mathbf{H}_k \mathbf{s}_k. \tag{24}$$

A fórmula, portanto, é:

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{s}_k \mathbf{s}_k^T}{\mathbf{s}_k^T \mathbf{y}_k} - \frac{\mathbf{B}_k \mathbf{y}_k \mathbf{y}_k^T \mathbf{B}_k}{\mathbf{y}_k^T \mathbf{B}_k \mathbf{y}_k}. \tag{25}$$

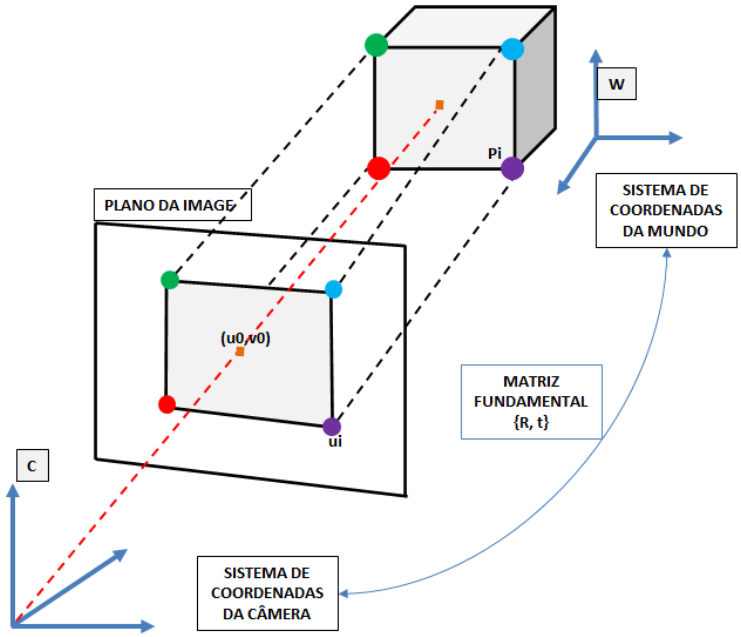
A direção $\mathbf{d}_k$ é obtida resolvendo-se um sistema linear. No entanto, na prática, a atualização do $\mathbf{H}_k$ é feita na fatoração de Cholesky (Equação 26), o que significa que a complexidade do BFGS é o igual ao DFP. O uso de uma fatoração de Cholesky é útil para verificar se $\mathbf{H}_k$ é numericamente positivo definitivo, uma vez que essa propriedade pode ser perdida devido a erros numéricos.

$$\mathbf{H}_k = \mathbf{C}_k \mathbf{C}_k^T \tag{26}$$

2.14 Bundle Adjustment

O problema da otimização para estimar a estrutura da cena juntamente com a posição da câmera é chamado de *Bundle Adjustment* (BA). O método BA utiliza uma minimização não linear para estimar a estrutura da cena quando as poses da câmera são imutáveis.

Figura 16 – Ilustração da aplicabilidade e limitações da matriz fundamental.



Fonte: do autor.

Conforme ilustra a Figura 16, quando usamos a matriz fundamental para estimar a posição e orientação da câmera no espaço, fazemos isso utilizando as *features* extraídas da imagem, porém, ficamos sujeitos às restrições que surgem quando ocorrem oclusões de *features*, que não são mais encontradas e que podem afetar negativamente o processo de estimação da pose da câmera.

O método BA trata todo o problema de estimativa da pose como um problema gigantesco de mínimos quadrados não lineares, e tenta estimar a pose da câmera, utilizando as *features* presentes em comum em um conjunto de imagens consecutivas, com o objetivo de encontrar uma solução aproximada para todas as imagens.

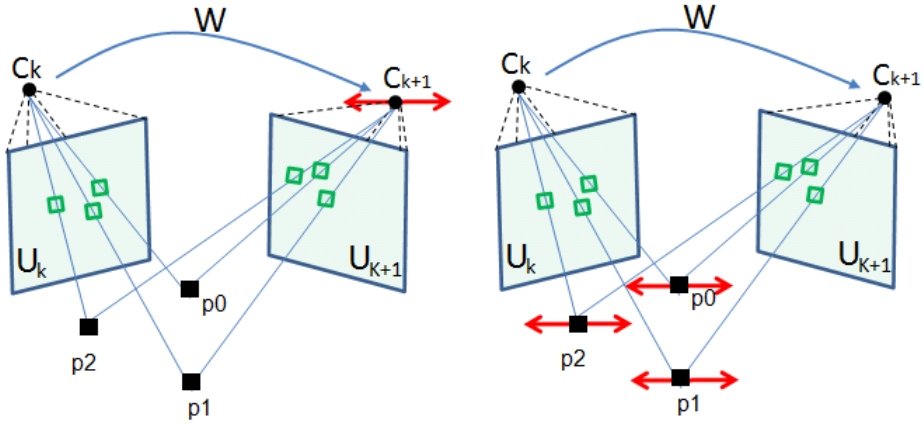
O algoritmo Perspectiva-n-Pontos (PnP) é intimamente relacionado ao BA, e pode ser definido como o problema de estimar a pose de uma câmera calibrada dado um conjunto de n pontos tridimensionais no espaço de coordenadas mundo e suas correspondentes projeções 2D nas imagens, através da minimização do erro de reprojeção (FISCHLER; BOLLES, 1981).

Existem diversas variantes do algoritmo PnP atualmente disponíveis na literatura, como exemplo o algoritmo P3P apresentado por Gao et. al (GAO *et al.*, 2003), que demonstrou a aplicação de um algoritmo PnP na sua forma mais elementar. Outros bons exemplos são os algoritmos EPnP (LEPETIT; MORENO-NOGUER; FUA, 2009) e P5P (NISTER, 2004), bem como métodos iterativos não lineares para um número arbitrário de pontos como Gauss-Newton (GN) (TRIGGS *et al.*, 2000) e Levenberg-Marquardt (LM) (BJORCK *et al.*, 1996). Quando as câmeras são calibradas e seus parâmetros intrínsecos são conhecidos, é possível obter a reconstrução euclidiana ou métrica do cenário, onde as estimativas diferem unicamente dos valores reais (*ground truth*) devido a uma transformação de similaridade desconhecida (HARTLEY; ZISSERMAN, 2004b). Na situação onde os parâmetros intrínsecos da câmera não são conhecidos, a reconstrução pode ser realizada, porém, somente até uma ambiguidade projetiva, onde não se pode garantir que as linhas projetadas sejam paralelas ou perpendiculares.

2.15 Resection-Intersection

O método *Resection-Intersection* (RI), apresentado por Triggs et al. (TRIGGS *et al.*, 2000), descreve uma abordagem de estimação da pose da câmera diferente do clássico *Bundle-Adjustment*. Ao contrário do método *Bundle-Adjustment*, que tenta otimizar simultaneamente a posição das *features* no espaço tridimensional e as poses da câmera como um único problema de otimização esparso, o método RI tenta otimizar separadamente a posição das *features* (*Intersection*) e da pose da câmera (*Resection*), conforme ilustrado na Figura 17.

Figura 17 – Ciclo de funcionamento do *Resection-Intersection*.



(a) Resection (b) Intersection

Fonte: Adaptado de (PEREIRA, 2018).

Na Figura, C_k representa a posição e orientação da câmera no tempo k, **w** é o vetor que

contém a translação relativa da câmera e sua rotação, do tempo k para o tempo $k+1$. O um conjunto de pontos 3D é representado por **P** e **U** é o conjunto que contém da localização das *features* no plano das imagens k e $k+1$. O principal diferencial deste método é utilizar a variação da posição 3D das *features* extraídas ao calcular os ajustes incrementais da pose da câmera. Contudo, para fazer isso é necessário aplicação do procedimento de triangulação (seis operações para cada atualização da pose da câmera), caracterizando a triangulação como um elemento crítico da estimativa de posicionamento da câmera. Para tentar reduzir o custo da estimativa da posição da característica 3D, o método rápido de triangulação, descrito em (PEREIRA *et al.*, 2018), foi utilizado nesta implementação.

2.16 Ambiguidade da Similaridade da Cena

A recuperação das distâncias reais ou escala absoluta das *features* detectadas na cena (na escala métrica) de um sistema de OVM é uma das etapas fundamentais da Odometria Visual. A escala real da cena não pode ser estimada diretamente devido à ambiguidade da similaridade da cena, isto é, a profundidade real das *features* não pode ser recuperada diretamente das imagens monoculares. Na Figura 18 temos um exemplo clássico de ambiguidade de similaridade.

Figura 18 – Exemplo clássico de ambiguidade de similaridade.

(a) (b)

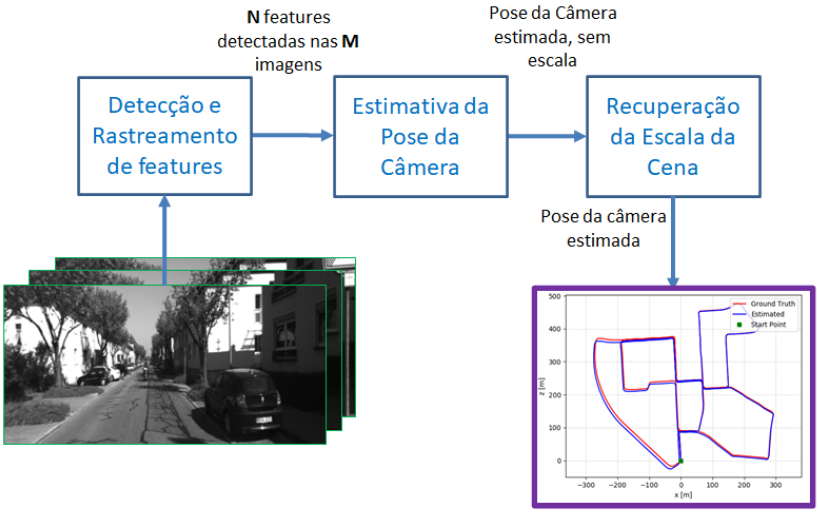
Fonte:(ALAMEDY, 2016)

Ao olhar para a Figura 18 (a), nos deparamos com uma imagem comum do início do século XVIII, porém, em (b) descobrimos que é apenas uma maquete realística, e que não existem informações em imagens monoculares capazes de nos fornecer diretamente o tamanho absoluto. Portanto, sistemas OVM devem utilizar os parâmetros intrínsecos da câmera e métodos geométricos para realizar a estimação real da escala da cena. Neste trabalho, o método utilizado para classificação das *feature* que estão na região do solo foi o mesmo utilizado por Pereira et al. (PEREIRA *et al.*, 2018).

2.17 Configuração Básica de Odometria Visual Monocular Esparsa

Sistemas de navegação baseados em OVM, em especial esparsa, são problemas divididos em duas etapas inter-relacionadas: a estimativa da posição da câmera no momento em que as imagens foram registradas e a inferência da estrutura da cena 3D.

Figura 19 – Fluxograma genérico de OVM.



Fonte: do autor.

Embora os resultados de OVM estejam relacionados apenas à posição e orientação da câmera, a estrutura 3D da cena visualizada deve ser estimada e os erros na estimativa de cena refletem nos resultados de localização. O diagrama de blocos de um sistema OVM, generalizado em três unidades, está ilustrado na Figura 19.

O primeiro bloco do diagrama acima é o da unidade detecção e rastreamento de *features*, onde são selecionadas duas ou mais imagens da cena; nessas imagens são extraídas *features* (pontos de interesse, linhas, etc.), que por sua vez são localizadas em pelo menos duas imagens consecutivas. Na segunda unidade, temos a parte de interpretação geométrica e localização da câmera na cena, isto é, nessa etapa é feita uma estimativa da posição da câmera em relação às *features* extraídas. Por fim, na última unidade, a escala da cena é estimada e a pose e localização da câmera são ajustadas para a escala real da cena.

3 REVISÃO DA LITERATURA

A pesquisa sobre veículos inteligentes visando a automatização e melhoria da segurança do tráfego remonta à década de 1990 (TSUGAWA, 2005). Desde então, ADASs (*Advanced Driver Assistance* Systems) e Sistemas de Navegação Autônomos (SNA) têm ganhado cada vez mais relevância como um dos componentes fundamentais dessa linha de pesquisa. Contudo, a fomentação dos estudos de mapeamento e orientação tridimensional começou muito antes. Na aurora da década de 80, os trabalhos de Smith e Cheeseman (SMITH; CHEESEMAN, 1986) e Durrant-Whyte (DURRANT-WHYTE, 1988) demonstraram como os marcos de um mapa estão relacionados estatisticamente e concluíram que a correlação entre marcos do mapa crescia à medida que novas observações dos marcos fossem feitas. Pouco tempo depois, Smith et al. (SMITH; SELF; CHEESEMAN, 1990) publicou um trabalho demonstrando que, no mapeamento de um ambiente desconhecido, todos os marcos estão correlacionados em razão ao erro de localização do robô. Evidenciou-se, a partir deste ponto, a importância de solucionar o problema de localização simultaneamente ao problema de mapeamento, uma vez que estes estão intrinsecamente relacionados.

De forma simultânea, outros pesquisadores trabalhavam com diferentes métodos de navegação, como o trabalho de Raja Chatila e Laumond com sonares (CHATILA; LAUMOND, 1985)) e o trabalho com visão computacional de Faugeras e Ayache (AYACHE; FAUGERAS, 1988). Um dos primeiros trabalhos que visa o cálculo da estimativa da pose da câmera utilizando poucos pontos foi o trabalho de Horaud (HORAUD *et al.*, 1989), que formalizou o problema de *Perspective-n-Point* de forma objetiva ao demonstrar a quantidade mínima de pontos para cada solução analítica necessária para o cálculo de posição da câmera.

Devido à alta complexidade computacional e às condições de convergência da solução, alguns trabalhos (RENCKEN, 1993; LEONARD; DURRANT-WHYTE, 2004) foram publicados na tentativa de dividir a solução em várias soluções menores. Contudo, o trabalho PTAM de Klein e Murray (KLEIN; MURRAY, 2007) foi o primeiro trabalho a dividir com expressividade o procedimento de rastreamento de *features* de imagem e esti-

mativa da pose geométrica em dois *pipelines* simultâneos, utilizando otimização numérica não linear ao invés do filtro Kalman. Isto lhe permitia gerenciar milhares de *features* em tempo real, graças ao método de mapeamento separado. Embora o trabalho de PTAM tenha sido de grande importância, sendo fonte de influência para diversos outros trabalhos (SONG; CHANDRAKER; GUEST, 2016; PERSSON *et al.*, 2015; SHI; TOMASI, 1994), ele não possuía uma inicialização automática, sendo necessário mover a câmera em determinada direção no início do processamento.

Dentre os trabalhos de OVM mais recentes, a estrutura de *pipeline* é a mais difundida e são os que apresentam os melhores resultados de acordo com o *rank* de classificados no site do KITTI (PERSSON *et al.*, 2015). Essa estrutura pode ser generalizada nas seguintes etapas: detecção de *features*, rastreamento de *features*, estimativa de pose e recuperação de escala. A detecção de características geralmente ocorre em cada nova imagem como nos trabalhos de Fanani (FANANI *et al.*, 2017) e Mirabdollah (MIRABDOLLAH; MERTSCHING, 2015), ou utilizando um subconjunto de imagens selecionadas chamadas quadros-chave (SONG; CHANDRAKER; GUEST, 2016).

Um dos primeiros sistema SLAM (*Simultaneous Localization and Mapping*) idealizado para utilizar câmeras monoculares foi MonoSlam (DAVISON *et al.*, 2007). Esse sistema utiliza um filtro de Kalman Estendido para atualizar um mapa probabilístico da cena tridimensional e estabelece os pontos tridimensionais da cena e o movimento da câmera como variáveis de estado com probabilidade gaussiana.

Os trabalhos de OVM que lidam com um grande número de informações extraídas das imagens são denominados sistemas do tipo densos ou semi-densos. Esses métodos são muito bastante utilizados quando se deseja construir um mapa tridimensional e detalhado da cena, pois a profundidade é estimada para todos os píxeis que apresentam um valor de gradiente relevante. Entretanto, devido à grande quantidade de informações, sistemas densos tendem a ser mais suscetíveis a *outliers* em cenas não estáticas. Por outro lado, sistemas do tipo esparsos utilizam um pequeno número de informação rastreáveis (*features*), extraídas com métodos de detectores de arestas, como Shi-Tomasi (SHI; TOMASI, 1994) ou FAST (ROSTEN; DRUMMOND, 2006), ou algoritmos descritores invariantes à escala de características, como SIFT (LOWE, 1999) ou ORB (RUBLEE *et al.*, 2011) e rastreados com métodos de busca com Lucas-Kanade (YVES BOUGUET, 2000),

A estimativa da pose da câmera é o processo usado para encontrar a posição e a orientação da câmera que minimiza a diferença entre as *features* extraídas da imagem e sua projeção, construída no espaço tridimensional do plano da imagem. A pose da câmera pode ser extraída diretamente da matriz fundamental ou estimada através da minimização do erro de reprojeção utilizando algoritmos baseados em PnP (*Perspective-n-Point*). Isto foi demonstrado por Mirabdollah et al. (MIRABDOLLAH; MERTSCHING, 2015, 2014), que apresentou um algoritmo PnP de cinco pontos (NISTER, 2004) que realiza a estimativa da pose da câmera utilizando um algoritmo de triangulação *multiframe* para as

posições dos pontos.

O popular método *Bundle Adjustment* (BA) é uma técnica utilizada para refinar simultaneamente a pose da câmera e as posições das *features*. O trabalho proposto por Pereira et al. (PEREIRA *et al.*, 2017) usa três imagens como entrada, combinadas com um ajuste de BA com configuração de estimativas cíclicas para construir um sistema com grande precisão. Uma otimização não linear é usada por Fanani et al. (FANANI *et al.*, 2017), e posteriormente explorado por Engel et al. (ENGEL; KOLTUN; CREMERS, 2016), para minimizar a diferença de intensidade dos píxeis em torno das *features* extraídas (ou seja, discrepância de intensidade de pixel) ao invés da distância de reprojeção usual.

No trabalho de Engel et al. (ENGEL; STüCKLER; CREMERS, 2015), os autores propõem uma solução para Odometria Visual, capaz de ser executada em tempo real em uma CPU. Essa solução realiza a reconstrução 3D utilizando mapas semi-densos de profundidade inversa, construídos considerando cada profundidade como uma distribuição de probabilidade gaussiana com a medida atualizada a cada nova imagem. Diversas evoluções dessa abordagem foram publicadas nos últimos anos. Embora essa abordagem implica em alto custo computacional para determinar a contribuição de cada *feature* no cálculo da posição da câmera, são necessárias pouquíssimas *features* para obter-se uma precisão adequada da estimativa da pose da câmera e seu custo total acaba sendo menor que os métodos baseados em fluxo óptico. Essa abordagem vem evoluindo gradativamente nos últimos anos e serviu de base para o desenvolvimento de diversos outros trabalhos de Odometria Visual (QUAN; LAN, 1999; PEREIRA *et al.*, 2017). Nas próximas seções os trabalhos mais recentes de OVM serão comentados e comparados.

3.1 MPVue

Muitos autores buscaram acelerar partes de seus algoritmos utilizando hardwares específicos para isso. Dentre os trabalhos mais recentes está o trabalho de Ilha et al. (ILHA, 2018), que apresentou uma nova plataforma denominada de MPVue, composta por um Sistema Multi-Processador Heterogêneo de Memória Distribuída em Chip estruturado sobre uma rede em chip (NoC) em malha 2D (DM-HMPSoC), adaptada para realizar eficientemente o paralelismo de comunicação e execução em sistemas embarcados, visando aplicações de visão computacional.

Ambos os processadores propostos, LaPSIman e LaPSIno, foram baseados na plataforma PULPino (TRABER; GAUTSCHI; SCHIAVONE, 2016). Os detalhes da implementação dos dois *cores* foram descritos em conjunto com os testes realizados em cada um para a verificação da implementação. A arquitetura de software embarcado foi definida utilizando o modelo de programação orientada a serviços, montada em um modelo *bottom-up*, desde as funções de baixo nível até os módulos ligados à parte importante do sistema. Para testar a funcionalidade e a viabilidade do sistema, foram desenvolvidos

algoritmos de Odometria Visual em linguagem em C, que utiliza o fluxo óptico para estimar a pose da câmera.

Além disso, definições de algoritmos e estrutura de suporte são apresentadas para tornar possível a implementação do sistema final em FPGA. Uma análise de erros é realizada demonstrando uma forma de filtragem eficaz na eliminação de pontos errados do cálculo, antes que eles poluam a nuvem de pontos gerada.

3.2 MLM-SFM

No trabalho de Song et. al (SONG; CHANDRAKER; GUEST, 2013) foi proposto um sistema OVM que utiliza uma configuração multi-thread e que apresentou um significativo aprimoramento na precisão da estimativa da pose da câmera. O sistema proposto ajustava a pose da câmera e as posições dos pontos 3D extraídos das imagens de entrada utilizando um método baseado no conceito de *Resection-Intersection*, utilizando uma janela deslizante dos últimos 10 *frames*. O pacote SBA (LOURAKIS; ARGYROS, 2009) foi aplicado para criar uma sequência de operações que alternava entre o processo de triangulação e de estimativa da pose da câmera. A escala foi estimada através da triangulação dos pontos no solo combinada com uma projeção de solo fixa.

O trabalho foi aprimorado posteriormente em 2016 (SONG; CHANDRAKER; GUEST, 2016). Na nova versão, a estimativa de escala foi aprimorada pela combinação de várias estimativas. Para cada nova estimativa, o erro de distribuição era aproximado a uma distribuição Gaussiana e um Filtro de Kalman (KF) foi aplicado para combinar as diferentes configurações em uma estrutura estatística. Quando proposta pela primeira vez, essa solução apresentou precisão comparável aos sistemas com visão estéreo da época, com resultados que superaram todos os outros sistemas OVM publicados até o momento. Um método de homografia densa foi usada para se obter uma melhor projeção do plano terrestre.

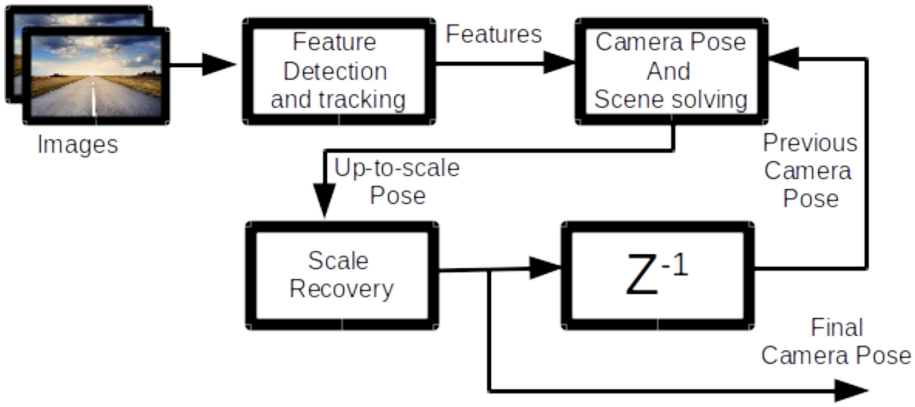
O detector de arestas FAST e descritores ORB foram usados para detecção e rastreamento de *features*, que alimentavam um algoritmo PnP de 4 pontos (LEPETIT; MORENO-NOGUER; FUA, 2009) para calcular a pose de câmera. Além disso, os objetos detectados na cena foram usados para identificar linhas verticais e aproximar o ângulo de inclinação da câmera.

3.3 RI_MVO

A solução RI_MVO ocupou o topo dos trabalhos OVM no *rank* do banco de dados KITTI até o final de 2018 (PEREIRA *et al.*, 2018). Neste trabalho, um sistema OVM baseado no conceito *Resection-Intersection* (TRIGGS *et al.*, 2000), que otimiza a posição das *features* no espaço 3D e a pose da câmera em etapas alternadas foi apresentado. O

fluxograma de operação do sistema RI_MVO esta ilustrado Figura 20.

Figura 20 – Arquitetura do sistema RI_MVO.



Fonte: (PEREIRA, 2018).

Na figura, o módulo *Feature Detection and Tracking* recebe como entrada as imagens provenientes de uma câmera instalada em um veículo e identifica a correspondências entre as *features*. O módulo *The Camera Pose and Scene Solving* usa as *features* rastreadas nas imagens de entrada e a pose da câmera anteriormente estimada para calcular a pose da câmera atual. O módulo *Scale Recovery* calcula e ajusta a escala de translação do movimento da câmera. O símbolo Z^{-1} representa uma memória usada para fornecer um atraso de um quadro na pose da câmera (PEREIRA, 2018).

O método proposto fundamenta-se na otimização não linear de Gauss-Newton para minimizar o erro de reprojeção e estimar a pose da câmera. Sendo suas principais contribuições um algoritmo de triangulação otimizado e o uso do desvio médio do erro de reprojeção no cálculo do ajuste incremental, utilizado na estimativa da pose da câmera. Além disso, o método proposto realiza a estimativa da pose de câmera utilizando apenas duas imagens, em uma configuração que combina simplicidade, desempenho e robustez como nenhum outro algoritmo OVM do tipo esparso, baseado no conceito RI.

A solução RI_MVO foi desenvolvida e avaliada utilizando o popular banco de dados KITTI (GEIGER; LENZ; URTASUN, 2012). Os resultados obtidos na estimativa da pose da câmera superam os resultados de todos os trabalhos OVM baseados no conceito RI anteriormente publicados. Devido ao seu grande desempenho, a solução RI_MVO foi utilizado como principal referência ao longo do desenvolvimento e validação da presente tese.

3.4 DVSO

No trabalho de Yang et al. (YANG *et al.*, 2018) um novo sistema denominado *DVSO: Deep Virtual Stereo Odometry* foi proposto. Nele os autores apresentaram um novo sistema de Odometria Visual Monocular, que recupera a escala métrica e reduz o *drift* da estimação da escala da cena através da aplicação de técnicas de *Deep Learning*, que utilizam reconstruções auto-supervisionadas de perdas e previsões de profundidade esparsas.

O sistema constrói mapas de profundidade monoculares das imagens de entrada, aplicados na inicialização das profundidades esparsas, para se obter uma escala métrica consistente da cena, resultando em uma estimação da pose da câmera mais precisa e estável. A rede foi treinada de forma semi-supervisionada e foto-consistente com imagens estereoscópicas consistentes com reconstruções esparsas provenientes do método DSO estéreo.

Os métodos de aprendizagem baseados em dados têm demonstrado um desempenho mais robusto e preciso, especialmente em algumas das cenas mais desafiadoras do *bechmark* KITTI. A avaliação do sistema proposto demonstra que DVSO se equipara aos métodos monoculares de última geração e atinge resultados comparáveis aos métodos Odometria Visual Estéreo.

3.5 SGANVO

Dentre os trabalhos mais recentes de OVM, os que utilizam métodos de aprendizagem profunda não supervisionados de ponta a ponta tem alcançado resultados significativamente superiores aos métodos de OVM geométricas e de egomoção tradicionais.

Feng et al. (FENG; GU, 2019) propos um novo sistema baseado em redes neurais não supervisionadas aplicadas ao problema de estimação da profundidade de imagens, denominado SGANVO: *Stacked Generative Adversarial Network*.

O novo sistema utiliza uma fila de camadas neurais GAN (*Generative adversarial network*), das quais a camada mais baixa estima a profundidade e deslocamento, enquanto as camadas mais altas estimam as características espaciais da cena. Podendo também ser utilizado para capturar a dinâmica temporal da cena devido ao uso de uma representação recorrente entre as camadas. Um rede codificadora-decodificadora foi amplamente utilizada na estimativa de profundidade e o RCNN proposto trouxe melhorias significativas na estimativa do movimento do ego.

Os resultados obtidos pelo SGANVO demonstraram que o método proposto obteve resultados superiores a todos os trabalhos de OVM atualmente submetidos no banco de dados KITTI. Embora, este, não esteja listado no *rank*.

3.6 D3VO

Atualmente, o trabalho mais recente de Yang et al. (YANG *et al.*, 2020) ocupa topo do *rank* de trabalhos submetidos no popular banco de dados KITTI. O novo método OVM, denominado *D3VO: Deep Depth, Deep Pose and Deep Uncertainty for Monocular Visual Odometry*, propõem um novo *framework* para OVM que explora redes neurais profundas em três níveis integrados de previsões: profundidade monocular, incerteza fotométrica e pose relativa da câmera.

Antes de realizar a etapa de treinamento da rede, pares de imagens com condições de iluminação semelhantes foram previamente agrupadas. Além disso, as incertezas fotométricas de píxeis nas imagens de entrada são previamente modeladas, o que melhora a precisão da estimativa de profundidade. O treinamento realizado foi do tipo auto-supervisionado, utilizando vídeos estéreo e sem qualquer supervisão externa. A nova rede aborda explicitamente a mudança de iluminação no conjunto de treinamento com brilho preditivo parâmetros de transformação.

O sistema proposto obteve bons resultados utilizando os *benchmarks* KITTI e EuRoC MAV, rivalizando com os principais métodos mono-inerciais e estéreo-inerciais que utilizam apenas uma única câmera. Os resultados mostram que o D3VO supera a maioria dos métodos de OVM tradicionais de última geração. Ele também atinge resultados comparáveis à odometria estéreo/LiDAR de última geração no *rank* KITTI e à Odometria Visual inercial de última geração no EuRoC MAV, usando apenas uma única câmera.

3.7 OV2SLAM

Muitas aplicações de SLAM em realidade aumentada, robótica ou direção autônoma, requerem soluções versáteis, robustas e precisas, na maioria das vezes com capacidade em tempo real. Nesta óptica, Ferrera et. al. (FERRERA *et al.*, 2021) apresentaram o sistema OV2SLAM, um algoritmo VSLAM completo que visa preencher a lacuna entre precisão, robustez e capacidade. O sistema OV2SLAM foi projetado para ser um sistema versátil, capaz de ser utilizado em configurações terrestres ou aéreas, em ambientes tanto internos quanto externos. A arquitetura do sistema proposto foi desenvolvida respeitando as restrições de operação em tempo real sem sacrificar a precisão da estimação da pose da câmera.

O sistema proposto pode ser utilizado nas configurações de câmera monocular e estéreo, em mapa com diferentes escalas e taxas de quadros que variam de alguns Hertz até várias centenas de Hertz. O sistema proposto foi avaliado em vários conjuntos de dados, tanto em configurações monoculares quanto em estéreo, e os resultados apresentados mostram performances de última geração.

3.8 Comparação de Trabalhos

Neste capítulo foi apresentado uma breve revisão dos trabalhos mais relevantes de OVM. A Tabela 1 descreve as características do principais trabalhos citados neste capítulo e atualmente listados no *rank* do conjunto de dados KITTI (GEIGER; LENZ; URTASUN, 2012).

Tabela 1 – Tabela de comparação entre trabalhos

Solução	**Detecção**	**Rastreamento**	**Est. da Pose**	**Erro de Transl[%]**
MLM-SFM	FAST	ORB	EPnP	2,54
RI_MVO	Shi-Tomasi	LK	RI-GN	1,03
OV2SLAM	FAST/BRIEF	LK	P3P	0,94
DVSO	-	-	RCNN	0,90
D3VO	-	-	RCNN	0,88

Na tabela, Shi-Tomasi (SHI; TOMASI, 1994), FAST (ROSTEN; DRUMMOND, 2006) e BRIEF (PERSSON *et al.*, 2015) são métodos detectores de arestadas utilizados na extração de *features* das imagens. O método ORB é um método utilizado tanto na detecção quanto no rastreamento de *features* (RUBLEE *et al.*, 2011). LK é a expressão referente ao rastreador Lucas-Kanade (LUCAS; KANADE, 1981). O algoritmo RI-GN faz referência ao método de *Resection-Intersection* baseado na otimização não linear de Gauss-Newton, conforme descrito nas Seções 2.10 e 2.15.

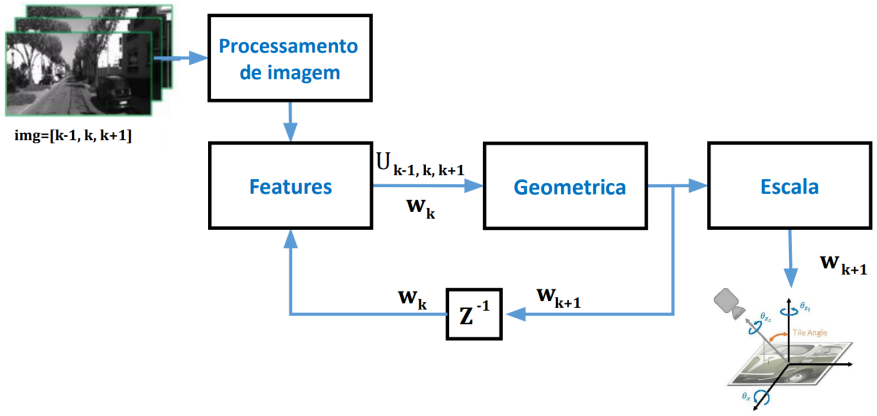
EPnP e P3P são variações do método *Perspective-n-Point*, utilizados para estimar a pose da câmera, conforme descrito na Seção 2.14. RCNN é a expressão referente a redes neurais convolucionais baseadas na região (CAO; NIU; DOU, 2016). A última coluna traz o erro médio de translação medido pelos desenvolvedores do banco de dados KITTI, para uma ideia da precisão relativa de cada solução.

Por inspeção, é possível verificar que, atualmente, os trabalhos que utilizam técnicas de *deep learning* são os trabalhos que apresentam os melhores resultados de OVM. Contudo, os trabalhos de OVM geométricos apresentam resultados bastante próximos, e dessa forma, demonstram potencial para obter resultados ainda mais precisos, sem a necessidade de uma etapa de aprendizagem.

4 ARQUITETURA DO SISTEMA

A arquitetura do sistema implementado foi baseada no fluxograma de operação básico de OVM apresentado na Seção 2.17. O fluxograma de operação do novo sistema está ilustrado na Figura 21. O novo sistema é dividido em quatro unidades: Processamento de Imagem, Features, Geometrica e Escala.

Figura 21 – Fluxograma de operação do sistema OVM esparso proposto neste trabalho.



Fonte: do autor.

Inicialmente, o sistema recebe de entrada três imagens tiradas sequencialmente, denominadas $k-1$, k e $k+1$ e a pose da câmera estimada anteriormente ($\mathbf{w}_k$). As imagens de entrada são combinadas em três pares únicos: ($\mathbf{I}_k$, $\mathbf{I}_{k+1}$), ($\mathbf{I}_{k-1}$, $\mathbf{I}_k$) e ($\mathbf{I}_{k+1}$,$\mathbf{I}_{k-1}$). Contudo, somente o método de refinamento cíclico utiliza os três pares para estimar a nova pose da câmera. Os demais métodos utilizam somente o par ($\mathbf{I}_k$, $\mathbf{I}_{k+1}$) para estimar a nova pose da câmera.

Quando o sistema é inicializado, a primeira estimativa da posição da câmera deve ser realizada sem o conhecimento prévio do movimento, isto é, sem uma estimativa da pose anterior como referência ou alguma informação sobre a velocidade do veículo neste momento. Para contornar isso, o sistema faz três estimativas iniciais de movimento da

câmera, no sentido de movimento do veículo, com três velocidades diferentes definidas experimentalmente. A solução com menor erro de reprojeção é selecionada.

Depois da etapa de inicialização, no inicio de cada nova iteração do sistema uma processo de atualização de algumas variáveis é realizado. Começando pelo conjunto de imagens entrada, que é atualizado da seguinte forma: $\mathbf{I}_{k-1} = \mathbf{I}_k$ e $\mathbf{I}_k = \mathbf{I}_{k+1}$. Em seguida, o sistema recebe uma nova imagem da cena e a define como a nova imagem ($\mathbf{I}_{k+1}$). Por fim, as matrizes que contém a localização das *features* de cada imagem $k-1$ e k são atualizados: $\mathbf{U}_{k-1} = \mathbf{U}_k$ e $\mathbf{U}_k = \mathbf{U}_{k+1}$.

4.1 Processamento de Imagem

Na unidade Processamento de Imagem as imagens de entrada do sistema são ajustadas para enaltecer suas características, de forma a possibilitar uma detecção e rastreamento de *features* mais eficaz e estável. O ajuste utilizado neste trabalho é uma Equalização de Histograma Adaptativo CLAHE, comentada na Seção 2.7, com o objetivo de normalizar a distribuição de iluminação da imagem, para diminuir o efeito negativo da variação de iluminação do ambiente no processo de detecção e rastreamento das *features*.

Figura 22 – Ilustração do resultado da equalização adaptativa de histograma.

Fonte: Adaptado de (GEIGER; LENZ; URTASUN, 2012).

A Figura 22 ilustra o resultado da equalização adaptativa de histograma nas imagens de entrada. O algoritmo de equalização adaptativa de histograma CLAHE divide a imagem em pequenos blocos chamados "tiles", neste trabalho definido com o tamanho 8×8. Em seguida, para cada um desses blocos é realizada a equalização do histograma. Para evitar que a equalização do histograma amplifique ruídos na imagem, um valor de limiar de contraste é utilizado sobre cada tiles. O valor de limiar utilizado foi de 1.5. A função utilizada para realizar equalização adaptativa de histograma foi "clahe" da biblioteca OpenCV (BRADSKI, 2000).

4.2 Detecção e Rastreamento de Features

A detecção de boas *features* é um passo fundamental da Odometria Visual e impacta diretamente tanto na precisão da estimação da pose da câmera quanto no cálculo da escala da cena. Na Odometria Visual uma grande quantidade de *features* é necessária, podendo chegar na ordem de milhares em sistemas esparsos e na ordem de milhões em sistemas densos. Dessa forma, é necessário utilizar técnicas de detecção de *features* que apresentem um boa relação custo de processamento/benefício.

Nesta etapa, temos a unidade Features, onde *features* são extraídas e rastreadas nas imagens de entrada. O termo "*features*" utilizado neste trabalho refere-se a uma região da imagem. Contudo, somente a localização da referência dessas *features* no plano da imagem é salvo, de forma a reduzir o número de informações que o sistema deve armazenar. Nessa etapa, o sistema recebe como entrada o conjunto que contém as localizações das *features* $\mathbf{U}_k$ da iteração passada do sistema, a pose da câmera estimada na iteração passada ($\mathbf{w}_k$) e o par de imagens $\mathbf{I}_k$ e $\mathbf{I}_{k+1}$. O procedimento para detectar e rastrear *features* está descrito nos passos listados a seguir:

1. Utilizando um método de detecção, novas *features* são extraídas na imagem $\mathbf{I}_k$.

2. A localização das novas *features* são comparadas com as antigas (recebidas de entrada $\mathbf{U}_k$) e somente são mantidas aquelas que estão afastadas a um distância de pelos menos 15 píxeis das antigas. A matriz $\mathbf{U}_k$ é atualizada com o resultante dessa filtragem.

3. Uma transformação de perspectiva é aplicada sobre a imagem $\mathbf{I}_{k+1}$ para aproximá-la da imagem $\mathbf{I}_k$, com o objetivo de facilitar o processo de rastreamento das *features*. O procedimento para realizar a transformação de perspectiva é descrito em mais detalhes em (PEREIRA *et al.*, 2017).

4. Um método de rastreamento é utilizado para rastrear as *features* $\mathbf{U}_k$ na imagem $\mathbf{I}_{k+1}$ com a perspectiva modificada. Utilizando as localizações das *features* contidas na matriz $\mathbf{U}_k$ o método de rastreamento consegue recuperar a região da imagem $\mathbf{I}_k$ que descreve a *features* e a utiliza para realizar o rastreamento na outra imagem. Ao final, a matriz $\mathbf{U}_{k+1}$ é atualizada com o conjunto de *features* que foram rastreadas.

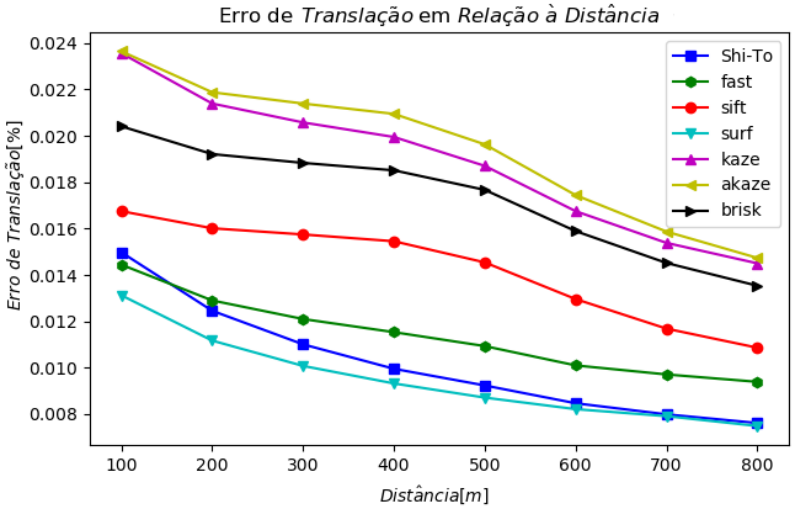
Para realizar a detecção de *features* e localizá-las nas imagens subsequentes foram utilizadas as técnicas de Shi-Tomasi e Lucas-Kanade da biblioteca OpenCV (BRADSKI, 2000). A definição do método de detecção de *features* foi feita após a realização de um estudo, onde foram testadas as principais métodos disponíveis na literatura, tais como: Shi-Tomasi, SURF, SIFT, FAST, AKAZE, KAZE e BRISK (IDRIS; RAZAK, 2009).

Lucas-Kanade (LUCAS; KANADE, 1981) é usado para rastrear as *Features* entre as imagens. O estudo foi realizado utilizando o primeiro cenário do banco de dados KITTI e os resultados obtidos estão ilustrados na Tabela 2 e Figura 23.

Tabela 2 – Desempenho das principais técnicas de detecção de *features*.

Nome da Técnica	**Tempo Médio(s)**	**Nº Médio de *Features* Detectadas**
ORB	0,2596	559
FAST	0,2776	547
SHI-TO	0,2868	495
AKAZE	0,3155	570
SURF	0,3353	573
SIFT	0,3389	565
KAZE	0,5518	565
BRISK	0,6323	574

Figura 23 – Ilustração dos resultados obtidos com cada uma das técnicas de extração de *features* propostas.



Fonte: do autor

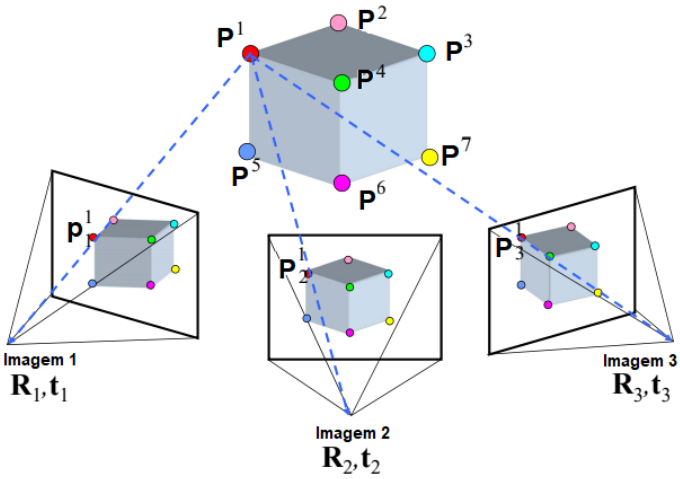
A Tabela 2 mostra o tempo médio e o número médio de *Features* detectadas para cada uma das oito técnicas de detecção testadas. Com base na tabela, fica evidente que o algoritmo ORB é o mais rápido e o algoritmo BRISK é o mais lento dentre todos. A Figura 23 ilustra o erro de translação em relação à distância média percorrida, obtido utilizando-se cada um dos métodos de extração de *Features* das imagens. Para realizar este experimento, um limite máximo de até 600 *Features* foi estabelecido de forma a homogenizar os resultados.

Com base nos resultados acima, o algoritmo de rastreamento SURF foi o que apresentou o melhor resultado. Porém, o método de Shi-Tomasi apresentou um resultado similar ao do SURF com um custo computacional menor, e devido a isso, foi escolhido para ser utilizado neste trabalho. O algoritmo ORB foi removido dos resultados, pois apresentou um desempenho muito inferior aos demais.

4.3 Estimação da Pose da Câmera

Na unidade Geometrica, a última posição relativa da câmera estimada e as *features* extraídas nas imagens de entrada são utilizadas para estimar a posição da câmera em relação à cena. Para tanto, novos métodos de estimação da pose da câmera foram desenvolvidos ao longo deste trabalho, baseados no conceito de *Resection-Intersection*, revisado na Seção 2.15.

Figura 24 – Exemplificação do procedimento realizado na etapa Geometrica



Fonte: Adaptado de (GITTEL, 2014).

Os novos métodos RI implementados e o método de refinamento cíclico estão descritos no Capítulo 5. Os métodos desenvolvidos utilizam o erro de reprojeção para ajustar a última posição relativa da câmera estimada, e dessa forma, obter a próxima posição da câmera. Cada método proposto foi implementado utilizando um algoritmo de otimização numérica diferente, com o objetivo de se verificar a sua eficácia quando aplicados no problema de Odometria Visual.

De forma complementar, um método de refinamento cíclico foi elaborado para maximizar o resultado de estimação da pose da câmera. Nesse método, três imagens consecutivas da cena são utilizadas, conforme ilustra a Figura 24. Essas três imagens são agrupadas em três pares únicos.

O método estimativa a nova pose da câmera utilizando o primeiro par de imagens e a pose da câmera estimada na iteração passada do sistema. Em seguida, o procedimento é repetido utilizando o segundo par de imagens e a pose da câmera estimada utilizando o primeiro par de imagens. Por fim, o mesmo procedimento é realizado apara a última dupla de imagens. Esse ciclo é então repetido N vezes e a pose da câmera obtida com o menor erro de reprojeção entre todos os ciclos é a escolhida.

4.4 Recuperação da Escala da Cena

Conforme abordado na Seção 2.16, a ambiguidade da similaridade da cena impede que a escala da cena seja calculada diretamente em sistemas OVM. O sucesso na estimativa da escala é baseado na estimativa da distância absoluta dos pontos no solo. Dessa forma, a seleção de um ponto estático que esteja realmente no solo e corretamente rastreado é fundamental. Embora definir esse ponto possa parecer uma tarefa trivial à primeira vista, distinguir as *features* estáticas presentes no solo de objetos como carros estacionados, árvores, *features* de baixa qualidade (em asfalto de baixa textura) e de objetos não estáticos (pedestres, bicicletas, etc.) podem revelar-se desafiador.

Ignorando-se as rotações usualmente insignificantes em torno do eixo óptico, e comparando a profundidade da triangulação e a profundidade esperada do solo, toda a escala de cena pode ser estimada através da técnica de similaridade de triângulo. Para tanto, devemos utilizar somente as *features* que estão no plano do chão na cena (asfalto, calçada). Para realizar a classificação das *features* que estão no asfalto ou não, geralmente se define empiricamente uma área estática.

Um estudo foi realizado, onde foram testando diversos formatos de área, para classificar as *features* que estão no solo. Três formatos de regiões foram testadas: triangular, trapezoidal e em forma de arco, conforme ilustra a Figura 25.

Figura 25 – Ilustração das formas de região testadas.

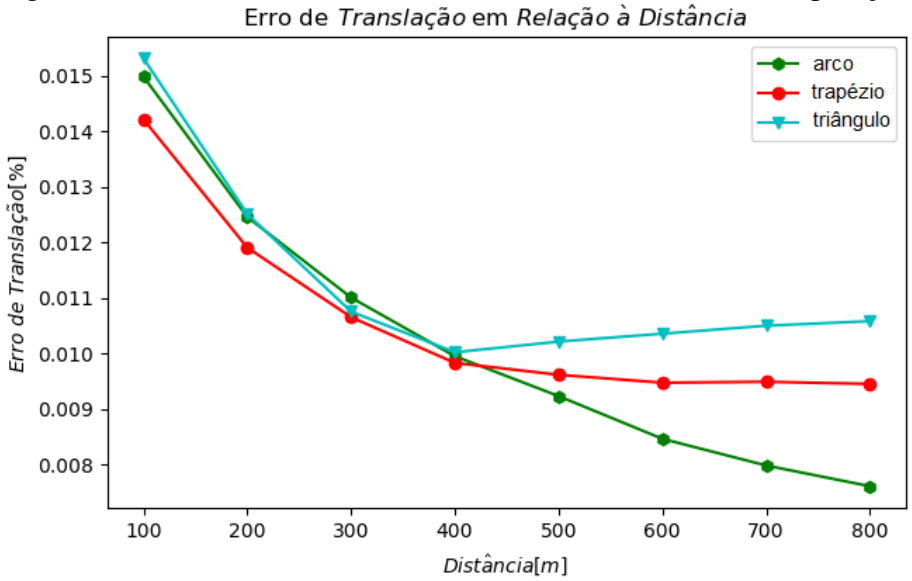
Fonte: Adaptado de (GEIGER; LENZ; URTASUN, 2012).

A classificação das *features* presentes dentro da forma de arco pode ser obtida através da Inequação 27.

$$(u[i] - img\_lenght)^2 + \frac{(v[i] - img\_hight)^2}{q_u} < th\_arc. \tag{27}$$

Na inequação, img_lenght e img_hight são as dimensões verticais e longitudinais da imagem. Os parâmetros u e v são as coordenadas da i-ésima *features* analisada. O parâmetro q_u é um fator utilizado para alargar a área do solo (experimentalmente definido como 6). O parâmetro th_arc é o valor limiar do arco, neste trabalho definido empiricamente como 160^2. A área de forma triangular foi definida utilizando as seguintes coordenadas da imagem: $(0, 375)$, $(555, 190)$ e $(1240, 375)$. A área de forma trapezoidal foi definida utilizando as seguintes coordenadas da imagem: $(0, 375)$, $(315, 215)$, $(735, 215)$ e $(1240, 375)$.

Figura 26 – Resultados obtidos utilizando diferentes áreas de separação.



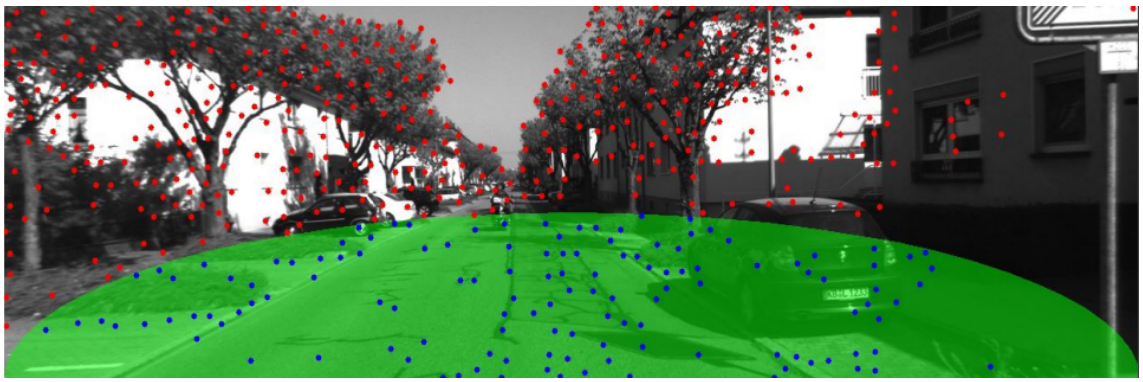
Fonte: do autor

Uma vez que as *features* foram classificadas, a estimação da escala pode ser realizada. A Figura 26 ilustra o erro de translação em relação à distância média percorrida obtido no primeiro cenário do banco de dados KITTI, utilizando cada um dos três formatos de áreas apresentados. Por inspeção dos resultados, o formato de área de trapézio foi o que apresentou os melhores resultados até a distância de 400 metros, contudo, demonstrou uma estabilização nas distâncias acima de 400 metros. Os resultados obtidos utilizando a forma de área em arco foram inferiores aos obtidos pela forma de área de trapézio em distâncias menores que 400 metros, porém, não muito distantes. Para distâncias maiores que 400 metros a forma de arco apresentou resultados significativamente melhores, e por esse motivo, foi o formato de área definido neste trabalho para realizar a classificação de *Features* que estão no solo.

A Figura 27 ilustra um exemplo de classificação que é realizado utilizando um região

na imagem em forma de arco. Na figura, a área em verde representa a forma de arco utilizada, o círculos azuis representam as *Features* identificadas como sendo da região do solo e a os círculos em vermelho as que não são.

Figura 27 – Exemplo da área de classificação da *features* presentes no solo.



Fonte: Adaptado de (GEIGER; LENZ; URTASUN, 2012).

Por fim, uma vez que as *features* na região do solo foram identificadas é necessário identificar a melhor *feature* entre elas para realizar a estimativa da escala. Essa *feature* deve ser eleita considerando diversos fatores como: *features* não estáticos na imagem, a sua altura em relação ao solo e qualidade de rastreabilidade. Neste trabalho, o procedimento utilizado para definição dessa *feature* foi o proposto por (PEREIRA, 2018).

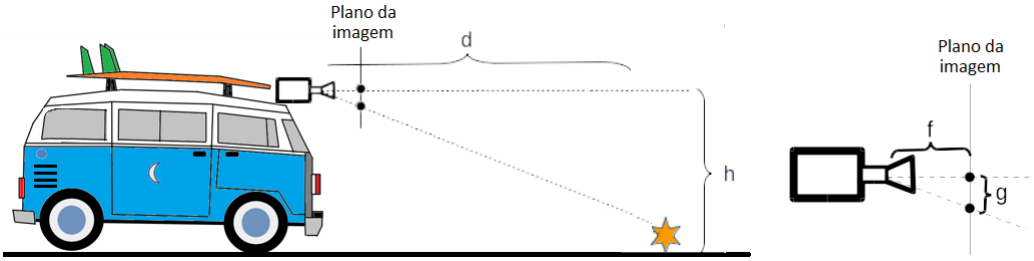
A estratégia mais comum para estimar a escala absoluta da cena em aplicações direcionadas a veículos equipados com câmera foi proposta por Scaramuzza (SCARAMUZZA *et al.*, 2009), e consiste em utilizar a altura da câmera e a posição relativa do solo (altura da câmera em relação ao solo) para estimar a profundidade da cena e a escala do movimento proposto. Na Figura 28 (a) temos um veículo equipado com uma câmera, fixada na parte frontal e a projeção de uma *feature* no espaço mundo e coordenadas, representada por uma estrela de cor amarela, localizada no chão da cena.

A posição da *feature* em relação à câmera forma um triângulo retângulo de altura h (altura da câmera em relação ao chão) e distância horizontal d. Na Figura 28 (b) traz a ilustração de um outro triângulo retângulo de mesmo angulo, definido a longo do centro óptico da câmera, e a posição do plano da imagem. A distância horizontal f (em píxeis) é a distância focal da câmera e a altura g representa a distância vertical (em píxeis) da posição da *feature* no plano da imagem em relação ao centro da imagem.

A partir dos triângulos definidos nas Figuras 28 (a) e (b), uma semelhança de triângulo pode ser facilmente projetada como mostrado na Equação 28.

$$\frac{d}{h} = \frac{f}{g}. \tag{28}$$

Figura 28 – Estrutura de triângulos utilizadas para estimar a escala real da cena.



(a) Estrutura de triângulos da cena.

(b) Estrutura de triângulos da câmera.

Fonte: do autor.

Isolando d:

$$d = \frac{fh}{g}. \tag{29}$$

A escala absoluta pode ser ajustada aproximando-se a profundidade calculada das *features* no processo de triangulação d, com a distância calculada na Equação 29. Nessa Equação assume-se que a rotação ao longo do eixo z seja insignificante e que a rotação ao longo do eixo y (movimento na direção horizontal) não afete a estimativa de escala. É válido, ainda, afirmar que este modelo ignora as pequenas oscilações de rotação no eixo x (inclinação da câmera), embora este parâmetro apresente grande impacto na estimativa de escala.

A Inequação 27, utilizada para classificar *features* que estão no solo, assim como as áreas em forma de triângulo e trapézio, foram obtidas experimentalmente utilizando o banco de dados do KITTI. Representando uma restrição do método de recuperação da escala utilizado neste trabalho. Dessa forma, para diferentes banco de dados os parâmetros da Inequação 27 e o formato das outras áreas devem ser ajustados. Outra limitação pode ser atribuída as lombas (ruas muito íngremes), das quais o método descrito nesta seção não pode ser aplicado diretamente.

5 MÉTODOS DE ESTIMAÇÃO DA POSE DA CÂMERA

Este capítulo descreve os métodos de estimação da pose da câmera propostos neste trabalho. Quatro novos métodos de estimação da pose da câmera baseados em RI foram desenvolvidos. Cada método foi implementado utilizando um algoritmo de otimização numérica diferente, sendo eles: Gauss-Newton, Levenberg-Marquardt, Davidon–Fletcher–Powell e Broyden–Fletcher–Goldfarb–Shanno.

Um teste de convergência com região de busca expandida, utilizado para aprimorar os métodos RI baseados em Gauss-Newton e Levenberg-Marquardt também é proposto neste capítulo. Por fim, o método de refinamento cíclico, utilizado para maximizar a precisão da estimação da pose da câmera é apresentado. Os resultados obtidos com cada um desse métodos são apresentados no Capítulo 6.

5.1 Resection-Intersection Baseado na Otimização de GN

Esta seção descreve a implementação de uma variação do algoritmo RI baseado no método de otimização não linear Gauss-Newton (GN). Quando aplicado, este método realiza uma otimização iterativa que ajusta os parâmetros **w** para minimizar a função residual f, conforme descreve a Equação 30.

$$\underset{\mathbf{w}}{\operatorname{argmin}} f(\mathbf{w}) = \sum_{j=0}^{n-1} r_j^2(\mathbf{w}). \tag{30}$$

Em cada iteração, o algoritmo calcula um incremento de atualização de acordo com a Equação 31. Nesta equação, **r** é o resíduo a ser minimizado, **J** é a matriz jacobiana de derivadas parciais de **r** em relação a cada parâmetro de **w** e s é a etapa de iteração.

$$\mathbf{w}^{(s+1)} = \mathbf{w}^{(s)} + \Delta\mathbf{w} = \mathbf{w}^{(s)} - (\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\mathbf{r}(\mathbf{w}). \tag{31}$$

Neste trabalho, a diferença de projeção padrão é utilizada como forma de medir o erro. Isto é, $\mathbf{r} = (\Delta_0 u, \Delta_0 v, \Delta_1 u, ...\Delta_{n-1} v)^T$, onde $\Delta_i u$ e $\Delta_i v$ são as diferenças horizontais e verticais da projeção em relação à posição da i^{th} *feature* observada: $\Delta_i u =_i u -_i \hat{u}$.

Dado um conjunto de n *features*, $\mathbf{r}$ é um vetor coluna com $2n$ elementos, $\mathbf{J}$ é uma matriz com dimensão $2n \times 6$, onde cada coluna $i \in [0, 5]$ é a derivada parcial de $\mathbf{r}$ com relação a i-ésimo elemento do parâmetro $\mathbf{w}$. O cálculo da matriz Jacobiana, dos resíduos e da nova pose da câmera ($\mathbf{w}_{k+1}$) é apresentada no Algoritmo 1.

Algorithm 1: RI baseado em otimização numérica de Gauss-Newton

1 **Function** *GetPosRI_GN* ($\boldsymbol{U}_k, \boldsymbol{U}_{k+1}, \boldsymbol{w}_k, MaxTry, \gamma$) **:**
2 $\quad \mathbf{m} = ones(lenght(\mathbf{U}_k)$
3 $\quad$ **for** i *in range(*$MaxTry$*)* **do**
4 $\quad\quad \mathbf{P} = triangulate(\mathbf{w}_k, \mathbf{U}_k[\mathbf{m}], \mathbf{U}_{k+1}[\mathbf{m}])$
5 $\quad\quad \hat{\mathbf{U}}_{k+1} = project(\mathbf{w}_k, \mathbf{P})$
6 $\quad\quad \mathbf{r} = vec(\hat{\mathbf{U}}_{k+1} - \mathbf{U}_{k+1})$
7 $\quad\quad \mathbf{d} = \sqrt{\mathbf{r}_x^2 + \mathbf{r}_y^2}$
8 $\quad\quad$ **if** $median(\boldsymbol{d}[\boldsymbol{m}]) < \gamma$ **then**
9 $\quad\quad\quad break$
10 $\quad\quad$ **end**
11 $\quad\quad \mathbf{m} = \mathbf{d} < min(10, percentile(\mathbf{d}, 100 - (4i + 4)))$
12 $\quad\quad$ */*Triangulation, projection and difference*/*
13 $\quad\quad$ **for** $i < 5$ **do**
14 $\quad\quad\quad \mathbf{ws}_{k,i} = \mathbf{w}_k + \mathbf{w}[i]_\Delta$
15 $\quad\quad\quad \mathbf{P}_{\Delta,i} = triangulate(\mathbf{ws}_{k,i}, \mathbf{U}_k[\mathbf{m}], \mathbf{U}_{k+1}[\mathbf{m}])$
16 $\quad\quad\quad \hat{\mathbf{U}}_{k+1,i} = project(\mathbf{ws}_{k,i}, \mathbf{P}_{\Delta,i})$
17 $\quad\quad\quad \mathbf{J}_i = vec(\hat{\mathbf{U}}_{k+1,i}[\mathbf{m}] - \mathbf{U}_{k+1}[\mathbf{m}])/(\mathbf{w}[i]_\Delta)$
18 $\quad\quad$ **end**
19 $\quad\quad \mathbf{w}_\Delta = [\mathbf{J}^T(\mathbf{J}^T\mathbf{J})^{-1}\mathbf{r})]$
20 $\quad\quad \mathbf{w}_{k+1} = CheckForConvergence(\mathbf{w}_k, \mathbf{w}_\Delta, \mathbf{r}, 4)$
21 $\quad$ **end**
22 $\quad$ **return** $\mathbf{w}_{k+1},\ median(\mathbf{d}[\mathbf{m}])$
23 **end**

O algoritmo recebe como entrada as matrizes $\mathbf{U}_k$ e $\mathbf{U}_{k+1}$ que contém a localização das features nas imagens $\mathbf{I}_k$ e $\mathbf{I}_{k+1}$. A pose da câmera estimada anteriormente ($\mathbf{w}_k$), o máximo número de iterações do algoritmo ($MaxTry$) e o critério de parada(γ). Neste trabalho os valores de γ e $MaxTry$ utilizados foram 1×10^{-1} e 18. Entre as linhas 4 e 10 a etapa de *Intersection* é executada. Nesta etapa, a posição das *feature* no espaço 3D é calculada, em seguida, a sua reprojeção no plano da imagem é realizada. Por fim, o erro de reprojeção quadrático é calculado e o critério de parada testado. Entre as linhas 13 e 21 a etapa de *Resection* é executada. Nessa etapa, o erro de reprojeção quadrático é utilizado para calcular uma nova matriz Jacobiana (**J**), que é posteriormente utilizada para calcular

o novo ajuste a ser aplicado, e que definira a nova pose da câmera ($\mathbf{w}_{k+1}$).

Os resíduos são calculados em três etapas: (1) a técnica de triangulação é aplicada para atualizar a posição 3D das *features*, (2) é feita a projeção das posições trianguladas no plano da imagem e em (3) a diferença de projeção é calculada utilizando as *features* projetadas no plano da imagem. Por simplicidade de notação, a matriz constante de parâmetros intrínsecos da câmera **K** foi suprimida na descrição do algoritmo. A função de vetorização vec transforma uma matriz $n \times 2$ em um vetor de tamanho $2n$. Nenhuma informação prévia sobre a posição 3D do ponto é usada.

A matriz jacobiana **J** é calculada numericamente e uma pequena variação é aplicada a cada componente do vetor **w** para aproximar a derivada parcial $\partial\hat{\mathbf{U}}/\partial\mathbf{w}_k[i]$. No algoritmo $w_{\Delta,i}$ é um vetor cujo i-ésimo elemento é um valor pequeno, definido experimentalmente, na ordem de 10^{-6} e todos os outros elementos são zero. As funções de $triangulate$ e $project$ representam a função triangulação e projeção regular de um ponto em uma imagem, anteriormente descritas nas Seções 2.9 e 2.4. Na etapa de atualização, são realizadas ao todo sete operações de triangulação e projeção, uma para o residual e uma para cada um dos seis parâmetros da posição da câmera. O *loop* de atualização é interrompido quando o erro de reprojeção mediano cai abaixo de um limite.

Neste trabalho, assumimos que a escala do movimento provavelmente não será mantida quando uma nova pose da câmera for estimada. Logo, a escala da cena precisa ser constantemente estimada. No cenário de aplicação utilizado, a direção de translação dominante é ao longo do eixo z (movimento para a frente), e não são esperadas mudanças bruscas na velocidade do veículo no intervalo entre imagens. A etapa de atualização é calculada para os primeiros cinco elementos da matriz que descreve a pose relativa da câmera. A última coluna do Jacobiano não é calculada e seis triangulações e projeções são executadas para cada ciclo de atualização. Naturalmente, a velocidade do carro não é constante, mas varia suavemente. Por fim, a translação real é ajustada com base na extração de escala absoluta, apresentada no Seção 4.4.

5.2 Resection-Intersection Baseado na Otimização de LM

Esta seção descreve um algoritmo *Resection-Intersection* para estimação da pose da câmera baseado no método de otimização não linear Levenberg-Marquardt (LM) (TRIGGS *et al.*, 2000). Da mesma forma que o método de otimização de Gauss-Newton, este método realiza uma otimização iterativa que ajusta os parâmetros **w** para minimizar a função residual f, conforme descreve a Equação 32.

$$\underset{\mathbf{w}}{\operatorname{argmin}} f(\mathbf{w}) = \sum_{j=0}^{n-1} r_j^2(\mathbf{w}). \tag{32}$$

Em cada iteração, o algoritmo calcula um incremento de atualização de acordo com a Equação 33. Contudo, diferentemente do método de GN, um fator de amortecimento é aplicado sobre o ajuste.

$$\mathbf{w}^{(s+1)} = \mathbf{w}^{(s)} + \alpha\Delta\mathbf{w} = \mathbf{w}^{(s)} - \alpha(\mathbf{J}^T(\mathbf{J}^T\mathbf{J})^{-1}\mathbf{r}(\mathbf{w})). \tag{33}$$

Nesta Equação, **r** é o resíduo a ser minimizado, **J** é a matriz jacobiana de derivadas parciais de **r** em relação a cada parâmetro de **w** e s é a etapa de iteração.

$$\alpha = (\mathbf{J}^T\mathbf{J}) + \lambda\mathbf{I}. \tag{34}$$

Onde α é o fator de amortecimento de Levenberg-Marquardt e neste trabalho foi definido como mostra a Equação 34. O assim chamado parâmetro de Marquardt λ é definido em cada iteração com base nas ocorrências de divergências. Quando ocorre uma divergência, o parâmetro Marquardt é aumentado gradualmente até que haja uma diminuição em $f(w)$. Na próxima iteração, os sistema inicializa com o valor λ anterior e começar a diminuí-lo gradualmente até que um valor de corte seja atingido.

De forma similar ao GN, o vetor $\mathbf{r} = (\Delta_0 u, \Delta_0 v, \Delta_1 u, ...\Delta_{n-1} v)^T$ representa o erro de reprojeção entre a diferença horizontal $\Delta_i u$ e a diferença vertical $\Delta_i v$ da i^{th} *feature* observada. O algoritmo que descreve o método RI baseado na otimização de LM é apresentado no Algoritmo 2.

O funcionamento do Algoritmo 2 é similar ao apresentado no Algoritmo 1. O algoritmo recebe como entrada as matrizes $\mathbf{U}_k$ e $\mathbf{U}_{k+1}$ que contém a localização das *features* nas imagens $\mathbf{I}_k$ e $\mathbf{I}_{k+1}$. A pose da câmera estimada anteriormente($\mathbf{w}_k$), o máximo número de iterações do algoritmo ($MaxTry$) e o critério de parada(γ). Entre as linhas 4 e 10 a etapa de *Intersection* é executada. Nesta etapa, a posição das *feature* no espaço 3D é calculada, em seguida, a sua reprojeção no plano da imagem é realizada.

Por fim, o erro de reprojeção quadrático é calculado e o critério de parada testado. Entre as linhas 13 e 21 a etapa de *Resection* é executada. Nessa etapa, o erro de reprojeção quadrático é utilizado para calcular uma nova matriz Jacobiana (**J**), que é posteriormente utilizada para calcular o novo ajuste a ser aplicado, e que definira a nova pose da câmera ($\mathbf{w}_{k+1}$). Neste trabalho os valores de γ, λ e $MaxTry$ utilizados foram 1×10^{-1}, 1×10^{-8} e 18

A diferença entre o Algoritmo 1 e o Algoritmo 2 se concentra na linha 19, onde o fator de amortecimento de Marquardt (α) é calculado e na linha 20 onde ele é aplicado sobre o ajuste $\mathbf{w}_\Delta$. O teste de convergência com região de busca expandida também é utilizado neste método, com o objetivo de maximizar o processo de otimização numérica. O teste de convergência esta descrito na próxima seção.

Algorithm 2: RI baseado em otimização numérica de Levenberg-Marquardt

Function *GetPosRI_LM* ($\boldsymbol{U}_k, \boldsymbol{U}_{k+1}, \boldsymbol{w}_k, MaxTry, \gamma$) **:**
- $\mathbf{m} = ones(lenght(\mathbf{U}_k)$
- **for** i *in range(*$MaxTry$*)* **do**
  - $\mathbf{P} = triangulate(\mathbf{w}_k, \mathbf{U}_k[\mathbf{m}], \mathbf{U}_{k+1}[\mathbf{m}])$
  - $\hat{\mathbf{U}}_{k+1} = project(\mathbf{w}_k, \mathbf{P})$
  - $\mathbf{r} = vec(\hat{\mathbf{U}}_{k+1} - \mathbf{U}_{k+1})$
  - $\mathbf{d} = \sqrt{\mathbf{r}_x^2 + \mathbf{r}_y^2}$
  - **if** $median(\boldsymbol{d}[\boldsymbol{m}]) < \gamma$ **then**
    - $break$
  - **end**
  - $\mathbf{m} = \mathbf{d} < min(10, percentile(\mathbf{d}, 100 - (4i + 4)))$
  - */*Triangulation, projection and difference*/*
  - **for** $i < 5$ **do**
    - $\mathbf{ws}_{k,i} = \mathbf{w}_k + \mathbf{w}[i]_\Delta$
    - $\mathbf{P}_{\Delta,i} = triangulate(\mathbf{ws}_{k,i}, \mathbf{U}_k[\mathbf{m}], \mathbf{U}_{k+1}[\mathbf{m}])$
    - $\hat{\mathbf{U}}_{k+1,i} = project(\mathbf{ws}_{k,i}, \mathbf{P}_{\Delta,i})$
    - $\mathbf{J}_i = vec(\hat{\mathbf{U}}_{k+1,i}[\mathbf{m}] - \mathbf{U}_{k+1}[\mathbf{m}])/(\mathbf{w}[i]_\Delta)$
  - **end**
  - $\alpha = (\mathbf{J}^T\mathbf{J}) + \lambda\mathbf{I}$
  - $\mathbf{w}_\Delta = \alpha \times [\mathbf{J}^T(\mathbf{J}^T\mathbf{J})^{-1}\mathbf{r})]$
  - $\mathbf{w}_{k+1} = CheckForConvergence(\mathbf{w}_k, \mathbf{w}_\Delta, \mathbf{r}, 4)$
- **end**
- **return** $\mathbf{w}_{k+1},\ median(\mathbf{d}[\mathbf{m}])$

end

5.3 Teste de Convergência Expandido

O teste de convergência é comumente utilizado por outros autores (PEREIRA *et al.*, 2018; TRIGGS *et al.*, 2000) para verificar as redondezas da pose da câmera estimada em cada iteração do algoritmo RI, com o objetivo de contornar possíveis mínimos locais, e dessa forma, maximizar a sua precisão. O teste de convergência com região de busca expandida é utilizado neste trabalho nos algoritmos RI baseados nas otimizações numéricas de GN e LM. Os algoritmos RI baseados em DFP e BFGS se mostraram eficientes em evitar mínimos locais e não necessitam do teste de convergência. O algoritmo que descreve o teste de convergência expandida utilizado neste trabalho está descrito no Algoritmo 3. A diferença do teste de convergência proposto neste trabalho em relação aos utilizados em outros trabalhos reside na busca feita utilizando um ajuste negativo, resultando assim,

em uma busca mais abrangente, e consequentemente, mais eficaz.

O Algoritmo 3 é inicializado com a nova pose da câmera, com o ajuste da iteração atual, o erro de reprojeção atual e com o número máximo de iteração que o algoritmo deve performar. Começa-se calculando uma pequena correção (**step**) na linha 5. Entre as linhas 6 e 10, o algoritmo aplica a pequena correção na pose atual da câmera nas direções positivas e calcula o erro de reprojeção utilizando a pose da câmera (**w1**) ajustada. Entre as linhas 12 e 15, o mesmo procedimento é realizado, porém, utilizando um ajuste na direção negativa. Entre as linhas 16 e 23, o algoritmo verifica qual deles tem o menor valor de erro de reprojeção e se os erros atuais são melhores do que os encontrados nas iterações passadas.

Algorithm 3: Algoritmo de verificação de convergência da otimização

```
1  Function CheckForConvergence (w_k, w_Δ, e_k, MaxTry) :
2      e_p1, e_p2 = mean(abs(e_k))
3      w_1, w_2 = w_k, w_k
4      for j in range(MaxTry) do
5          step = w_Δ/2
6          /* Positive increment */
7          w_1 = w_1 + step
8          P = triangulate(w_1, U_k, U_{k+1})
9          Û_{k+1} = project(w_k, P)
10         e1 = mean(abs(Û_{k+1} − U_{k+1}))
11         /* Negative increment */
12         w_2 = w_2 − step
13         P = triangulate(w_2, U_k, U_{k+1})
14         Û_{k+1} = project(w_k, w)
15         e2 = mean(abs(Û_{k+1} − U_{k+1}))
16         if (e1 < e2)and(e1 < e_p1)and(e1 < e_p2) then
17             w_{k+1} = w_1
18             e_p1 = e1
19         end
20         if (e2 < e1)and(e2 < e_p2)and(e2 < e_p1) then
21             w_{k+1} = w_2
22             e_p2 = e2
23         end
24     end
25     return (w_{k+1})
26 end
```

5.4 Resection-Intersection Baseado na Otimização de DFP

Esta seção descreve um algoritmo *Resection-Intersection* para estimação da pose da câmera baseado no método de otimização não linear Davidon-Fletcher-Powell (DFP) (SALIH; ABBO; ABDULLAH, 2016). A ideia por trás dos métodos Quase-Newton é fazer uma aproximação iterativa da inversa da matriz Hessiana, de forma que:

$$\lim_{i\to\infty} \mathbf{H}_i = \nabla^2 J(\mathbf{w})^{-1}. \tag{35}$$

Algorithm 4: RI baseado em otimização numérica de Davidon-Fletcher-Powell

Function *GetPosRI_DFP* ($\boldsymbol{U}_k, \boldsymbol{U}_{k+1}, \boldsymbol{w}_k, MaxTry, \gamma$) **:**
- $\mathbf{r} = (\mathbf{U}_{k+1} - \mathbf{U}_{k+1})$
- $\nabla f(\mathbf{w}_k) = \mathbf{r}/(\mathbf{w}_\Delta)$
- $\mathbf{g}_k = \nabla f(\mathbf{w}_k)^T \mathbf{r}$
- $\mathbf{H}_k = [\nabla f(\mathbf{w}_k)^T \nabla f(\mathbf{w}_k)]^{-1}$
- /**For each Hassian col: cal* $\partial \hat{U} \partial w_k[i]$ */
- **while** $k \,\epsilon\, [0, MaxTry]$ **do**
  - $\mathbf{P} = triangulate(\mathbf{w}_k, \mathbf{U}_k, \mathbf{U}_{k+1})$
  - $\hat{\mathbf{U}}_{k+1} = project(\mathbf{w}_k, \mathbf{P})$
  - $\mathbf{r}_k = vec(\hat{\mathbf{U}}_{k+1} - \mathbf{U}_{k+1})$
  - $\mathbf{d} = \sqrt{\mathbf{r}_{kx}^2 + \mathbf{r}_{ky}^2}$
  - **if** $median(\boldsymbol{d}) < \gamma$ **then**
    - $break$
  - **end**
  - $\nabla f(\mathbf{w}_{k+1}) = \mathbf{r}_k/(\mathbf{w}[k]_\Delta)$
  - $\mathbf{g}_{k+1} = \nabla f(\mathbf{w}_{k+1})^T \mathbf{r}_k$
  - $\mathbf{y}_{k+1} = \mathbf{g}_{k+1} - \mathbf{g}_k$
  - $\mathbf{H}_{k+1} = \mathbf{H}_k + \frac{\mathbf{s}_k \mathbf{s}_k^T}{\mathbf{s}_k^T \mathbf{y}_k} - \frac{\mathbf{H}_k \mathbf{y}_k \mathbf{y}_k^T \mathbf{H}_k}{\mathbf{y}_k^T \mathbf{H}_k \mathbf{y}_k}$
  - $\mathbf{d}_k = -\mathbf{H}_k \mathbf{g}_k$
  - /***Line Search** to compute optimal step size α_k *
  - $\mathbf{s}_k = \alpha_k \mathbf{d}_k$
  - $\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{s}_k$
- **end**
- **return** $\mathbf{w}_{k+1}, median(\mathbf{d})$

end

O método Davidon-Fletcher-Powell (DFP) é considerado um dos métodos de otimização mais sofisticados na solução de problemas de otimização não linear irrestrita e repre-

sentam o ápice do desenvolvimento de algoritmos otimização numérica para problemas quadráticos.

A cada passo, a inversa da Hessiana é aproximada pela soma de duas matrizes simétricas de grau 1, procedimento que é geralmente chamado de correção de grau 2 (BURDEN; FAIRES, 2004). Na mesma óptica do método GN, o algoritmo DFT foi utilizado para minimizar o erro de reprojeção no processo de estimação da pose da câmera. O Algoritmo 4 descreve o procedimento para aplicar o otimizador DFP para minimizar o erro de reprojeção.

O Algoritmo 4 recebe como entrada as matrizes $\mathbf{U}_k$ e $\mathbf{U}_{k+1}$ que contém a localização das *features* nas imagens $\mathbf{I}_k$ e $\mathbf{I}_{k+1}$. A pose da câmera estimada anteriormente($\mathbf{w}_k$), o máximo número de iterações do algoritmo ($MaxTry$) e o critério de parada(γ). Inicialmente o erro de reprojeção ($\mathbf{r}$), a Hessiana inversa ($\mathbf{H}_k$) e norma métrica ($\mathbf{g}_k$) são inicializados. A etapa de *Intersection* é realizada entre as linhas 8 e 14. Onde o erro de reprojeção para a pose da câmera atual ($\mathbf{w}_k$) é calculado e testado para ver se já está abaixo do critério de parada(γ). Em seguida, na etapa de *Resection* a direção de descida ($\mathbf{s}_k$) é calculada, com base na nova matriz hessiana ($\mathbf{H}_{k+1}$) e norma métrica ($\mathbf{g}_{k+1}$) descritas entre as linhas 15 e 21. Por fim, na linha 22, o novo ajuste ($\mathbf{s}_k$) é aplico sobre a pose da câmera ($\mathbf{w}_k$). Neste trabalho os valores de γ e $MaxTry$ utilizados foram 1×10^{-1} e 18.

5.5 Resection-Intersection Baseado na Otimização BFGS

Esta seção descreve um algoritmo *Resection-Intersection* para estimação da pose da câmera baseado no algoritmo de otimização não linear de Broyden-Fletcher-Goldfarb-Shanno (BFGS) (SALIH; ABBO; ABDULLAH, 2016), descrito na Seção 2.13. Da mesma forma que o método de otimização DFP, a ideia por trás dos métodos BFGS é fazer uma aproximação iterativa da inversa da matriz Hessiana, através da análise do gradiente da função custo por um método secante generalizado (BONNANS *et al.*, 2006). O algoritmo de otimização BFGS utilizado neste trabalho foi construído de forma similar ao algoritmo DFP, e está descrito no Algoritmo 5.

Sua principal diferença é a forma como a nova Hessiana $\mathbf{B}_k$ é calculada (linha 18), onde os vetores $\mathbf{s}_k$ e $\mathbf{y}_k$ assumem papéis inversos, conforme ilustra a Equação 36.

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{s}_k \mathbf{s}_k^T}{\mathbf{s}_k^T \mathbf{y}_k} - \frac{\mathbf{B}_k \mathbf{y}_k \mathbf{y}_k^T \mathbf{B}_k}{\mathbf{y}_k^T \mathbf{B}_k \mathbf{y}_k}. \tag{36}$$

O Algoritmo 5 recebe como entrada as matrizes $\mathbf{U}_k$ e $\mathbf{U}_{k+1}$ que contém a localização das *features* nas imagens $\mathbf{I}_k$ e $\mathbf{I}_{k+1}$. A pose da câmera estimada anteriormente($\mathbf{w}_k$), o máximo número de iterações do algoritmo ($MaxTry$) e o critério de parada(γ). Inicialmente o erro de reprojeção ($\mathbf{r}$), a Hessiana inversa ($\mathbf{B}_k$) e norma métrica ($\mathbf{g}_k$) são inicializados. A etapa de *Intersection* é realizada entre as linhas 8 e 14. Onde o erro de reprojeção para

a pose da câmera atual ($\mathbf{w}_k$) é calculado e testado para ver se já está abaixo do critério de parada(γ). Em seguida, na etapa de *Resection* a direção de descida ($\mathbf{s}_k$) é calculada, com base na nova matriz hessiana ($\mathbf{B}_{k+1}$) e norma métrica ($\mathbf{g}_{k+1}$) descritas entre as linhas 15 e 21. Por fim, na linha 22, o novo ajuste ($\mathbf{s}_k$) é aplico sobre a pose da câmera ($\mathbf{w}_k$). Neste trabalho os valores de γ e $MaxTry$ utilizados foram 1×10^{-1} e 18.

Algorithm 5: RI baseado em otimização numérica de BFGS

1 **Function** *GetPosRI_BFGS* ($\boldsymbol{U}_k, \boldsymbol{U}_{k+1}, \boldsymbol{w}_k, MaxTry, \gamma$) **:**
2 $\quad \mathbf{r} = (\mathbf{U}_{k+1} - \mathbf{U}_{k+1})$
3 $\quad \nabla f(\mathbf{w}_k) = \mathbf{r}/(\mathbf{w}_\Delta)$
4 $\quad \mathbf{g}_k = \nabla f(\mathbf{w}_k)^T \mathbf{r}$
5 $\quad \mathbf{B}_k = [\nabla f(\mathbf{w}_k)^T \nabla f(\mathbf{w}_k)]^{-1}$
6 $\quad$ /**For each Hassian col: cal* $\partial \hat{U} \partial w_k[i]$ */
7 $\quad$ **while** $k \, \epsilon \, [0, Maxtry]$ **do**
8 $\quad\quad \mathbf{P} = triangulate(\mathbf{w}_k, \mathbf{U}_k, \mathbf{U}_{k+1})$
9 $\quad\quad \hat{\mathbf{U}}_{k+1} = project(\mathbf{w}_k, \mathbf{P})$
10 $\quad\quad \mathbf{r}_k = vec(\hat{\mathbf{U}}_{k+1} - \mathbf{U}_{k+1})$
11 $\quad\quad \mathbf{d} = \sqrt{\mathbf{r}_{kx}^2 + \mathbf{r}_{ky}^2}$
12 $\quad\quad$ **if** $median(\boldsymbol{d}) < \gamma$ **then**
13 $\quad\quad\quad break$
14 $\quad\quad$ **end**
15 $\quad\quad \nabla f(\mathbf{w}_{k+1}) = \mathbf{r}_k/(\mathbf{w}[k]_\Delta)$
16 $\quad\quad \mathbf{g}_{k+1} = \nabla f(\mathbf{w}_{k+1})^T \mathbf{r}_k$
17 $\quad\quad \mathbf{y}_{k+1} = \mathbf{g}_{k+1} - \mathbf{g}_k$
18 $\quad\quad \mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{s}_k \mathbf{s}_k^T}{\mathbf{s}_k^T \mathbf{y}_k} - \frac{\mathbf{B}_k \mathbf{y}_k \mathbf{y}_k^T \mathbf{B}_k}{\mathbf{y}_k^T \mathbf{B}_k \mathbf{y}_k}$
19 $\quad\quad \mathbf{d}_k = -\mathbf{B}_k^{-1} \mathbf{g}_k$
20 $\quad\quad$ /***Line Search** to compute optimal step size α_k *
21 $\quad\quad \mathbf{s}_k = \alpha_k \mathbf{d}_k$
22 $\quad\quad \mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{s}_k$
23 $\quad$ **end**
24 $\quad$ **return** $\mathbf{w}_{k+1}, median(\mathbf{d})$
25 **end**

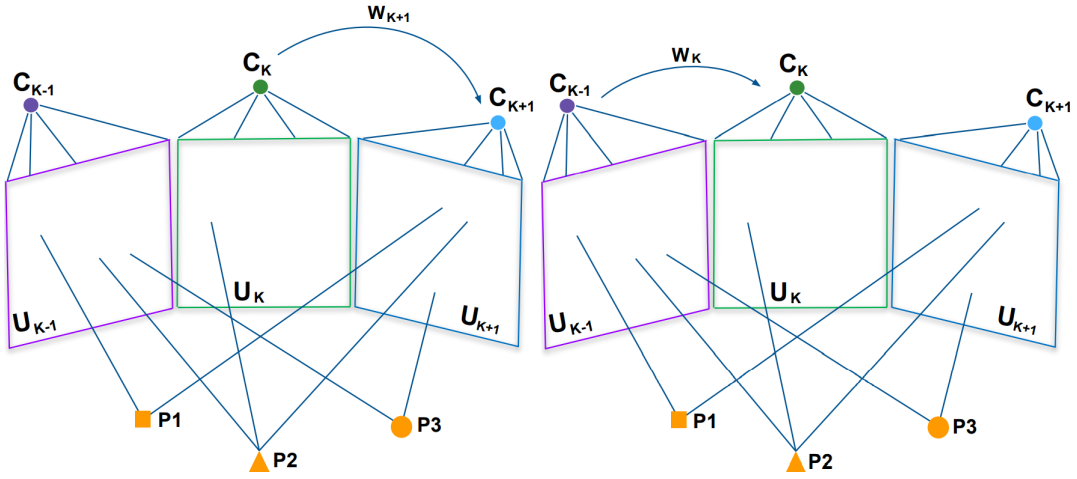
5.6 Resection-Intersection com Refinamento Cíclico

O número de *features* utilizado para estimar a pose da câmera tem impacto direto na sua precisão. Em um cenário com muitas árvores, mal iluminado ou pouco estático,

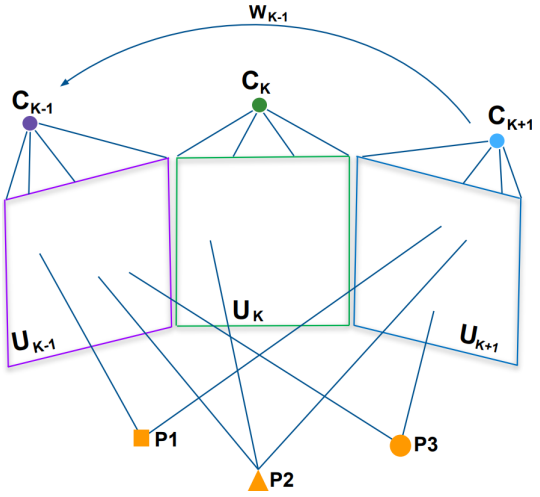
muitas das *features* extraídas podem ser classificadas como *outliers*, e acabam não sendo utilizadas no processo de estimação da pose da câmera.

O refinamento cíclico consiste em um método que tenta otimizar um conjunto de três poses de câmera utilizando uma combinação de três pares de imagens em um ciclo realimentado de estimativas sucessivas. Este método aumenta significativamente o número de *features*, ao mesmo tempo em que melhora as classificações de *features* ruins, resultando em uma estimativa de pose de câmera mais precisa (GEIGER; LENZ; URTASUN, 2012).

Figura 29 – Ilustração do ciclo de operação do método de refinamento cíclico.



(a) Primeiro passo do Refinamento Cíclico. (b) Segundo passo do Refinamento Cíclico.



(c) Terceiro passo do Refinamento Cíclico.
Fonte: Adaptado de (PEREIRA, 2018).

Inicialmente, o método recebe como entrada a última pose da câmera estimada ($\mathbf{w}_k$) e as coordenadas (u, v) das *features* extraídas nas três imagens de entrada, porém, somente as *features* que foram respectivamente detectadas nas três imagens são utilizadas. Essa filtragem resulta em uma diminuição das *features* que sofrem oclusões eminentes. As *features* são agrupadas em três pares únicos: ($\mathbf{U}_k$, $\mathbf{U}_{k+1}$), ($\mathbf{U}_{k-1}$, $\mathbf{U}_k$) e ($\mathbf{U}_{k+1}$, $\mathbf{U}_{k-1}$).

O processo de refinamento cíclico utilizado neste trabalho é ilustrado na Figura 29. Na figura, as matrizes $\mathbf{C}_{k-1}$, $\mathbf{C}_k$ e $\mathbf{C}_{k-1}$ representam três câmeras centradas, e equivalem a três imagens únicas tiradas com a mesma câmera em três momentos diferentes. Todas as três câmeras visualizam três pontos na cena ($\mathbf{P1}, \mathbf{P2}, \mathbf{P3}$) que representam as *features* extraídas e localizadas nessas três imagens.

No primeiro passo, (a), a posição relativa da câmera $\mathbf{C}_k$ para a câmera $\mathbf{C}_{k+1}$ é estimada utilizando o método RI. O método RI recebe como entrada o primeiro conjunto de *features* ($\mathbf{U}_k$, $\mathbf{U}_{k+1}$) e a última pose da câmera estimada ($\mathbf{w}_k$) e gera uma nova pose da câmera denominada $\mathbf{w}_{k+1}$. No segundo passo, (b), a posição relativa da câmera $\mathbf{C}_{k-1}$ para a câmera $\mathbf{C}_k$ é estimada utilizando o método RI. O método RI recebe como entrada o segundo conjunto de *features* ($\mathbf{U}_{k-1}$, $\mathbf{U}_k$) e a pose da câmera estimada no passo anterior, (a), e gera uma nova pose da câmera ($\mathbf{w}_k$).

No terceiro passo, (c), a posição relativa da câmera $\mathbf{C}_{k+1}$ para a câmera $\mathbf{C}_{k-1}$ é estimada utilizando o método RI. O método RI recebe como entrada o terceiro e último conjunto de *features* ($\mathbf{U}_{k+1}$, $\mathbf{U}_{k-1}$) e a pose da câmera estimada no segundo passo (b). O resultado dessa operação é uma nova pose da câmera ($\mathbf{w}_{k-1}$). O Algoritmo 6 descreve a implementação do método RI na configuração de refinamento cíclico desenvolvido neste trabalho.

Algorithm 6: Algoritmo refinamento cíclico

```
1  Function CY_RI (U_{k-1}, U_k, U_{k+1}, w_k, MaxTry, γ) :
2      n_cyc = 0,  w_{k-1} = w_k
3      while (n_cyc < MaxTry) do
4          w_{k+1}, err_1 = GetPosRI(U_k, U_{k+1}, w_{k-1})
5          w_k, err_2 = GetPosRI(U_{k-1}, U_k, w_{k+1})
6          w_{k-1}, err_3 = GetPosRI(U_{k+1}, U_{k-1}, w_k)
7          if err_1 < γ then
8              break;
9          end
10         n_cyc = n_cyc + 1
11     end
12     return (w_{k+1})
13 end
```

O Algoritmo 6 recebe como entrada as matrizes $\mathbf{U}_k$ e $\mathbf{U}_{k+1}$ que contém a localização das *features* nas imagens $\mathbf{I}_k$ e $\mathbf{I}_{k+1}$. A pose da câmera estimada anteriormente($\mathbf{w}_k$), o máximo número de iterações do algoritmo ($MaxTry$) e o critério de parada(γ). O Algoritmo começa estimando uma nova pose da câmera na linha 4, utilizando o algoritmo do método RI apresentado nas seções anteriores. O algoritmo RI recebe como entrada as *features* ex-

traídas nas imagens $k-1$, k e $k+1$ e a pose da câmera estimada anteriormente pelo sistema ($\mathbf{w}_k$). O resultado dessa operação é uma nova pose da câmera, denominada $\mathbf{w}_{k+1}$. Em seguida, uma segunda pose da câmera é estimada na linha 5, utilizando as *features* extraídas nas imagens $k-1$ e k e a pose da câmera inversa estimada no passo anterior ($\mathbf{w}_{k+1}$). A nova pose é denominada de ($\mathbf{w}_k$).

Por fim, uma terceira e última pose da câmera ($\mathbf{w}_{k-1}$) é estimada na linha 6, utilizando as *features* extraídas nas imagens $k+1$ e $k-1$ e a pose da câmera inversa estimada anteriormente ($\mathbf{w}_k$). O algoritmo verifica se o erro de reprojeção err_1 caiu abaixo do valor de threshold γ, e caso verdadeiro o ciclo é interrompido, senão o ciclo continua até o valor máximo de iteração $MaxTry$ ser alcançado. A pose estimada que apresentar o menor erro reprojeção é a selecionada. A função $GetPosRI$ utilizado neste algoritmo foi o apresentado na Seção 5.2. Os valores de γ e $MaxTry$ utilizados foram 1×10^{-1} e 18.

6 RESULTADOS

Este capítulo apresenta os resultados obtidos na validação do sistema proposto e dos estudos realizados. Todos os algoritmos utilizados durante a etapa de validação foram implementados utilizando a linguagem de programação Python, e embarcados em uma placa Raspberry Pi 4B para realizar a validação do sistema proposto neste trabalho.

A validação foi realizada utilizando o banco de dados KITTI, assim como suas métricas de avaliação: erro de translação em relação à distância média percorrida, erro de translação em relação à velocidade média, erro de rotação em relação à distância média percorrida, erro de rotação em relação à velocidade média. A principal métrica de avaliação definida neste trabalho foi o erro de translação em relação à distância média percorrida, pois este reflete diretamente na precisão da posição estimada da câmera. Todas as variações do sistema RI proposto foram avaliadas de forma a possibilitar uma inferência individual de cada método.

Para facilitar a identificação das variações do sistema RI implementado, cada variação foi nomeada da seguinte forma: o sistema RI baseado na otimização numérica de Gauss-Newton foi nomeado como GN_RI, o baseado na otimização numérica de Levenberg-Marquardt nomeado como LM_RI, o baseado na otimização numérica de Davidon–Fletcher– Powell foi nomeado como DFP_RI e o baseado na otimização numérica de Broyden–Fletcher–Goldfarb–Shanno foi nomeado como BFGS_RI. Por fim, o sistema RI com refinamento cíclico e otimização numérica de Levenberg-Marquardt foi nomeado como CY_RI.

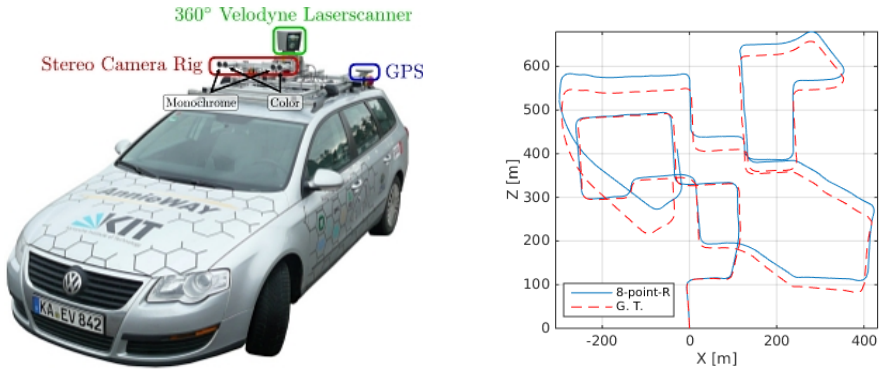
Alguns valores constantes foram ajustados, tanto na unidade de Features quanto na unidade de Escala (Figure 21). Esses parâmetros estão relacionados à zona de solo e dependem do gradiente de imagem aceitável, posição e resolução da câmera, posição do plano de solo e altura da câmera. O ajuste desses parâmetros é discutido na Seção 6.4.

Nas próximas seções, são apresentados o banco de dados utilizado no desenvolvimento e validação deste trabalho, o ambiente de desenvolvimento, o hardware utilizado nos testes de desempenho seguido pela seção que descreve a calibração inicial do sistema e os resultados da validação dos sistemas propostos.

6.1 Banco de Dados KITTI

As imagens de cenas reconstruídas e os caminhos estimados forneçam uma ideia subjetiva da precisão de um algoritmo 3D. Para avaliar objetivamente a precisão de um sistema utilizamos neste trabalho como referência um trajeto instrumentado disponível num conjunto de dados público que fornece as imagens da câmera embarcada em um veículo e também o registro da posição do veículo obtido de instrumentos. Além disso, o sistema aceita o resultado de propostas de algoritmos onde os autores postam os resultados de suas implementações para avaliação de desempenho. Neste trabalho, o banco de dados KITTI (GEIGER; LENZ; URTASUN, 2012) foi escolhido como aplicação alvo.

Figura 30 – Tipos de Odometria Visual e exemplo de estimação de trajetória.



(a) Veículo equipado com câmera e GPS.

(b) Trajetória estimada por Odometria Visual

Fonte: (GEIGER; LENZ; URTASUN, 2012).

O conjunto de dados elaborado para validação e desenvolvimento de aplicações de Odometria Visual é composto por uma coleção de 22 trajetórias em que um carro equipado com câmeras e um GPS de alta precisão percorre cidades e rodovias. As trajetórias são armazenadas como uma sequência de imagens, obtidas a uma taxa de dez quadros por segundo. Para cada imagem, a posição e orientação da câmera é obtida através de GPS e é salva com a finalidade se obter um *ground truth* do percurso. Porém, para apenas onze dessas trajetórias o *ground truth* é fornecido, reservando as outras onze somente para avaliação.

As trajetórias são calculadas usando as imagens fornecidas e os resultados estimados podem ser enviados para o site dos desenvolvedores do banco de dados (GEIGER; LENZ; URTASUN, 2012). Erros de rotação e translação são publicados e podem ser usados para comparação com outras soluções.

As métricas de avaliação propostas pelos desenvolvedores do banco de dados são o erro de rotação em graus por metro [deg/m] e os erros de translação em relação à porcenta-

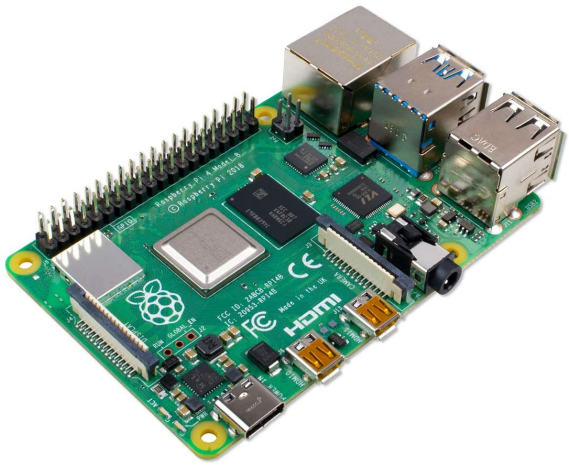
gem total [%] em relação à distância média percorrida e velocidade média. As trajetórias são divididas em segmentos de 100 a 800 metros e os erros são calculados sobre todos os segmentos de trajetória. O veículo utilizado para criar o banco de dados KITTI é ilustrado na Figura 30 (a) e uma típica trajetória obtida com o processo de Odometria Visua, onde a linha vermelha representa a trajetória estimada por GPS e a linha azul representa a posição estimada da câmera, está ilustrada na Figura 30 (b).

O banco de dados utilizado neste trabalho foi obtido diretamente na página web do KITTI, disponível em (GEIGER ANDREAS LENZ PHILIP, 2020). Até a conclusão deste texto, o dataset era de versão única.

6.2 Raspberry Pi

Raspberry Pi é uma série de computador, onde o hardware é integrado numa única placa, desenvolvida no Reino Unido pela Fundação Raspberry Pi (FOUNDATION, 2021). É popularmente chamado de Minipc, porque pode ser conectado a um monitor de computador ou TV, e a um teclado e um mouse padrão, proporcionando a experiência de se utilizar um desktop, ao mesmo tempo que proporciona um ambiente para execução e teste de softwares desenvolvidos. Raspberry Pi 4B está ilustrada na Figura 31.

Figura 31 – Ilustração da Raspberry Pi 4 modelo B.



Fonte: (FOUNDATION, 2021).

Nesse trabalho, uma Raspberry Pi 4B foi utilizada com o objetivo de avaliar o desempenho do sistema proposto em um hardware embarcado.

O Raspberry Pi 4 modelo B é baseado em um System-On-a-Chip (SoC) Broadcom BCM2711, que inclui um processador Cortex-A72 (ARM v8) 64-bit de 1.5GHz, GPU VideoCore VI, e 4GB MB de memória RAM (em sua última revisão) e uma entrada de

cartão SD para armazenamento de dados. O *firmware* é de código fechado, porém há versões não oficiais de código aberto. As principais especificações técnicas do modelo de Raspberry Pi 4 utilizado na validação deste trabalho estão descritas na Tabela 3.

Tabela 3 – Tabela de características da Raspberry Pi 4B

Features/Spec	**Raspberry Pi 4 Model B**
Release Date	24th June 2019
SoC Type (Processor)	Broadcom BCM2711 (with metal cover)
Core Type	Cortex-A72 64-bit (ARMv8)
No. of Cores	Quad-Core
GPU	VideoCore VI
Multimedia	H.265 decode (4Kp60) H.264 decode (1080p60) H.264 encode (1080p30) OpenGL ES 1.1, 2.0,3.0 Graphics
CPU Clock	1,5 GHz
Memory/OS Storage	microSD
RAM	4GB
Ethernet	True Gigabit Ethernet
USB Port	2 xUSB 3.0 + 2 x USB 2.0
HDMI	2 x micro HDMI support Dual Display
WiFi	802.11 b/g/n/ac (2,4 GHz + 5GHz & Shielded)
Bluetooth	5.0 + BLE (Shielded)
Antenna	PCB Antenna
GPIO	40 pins (fully backwards-compatible with previous boards)
Operation System	Raspian (>24 June 2019)
Dimension	85mm x 56mm
Power Input	5 V via USB Type C (upto 3A) 5V via GPIO header (upto 3A) Power over Ethernet, requires PoE HAT

Fonte: Adaptado de (FOUNDATION, 2021).

A Raspberry Pi é compatível com sistemas operativos baseados em GNU/Linux e Windows 10 IoT. Contudo, a Fundação Raspberry Pi disponibiliza o seu próprio sistema operacional denominado Raspberry Pi OS ou Raspbian, uma distribuição GNU/Linux de 32 bit baseada em Debian. A fundação também disponibiliza sistemas operativos fornecidos por terceiros, tais como Ubuntu, Windows 10 IoT Core, RISC OS, e LibreELEC (distribuição especializada para central multimídia) (FOUNDATION, 2021).

As linguagens de programação recomendadas são Python e Scratch, porém, qualquer linguagem que possa ser compilada na arquitetura ARMv6 pode ser usada para o desenvolvimento de software.

Por fim, a Raspberry Pi foi utilizada neste trabalho visando a aplicação prática do sistema OVM proposto. Foi escolhida por apresentar um bom desempenho em termos de processamento, aliado a um baixo consumo de energia. E por ser tratar de um hardware conhecido e de preço acessível, pode ser facilmente utilizado na comparação com futuros trabalhos.

6.3 Ambiente de Desenvolvimento

O sistema desenvolvido neste trabalho foi desenvolvidos inteiramente na linguagem Python (VAN ROSSUM; DRAKE, 2009). As bibliotecas de desenvolvimentos utilizadas foram NumPy, Matplotlib e OpenCV.

OpenCV (BRADSKI, 2000) é uma biblioteca multiplataforma de código aberto. para o uso acadêmico e comercial, para o desenvolvimento de software na área de Visão computacional. Possui diversos módulos de Processamento de Imagens para Estrutura de dados, Álgebra Linear, Filtros de imagem, calibração de câmera, reconhecimento de objetos, análise estrutural entre outros.

NumPy (HARRIS *et al.*, 2020) é uma biblioteca de código aberto para a linguagem de programação Python, que suporta o processamento de matrizes multi-dimensionais, juntamente com uma coleção de funções matemáticas de alto nível para operar sobre estas matrizes. Matplotlib (HUNTER, 2007) é uma biblioteca de software aberto utilizada para criação de gráficos e visualizações de dados em geral.

6.4 Calibração Inicial do Sistema

No decorrer deste trabalho, os parâmetros de ajuste dos métodos e algoritmos utilizados foram definidos. De forma complementar, esta seção descreve o ajuste inicial dos demais parâmetros dos sistema.

Os seguintes parâmetros foram usados na inicialização do sistema para a realização da validação: a pose inicial da câmera foi definida como $w = (0, 0, 0, 0, 0, 1)$ e estabelecido que a direção de movimento inicial da câmera é sempre para a frente. Na etapa de estimativa de pose da câmera, o *loop* de estimação dos algoritmos RI para quando a mediana do erro de projeção cai abaixo de $0,1$ píxeis. Este valor foi definido experimentalmente com base na mediana do erro de reprojeção obtido durante os testes nos onze primeiros cenários do banco de dados.

O algoritmo de detecção de *features* Shi-tomasi recebe como configuração de entrada o tamanho de janela de 3×3, um nível de qualidade de $0,01$ e uma distância mínima

de 15 píxeis. O algoritmo Lucas-Kanade utilizado foi configurado com 3 níveis de pirâmide e um tamanho de janela de 21×21. Todos esse valores foram definidos de forma experimental. O número médio de *features* detectadas gira em torno de 100 e 460 por imagens.

No conjunto de dados KITTI, a pose da câmera gravada por GPS e os parâmetros de calibração do sistema utilizados na criação do dataset são fornecidos para as primeiras onze sequências e utilizados para ajustar os parâmetros a seguir. Os parâmetros de calibração da câmera foram definidos como $f_x = 400$, $f_y = 400$, $c_x = 160$, $c_y = 120$. A altura da câmera e relação ao solo foi definida como $1,65$ metros. O *ground truth* fornecido é utilizado na imagens de resultado Capítulo 8, de forma a permitir uma comparação visual da trajetória estimada pelos sistemas propostos e o valor medido por GPS fornecido.

6.5 Validação do Sistema Proposto

Esta seção apresenta os resultados alcançados pelos sistemas propostos neste trabalho. A avaliação foi realizada utilizando os onze primeiros cenários do banco de dados KITTI, uma vez que os *ground truth* necessários para realizar a avaliação de desempenho somente estão disponíveis para esses onze primeiros cenários.

As métricas de avaliação utilizadas são as mesmas recomendadas pelo KITTI, sendo: erro de translação da pose da câmera em relação à distância média percorrida, erro de translação da pose da câmera em relação à velocidade média, erro de rotação da pose da câmera em relação à distância média percorrida e erro de rotação da pose da câmera em relação à velocidade média. O objetivo é obter o menor erro possível nessas quatro métricas. Dentre essas métricas citadas, as de maior peso na literatura são o erro de translação e rotação em relação à distância média percorrida.

O trabalho de Pereira et al. (PEREIRA *et al.*, 2018) representa, hoje, o trabalho de Odometria Visual Monocular baseado no conceito de *Resection-Intersection* com os melhores resultados no banco de dados KITTI, tendo atingido um erro de translação em relação à distância média percorrida igual a $1,02\%$ e um erro de rotação em relação à distância média percorrida igual a $0,0028deg/m$. Dessa forma, ele será utilizado neste trabalho como marco de comparação com o sistema proposto aqui. Será denominado RI_MVO e representado nos gráficos de resultados na forma de uma linha tracejada na cor verde.

Todos os sistemas RI propostos neste trabalho foram avaliados e são comentados a seguir. Os resultados obtidos pelo sistema RI utilizando os algoritmos de otimização numérica de Gauss-Newton (GN), Levenberg-Marquardt (LM), Davidon–Fletcher–Powell (DFP) e Broyden–Fletcher–Goldfarb–Shanno (BFGS) estão descritos na Tabela 4 e foram obtidos utilizando os onze primeiros cenários do banco de dados KITTI. Os resultados apresentados na Tabela 4 foram gerados utilizando a ferramenta fornecida pelo banco

Tabela 4 – Resultados utilizando os onze primeiros cenários do KITTI.

Seq	**NIm**	**T[%]**					**R[deg.10^{-3}/m]**				
		GN	**LM**	**DFP**	**BFGS**	**CY**	**GN**	**LM**	**DFP**	**BFGS**	**CY**
00	4540	1,09	0,90	0,86	0,82	0,73	2,86	2,86	2,70	2,71	2,81
01	1100	1,20	1,15	1,00	1,03	1,25	3,15	3,09	2,95	2,73	2,75
02	4660	0,89	0,89	0,84	0,84	0,85	2,11	2,18	2,19	2,31	2,12
03	800	1,12	1,00	0,90	0,95	1,03	2,97	2,86	2,56	2,56	2,06
04	270	1,12	0,96	1,10	1,08	1,04	2,40	2,58	1,56	1,7	2,75
05	2760	0,61	0,64	0,65	0,65	0,56	2,06	2,35	2,19	2,12	1,78
06	1100	0,60	0,61	0,66	0,66	0,66	2,17	2,46	2,14	2,09	2,29
07	1100	0,60	0,67	0,67	0,68	0,45	3,43	2,69	2,89	2,62	2,46
08	4070	1,03	0,97	0,98	0,95	0,99	2,75	2,75	2,70	2,61	2,64
09	1590	0,90	0,89	0,91	0,95	0,95	1,83	2,06	1,71	1,73	1,78
10	1200	1,27	1,31	1,64	1,34	1,35	2,23	2,41	2,14	2,22	2,35
Média		**0,95**	**0,89**	**0,88**	**0,88**	**0,86**	**2,57**	**2,57**	**2,52**	**2,52**	**2,34**

de dados. Na tabela, o sulfixo "_RI" dos nomes dos sistema propostos foi omitido para simplificação de notação.

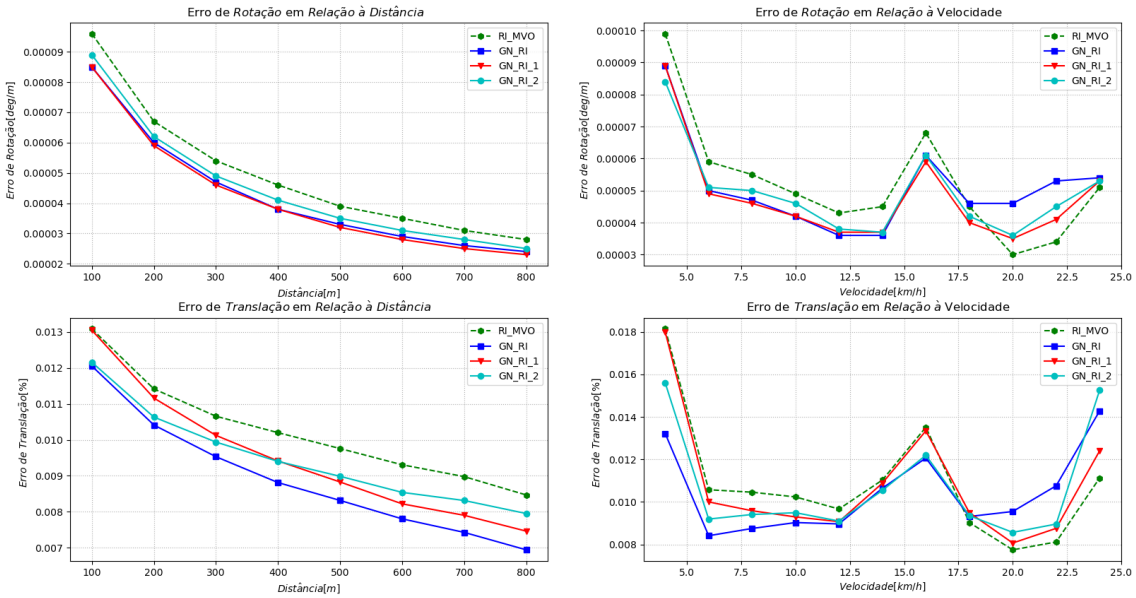
A primeira coluna (Seq) da tabela representa o cenário do banco de dados KITTI. A segunda coluna (NIm) informa o número de imagens em cada um desses cenários. A coluna T[%] representa o erro de translação em relação à distância média dado em porcentagem e R[deg.10^{-3}/m] representa o erro de rotação em relação à distância média percorrida, dado em milésimos graus por metro.

Através da análise dos resultados obtidos pelo sistema GN_RI é possível verificar uma grande estabilidade no erro de rotação em relação à distância média percorrida ao longo de todos os cenários, tendo alcançado um erro de rotação médio de $0,00257deg/m$. O erro de translação em relação à distância média percorrida obtido foi de $0,957\%$, o que representa uma melhoria significativa em relação a outros trabalhos baseados na abordagem de RI (PEREIRA *et al.*, 2018), porém, não tão estável ao longo de todos os cenários como o obtido no erro de rotação. Na Figura 32 estão ilustrados quatro gráficos de erros obtidos com o sistema GN_RI.

No gráfico superior esquerdo, está ilustrado o erro de rotação da pose estimada em relação à distância percorrida pelo veículo. No gráfico superior direito, está ilustrado o erro de rotação da pose estimada em relação à velocidade do veículo. No gráfico inferior esquerdo, está ilustrado o erro de translação da pose estimada em relação a distância percorrida pelo veículo e no gráfico inferior direito, está ilustrado o erro de translação da pose estimada em relação à velocidade do veículo.

Cada um dos gráficos da figura contém quatro curvas de erro. A primeira curva, em

Figura 32 – Resultados obtidos utilizando o sistema GN_RI.



Fonte: do autor.

verde tracejado, representa o resultado obtido pelo sistema RI_MVO proposto por Pereira et al. (PEREIRA *et al.*, 2018). A segunda curva, em azul claro, representa o sistema GN_RI proposto neste trabalho utilizando a etapa de Pré-Processamento de Imagem descrita na Seção 4.1, porém, sem o teste de convergência com região de busca expandida proposto na Seção 5.3. A terceira curva, em vermelho, representa o sistema GN_RI proposto neste trabalho utilizando o teste de convergência com região de busca expandida, porém, sem a etapa de Pré-Processamento de Imagem. Por fim, a quarta e última curva, em azul escuro, representa o sistema GN_RI proposto neste trabalho utilizando o teste de convergência com região de busca expandida e a etapa de Pre-Processamento de Imagem.

Por inspeção, é possível verificar que tanto o inclusão da etapa de Pré-Processamento quanto a do teste de convergência com região de busca expandida propostos resultaram em uma melhoria, de forma individual, em todos as métricas de erro, principalmente em relação ao erro de translação e rotação em relação à distância. Contudo, os erros de translação e rotação em relação à velocidade apresentaram uma piora para velocidade acima de 17,5 km/h, porém, indicam uma tendência de correção em velocidade maiores. Através da combinação da etapa de Pré-Processamento e do teste de convergência com região de busca expandida foi possível obter resultados ainda melhores. Os resultados dessa combinação sugerem uma rápida convergência do sistema proposto em relação ao sistema RI_MVO, tanto no erro de translação quanto no erro de rotação em relação à distância média percorrida.

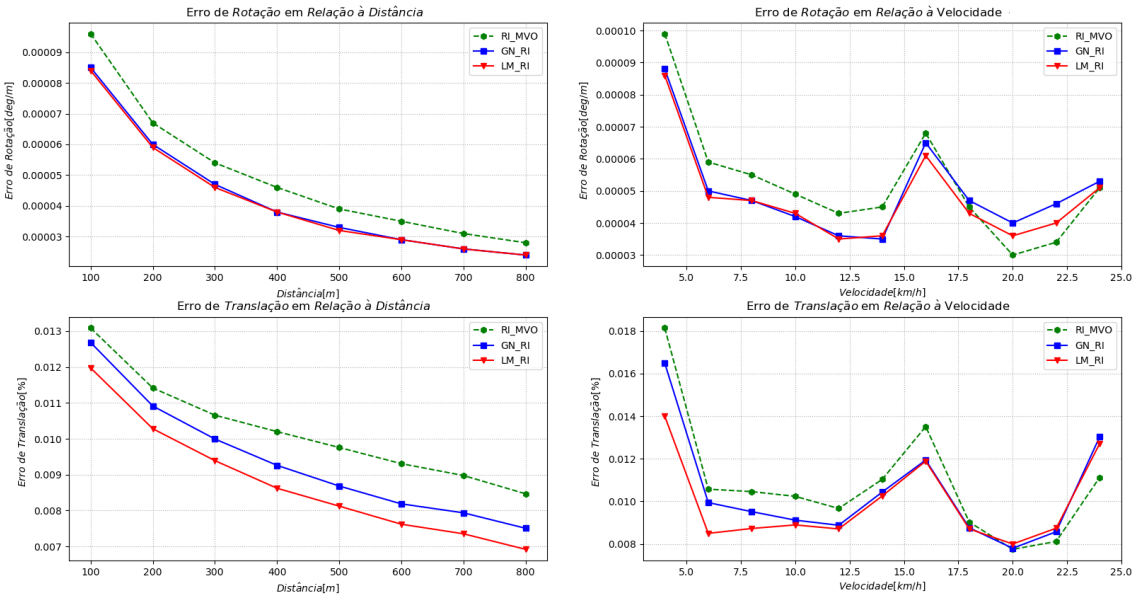
Um problema comum em abordagens RI baseado na otimização de GN é a divergência

quando o passo de otimização é muito grande. O sistema RI com a otimização LM foi implementado para contornar esse problema.

Com base na Tabela 4, é possível verificar que os resultados obtidos pelo sistema LM_RI, assim como o sistema GN_RI, apresentaram uma grande estabilidade no erro de rotação em relação à distância média percorrida ao longo de todos os cenários, tendo alcançado um erro de rotação médio de $0,00252 deg/m$. Contudo, o erro de translação em relação à distância média percorrida obtido foi $0,898\%$, atestando a superioridade do algoritmo LM em relação ao GN, no que diz respeito ao problema de estimação da pose da câmera em sistemas baseados no conceito RI.

Os resultados ilustrados na Figura 33 mostram que a taxa da descida da curva do erro de translação foi significativamente aprimorada em relação ao sistema GN_RI (curva em azul escuro na figura). A mesma oscilação na precisão do erro de translação e rotação em relação à velocidade foi identificada, porém, em uma intensidade menor, confirmando outro efeito benéfico do sistema RI_LM em relação ao GN_RI.

Figura 33 – Resultados obtidos utilizando o sistema LM_RI.

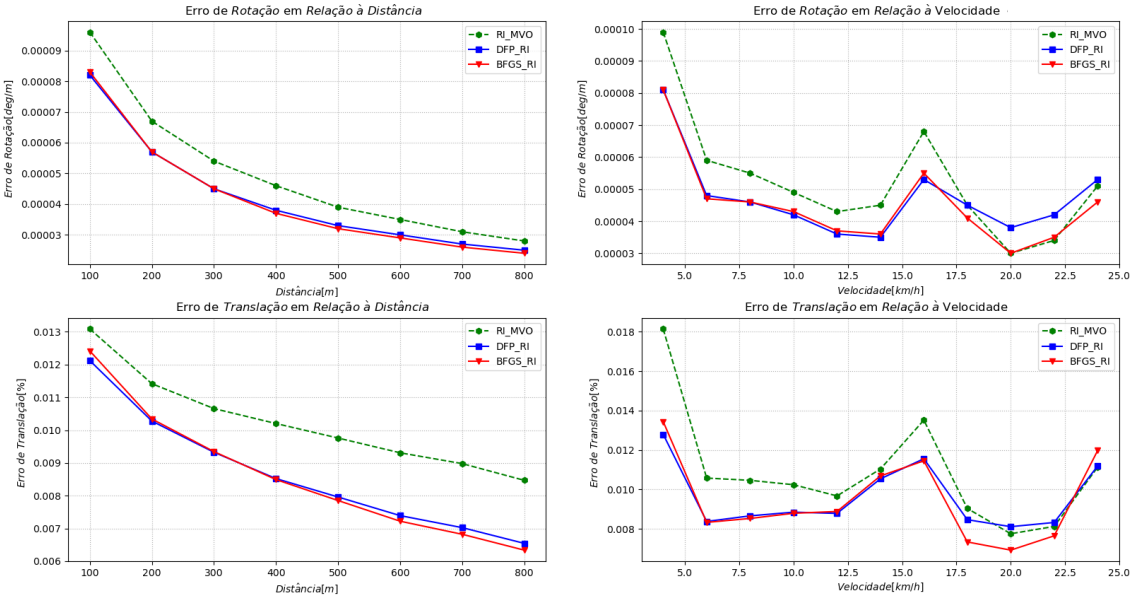


Fonte: do autor.

Os sistemas DFP_RI e BFGS_RI obtiveram resultados bastante similares, o que indica que a estabilidade do tamanho de passo do algoritmo de otimização (matriz Hessiana), principal diferencial do algoritmo de otimização BFGS em relação ao DFP, tem pouco ou nenhum efeito no resultado final de otimização da pose da câmera. Contudo, os resultados obtidos foram superiores aos obtidos com os sistemas GN_RI e LM_RI, onde os métodos DFP_RI e BFGS_RI obtiveram erros de translação em relação à distância média percorrida igual a $0,886\%$ e $0,883\%$. Conforme ilustra a Figura 34, o erro de rotação em relação

à distância média percorrida obtido pelo método DFP_RI foi semelhante aos alcançados pelo GN_RI e LM_RI, contudo, a oscilação do erro de rotação e translação em relação à velocidade foi levemente aumentada. Por outro lado, o método BFGS_RI foi capaz de atenuar o erro de rotação e translação em relação à velocidade média, obtendo de forma geral, resultados notavelmente melhores que os demais apresentados até o momento.

Figura 34 – Resultados obtidos utilizando o sistema DFP_RI e BFGS_RI.



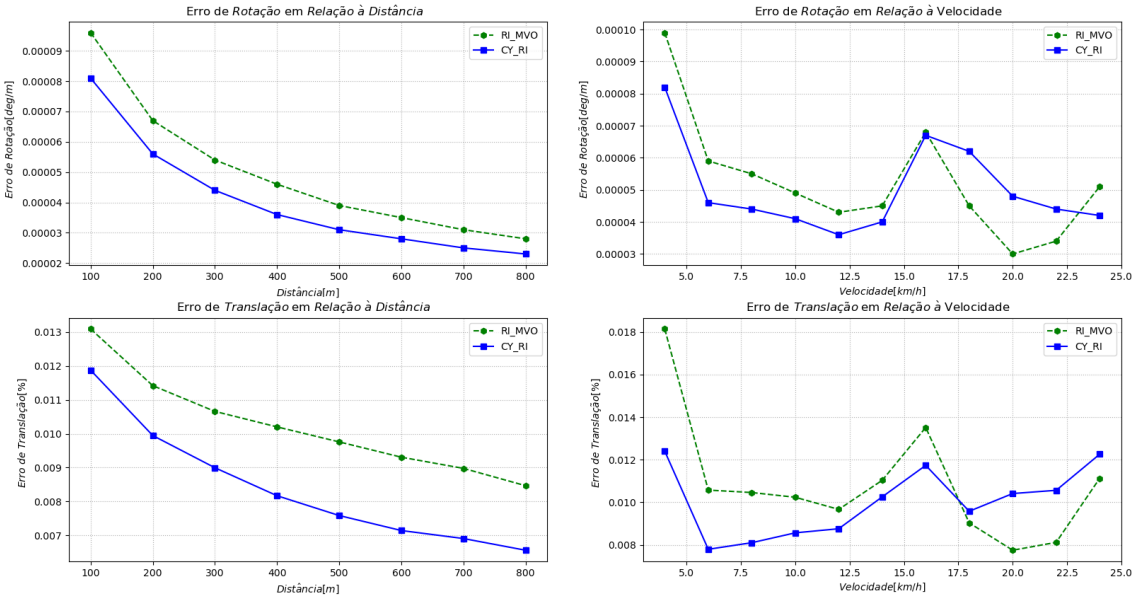
Fonte: do autor.

Por fim, a coluna CY da Tabela 4 e a Figura 35 mostra os resultados obtidos com o sistema RI com otimização numérica de LM e refinamento cíclico (CY_RI). Os resultados obtidos foram os melhores em termos de precisão da estimativa da pose da câmera alcançados neste trabalho. Comprovando a eficácia do refinamento cíclico na potencialização dos resultados de estimação da pose da câmera. Os erros de translação e rotação em relação à distância média percorrida obtidos foram respectivamente de $0,86\%$ e o $0,0024deg/m$. Por inspeção da Figura 35 constatou-se que o refinamento cíclico também magnificou a oscilação positiva do erro em velocidade acima de $17km/h$. Entretanto, demonstrou uma tendência de diminuição dessa oscilação em velocidade acima de $22,5km/h$.

A Figura 36 ilustra os resultados de todos os métodos. Por inspeção, é possível verificar que todos os métodos sofrem com uma perturbação no erro de translação e rotação em relação à velocidade média, quando o veículo trafega na faixa de velocidade de $15km/h$. Também é possível verificar que o método CY_RI é o mais sensível a essa perturbação, porém, demonstra uma tendência de recuperação um pouco maior que os demais métodos.

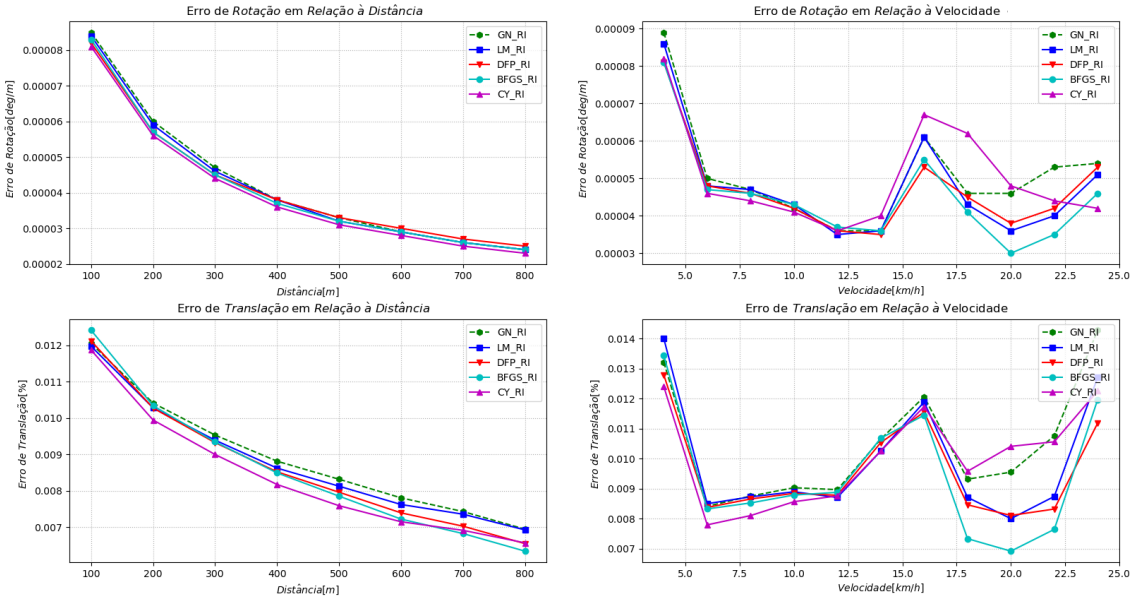
Uma possível explicação para essa pertubação pode estar relacionada a vibração me-

Figura 35 – Resultados obtidos utilizando o sistema CY_RI.



Fonte: do autor.

Figura 36 – Ilustração dos resultados de todos os métodos.



Fonte: do autor.

cânica do veículo sobre o suporte da câmera instalada nessa faixa de velocidade. Outra possível explicação, embora pouco provável, pode não estar relacionada à velocidade do veículo e sim a um trecho do cenário do banco de dados em que o veículo trafega utilizando essa faixa de velocidade.

Figura 37 – Exemplo de trechos desafiadores.

(a)

(b)
Fonte: (TRIGGS *et al.*, 2000).

Dentre os trechos mais desafiadores do KITTI está o indicado na Figura 37 (a), onde um caminhão corta a frente do veículos equipado com a câmera enquanto ele estava em movimento, resultando em uma perturbação no erro de rotação. Outro trecho desafiador pode ser observado quando o veículo trafega em alta velocidade por uma rodovia com guarda-corpo, conforme ilustra a Figura 37 (b). Os guarda-corpos por serem quase idênticos entre si, dificultam o processo de rastreamento de *features* e eliminação de *outliers*. Por fim, algumas das trajetórias estimadas pelos sistemas propostos estão ilustradas nas Figuras 39, 40, 41 e 42 no Apêndice.

6.6 Avaliação do Esforço Computacional

Esta seção apresenta uma análise do esforço computacional do sistema proposto neste trabalho. A Tabela 5 apresenta o tempo de processamento médio e o número médio de *features* detectadas em cada um dos onze primeiros cenários do banco de dados KITTI. A coluna "Seq" indica o nome de sequências do KITTI, a coluna "NIm" indica o número de imagens nas sequências, a coluna "Custo Computacional Médio" mostra o tempo médio de processamento para cada sequência e a coluna "Número Médio de Features" o número médio de *features* processadas em cada sequência.

A simulações foram realizadas utilizando o conjunto das onze primeiras sequências do KITTI em um Intel Core i5@2,3 GHz. Com base nos resultados apresentados na Tabela 5, é possível verificar que o custo de processamento médio do método LM_RI foi quase idêntico ao do método GN_RI, e que quando somado ao seu desempenho na avaliação com o banco de dados do KITTI na seção anterior, fortalece ainda mais a constatação da

Tabela 5 – Tempo de execução médio obtido nos onze primeiros cenários do KITTI.

Seq	**NIm**	**Custo Computacional Médio[s]**				
		GN	**LM**	**DFP**	**BFGS**	**CY**
00	4540	0,07826	0,07940	0,1022	0,1057	0,2165
01	1100	0,07583	0,07590	0,1116	0,1058	0,2204
02	4660	0,07767	0,07757	0,1057	0,1074	0,2289
03	800	0,07404	0,07382	0,1064	0,1081	0,2376
04	270	0,07277	0,07335	0,1101	0,1120	0,2243
05	2760	0,07452	0,07453	0,1026	0,1042	0,1941
06	1100	0,06866	0,06836	0,1059	0,1063	0,1957
07	1100	0,07821	0,08109	0,1056	0,1113	0,2362
08	4070	0,07796	0,07852	0,1068	0,1139	0,2142
09	1590	0,06336	0,06359	0,1032	0,1045	0,1783
10	1200	0,07971	0,07952	0,1121	0,1103	0,1905
Média		**0,07464**	**0,07506**	**0,1066**	**0,1081**	**0,2124**

excelência do método de otimização LM sobre o GN, no problema de estimação da pose da câmera.

O custo computacional do método DFP_RI e BFGS_RI foi mais elevado do que os métodos GN_RI e LM_RI, contudo, muito mais estáveis. O método CY_RI apresentou um custo computacional bem mais elevado em relação aos ouros métodos, da ordem de $0,2$ segundos. Esse custo mais elevado já era esperado, uma vez que o método realiza muito mais operações matemáticas e utiliza mais informações do que os demais métodos propostos. Contudo, os resultados obtidos na estimação da pose da câmera apresentado na seção anterior, legítima a utilização do método.

Tabela 6 – Tempo de execução médio das principais etapas do sistema.

	Tempo Médio de Execução/frame [s]				
Etapa	**GN**	**LM**	**DFP**	**BFGS**	**CY**
Proc.Imagem	0,147178	0,147713	0,147165	0,148182	0,165313
Features	0,000051	0,000051	0,000051	0,000050	0,000050
Geometrica	0,129469	0,135180	0,266314	0,264885	0,578095
Escala	0,029729	0,031572	0,031554	0,031422	0,032066
Total	**0,306427**	**0,314516**	**0,445084**	**0,444539**	**0,775524**

A Tabela 6 descreve o custo computacional médio obtido em cada etapa do sistema proposto neste trabalho, e tem por finalidade servir de referência indicativa da capacidade de operação em tempo real do sistema proposto. Os valores resultantes foram obtidos em uma Raspberry Pi 4, utilizando o sistema operacional Raspian Lite, conforme descrito na

Seção 6.2. A validação foi feita utilizando os onze primeiros cenários do banco de dados KITTI.

Com base nos resultados analisados é possível verificar que "Proc.Imagem" e "Geometrica" são as duas etapas de maior custo computacional do sistema proposto. O custo computacional das etapas "Features"e "Escala"se mantiveram relativamente constantes em todos os métodos. Esse resultado já era esperado, uma vez que essas etapas são comuns para todos os métodos RI implementados.

Tabela 7 – Número de Iterações vs. Erro de Reprojeção.

Número da Imagem	**Iterações**	**Erro de Reprojeção**			
		GN_RI	**LM_RI**	**DFP_RI**	**BFGS_RI**
653	1	0,97755	0,91755	0,09693	0,11024
653	2	0,74273	0,71678	0,08871	0,10281
653	3	0,58360	0,52803	0,08340	0,09464
653	4	0,38086	0,32399	0,07353	0,09270
653	5	0,22912	0,18807	0,069212	0,08869
653	6	0,10326	0,08357	0,06369	0,07973
653	7	0,07030	0,06478	0,06170	0,07607
653	8	0,06566	0,06436	0,05603	0,07243
653	9	0,06051	0,05816	0,05444	0,06872
653	10	0,05610	0,05372	0,04954	0,06410
653	11	0,05267	0,04900	0,04629	0,05911
653	12	0,04793	0,04529	0,04238	0,05161
653	13	0,04301	0,040562	0,03889	0,04712
653	14	0,03936	0,03619	0,03395	0,03969
653	15	0,03736	0,03181	0,02802	0,03224
653	16	0,03323	0,02860	0,02550	0,02691
653	17	0,02872	0,02506	0,02238	0,02323

Por fim, a Tabela 7 apresenta o número de iterações e o erro de reprojeção obtidos utilizando-se cada um dos métodos de otimização numérica. A imagem 653 do cenário 3 do banco de dados KITTI foi utilizada para gerar os dados apresentados. Todos os algoritmos foram limitados a um máximo de 17 iterações ou interrompidos quando o erro de reprojeção caísse abaixo de $0,01$. Por inspeção, é possível verificar identificar que o algoritmo LM converge mais rápido do que o algoritmo de GN, ao passo que, os algoritmo DFP e BFGS convergem ainda mais rápido, embora entre esses dois últimos a taxa de decaimento da curva de erro tenha sido bastante similar.

6.7 Discussão e Comparação

Os resultados obtidos na seção anterior são comentados nessa seção, assim como uma análise comparativa com outros trabalhos na literatura. Os resultados apresentados estão sumarizados na Tabela 8. Ela é composta pela coluna "Sistema", que informa o nome do método ou trabalho, pela colunas $T[\%]$ e $R[deg/m]$, que representam o erro de translação e rotação em relação à distância média percorrida, pela coluna "T.Proc" que ilustra o tempo de processamento médio e pela coluna "Ambiente de Teste" que resume as configurações de hardware utilizadas na validação do métodos. De forma a possibilitar a comparação dos resultados deste trabalho com os disponíveis na literatura, também foram incluídos os resultados obtidos nos trabalhos RI_MVO (PEREIRA *et al.*, 2018), DVSO (YANG *et al.*, 2018) e D3VO (YANG *et al.*, 2020). Os trabalhos DVSO e D3VO representam os dois melhores trabalhos de OVM disponíveis no *rank* do KITTI até a conclusão deste texto. Os resultados dos métodos DVSO e D3VO foram retirados diretamente do *rank* do KITTI e o resultados do RI_MVO foram retirados de (PEREIRA, 2018).

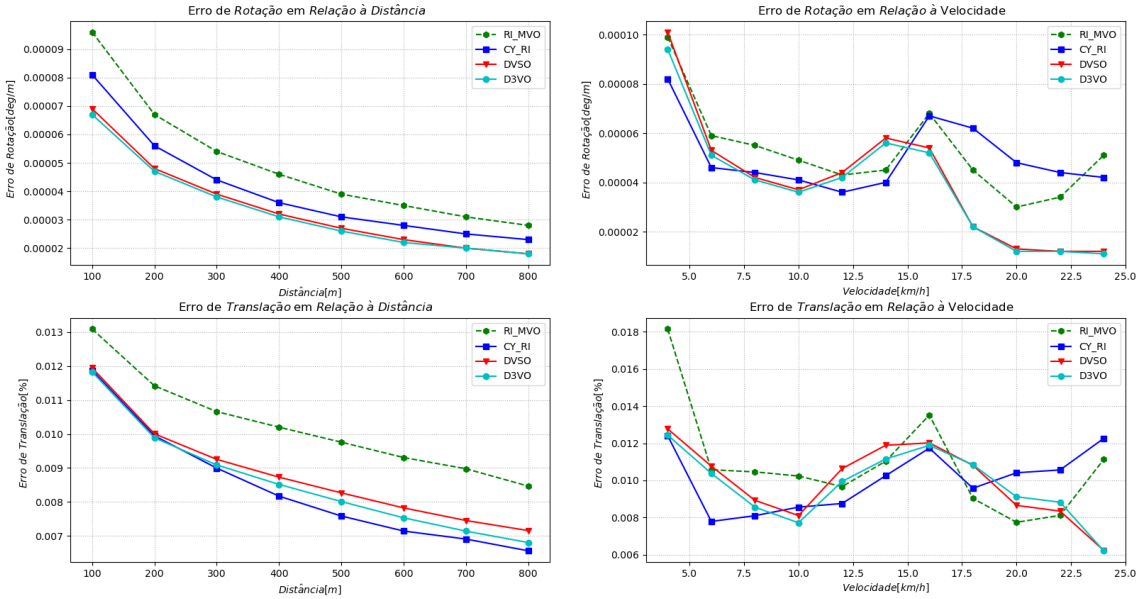
Tabela 8 – Tabela de comparação dos resultados

Sistema	**T[%]**	**R[deg/m]**	**T.Proc**	**Ambiente de Teste**
RI_MVO	1,02	0,0028	0,07	1 core @ >2,5 GHz (Python + C/C++)
GN_RI	0,95	0,0025	0,09	1 cores @ >2,3 GHz (Python + C/C++)
LM_RI	0,89	0,0025	0,09	1 cores @ >2,3 GHz (Python + C/C++)
DFP_RI	0,88	0,0025	0,09	1 cores @ >2,3 GHz (Python + C/C++)
BFGS_RI	0,88	0,0025	0,09	1 cores @ >2,3 GHz (Python + C/C++)
CY_RI	0,86	0,0024	0,20	1 core @ >2,3 GHz (Python + C/C++)
DVSO	0,90	0,0032	0,10	GPU @ > 2,5 GHz (C/C++)
D3VO	0,88	0,0021	0,10	1 core @ > 2,5 GHz (C/C++)

Através da comparação dos resultados dos métodos GN_RI e RI_MVO, é possível verificar o efeito da inclusão da equalização de histograma adaptativo na primeira etapa do sistema (Processamento de imagem) e do teste de convergência com região de busca expandida no erro de translação e rotação em relação à distância média percorrida, sendo sua desvantagem principal o custo computacional que foi um pouco maior. Os resultados dos métodos GN_RI e LM_RI, quando comparados, evidenciam o efeito positivo que o algoritmo LM tem na diminuição do erro da estimativa da pose da câmera em relação ao GN. O esforço computacional medido indica um leve aumento no método GN_RI e LM_RI, o que é esperado, uma vez que, foram incluídas a unidade de pré-processamento e teste de convergência com região de busca expandida.

Os métodos DFP_RI e BFGS_RI, apresentaram resultados bastante similares entre si, porém, a utilização de ambos os métodos apresentou resultados superiores ao métodos

Figura 38 – Ilustração dos resultados dos trabalhos RI_MVO, DVSO e D3VO e pelo sistema CY_RI implementado neste trabalho.



Fonte: do autor.

GN_RI e LM_RI, comprovando as vantagens que os algoritmos de otimização numérica DFP e BFGS tem sobre os demais utilizados neste trabalho.

O método CY_RI obteve os melhores resultados dentre os sistemas propostos neste trabalho. Os resultados indicam uma diminuição de aproximadamente $15,7\%$ e $14,3\%$ dos erros de translação e rotação em relação à distância média percorrida quando comparado com o trabalho RI_MVO. Entretanto, obteve um custo computacional significativamente maior em relação aos outros métodos apresentados. Em uma análise geral da Tabela 8, é possível notar que a precisão alcançada, em termos de erro de translação em relação à distância média percorrida, é superior a maioria dos trabalhos apresentados.

Uma comparação mais abrangente entre os resultados obtidos neste trabalho e os resultados obtidos por outros trabalhos pode ser feita através da Figura 38. Na figura, são ilustrados os resultados obtidos pelos sistemas RI_MVO, CY_RI, D3VO e DVSO. À primeira vista fica evidente que o comportamento da oscilação do erro em relação à velocidade do sistema CY_RI é semelhante ao do sistema RI_MVO, o que é esperado, uma vez que ambos os algoritmos minimizam a diferença de projeção, destacando um problema em comum, com a oscilação do erro de translação e rotação em relação à velocidade média quando o veículo trafega entre $15km/h$ e $17,5km/h$. O sistema CY_RI obteve um erro de translação em relação à distância média percorrida superior ao dos outros sistemas. Contudo, apresentou um erro de rotação em relação à distância média percorrida inferior ao dos sistemas D3VO e DVSO. Também é possível observar que o CY_RI ob-

teve resultados melhores nos erros de translação e rotação em relação à velocidade média até $15km/h$.

Normalmente, a análise de tempo de execução entre diferentes trabalhos não é uma tarefa trivial, uma vez que diferentes soluções são desenvolvidas usando diferentes ambientes e hardwares. A análise da complexidade computacional mostra que todos os métodos propostos neste trabalho apresentam um aumento no tempo de execução em relação ao RI proposto por Pereira et al. (PEREIRA *et al.*, 2018), porém, as melhorias alcançadas no erro de translação e erro de rotação justificam esse aumento. Com exceção do método CY_RI, todos os métodos propostos neste trabalho obtiveram valores de tempo de execução menores que os trabalhos D3VO e DVSO.

Os resultados alcançados atestam um importante avanço nos sistemas de Odometria Visual Monocular baseados na abordagem RI e com probabilidade de apresentar bom desempenho em outras aplicações de visão computacional diferentes do foco de aplicação deste trabalho. Dentre as principais contribuições deste trabalho em relação ao sistema RI_MVO, pode-se destacar a inclusão da etapa de pre-processamento das imagens de entrada, o uso de diferentes métodos de otimização matemática no método RI, o aprimoramento do teste de convergência e criação do método RI com refinamento cíclico. Sendo valido realçar que todas as contribuições citadas resultaram de forma individual em um aprimoramento do resultado final do sistema OVM esparso proposto neste trabalho. Outra novidade foi o fato dos sistemas propostos terem sido embarcados e validados em uma placa Raspberry Pi, com a finalidade de se verificar sua capacidade de operar em tempo real.

7 CONCLUSÃO

Este trabalho apresentou a implementação de um conjunto de novos sistemas de OVM do tipo esparso baseado no conceito *Resection-Intersection* com foco na aplicação em veículos autônomos. Ao longo do trabalho, foram feitas simulações com diferentes métodos de detecção de *features* para identificar qual método apresentava os melhores resultados em termos de precisão de estimação da pose da câmera e de custo computacional. Com base nos resultados apresentados o método Shi-Tomasi foi o escolhido. Três tipos de regiões para realizar a separação das *features* que estão somente no solo da cena utilizado na etapa de recuperação da escala foram avaliadas. A separação utilizando a área em forma de arco foi a que apresentou melhores resultados. Também foi constatado que a equalização adaptativa do histograma das imagens de entrada do sistema melhorou de forma significativa a detecção e rastreamento das *features* e consequentemente a precisão na estimação da pose da câmera, justificando a inserção da etapa de pré-processamento no sistema.

O Capítulo 5 apresentou um conjunto de novos sistemas de OVM baseados em RI. O primeiro sistema implementado foi um RI construído com base no algoritmo de Gauss-Newton (GN_RI), que embora não seja uma novidade na literatura, foi aprimorado através da inclusão do teste de convergência com região de busca expandida proposto. Através da análise dos resultados se constatou que o algoritmo de teste de convergência com região de busca expandida foi capaz de magnificar a precisão na estimativa da pose da câmera. Os resultados obtidos foram superiores ao trabalho RI_MVO, confirmando a eficácia das inovações propostas.

O sistema RI construído com base no algoritmo de otimização numérica de Levenberg-Marquardt (LM_RI) apresentou resultados ainda melhores de erro de translação em relação à distância média percorrida, indicando que o ajuste de Levenberg-Marquardt inserido é capaz de melhor a resposta do processo de otimização numérica quando ocorrem problemas de mínimos locais. Em seguida, foram propostos dois novos sistemas RI construídos com base nos algoritmos de otimização numérica de Davidon–Fletcher–Powell (DFP) e Broyden–Fletcher–Goldfarb–Shanno (BFGS), que até a conclusão deste texto, não tinham

sido utilizados no problema de estimação da pose da câmera em sistema baseados em RI, e dessa forma, representam dois métodos de RI inéditos em relação a literatura atual.

A implementação de sistemas RI com base em diferente métodos de otimização realizada neste trabalho, também possibilitou a comparação da precisão e desempenho do uso de cada um desse métodos de otimização no problema de estimação da pose da câmera. Outra contribuição foi a proposta de um novo sistema RI com base no refinamento cíclico (CY_RI). O erro de translação em relação à distância média percorrida obtido foi superior ao dos outros sistemas propostos neste trabalho e de outros trabalhos baseados no conceito RI disponíveis na literatura atual. Embora o seu custo computacional tenha sido superior ao dos outros métodos propostos neste trabalho, ele se mantém no mesmo patamar dos melhores trabalhos do *rank* do KITTI.

Os objetivos iniciais listados na Seção 1.3, descreviam a elaboração de um novo sistema de OVM, baseado no conceito RI, capaz de atingir alta precisão na estimativa da pose da câmera aliado a um baixo custo computacional, para ser embarcado em um hardware de capacidade moderada. Com base no trabalho apresentado, é possível afirmar que todos os objetivos listados foram alcançados. Além disso, os resultados obtidos por todos os métodos propostos foram superiores ao de outros trabalhos baseados no conceito RI publicados anteriormente e rivalizam com os melhores trabalhos atualmente classificados no banco de dados KITTI.

Trabalhos futuros incluirão a paralelização do sistema proposto e a sua implementação em linguagem de programação C/C++, com o objetivo de embarcá-lo em um sistema Multiprocessador baseado em RISC-V (ILHA, 2018).

REFERÊNCIAS

ALAMEDY, A. **Pixel Show blog**. Disponível em: <https://zupi.pixelshow.co/ali-alamedy-construiu-um-estudio-de-fotografia-em-miniatura/>. Acesso em: maio 2021.

ALVES, T. W. **Sistema de detecção em tempo real de faixas de sinalização de trânsito para veículos inteligentes utilizando processamento de imagem**. 2017. Tese (Doutorado em Engenharia Elétrica) — Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, BR, 2017.

AYACHE, N.; FAUGERAS, O. D. Building, Registrating, and Fusing Noisy Visual Maps. **The International Journal of Robotics Research**, USA, v. 7, n. 6, p. 45–65, dec 1988.

BERGEN, M. **Meet the Companies Building Self-Driving Cars for Google and Tesla (And Maybe Apple)**. Disponível em: <https://www.vox.com/2015/10/27/11620026/meet-the-companies-building-self-driving-cars-for-google-and-tesla>. Acesso em: Outubro 2021.

BJORCK, A. *et al.* **Numerical Methods for Least Squares Problems**. 1st. ed. Filadélfia, EUA: Society for Industrial and Applied Mathematics, 1996.

BONNANS, J.-F. *et al.* **Numerical Optimization Theoretical and Practical Aspects**. 2nd. ed. Berlin: Springer, 2006.

BRADSKI, G. The OpenCV Library. **Dr. Dobb's Journal of Software Tools**, USA, v. 1, p. 1, 1 2000.

BURDEN, R.; FAIRES, J. **Numerical Analysis**. 1st. ed. Boston: Cengage Learning, 2004.

CAO, Y.; NIU, X.; DOU, Y. Region-based convolutional neural networks for object detection in very high resolution remote sensing images. *In*: INTERNATIONAL CONFERENCE ON NATURAL COMPUTATION, FUZZY SYSTEMS AND KNOWLEDGE DISCOVERY (ICNC-FSKD), 12., 2016. **Proceedings [...]** IEEE, 2016. p. 548–554.

CASTLEMAN, K. R. **Digital Image Processing**. 1st. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 1996.

CHATILA, R.; LAUMOND, J. Position referencing and consistent world modeling for mobile robots. *In*: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 1985., 1985. **Proceedings [...]** IEEE, 1985. v. 2, p. 138–145.

DAVISON, A. J. *et al.* MonoSLAM: real-time single camera slam. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Piscataway, NJ, v. 29, n. 6, p. 1052–1067, June 2007.

DESAI, A.; LEE, D. Visual Odometry Drift Reduction Using SYBA Descriptor and Feature Transformation. **IEEE Transactions on Intelligent Transportation Systems**, Piscataway, NJ, v. 17, n. 7, p. 1839–1851, July 2016.

DURRANT-WHYTE, H. F. Uncertain geometry in robotics. **IEEE Journal on Robotics and Automation**, Piscataway, NJ, v. 4, n. 1, p. 23–31, 1 1988.

ENGEL, J. **Computer Vision Research**. Disponível em: <https://jakobengel.github.io/>. Acesso em: maio 2021.

ENGEL, J.; KOLTUN, V.; CREMERS, D. **Direct Sparse Odometry**. Disponível em: <https://vision.in.tum.de/research/vslam/dso>. Acesso em: maio 2020.

ENGEL, J.; STURM, J.; CREMERS, D. Semi-dense Visual Odometry for a Monocular Camera. *In*: IEEE INTERNATIONAL CONFERENCE ON COMPUTER VISION, 2013., 2013. **Proceedings [...]** IEEE, 2013. p. 1449–1456.

ENGEL, J.; STüCKLER, J.; CREMERS, D. Large-scale direct SLAM with stereo cameras. *In*: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS), 2015., 2015. **Proceedings [...]** IEEE, 2015. p. 1935–1942.

FANANI, N. *et al.* Predictive monocular odometry (PMO): what is possible without ransac and multiframe bundle adjustment? **Image and Vision Computing**, Amsterdã, NL, v. 68, p. 1, 08 2017.

FENG, T.; GU, D. SGANVO: unsupervised deep visual odometry and depth estimation with stacked generative adversarial networks. **IEEE Robotics and Automation Letters**, [S.l.], v. 4, n. 4, p. 4431–4437, Oct 2019.

FERRERA, M. *et al.* OV^2SLAM: a fully online and versatile visual slam for real-time applications. **IEEE Robotics and Automation Letters**, New York, NY, USA, v. 6, n. 2, p. 1399–1406, 1 2021.

FISCHLER, M. A.; BOLLES, R. C. Random Sample Consensus: a paradigm for model fitting with applications to image analysis and automated cartography. **Commun. ACM**, New York, NY, USA, v. 24, n. 6, p. 381–395, June 1981.

FORSTER, C.; PIZZOLI, M.; SCARAMUZZA, D. SVO: fast semi-direct monocular visual odometry. *In*: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA), 2014., 2014. **Proceedings [...]** IEEE, 2014. p. 15–22.

FOUNDATION, T. R. P. **Raspberry Pi Hardware**. Disponível em: <https://www.raspberrypi.org/>. Acesso em: abril 2021.

GALILEU, R. **Pouso da Apollo 11 teria transportado germes da Lua para a Terra**. Disponível em: <https://revistagalileu.globo.com/Ciencia/Espaco/noticia/2019/07/pouso-da-apollo-11- pode-ter-trazido-germes-da-lua-para-terra.html/>. Acesso em: abril 2021.

GAO, X.-S. *et al.* Complete solution classification for the perspective-three-point problem. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Piscataway, NJ, v. 25, n. 8, p. 930–943, Aug 2003.

GEIGER, A.; LENZ, P.; URTASUN, R. Are we ready for autonomous driving? The KITTI Vision Benchmark Suite. **IEEE Computer Society Conference on Computer Vision and Pattern Recognition.**, New York, NY, USA, v. 1, p. 3354–3361, 05 2012.

GEIGER ANDREAS LENZ PHILIP, S. C. **Visual Odometry / SLAM Evaluation 2012**. Disponível em: <http://www.cvlibs.net/datasets/kitti/eval_odometry.php/>. Acesso em: Junho 2021.

GERNOT, H. **Luminance Models For The Grayscale Conversion**. 1st. ed. São Paulo, SP, Brasil: Mcgraw Hill do Brasil, 2001.

GITTEL, B. **Multi-view Geometry**. Disponível em: <https://www.slideserve.com/gittel/multi-view-geometry/>. Acesso em: Junho 2021.

GOMEZ-OJEDA, R.; GONZALEZ-JIMENEZ, J. Robust stereo visual odometry through a probabilistic combination of points and line segments. *In*: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA), 2016., 2016. **Proceedings [...]** [S.l.: s.n.], 2016. p. 2521–2526.

GONZALEZ, R. C.; WOODS, R. E. **Digital Image Processing**. 3rd. ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006.

GRAMFORT, A. **(Quasi-)Newton methods**. Disponível em: <http://www.lix.polytechnique.fr/bigdata/mathbigdata/wp-content/uploads/2014/09/notes_quasi_newton.pdf/>. Acesso em: abril 2021.

HARRIS, C. R. *et al.* Array programming with NumPy. **Nature**, UK, v. 585, n. 7825, p. 357–362, Sept. 2020.

HARRIS, C.; STEPHENS, M. A combined corner and edge detector. *In*: IN PROC. OF FOURTH ALVEY VISION CONFERENCE, 1988. **Proceedings [...]** University of Sheffield Printing Office, 1988. p. 147–151.

HARTLEY, R. I.; STURM, P. Triangulation. **Computer Vision and Image Understanding**, Amsterdã, NL, v. 68, n. 2, p. 146–157, 1 1997.

HARTLEY, R.; ZISSERMAN, A. **Multiple View Geometry in Computer Vision**. 2. ed. Cambridge: Cambridge University Press, 2004.

HARTLEY, R.; ZISSERMAN, A. **Multi-View Geometry in Computer Vision**. 2nd. ed. Cambridge: Cambridge University Press, 2004.

HOÀNG-ÂN, L. **Camera model**: intrinsic parameters. Disponível em: <https://lhoangan.github.io/camera-params/>. Acesso em: Outubro 2021.

HOELSCHER, I. G. **Detecção e classificação de sinalização vertical de trânsito em cenários complexos**. 2017. Tese (Doutorado em Engenharia Elétrica) — Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, BR, 2017.

HORAUD, R. *et al.* An analytic solution for the perspective 4-point problem. *In*: CVPR '89: IEEE COMPUTER SOCIETY CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 1989. **Proceedings [...]** IEEE, 1989. p. 500–507.

HUNTER, J. D. Matplotlib: a 2d graphics environment. **Computing in science & engineering**, New York, NY, USA, v. 9, n. 3, p. 90–95, 1 2007.

IDRIS; RAZAK, Z. Review of Feature Detection Techniques for Simultaneous Localization and Mapping and System on Chip Approach. **Information Technology Journal**, Piscataway, NJ, v. 8, p. 250–262, 03 2009.

ILHA, G. **MPVUE - plataforma multiprocessador em chip para visão computacional**. 2018. Tese (Doutorado em Engenharia Elétrica) — Universidade Federal do Rio Grande do Sul, Porto Alegre, 2018.

INAVI. **Lane Departure Warning System (LDWS) in THINKWARE Dash Cam**. Disponível em: <https://www.youtube.com/watch?v=CJqVCVzGgS0>. Acesso em: junho 2020.

KEJARIWAL, A.; ORSINI, F. On the Definition of Real-Time: applications and systems. *In*: IEEE TRUSTCOM/BIGDATASE/ISPA, 2016., 2016. **Proceedings [...]** [S.l.: s.n.], 2016. p. 2213–2220.

KITT, B.; GEIGER, A.; LATEGAHN, H. Visual odometry based on stereo image sequences with RANSAC-based outlier rejection scheme. *In*: IEEE INTELLIGENT VEHICLES SYMPOSIUM, 2010., 2010. **Proceedings [...]** IEEE, 2010. p. 486–492.

KLEIN, G.; MURRAY, D. Parallel tracking and mapping for small AR workspaces. **ISMAR. IEEE**, Piscataway, NJ, v. 1, n. 1, p. 225–234, 01 2007.

LDSV. **Veículos Autônomos**. Disponível em: <http://www.usp.br/ldsv/?page_id=1856>. Acesso em: Outubro 2021.

LEE, G. H.; FAUNDORFER, F.; POLLEFEYS, M. Motion Estimation for Self-Driving Cars with a Generalized Camera. *In*: IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2013., 2013. **Proceedings [...]** IEEE, 2013. p. 2746–2753.

LEONARD, J.; DURRANT-WHYTE, H. **Directed Sonar Sensing for Mobile Robot Navigation**. 1st. ed. New York, NY: Springer, 2004.

LEPETIT, V.; MORENO-NOGUER, F.; FUA, P. EPnP: an accurate o(n) solution to the pnp problem. **International Journal of Computer Vision**, New York, NY, USA, v. 81, p. 155–166, 02 2009.

LEVIN, D. **Google and Tesla Say Self-Driving Tech Will Accelerate Exponentially Within 24 Months**. Disponível em: <https://www.forbes.com/sites/doronlevin/2019/04/24/google-and-tesla-say-self-driving-tech-will-accelerate-exponentially-within-twenty-four-months/?sh=7ca4e1325e51>. Acesso em: Outubro 2021.

LOURAKIS, M. I. A.; ARGYROS, A. A. SBA: a software package for generic sparse bundle adjustment. **ACM Transactions on Mathematical Software**, New York, NY, USA, v. 1, n. 1, p. 1–30, 1 2009.

LOWE, D. G. Object recognition from local scale-invariant features. *In*: SEVENTH IEEE INTERNATIONAL CONFERENCE ON COMPUTER VISION, 1999. **Proceedings [...]** IEEE, 1999. v. 2, p. 1150–1157.

LUCAS, B. D.; KANADE, T. An Iterative Image Registration Technique with an Application to Stereo Vision. *In*: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 7., 1981. **Proceedings [...]** Morgan Kaufmann Publishers Inc., 1981. v. 2, p. 674–679.

MAMAT, M. *et al.* Derivative free Davidon-Fletcher-Powell (DFP) for solving symmetric systems of nonlinear equations. **IOP Conference Series: Materials Science and Engineering**, USA, v. 332, 03 2018.

MENG, X.; HONG, F.; CHEN, Y. Monocular simultaneous localization and mapping with a modified covariance Extended Kalman Filter. *In*: IEEE INTERNATIONAL CONFERENCE ON INTELLIGENT COMPUTING AND INTELLIGENT SYSTEMS, 2009., 2009. **Proceedings [...]** Springer Berlin Heidelberg, 2009. v. 2, n. 1, p. 900–903.

MIRABDOLLAH, M. H.; MERTSCHING, B. On the Second Order Statistics of Essential Matrix Elements. *In*: PATTERN RECOGNITION, 2014. **Proceedings [...]** Springer International Publishing, 2014. p. 547–557.

MIRABDOLLAH, M. H.; MERTSCHING, B. Fast Techniques for Monocular Visual Odometry. *In*: PATTERN RECOGNITION, 2015. **Proceedings [...]** Springer International Publishing, 2015. p. 297–307.

MORAVEC, H. P. **Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover**. 1980. Tese (Doutorado em Engenharia Elétrica) — Stanford University, Stanford, CA, USA, 1980. AAI8024717.

NISTER, D. An efficient solution to the five-point relative pose problem. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Piscataway, NJ, v. 26, n. 6, p. 756–770, June 2004.

PEDRINI, H.; SCHWARTZ, W. R. **Análise de Imagens Digitais**: princípios, algoritmos e aplicações. 1st. ed. São Paulo: Cengage Learning, 2007.

PEREIRA, F. I. **High precision monocular visual odometry**. 2018. Tese (Doutorado em Engenharia Elétrica) — Universidade Federal do Rio Grande do Sul, Porto Alegre, 2018.

PEREIRA, F. I. *et al.* Monocular Visual Odometry with Cyclic Estimation. *In*: SIBGRAPI CONFERENCE ON GRAPHICS, PATTERNS AND IMAGES, 2017., 2017. **Proceedings [...]** SIBGRAPI, 2017. p. 1–6.

PEREIRA, F. I. *et al.* A Novel Resection–Intersection Algorithm With Fast Triangulation Applied to Monocular Visual Odometry. **IEEE Transactions on Intelligent Transportation Systems**, [S.l.], v. 19, n. 11, p. 3584–3593, 1 2018.

PERSSON, M. *et al.* Robust stereo visual odometry from monocular techniques. *In*: IEEE INTELLIGENT VEHICLES SYMPOSIUM (IV), 2015., 2015. **Proceedings [...]** IEEE, 2015. p. 686–691.

PIPER, K. **It's 2020. Where are our self-driving cars?** Disponível em: <https://www.vox.com/future-perfect/2020/2/14/21063487/self-driving-cars-autonomous-vehicles-waymo-cruise-uber>. Acesso em: Outubro 2021.

PIZER, S. M. *et al.* Adaptive histogram equalization and its variations. **Computer Vision, Graphics, and Image Processing**, Amsterdã, NL, v. 39, n. 3, p. 355–368, 1 1987.

QUAN, L.; LAN, Z. Linear N-point camera pose determination. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Piscataway, NJ, v. 21, n. 8, p. 774–780, 1 1999.

RADKE, R. J. **Computer Vision for Visual Effects**. 3rd. ed. Cambridge, UK: Cambridge University Press, 2012.

RENCKEN, W. D. Concurrent localisation and map building for mobile robots using ultrasonic sensors. *In*: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS '93), 1993., 1993. **Proceedings [...]** IEEE, 1993. v. 3, p. 2192–2197.

ROSTEN, E.; DRUMMOND, T. Machine Learning for High-Speed Corner Detection. *In*: COMPUTER VISION – ECCV 2006, 2006. **Proceedings [...]** Springer Berlin Heidelberg, 2006. p. 430–443.

RUBLEE, E. *et al.* ORB: an efficient alternative to sift or surf. *In*: INTERNATIONAL CONFERENCE ON COMPUTER VISION, 2011., 2011. **Proceedings [...]** [S.l.: s.n.], 2011. v. 1, n. 1, p. 2564–2571.

SADEKAR, K. **image-formation-with-pinhole-camera-model**. Disponível em: <https://learnopencv.com/camera-calibration-using-opencv/image-formation-with-pinhole-camera-model//>. Acesso em: Maio 2021.

SALIH, B.; ABBO, K.; ABDULLAH, Z. Partial Davidon, Fletcher and Powell (DFP) of quasi newton method for unconstrained optimization. **ikrit Journal of Pure Science**, Iraque, v. 21, p. 2415–1726, 1 2016.

SCARAMUZZA, D. *et al.* Absolute scale in structure from motion from a single vehicle mounted camera by exploiting nonholonomic constraints. *In*: IEEE 12TH INTERNATIONAL CONFERENCE ON COMPUTER VISION, 2009., 2009. **Proceedings [...]** IEEE, 2009. p. 1413–1419.

SHI, J.; TOMASI. Good features to track. *In*: PROCEEDINGS OF IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 1994., 1994. **Proceedings [...]** IEEE, 1994. p. 593–600.

SILVA, L. N. d. C. **Análise e síntese de estrategias de aprendizado para redes neurais artificiais**. 1998. Tese (Doutorado em Engenharia Elétrica) — Unicamp, São Paulo, SP, BR, 1998.

SMITH, R. C.; CHEESEMAN, P. On the Representation and Estimation of Spatial Uncertainty. **The International Journal of Robotics Research**, Amsterdã, NL, v. 5, n. 4, p. 56–68, 1 1986.

SMITH, R.; SELF, M.; CHEESEMAN, P. **Estimating Uncertain Spatial Relationships in Robotics**. 1st. ed. New York, NY: Springer New York, 1990.

SONG, S.; CHANDRAKER, M.; GUEST, C. C. Parallel, real-time monocular visual odometry. *In*: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 2013., 2013. **Proceedings [...]** IEEE, 2013. p. 4698–4705.

SONG, S.; CHANDRAKER, M.; GUEST, C. C. High Accuracy Monocular SFM and Scale Correction for Autonomous Driving. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Piscataway, NJ, v. 38, n. 4, p. 730–743, April 2016.

TARDIF, J.; PAVLIDIS, Y.; DANIILIDIS, K. Monocular visual odometry in urban environments using an omnidirectional camera. *In*: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS, 2008., 2008. **Proceedings [...]** IEEE, 2008. p. 2531–2538.

THOME, A. G. **Aquisição e Representação da Imagem Digital**. Disponível em: <http://equipe.nce.ufrj.br/thome/p_grad/nn_img/transp/c2_aquis_v2.pdf>. Acesso em: julho 2020.

TRABER, A.; GAUTSCHI, M.; SCHIAVONE, P. D. **RI5CY Core**: Datasheet. Disponível em: <https://www.pulp-platform.org/docs/ri5cy_user_manual.pdf>. Acesso em: maio 2020.

TRIGGS, B. Factorization methods for projective structure and motion. *In*: CVPR IEEE COMPUTER SOCIETY CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 1996. **Proceedings [...]** IEEE, 1996. p. 845–851.

TRIGGS, B. *et al.* Bundle Adjustment – A Modern Synthesis. *In*: VISION ALGORITHMS: THEORY AND PRACTICE, LNCS, 2000. **Proceedings [...]** Springer Verlag, 2000. p. 298–375.

TSUGAWA, S. Issues and Recent Trends in Vehicle Safety Communication Systems. **IATSS Research**, Amsterdã, NL, v. 29, n. 1, p. 7–15, Fev. 2005.

VAN ROSSUM, G.; DRAKE, F. L. **Python 3 Reference Manual**. 1st. ed. Scotts Valley, CA: CreateSpace, 2009.

WÖHLER, C. **Triangulation-Based Approaches to Three-Dimensional Scene Reconstruction**. 1st. ed. London: Springer London, 2013.

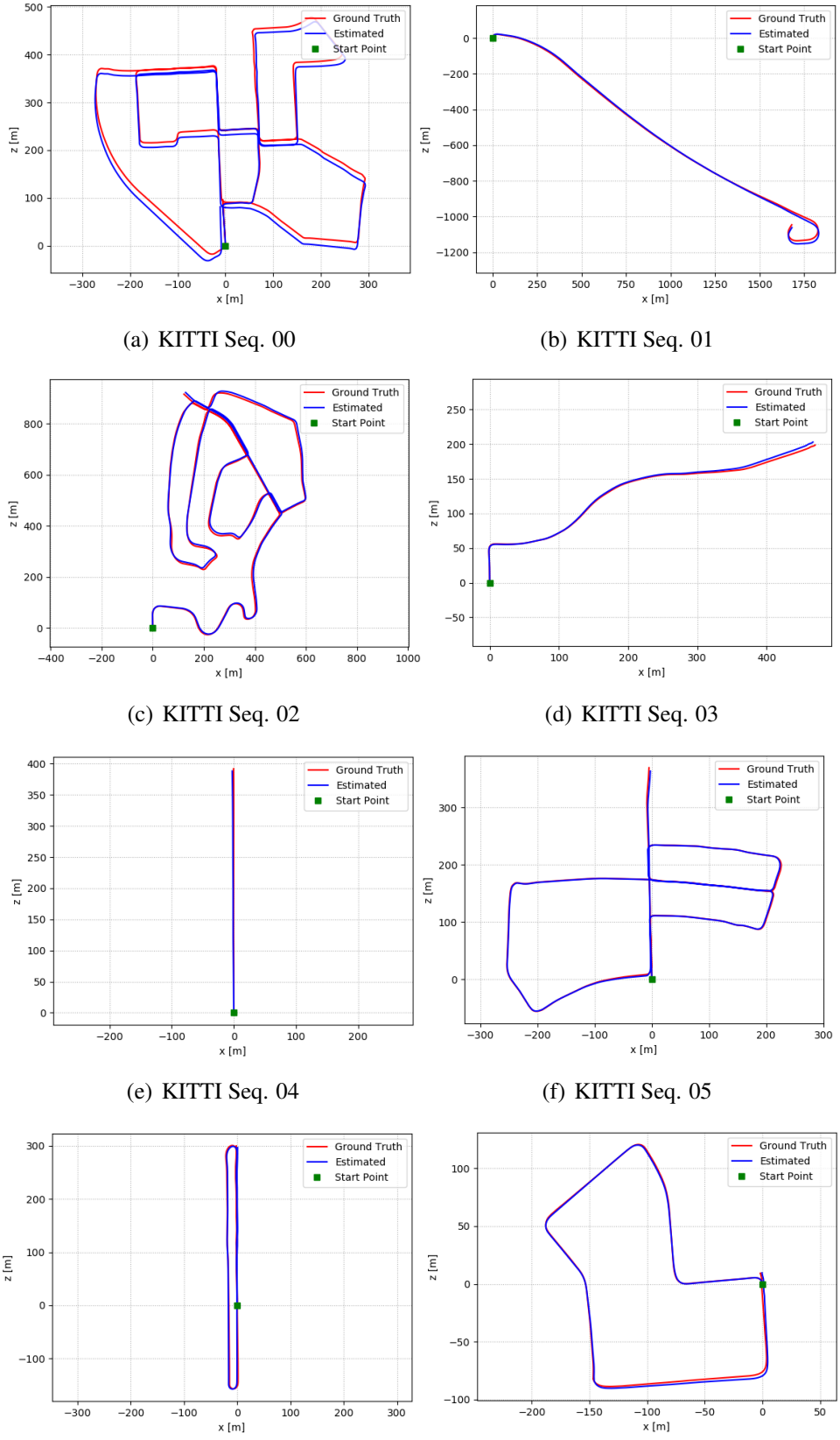
YANG, N. *et al.* Deep Virtual Stereo Odometry: leveraging deep depth prediction for monocular direct sparse odometry. *In*: COMPUTER VISION – ECCV 2018, 2018. **Proceedings [...]** Springer International Publishing, 2018. p. 835–852.

YANG, N. *et al.* D3VO: deep depth, deep pose and deep uncertainty for monocular visual odometry. **2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**, New York, NY, USA, v. 2003.01060, p. 1278–1289, 06 2020.

YVES BOUGUET, J. Pyramidal implementation of the Lucas Kanade feature tracker. **Intel Corporation, Microprocessor Research Labs**, Piscataway, NJ, v. 1, n. 1, p. 01, 03 2000.

8 APÊNDICE

Figura 39 – Trajetórias de 00 até 07 estimadas pelo método GN_RI.



(a) KITTI Seq. 00

(b) KITTI Seq. 01

(c) KITTI Seq. 02

(d) KITTI Seq. 03

(e) KITTI Seq. 04

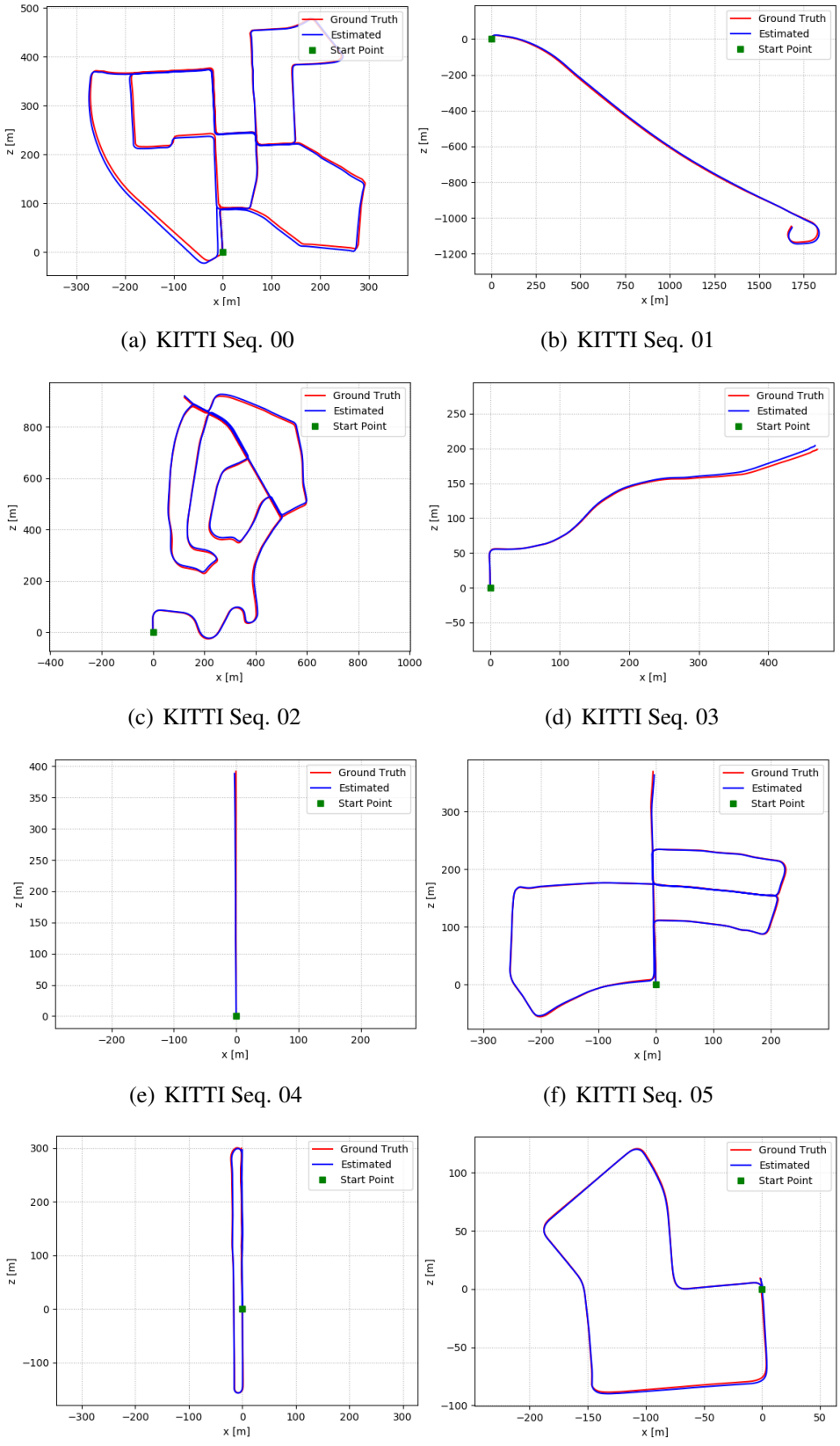
(f) KITTI Seq. 05

(g) KITTI Seq. 06

(h) KITTI Seq. 07

Fonte: do autor.

Figura 40 – Trajetórias de 00 até 07 estimadas pelo método LM_RI.



(a) KITTI Seq. 00

(b) KITTI Seq. 01

(c) KITTI Seq. 02

(d) KITTI Seq. 03

(e) KITTI Seq. 04

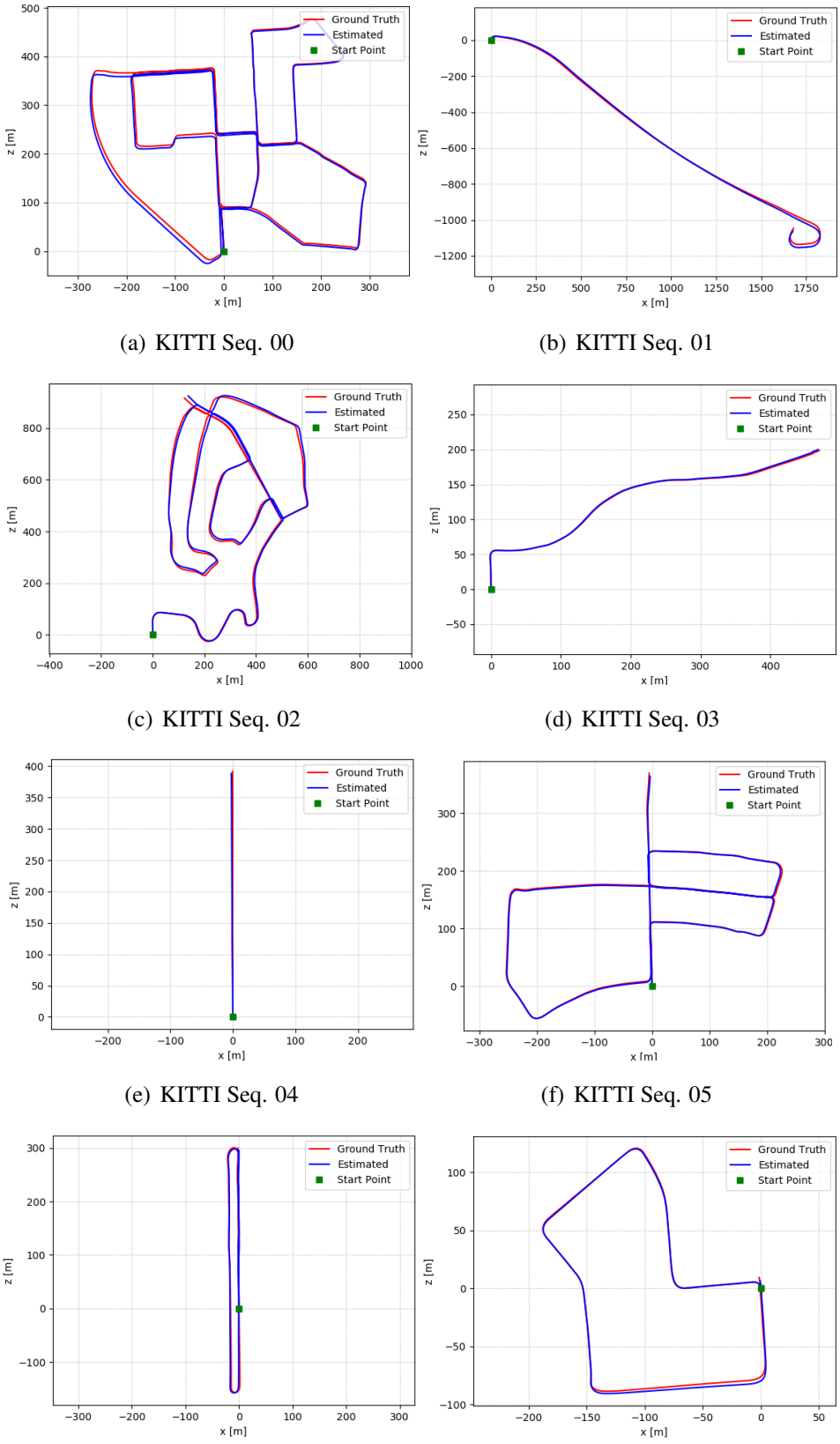
(f) KITTI Seq. 05

(g) KITTI Seq. 06

(h) KITTI Seq. 07

Fonte: do autor.

Figura 41 – Trajetórias de 00 até 07 estimadas pelo método DFP_RI.



(a) KITTI Seq. 00

(b) KITTI Seq. 01

(c) KITTI Seq. 02

(d) KITTI Seq. 03

(e) KITTI Seq. 04

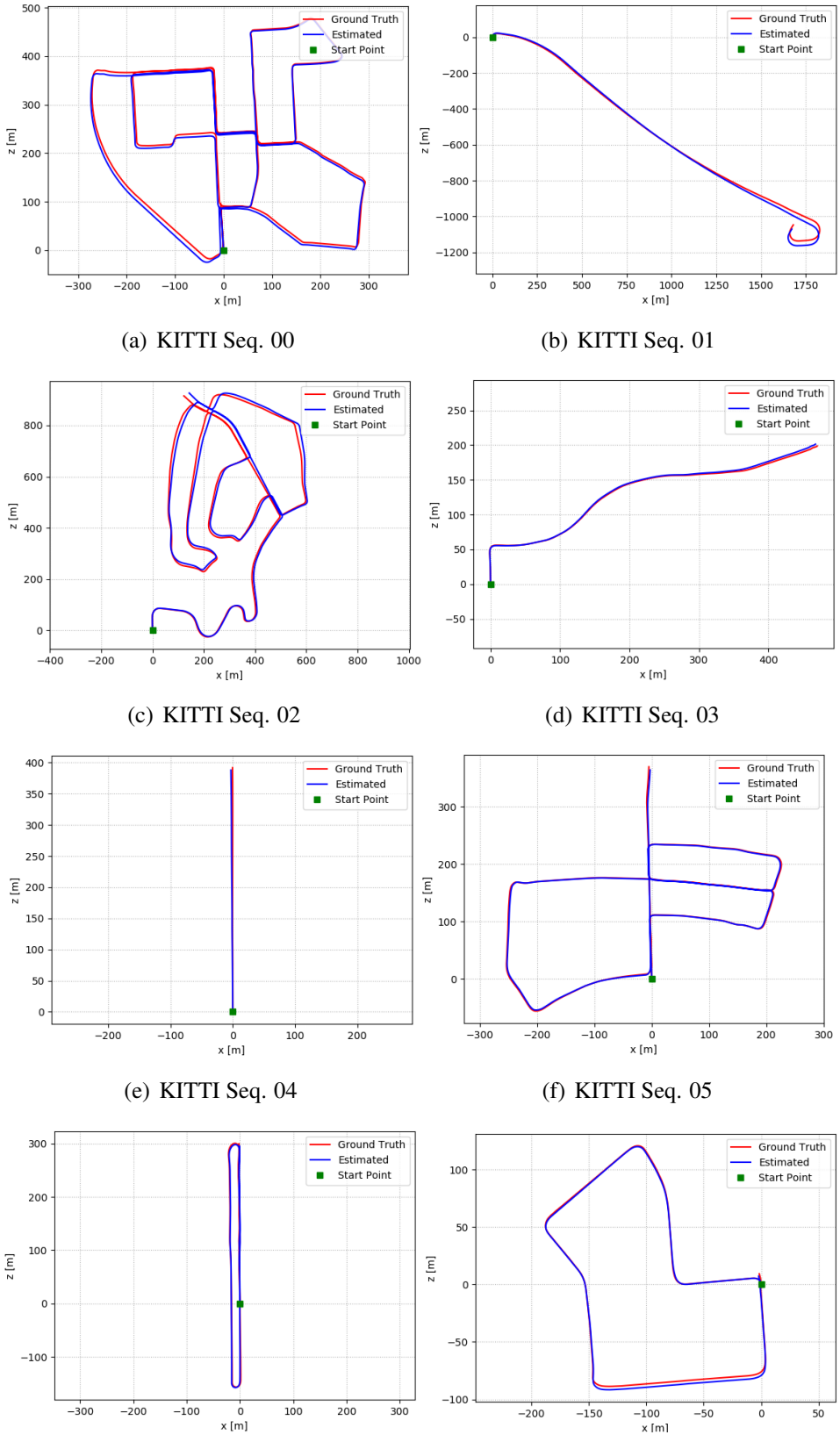
(f) KITTI Seq. 05

(g) KITTI Seq. 06

(h) KITTI Seq. 07

Fonte: do autor.

Figura 42 – Trajetórias de 00 até 07 estimadas pelo método BFGS_RI.



(a) KITTI Seq. 00

(b) KITTI Seq. 01

(c) KITTI Seq. 02

(d) KITTI Seq. 03

(e) KITTI Seq. 04

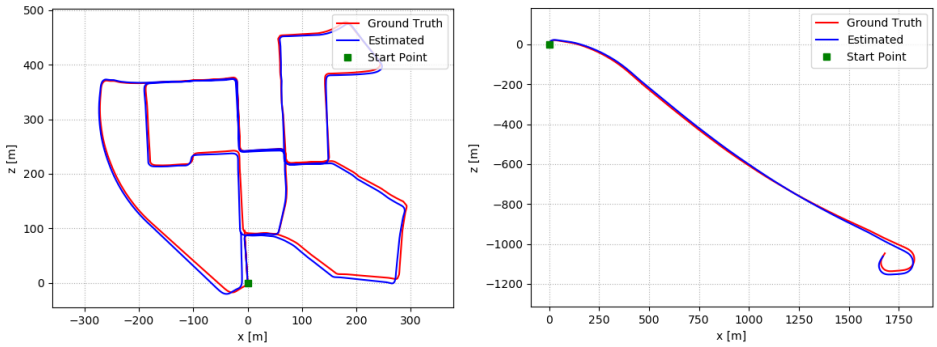
(f) KITTI Seq. 05

(g) KITTI Seq. 06

(h) KITTI Seq. 07

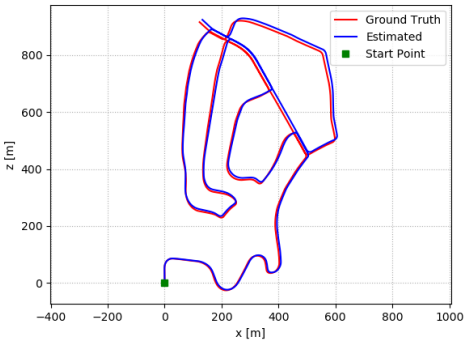
Fonte: do autor.

Figura 43 – Trajetórias de 00 até 07 estimadas pelo método CY_RI.

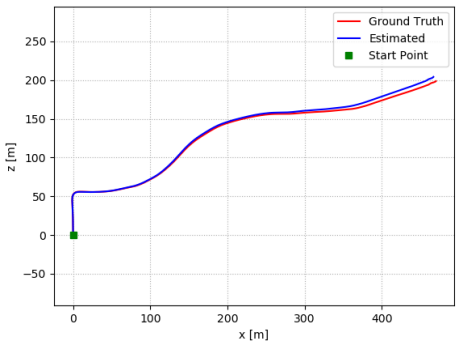


(a) KITTI Seq. 00

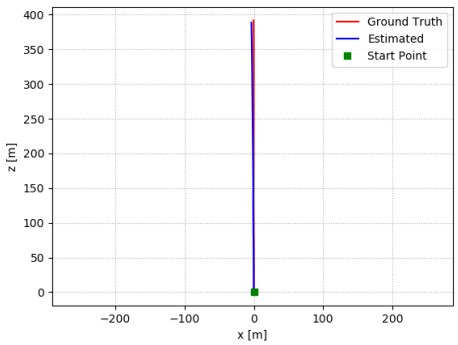
(b) KITTI Seq. 01



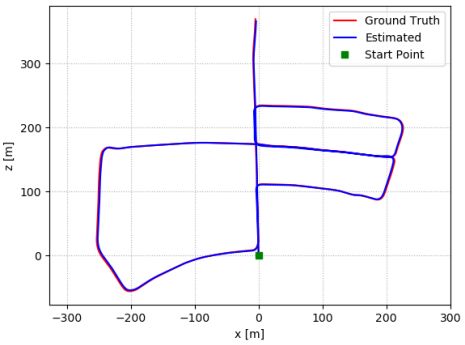
(c) KITTI Seq. 02



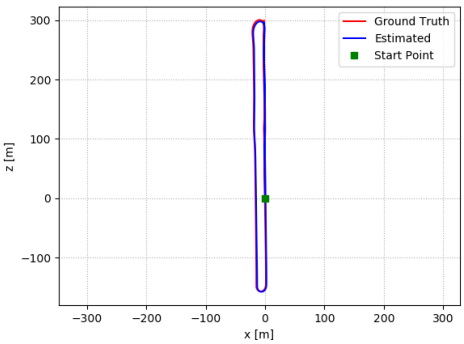
(d) KITTI Seq. 03



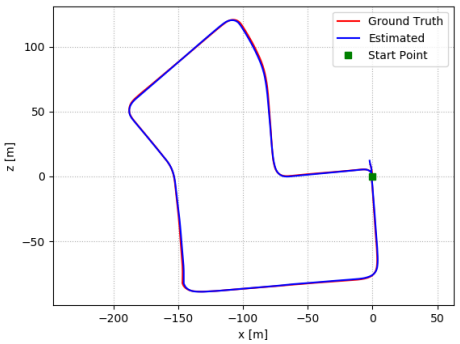
(e) KITTI Seq. 04



(f) KITTI Seq. 05



(g) KITTI Seq. 06



(h) KITTI Seq. 07

Fonte: do autor.