



Universidade Federal da Paraíba
Centro de Tecnologia
Programa de Pós-Graduação em Engenharia Mecânica
Doutorado



UMA ABORDAGEM DISTRIBUÍDA PARA A IMPLEMENTAÇÃO DE SISTEMAS DE SLAM VISUAL BASEADOS EM RECONSTRUÇÃO GEOMÉTRICA

por

OTACÍLIO DE ARAÚJO RAMOS NETO

*Tese de Doutorado apresentada à Universidade Federal da
Paraíba para obtenção do grau de Doutor*

OTACÍLIO DE ARAÚJO RAMOS NETO

**UMA ABORDAGEM DISTRIBUÍDA PARA A
IMPLEMENTAÇÃO DE SISTEMAS DE SLAM VISUAL
BASEADOS EM RECONSTRUÇÃO GEOMÉTRICA**

Tese de doutorado submetida ao Programa de Pós-Graduação em Engenharia Mecânica do Centro de Tecnologia da Universidade Federal da Paraíba em cumprimento às exigências para obtenção do grau de Doutor em Ciências no Domínio da Engenharia Mecânica.

Orientador: Prof. Abel Cavalcante Lima Filho, D.Sc.

Orientador: Prof. Tiago Pereira do Nascimento, Ph.D.

João Pessoa – Paraíba, Brasil

©Otacílio de Araújo Ramos Neto – otaciliodearaujo@gmail.com

Catálogo na publicação
Seção de Catalogação e Classificação

R175a Ramos Neto, Otacilio de Araujo.

Uma Abordagem Distribuída Para a Implementação de
Sistemas de SLAM Visual Baseados em Reconstrução
Geométrica / Otacilio de Araujo Ramos Neto. - João
Pessoa, 2018.

96 f. : il.

Orientação: Abel Cavalcante Lima Filho.

Coorientação: Tiago Pereira do Nascimento.

Tese (Doutorado) - UFPB/CT.

1. VSLAM, SLAM Visual, SLAM por métodos indiretos. I.
Filho, Abel Cavalcante Lima. II. Nascimento, Tiago
Pereira do. III. Título.

UFPB/BC

UMA ABORDAGEM DISTRIBUIDA PARA A IMPLEMENTAÇÃO DE SISTEMAS DE SLAM VISUAL BASEADOS EM RECONSTRUÇÃO GEOMÉTRICA

por

OTACÍLIO DE ARAÚJO RAMOS NETO

Tese aprovada em 30 de agosto de 2018

Prof. Dr. ABEL CAVALCANTE LIMA FILHO
Orientador - UFPB

Prof. Dr. TIAGO PEREIRA DO NASCIMENTO
Coorientador - UFPB

Prof. Dr. FRANCISCO ANTONIO BELO
Examinador Interno - UFPB

Prof. Dr. MOISÉS DANTAS DOS SANTOS
Examinador Interno - UFPB

Prof. Dr. LUIZ MARCOS GARCIA GONÇALVES
Examinadora Externa - UFRN

Prof. Dr. ANDERSON ABNER DE SANTANA SOUZA
Examinador Externo - UERN

Dedicatória

Ao aluno desconhecido.

Agradecimentos

- Aos meus pais, Maria da Conceição Alves de Araújo e Roberto Alves de Araújo pelo investimento nos estudos dos filhos;
- Aos meus orientadores, professor Abel Cavalcante Lima Filho e Tiago Pereira do Nascimento pelo apoio e auxílio;
- À minha esposa Diana Marques Martins Chacon pelo apoio e incentivo;
- Aos meus irmãos João Quirino Alves Neto e Paulo Roberto Alves de Araújo pelo apoio;
- Ao amigo Ruan Delgado Gomes pelo apoio e incentivo acadêmico;
- Aos alunos e colegas do IFPB-Campus Guarabira pelo apoio na instituição, em especial aos professores Erick Melo e Cristiano Elias;
- A Bruno Marques Ferreira da Silva pelas explicações;
- Aos professores que gentilmente aceitaram participar da banca de avaliação;
- Ao IFPB e ao povo brasileiro pela oportunidade de me dedicar exclusivamente aos estudos;
- A IHWH pela vida e oportunidade de crescimento pessoal e intelectual.

For since the fabric of the universe is most perfect, and is the work of a most wise Creator, nothing whatsoever takes place in the universe in which some relation of maximum and minimum does not appear.

— L. Euler

Resumo

Os sistemas de SLAM Visual são componentes importantes para a navegação de veículos autônomos por simplificarem o problema do planejamento de trajetória, transformando-o na verificação de todos os espaços navegáveis para o local desejado. Atualmente, existem várias soluções para o problema do SLAM, como SLAM por EKF, SLAM por filtro de partículas e o *Graph SLAM*. O *Graph SLAM* usando câmeras como sensores é um método de SLAM Visual intensamente pesquisado nos últimos anos devido à disponibilidade de câmeras de baixo custo e à quantidade significativa de informações do ambiente que elas disponibilizam. Muitos dos sistemas de SLAM Visual empregam reconstrução geométrica baseados no paradigma de processamento paralelo desenvolvido no PTAM. Este tipo de sistema não é escalável para processadores com mais de dois núcleos. Os vários métodos de SLAM baseados no PTAM também não foram projetados para escalar para todos os núcleos de processadores modernos nem para executar como um sistema distribuído. Este trabalho propõe uma modificação na arquitetura deste tipo de método de SLAM para que possa escalar para todos os núcleos de um processador ou para os processadores de um sistema multiprocessador fracamente acoplado. O sistema modificado de acordo com a técnica proposta neste trabalho mostrou uma diminuição considerável no tempo necessário para estimar a posição da câmera e a posição dos pontos de referência. Esse desempenho aprimorado resultou da capacidade adicional de escalar para todos os núcleos de processador, bem como para o sistema de multiprocessadores fracamente acoplados.

Palavras-chave: VSLAM, SLAM Visual, SLAM por métodos indiretos e reconstrução geométrica.

Abstract

Visual SLAM systems are important components for the navigation of autonomous vehicles as they simplify the problem of path planning, turning it into verification of all the navigable spaces to the desired location. Currently, there are several solutions to the SLAM problem, such as SLAM by EKF, SLAM by particle filter and the Graph SLAM. Graph SLAM using cameras as sensors is a Visual SLAM method intensively researched in recent years due to the availability of low-cost cameras and the significant amount of environmental information they make accessible. Most of the Visual SLAM systems employs geometric reconstruction based on the parallel processing paradigm developed in the PTAM. This type of system is not scalable for processors with more than two cores. The various PTAM-based SLAM methods were also not designed to scale to all modern processor cores or to execute as a distributed system. This work proposes a modification in the architecture of this type of SLAM method so that it can scale to all processor's cores or the processors of a weakly coupled multiprocessor system. The modified system according to the technique proposed in this work showed a considerable decrease in the time needed to estimate the position of the camera and the position of the reference points. This improved performance resulted from the additional ability to scale to all processor's cores as well the weakly coupled multiprocessor system.

Keywords: VSLAM, Visual SLAM, SLAM by indirect methods and geometric reconstruction.

Sumário

1	Introdução	1
1.1	Metodologia	3
1.2	Hipótese	5
1.3	Contribuições e Aplicações	6
1.4	Organização do Texto	6
2	Estado da Arte	7
2.1	Tipos de Sistemas de SLAM	7
2.2	SLAM Visual	9
2.2.1	Odometria Visual por Métodos Diretos	10
2.2.2	Odometria Visual por Métodos Semi-Diretos	11
2.2.3	Odometria Visual por Métodos Indiretos	11
2.2.4	SLAM por Reconstrução Geométrica	11
3	Fundamentos Teóricos	16
3.1	Elementos em um Sistema de SLAM Visual	16
3.1.1	Modelagem da Pose do Robô	17
3.1.2	Modelagem da Câmera	21
3.1.3	Modelagem do Mapa	23
3.2	Componentes do Sistema de SLAM Visual	23
3.2.1	Identificação de Pontos de Interesse	24
3.2.2	Cálculo dos Descritores	25
3.2.3	Associação de Pontos de Interesse	26
3.2.4	Estimação de Movimento	26
3.2.5	Triangulação dos Pontos de Referência	27
3.2.6	Adição de <i>Keyframes</i>	27
3.2.7	Ajuste de Feixe Local	28
3.2.8	Detecção de Fechamento de Laço	30

3.2.9	Ajuste de Feixe Global	31
3.2.10	Detecção de “Robô Sequestrado”	31
3.3	Algoritmo de SLAM Visual	32
4	Modificações Propostas	37
4.1	Pipeline de Processamento de Imagens	37
4.2	Proposta de Melhoria	40
4.3	Distribuição das Imagens	42
4.4	Remontagem dos Dados	44
4.5	ORB-SLAM	48
4.6	Modificações	48
4.7	Outras Aplicações	49
5	Experimentos	50
5.1	Experimento 1	50
5.1.1	Discussão	59
5.2	Experimento 2	60
5.2.1	Discussão	69
5.3	Experimento 3	69
5.3.1	Discussão	78
5.4	Comentários Gerais	79
6	Conclusões	81
A	Algoritmos	92

Lista de Figuras

2.1	Arquitetura do PTAM	13
2.2	Arquitetura do ORB-SLAM	14
2.3	Arquitetura do S-PTAM	14
3.1	Representação do corpo rígido	18
3.2	Exemplo de grafo do SLAM Visual	23
3.3	Associação de pontos de interesse	26
4.1	Sequência de operações da linha de execução 1	38
4.2	Tempo para estimação da pose	39
4.3	<i>Outliers</i> do tempo para estimação da pose	39
4.4	Sequência de operações com uma linha de execução	41
4.5	Sequência de operações com duas linhas de execução	42
4.6	Modificação do ORB-SLAM para adição de duas linhas de execução	43
4.7	Árvore vermelho-preto para remontagem de imagens	45
4.8	Correspondência entre <i>threads</i> e objetos	48
5.1	Perdas de rastreamento com o sistema executando em uma única máquina.	51
5.2	Tempo médio para estimação da pose com o sistema executando em uma única máquina.	52
5.3	Diagrama de caixa do tempo para estimação da pose com o sistema executando em uma única máquina.	53
5.4	Tempo médio que uma imagem permanece no <i>pipeline</i> com o sistema executando em uma máquina.	54
5.5	Diagrama de caixa do tempo que uma imagem permanece no <i>pipeline</i> com o sistema executando em uma máquina.	54
5.6	Uso de CPU com todo o sistema executando em uma única máquina.	55
5.7	Mapa de referência com o sistema executando em uma máquina.	56
5.8	Erro de translação percentual contra a distância percorrida.	57

5.9	Erro de translação percentual contra a velocidade.	57
5.10	Erro de rotação grau/metro.	58
5.11	Erro de rotação grau/metro contra a velocidade.	58
5.12	Número de perdas de rastreo para processos executando no processador i7 3517U	61
5.13	Tempo para a estimação da pose de forma distribuída para processos executando no processador i7 3517U	61
5.14	Diagrama de caixa do tempo para a estimação da pose para processos executando no processador i7 3517U	62
5.15	Ajuste da curva para o tempo médio de estimação da pose para processos executando no processador i7 3517U	63
5.16	Tempo médio da imagem no <i>pipeline</i> para processadores i7	63
5.17	Diagrama de caixa do tempo da imagem no <i>pipeline</i> para processadores i7	64
5.18	Uso de CPU para execução distribuída em processadores i7	65
5.19	Comparação com o mapa de referência para o sistema executando em processadores i7.	65
5.20	Erro de translação percentual contra a distância percorrida.	66
5.21	Erro de translação percentual contra a velocidade.	67
5.22	Erro de rotação grau/metro.	67
5.23	Erro de rotação grau/metro contra a velocidade.	68
5.24	Gráfico da perda de rastreo para processos executando nos processadores ARM	70
5.25	Tempo para estimação da pose para processadores de menor desempenho	71
5.26	Diagrama de caixa do tempo para estimação da pose usando processadores de menor desempenho	71
5.27	Ajuste da curva do tempo para estimação da pose para processadores de menor desempenho	72
5.28	Tempo da imagem dentro do <i>pipeline</i> para processadores ARM	73
5.29	Diagrama de caixa do tempo dentro do <i>pipeline</i> para processadores ARM	73
5.30	Gráfico do uso de CPU para extração de pontos e cálculo de descritores executando nos processadores de menor desempenho	74
5.31	Comparação da trajetória obtida para o sistema distribuído executando em processadores de menor desempenho.	75
5.32	Erro de translação percentual contra a distância percorrida.	76
5.33	Erro de translação percentual contra a velocidade.	76
5.34	Erro de rotação grau/metro para a distância percorrida.	77
5.35	Erro de rotação grau/metro contra a velocidade.	77

Lista de Tabelas

5.1	Perdas de rastreio e erro RMS para a execução na mesma máquina.	59
5.2	Perdas de rastreio e erro RMS para processos executando no processador i7 3517U.	68
5.3	Perdas de rastreio e erro RMS para processos executando em processadores de menor desempenho.	78

Lista de Algoritmos

3.1	Pseudo código do algoritmo de estimação da rotação e translação entre duas poses	33
3.2	Algoritmo de <i>front-end</i> do SLAM Visual Monocular	34
3.3	Esboço do algoritmo de Levenberg-Marquardt com o cálculo do incremento para poses usando variedades lineares	35
3.4	Esboço do algoritmo para cálculo do gradiente e Hessiano para o sistema linear esperso	36
4.1	Algoritmo de envio de imagens	44
4.2	Algoritmo de adição de dados na árvore vermelho-preto	45
4.3	Algoritmo de recuperação dos pontos de interesse e descritores da árvore vermelho-preto	46
4.4	Algoritmo de recebimento das imagens	47
A.1	Estima rotação e translação	93
A.2	Algoritmo de <i>front-end</i> do SLAM Visual Monocular	94
A.3	Algoritmo de Levenberg-Marquardt com o cálculo do incremento para poses usando variedades lineares	95
A.4	Algoritmo para cálculo do gradiente e Hessiano para o sistema linear esperso	96

Lista de Siglas

DSO	–	<i>Direct Sparse Odometry</i>
DTAM	–	<i>Dense tracking and mapping</i>
EKF	–	<i>Extended Kalman Filter</i>
FPGAs	–	<i>Field-Programmable Gate Array</i>
GPS	–	<i>Global Positioning System</i>
GPU	–	<i>Graphics Processing Unit</i>
IEEE	–	<i>Institute of Electrical and Electronics Engineers</i>
KITTI	–	<i>Karlsruhe Institute of Technology</i>
LSD-SLAM	–	<i>Large-Scale Direct Monocular SLAM</i>
OpenCV	–	<i>Open Source Computer Vision</i>
ORB-SLAM	–	<i>Oriented FAST and Rotated BRIEF SLAM</i>
ORB	–	<i>Oriented FAST and Rotated BRIEF</i>
PTAM	–	<i>Parallel Tracking and Mapping</i>
REMODE	–	<i>REgularized MOnocular Depth Estimation</i>
ScaViSLAM	–	<i>Scale Visual SLAM</i>
SLAM	–	<i>Simultaneous Localization and Mapping</i>
S-PTAM	–	<i>Stereo Parallel Tracking and Mapping</i>
VSLAM	–	<i>Visual Simultaneous Localization and Mapping</i>

Lista De Símbolos

P	–	Conjunto de parâmetros que definem o estado do sistema
$\hat{P}$	–	Valor estimado do conjunto de parâmetros que definem o estado do sistema
ϵ	–	Vetor diferença entre o valor da função f para os valores corretos dos parâmetros e os estimados
$\tau_{\times}$	–	Matriz antissimétrica $\begin{bmatrix} 0 & -\tau_z & \tau_y \\ \tau_z & 0 & -\tau_x \\ -\tau_y & \tau_x & 0 \end{bmatrix}$
F	–	Matriz fundamental
E	–	Matriz essencial
α_x	–	Fator de escala em x
α_y	–	Fator de escala em y
γ	–	Parâmetro inclinação da câmera
δ_x	–	Ponto principal em x
δ_y	–	Ponto principal em y
x'	–	Posição calibrada do pixel x
y'	–	Posição calibrada do pixel y
$\mathbf{x}'$	–	Vetor calibrado $[x' \ y' \ 1]^T$
$\Delta\Theta$	–	Incremento para os parâmetros do estado do sistema
$\bar{\mathbf{p}}^*$	–	Valores otimizados para as poses na janela de otimização
$\bar{\mathbf{X}}^*$	–	Valores otimizados para os pontos de referência na janela de otimização
ν	–	Parâmetro do algoritmo de Levenberg–Marquardt
α	–	Parâmetro de confiança para aceitação ou rejeição da hipótese nula.

CAPÍTULO 1

Introdução

Ao implementar o *software* de um robô móvel como um sistema distribuído pode-se aumentar a quantidade de recursos computacionais disponíveis para o processamento e armazenamento de dados, além de permitir a integração de componentes fixos para execução remota de *software*, com uma fonte de energia virtualmente ilimitada. A possibilidade de adição de recursos são muitas, da mesma forma que os benefícios, e em muitas áreas, tais como: acesso à bibliotecas de imagens, mapas, trajetórias e dados de objetos (*big data*); acesso a sistemas de computação paralela sob demanda para análise estatística, aprendizagem e planejamento de movimento. Desde que possuam meios para a transmissão de dados, robôs de pequeno porte e baixo custo podem ter suas limitações óbvias de poder computacional, armazenamento e fonte de energia contornados, permitindo um melhoramento significativo no seu funcionamento.

Além das aplicações tradicionais de aprendizado coletivo, compartilhamento de trajetória e estratégias de controle, os sistemas de mapeamento e localização simultâneos podem obter vantagens quando implementados como um sistema distribuído. Devido a sua complexidade, e necessidade destes sistemas de criar mapas cada vez maiores, implementações estritamente embarcadas apresentam um gargalo natural, causado pelos recursos limitados, que é quebrado naturalmente em uma implementação distribuída. Sendo esta uma boa estratégia para a solução do problema do SLAM em robôs de baixo custo.

O problema do Mapeamento e Localização Simultâneos (SLAM) é tido como fundamental para a robótica móvel e, por isso, tem sido pesquisado exhaustivamente nos últimos anos (DHIMAN; DEODHARE; KHEMANI, 2015; FUENTES-PACHECO; RUIZ-ASCENCIO; RENDÓN-MANCHA, 2015; TAKETOMI; UCHIYAMA; IKEDA, 2017). Sua importância advém, dentre outras razões, da necessidade de operação autônoma dos robôs, pois quando estes possuem um mapa do ambiente ou podem construí-lo, o problema da cobertura torna-se na verificação de todos os espaços navegáveis até que o local desejado seja encontrado ou que todo o mapa seja coberto (MATARIĆ, 2007). Além disso, a navegação

pode ser feita de forma independente de sinais de GPS, aumentando assim a capacidade de operação autônoma em áreas de sombra, tais como minas e locais de desabamento. Dessa forma, a autonomia é necessária em vários níveis de funcionalidade, seja no processamento de dados de sensores para obter informação do ambiente, na formulação de planos para alcançar objetivos e no planejamento de rotas para deslocar-se até uma localização desejada (DHIMAN; DEODHARE; KHEMANI, 2015).

O SLAM possui uma variante, chamada de SLAM Visual (VSLAM¹), na qual imagens são utilizadas como fonte de informação externa (DAVISON, 2003; LATEGAHN; GEIGER; KITT, 2011; FUENTES-PACHECO; RUIZ-ASCENCIO; RENDÓN-MANCHA, 2015). Nesta, quando se usa a abordagem baseada em referências, são extraídos pontos de interesse ao processar as imagens. Esses pontos são a pedra fundamental dos sistemas de SLAM Visual baseados em odometria visual (NISTER; NARODITSKY; BERGEN, 2004; SCARAMUZZA; FRAUNDORFER, 2011) que utilizam algoritmos de estimação de movimento, como o algoritmo dos cinco pontos (NISTER, 2004) ou o algoritmo dos oito pontos (HARTLEY, 1997), no processo de reconstrução geométrica e mapeamento. Não obstante, além da estimação de movimento, os sistemas de SLAM baseados em reconstrução geométrica normalmente possuem uma etapa de otimização do mapa por meio do ajuste não linear da localização tridimensional dos pontos de referência e da pose do robô (PRINCE, 2012; HARTLEY; ZISSERMAN, 2004; TAKETOMI; UCHIYAMA; IKEDA, 2017; KÜMMERLE et al., 2011). Esta etapa ocorre localmente ou por todo um caminho já percorrido, quando é detectado um fechamento de laço. A detecção de fechamento de laço também é uma etapa fundamental em sistemas modernos de SLAM Visual (GUCLU; CAN, 2017) sendo que uma forma de implementá-la é por meio do uso de *bag of words*², onde as palavras visuais são os descritores. A realização de todas essas operações demandam de recursos computacionais consideráveis (principalmente quando implementadas em sistemas embarcados), por envolverem operações matemáticas elaboradas tais como extração de pontos de interesse, cálculo de descritores e ajustes de poses e pontos de referência (KLEIN; MURRAY, 2007). Sendo que, além disso, elas devem ser realizadas várias vezes por segundo em aplicações de tempo real (RUBLEE et al., 2011; SINHA et al., 2006). Caso o desempenho não seja suficiente para estes requisitos, o sistema é obrigado a deslocar-se mais lentamente para não perder o rastreo dos pontos de referência e produzir um mapa incompleto. Dessa forma, os sistemas de SLAM Visual baseados em reconstrução geométrica possuem como um dos fatores limitantes da velocidade de deslocamento a velocidade com que realizam a estimação da pose da câmera e posição dos pontos de referência visualizados em cada pose. Pois, quanto mais rápido a câmera se desloca, mais distantes espacialmente estão as poses

¹Existe um algoritmo chamado de vSLAM (KARLSSON et al., 2005), mas que também usa a odometria das rodas.

²Representação que tenta caracterizar uma imagem inteira ou grande parte dela resumindo a estatística dos descritores de todos os pontos de interesse da região (PRINCE, 2012).

em que as imagens foram capturadas, diminuindo assim o número de correspondências entre os pontos de interesse encontrados nas sequências de imagens.

O objeto de estudo desta tese consiste em uma proposta de modificação do paradigma normalmente empregado na construção dos sistemas de SLAM Visual baseados em reconstrução geométrica. Ao se implementar o *software* de SLAM Visual usando uma arquitetura construída com base no paradigma proposto, o sistema deve aumentar a taxa de processamento de imagens e velocidade de reconstrução da trajetória, permitindo assim um deslocamento mais rápido da câmera. Além disso, devido ao novo paradigma, o sistema de SLAM Visual pode ser construído como um sistema distribuído e escalável de acordo com os recursos disponíveis no computador ou rede em que ele está executando. Dessa forma, estas novas propriedades tornam-o bastante adequado para execução em *clusters* de computadores locais e que crescem de acordo com os requisitos de desempenho.

1.1 Metodologia

A velocidade com que um sistema de SLAM Visual baseado em reconstrução geométrica consegue extrair pontos de interesse, calcular seus descritores e realizar a associação com os pontos de interesse de uma imagem anterior está diretamente relacionada com a velocidade com que a câmera pode se deslocar sem impossibilitar a estimação do movimento entre imagens sucessivas. Uma câmera com movimentação que demande de processamento além da capacidade do sistema fará com que a correspondência entre pontos de interesse seja insuficiente, forçando a uma redução na sua velocidade de movimentação com o objetivo de aumentar a similaridade entre as imagens. Assim, é desejável que o rastreamento seja feito no menor tempo possível de maneira a não restringir a velocidade da câmera.

A solução trivial em que se emprega uma CPU com maior capacidade de processamento pode ser impraticável porque, dependendo do desempenho desejado, a CPU adequada não é conveniente para aplicações móveis e de baixo custo. Além disso, o volume do robô, forma e outros fatores são limitações óbvias para a capacidade de processamento e armazenamento (WAN et al., 2016). Uma outra abordagem é a busca por algoritmos extratores de pontos de interesse e cálculo de descritores mais rápidos, pois sistemas de SLAM Visual podem precisar extrair centenas ou até mesmo milhares deles para cada imagem. Esta abordagem é usada em sistemas como o ORB-SLAM (MUR-ARTAL; MONTIEL; TARDÓS, 2015), que usa o extrator e descritor do tipo ORB (RUBLEE et al., 2011), resultando em um sistema com menores requisitos computacionais do que quando utilizando extratores do tipo SIFT (LOWE, 1999) ou SURF (BAY et al., 2008). Contudo, o uso do ORB não é uma solução definitiva porque, ainda assim, existe a necessidade do grande número de pontos de interesse, como também o seu uso não exclui a possibilidade de melhoramento em outras partes do algoritmo de SLAM.

Adicionalmente ao uso de detectores e descritores que demandem de menor poder computacional, sistemas de SLAM Visual baseados em reconstrução geométrica empregam um paradigma de paralelização de atividades nos vários núcleos de uma CPU. Esta abordagem foi proposta inicialmente no PTAM (KLEIN; MURRAY, 2007) e utilizada em vários outros sistemas de SLAM, como o S-PTAM (PIRE et al., 2017, 2015), ORB-SLAM, Scalable Visual SLAM (ScaViSLAM) (STRASDAT et al., 2011) e sistemas de odometria visual (SILVA; BURLAMAQUI; GONÇALVES, 2012), sendo um padrão *de facto* nesse tipo de sistema. Também é possível utilizar paralelização na extração dos pontos-chave e o cálculo dos descritores utilizando GPUs (ACHARYA; BABU; VADHIYAR, 2014; KIM et al., 2009; SHARMA, 2016), sendo que esta técnica possui implementações, inclusive em bibliotecas de código aberto, como a OpenCV (ITSEEZ, 2015). Esta é uma abordagem que apresenta bons resultados mas que também possui limitações já que nem sempre uma GPU pode ser embarcada no sistema móvel, como também apresentam consumo de energia considerável quando comparado ao processador principal. Além disso, a extração dos pontos de interesse é apenas uma parte do problema, de maneira que um extrator muito rápido pode ser limitado pelo tempo de outras tarefas do sistema, como a etapa de ajuste de poses e pontos. Por fim, também existem abordagens utilizando paralelização em FPGAs, como no trabalho de Aguilar-González e Arias-Estrada (2016).

A abordagem utilizada nesta tese é de remover da tarefa de rastreo o procedimento de associação de pontos de interesse, unificando-o com a tarefa de ajuste de poses e pontos. Adicionalmente, propõe-se também que a etapa de extração de pontos de interesse e cálculo de descritores tenha mais de uma instância de execução em paralelo, de maneira que o processamento de uma imagem não precise aguardar a conclusão da anterior para iniciar a extração dos pontos de interesse. Esta é uma quebra de paradigma com relação a sistemas como o PTAM, S-PTAM, ScaViSLAM e ORB-SLAM, que tradicionalmente mantêm a etapa de associação junto com a extração e cálculo de descritores, enquanto o ajuste local de poses e pontos executa em uma linha de execução separada. Além disso, o número variável de instâncias de extração de pontos e cálculo de descritores faz com que surja um requisito de vários núcleos de processamento, o qual é facilmente solucionado quando o *software* é escalável e executa em um sistema com vários núcleos, seja no mesmo processador, em uma rede local ou remota.

A mudança do procedimento de associação para a linha de execução do ajuste baseia-se na observação de que ele usa uma heurística que necessita de uma estimativa da pose atual da câmera para calcular a projeção provável de cada ponto, sendo que a melhor estimativa disponível é calculada pelo procedimento de ajuste de poses e pontos. Com relação a execução da etapa de extração de pontos de interesse e cálculo de descritores em várias tarefas, isto é feito de duas formas: na primeira são usadas várias linhas de execução na mesma CPU, onde cada uma é responsável pelo processamento de uma imagem; na segunda

são utilizados tanto as várias linhas como também várias CPUs, de maneira a distribuir o trabalho de extração dos pontos de interesse e cálculo dos descritores entre os vários núcleos de várias máquinas.

1.2 Hipótese

O método proposto deve ser capaz de aumentar a capacidade de processamento de imagens junto com o ajuste das poses da câmera e localização dos pontos de referência no espaço, em decorrência da capacidade de escalar para todos os núcleos de um processador, como também para todos os núcleos de todos os processadores conectados por uma rede de alta velocidade. Nos dois casos, espera-se que o sistema utilize todo o poder computacional disponível de maneira a conseguir aumentar a quantidade de poses estimadas por segundo. Essas características devem torná-lo adequado para aplicações em que o processador utilizado possui vários núcleos, ou para um ambiente onde o sistema responsável por realizar o SLAM Visual está executando em um *cluster* local e que deve escalar de acordo com os requisitos de desempenho da aplicação. Tomando como motivação estes cenários de uso, a hipótese a seguir é proposta.

Hipótese: *É possível diminuir o tempo médio necessário para a estimação da pose da câmera e rastreo de pontos de referência entre imagens capturadas em instantes de tempo consecutivos, em um sistema de SLAM Visual, paralelizando a tarefa de extração de pontos de interesse e cálculo de descritores em relação a tarefa de associação de pontos e ajuste de poses e pontos, desde que as instâncias de processamento utilizadas possam ser alocadas para execução individual nos vários núcleos de um único processador ou em um conjunto de núcleos conectados por uma rede de alta velocidade.*

A hipótese é verificada modificando o algoritmo do ORB-SLAM (MUR-ARTAL; MONTIEL; TARDÓS, 2015) e utilizando o *dataset*³ KITTI (2016). O conjunto de imagens e trajetória de referência usados são o de odometria visual 00. As métricas empregadas são o número de perdas de rastreo, o tempo para a estimação da pose, o tempo que a imagem permanece no *pipeline* desde a captura até a conclusão da estimação da pose, qualidade do mapa obtido e comportamento dos gráficos dos erros de translação e de rotação. O escopo do método é limitado ao conjunto de imagens em escala de cinza do referido *dataset* contendo poucos pedestres e veículos em movimento.

³Conjunto de imagens, parâmetros de calibração, medições de sensores e a trajetória de referência correspondente para as imagens (obtida por meio de sensores de alta precisão).

1.3 Contribuições e Aplicações

A primeira contribuição resultante deste trabalho consiste na modificação proposta ao paradigma dos sistemas de SLAM Visual baseados em reconstrução geométrica que utilizam os princípios estabelecidos no PTAM. A segunda contribuição consiste no algoritmo de divisão e distribuição das imagens junto com o algoritmo de remontagem das posições dos pontos de interesse e descritores. A terceira contribuição apresenta-se no estudo comparativo realizado entre o *software* com as modificações propostas e a implementação de referência de um sistema de SLAM Visual baseado em reconstrução geométrica, no caso o ORB-SLAM.

Além das contribuições citadas anteriormente, os resultados obtidos podem ser usados para embasar a construção de sistemas de SLAM Visual que operam como um sistema distribuído e escalável, tanto em uma única máquina como também em um conjunto de máquinas conectadas por uma rede veloz.

1.4 Organização do Texto

O restante desta tese está organizada da seguinte forma. No capítulo 2 são descritos os sistemas de SLAM Visual no estado da arte, com atenção especial aos que utilizam métodos indiretos. No capítulo 3 os fundamentos para a construção de um sistema de SLAM Visual baseado em reconstrução geométrica são apresentados. No capítulo 4 é descrito o *pipeline* de processamento de imagens e apresentado o embasamento para a melhoria proposta nesta tese. No capítulo 5 são descritos os experimentos utilizados para analisar quantitativamente e qualitativamente as melhorias propostas. Por fim, no capítulo 6 são apresentadas as conclusões do trabalho e expectativas para projetos futuros.

CAPÍTULO 2

Estado da Arte

Neste capítulo é apresentado um breve histórico do problema do SLAM, os fundamentos dos principais tipos de sistemas de SLAM e, por fim, as abordagens utilizadas na construção de sistemas de SLAM Visual.

2.1 Tipos de Sistemas de SLAM

Durante a *IEEE Robotics and Automation Conference* de 1986, a comunidade de pesquisadores em robótica reconheceu a importância do problema do mapeamento e localização simultânea como algo fundamental para a robótica móvel e com dificuldades conceituais e computacionais que precisavam ser resolvidas. Alguns avanços se sucederam nos anos seguintes, porém a comunidade abordava o mapeamento e a localização como sendo problemas distintos, atacando-os individualmente. A verdadeira quebra de paradigma ocorreu com a percepção de que a formulação do mapeamento e localização de forma simultânea, como um único problema, adicionava propriedades de convergência a este (DURRANT-WHYTE; BAILEY, 2006). A definição de sua estrutura, apresentação de resultados de convergência e a criação do acrônimo SLAM foram feitos em um artigo apresentado em 1995 no *International Symposium on Robotics Research* (DURRANT-WHYTE; RYE; NEBOT, 1996; DURRANT-WHYTE; BAILEY, 2006). Em seguida, a teoria básica para a convergência e muitos dos resultados iniciais foram desenvolvidos por Csorba Jeffrey K. Uhlmann (1996), Csorba (1997). Logo após, vários outros grupos de pesquisa que trabalhavam anteriormente em mapeamento e localização começaram a trabalhar seriamente com SLAM (DURRANT-WHYTE; BAILEY, 2006). O *Simultaneous Localization and Mapping - SLAM*, foi então bem definido e compreendido como sendo o problema de adicionar a um robô a capacidade de construir um mapa do ambiente em que ele se encontra ao mesmo tempo em que se localiza no mapa. Sua solução é reconhecidamente de grande importância para a comunidade

de robótica por verdadeiramente adicionar autonomia na navegação a um robô (ROLAND; REZA; DAVIDE, 2011).

Atualmente, o problema do SLAM em ambientes pequenos e estáticos é considerado como solucionado, de maneira que as pesquisas concentram-se no SLAM de grandes ambientes dinâmicos. Hoje, estão disponíveis vários *software* e trabalhos na literatura sobre o SLAM, sendo que estes podem ser agrupados nas categorias de soluções a seguir: o SLAM baseado no Filtro Estendido de Kalman (EKF); o SLAM baseado em filtro de partículas; *Graph SLAM* e o SLAM baseado em odometria visual.

O SLAM baseado no Filtro Estendido de Kalman é, historicamente, a proposta de solução mais antiga (ROLAND; REZA; DAVIDE, 2011), com os artigos de Smith e Cheeseman (1986), Smith, Self e Cheeseman (1990) e Moutarlier e Chatila (1989, 1990) introduzindo a sua formulação. Neste tipo de SLAM, a pose do robô p , como também todos os pontos de referência m , são modelados como variáveis aleatórias gaussianas. A solução utiliza uma abordagem probabilística, onde o sistema mantém um vetor de estados com a média de cada variável aleatória e uma matriz de covariância Σ (conforme as equações 2.1 e 2.2), que são atualizados a cada iteração t .

$$y_t \doteq [p_t \ m_0 \ \dots \ m_{n-1}]^T \quad (2.1)$$

$$\Sigma_t \doteq \begin{bmatrix} \Sigma_{p_t} & \Sigma_{p_t m_0} & \dots & \Sigma_{p_t m_{n-1}} \\ \Sigma_{m_0 p_t} & \Sigma_{m_0} & \dots & \Sigma_{m_0 m_{n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma_{m_{n-1} p_t} & \Sigma_{m_{n-1} m_0} & \dots & \Sigma_{m_{n-1}} \end{bmatrix} \quad (2.2)$$

O algoritmo calcula a crença em um estado atual, representado pela variável aleatória gaussiana multivariável, utilizando a crença no estado anterior, a entrada de controle e as medições no estado atual. Possui 4 partes principais, sendo que na primeira ele atualiza a predição do estado atual, na segunda ele realiza a associação de pontos de referência do estado atual com o estado anterior, na terceira ele atualiza o vetor de estados a partir da associação de dados e na quarta ele adiciona novos pontos de referência ao vetor de estados. O algoritmo completo para um robô com tração diferencial pode ser encontrada no trabalho de Thrun, Burgard e Fox (2005) e Roland, Reza e Davide (2011). Uma descrição mais genérica, aplicável a outros tipos de robôs, pode se encontrada no trabalho de Fernández-Madrigal e Claraco (2012).

Os algoritmos de SLAM baseados em filtros de partículas utilizam uma representação para a crença no estado do sistema na forma de K possíveis estados diferentes, os quais se convencionou chamar de partículas. Cada partícula $X^{[k]}$ armazena uma estimativa do caminho do robô $\dot{p}^{[k]}$ e um conjunto de estimativas da posição de cada ponto de referência do

mapa (ROLAND; REZA; DAVIDE, 2011) para o instante t , conforme ilustrado pela equação 2.3.

$$X_t^{[k]} \doteq \{p_t^{[k]}, N(\mu_{t,0}^{[k]}, \Sigma_{t,0}^{[k]}); N(\mu_{t,1}^{[k]}, \Sigma_{t,1}^{[k]}); \dots; N(\mu_{t,n-1}^{[k]}, \Sigma_{t,n-1}^{[k]})\} \quad (2.3)$$

Foi graças ao trabalho de Blackwell (1947) e RAO (1945) que os filtros de partículas puderam ser empregados no SLAM (sendo por isso chamados de Rao-Blackwellized), com sua primeira aplicação direta no problema feita por Doucet et al. (2000). Atualmente, uma implementação bastante eficiente é a de Montemerlo et al. (2002, 2003) que criou o termo *FastSLAM* (ROLAND; REZA; DAVIDE, 2011). Uma descrição completa do algoritmo também pode se encontrada no trabalho de Thrun, Burgard e Fox (2005).

A formulação do problema do SLAM baseado em grafos surgiu das aplicações em cenários de larga escala nos quais as abordagens utilizando Filtro Estendido de Kalman e Filtro de Partículas se mostraram ineficientes devido ao crescimento de forma quadrática ou exponencial da complexidade matemática destes algoritmos (FERNÁNDEZ-MADRIGAL; CLARACO, 2012). A formulação baseada em grafos foi proposta primeiro por Lu e Milios (1997), contudo, levou algum tempo para ela se tornar popular devido a dificuldades com a operação de minimização do erro usando as técnicas de álgebra linear disponíveis na época (GRISSETTI et al., 2010). A ideia fundamental é que o problema do SLAM pode ser modelado como um grafo onde os vértices são as poses ao longo do tempo e as arestas representam as restrições entre essas poses. No *Graph SLAM*, as arestas não correspondem necessariamente a medições, mas à informação obtida ao marginalizar o grafo original de medições e estimação do movimento, criando, inclusive, arestas entre nós que não estavam previamente conectados (FERNÁNDEZ-MADRIGAL; CLARACO, 2012). De posse do grafo, o *software* de SLAM realiza uma maximização da verossimilhança das poses.

2.2 SLAM Visual

O SLAM Visual é o principal tipo abordado neste trabalho. Esta não é necessariamente uma nova categoria, pois o que diferencia um tipo de outro é, principalmente, o motor estatístico utilizado na estimação dos valores das poses e pontos. O SLAM Visual pode utilizar como motor estatístico qualquer um dos citados na seção 2.1 (entre outros), contudo é comum o uso daqueles baseados em grafos. Uma dificuldade adicional desses sistemas que utilizam imagens é que, câmeras monoculares ou estéreo não fornecem informações de distância dos pontos de referência para a câmera. Dessa forma, a profundidade precisa ser calculada a partir da projeção do ponto de referência na imagem.

Alguns sistemas de SLAM Visual são construídos tomando como base sistemas de odometria visual, adicionando os componentes necessários a um sistema de SLAM completo. A odometria visual empregada pode ser dividida em três grupos, de acordo com a forma de análise das imagens, as que utilizam os chamados métodos indiretos, as que utili-

zam os métodos semi-diretos e as que utilizam os métodos diretos. A odometria visual por métodos indiretos realiza um processamento prévio da imagem em busca de elementos identificáveis entre imagens sucessivas. De posse desses elementos pode-se aplicar algoritmos de estimação de movimento da câmera e reconstrução tridimensional de cena, combinados com várias heurísticas, para realizar a estimação do movimento da câmera e reconstrução geométrica da posição dos pontos de interesse. Já a odometria visual por métodos diretos não possui essa etapa de processamento prévio, fazendo uso diretamente dos valores dos *pixels* da imagem em um modelo probabilístico para a estimação do movimento. Por fim, os métodos semi-diretos combinam técnicas dessas duas abordagens.

2.2.1 Odometria Visual por Métodos Diretos

Os métodos diretos baseiam-se na minimização do erro fotométrico das câmeras utilizadas (ENGEL; KOLTUN; CREMERS, 2017). Para isso, aplica-se uma modelagem probabilística para estimar a profundidade dos *pixels* analisados (FORSTER; PIZZOLI; SCARAMUZZA, 2014) e calcula-se o movimento aproximado da câmera (ENGEL; STURM; CREMERS, 2013). Os métodos diretos podem ser divididos em grupos menores de acordo com o emprego, ou não, de um modelo probabilístico para a estimação da geometria. Dependendo dos modelos, pode ser necessário o uso de GPUs na sua implementação. Dessa forma, tem-se buscado estratégias para eliminar este requisito obtendo-se algoritmos que possam operar on-line, sem a necessidade de GPUs.

A odometria visual por métodos diretos é dita *densa* ou *semi-densa* quando sua formulação faz uso de uma combinação de erro fotométrico junto com um modelo de probabilidade a priori para a estimação da geometria (ENGEL; KOLTUN; CREMERS, 2017). Como exemplo de sistemas que usam essa abordagem pode-se citar o LSD-SLAM (ENGEL; SCHÖPS; CREMERS, 2014), REMODE (PIZZOLI; FORSTER; SCARAMUZZA, 2014) e o DTAM (NEWCOMBE; LOVEGROVE; DAVISON, 2011). No LSD-SLAM, os objetos de reconstrução são limitados às áreas que possuem um gradiente de intensidade, dessa forma o LSD-SLAM ignora áreas sem texturas, diferentemente do DTAM que reconstrói áreas completas (TAKETOMI; UCHIYAMA; IKEDA, 2017). O resultado é a diminuição dos requisitos computacionais, de maneira que o LSD-SLAM consegue executar em tempo real sem a necessidade de implementações empregando GPUs, tal como faz o DTAM e o REMODE.

A odometria visual por métodos diretos é dita *esparsa* quando utiliza apenas a minimização do erro fotométrico sem o uso de modelos de probabilidade a priori geométricos (ENGEL; KOLTUN; CREMERS, 2017). Seu emprego com um método de otimização não linear foi feito primeiramente no DSO (ENGEL; KOLTUN; CREMERS, 2017), contudo a proposta inicial, feita por Jin, Favaro e Soatto (2003), empregava filtro estendido de Kalman.

2.2.2 Odometria Visual por Métodos Semi-Diretos

A odometria visual por métodos semi-diretos emprega técnicas tanto dos métodos diretos quanto dos indiretos. Tal como no primeiro, faz uso do rastreo e triangulação de *pixels* que são caracterizados por um grande gradiente de imagem. Já dos métodos indiretos, utilizam a abordagem do modelo baseado em referências para a otimização conjunta do movimento da câmera e estrutura do ambiente (FORSTER; PIZZOLI; SCARAMUZZA, 2014). A primeira implementação que fez uso dos métodos semi-diretos, empregando-os em odometria visual, foi feita no SVO (FORSTER; PIZZOLI; SCARAMUZZA, 2014).

2.2.3 Odometria Visual por Métodos Indiretos

A Odometria Visual por métodos indiretos possui uma etapa de processamento prévio da imagem onde tenta-se identificar elementos específicos, tal como pontos ou linhas, que se repetem em imagens sucessivas. De posse do rastreo desses elementos, estima-se a localização inicial no espaço para então executar uma etapa de minimização do erro de reprojeção, obtido pela avaliação de um modelo geométrico da pose e funcionamento da câmera, com o objetivo de maximizar a verossimilhança dos valores desconhecidos. A formulação original foi criada por Brown (1958), considerando apenas as observações das poses para os pontos de referência, com as primeiras aplicações sendo feitas no contexto de reconstrução visual de mapas cartográficos a partir de imagens aéreas (TRIGGS et al., 2000)(FERNÁNDEZ-MADRIGAL; CLARACO, 2012).

2.2.4 SLAM por Reconstrução Geométrica

Os sistemas de odometria visual que empregam métodos indiretos podem ser usados como componente básico para a construção de sistemas de SLAM Visual por reconstrução geométrica. Para isso, é necessário adicionar os componentes ausentes, mas necessários para estes sistemas. Isso inclui a capacidade de armazenar grandes mapas, o suporte à detecção de fechamento de laço; e o tratamento do problema do robô sequestrado. Finalmente, sistemas de SLAM Visual baseados em reconstrução geométrica empregam um grafo que relaciona um conjunto de poses e elementos no espaço como sendo o modelo de mapa. Como exemplo de sistemas baseados em reconstrução geométrica pode-se citar o PTAM (KLEIN; MURRAY, 2007), S-PTAM (PIRE et al., 2017), ScaViSLAM (STRASDAT et al., 2011), ORB-SLAM (MUR-ARTAL; MONTIEL; TARDÓS, 2015) e ORB-SLAM2 (MUR-ARTAL; TARDÓS, 2017). Dos sistemas de SLAM Visual, aqueles que empregam reconstrução geométrica são utilizados com mais frequência para aplicações on-line devido ao seu melhor desempenho que as primeiras abordagens que utilizavam métodos diretos densos.

As primeiras implementações de SLAM Visual por reconstrução geométrica não podiam realizar o ajuste de poses e pontos e processar as imagens ao mesmo tempo porque a grande maioria dos computadores possuía processadores com apenas um núcleo. Esses sistemas apresentavam dificuldades para operar on-line, pois um conjunto considerável de operações precisavam ser realizadas no intervalo de tempo entre a captura e processamento de duas imagens consecutivas. A mudança de paradigma veio com os processadores com mais de um núcleo e a publicação do artigo *Parallel Tracking and Mapping for Small AR Workspaces* (KLEIN; MURRAY, 2007), onde foi descrito a técnica utilizada no PTAM¹. Sendo a sua grande contribuição a nova abordagem utilizada para as tarefas de mapeamento e localização. No PTAM, existem duas linhas de execução trabalhando em paralelo onde, de uma forma sucinta, uma é responsável pelo rastreamento de pontos de referência nas imagens e a outra por realizar a otimização da posição dos pontos de referência e das poses da câmera no espaço. Mais especificamente, uma linha emprega heurísticas na associação de pontos, realizando a estimação da posição esperada do ponto de interesse em uma imagem a partir da projeção do ponto de referência por meio da transformação correspondente ao deslocamento mais recente da câmera e da matriz de calibração. A outra linha realiza o processo de minimização não linear do erro de reprojeção dos valores iniciais utilizando uma abordagem de janela deslizando. Assim, apenas as últimas poses são empregadas no processo, de maneira a tornar a solução factível para os requisitos de operação on-line. Uma ilustração das operações realizadas pelas linhas de execução do PTAM é exibida na figura 2.1.

¹O PTAM é um sistema idealizado para aplicações de realidade aumentada que consegue mapear pequenos ambientes.

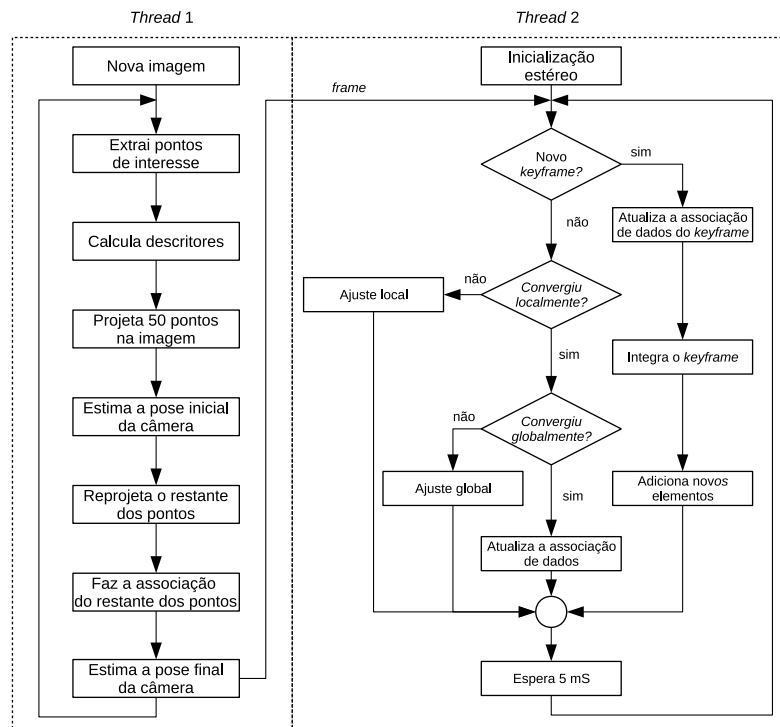


Figura 2.1: Ilustração do fluxograma e organização do PTAM em duas linhas de execução. Baseado no trabalho de Klein e Murray (2007).

Embora o PTAM tenha introduzido ideias importantes, seu projeto é para aplicações em sistemas de realidade aumentada. Isto é um fator limitante quanto ao seu emprego no mapeamento e localização simultâneos de grandes áreas. Logo, foram criados *software* de SLAM que utilizam as ideias presentes no PTAM, mas também adicionaram outras técnicas com o objetivo de realizar o mapeamento de grandes áreas enquanto se localizam nelas. O ORB-SLAM, o ORB-SLAM2² (MUR-ARTAL; TARDÓS, 2017), o ScaViSLAM (STRASDAT et al., 2011) e o S-PTAM são exemplos desse tipo de sistema. O ORB-SLAM e depois o ORB-SLAM2 utilizam várias ideias do PTAM, sendo que adicionaram, entre outras coisas, o uso do extrator de pontos de interesse e cálculo de descritores ORB e o suporte à detecção de fechamento de laço por meio de *bag of words*, junto com a otimização de trajetória pertencente ao laço. A detecção de fechamento de laço e respectiva otimização foram implementados em uma terceira linha de execução não existente no PTAM original. Além disso, o ORB-SLAM realiza a otimização por meio da biblioteca g2o (KÜMMERLE et al., 2011). Uma ilustração das principais linhas de execução existentes no ORB-SLAM, junto com as operações realizadas por elas, é apresentada na figura 2.2.

²Uma extensão do ORB-SLAM com suporte à câmeras estéreo e RGB-D.

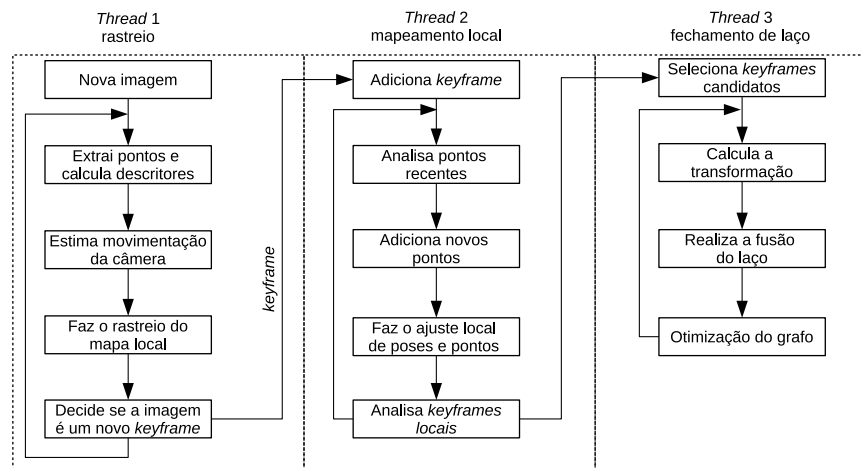


Figura 2.2: Ilustração do fluxograma e organização em linhas de execução do ORB-SLAM. Baseado no trabalho de Mur-Artal, Montiel e Tardós (2015).

O terceiro *software* de SLAM abordado, baseado em reconstrução geométrica, é o S-PTAM. Este, tal como o ORB-SLAM, estende as ideias desenvolvidas no PTAM com relação a estratégia de rastreo e mapeamento paralelo e, também, adiciona uma terceira linha de execução responsável por detectar o fechamento de laço na trajetória. Adicionalmente, o S-PTAM emprega duas câmeras para funcionamento com visão estéreo. Isto permite a criação do mapa com as medidas próximas aos valores reais, ao invés de relacionadas por um fator de escala, e evita o problema da inicialização do mapa, bastante comum em sistemas monoculares (PIRE et al., 2017). Pode-se ver a organização em linhas de execução com as respectivas tarefas empregadas no S-PTAM na figura 2.3.

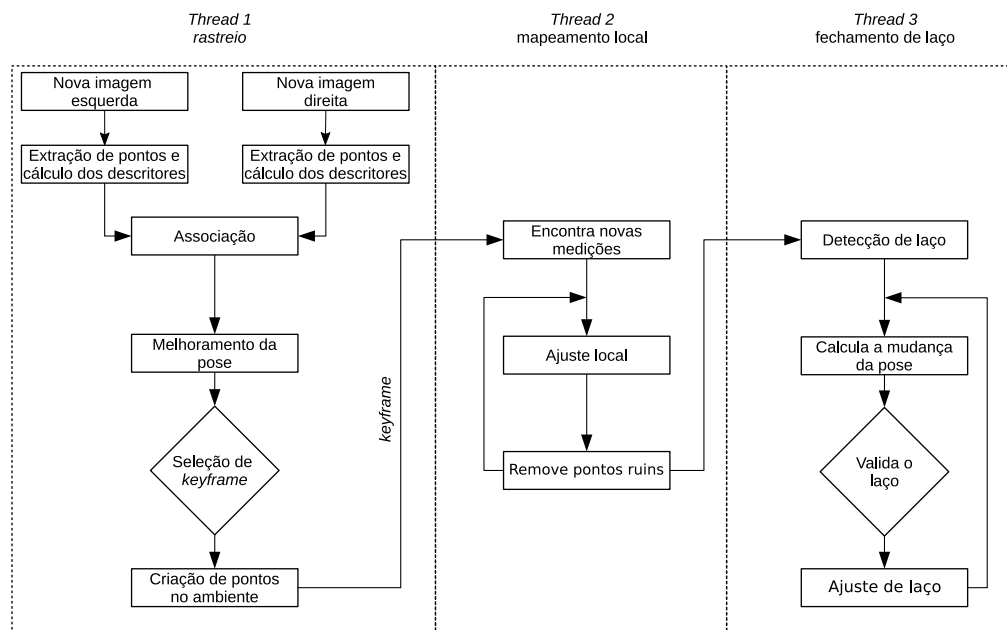


Figura 2.3: Ilustração simplificada do fluxograma e organização em linhas de execução do S-PTAM. Baseado no trabalho de Pire et al. (2017).

Outro *software* de SLAM é o ScaViSLAM de Strasdat et al. (2011). Este emprega um esquema de otimização em janela dupla e restrições leves que os autores afirmam ser superior à abordagem com janela ativa e quadros fixos. Contudo, o ScaViSLAM mantém a organização que apresenta a divisão entre *frond-end* e *back-end* por também estender as ideias básicas do PTAM.

Observando as figuras 2.1, 2.2 e 2.3 e a descrição do ScaViSLAM percebe-se que todos eles possuem uma linha de execução responsável por realizar o rastreo de pontos entre imagens sucessivas, uma segunda linha responsável por fazer um ajuste de uma mapa local e, no caso do ORB-SLAM e S-PTAM, uma terceira linha responsável por fazer a detecção de fechamento de laço. Assim, é nítido que estes sistemas baseados em reconstrução geométrica empregam uma estrutura de divisão básica em no mínimo duas linhas de execução, adicionando uma terceira para a detecção de fechamento de laço seguida da otimização da trajetória do laço. Esta divisão foi feita em decorrência da observação de que o processo de otimização pode ser demorado em relação ao tempo entre a captura das imagens, impedindo assim a operação on-line.

A organização interna desses sistemas apresentam uma grande semelhança na divisão das tarefas em, pelo menos, duas linhas de execução. Uma responsável pelo rastreo de pontos de referência e a outra pelo ajuste de poses e pontos. Esta organização repete-se nas três implementações descritas, com blocos básicos apresentando funções semelhantes. Em todas elas os blocos responsáveis pela extração de pontos de interesse e cálculo de descritores estão presentes na mesma linha de execução. A partir destas observações, o capítulo 4 apresenta o melhoramento proposto neste trabalho.

CAPÍTULO 3

Fundamentos Teóricos

Entender o que é o problema do SLAM em sua essência é o principal passo para solucioná-lo. No capítulo 2 foram citadas quatro abordagens, onde cada uma é mais adequada para a aplicação em uma determinada situação, por possuir características específicas que facilitam a sua implementação. Além disso, essas abordagens não estão associadas necessariamente a um tipo de sensor, embora a própria técnica do *Bundle Adjustment*¹ tenha suas raízes nos trabalhos de visão computacional. Assim, a literatura relata que técnicas como o EKF e o filtro de partículas são adequadas para SLAM em ambiente pequenos e médios (FERNÁNDEZ-MADRIGAL; CLARACO, 2012). O SLAM baseado em filtro de partículas também é tido como um dos mais fáceis de implementar (THRUN; BURGARD; FOX, 2005). Já o *Graph SLAM*, que nasceu da necessidade de realizar o SLAM em grandes ambientes, tem bons resultados nas aplicações de mapeamento de grandes áreas, com pesquisadores relatando o fechamento de laço em mapas com cerca de um milhão de pontos de referência (FRESE; SCHRODER, 2006)(FERNÁNDEZ-MADRIGAL; CLARACO, 2012) quando utilizando este tipo de SLAM. Por fim, o SLAM Visual com *bundle adjustment* tanto consegue modelar grandes ambientes como emprega câmeras, pois é nas aplicações com este tipo de sensor que ele tem suas raízes.

3.1 Elementos em um Sistema de SLAM Visual

Muito pode ser dito sobre o que é o SLAM baseado em reconstrução geométrica, mas a forma compreensível de descrevê-lo é como um grande problema de otimização. Sendo o mapa formado por um conjunto pontos de referência no espaço e a trajetória percor-

¹Método bastante popular na comunidade de visão computacional empregado para estimar simultaneamente a pose da câmera e os pontos de referência (FERNÁNDEZ-MADRIGAL; CLARACO, 2012).

rida a sequência de poses, o que se procura como solução do problema é encontrar o conjunto de parâmetros $\hat{P}$, correspondentes às poses e pontos que, dada a equação

$$\dot{\Gamma} = f(\hat{P}) - \epsilon, \quad (3.1)$$

minimiza o erro $\|\epsilon\|$. O problema é simples no seu objetivo, mas engloba um conjunto considerável de conhecimentos necessários à implementação da solução. É preciso definir como os parâmetros são modelados (especialmente a pose do robô e matriz de calibração da câmera), fazer a estimação do movimento (seja ela por técnicas de visão computacional ou heurísticas sobre a forma de movimento do robô), fazer a associação de dados e medições de sensores diferentes (tais como câmeras, sensores ultrassônicos e *lasers*) e, por fim, resolver grandes problemas de otimização, aplicando heurísticas e explorando características especiais das matrizes Jacobiana, Hessiana, e da pose do robô (a matriz normalmente é esparsa e a otimização é feita calculando os incrementos para a pose em uma variedade linear).

3.1.1 Modelagem da Pose do Robô

O robô pode ser modelado como um corpo rígido no espaço tridimensional. Assim, um mínimo de três coordenadas são necessárias para a localização. Todavia, apenas elas não bastam pois, além da localização no espaço, também é necessário modelar a orientação do corpo rígido. Para representar este estado do corpo rígido no espaço tridimensional, comumente chamado de pose, pode-se utilizar várias formas de parametrização, entre elas ângulos de Euler, parametrização por quatérnions ou um grupo Euclidiano especial SE(3).

Quando a pose é parametrizada no SE(3), todas as poses possíveis do corpo rígido podem ser descritas pelo espaço dos movimentos ou transformações euclidianas especiais (MA et al., 2004) definidas por:

$$SE(3) \doteq \left\{ \mathbf{g} = \begin{bmatrix} \mathbf{\Omega} & \boldsymbol{\tau} \\ \mathbf{0} & 1 \end{bmatrix} \mid \mathbf{\Omega} \in SO(3), \boldsymbol{\tau} \in \mathbb{R}^3 \right\} \subset \mathbb{R}^{4 \times 4} \quad (3.2)$$

Neste caso $\mathbf{g}$, que é a matriz da pose do corpo rígido no espaço, está usando a representação homogênea. Com um abuso de notação, pode-se também usar $\mathbf{g} = \begin{bmatrix} \mathbf{\Omega} & \boldsymbol{\tau} \end{bmatrix}$ para implementações em que se busca economizar espaço de memória. Nesta representação $\boldsymbol{\tau}$ é o vetor coluna de três elementos $[\tau_x \ \tau_y \ \tau_z]^T$ correspondente à posição do corpo no espaço e $\mathbf{\Omega} \in SO(3)$ é a matriz que modela a orientação da pose. Utilizando um conjunto de três eixos para representar a orientação do corpo rígido, tal como apresentado na figura 3.1, sua conversão para o $SO(3)$ é direta desde que os eixos sejam unitários e ortogonais entre si. Satisfeitas essas condições, os eixos x , y , z do corpo rígido são armazenados nas colunas 1, 2 e 3 da matriz $\mathbf{\Omega}$.

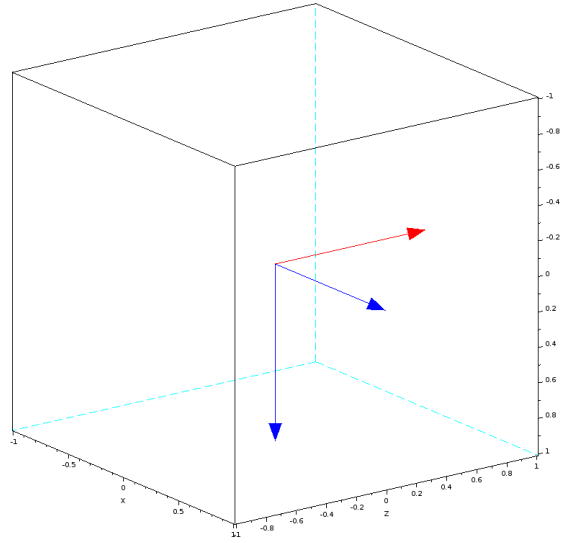


Figura 3.1: Representação da orientação de um corpo rígido usando três eixos unitários e ortogonais entre si. O eixo vermelho indica a direção para onde o objeto está virado e os dois eixos azuis o quanto o objeto está “girando” em torno do eixo vermelho. Nesta imagem o objeto está posicionado na origem.

A modelagem da pose do corpo rígido utilizando a matriz $\mathbf{g} \in SE(3)$ apresenta algumas propriedades importantes. Uma é a capacidade de realizar uma sequência de operações lineares, sejam de rotações, translações ou qualquer outra (desde que pertença ao $SE(3)$), combinando todas em uma única matriz de transformação e realizando o produto desta matriz com a matriz de parametrização $\mathbf{g}$. Neste caso, devido às propriedades do grupo Euclidiano especial, a matriz de transformação resultante pode ser calculada como descrito pela equação 3.3

$$\mathbf{g}' = \mathbf{g}_1 \mathbf{g}_2 \dots \mathbf{g}_N = \begin{bmatrix} \prod_{i=1}^N \Omega_i & \sum_{i=1}^N \left[\prod_{j=1}^{i-1} \Omega_j \right] \tau_i \\ \mathbf{0} & 1 \end{bmatrix} \in SE(3). \quad (3.3)$$

Outra propriedade é a capacidade de calcular a matriz inversa da representação, apenas realizando operações básicas nos elementos da matriz $\mathbf{g}$ sem a sobrecarga do cálculo convencional da matriz inversa, tal como descrito na equação 3.4

$$\mathbf{g}^{-1} = \begin{bmatrix} \Omega^T & -\Omega^T \tau \\ \mathbf{0} & 1 \end{bmatrix} \in SE(3). \quad (3.4)$$

Estas propriedades são úteis em várias situações, como por exemplo, quando fazendo a mudança de base de pontos do espaço tridimensional, tal como das coordenadas do mundo para as coordenadas do corpo rígido. Estas e outras propriedades podem ser exploradas para o ganho de desempenho em uma implementação computacional. A representação da pose no $SE(3)$ é aplicado, principalmente, nos algoritmos de otimização devido a matriz também ser uma variedade linear, contudo seu uso para o registro das poses não é muito adequado devido a necessidade de armazenar 12 elementos para cada matriz. Uma descrição muito

mais aprofundada deste tipo de representação da pose e suas propriedades é feita por Ma et al. (2004).

Outra forma de representar a pose da câmera ou do robô é utilizando a parametrização por ângulos de Euler. Esta abordagem combina as três coordenadas espaciais com um conjunto dos três ângulos: cabeceio (*yaw*) ψ , empinamento (*pitch*) χ e balanceio (*roll*) ϕ ; formando um vetor de seis elementos $\mathbf{p}$, tal como apresentado na equação 3.5

$$\mathbf{p}_E \doteq \begin{bmatrix} x \\ y \\ z \\ \psi \\ \chi \\ \phi \end{bmatrix}. \quad (3.5)$$

Neste trabalho é utilizada a convenção de Tait-Bryan para o significado dos ângulos. Sendo assim, eles são os mesmos empregados na navegação aérea. Diebel (2006) chama a atenção para a existência de 12 tipos diferentes de parametrizações baseadas em ângulos de Euler (FERNÁNDEZ-MADRIGAL; CLARACO, 2012), onde o que as diferencia é a ordem em que os ângulos de $\mathbf{p}_E$ são aplicados. Atendo-se apenas ao esquema de parametrização utilizado neste trabalho é possível, a partir dos ângulos, calcular a matriz de pose no $SE(3)$. Para isto, o vetor $\boldsymbol{\tau}$ pode ser obtido diretamente dos componentes x , y e z e a matriz $\boldsymbol{\Omega}$ é calculada utilizando a equação 3.6 (FERNÁNDEZ-MADRIGAL; CLARACO, 2012)

$$\boldsymbol{\Omega} = \begin{bmatrix} \cos(\phi)\cos(\chi) & \cos(\phi)\sin(\chi)\sin(\psi) - \sin(\phi)\cos(\psi) & \cos(\phi)\sin(\chi)\cos(\psi) + \sin(\phi)\sin(\psi) \\ \sin(\phi)\cos(\chi) & \sin(\phi)\sin(\chi)\sin(\psi) + \cos(\phi)\cos(\psi) & \sin(\phi)\sin(\chi)\cos(\psi) - \cos(\phi)\sin(\psi) \\ -\sin(\chi) & \cos(\chi)\sin(\psi) & \cos(\chi)\cos(\psi) \end{bmatrix}. \quad (3.6)$$

Por fim, a parametrização $\mathbf{p}_E$ pode ser obtida a partir da matriz da pose no $SE(3)$ utilizando os mesmos valores de x , y e z daquelas do vetor $\boldsymbol{\tau}$ e os ângulos a partir do conjunto de equações 3.7 (FERNÁNDEZ-MADRIGAL; CLARACO, 2012)

$$\begin{aligned} \chi &= \text{atan2}(-\Omega_{3,1}, \sqrt{\Omega_{1,1}^2 + \Omega_{2,1}^2}) \\ \text{se } \chi = -\frac{\pi}{2} &\rightarrow \begin{cases} \phi = \text{atan2}(-\Omega_{2,3}, -\Omega_{1,3}) \\ \psi = 0 \end{cases} \\ \text{se } \chi = \frac{\pi}{2} &\rightarrow \begin{cases} \phi = \text{atan2}(\Omega_{2,3}, \Omega_{1,3}) \\ \psi = 0 \end{cases} \\ \text{senão} &\rightarrow \begin{cases} \phi = \text{atan2}(\Omega_{2,1}, \Omega_{1,1}) \\ \psi = \text{atan2}(\Omega_{3,2}, \Omega_{3,3}) \end{cases} \end{aligned} \quad (3.7)$$

A terceira forma de parametrização, a que utiliza quatérnions, combina três coordenadas espaciais com um quatérnion unitário, formando um vetor de sete elementos $\mathbf{p}_q$, tal como apresentado na equação 3.8

$$\mathbf{p}_q \doteq \begin{bmatrix} x \\ y \\ z \\ q_r \\ q_x \\ q_y \\ q_z \end{bmatrix}, \text{ onde } q_r^2 + q_x^2 + q_y^2 + q_z^2 = 1. \quad (3.8)$$

As três coordenadas espaciais correspondem ao mesmo vetor $\boldsymbol{\tau}$ da pose $\mathbf{g} \in SE(3)$. Já a parte real q_r e os três componentes imaginários q_x , q_y e q_z do quatérnion podem ser calculados usando o conjunto de equações 3.9 (FERNÁNDEZ-MADRIGAL; CLARACO, 2012)

$$\begin{aligned} q_r &= \cos\left(\frac{\psi}{2}\right)\cos\left(\frac{\chi}{2}\right)\cos\left(\frac{\phi}{2}\right) + \sin\left(\frac{\psi}{2}\right)\sin\left(\frac{\chi}{2}\right)\sin\left(\frac{\phi}{2}\right) \\ q_x &= \sin\left(\frac{\psi}{2}\right)\cos\left(\frac{\chi}{2}\right)\cos\left(\frac{\phi}{2}\right) - \cos\left(\frac{\psi}{2}\right)\sin\left(\frac{\chi}{2}\right)\sin\left(\frac{\phi}{2}\right) \\ q_y &= \cos\left(\frac{\psi}{2}\right)\sin\left(\frac{\chi}{2}\right)\cos\left(\frac{\phi}{2}\right) + \sin\left(\frac{\psi}{2}\right)\cos\left(\frac{\chi}{2}\right)\sin\left(\frac{\phi}{2}\right) \\ q_z &= \cos\left(\frac{\psi}{2}\right)\cos\left(\frac{\chi}{2}\right)\sin\left(\frac{\phi}{2}\right) - \sin\left(\frac{\psi}{2}\right)\sin\left(\frac{\chi}{2}\right)\cos\left(\frac{\phi}{2}\right) \end{aligned} \quad (3.9)$$

onde os ângulos ψ , χ e ϕ são os mesmos da parametrização utilizando os ângulos de Euler e podem ser obtidos a partir do conjunto de equações 3.7. Por fim, é também possível calcular a matriz $\mathbf{g}$ da pose no $SE(3)$ a partir da parametrização usando quatérnions. O vetor $\boldsymbol{\tau}$ novamente é o mesmo dado pelas coordenadas x , y e z e a matriz $\boldsymbol{\Omega}$ é dada pela equação 3.10

$$\boldsymbol{\Omega} = \begin{bmatrix} q_r^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y - q_r q_z) & 2(q_x q_z + q_r q_y) \\ 2(q_x q_y + q_r q_z) & q_r^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_y q_z - q_r q_x) \\ 2(q_x q_z - q_r q_y) & 2(q_y q_z + q_r q_x) & q_r^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix}. \quad (3.10)$$

Comparando a duas últimas representações para a pose, a baseada em quatérnions apresenta vantagens significativas sobre a que usa ângulos de Euler. A primeira é que nela não existe o problema conhecido como *gimbal lock* quando χ se aproxima de $\pm\frac{\pi}{2}$ levando a uma matriz de rotação degenerada. A segunda vantagem é que, também é possível representar transformações lineares por meio da parametrização usando quatérnions e combinar sequências delas em uma única transformação, tal como quando usando a matriz de pose $\mathbf{g} \in SE(3)$. Assim, existe uma tabela de equivalência entre operações de elementos usando representações como poses no $SE(3)$ e usando quatérnions. Por fim, a representação para a pose utilizando quatérnions necessita armazenar apenas 7 valores contra os 12 da representação no $SE(3)$, tornando-os mais interessantes para este tipo de aplicação. Uma descrição

muito mais aprofundada das operações matemáticas envolvidas nessa equivalência foi feita por Fernández-Madrigal e Claraco (2012) e Blanco (2014).

Ao se utilizar as representações das poses por quatérnions ou grupos euclidianos especiais no $SE(3)$, existe um conjunto de operações, como composição de elementos ou mudanças de bases, que podem ser realizadas nas duas formas de representação. Assim, na literatura é comum utilizar os dois operadores $\oplus$ e $\ominus$ para descrever as transformações de corpo rígido sobre as representações da pose (SMITH; CHEESEMANN, 1986; THRUN; BURGARD; FOX, 2005), com o objetivo de simplificar a notação, sendo que a definição dos operadores depende da representação utilizada, seja $\mathbf{g}$ ou $\mathbf{p}_q$. Desta forma, neste texto também é utilizado a nomenclatura $\mathbf{p}$ para representações $\mathbf{g} \in SE(3)$, por não invalidar as equações.

3.1.2 Modelagem da Câmera

O uso de câmeras abre um grande número de possibilidades de aplicações devido a quantidade de informação que pode ser obtida por meio de imagens. Quando aplicada em um sistema de SLAM Visual, duas características são necessárias para a integração no sistema. A primeira é a pose $\mathbf{p}_c$ em relação a pose do robô e a segunda é a função que mapeia os raios luminosos provenientes do ambiente na superfície da imagem formada na câmera.

A pose da câmera é importante porque ela não está necessariamente posicionada e alinhada com a pose do robô, sendo comum diferenças devido à montagem. Contudo, pode-se calcular a pose da câmera no mundo e a posição de um objeto no sistema de coordenadas da câmera desde que sejam conhecidas tanto a pose do robô como a pose da câmera em relação a pose do robô. De posse da pose $\mathbf{p}_c$ da câmera e da pose $\mathbf{p}_r$ do robô, a posição final da câmera no espaço tridimensional pode ser calculada por meio da composição das duas poses, ou duas parametrizações. Assim, a pose final da câmera no mundo, ou parâmetros extrínsecos, é dada pela equação 3.11

$$\mathbf{p}'_c = \mathbf{p}_r \oplus \mathbf{p}_c. \quad (3.11)$$

Já para um ponto $\mathbf{X}$ com coordenadas tridimensionais em relação ao mundo, sua posição no sistema de coordenadas da câmera é dada pela equação 3.12

$$\mathbf{X}' = \mathbf{X} \ominus \mathbf{p}'_c. \quad (3.12)$$

Quando o parâmetro $\mathbf{p}'_c$ da equação 3.12 contiver uma pose representada por uma matriz pertencente ao $SE(3)$, e estiver sendo utilizada em operações que não necessitam da representação homogênea, tal como a equação 3.15, deve-se descartar a quarta linha.

Quanto à função que rege a regra de projeção dos pontos na imagem, esta é dependente do tipo e de alguns parâmetros específicos da câmera. O tipo define a função, sendo bastante comum aquele da câmera *pinhole*. Esta é modelada por parâmetros intrínsecos e extrínsecos. Os primeiros são organizados em uma matriz Λ , chamada de “matriz de parâmetros intrínsecos”, apresentada na equação 3.13. A matriz de parâmetros intrínsecos é obtida durante o processo de calibração da câmera. Seus elementos relacionam as posições do ponto de referência no espaço com os valores em x e y da projeção na imagem. α_x e α_y são os valores de escala que relacionam a posição x e y na imagem com as posições no espaço, enquanto δ_x e δ_y são a posição dos *pixels* centrais da imagem. Finalmente, o parâmetro γ adiciona uma dependência da projeção em x com a altura do ponto no espaço (PRINCE, 2012).

$$\Lambda \doteq \begin{bmatrix} \alpha_x & \gamma & \delta_x \\ 0 & \alpha_y & \delta_y \\ 0 & 0 & 1 \end{bmatrix} \quad (3.13)$$

Quanto aos parâmetros extrínsecos, estes modelam a pose $\mathbf{p}'_c$ da câmera no espaço, podendo ser usada qualquer uma das representações definidas anteriormente.

Definindo que a função da projeção F retorne um vetor homogêneo de três coordenadas, $\mathbf{x} \doteq [x \ y \ 1]^T$, contendo a localização do ponto na imagem, para a câmera *pinhole*, a operação de projeção é descrita pela equação 3.14

$$\lambda F \doteq \Lambda \mathbf{X}' \quad (3.14)$$

que ao ser expandida se torna a equação 3.15

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \frac{1}{\lambda} \begin{bmatrix} \alpha_x & \gamma & \delta_x \\ 0 & \alpha_y & \delta_y \\ 0 & 0 & 1 \end{bmatrix} [\mathbf{X} \ominus \mathbf{p}'_c], \quad (3.15)$$

onde o fator de escala λ é igual ao elemento da terceira linha do vetor coluna dado por $\mathbf{X} \ominus \mathbf{p}'_c$. A equação 3.15 é importante em um sistema de SLAM Visual porque é com base nela que é definida a função do erro de reprojeção dos valores de x e y . Dependendo do problema, pode ser realizado o ajuste de todos parâmetros da função ou apenas dos parâmetros extrínsecos, no caso da matriz Λ já ter sido obtida por meio do processo de calibração. Uma descrição bem mais aprofundada do funcionamento da câmera *pinhole* pode ser encontrada no trabalho de Prince (2012).

3.1.3 Modelagem do Mapa

O mapa é formado pelo conjunto de poses $\dot{p}$ que a câmera assume ao longo do tempo junto com o conjunto $\dot{X}$ de pontos de referência existentes no espaço. Esses dois tipos de objetos são modelados como vértices de um grafo enquanto as arestas correspondem às medições feitas dos pontos visíveis em cada pose. Em um sistema de SLAM Visual que emprega apenas sensores exteroceptivos², as arestas partem unicamente das poses para os pontos de referência, já para sistemas que também empregam sensores proprioceptivos³ também podem existir arestas de poses para poses, correspondendo à medição do deslocamento por sensores, como por exemplo o de odometria. Uma ilustração de um pequeno mapa exemplo para um sistema de SLAM Visual que usa apenas sensores exteroceptivos é apresentado na figura 3.2.

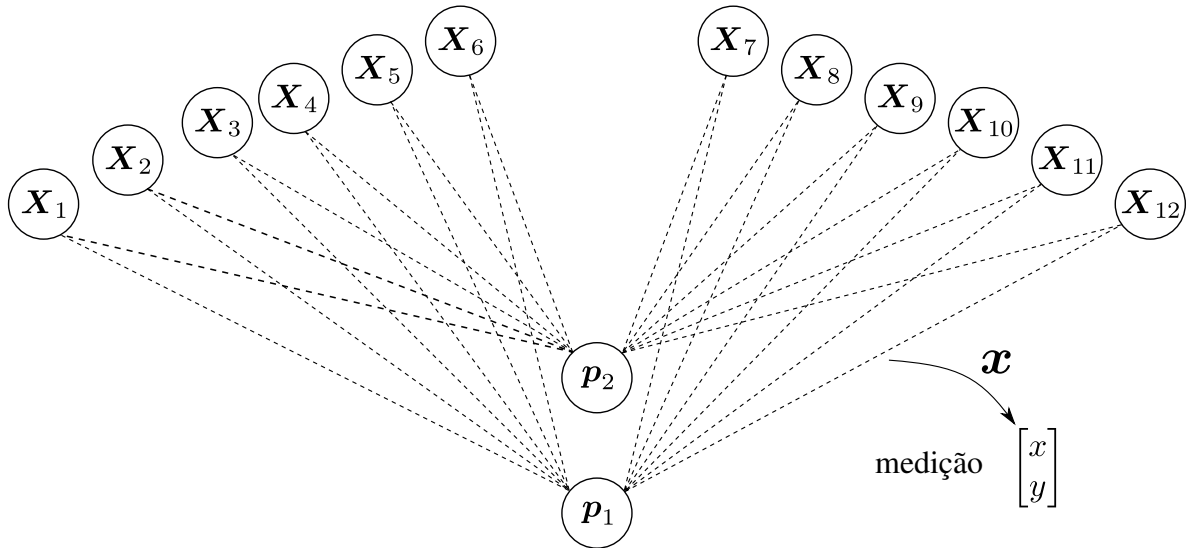


Figura 3.2: Exemplo de grafo para um SLAM Visual.

3.2 Componentes do Sistema de SLAM Visual

Seja $\dot{p}$ o conjunto das poses que a câmera assume, $\dot{x}$ o conjunto de todas as projeções de pontos de referência, $\dot{X}$ o conjunto de todos os pontos de referência no espaço tridimensional e Λ os parâmetros intrínsecos da câmera, um sistema de SLAM Visual baseado em reconstrução geométrica pode ser descrito como sendo capaz de encontrar os parâmetros $\dot{X}$ e $\dot{p}$ que maximizam a probabilidade do conjunto de projeções $\dot{x}$, de acordo com a equação 3.16 (PRINCE, 2012)

$$\dot{X}, \dot{p} \doteq \arg \max_{\dot{X}, \dot{p}} \left[\sum_{i=1}^I \sum_{j=1}^J \log [Pr(x_{ij} | X_i, \Lambda, p_j)] \right], \quad (3.16)$$

²Sensores que capturam informações do ambiente em que o robô se encontra.

³Sensores que medem valores internos ao sistema do robô (ROLAND; REZA; DAVIDE, 2011).

onde $\dot{\hat{X}}$ e $\dot{\hat{p}}$ são os valores dos argumentos $\hat{X}$ e $\hat{p}$ que maximizam a probabilidade das medições $\dot{x}$.

A equação 3.16 não possui uma solução algébrica fechada, então a abordagem normalmente adotada é a de realizar um processo iterativo de minimização do erro de reprojeção dos pontos de referência. Contudo, para que o sistema opere de forma on-line várias heurísticas são aplicadas como também algumas suposições são feitas, como por exemplo a otimização utilizando uma janela deslizante sobre apenas um conjunto de poses. Além disso, desde a criação do PTAM, consolidou-se nas implementações a abordagem em que o sistema é dividido em duas linhas de execução trabalhando em paralelo. Uma linha é responsável por fazer a captura das imagens, extração de pontos de interesse e cálculo de descritores, associação de pontos e estimação inicial das poses e pontos de referência. A outra linha realiza o ajuste de feixe com o objetivo de encontrar $\dot{\hat{X}}$ e $\dot{\hat{p}}$ resolvendo iterativamente a equação 3.16. É comum os *software* de SLAM capazes de encontrar uma solução aproximada para a equação 3.16 possuírem na sua implementação os módulos responsáveis pelas tarefas a seguir:

1. Identificação de pontos de interesse;
2. Cálculo dos descritores;
3. Associação dos pontos de interesse;
4. Estimação de movimento;
5. Triangulação dos pontos de referência;
6. Adição de *keyframes*⁴;
7. Ajuste de feixe local;
8. Detecção de fechamento de laço;
9. Ajuste de feixe global;
10. Detecção de “robô sequestrado”.

3.2.1 Identificação de Pontos de Interesse

O módulo responsável pela identificação de pontos de interesse executa a primeira etapa de processamento da imagem. Sua função é selecionar pontos que contém informação visual que pode ser reproduzida e encontrada em imagens diferentes (PRINCE, 2012).

⁴Quadro chave que define uma pose e medições utilizadas no processo de otimização.

Existem várias técnicas para a identificação desses pontos, sendo que a literatura disponibiliza um grande número de algoritmos com essa finalidade, como por exemplo: BRISK (LEUTENEGGER; CHLI; SIEGWART, 2011), SURF (BAY et al., 2008), SIFT (LOWE, 2004), FAST (ROSTEN; PORTER; DRUMMOND, 2010), A-KAZE (ALCANTARILLA JESUS NUEVO, 2013) e ORB (RUBLEE et al., 2011). A identificação de pontos de interesse implica no processamento da imagem, algo que pode ser demorado quando comparado ao tempo entre imagens para câmeras que operam a trinta, ou mais, quadros por segundo. Além disso, também é desejável que o algoritmo usado na identificação dos pontos de interesse seja robusto à rotações e variações de escala na imagem. Autores como Mur-Artal, Montiel e Tardós (2015) usam estes requisitos para excluïrem identificadores como o BRIEF (CALONDER et al., 2010) e LDB (YANG; CHENG, 2012) de algoritmos de SLAM capazes de operar on-line.

A importância dos pontos de interesse está no seu uso para a estimação do movimento e também reconhecimento de locais visitados previamente. Para a estimação do movimento é utilizada a posição da projeção $\mathbf{x} = [x \ y \ 1]^T$ dos pontos nas imagens, combinada com os parâmetros da câmera. Estes são os dados de entrada de algoritmos como o dos oito pontos ou cinco pontos. Quanto ao reconhecimento de locais visitados previamente, espera-se que os pontos de interesse se repitam entre passagens sucessivas capturadas no mesmo local.

3.2.2 Cálculo dos Descritores

O passo seguinte após a identificação dos pontos de interesse é calcular os descritores destes pontos. Os descritores podem ser definidos, de uma forma intuitiva, como sendo a assinatura de regiões de interesse, ou mesmo de toda imagem (dependendo do descritor utilizado). Sua importância advém da necessidade de comparar ou armazenar informação das regiões dos pontos de interesse, pois trabalhar diretamente com partes das imagens seria algo muito custoso. Enquanto o problema de encontrar os pontos de interesse está associado com o processamento de toda a imagem, o cálculo dos descritores para um ponto de interesse está associado apenas à região ao redor do ponto.

Descritores podem ser implementados como vetores binários ou histogramas normalizados, tal como o SIFT (LOWE, 2004) e o HOG (DALAL; TRIGGS, 2005) (PRINCE, 2012). Em sistemas de SLAM Visual, possuem duas funções principais. A primeira é servir de elemento de comparação para pontos de interesse entre imagens diferentes (CAO; CHEN; FAN, 2016). É comparando os descritores que o sistema decide se dois pontos de interesse em imagens diferentes correspondem ao mesmo ponto de referência. A segunda função consiste em utilizar os descritores em algoritmos de reconhecimento de imagens (QU et al., 2014). Neste caso, eles funcionam como palavras visuais em algoritmos baseados em

bag of words que são utilizados, principalmente, no módulo responsável pela detecção de fechamento de laço.

3.2.3 Associação de Pontos de Interesse

A etapa da associação de pontos tem por objetivo identificar pontos de interesse provenientes do mesmo ponto de referência e que se repetem entre imagens. Na sua implementação mais simples, consiste em uma busca por força bruta, onde cada descritor do ponto de interesse em uma imagem é comparado com os de outra imagem até encontrar aquele que apresenta as maiores semelhanças segundo a métrica empregada na comparação. Contudo, esta abordagem é altamente ineficiente e nos sistemas de SLAM Visual algumas heurísticas são empregadas para diminuir o tempo necessário para a conclusão da operação. Quando um modelo de movimento para a câmera não está disponível, pode-se utilizar heurísticas como restringir a busca de pontos de uma imagem em regiões específicas da outra imagem. Esta heurística baseia-se na suposição de que a câmera tem um movimento suave e que por isso os pontos deslocam-se continuamente nas imagens, sem grandes deslocamentos. Uma outra heurística muito empregada é estimar a localização mais provável de um ponto de interesse a partir do modelo de movimento da câmera e de uma possível posição do ponto de referência no espaço tridimensional. Esta outra heurística baseia-se na suposição de que, quando se tem uma estimativa da posição do ponto de referência, um modelo de movimento para a câmera e a matriz de calibração da câmera, pode-se calcular a projeção mais provável do ponto na nova imagem e utilizar isto para restringir a área de busca. A nova pose da câmera pode ser encontrada pela equação 3.17, sendo $\mathbf{p}_m$ o modelo de movimento da câmera. A figura 3.3 exemplifica a associação entre pontos de interesse para imagens consecutivas.

$$\mathbf{p}_f = \mathbf{p}'_c \oplus \mathbf{p}_m \quad (3.17)$$

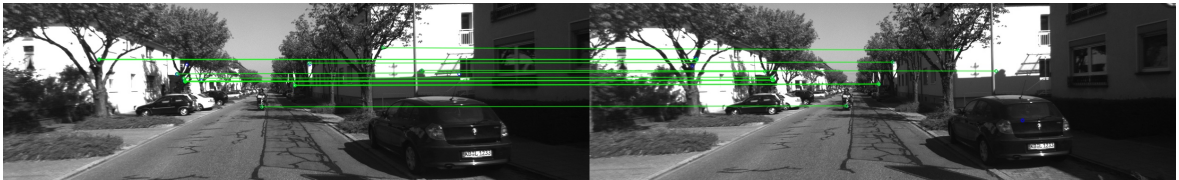


Figura 3.3: Associação de pontos de interesse entre imagens consecutivas.

3.2.4 Estimação de Movimento

A tarefa de estimação de movimento consiste em encontrar uma estimativa da transformação linear que leva a câmera da pose $\mathbf{p}_1$ para a pose $\mathbf{p}_2$. Quando a câmera move-se no espaço a uma velocidade aproximadamente constante, durante o intervalo de tempo entre a captura de imagens sucessivas, pode-se utilizar o modelo de movimento anterior, caso se

disponha de um, como sendo o modelo atual. Isto é possível porque existe uma etapa subsequente de *bundle adjustment* onde o modelo de movimento da câmera pode ser corrigido. Quando não se tem nenhuma estimativa do movimento, mas um número suficiente de pontos de interesse associados, pode-se utilizar o algoritmo dos oito pontos (HARTLEY, 1997), o algoritmo dos cinco pontos (NISTER, 2004) ou a homografia entre imagens sucessivas para calcular uma estimativa inicial do movimento da câmera. Mesmo se esta segunda técnica for aplicada, o modelo de movimento obtido para a câmera também passa pela etapa subsequente de ajuste de feixe.

3.2.5 Triangulação dos Pontos de Referência

A triangulação dos novos pontos de referência corresponde a etapa do cálculo da posição inicial do ponto no espaço tridimensional. Estes valores, junto com estimativas iniciais do movimento da câmera, são utilizados na etapa subsequente de ajuste de feixe (*bundle adjustment*). A posição dos pontos de referência X no espaço podem ser obtidas aplicando a equação 3.18

$$\begin{bmatrix} \Omega_{3,1_1}x'_1 - \Omega_{1,1_1} & \Omega_{3,2_1}x'_1 - \Omega_{1,2_1} & \Omega_{3,3_1}x'_1 - \Omega_{1,3_1} \\ \Omega_{3,1_1}y'_1 - \Omega_{2,1_1} & \Omega_{3,2_1}y'_1 - \Omega_{2,2_1} & \Omega_{3,3_1}y'_1 - \Omega_{2,3_1} \\ \vdots & \vdots & \vdots \\ \Omega_{3,1_J}x'_J - \Omega_{1,1_J} & \Omega_{3,2_J}x'_J - \Omega_{1,2_J} & \Omega_{3,3_J}x'_J - \Omega_{1,3_J} \\ \Omega_{3,1_J}y'_J - \Omega_{2,1_J} & \Omega_{3,2_J}y'_J - \Omega_{2,2_J} & \Omega_{3,3_J}y'_J - \Omega_{2,3_J} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} \tau_{x_1} - \tau_{z_1}x'_1 \\ \tau_{y_1} - \tau_{z_1}y'_1 \\ \vdots \\ \tau_{x_J} - \tau_{z_J}x'_J \\ \tau_{y_J} - \tau_{z_J}y'_J \end{bmatrix}. \quad (3.18)$$

Sendo Ω_{ij} e τ_i os elementos da matriz Ω e do vetor τ quando as transformações lineares estão representadas no $SE(3)$. É importante notar também que, as transformações lineares correspondentes a cada pose são obtidas a partir da equação 3.19 e que os valores encontrados para $X \doteq [u \ v \ w]^T$ estão no sistema de coordenadas de p_1 . Já os elementos $[x'_i \ y'_i]^T$ podem ser calculados a partir da projeção original e dos parâmetros da câmera calibrada, de acordo com a equação 3.20.

$$p'_i = \ominus(p_i \ominus p_1) \quad (3.19)$$

$$\begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} = \Lambda^{-1} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \quad (3.20)$$

3.2.6 Adição de *Keyframes*

Nas primeiras soluções para o problema do SLAM Visual, as técnicas baseadas em filtros eram as mais usadas (DAVISON et al., 2007)(CIVERA; DAVISON; MONTIEL, 2008)(CHIUSO et al., 2002)(EADE; DRUMMOND, 2006). Nelas, todas as imagens captu-

radas são utilizadas na estimação da pose. O problema desta abordagem é que, dependendo da forma como a câmera se desloca, as diferenças entre imagens subsequentes podem ser muito pequenas ou mesmo desprezíveis. Isso ocorre em situações onde a câmera está parada, por exemplo, ou quando a taxa de captura de quadros é muito alta, capturando assim imagens virtualmente idênticas. Esta adição indiscriminada de imagens (capturadas praticamente na mesma pose) ao sistema faz com que recursos computacionais sejam desperdiçados e erros de linearização sejam adicionados ao mapa (MUR-ARTAL; MONTIEL; TARDÓS, 2015). Uma solução para este problema é fazer uma seleção de imagens com diferenças suficientes para que a suposição de que a câmera realizou deslocamento considerável entre imagens seja verdadeira, este é o princípio de funcionamento da abordagem baseada em *keyframes* (MOURAGNON et al., 2006)(KLEIN; MURRAY, 2007). Mur-Artal, Montiel e Tardós (2015) afirmam que o uso de *keyframes* torna a otimização mais custosa, porém mais precisa. Outros autores, Strasdat, Montiel e Davison (2012), demonstraram que as abordagens que utilizam *keyframes* são mais precisas que aquelas baseadas em filtragens, para um mesmo custo computacional.

Ao adicionar um *keyframe* ao mapa, este pode ser modelado como uma pose da câmera e um conjunto de medições dos pontos de referência visualizados a partir desta pose. A relação entre *keyframes* subsequentes, como também entre os pontos de referência visualizados, pode ser feita pelas arestas de um grafo.

3.2.7 Ajuste de Feixe Local

Após a estimação inicial do movimento da câmera e a triangulação da posição no espaço dos pontos de referência, é necessário realizar uma otimização destes valores pois a triangulação realizada pela equação 3.18 minimiza apenas o erro algébrico. Esta tarefa é comumente feita por meio da técnica de otimização por Gradiente Descendente Não-linear, também chamada de ajuste de feixe ou *bundle adjustment*. O procedimento consiste em realizar a minimização iterativa de uma função de custo que fornece uma métrica do erro do sistema. Formalmente, o procedimento é descrito pela equação 3.21 para um sistema que não possui arestas (medições) de distâncias entre poses

$$F(\Theta) = \frac{1}{2} \sum_{\{i,j\} \in \dot{\Gamma}} r_{i,j}^T(\Theta) \Sigma_{i,j}^{-1} r_{i,j}(\Theta), \quad (3.21)$$

onde r é o vetor de resíduos para o conjunto de parâmetros Θ do sistema, Σ é a matriz de covariância e $\dot{\Gamma}$ é o conjunto de todos os valores medidos. A função $F(\Theta)$ é não linear,

sendo utilizada a sua expansão de segunda ordem por série de Taylor como ponto de partida para a dedução do incremento para o estado do sistema na iteração k .

$$F(\Theta_k + \Delta\Theta) \approx \hat{F}_k(\Theta + \Delta\Theta) = F(\Theta_k) + \mathbf{h}_k^T \Delta\Theta + \frac{1}{2} \Delta\Theta^T \mathbf{H}_k \Delta\Theta \quad (3.22)$$

Onde:

$$\mathbf{h}_k = \nabla_{\Theta} F(\Theta) \Big|_{\Theta=\Theta_{k-1}} \quad (3.23)$$

é o vetor gradiente e

$$\mathbf{H}_k = \nabla_{\Theta}^2 F(\Theta) \Big|_{\Theta=\Theta_{k-1}}^T \quad (3.24)$$

é a matriz Hessiana. Tomando a derivada parcial da equação 3.22 em relação a $\Delta\Theta$ tem-se:

$$\frac{\partial F(\Theta_k + \Delta\Theta)}{\partial \Delta\Theta} \approx \frac{\partial \hat{F}_k(\Theta + \Delta\Theta)}{\partial \Delta\Theta} = \mathbf{h} + \mathbf{H}_k \Delta\Theta \quad (3.25)$$

Fazendo $\mathbf{h} + \mathbf{H}_k \Delta\Theta^* = 0$ o incremento $\Delta\Theta^*$ é calculado resolvendo:

$$\mathbf{H}_k \Delta\Theta_k^* = -\mathbf{h}_k, \quad (3.26)$$

que é o incremento utilizado no algoritmo iterativo de Gauss-Newton. Pode-se usar também:

$$(\mathbf{H}_k + \nu_k \mathbf{I}) \Delta\Theta_k^* = -\mathbf{h}_k, \quad (3.27)$$

para o cálculo do incremento no algoritmo de Levenberg-Marquardt. O cálculo de $\Delta\Theta_k^*$ na sua forma mais simples envolve encontrar uma matriz inversa, sendo que esta pode ter dimensões consideráveis. Contudo, a matriz a ser invertida é esparsa e sua estrutura pode ser explorada evitando o cálculo da matriz inversa. Formas eficientes de implementar o algoritmo de Levenberg-Marquardt explorando as características da matriz de Hessianos podem ser encontradas no trabalho de Fernández-Madrigal e Claraco (2012) e Hartley e Zisserman (2004). Além de explorar as propriedades de matriz esparsa, deve-se salientar que a otimização é feita com o incremento para a pose sendo calculado em uma variedade linear com o objetivo de aumentar a robustez e desempenho do sistema, sendo essa técnica bastante utilizada em problemas de reconstrução geométrica ou estimação de movimento (MA; KOSECKÁ; SASTRY, 1999)(LEE; MOORE, 2004)(HELMKE et al., 2007).

Nos sistemas de SLAM Visual, o cálculo do vetor de resíduo r , empregado na equação 3.21 pode ser deduzido a partir da equação 3.15. Utilizando a representação para a pose na forma matricial, considerando a câmera calibrada e definindo a pose inicial como

origem do sistema de coordenadas, para um ponto e duas poses pode-se escrever o conjunto de equações a seguir:

$$\lambda_1 \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix}, \quad (3.28)$$

$$\lambda_2 \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} \omega_{1,1} & \omega_{1,2} & \omega_{1,3} & \tau_x \\ \omega_{2,1} & \omega_{2,2} & \omega_{2,3} & \tau_y \\ \omega_{3,1} & \omega_{3,2} & \omega_{3,3} & \tau_z \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix}. \quad (3.29)$$

Resolvendo para x e y em cada pose, obtém-se:

$$x_1 = \frac{u}{w}, \quad (3.30)$$

$$y_1 = \frac{v}{w}, \quad (3.31)$$

$$x_2 = \frac{u_1\omega_{1,1} + v_1\omega_{1,2} + w_1\omega_{1,3} + \tau_x}{u_1\omega_{3,1} + v_1\omega_{3,2} + w_1\omega_{3,3} + \tau_z}, \quad (3.32)$$

$$y_2 = \frac{u_1\omega_{2,1} + v_1\omega_{2,2} + w_1\omega_{2,3} + \tau_y}{u_1\omega_{3,1} + v_1\omega_{3,2} + w_1\omega_{3,3} + \tau_z}. \quad (3.33)$$

O sistema possui quatro equações e quinze incógnitas, restando então onze variáveis indeterminadas. Entretanto, ao adicionar novos pontos, ele cresce quatro equações, para três incógnitas a cada ponto adicionado, diminuindo o número de variáveis indeterminadas. Isto faz com que o sistema torne-se determinado a partir de um número mínimo de pontos desde que ele possua duas ou mais poses.

Mesmo explorando as propriedades do sistema de equações, e a otimização empregando variedades lineares para o incremento das poses, a etapa de otimização pode ser demorada para os requisitos de um sistema que opera em tempo real. Devido a isso, a estratégia normalmente usada na implementação é realizar a otimização empregando a técnica chamada de janela deslizante. Esta consiste de um conjunto de *keyframes*, normalmente contendo os últimos adicionados, e dos pontos de referência visíveis a partir deles. A medida que a câmera se desloca, novos *keyframes* são adicionados e o conjunto utilizado é modificado, removendo o elemento mais antigo, por exemplo, e adicionando o calculado mais recentemente.

3.2.8 Detecção de Fechamento de Laço

A etapa de detecção de fechamento de laço tem como função verificar se *keyframes* diferentes, que foram adicionados depois de um deslocamento considerável, são do mesmo

local. Caso seja verdadeiro, isso indica que a câmera está revisitando uma localização prévia, caracterizando o fechamento de laço. O problema pode ser abordado como sendo o de reconhecimento de local e a implementação da solução normalmente é feita como uma linha de execução que trabalha em paralelo aos demais módulos.

Para a solução do reconhecimento de local, pode-se utilizar técnicas de comparação de imagem a imagem pois, segundo o trabalho de Williams et al. (2009), estas escalam melhor para grandes ambientes (MUR-ARTAL; MONTIEL; TARDÓS, 2015). A comparação pode ser feita usando uma abordagem baseada em *bag of words* (NISTER; STEWENIUS, 2006), onde é calculada uma palavra binária (GALVEZ-LÓPEZ; TARDOS, 2012) para cada ponto de interesse. Cada imagem é então caracterizada como contendo uma coleção de palavras visuais onde uma função de distribuição de probabilidade fornece a probabilidade dela ter sido capturada no mesmo local de uma outra imagem.

3.2.9 Ajuste de Feixe Global

A etapa de ajuste de feixe global, ou *global bundle adjustment* é realizada depois que um fechamento de laço é detectado. A função desta operação é reduzir o erro cumulativo do processo de otimização, sendo realizada sobre todo o conjunto de *keyframes* e pontos de referência que fazem parte da trajetória do laço. A solução para o *global bundle adjustment* é muito semelhante à da etapa de *local bundle adjustment*, diferindo principalmente no número de *keyframes* empregados na otimização, já que este último utiliza somente aqueles contidos na janela deslizante enquanto no ajuste global são processados todos os *keyframes* pertencentes ao laço.

3.2.10 Detecção de “Robô Sequestrado”

Esta tarefa não é necessariamente uma das funções de um *software* de SLAM Visual, mas ela é necessária para adicionar robustez ao sistema. A detecção de “robô sequestrado” ocorre quando o número de correspondências encontradas pela associação de pontos de interesse está abaixo de um limar. A causa pode ser a movimentação rápida do robô para uma outra localização por um agente externo como também pode ser motivado pela incapacidade do sistema de processar as imagens na velocidade necessária para fazer a associação de pontos. Assim, é impossível ao *software* fazer a estimação de movimento e o estado do sistema deve mudar para um reinício do processo de mapeamento (criando um novo mapa) ou para um modo de relocalização no mapa existente.

3.3 Algoritmo de SLAM Visual

A seguir são apresentados os algoritmos fundamentais para a criação do sistema de SLAM Visual por métodos indiretos e reconstrução geométrica. São listados quatro algoritmos em pseudocódigo, sendo que para cada um existe uma listagem mais detalhada nos anexos. Destes, o algoritmo 3.1 (A.1) é responsável por fazer a estimativa do movimento entre duas imagens usando o algoritmo dos oito pontos ou a homografia entre as imagens. O algoritmo 3.2 (A.2) é responsável por realizar a estimativa inicial da trajetória e rastreamento dos pontos de referência. Este, normalmente, possui a sua própria linha de execução e sua função é calcular os valores iniciais das poses e pontos de referência e manter a janela de otimização. A tarefa de melhoramento da estimativa inicial da trajetória é realizada pelos algoritmos 3.3 (A.3) e 3.4 (A.4). O primeiro é responsável pela otimização propriamente dita, sendo que ele necessita do cálculo do gradiente e Hessiana, que são esparsos. O cálculo do gradiente e Hessiana foram separados no algoritmo 3.4 (A.4), com o objetivo de simplificar o entendimento. Estes dois últimos são adaptações diretas daqueles existentes no trabalho de Fernández-Madrigal e Claraco (2012) e são apresentados neste texto com o objetivo de reunir em um único local todos os principais algoritmos necessários ao sistema. Além destas referências, é necessária a compreensão da otimização empregando variedades lineares no cálculo do incremento das poses, pois isto adiciona robustez e aumenta a velocidade de convergência do sistema. Apresentar os fundamentos matemáticos da otimização usando variedades lineares para sistemas de SLAM está além do escopo deste trabalho. Para um estudo sobre o assunto, o leitor deve consultar referências como o livro texto de Fernández-Madrigal e Claraco (2012), o tutorial sobre parametrização de transformações e otimizações empregando variedades lineares de Blanco (2014) e o tutorial de Dellaert (2014) para uma descrição da aplicação para SLAM Visual. Uma introdução sobre variedades lineares diferenciáveis e geometria Riemanniana é feita no livro texto de Boothby (1975) e uma abordagem específica para algoritmos de otimização é feita no livro de Absil, Mahony e Sepulchre (2007).

Algoritmo 3.1: Estima rotação e translação

Entrada: Conjunto de pontos de interesse de duas imagens consecutivas

Entrada: Associação dos pontos entre as duas imagens

Entrada: Parâmetros intrínsecos da câmera

Saída: Modelo do movimento entre poses

- 1 Estima a matriz fundamental;
 - 2 **se** *matriz não é singular* **então**
 - 3 Faz um refinamento da associação entre os pontos;
 - 4 Estima novamente a matriz fundamental;
 - 5 Calcula a matriz essencial;
 - 6 Decompõe a matriz em valores singulares;
 - 7 Estima duas translações mudando o sinal;
 - 8 Estima duas matrizes $\in SO(3)$;
 - 9 Triangula três posições no espaço para um ponto de referência;
 - 10 Selecione como resposta a translação e matriz em que o ponto de referência está na frente das duas poses;
 - 11 **senão**
 - 12 Tenta calcular o modelo de movimento usando a homografia entre as imagens;
-

Algoritmo 3.2: Algoritmo de *front-end* do SLAM Visual Monocular

```

1  enquanto verdadeiro faça
2      leia uma imagem;
3      calcule os pontos de interesse da imagem;
4      calcule os descritores dos pontos de interesse da imagem;
5      crie um quadro com informação da pose inicial unitária, pontos de interesse e
        descritores;
6  se estado = INICIALIZANDO então
7      se quadro anterior não é nulo então
8          realize a associação entre os pontos do quadro anterior com o atual;
9          estima rotação e translação(quadro anterior, quadro atual, associação entre
            pontos, parâmetros da câmera);
10         calcule a nova pose a partir da anterior e da estimativa de rotação e
            translação;
11         triangula a posição no espaço dos pontos de referência baseado nos pontos
            de interesse dos quadros;
12         adicione a pose atual ao quadro atual;
13         adicione o quadro atual ao grafo da trajetória realizada pela câmera;
14         atualize o índice inicial da janela de otimização;
15         atualize o índice final da janela de otimização;
16         atualize o estado para MAPEANDO;
17     senão
18         quadro atual passa a ser o quadro anterior;
19 senão
20     calcule o modelo de movimento usando as duas últimas poses;
21     utilize este modelo como sendo o movimento do quadro atual;
22     realize a associação entre os pontos de interesse do quadro anterior com o
        atual;
23     se numero de associações < limiar para adicionar um keyframe então
24         triangula a posição no espaço de mais pontos de referência usando o
            quadro anterior, o atual e os parâmetros da câmera;
25         adicione estes novos pontos de referência ao mapa;
26         adicione o quadro atual ao grafo da trajetória realizada pela câmera;
27         realize a associação de pontos do quadro inicial da janela de otimização
            para o quadro atual;
28         enquanto (número de pontos associados < limiar para janela) e (inicio
            da janela < fim da janela) faça
29             incremente o inicio da janela;
30             se Inicio da janela < fim da janela então
31                 realiza a associação de pontos do quadro inicial da janela de
                    otimização para o quadro atual;
32         quadro atual passa a ser o último quadro da janela de otimização;

```

Algoritmo 3.3: Esboço do algoritmo Levenberg-Marquardt para matrizes esparsas e otimização com o cálculo do incremento para poses usando variedades lineares

Entrada: poses da janela de otimização

Entrada: arestas ligando poses e pontos de referência

Entrada: pontos de referência na janela de otimização

Saída: poses e pontos de referência otimizados

```
1 constrói um sistema linear esparso com poses, arestas de medições e pontos de
  referência na janela de otimização;
2 enquanto numero de iterações < que o máximo e não concluiu faça
3   calcule o incremento utilizando o algoritmo de Cholesky para sistemas esparsos;
4   se o incremento é menor que o limiar então
5     | otimização concluída;
6   senão
7     | adiciona o incremento aos pontos e poses modelados como uma variedade
      | linear;
8     | se o sistema está convergindo então
9       | incrementa o número de iterações;
10      | verifica se a condição de convergência foi atingida;
11      | monta o novo sistema linear esparso;
12      | calcula um novo fator multiplicador;
13    | senão
14      | reverte o valor das poses e pontos para o estado anterior;
15      | calcula novos fatores multiplicadores;
16      | incrementa o número de iterações;
17      | reverte o sistema linear para o estado anterior;
18   | retorna os novos valores das poses e pontos de referência;
```

(FERNÁNDEZ-MADRIGAL; CLARACO, 2012)

Algoritmo 3.4: Esboço do algoritmo para cálculo do gradiente e Hessiano para o sistema linear esparsa de um sistema de SLAM Visual.

Entrada: poses da janela de otimização

Entrada: conjunto das arestas com as medições na janela de otimização

Entrada: pontos de referência na janela de otimização

Saída: vetor gradiente e matriz Hessiano para um incremento na variedade linear

- 1 inicialize o vetor gradiente com zero;
 - 2 inicialize a matriz Hessiano com a estrutura esparsa;
 - 3 **para cada** *no apontado por uma aresta na janela de otimização* **faça**
 - 4 | calcule os Jacobianos dos incrementos nas variedades lineares;
 - 5 **para cada** *aresta no grafo da janela de otimização* **faça**
 - 6 | calcule o resíduo no espaço utilizado para parametrização;
 - 7 | calcule o jacobiano dos resíduos nas variedades lineares;
 - 8 | calcule o jacobiano utilizando o jacobiano dos resíduos nas variedades lineares;
 - 9 | calcule os resíduos na variedade linear;
 - 10 | calcule a incerteza da observação na variedade linear;
 - 11 | atualize o vetor gradiente;
 - 12 | atualize a matriz Hessiano;
-

(FERNÁNDEZ-MADRIGAL; CLARACO, 2012)

CAPÍTULO 4

Modificações Propostas

Neste capítulo é feita uma descrição minimalista de como funciona a execução paralela das tarefas de extração de pontos de interesse e cálculo de descritores e a tarefa de ajuste de poses e pontos dos sistemas de SLAM Visual por métodos indiretos baseados em reconstrução geométrica. O objetivo consiste em descobrir como o tempo total necessário para a conclusão da etapa de estimação da pose está relacionado com às tarefas realizadas na linha de execução responsável pela extração de pontos de interesse e cálculo de descritores e na linha de execução responsável pelo ajuste de poses e pontos. A partir desta análise é feita uma proposta de melhoria com o objetivo de diminuir o tempo total necessário por imagem para a conclusão da estimação inicial da pose e posição no espaço dos pontos de referência.

4.1 Pipeline de Processamento de Imagens

Tal como relatado no capítulo 2, os sistemas de SLAM Visual baseados em métodos indiretos possuem uma linha de execução responsável por realizar o rastreamento de pontos de referência nas imagens capturadas pelas câmeras e uma segunda linha de execução responsável por fazer a otimização das poses da câmera e dos pontos de referência a partir das estimativas iniciais obtidas da tarefa de rastreamento. As operações destas linhas de execução são ilustradas nas figuras 2.1, 2.2 e 2.3 de acordo com cada *software* de SLAM, mas com ênfase na implementação do algoritmo, não exibindo nenhuma informação a respeito dos tempos necessários para execução dos blocos individualmente. Assim, observa-se pelas figuras que todos esses sistemas possuem uma etapa de extração de pontos de interesse e cálculo de descritores envolvendo o processamento das imagens capturadas pelas câmeras. Pode-se então, de uma forma sucinta, agrupar todas as operações realizadas nesta linha de execução em dois grupos principais. As relacionadas à extração dos pontos de interesse e cálculo dos seus descritores e as demais operações, estejam elas relacionadas à associação destes pontos entre imagens sucessivas ou demais operações como aquelas relacionadas à decisão pela inserção

de novos *keyframes*. Assim, durante a operação normal de um sistema deste tipo, a linha de execução 1 executa continuamente esta sequência de tarefas conforme ilustrado na figura 4.1.

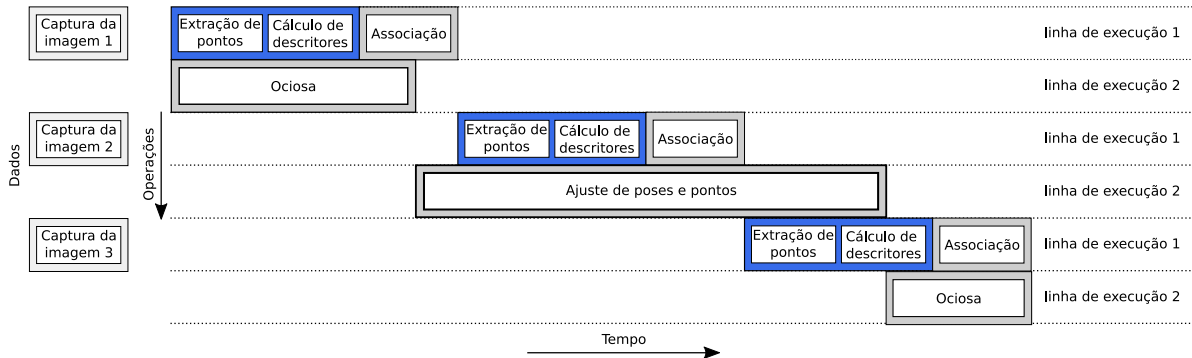


Figura 4.1: Ilustração minimalista da sequência de operações realizadas pelas duas linhas de execução tradicionalmente implementadas em um sistema de SLAM Visual por métodos indiretos baseado em reconstrução geométrica.

É importante salientar que, a figura 4.1 é uma representação extremamente minimalista das tarefas realizadas pelas duas linhas de execução. Nela, é dado ênfase às tarefas que são objetos de análise deste trabalho por demandarem de um tempo considerável para conclusão quando comparado ao tempo entre a captura das imagens. Um ponto relevante para se observar na figura é que a linha de execução responsável pelo ajuste de poses e pontos não executa de forma sincronizada com a etapa de associação, mas em intervalos de tempo constantes (5 mS para o PTAM e 3 mS para o ORB-SLAM) a linha responsável pelo ajuste começa a sua execução até a conclusão da tarefa. É importante lembrar que um dos resultados da execução desta segunda linha, a pose otimizada da câmera, é utilizada na linha de execução 1, caracterizando a situação como um problema de sincronização entre linhas de execução.

Dado que são extraídos mais de 1000 pontos de interesse no PTAM e 2000 na configuração padrão do ORB-SLAM quando avaliado com o *dataset* KITTI, uma análise do tempo gasto no processamento de todos esses pontos é importante para indicar quais tarefas demoram mais para a conclusão. Assim, foi feita a medição do tempo gasto por imagem na extração de pontos de interesse e cálculo dos descritores, associação destes pontos entre imagens consecutivas e estimação da pose e posição dos pontos de referência no espaço. O resultado é exibido nas figuras 4.2 e 4.3.

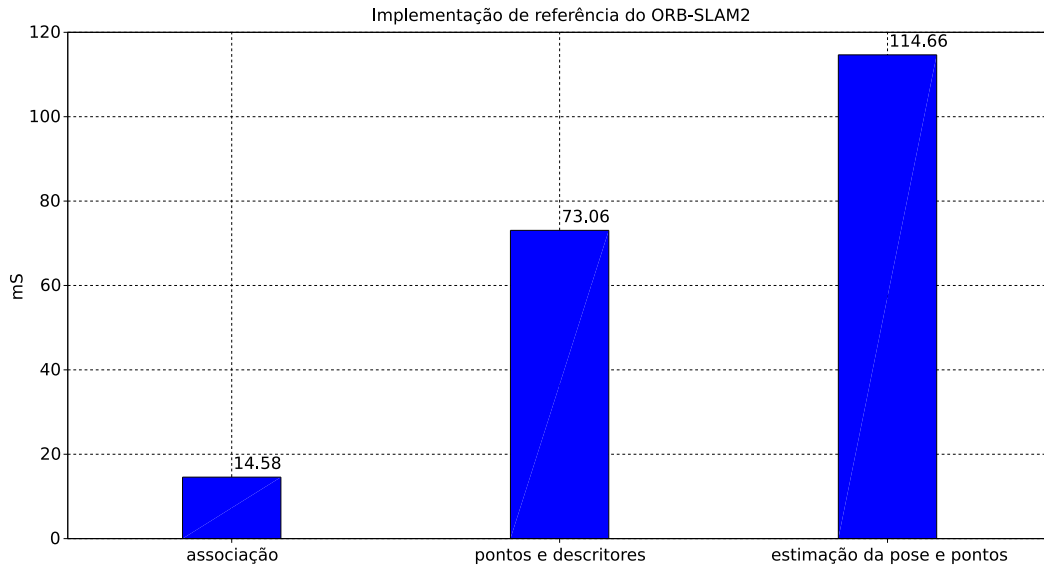


Figura 4.2: Tempos médios para as etapas de associação dos pontos de interesse entre imagens, extração de pontos de interesse e cálculo dos descritores e tempo total entre imagens. Implementação de referência do ORB-SLAM2 executando em um processador i7 7500U.

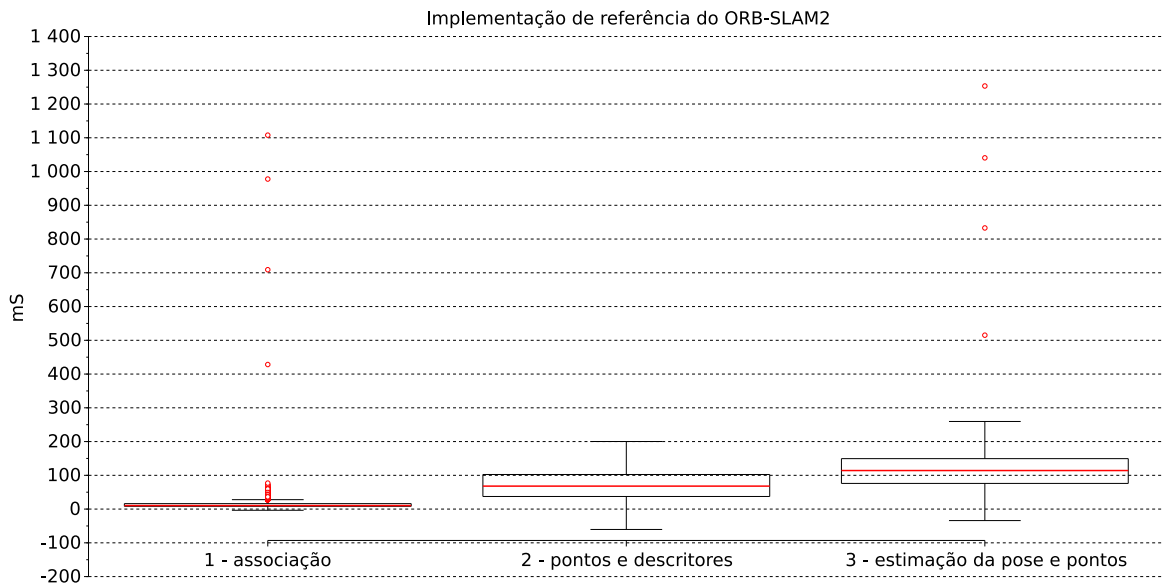


Figura 4.3: *Outliers* dos tempos para estimação da pose e pontos no espaço. Implementação de referência do ORB-SLAM2 executando em um processador i7 7500U.

Pode-se ver a partir das figuras 4.2 e 4.3 que o tempo gasto no processamento dos pontos de interesse (extração e cálculo dos descritores) é muito maior que o tempo necessário para a associação dos pontos entre imagens sucessivas. A partir deste resultado pode-se levantar a questão do que fazer para diminuir o tempo que o sistema necessita aguardar pela conclusão da etapa de extração de pontos de interesse e cálculo dos descritores das imagens, pois este é um fator limitante para o tempo de estimação da pose e, consequentemente, para a velocidade de movimentação da câmera. A resposta mais simples é diminuir o número de pontos calculados, porém esta pode não ser uma opção possível pois isto pode aumentar as

chances de alguns problemas durante a execução do algoritmo de SLAM Visual, como por exemplo:

- Um número pequeno de pontos de interesse pode impedir que o algoritmo de refinamento da associação selecione bons pares de pontos entre imagens diferentes. Isto pode ocorrer, por exemplo, quando a diferença entre o primeiro ponto selecionado para completar o par e o segundo candidato mais adequado é baixa, indicando uma chance alta da associação estar errada (PRINCE, 2012);
- Os algoritmos de detecção de fechamento de laço empregando *bag of words* utilizam os pontos de interesse para o reconhecimento de lugares previamente visitados. Quanto maior o número de pontos de interesse, é menos provável que imagens de locais diferentes sejam consideradas como sendo do mesmo local (falso positivo);
- Ao se utilizar o RANSAC (FISCHLER; BOLLES, 1981) no algoritmo de estimação do movimento da câmera, uma menor probabilidade de associações erradas entre pontos (garantida pelos requisitos do algoritmo de associação) faz com que seja menor o número de testes necessários para uma determinada certeza de encontrar, nas amostras aleatórias, ao menos um conjunto livre de *outliers*, quando estimando o movimento mais provável da câmera (HARTLEY; ZISSERMAN, 2004);
- Mesmo que o algoritmo de associação garanta uma baixa probabilidade de associações erradas, pode acontecer que alguns desses pontos estejam sobre um mesmo plano. Nesse caso o algoritmo de estimação de movimento vai falhar, devido ao sistema linear formado não possuir uma solução única (PRINCE, 2012). Assim, o cálculo do número mínimo de tentativas utilizado pelo RANSAC retorna um valor abaixo do necessário. Uma forma de diminuir a chance desse problema ocorrer é ter muitos pontos e espalhados por toda a imagem.

Dessa forma, é desejável uma outra abordagem para a diminuição dos tempos relevando a escolha de diminuir o número de pontos de interesse para a última opção.

4.2 Proposta de Melhoria

Enquanto o PTAM dividiu o algoritmo de SLAM Visual em duas partes executando em paralelo, neste trabalho propõe-se a modificação da linha de execução responsável pelo rastreo, removendo dela o código que faz o processamento das imagens, e executando-o em outra linha de execução, podendo inclusive adicionar mais de uma. Além disso, propõe-se também que estas linhas de execução sejam responsáveis apenas pela extração de pontos de interesse e cálculo dos descritores, movendo as demais tarefas para a linha de execução responsável pelo ajuste de poses e pontos. Isto é motivado pela dependência da heurística

usada na associação e o valor da pose otimizada, para calcular a posição mais provável da projeção dos pontos de referência nas imagens. Também foi modificada a heurística utilizada na decisão de adição de um novo *keyframe*, substituindo a regra “linha de execução ociosa” (presente no ORB-SLAM) por “limiar inferior de pontos associados atingido”. Por fim, as duas linhas de execução foram modeladas de acordo com o problema da sincronização entre processos, de maneira que a linha responsável pela associação executa apenas quando existem descritores disponíveis, para só então verificar se é necessário a adição de um novo *keyframe* e então realizar a otimização de poses e pontos de referência. Assim, a proposta de modificação apresentada é uma quebra de paradigma do PTAM. O objetivo é diminuir o tempo médio que o sistema necessita para a conclusão da estimação da pose e pontos referentes a cada imagem por meio da adição da capacidade de executar o processamento de imagens diferentes de forma paralela em relação à associação de pontos, ajuste de poses e também ao processamento de outras imagens. Para isso, a sequência de operações da figura 4.1 é modificada de maneira que elas sejam realizadas em um *pipeline* com pelo menos duas linhas de execução. Uma ilustração do funcionamento desta abordagem é feita na figura 4.4. Nesta, utilizou-se blocos com comprimentos proporcionais aos tempos exibidos na figura 4.2 (exceto para o ajuste de poses e pontos), na cor azul para a extração de pontos de interesse e cálculo de descritores e na cor cinza para a associação de pontos.

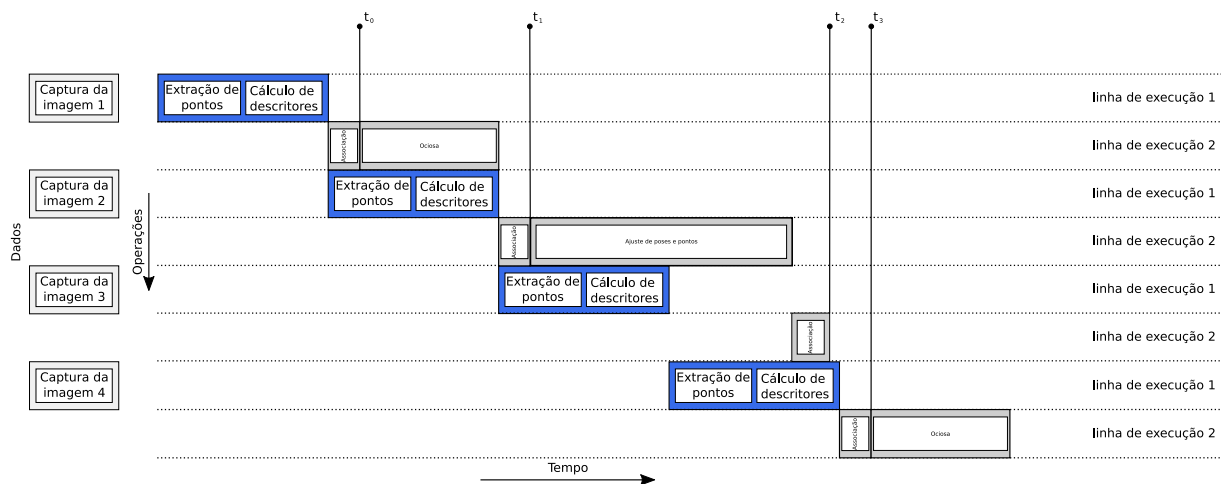


Figura 4.4: Ilustração minimalista de uma sequência de operações quando o processamento da imagem é executado em uma linha de execução e a associação de pontos na mesma linha de execução do ajuste de poses e pontos.

Observando a figura 4.4 é possível perceber que nesta abordagem o sistema não aguarda mais a conclusão da associação para iniciar o processamento da imagem seguinte, bastando para isso a conclusão da etapa de extração de pontos de interesse e cálculo dos descritores. Em paralelo, na linha de execução 2, é feita a associação dos pontos de interesse e ajuste de poses e pontos caso um novo *keyframe* tenha sido adicionado. Em um sistema com

mais uma linha de execução pode-se diminuir ainda mais o tempo médio entre a conclusão da associação dos pontos de interesse das imagens. Isso é ilustrado na figura 4.5.

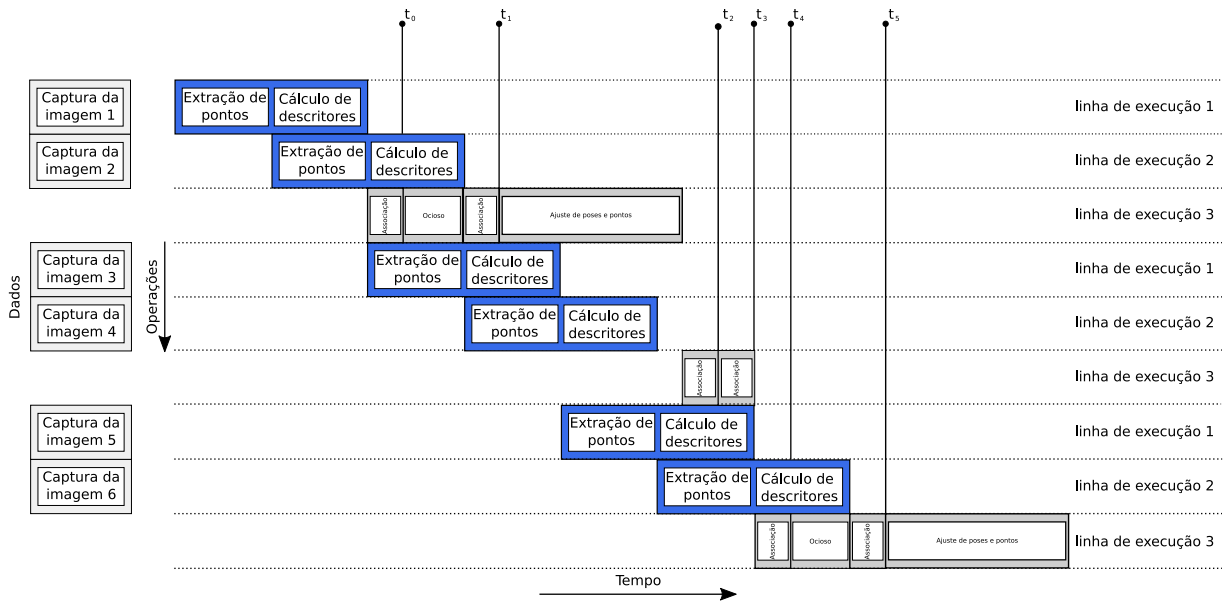


Figura 4.5: Ilustração minimalista de uma sequência de operações quando existem duas linhas de execução disponíveis para a extração de pontos de interesse e cálculo dos descritores e uma para a associação de pontos e ajuste de poses e pontos.

4.3 Distribuição das Imagens

Para implementar as ideias propostas na seção 4.2 é necessário a criação de um método de distribuição das imagens e coleta dos resultados para que o rastreo seja realizado. Neste trabalho, isto foi feito empregando o modelo de algoritmo paralelo que explora o *paralelismo de dados* como também o modelo que explora a relação de processos *mestre e escravo*. Os algoritmos resultantes apresentam características dos dois modelos porque tanto as mesmas operações são realizadas nas imagens de forma concorrente (KUMAR, 2002) como também existe um conjunto de processos cuja função é executar essas operações a medida que as imagens são enviadas para eles pelo processo mestre e de acordo com a fila de processos prontos. Na implementação desta solução, utilizou-se o paradigma da troca de mensagens. Neste, a distribuição das imagens e coleta dos resultados é feita por meio de mensagens para um conjunto de processos em execução tanto local como em máquinas remotas. Isto tem como objetivo verificar a viabilidade da implementação do algoritmo de SLAM Visual como um sistema distribuído onde todos os processos envolvidos podem executar em um único computador, como conectados por uma rede *Ethernet*¹, envolvendo também máquinas de arquiteturas diferentes.

¹Arquitetura de redes locais baseada em no envio de pacotes.

A tarefa de envio da imagem começa imediatamente após a sua captura. Inicialmente o sistema precisa decidir para qual processo enviar a imagem. Para isso, a linha de execução responsável pela captura mantém uma fila de processos prontos. Nesta fila, são adicionados os processos que enviaram uma mensagem ao processo mestre após concluir sua tarefa mais recente. O sistema remove a linha de execução ociosa mais antiga da fila e envia a imagem para ela. Também é possível fazer a divisão da imagem em partes menores e enviar cada parte para uma linha de execução ociosa, obedecendo a ordem em que estas concluíram seus trabalhos, de acordo com a fila de processos prontos. Nos casos em que a imagem é dividida isto é feito na forma de linhas e colunas, formando uma matriz, sendo que cada elemento da matriz é ele próprio uma imagem, contendo uma região da imagem completa. O número de linhas e colunas em que a imagem é dividida são parâmetros informados no momento em que o sistema é iniciado e não mudam durante todo o tempo de execução. As imagens, ou partes delas, são enviadas totalmente decodificadas e em escala de cinza, tal como são organizadas nas matrizes do OpenCV (ITSEEZ, 2015) utilizando o algoritmo 4.1 na sua distribuição. Devido à implementação ser feita na forma de um sistema distribuído, o envio emprega a biblioteca OpenMPI (GABRIEL et al., 2004) para comunicação. Uma ilustração da organização com duas instâncias da linha de execução responsável pela extração de pontos de interesse e cálculo de descritores, na organização do ORB-SLAM, é feita na figura 4.6.

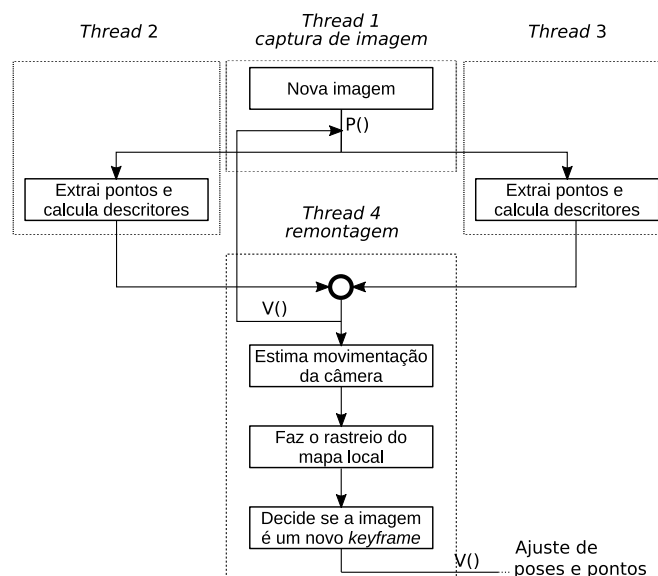


Figura 4.6: Adição de duas linhas de execução na organização do ORB-SLAM para o processamento paralelo das imagens.

Algoritmo 4.1: Algoritmo de envio de imagens**Entrada:** lista de buffers de rede**Entrada:** lista de processos disponíveis**Entrada:** imagem**Entrada:** vetor de semáforos dos processos**Entrada:** semáforo de envio de imagens**Entrada:** semáforo de processo disponível**Saída:** listas atualizadas e imagem enviada aos processos

```

1 P(semáforo de envio de imagens);
2 para  $i \leftarrow 1$  até imagem.linhas faça
3   para  $j \leftarrow 1$  até imagem.colunas faça
4     P(semáforo de processo disponível);
5     remove o processo mais antigo da fila de processos disponíveis;
6     remove o buffer mais antigo da fila de buffers de rede;
7     se buffer de rede está sendo utilizado então
8       └─ aguarde a conclusão do envio;
9     adicione o buffer de rede no fim da fila de buffers de rede;
10    copie o bloco da imagem para o buffer de rede;
11    envie o buffer de rede de forma não bloqueante para o processo mais antigo;
12    se semáforo do respectivo processo  $\neq -1$  então
13      └─ V(semáforo do respectivo processo);

```

4.4 Remontagem dos Dados

Após a conclusão do cálculo dos descritores, o processo escravo envia as posições dos pontos de interesse e os valores dos descritores para o processo responsável por fazer a estimação do movimento. Neste, antes que seja feito o rastreamento entre imagens sucessivas, é realizada a remontagem e reordenação dos blocos de dados. Este passo é necessário porque não existe nenhuma garantia que os blocos serão entregues na ordem correta. Para a remontagem, a estrutura de dados empregada é uma árvore vermelho-preto (GUIBAS; SEDGEWICK, 1978) onde cada nó armazena duas listas ligadas ordenadas, uma para os pontos de interesse e a outra para os descritores, conforme ilustrado na figura 4.7.

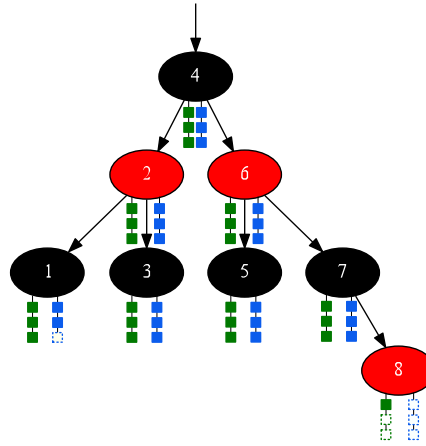


Figura 4.7: Ilustração da árvore vermelho-preto utilizada na remontagem de um grupo de imagens que utiliza a divisão de 1 linha e três colunas. A lista de pontos de interesse é representada pelos blocos verdes e a de descritores pelos blocos azuis. Os blocos pontilhados ainda não foram recebidos pelo processo responsável por realizar a remontagem.

Essa estrutura foi utilizada porque, caso algum bloco de pontos de interesse ou descritores de uma imagem demore a ser entregue, uma lista ligada de imagens cresceria rapidamente em uma situação onde os blocos das imagens seguintes são entregues normalmente. Uma árvore não balanceada também degeneraria para uma lista, resultando em uma busca quase sequencial ao se inserir outros blocos. Para identificação das imagens, o número de cada uma é utilizado como chave de cada nó na árvore vermelho-preto. Quanto às listas ligadas, elas são ordenadas e utilizam como identificador único uma combinação do número da linha e a coluna que cada bloco ocupa na divisão da imagem. No processo de remontagem os algoritmos 4.2 e 4.3 tem uma importância maior. O primeiro adiciona os pontos de interesse ou descritores na respectiva lista presente no nó correspondente da árvore vermelho-preto e o segundo algoritmo verifica se existe um bloco pronto e, caso verdadeiro, retorna os ponto de interesse e descritores do bloco.

Algoritmo 4.2: Algoritmo de adição de pontos de interesse e descritores nas listas da árvore vermelho-preto responsável por organizar os dados para a remontagem.

Entrada: mensagem

Saída: listas atualizadas

```

1 no ← busca na árvore vermelho-preto(mensagem.número da imagem);
2 se no = NULO então
3   | no ← cria nó;
4   | adiciona nó na árvore;
5 se mensagem.tipo = KEYPOINTS então
6   | adiciona na lista(no.keypoints, mensagem);
7 senão
8   | adiciona na lista(no.descritores, mensagem);
  
```

Algoritmo 4.3: Algoritmo de recuperação do vetor de pontos de interesse e matriz de descritores correspondente à imagem mais antiga presente na árvore vermelho-preto.

Entrada: árvore vermelho-preto

Saída: vetor de pontos de interesse

Saída: matriz de descritores

- 1 nó mais antigo $\leftarrow$ árvore mínimo(árvore vermelho-preto);
 - 2 **se** *nó mais antigo* = *NULO* **então**
 - 3 **retorna** falso;
 - 4 **se** (*número de nós na fila de pontos de interesse do nó mais antigo* $\neq$ *número de linhas da imagem* $\cdot$ *número de colunas da imagem*) **ou** (*número de nós na fila de descritores do nó mais antigo* $\neq$ *número de linhas da imagem* $\cdot$ *número de colunas da imagem*) **então**
 - 5 **retorna** falso;
 - 6 **se** *nó mais antigo.número de sequência* $\neq$ *número esperado* **então**
 - 7 **retorna** falso;
 - 8 remove o nó mais antigo da árvore;
 - 9 copia os pontos de interesse do nó mais antigo para o vetor de retorno;
 - 10 copia os descritores do nó mais antigo para a matriz de retorno;
 - 11 **retorna** verdadeiro;
-

O algoritmo responsável pelo recebimento dos blocos de dados e por realizar a remontagem funciona como um produtor para a linha de execução encarregada do envio das imagens. Este tem o semáforo responsável pelo início do envio de uma nova imagem liberado quando o primeiro bloco de dados de uma imagem é recebido pelo processo que realiza a remontagem, seja ele de pontos de interesse ou descritores. O semáforo responsável pelo envio de parte da imagem (ou toda ela, caso não tenha sido dividida) é liberado quando algum processo sinaliza como disponível para o recebimento. Quanto ao processo encarregado do ajuste de poses e pontos, este funciona em sincronia com o algoritmo 4.4. A medida que os blocos de pontos de interesse e descritores são remontados e completam a sequência correta de imagens, o processo responsável pela remontagem aguarda até a conclusão de um possível ajuste que ainda esteja executando. Em seguida, ele realiza o procedimento de associação junto com a adição de possíveis *keyframes* (e outras tarefas), para então liberar o semáforo encarregado do ajuste. Da mesma forma que no algoritmo 4.1, a biblioteca OpenMPI foi

empregada no mecanismo de comunicação utilizado na implementação do algoritmo 4.4.

Algoritmo 4.4: Algoritmo de recebimento das imagens

```

1 terminados  $\leftarrow$  0;
2 liberarImagens  $\leftarrow$  teto((tamanhoGrupo-1) / (imagemLinhas·imagemColunas));
3 para  $i \leftarrow 0$  até liberarImagens faça
4   | V(semáforo de envio de imagens);
5 enquanto terminados < tamanhoGrupo-1 faça
6   | se semaforo dado disponivel  $\neq$  -1 então
7     | P(semáforo dado disponível);
8     | mensagem  $\leftarrow$  recebe mensagem de forma bloqueante;
9     | enquanto mensagem  $\neq$  NULO faça
10      | selecione COMANDO(mensagem) faça
11        | caso TERMINATE faça
12          | terminados  $\leftarrow$  terminados + 1;
13        | caso AVAILABLE faça
14          | adiciona processo no fim da fila de processos prontos;
15          | V(semáforo de processo disponível);
16        | caso DESCRIPTORS faça
17          | adiciona os descritores na árvore de remontagem;
18          | se primeiro bloco da imagem então
19            | V(semáforo de envio de imagens);
20        | caso KEYPOINTS faça
21          | adiciona os pontos de interesse na árvore de remontagem;
22          | se primeiro bloco da imagem então
23            | V(semáforo de envio de imagens);
24      | mensagem  $\leftarrow$  recebe mensagem de forma não bloqueante;
25      | se mensagem  $\neq$  NULO então
26        | P(semáforo dado disponível);
27    | enquanto nó mais antigo da árvore está montado faça
28      | P(semáforo aguarda ajuste);
29      | remove nó mais antigo da árvore;
30      | cria quadro e realiza associação;
31      | V(semáforo libera ajuste);

```

4.5 ORB-SLAM

Neste trabalho, o sistema de SLAM Visual utilizado como referência para a validação das técnicas propostas é o ORB-SLAM. Este é um dos mais populares sistemas de SLAM baseados em pontos de interesse e é largamente adotado para uma variedade de aplicações (YANG et al., 2018). Seu código fonte foi examinado com o objetivo de verificar como foi feita a implementação das linhas de execução e módulos ilustrados na figura 2.2. Concluiu-se que o mapeamento de linhas de execução em *threads* no ORB-SLAM é direto conforme ilustrado na figura 4.8. Nesta, estão listados todos os nomes dos objetos responsáveis pela implementação das *threads* com a sua respectiva função. Além daqueles listados na figura 2.2 existe também a presença de uma quarta *thread* responsável pela interface com o usuário, mas que pode ser desabilitado durante a execução do sistema.

Thread 1	Thread 2	Thread 3	Thread 4
rastreio	mapeamento local	fechamento de laço	interface gráfica
mpTracker	mpLocalMapper	mpLoopCloser	mpViewer

Figura 4.8: Correspondência entre *threads* e objetos responsáveis pela sua implementação no ORB-SLAM.

Além dos objetos que implementam as linhas de execução da figura 4.8, alguns outros também possuem uma maior relevância para o funcionamento do sistema pois manipulam abstrações diretas de conceitos importantes do SLAM Visual. São eles:

- *mpVocabulary* armazena o vocabulário de palavras visuais ORB, auxilia no reconhecimento de locais já visitados e na verificação de equivalência de características;
- *mpKeyFrameDatabase* é responsável por armazenar os *keyframes* e também auxiliar nas operações de relocalização e detecção de fechamento de laço;
- *mpMap* armazena os apontadores para todos os *keyframes* e pontos no mapa;
- *mpFrameDrawer* utilizado pelo módulo *mpViewer* para exibir a imagem que está sendo processada;
- *mpMapDrawer* utilizado pelo módulo *mpViewer* para exibir o mapa que está sendo processado no instante atual.

4.6 Modificações

Para adicionar o paralelismo no processamento das imagens, a implementação de referência do ORB-SLAM foi modificada alterando o módulo *mpTracker* para que este seja

capaz de dividir a imagem (caso seja necessário) e enviar para um ou mais processos disponíveis. Também foi adicionado uma outra linha de execução (*mpMPIReceiver*), que é responsável por receber os dados das imagens processadas e fazer a remontagem dos pontos de interesse e descritores, o rastreamento dos pontos e a estimação da pose. Esta linha de execução também aciona o procedimento de ajuste de poses e pontos, e aguarda a sua conclusão antes de verificar se existe outro bloco de dados pronto. Foram feitas três verificações diferentes para as modificações, sendo uma para cada caso de execução. Os casos de execução são os seguintes: todos os processos executando na mesma máquina; processos distribuídos em máquinas de mesma arquitetura; processos distribuídos em máquinas de arquiteturas diferentes. Em todos os casos é empregado a troca de mensagens por meio da biblioteca OpenMPI e de acordo com a fila de processos prontos para recepção da imagem mais recente. Espera-se que nas verificações o tempo para o cálculo do valor estimado da pose seja menor que o tempo necessário para a extração dos pontos de interesse e cálculo dos descritores para cada imagem individualmente; que o sistema tenha uma maior taxa de processamento de imagens; seja escalável; e com capacidade de operação tanto na mesma máquina, como de forma distribuída em uma rede local.

4.7 Outras Aplicações

Embora este trabalho aborde a extração dos pontos de interesse e cálculo dos descritores com aplicação em SLAM, a técnica apresentada pode ser utilizada em outras situações. Aquelas que podem obter benefício são as aplicações que precisam encontrar pontos de interesse e calcular descritores em sequências de imagens com pequenos intervalos de tempo entre elas, tais como imagens de um filme. Neste cenário (PRINCE, 2012) cita reconhecimento de cenas (FEI-FEI; PERONA, 2005), objetos (CSURKA et al., 2004; SIVIC et al., 2005) e busca através do filme (SIVIC; ZISSERMAN, 2003). A diferença destas aplicações para a de SLAM Visual é a dependência deste quanto ao conhecimento da ordem das imagens. No caso do SLAM, é necessário encontrar o movimento entre quadros. Isto pressupõe o conhecimento da sequência de captura e a associação dos pontos de interesse de quadros sucessivos. Quando as imagens são distribuídas, nada garante que os descritores sejam entregues na mesma sequência. Por esse motivo a técnica proposta faz uma reordenação dos descritores das imagens para a mesma ordem da sequência de captura permitindo que a associação entre pontos seja realizada. Devido a eventos aleatórios na comunicação entre as máquinas envolvidas, esta reordenação pode causar atrasos na associação e estimação de movimento, impactando no desempenho do sistema. Já nos demais casos citados, onde o objetivo é classificar a cena ou algum objeto dela, a dependência do conhecimento da sequência é menor. Dessa forma, a reordenação das imagens após o cálculo dos descritores pode ser removida, permitindo que o sistema atinja um desempenho melhor.

CAPÍTULO 5

Experimentos

Neste capítulo são relatados três experimentos com o objetivo de verificar o desempenho de um sistema de SLAM Visual quando modificado de acordo com as melhorias propostas neste trabalho, de maneira que a extração de pontos de interesse e cálculo de descritores sejam realizados de forma paralela por outros processos. Em todas elas foi empregado o método de paralelização por distribuição de tarefas e o paradigma de comunicação por mensagens. O primeiro experimento verifica o desempenho quando todo o sistema executa completamente em um único computador e a segunda e terceira em vários computadores. Pretende-se, na primeira, verificar se as modificações realmente melhoram o desempenho em uma comparação direta com o ORB-SLAM, quando um sistema é substituído por outro na mesma máquina. Nas duas seguintes o objetivo é verificar como o sistema escala quando executado de forma distribuída em máquinas com desempenho semelhante e quando utiliza-se uma configuração mista onde uma máquina de maior desempenho faz o ajuste de poses e pontos e várias máquinas de menor capacidade fazem a extração de pontos de interesse e cálculo de descritores.

5.1 Experimento 1

Neste primeiro experimento verifica-se a capacidade da modificação proposta de aumentar a quantidade de poses e pontos estimados por segundo, para um sistema de SLAM Visual baseado em métodos indiretos e como ela interfere na qualidade do mapa. Para isso, é feita uma comparação entre a versão do ORB-SLAM, com as modificações propostas, e a implementação de referência disponível no *github* (MUR-ARTAL, 2015). Utiliza-se como métrica o número de perdas de rastreamento do sistema, o tempo necessário para a estimação da pose, o tempo que uma imagem permanece no pipeline até que seja calculada a pose correspondente, o uso de CPU pelo sistema, a comparação qualitativa entre os mapas resultantes e a comparação do erro de rotação e translação dos sistemas modificados com o sistema original.

Na versão original do ORB-SLAM os testes foram feitos com a limitação de processamento de 10 imagens por segundo, presente na implementação de referência, como também sem essa limitação, deixando que o sistema funcione na maior velocidade possível na máquina em que ele executa. Uma outra modificação relevante feita nos dois sistemas é a adição da capacidade de carregar para a memória todas as imagens utilizadas. O objetivo é evitar problemas de latência do dispositivo de armazenamento. Por fim, a linha de execução responsável pela exibição das imagens e dos mapas foi desabilitada para evitar que ela também competisse com os processos responsáveis pela extração de pontos de interesse e cálculo de descritores. Foram feitas 20 repetições de cada configuração, utilizando a implementação de referência e a modificada, com as imagens divididas em uma, duas e quatro partes. O computador empregado nos testes possui um processador de arquitetura i7, modelo 7500U com 16 GB de memória RAM. As imagens utilizadas para validação dos sistemas são as disponíveis no *dataset* KITTI, sendo as de odometria visual 00.

A primeira observação notável é de que, sem a limitação de 10 imagens por segundo, a perda de rastreo ocorre em todas as tentativas de mapeamento feitas utilizando a versão original do ORB-SLAM. Optou-se então pelo uso, nas comparações, apenas da versão com a restrição de tempo, já que a perda de rastreo compromete grandemente a qualidade final do mapa. Também foi preciso resolver um *bug* no extrator ORB utilizado no ORB-SLAM (NETO, 2018) que causa a perda de rastreo em todas as tentativas de mapeamento que são realizadas. Este ocorria de forma aleatória e fazia com que a associação de pontos entre duas imagens fosse de zero pontos. Mais detalhes podem ser encontrados no relatório de *bug* de Neto (2018). Os resultados são comentados nos parágrafos a seguir.

A primeira métrica abordada é o número de perdas de rastreo, sendo os resultados obtidos exibidos na figura 5.1.

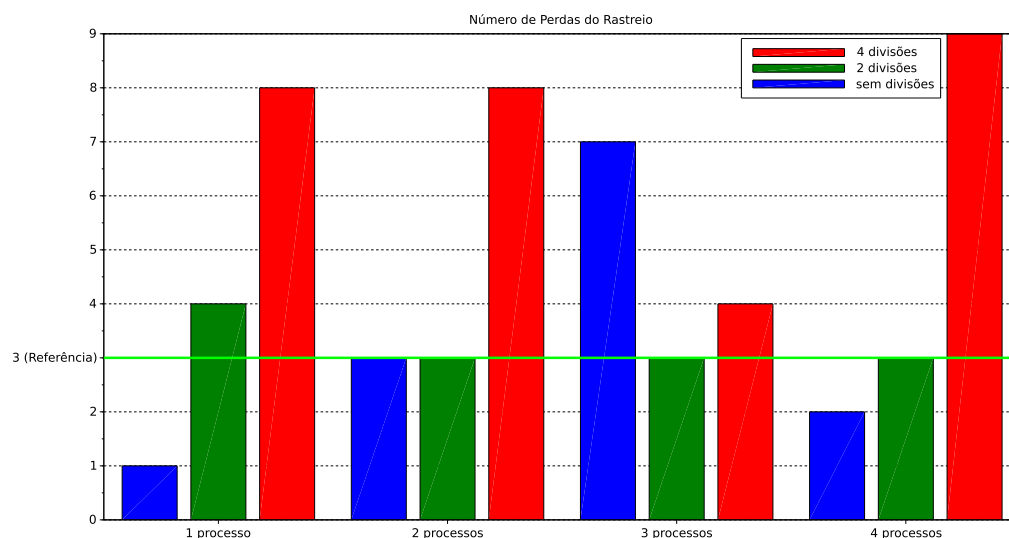


Figura 5.1: Perda de rastreo quando todo o sistema está executando em uma única máquina. O número de processos corresponde aos responsáveis por extrair os pontos de interesse e calcular os descritores.

Percebe-se que os menores valores para a perda de rastreo ocorrem para o sistema com as modificações propostas, sendo que o seu mínimo foi obtido quando se utiliza de imagens sem divisões e apenas um único processo para a extração dos pontos de interesse e cálculo de descritores da imagem. Nota-se que, um maior número e maior frequência de perdas de rastreo ocorrem quando a imagem é dividida e aumenta-se o número de processos. O aumento de perdas de rastreo associado ao número de divisões da imagem ocorre devido a diminuição de pontos de interesse úteis para o rastreo, já que os pontos localizados sobre as bordas não podem ser encontrados quando se divide a imagem.

A segunda métrica é o tempo médio para a estimação da pose da câmera e posição dos pontos de referência. Neste caso, deseja-se verificar se o sistema com as modificações propostas é capaz de estimar as poses e pontos de referência em um tempo menor do que aquele do sistema original. Esta é uma característica desejável porque, caso seja verdadeira, a câmera pode realizar uma captura de mais imagens por segundo, sendo estas aproveitadas no processo de estimação de poses e pontos. Dessa forma, a câmera pode deslocar-se mais rapidamente sem que isso cause a perda do rastreo dos pontos de referência.

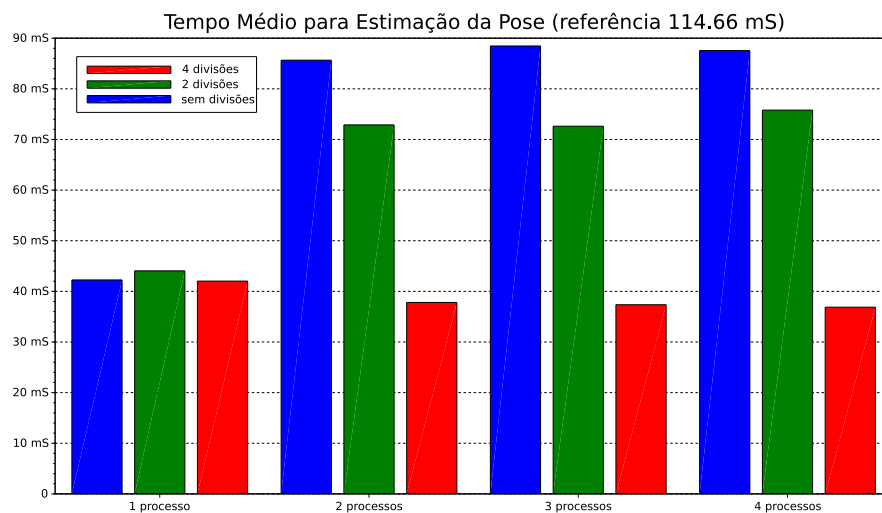


Figura 5.2: Tempo para a estimação da pose quando todo o sistema está executando em uma única máquina. Mesmo com a máquina executando mais processos que o número de núcleos do processador, o sistema apresentou um melhor desempenho que a versão original do ORB-SLAM em todos os casos que foram testados.

A partir das figura 5.2, percebe-se que o sistema modificado necessita de um tempo médio por imagem muito menor que o do sistema original quando apenas um processo foi utilizado para a extração de pontos de interesse e cálculo de descritores. Para os demais casos, o desempenho do sistema modificado ainda apresenta melhores resultados quando comparado com a implementação de referência, mas sem uma diferença tão significativa. A hipótese para a causa deste comportamento é que o sistema está saturado por ter mais linhas de execução e processos em execução do que núcleos de processador disponíveis, forçando

uma competição entre as linhas de execução junto com os processos pelos núcleos da CPU, diminuindo assim o desempenho do sistema como um todo.

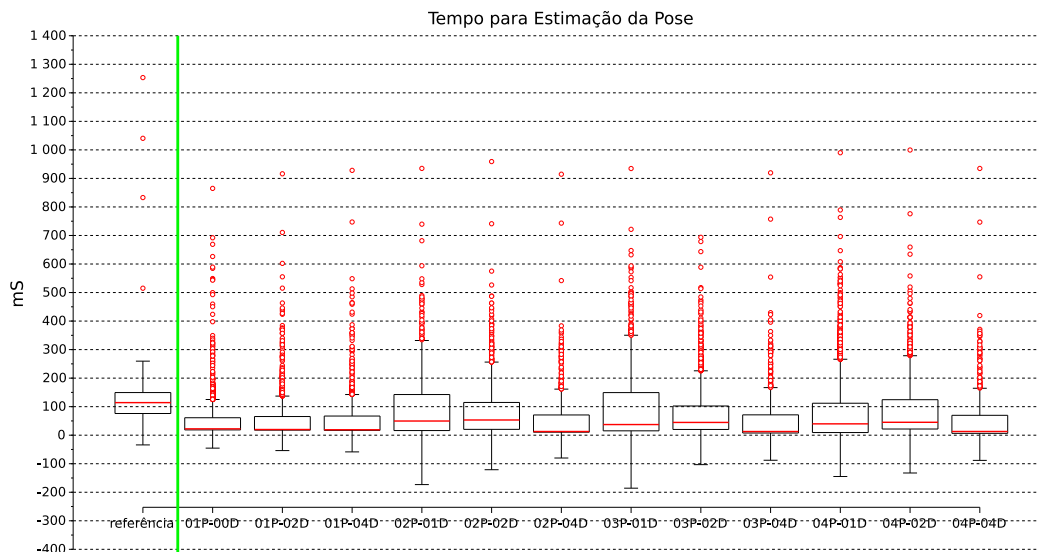


Figura 5.3: Diagrama de caixa do tempo para a estimação da pose quando todo o sistema está executando em uma única máquina. Mesmo com a máquina executando mais processos que o número de núcleos do processador, o sistema apresentou um melhor desempenho que a versão original do ORB-SLAM em todos os casos que foram testados.

Na figura 5.3 é apresentado o diagrama de caixa para o tempo de estimação da pose. Pode-se observar que, quando as modificações propostas neste trabalho são empregadas, existe um aumento do número de *outliers*¹ na medição dos tempos para estimação da pose. Isto acontece devido às características dos algoritmos de só remover uma pose da árvore para ajuste quando todas as poses anteriores foram removidas previamente.

A terceira métrica é o tempo que uma imagem permanece no *pipeline* do momento em que é capturada para a extração de pontos de interesse e cálculo de descritores, até a conclusão da estimação da pose e posição no espaço dos pontos de referência correspondentes aos pontos de interesse da imagem. Esta métrica é importante porque pode-se desejar utilizar a pose da câmera como parâmetro de decisão no sistema de controle de movimento. Neste caso, a demora para a estimação da pose pode ser um fator limitador para a movimentação da câmera. Os resultados são exibidos nas figuras 5.4 e 5.5.

¹Medições que apresentam grande afastamento das esperadas para a série.

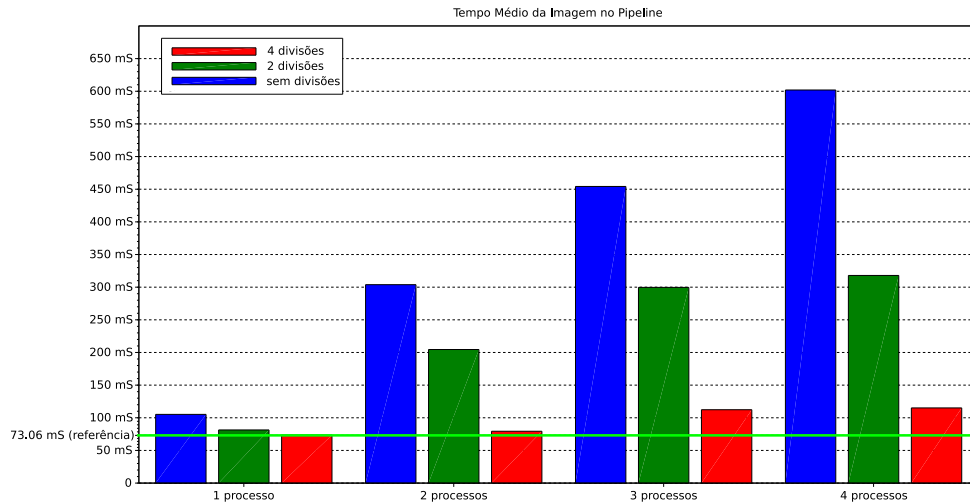


Figura 5.4: Tempo médio que uma imagem permanece no *pipeline* para a estimação da pose quando todo o sistema está executando em uma única máquina.

De acordo com as medições apresentadas na figura 5.4, percebe-se que existe um aumento no tempo médio que uma imagem permanece no *pipeline* do sistema até que a respectiva pose e posição dos pontos seja obtida. Este aumento foi mais perceptível quando não há a divisão das imagens, seguido pelo aumento do número de processos competindo pelos núcleos do processador, já que neste experimento se emprega um número total de processos maior do que o de núcleos.

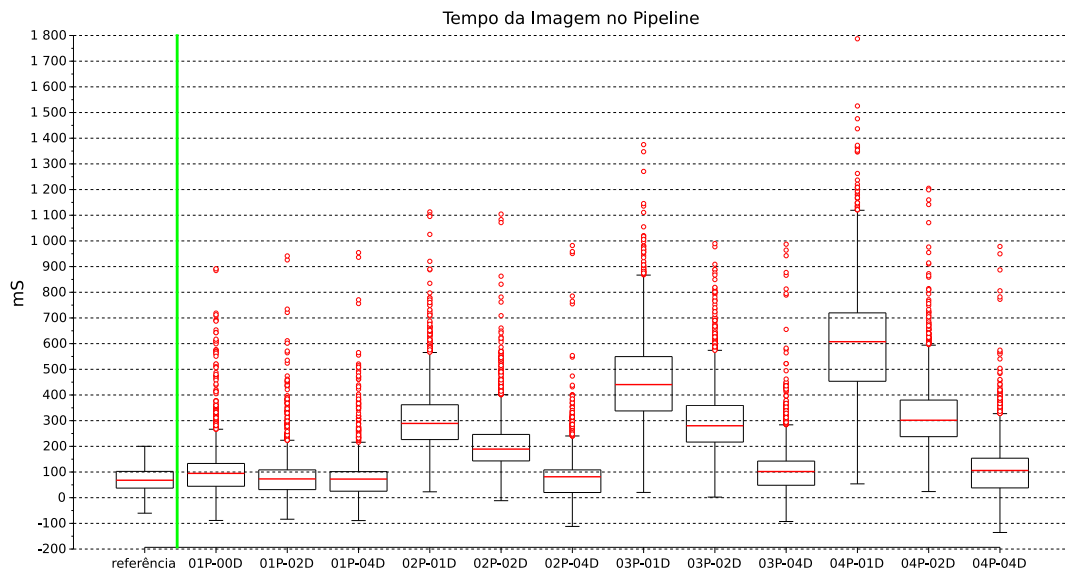


Figura 5.5: Diagrama de caixa do tempo que uma imagem permanece no *pipeline* para a estimação da pose quando todo o sistema está executando em uma mesma máquina.

No diagrama de caixa dos valores médios do tempo da imagem no *pipeline*, percebe-se um aumento na dispersão dos valores medidos da mesma forma que ocorre com tempo para estimação da pose. Esta também é uma consequência da forma como os algoritmos re-

movem as imagens da fila de prontos para realizar a estimação da pose e posição dos pontos de referência.

A quarta métrica empregada é o uso de CPU pelo sistema. Esta é importante porque o uso de CPU está relacionado diretamente com o consumo de energia durante a execução. Os resultados obtidos estão disponíveis na figura 5.6.

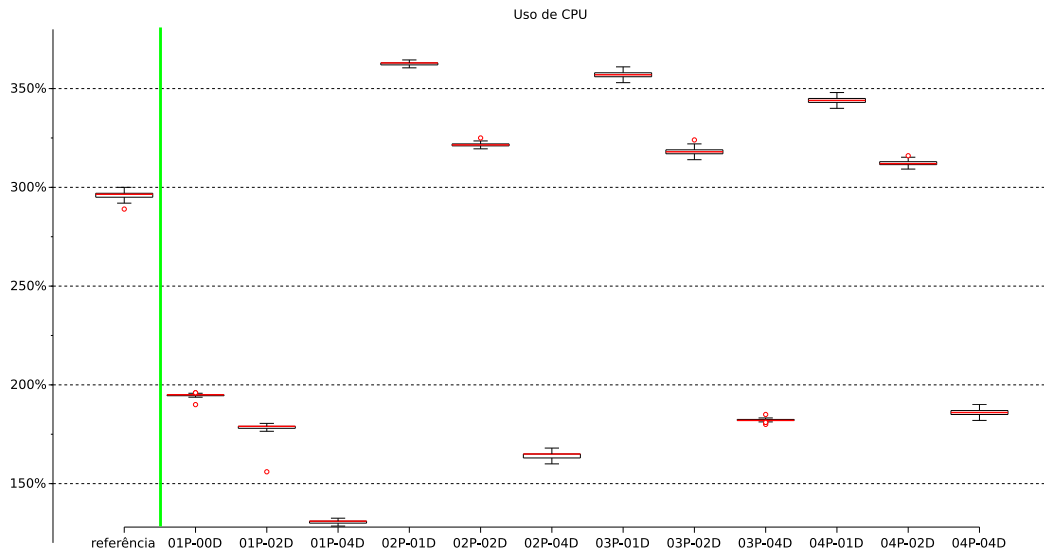


Figura 5.6: Diagrama de caixa do uso de CPU quando todo o sistema está executando em uma única máquina.

Observa-se na figura 5.6 que, quando o sistema modificado utiliza apenas um processo para a extração de pontos de interesse e cálculo de descritores, o uso de CPU é bem menor que a das medições feitas com o sistema original. Já para os demais casos, o uso foi maior, sem existir contudo uma compensação pelo aumento da qualidade de alguma das outras métricas. Novamente, a hipótese é que esta é uma consequência da concorrência entre os processos junto com as linhas de execução pelo uso da CPU.

A quinta métrica é a primeira do grupo que descreve a análise qualitativa e quantitativa da trajetória estimada. É exibida a trajetória de referência, a trajetória obtida com a implementação de referência do ORB-SLAM e mais três trajetórias, todas geradas com as modificações propostas e variando apenas o número de divisões na imagem. Estas configurações são escolhidas por corresponderem às situações de uso em que obtém-se os melhores resultados para as métricas anteriores, quando comparado com a versão original. Quanto aos resultados, pode-se observar pelo mapa da figura 5.7 que todas as versões do sistema apresentam um erro de posicionamento, em relação a trajetória correta, que algumas vezes torna-se bem perceptível. Este é um erro cumulativo, causado por ruído nas medições e incapacidade de estimar exatamente a pose da câmera. O erro propaga-se durante todo o percurso, sendo diminuído apenas ao se detectar um fechamento de laço e executar o ajuste de feixe em toda a trajetória do laço. Em linhas retas o erro cumulativo apresenta-se como um prolongamento ou encurtamento da linha. Nas curvas, o erro apresenta-se como uma

diferença para mais ou para menos no ângulo de abertura e também como desvios na pose da câmera. Nos resultados obtidos, os mapas gerados pelo sistema modificado apresentam todos uma forma semelhante àquela do mapa original. Contudo, existem locais em que eles se encontram mais distantes da referência que o mapa do ORB-SLAM original e em outros mais perto. Nas métricas seguintes é feita uma análise quantitativa desses erros.



Figura 5.7: Comparação do mapa entre os valores de referência para a pose (em azul), com a implementação de referência do ORB-SLAM e a implementação modificada. Todo o sistema executa em uma única máquina.

As figuras 5.8 e 5.9 apresentam o erro de translação de todos os mapas gerados quando comparados com a trajetória original. Na figura 5.8 percebe-se que o erro de translação percentual em relação à trajetória percorrida diminui com o aumento da distância mapeada, apresentando uma curva muito próxima da gerada pela versão original do ORB-SLAM. Isto indica que os sistemas modificados mantêm a característica de convergência da versão original, não importando o número de divisões da imagem. Já na figura 5.9 nota-se que existe uma faixa de velocidades para a câmera em que o ORB-SLAM original apresenta melhores resultados para a translação. Para este gráfico, as versões modificadas também exibem o mesmo comportamento da versão original, assim, as modificações não causaram nenhuma penalidade nesta métrica. É importante salientar que o sistema modificado obtém este resultado com um menor uso de CPU e tempo por imagem.

Nas figuras 5.10 e 5.11 são apresentados os resultados para os erros de rotação. Percebe-se, mais uma vez, que os gráficos dos sistemas modificados acompanham a mesma forma das curvas do sistema original, com uma pequena melhora. Da mesma maneira que o erro de translação, o erro de rotação diminui com o aumento da distância percorrida, con-

forme pode-se ver na figura 5.10. O mesmo ocorre com o erro de rotação relacionado a velocidade de deslocamento. Este apresenta uma região de erro mínimo, tal como exibido na figura 5.11, o que indica uma possível velocidade mais adequada para a operação do sistema.

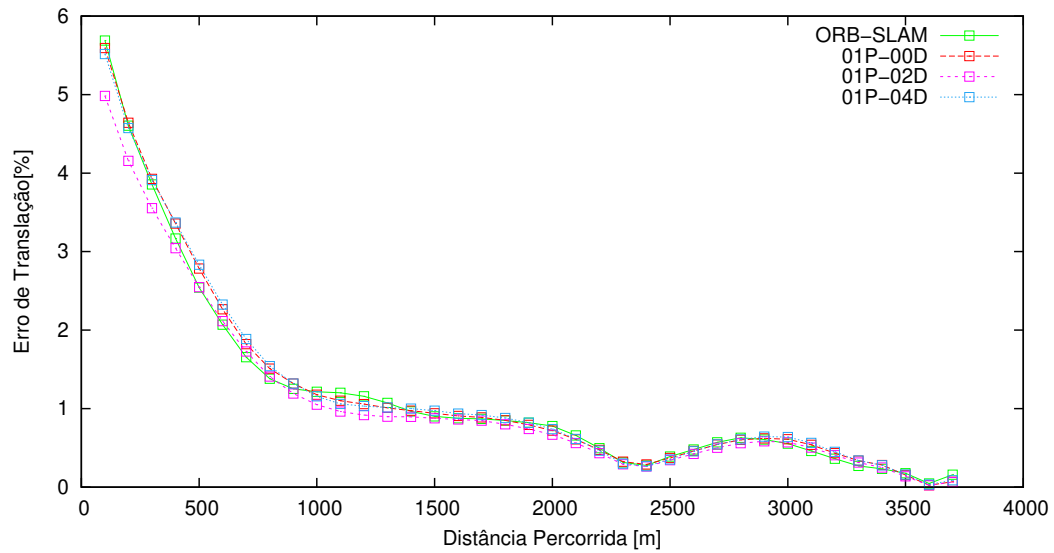


Figura 5.8: Erro de translação percentual contra a distância percorrida.

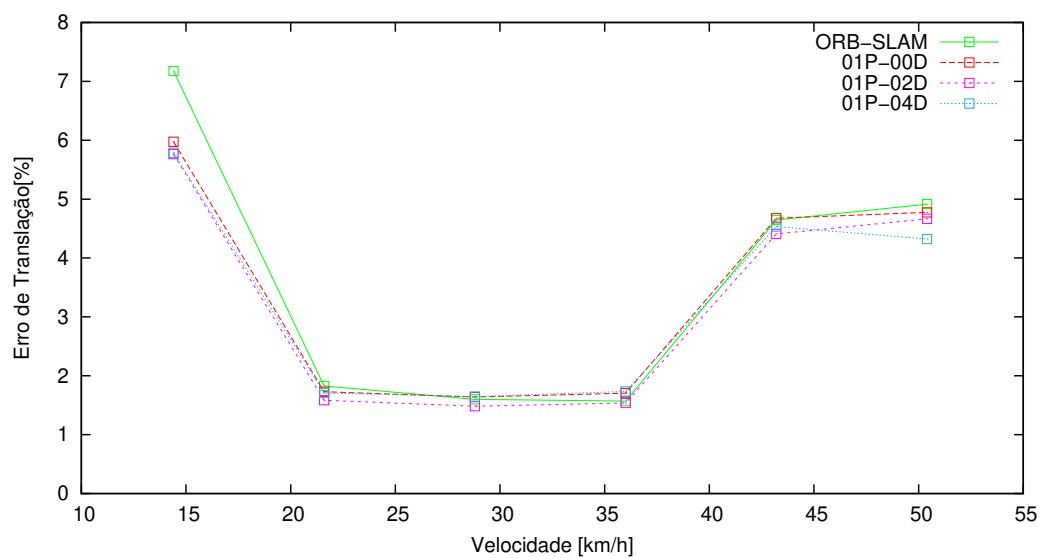


Figura 5.9: Erro de translação percentual contra a velocidade.

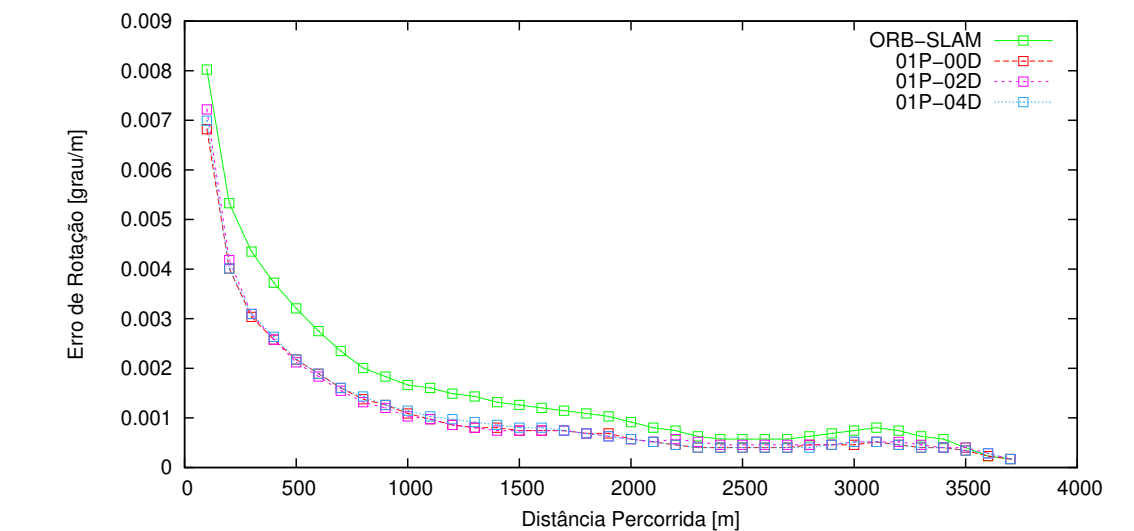


Figura 5.10: Erro de rotação grau/metro.

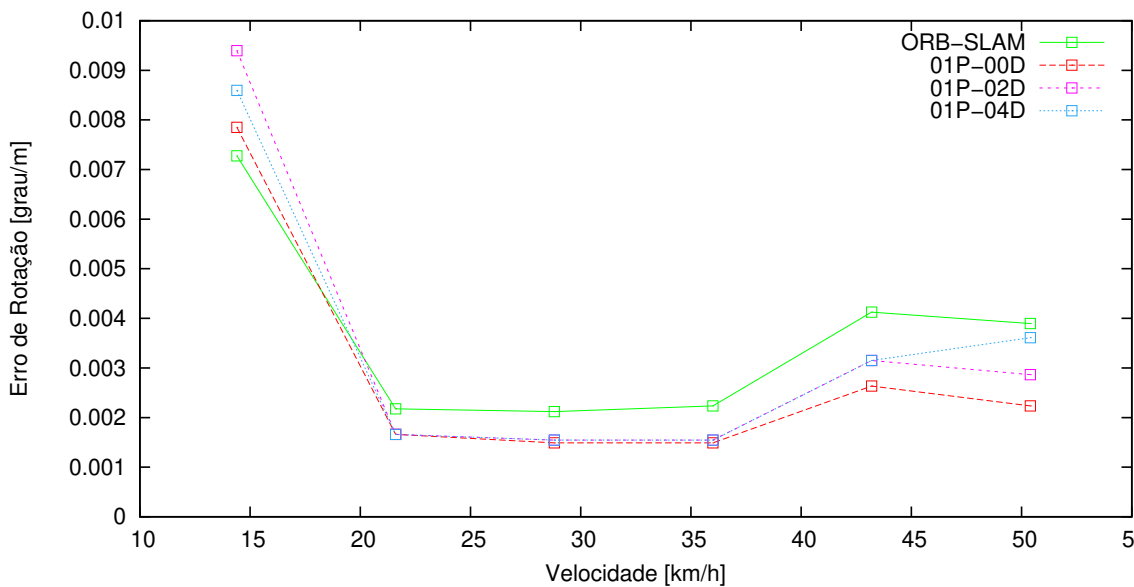


Figura 5.11: Erro de rotação grau/metro contra a velocidade.

Finalmente, a tabela 5.1 apresenta o número de perdas de rastreo junto com o erro RMS para todas as execuções realizadas na mesma máquina. Observa-se a partir da tabela

que, quando o número de perdas de rastreo é maior que zero o erro RMS em geral é muito alto. Os casos em que isto não ocorre deve-se a perda de rastreo ter ocorrido logo no começo do mapeamento.

Iteração	Referência	1P-1L-1C	1P-2L-1C	1P-2L-2C	2P-1L-1C	2P-2L-1C	2P-2L-2C	3P-1L-1C	3P-2L-1C	3P-2L-2C	4P-1L-1C	4P-2L-1C	4P-2L-2C
1	0-9.23	0-10.08	0-10.27	0-12.11	0-9.26	0-9.08	0-11.28	0-9.73	1-48.51	0-9.29	0-10.99	0-10.00	0-7.96
2	0-10.45	0-10.08	0-10.53	0-11.83	0-6.38	0-6.17	0-11.83	0-9.02	0-10.31	0-8.95	0-9.59	0-7.79	1-52.55
3	0-9.73	0-11.18	0-11.10	2-53.07	0-9.04	0-11.57	0-14.11	0-10.16	0-8.39	0-13.18	0-7.45	0-7.67	0-12.39
4	0-11.21	0-10.85	0-9.99	0-11.01	0-10.51	0-7.96	2-52.76	2-51.73	0-7.89	0-8.63	0-10.09	0-10.96	1-26.85
5	1-47.26	0-9.57	0-6.75	1-51.99	0-11.99	0-7.05	0-10.10	0-9.70	0-7.95	0-13.76	0-15.85	0-9.99	0-10.25
6	0-11.24	0-9.92	0-9.57	0-15.98	0-10.06	0-8.77	0-10.93	0-12.64	0-6.50	0-13.11	0-11.35	0-10.40	0-10.10
7	0-9.14	0-14.64	0-11.91	0-11.15	0-9.36	0-6.69	0-9.19	0-9.59	0-42.93	0-13.17	0-10.17	0-9.61	2-49.50
8	0-9.63	0-8.22	2-49.74	0-12.23	0-11.25	0-8.91	2-52.23	0-9.43	0-10.64	0-12.57	0-10.18	1-8.31	2-78.87
9	0-8.05	0-10.52	0-11.30	0-11.32	0-12.73	0-9.63	0-18.45	2-52.49	0-8.82	1-7.55	0-21.28	0-11.10	0-8.13
10	0-10.04	0-9.07	0-8.25	1-51.63	0-10.63	0-7.30	0-10.10	1-47.23	0-8.89	0-12.36	0-10.23	0-8.29	0-12.28
11	0-8.01	0-10.62	0-8.10	0-12.64	0-9.13	0-9.00	0-10.79	1-52.02	0-7.80	0-11.89	0-9.10	0-9.12	0-10.02
12	0-9.23	0-9.71	0-7.04	0-14.50	1-58.65	0-8.91	0-14.69	0-12.34	0-9.66	0-9.66	0-9.30	0-7.14	0-12.92
13	0-10.08	0-10.20	0-9.03	0-11.63	0-10.25	0-11.64	0-12.65	0-6.51	0-8.86	2-50.30	0-8.42	0-8.92	0-13.90
14	0-9.46	0-8.98	0-8.60	0-14.97	0-10.78	0-6.98	0-11.97	0-9.68	2-49.48	0-10.01	0-9.02	0-9.42	0-14.08
15	0-9.17	0-8.18	0-9.22	2-49.99	1-47.57	0-9.32	1-53.33	0-12.27	0-9.48	1-52.72	0-8.97	0-27.07	0-11.97
16	2-51.69	0-10.56	2-50.64	2-51.36	0-10.16	0-11.95	0-9.33	0-11.11	0-8.59	0-11.17	0-7.70	0-8.14	1-30.27
17	0-9.45	1-48.83	0-10.01	0-10.98	0-12.50	0-9.79	0-11.22	0-19.53	0-8.85	0-11.64	2-52.94	2-49.19	0-10.38
18	0-10.30	0-10.23	0-10.22	0-13.67	0-9.22	0-12.47	0-10.04	1-24.05	0-7.64	0-10.48	0-8.38	0-6.26	2-50.22
19	0-9.24	0-9.60	0-10.41	0-7.84	1-10.16	3-48.92	0-11.38	0-8.84	0-8.69	0-8.57	0-10.31	0-7.52	0-13.29
20	0-10.27	0-11.52	0-10.15	0-13.08	0-12.89	0-7.27	3-59.44	0-7.88	0-8.05	0-10.00	0-8.62	0-9.36	0-11.93

Tabela 5.1: Perdas de rastreo e erro RMS (m) para a execução na mesma máquina.

5.1.1 Discussão

A partir dos resultados obtidos, nota-se que ao aplicar as modificações propostas neste trabalho e comparar as métricas com as do sistema original, a versão modificada apresenta um melhor desempenho para o caso em que não existem divisões nas imagens e utiliza-se um único processo para a extração de pontos de interesse e cálculo de descritores. Nesta situação, o sistema modificado apresentou um aumento real no número de poses estimadas por segundo, com uma diminuição do uso de processador e da perda de rastreo. A qualidade final do mapa é semelhante a do sistema original, tendo piorado apenas no tempo que uma imagem permanece no *pipeline*. A queda de desempenho para os outros casos deve-se, provavelmente, tanto à saturação do processador como também a diminuição de pontos de interesse em imagens consecutivas (devido ao posicionamento sobre as bordas da divisão da imagem). Contudo, estas métricas podem ser analisadas novamente nos outros dois experimentos. Por fim, a comparação com o sistema original, sem o limite de tempo por imagem, não foi feita porque, neste caso, o sistema original perde o rastreo em todas as tentativas de mapeamento, o que impede a criação de um mapa completo tal como faz a versão modificada.

5.2 Experimento 2

Neste segundo experimento, o processador i7 7500U é novamente responsável por dividir e distribuir as imagens como também realizar a remontagem dos blocos de pontos de interesse e descritores, fazer a associação de pontos em imagens diferentes e ajustar as poses e pontos de referência. Entretanto, a extração de pontos de interesse e cálculo dos descritores é feita em um outro processador i7, modelo 3517U, com comunicação por meio de uma rede *Gigabit Ethernet*. Além do processamento da imagem inteira, as medições são feitas com divisões em duas e quatro partes. Quanto ao número de processos utilizados para cada um dos casos, estes variam de um até quatro. Da mesma forma que no primeiro experimento, é usado o mesmo conjunto de dados do *dataset KITTI*, com o procedimento de mapeamento sendo repetido 20 vezes por configuração. O objetivo deste experimento é verificar como o sistema escala quando executa de forma distribuída e como isto afeta as métricas utilizadas neste trabalho.

A primeira métrica avaliada é, novamente, o número de perdas de rastreo, com os resultados sendo exibidos na figura 5.12. Pode-se ver que, neste experimento, quando o sistema executa sem dividir as imagens não ocorre a perda de rastreo em nenhuma das oitenta vezes em que o mapeamento é realizado. Já nas configurações em que as imagens são divididas, observa-se que existe a perda de rastreo em um número quase sempre maior que o da implementação de referência. Mais uma vez, este resultado advém do menor número de pontos de interesse úteis, já que os que se localizam sobre a região de borda das imagens não são encontrados pelo algoritmo extrator do ORB quando a imagem é dividida. Os resultados desta primeira métrica são importantes para a hipótese deste trabalho porque observa-se um aumento real da robustez do sistema quando executando de forma distribuída em duas máquinas de capacidade semelhante.

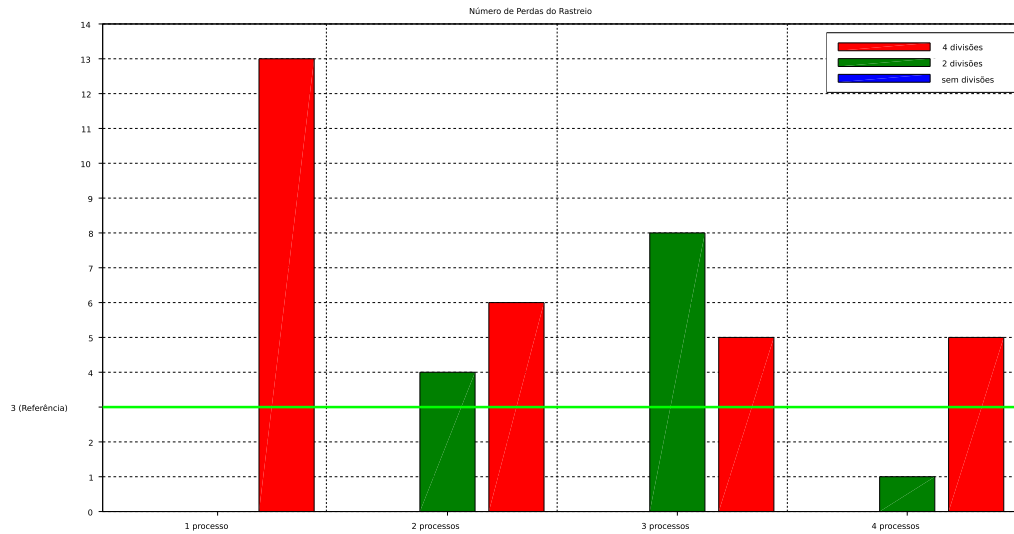


Figura 5.12: Número de perdas de rastreo quando a extração de pontos de interesse e cálculo de descritores é feita em uma outra máquina com processador i7 3517U e comunicando-se por meio de rede *Gigabit Ethernet*.

A segunda métrica empregada é o tempo para a estimação da pose da câmera em cada uma das imagens. Deseja-se verificar como este tempo diminui a medida que o número de processos encarregados da extração de pontos de interesse e cálculo de descritores aumenta. Os resultados são exibidos nas figuras 5.13 e 5.14.

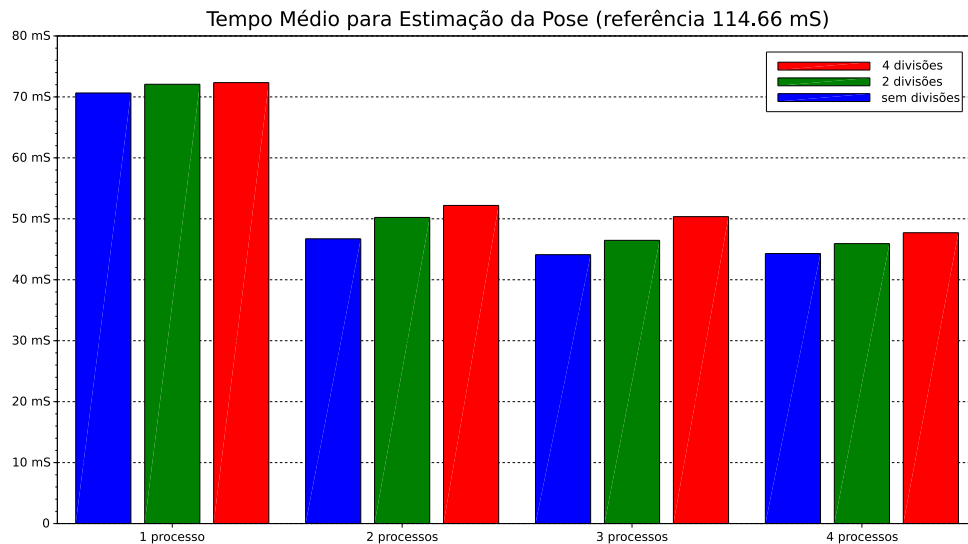


Figura 5.13: Tempo médio para a estimação da pose quando os processos responsáveis pela extração de pontos de interesse e cálculo de descritores executam de forma distribuída em um processador i7 3517U.

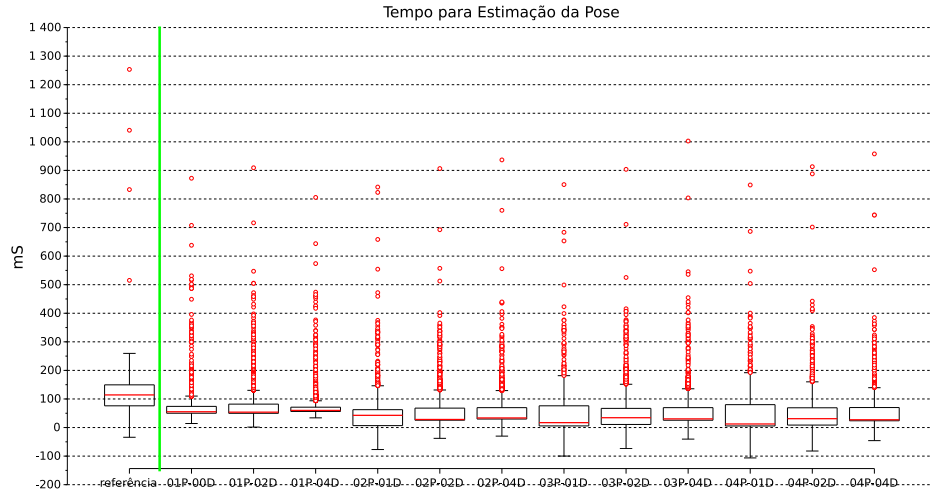


Figura 5.14: Diagrama de caixa do tempo médio para a estimação da pose e posição dos pontos de referência quando os processos responsáveis pela extração de pontos de interesse e cálculo de descritores executam em uma outra máquina de desempenho semelhante à responsável pelo ajuste de poses e posição dos pontos.

A partir dos gráficos das figuras 5.13 e 5.14 pode-se concluir que o sistema apresenta um desempenho melhor que a versão original. É notável que o tempo médio para a estimação da pose e pontos diminui a medida que o número de processos aumenta, mas apresentando um comportamento assintótico horizontalmente. Pelo diagrama de caixa da figura 5.14, observa-se que o efeito colateral da modificação do algoritmo é que o sistema passa a apresentar um espalhamento maior dos valores medidos, apresentando bem mais *outliers*. É importante salientar também que o teste estatístico de Kolmogorov–Smirnov foi aplicado nas medições e nenhuma delas apresenta distribuição normal para $\alpha = 0.05$, inclusive a versão original sem modificações.

Os resultados obtidos aparentam obedecer a uma função assintótica, ao variar unicamente o número de processadores empregados, de forma que é possível realizar o ajuste de uma curva com formato $t = \frac{A}{B^p} + C$ aos valores médios. Aplicando a implementação do algoritmo de otimização de Levenberg–Marquardt (HARTLEY; ZISSERMAN, 2004), obtém-se a equação 5.1 e a matriz de covariância Σ (equação 5.2) para o ajuste de uma curva conforme a figura 5.15.

$$t = \frac{271,18}{10,21^p} + 44,08 \text{ mS} \quad (5.1)$$

$$\Sigma = \begin{bmatrix} 47824481,00 & 1853867,20 & 185194,85 \\ 1853867,20 & 72386,39 & 7660,91 \\ 185194,85 & 7660,91 & 1809,87 \end{bmatrix} \quad (5.2)$$

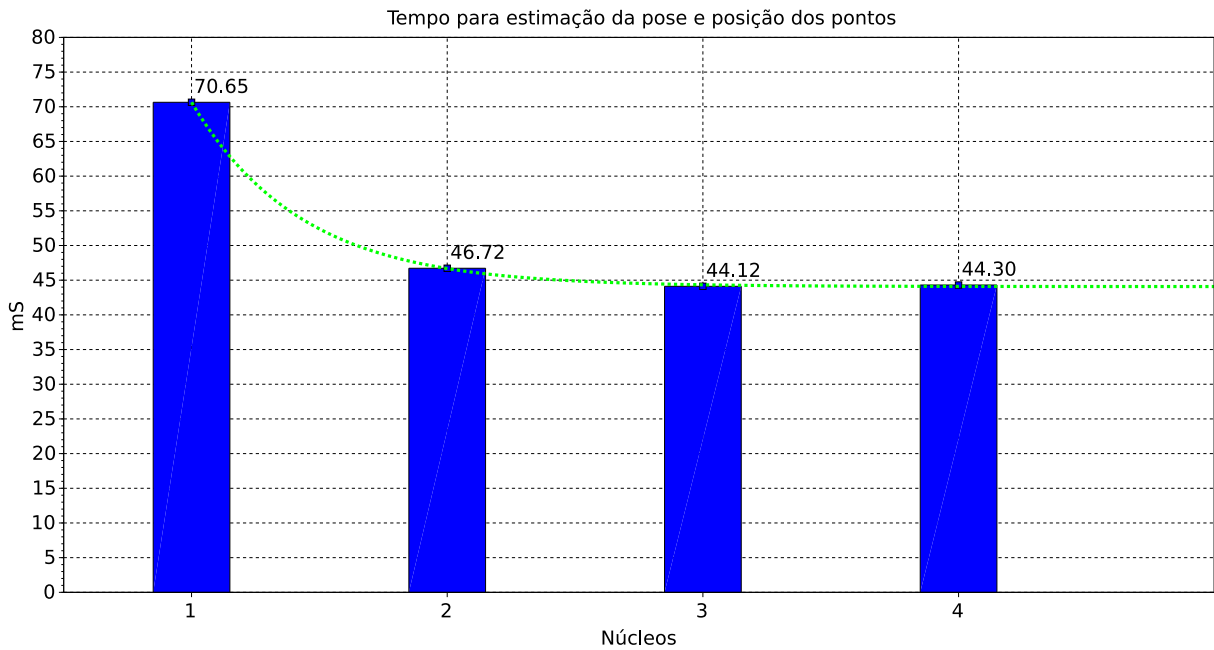


Figura 5.15: Ajuste da curva para o tempo médio de estimação da pose e posição dos pontos de referência para imagens sem divisões. Processos executando no processador i7 3517U.

Como terceira métrica, repete-se o tempo que uma imagem permanece no *pipeline* do sistema desde o momento da captura até a estimação da pose. Diferentemente do primeiro, neste experimento faz parte do processamento o envio da imagem para outra máquina e o recebimento dos pontos e descritores por meio da rede local, dessa forma é esperado um aumento do tempo médio de permanência de cada imagem no *pipeline*. Os resultados obtidos para os valores médios das medições são apresentados na figura 5.16 e o diagrama de caixa com informações dos *outliers* na figura 5.17.

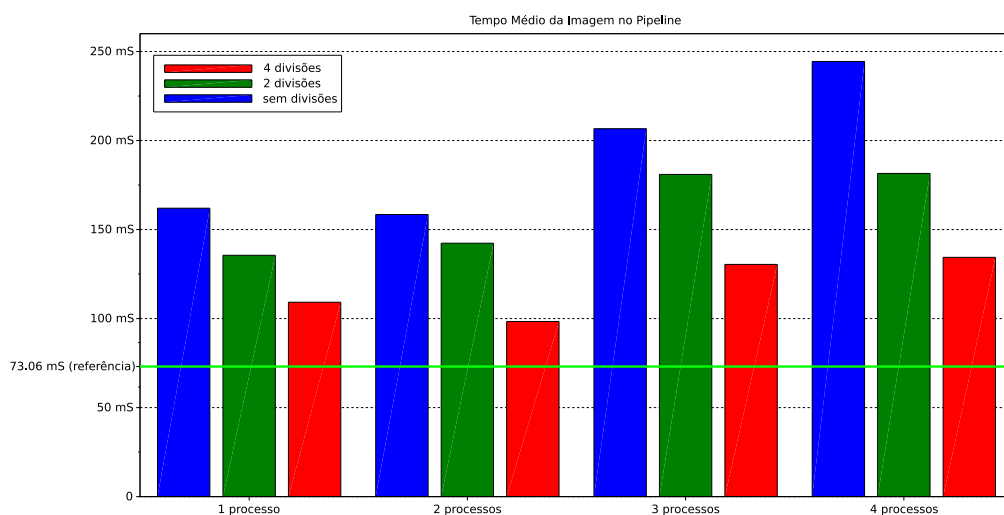


Figura 5.16: Tempo que a imagem permanece no *pipeline* desde a captura até a conclusão da estimação da pose quando o sistema executa de forma distribuída utilizando o processador i7 7500U para a otimização e o i7 3517U para extração de pontos de interesse e cálculo dos descritores.

Na figura 5.17 pode-se observar que, tal como acontece com o tempo para a estimação da pose, a adição das modificações faz com que o sistema aumente o tempo de permanência da imagem no *pipeline* e apresente medições que se distanciam dos valores médios. Além disso, o gráfico apresenta uma tendência de aumento do tempo no *pipeline* quando aumenta-se o número de processos empregados na extração de pontos de interesse e cálculo de descritores. A causa deste comportamento é que a remontagem dos dados e estimação da pose acontece na mesma ordem em que as imagens foram enviadas, forçando assim, a espera de uma imagem pela conclusão do processamento de outra (que está em outra máquina) no caso do recebimento fora de ordem. É preciso salientar que, embora esta métrica apresente um valor pior que o de referência, ela é compensada pela diminuição do valor médio do tempo entre a estimação das poses.

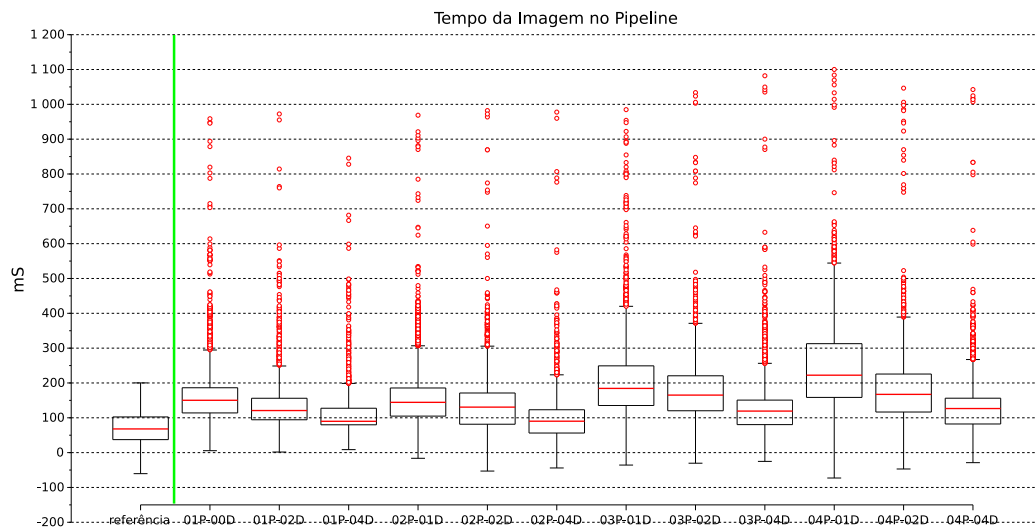


Figura 5.17: Diagrama de caixa do tempo que uma imagem permanece dentro do *pipeline* de processamento desde o momento da captura até a estimação da pose. Percebe-se que no sistema modificado existe um aumento do tempo médio com o número de processos e também um maior espalhamento dos tempos medidos.

O uso de CPU pela máquina responsável por fazer a estimação da pose é exibido na figura 5.18. Estes valores apresentam um valor bem mais baixo que o observado na figura 5.6, indicando assim que, quando a extração de pontos de interesse e cálculo de descritores é feita em outra máquina, pouco mais que um núcleo é necessário para a associação e otimização. Isto é um indicador que uma única máquina pode ajustar vários mapas ao mesmo tempo, requerendo pouco mais que um núcleo do processador por mapa.

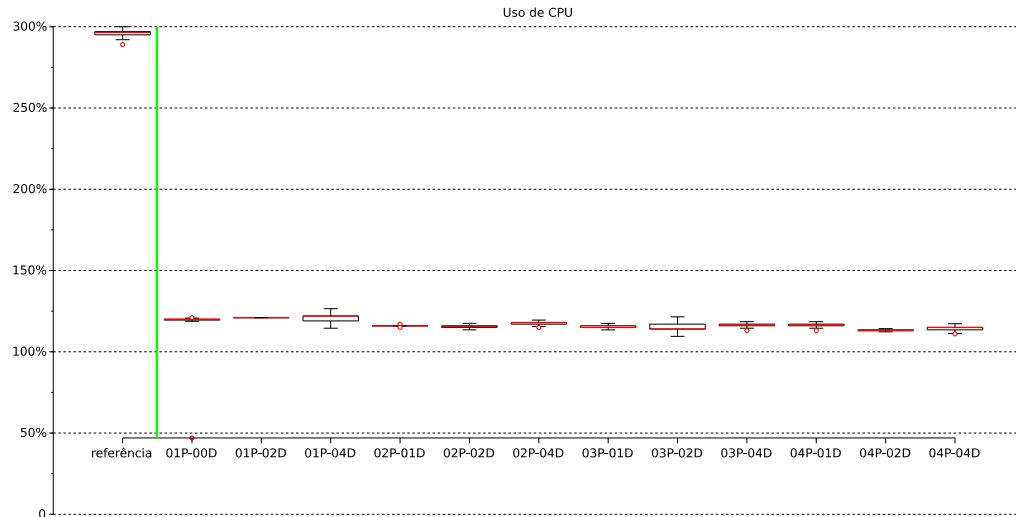


Figura 5.18: Uso de CPU quando o sistema executa de forma distribuída em máquinas com capacidade de processamento semelhante.

A quinta métrica aplicada é, novamente, a qualidade do mapa. Para isso, são plotados o mapa com os valores de referência, o mapa gerado a partir do ORB-SLAM original e quatro instâncias do mapa gerados pela versão modificada do ORB-SLAM executando de forma distribuída. A comparação é feita com os mapas produzidos pelas imagens sem divisões porque esta é a situação em que o sistema mostra um melhor desempenho para a métrica da perda de rastreamento (conforme o gráfico da figura 5.12). Os resultados são exibidos na figura 5.19.



Figura 5.19: Comparação com o mapa de referência, mapa da versão original do ORB-SLAM e mapas criados com a versão modificada do sistema executando a extração de pontos e cálculo de descritores em um processador i7 3517U comunicando-se por rede *Gigabit Ethernet*.

O resultado obtido, exibido na figura 5.19, é muito semelhante àquele do primeiro experimento, com as trajetórias próximas daquela produzida como resultado da execução do ORB-SLAM original, apresentando erros de rotação e translação com comportamento parecido e pelos mesmos motivos.

As métricas quantitativas são, mais uma vez, representadas pelos erros de translação e rotação. Nas figuras 5.20 e 5.21 estes erros são exibidos contra a distância percorrida e velocidade para a execução do sistema de forma distribuída empregada neste experimento. Percebe-se que a versão modificada mantém a propriedade de convergência do ORB-SLAM e continua apresentando uma faixa de velocidades onde a operação resulta no menor erro de translação, conforme a figura 5.21.

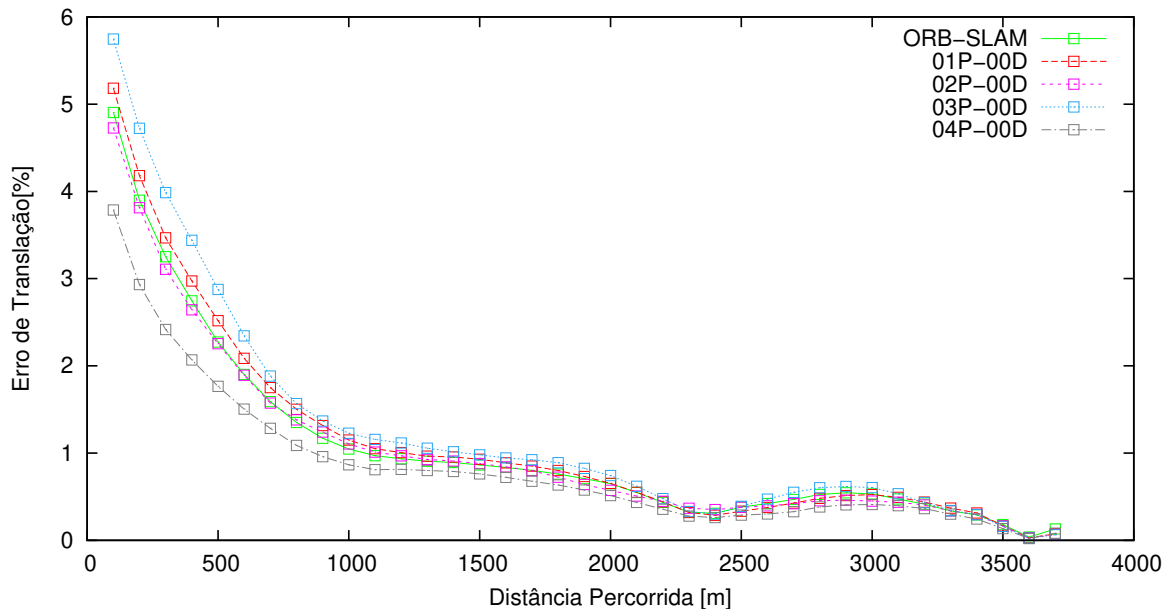


Figura 5.20: Erro de translação percentual contra a distância percorrida.

Da mesma forma que os erros de translação, os erros de rotação repetem o comportamento do sistema do primeiro experimento. Contudo, para o erro de rotação contra a velocidade de deslocamento, além da faixa de velocidades onde o sistema apresenta os melhores resultados, a maioria dos gráficos das iterações, para a implementação com as modificações propostas, apresentam um valor menor que o da implementação de referência quando executando em velocidades mais altas. A modificação que justifica essa diferença nos resultados é a mudança na heurística que decide pela adição de um novo *keyframe*. No sistema modificado a heurística consiste em um limite inferior para o número de pontos de interesse associados entre imagens sucessivas.

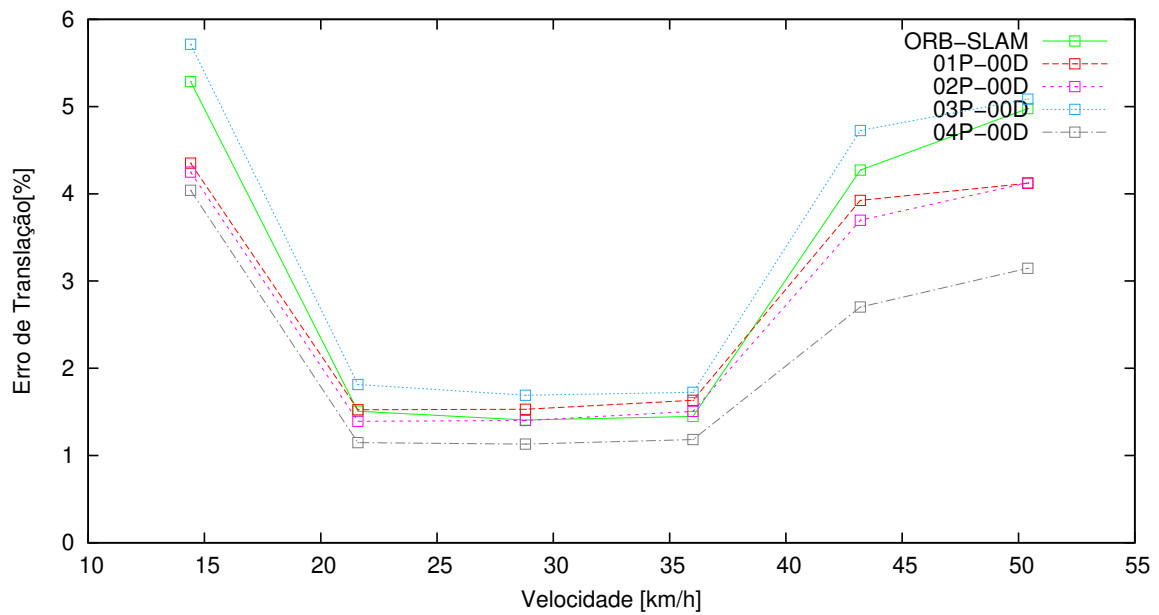


Figura 5.21: Erro de translação percentual contra a velocidade.

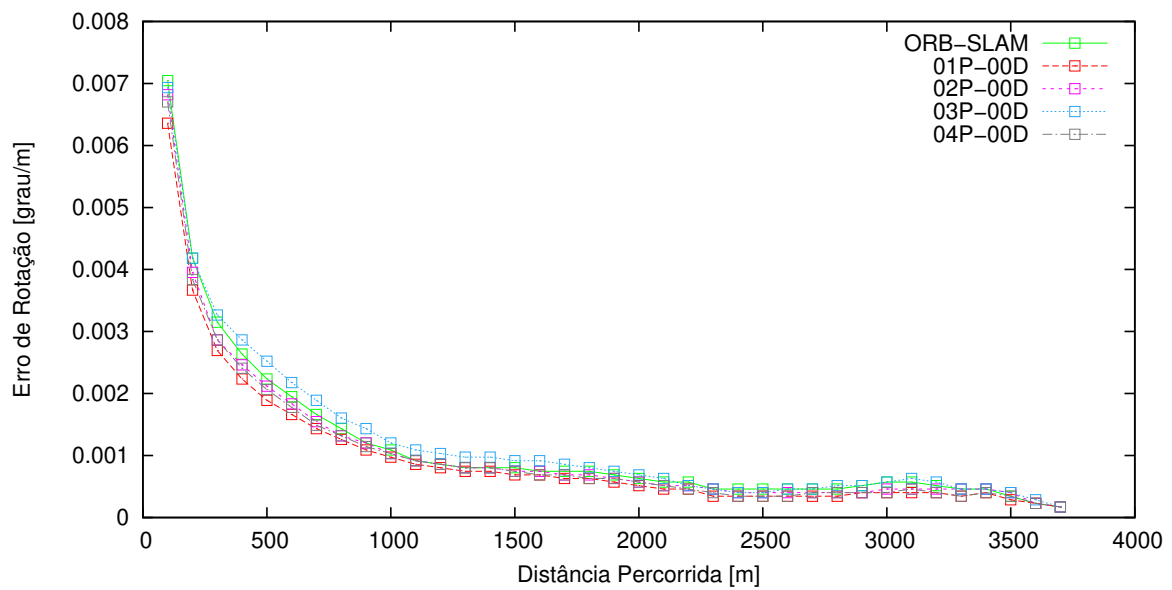


Figura 5.22: Erro de rotação grau/metro.

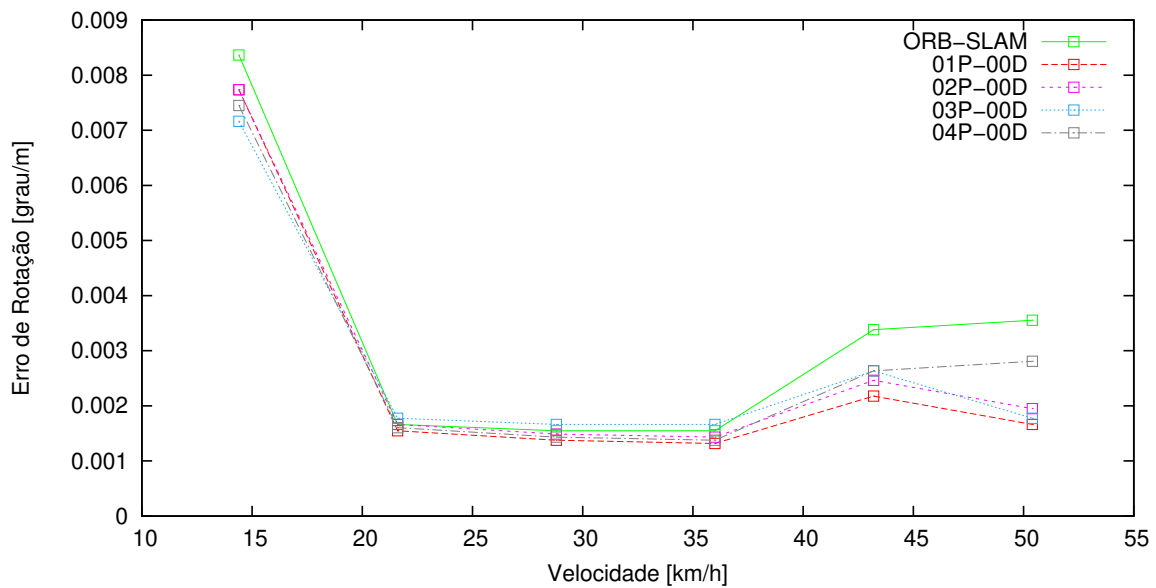


Figura 5.23: Erro de rotação grau/metro contra a velocidade.

Iteração	Referência	1P-1L-1C	1P-2L-1C	1P-2L-2C	2P-1L-1C	2P-2L-1C	2P-2L-2C	3P-1L-1C	3P-2L-1C	3P-2L-2C	4P-1L-1C	4P-2L-1C	4P-2L-2C
1	0-9.23	0-10.43	0-7.08	1-49.56	0-9.09	0-10.23	0-10.62	0-10.56	0-9.20	0-15.63	0-7.68	0-10.66	0-21.38
2	0-10.45	0-28.88	0-11.19	0-11.39	0-11.15	0-12.50	0-9.09	0-9.78	0-7.24	0-14.13	0-17.54	0-9.41	1-111.80
3	0-9.73	0-11.27	0-9.40	2-50.57	0-8.71	0-7.08	0-10.21	0-8.13	0-15.57	0-10.81	0-9.88	0-10.91	0-14.79
4	0-11.21	0-9.44	0-8.32	0-11.29	0-9.84	0-7.80	0-15.87	0-9.46	1-76.81	0-13.31	0-8.47	0-9.33	0-11.98
5	1-47.26	0-9.90	0-7.98	1-51.60	0-10.11	0-6.13	0-14.72	0-7.92	0-8.55	1-53.22	0-8.10	0-11.62	0-9.19
6	0-11.24	0-8.79	0-9.28	0-11.52	0-10.56	0-11.02	2-51.12	0-7.65	0-14.65	0-9.40	0-9.66	0-9.43	1-49.61
7	0-9.14	0-9.85	0-8.63	0-7.88	0-9.73	0-9.05	0-11.88	0-8.90	0-8.65	0-9.90	0-10.12	0-8.21	1-52.33
8	0-9.63	0-9.64	0-8.26	0-11.29	0-10.07	0-8.44	0-11.38	0-7.96	0-8.68	0-13.23	0-7.23	0-9.90	0-13.02
9	0-8.05	0-10.22	0-8.76	2-49.88	0-9.65	2-10.92	0-11.84	0-13.10	0-10.56	0-9.83	0-8.86	0-10.29	1-50.58
10	0-10.04	0-9.68	0-9.43	0-11.80	0-9.11	0-9.94	0-10.22	0-10.56	0-9.91	0-11.97	0-10.59	0-11.67	0-13.91
11	0-8.01	0-9.79	0-7.13	0-11.84	0-7.26	0-6.38	0-10.17	0-9.18	0-9.30	0-7.84	0-9.72	0-8.88	0-12.25
12	0-9.23	0-8.61	0-9.55	0-10.84	0-10.03	0-6.85	2-51.46	0-9.02	0-8.12	0-11.73	0-9.10	0-8.21	0-11.34
13	0-10.08	0-8.76	0-8.46	2-52.15	0-10.04	0-9.73	0-11.71	0-7.77	0-9.89	1-51.61	0-11.36	0-8.19	0-9.43
14	0-9.46	0-9.07	0-10.61	0-14.71	0-9.02	0-11.08	0-9.66	0-13.83	3-48.41	0-10.61	0-7.59	0-7.40	0-9.86
15	0-9.17	0-9.10	0-9.10	0-12.24	0-11.54	0-6.32	2-54.82	0-6.14	0-6.54	2-51.76	0-9.54	0-10.52	0-8.03
16	2-51.69	0-9.66	0-10.63	1-51.39	0-15.83	0-6.58	0-11.81	0-9.49	0-10.00	0-13.28	0-10.29	0-6.80	0-12.25
17	0-9.45	0-10.42	0-8.81	2-50.51	0-9.60	0-8.60	0-10.81	0-8.95	0-8.75	0-11.38	0-8.37	0-10.23	0-12.55
18	0-10.30	0-8.32	0-8.51	0-16.53	0-10.60	0-7.90	1-51.67	0-8.14	0-9.85	0-6.23	0-8.47	1-50.00	1-50.29
19	0-9.24	0-8.31	0-8.62	2-51.40	0-10.07	0-6.47	0-9.30	0-11.38	0-7.33	0-9.33	0-11.49	0-9.77	0-12.29
20	0-10.27	0-9.49	0-9.88	2-49.94	0-26.43	0-9.15	0-15.67	0-10.49	0-11.65	0-11.62	0-11.82	0-10.20	1-51.63

Tabela 5.2: Perdas de rastreo e erro RMS (m) para processos executando no processador i7 3517U.

Tal como foi feito no primeiro experimento, a tabela 5.2 apresenta o número de perdas de rastreo e erro RMS para cada iteração realizada do sistema. Novamente observa-

se que o erro RMS é alto quando existe perda de rastreamento, exceto para os casos em que a perda de rastreamento ocorre no começo do mapeamento.

5.2.1 Discussão

A partir das medições realizadas, pode-se concluir que é possível implementar um sistema de SLAM visual com processamento distribuído, e capaz de operar de forma on-line, quando as máquinas empregadas na extração de pontos de interesse e otimização de poses e posições dos pontos de referência possuem uma capacidade de processamento semelhante e estão conectadas por uma rede alta velocidade. O número de poses estimadas por segundo mostrou-se como sendo função do número de processadores empregados no sistema, mas exibindo um comportamento assintótico horizontalmente, como exibido pela figura 5.13. Este comportamento pode indicar um possível limite inferior para o tempo de estimação da pose, pois a assintota horizontal apresenta um valor próximo do obtido na figura 5.2 do primeiro experimento. Também é notório que, ao contrário do que foi observado no primeiro experimento, o sistema tende a diminuir o tempo para estimação da pose a medida que novos processadores são adicionados. Isto é um indicativo importante de que os resultados ruins do primeiro experimento devem-se, realmente, à saturação de processos competindo pelos núcleos do processador. Quanto ao mapa resultante, o desenho sobreposto e os gráficos dos erros de translação e rotação mostram que a qualidade do mapa e o comportamento das curvas de erro são bastante semelhantes, exibindo um resultado um pouco melhor para alguns casos do que o sistema original.

5.3 Experimento 3

O objetivo do terceiro experimento é avaliar se é possível realizar o SLAM de forma on-line e distribuída quando são empregados processadores de menor desempenho na extração de pontos de interesse e cálculo de descritores do que o responsável pelo ajuste de poses e pontos. Para isso, utiliza-se o computador equipado com o processador i7 7500U para dividir e distribuir as imagens, como também para realizar a remontagem dos blocos de pontos de interesse, descritores e efetuar a associação e otimização das poses e posições dos pontos de referência. Para a execução dos processos responsáveis pela extração dos pontos e cálculo de descritores, optou-se por processadores ARM Cortex-A53 de 1.2 Ghz instanciados em um SoC Broadcom BCM2837. Esta escolha tem por objetivo verificar como o sistema comporta-se ao escalar para um maior número de núcleos disponíveis na rede e como isto impacta nas métricas utilizadas neste trabalho. Nos processadores ARM, são executados até 4 processos (um em cada núcleo), com a imagem sendo dividida de uma até dezesseis partes, variando em 1, 2, 4, 6, 8, 10, 12, 14 e 16, onde a primeira divisão é em duas linhas e as

demais em colunas. O número de núcleos ARM utilizados variaram de um até dezesseis e a comunicação é feita por meio de rede *Ethernet* de 100 Mbps. As métricas empregadas são as mesmas dos experimentos anteriores, começando pelo número de perdas de rastreo. O resultado da primeira métrica é exibido na figura 5.24.

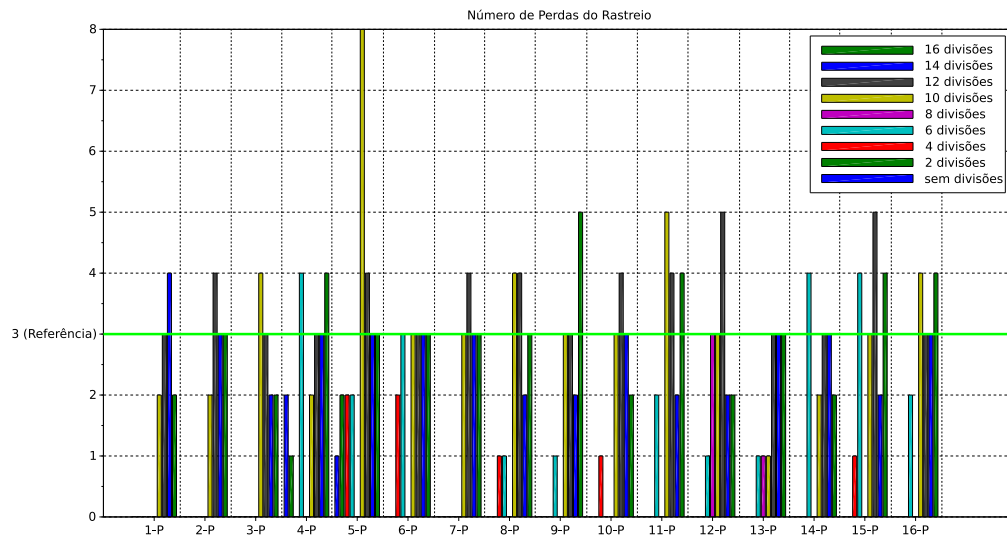


Figura 5.24: Gráfico da perda de rastreo para o sistema executando de forma distribuída em processadores ARM de menor desempenho.

De acordo com a figura 5.24, o sistema executando de forma distribuída apresenta perda de rastreo menor que a do sistema de referência, desde que sejam feitas no máximo 4 divisões na imagem. Este resultado reforça o do segundo experimento, ilustrado na figura 5.12, onde as rodadas que não empregavam divisões das imagens não apresentaram perdas de rastreo.

Tal como nos experimentos anteriores, a segunda métrica é o tempo médio para a estimação da pose e posição dos pontos de referência. O comportamento desta variável pode ser visto nas figuras 5.25 e 5.26.

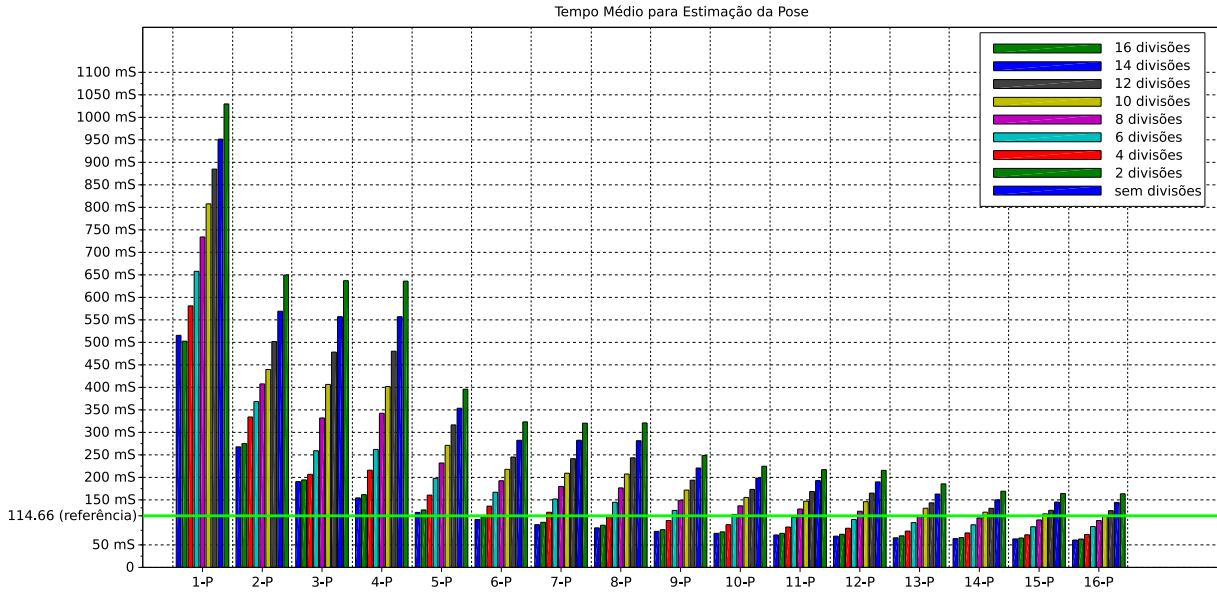


Figura 5.25: Tempo médio para a estimação da pose quando executando o sistema distribuído em vários processadores de menor desempenho.

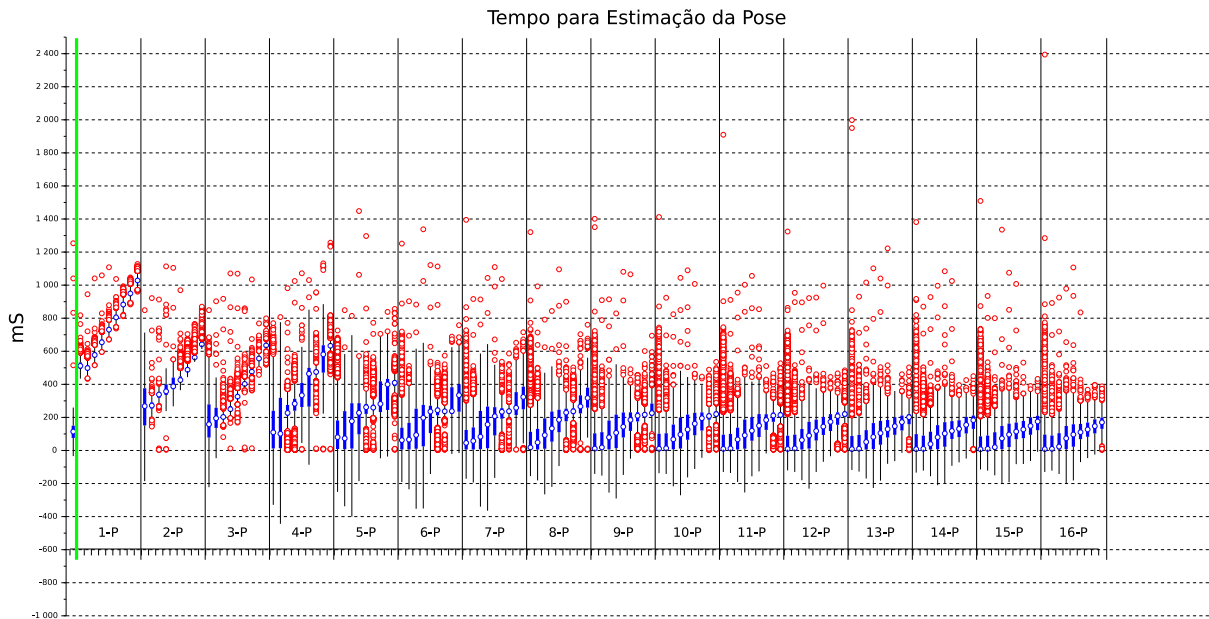


Figura 5.26: Diagrama de caixa do tempo médio para a estimação da pose quando usando o sistema distribuído em vários processadores de menor desempenho.

Na figura 5.25 percebe-se a diminuição do tempo médio com o aumento do número de processos, tal como indicou inicialmente a figura 5.13 do segundo experimento. Dessa forma, também é possível ajustar uma curva da forma $t = \frac{A}{B^p} + C$ aos dados, de maneira a estimar o número de processos necessário para um desempenho desejado. Obtém-se então a equação 5.3 e matriz de covariância Σ (equação 5.4) para o ajuste de uma curva conforme o gráfico da figura 5.27.

$$t = \frac{860,88}{1,96^p} + 75,66 \text{ mS} \quad (5.3)$$

$$\Sigma = \begin{bmatrix} 116579,50 & 268,19 & 3778,89 \\ 268,19 & 0,67 & 16,17 \\ 3778,89 & 16,17 & 1723,11 \end{bmatrix} \quad (5.4)$$

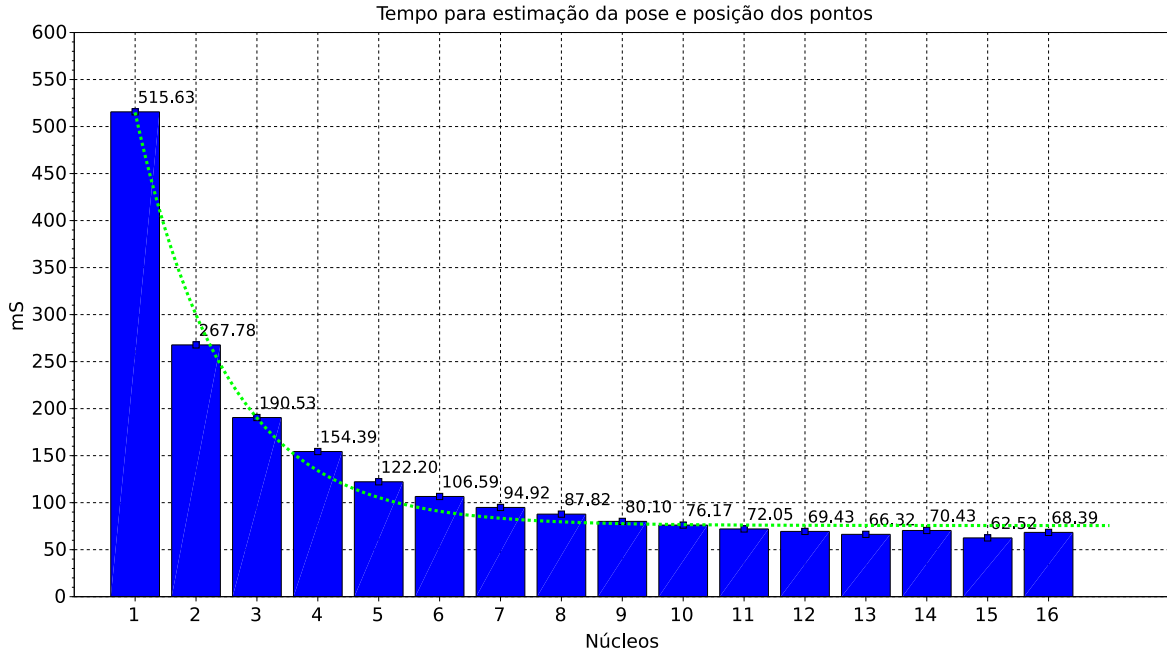


Figura 5.27: Ajuste da curva de tempo médio para estimação da pose e posição dos pontos de referência de acordo com o número de processos executando nos processadores ARM.

Quanto ao comportamento do tempo médio para cada execução do experimento, nota-se a partir da figura 5.26, que o sistema novamente passa a apresentar um maior número de medições individuais distantes da média devido às modificações realizadas no *pipeline* de processamento do ORB-SLAM, mas apresentando um valor médio de tempo por pose menor do que a implementação de referência, a partir do uso de oito núcleos ARM (que equivalem a dois processadores).

Os valores obtidos para a métrica do tempo da imagem no *pipeline* até a obtenção da pose é exibido nas figuras 5.28 e 5.29.

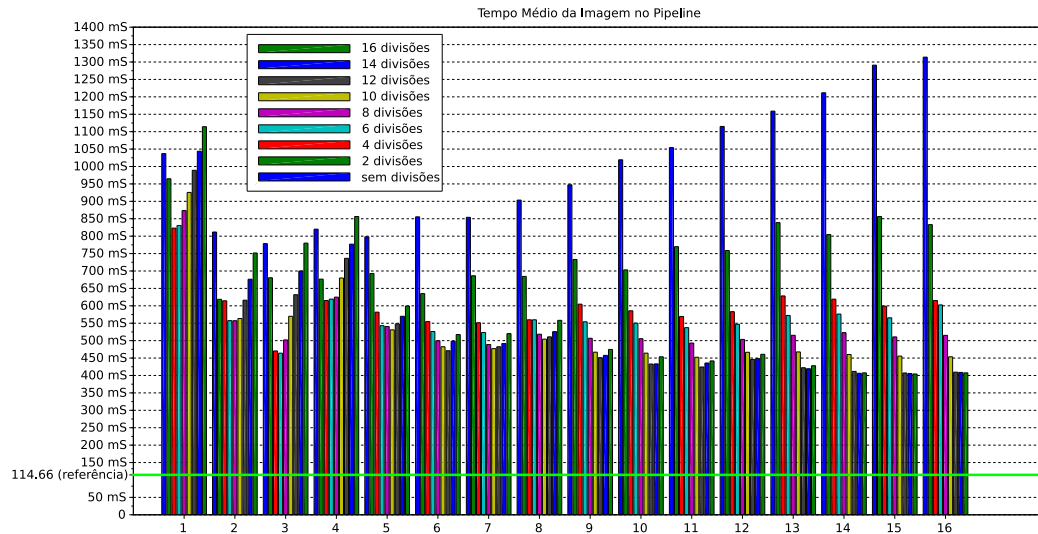


Figura 5.28: Tempo médio de permanência da imagem dentro do *pipeline* até a conclusão da estimação da pose e posição dos pontos de referência (processadores ARM).

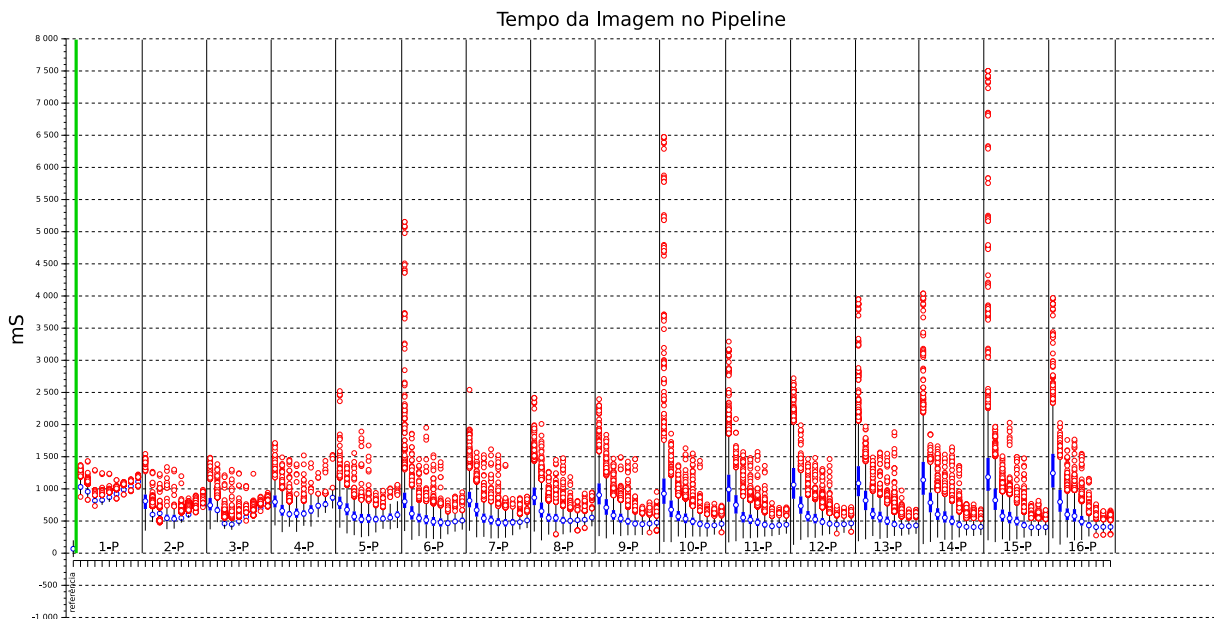


Figura 5.29: Diagrama de caixa do tempo de permanência da imagem dentro do *pipeline* até a conclusão da estimação da pose e posição dos pontos de referência (processadores ARM).

Os gráficos exibidos nas figuras 5.28 e 5.29 apresentam um aumento do tempo da imagem no *pipeline* a medida que se usa mais processadores, mas que diminui quando se aumenta o número de divisões. Nota-se também que este comportamento é mais visível quando se usa um maior número de processos na extração de pontos de interesse e cálculo de descritores. A causa do aumento do tempo da imagem no *pipeline* é a necessidade de estimar a pose na sequência em que as imagens são capturadas, o que força o sistema a aguardar pela conclusão do processamento da sequência de imagens. Já a diminuição do tempo quando se aumenta o número de divisões é causado pelo aumento do paralelismo no processamento de uma imagem específica, fazendo com que um menor número de imagens

diferentes sejam processadas ao mesmo tempo, diminuindo assim a fila de reordenação antes da estimação da pose.

A métrica do uso de CPU pelo processo responsável pelo ajuste de poses e posição dos pontos é apresentada no gráfico da figura 5.30. Nele, nota-se claramente uma grande diminuição do valor médio quando comparado com a implementação de referência. O resultado deste gráfico é semelhante ao da figura 5.18, indicando novamente que pode-se empregar o mesmo processador para realizar o ajuste de várias trajetórias ao mesmo tempo se a tarefa de extração de pontos de interesse e cálculo de descritores for feita em outro conjunto de processadores, podendo assim obter um sistema que é escalável com um menor custo para uma mesma qualidade do mapa final que aquele que executa completamente em uma única máquina de maior desempenho.

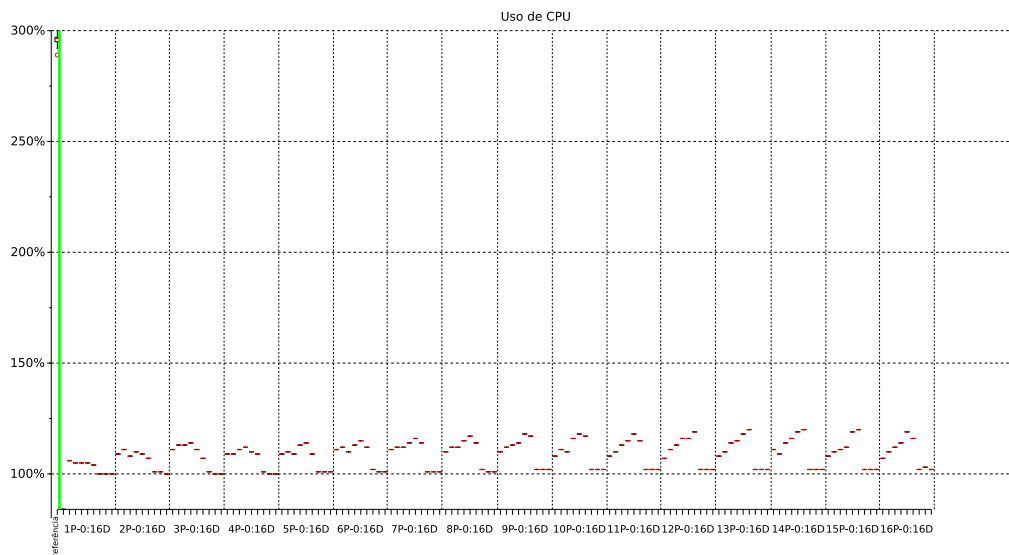


Figura 5.30: Gráfico do uso de CPU para extração de pontos e cálculo de descritores executando nos processadores de menor desempenho.

Quanto ao mapa obtido, ele é apresentado na figura 5.31, contendo a trajetória de posições de referência, a trajetória obtida com a versão original do ORB-SLAM e as trajetórias obtidas com um, dois, três e dezesseis processos executando a extração de pontos de interesse e cálculo de descritores. Percebe-se pela figura que as trajetórias são muito próximas daquela da implementação original, onde os erros existentes são dos mesmos tipos presentes nos mapas dos experimentos anteriores: o erro de translação e o erro de rotação.

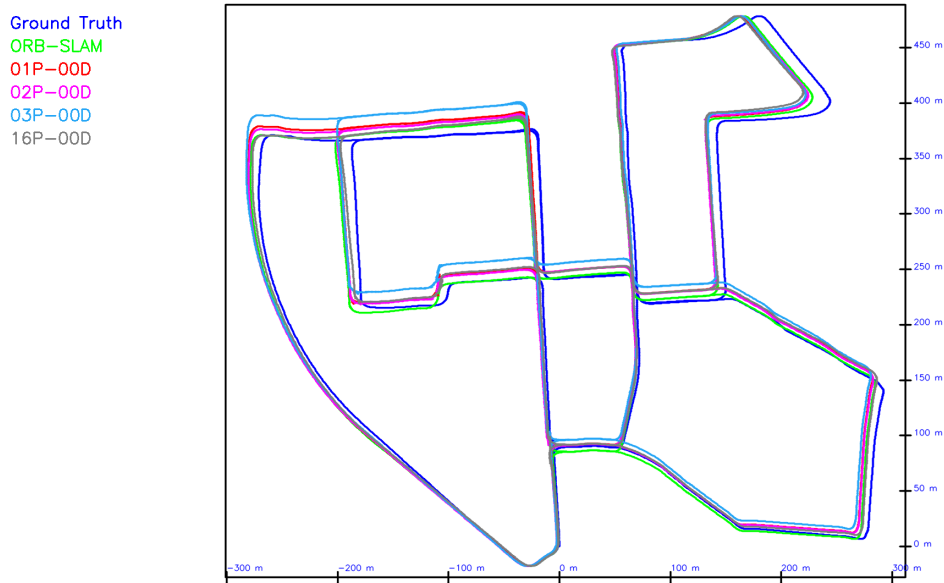


Figura 5.31: Comparação da trajetória obtida para o sistema distribuído executando em processadores de menor desempenho.

Todos os erros mantêm o mesmo padrão daqueles dos dois experimentos anteriores. O erro de translação diminui a medida que a distância percorrida aumenta com a trajetória de forma semelhante ao da versão original do ORB-SLAM, indicando que mesmo com a extração de pontos e cálculo de descritores executando em máquinas de menor desempenho, visualmente não houve uma diminuição da qualidade final do mapa.

Quanto ao erro de translação relacionado à velocidade de deslocamento da câmera, o gráfico mantém o mesmo formato daqueles dos experimentos anteriores, com um intervalo de velocidades onde o erro é menor. O resultado também indica que não houve perda de qualidade notável do mapa relacionada com as modificações, mesmo o sistema executando de forma distribuída em máquinas de menor desempenho. O erro de rotação também não mostra mudanças significativas, mantendo o mesmo padrão dos dois outros gráficos correspondentes dos experimentos anteriores, ou seja, diminuindo a medida que a distância percorrida aumenta e apresentando uma faixa de velocidades de operação onde o erro é menor.

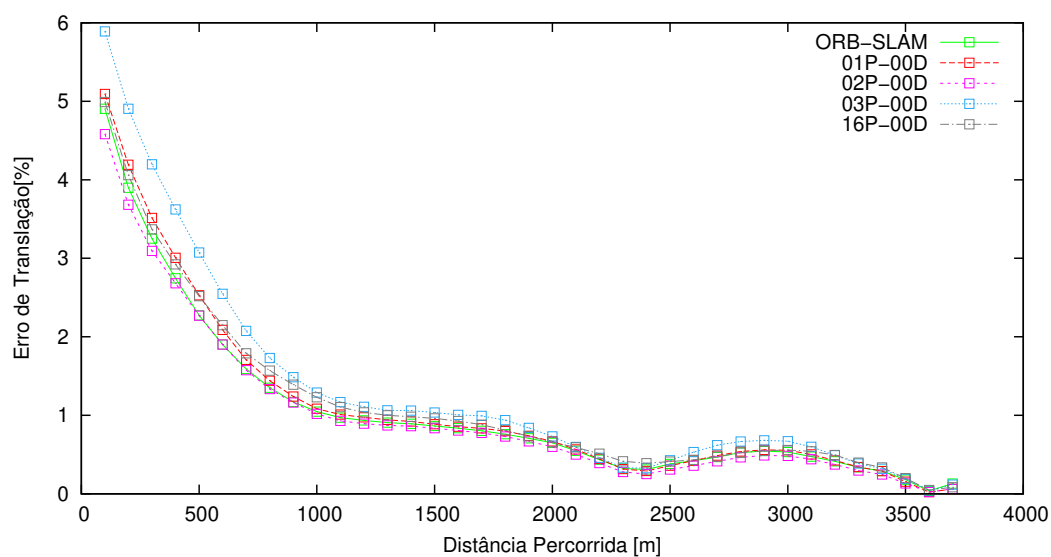


Figura 5.32: Erro de translação percentual contra a distância percorrida.

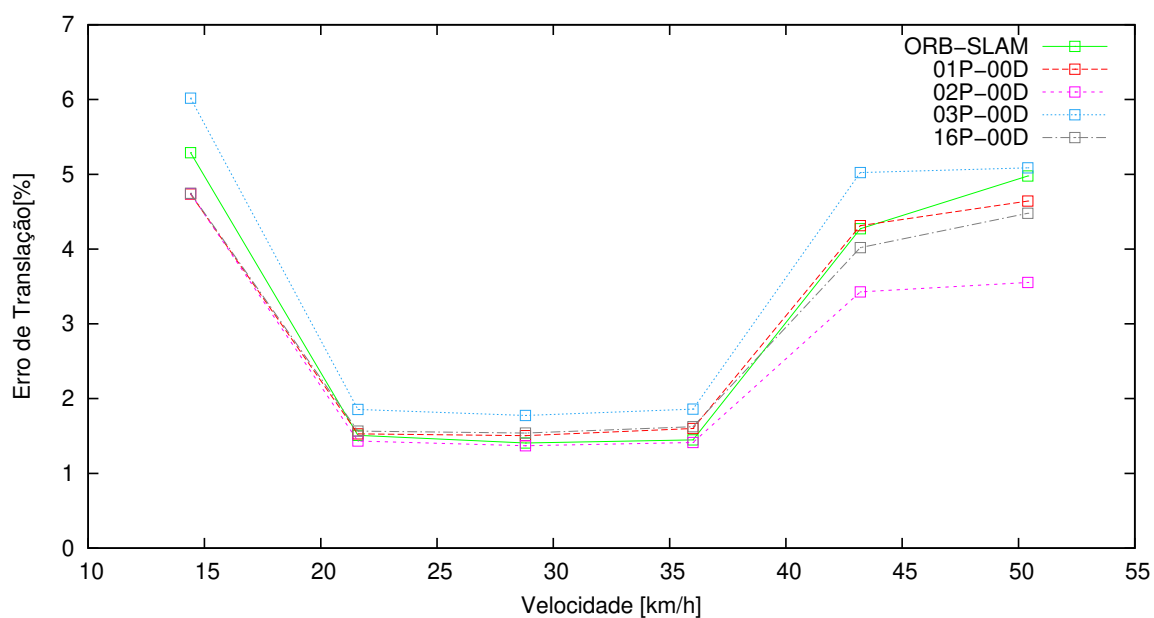


Figura 5.33: Erro de translação percentual contra a velocidade.

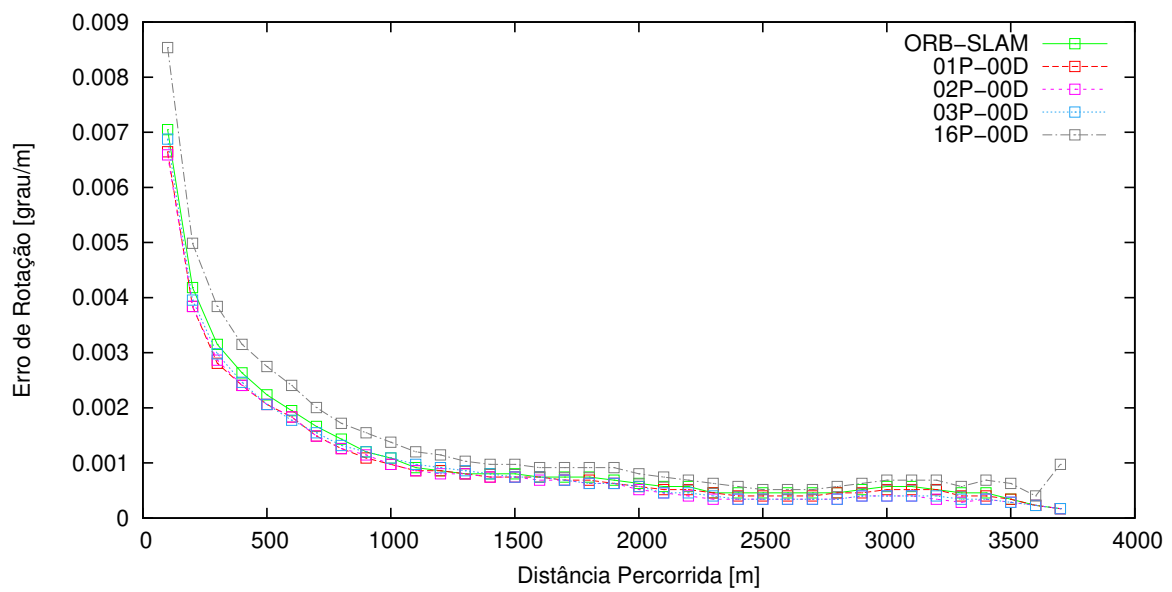


Figura 5.34: Erro de rotação grau/metro para a distância percorrida.

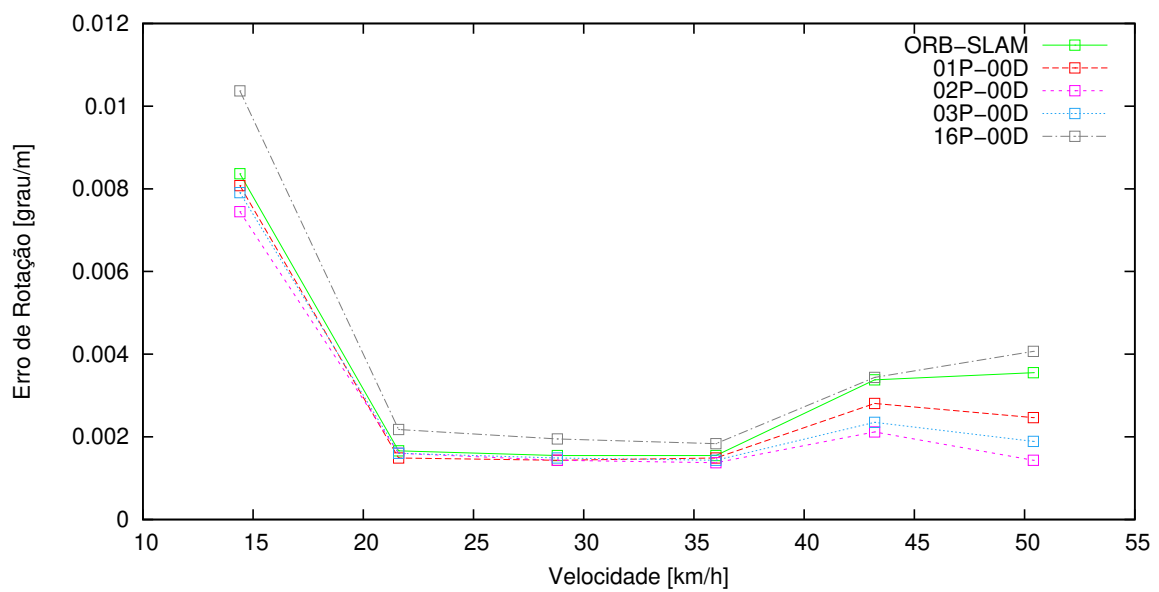


Figura 5.35: Erro de rotação grau/metro contra a velocidade.

A tabela 5.3 apresenta as perdas de rastreo e erro RMS para todas as execuções realizadas para o terceiro experimento. Percebe-se que quando o número de divisões da imagem cresce, as perdas de rastreo aumentam muito, causando um grande aumento no erro RMS. Esse comportamento acontece independente do número de processos utilizados.

0-9.23	1L-1C	2L-1C	2L-2C	2L-3C	2L-4C	2L-5C	2L-6C	2L-7C	2L-8C
1P	0-9.61	0-8.29	0-9.30	0-8.14	0-13.35	2-69.24	3-483.28	4-481.29	2-481.65
2P	0-8.85	0-10.08	0-9.05	0-8.47	0-13.19	2-78.22	4-484.76	3-481.51	3-481.35
3P	0-13.54	0-8.40	0-12.06	0-8.85	0-13.75	4-92.06	3-481.86	2-481.81	2-480.86
4P	2-52.00	1-47.50	0-12.43	4-26.21	0-11.37	2-69.31	3-484.52	3-481.50	4-482.13
5P	1-46.69	2-49.73	2-49.58	2-25.01	0-18.22	8-92.64	4-484.00	3-483.09	3-481.14
6P	0-10.19	0-8.83	2-28.29	3-26.21	0-10.29	3-92.27	3-485.12	3-480.04	3-481.80
7P	0-8.84	0-7.73	0-12.59	0-11.97	0-8.26	3-69.71	4-484.66	3-480.53	3-482.15
8P	0-10.49	0-9.91	1-51.85	1-44.92	0-12.62	4-90.61	4-485.61	2-481.83	3-481.93
9P	0-7.87	0-6.04	0-10.11	1-53.65	0-7.34	3-68.74	3-482.81	2-479.93	5-481.15
10P	0-7.24	0-8.94	1-51.26	0-11.33	0-11.78	3-69.81	4-483.23	3-482.05	2-481.77
11P	0-10.77	0-7.03	0-10.36	2-25.14	0-14.99	5-93.37	4-488.40	2-482.20	4-479.60
12P	0-10.39	0-9.33	0-17.72	1-59.08	3-57.41	3-68.96	5-483.66	2-482.56	2-482.37
13P	0-10.21	0-7.00	0-13.07	1-25.04	1-26.65	1-71.34	3-483.70	3-482.35	3-481.36
14P	0-24.96	0-10.44	0-13.35	4-26.98	0-14.35	2-71.54	3-484.54	3-482.65	2-480.32
15P	0-10.31	0-7.84	1-48.84	4-61.70	0-10.68	3-70.89	5-484.89	2-482.69	4-480.18
16P	0-10.19	0-7.29	0-12.30	2-21.35	0-10.09	4-90.14	3-489.67	3-480.17	4-480.68

Tabela 5.3: Perdas de rastreo e erro RMS (m) para processos executando em processadores de menor desempenho. Nesta tabela o valor para a execução da implementação de referência encontra-se no canto superior esquerdo.

5.3.1 Discussão

A partir dos resultados observados nas métricas do terceiro experimento pode-se concluir que é possível criar um sistema de SLAM distribuído onde são utilizados processadores de arquiteturas e capacidades de processamento diferentes, conforme pode-se ver pelos mapas da figura 5.31. Contudo, existem consequências para o funcionamento de um sistema como este. A primeira é o espalhamento dos tempos individuais para a estimação da pose de cada imagem conforme pode ser visto na figura 5.26. O espalhamento caracteriza-se como uma não uniformidade do tempo entre a estimação das poses. Sendo que, o critério para rotulá-lo como uma característica indesejável deve vir da aplicação, a partir do conhecimento de que ele pode diminuir a “suavidade” com que as imagens são processadas, apresentando-se como variações do tempo entre a estimação das poses, devido ao processamento em fila das imagens. A segunda é o aumento considerável do tempo no *pipeline*, que é o tempo médio que o sistema leva até concluir a estimação da pose da imagem a partir da sua captura. Este pode ser um limitador para o tempo de resposta de uma aplicação em robótica. Assim, estas duas características devem ser levadas em consideração ao se empregar a técnica proposta neste trabalho em sistemas de controle, contudo não adicionam nenhum grande problema quando se deseja apenas o mapa resultante da trajetória percorrida. Além

disso, os resultados indicam que pode-se reservar máquinas com maior desempenho para realizar a tarefa de ajuste de mais de um sistema ao mesmo tempo, enquanto máquinas mais baratas são utilizadas na extração de pontos e cálculo de descritores. Nota-se pelo gráfico da figura 5.25 que com apenas dois processadores ARM já se obtém o mesmo desempenho daquele do sistema executando completamente no processador i7.

5.4 Comentários Gerais

Neste capítulo foram descritas as verificações realizadas sobre como as mudanças propostas neste trabalho impactam o desempenho de um sistema de SLAM visual por métodos indiretos baseado em reconstrução geométrica. A verificação foi feita em três cenários diferentes. No primeiro, o sistema executa completamente em uma única máquina, tal como a versão original do sistema, porém com todo o suporte em *software* para a execução distribuída. No segundo experimento, o código responsável por extrair os pontos de interesse e calcular os descritores executa em um outro processador, de capacidade semelhante ao que realiza o ajuste de poses e pontos no espaço, comunicando-se com este por uma rede local. No terceiro, o código responsável por extrair os pontos de interesse e calcular os descritores executa em processadores de menor desempenho, mas com várias instâncias. No primeiro experimento, observa-se que o resultado é notavelmente melhor que a versão original do sistema quando a métrica é o tempo médio para estimação da pose, desde que a máquina não esteja executando mais processos do sistema que o número de núcleos disponíveis no processador. No segundo experimento, observa-se que o tempo médio para a estimação da pose cai quando o número de processos empregados aumenta, mas que apresenta um comportamento assintótico para o gráfico desta métrica. Quanto ao terceiro experimento, percebe-se também que o resultado para a métrica do tempo médio para estimação da pose possui um comportamento semelhante ao do segundo experimento, com o tempo diminuindo quando o número de processos aumenta. No segundo e terceiro experimentos, o número de perdas de rastreo ficou abaixo daquele da versão original, quando a imagem não foi dividida para todos os números de processos utilizados. Já no primeiro, isto só aconteceu quando um único processo foi utilizado. Este resultado parece indicar uma correlação entre o número de perdas de rastreo e o de processos utilizados na extração dos pontos de interesse e cálculo de descritores.

Quando executando em várias máquinas o sistema mostrou um melhor resultado nas métricas de “perda de rastreo” e “tempo para estimação da pose”, mas com uma piora na métrica de “tempo da imagem no *pipeline*”, caracterizando-se como uma situação de troca da melhora de uma métrica pela piora em outra. Esta característica deve-se ao algoritmo de estimação da pose, que funciona na sequência em que as imagens foram capturadas, armazenando-as em uma fila até que existam sequências ordenadas. Assim, ao se empregar

a técnica, é preciso especificar o ponto de operação desejado para a aplicação e só então determinar o número de processos necessários, de acordo com os gráficos e equações das figuras 5.15 e 5.27. A outra mudança nos resultados ocorre como o espalhamento dos tempos médios para a estimação da pose e da imagem no *pipeline*. Novamente causada pelo armazenamento dos dados em uma fila.

Quanto a qualidade final do mapa, este manteve valor semelhante àquela vista na versão original do OBR-SLAM. Sendo que em todos os casos, os gráficos de erro apresentaram o mesmo comportamento da versão original. Este é um resultado esperado, já que as mudanças tem por objetivo melhorar os tempos de execução e não os mapas das trajetórias obtidas. Nas situações em que isto ocorre, a justificativa para o resultado é a mudança da heurística de adição de *keyframes*.

Para os testes em que as imagens foram divididas, não houve uma melhora nos resultados do sistema. Isto foi causado pela diminuição de pontos de interesse nesses casos pois aqueles que estão muito próximos às bordas são perdidos quando a imagem é dividida. Para que a divisão das imagens adicione alguma vantagem é preciso adicionar uma borda de intersecção e um algoritmo para eliminar os pontos de interesse repetidos que aparecem em mais de uma divisão da imagem.

CAPÍTULO 6

Conclusões

Neste trabalho, foi verificada a hipótese de aumento do número de poses da câmera e posição dos pontos no espaço, estimados por segundo, por meio da modificação do paradigma básico dos sistemas de SLAM visual baseados em métodos indiretos e reconstrução geométrica. Após a conclusão dos experimentos, verificou-se que a hipótese é verdadeira quando o número de processos utilizados é menor ou igual ao número de núcleos de processamento disponíveis, sejam eles locais ou remotos. O sistema construído para verificação da hipótese mostra-se funcional e de desempenho crescente com o número de processadores, de acordo com as métricas usadas. Contudo, observa-se a existência de um limite assintótico para o desempenho máximo. Além deste limite não se percebe benefícios em aumentar o número de processos no sistema. Verifica-se também que, existem alguns efeitos indesejados ao aplicar a técnica proposta, tal como o aumento do tempo da imagem no *pipeline* em relação a implementação sem modificações. Além disso, os tempos de estimação para cada pose apresentaram um maior espalhamento de tempo médio em relação ao existente no sistema original.

A causa do espalhamento nos tempos está na lógica empregada no algoritmo de reconstrução dos blocos de pontos de interesse e descritores. Este faz a estimação da pose na sequência em que as imagens foram capturadas, não importando a sequência de recebimento. Dessa forma, o sistema sempre espera por dados de uma imagem antes de estimar a pose das que vem depois dela. Esta é uma característica que deve ser levada em consideração de acordo com a aplicação.

Quando o sistema utiliza grupos de computadores de menor desempenho na tarefa da extração de pontos de interesse e cálculo de descritores, verifica-se que o mapa resultante apresenta a mesma qualidade daquele criado quando se emprega apenas máquinas de maior desempenho e do mesmo tipo. Este resultado indica que a abordagem pode ser usada em um sistema escalável de baixo custo, e que emprega máquinas de diferentes arquiteturas. Neste

caso, a única métrica que apresentou uma piora foi a do tempo da imagem no *pipeline*. O que é justificável, já que nesta situação os dados precisam trafegar pela rede.

Ao utilizar o sistema de forma distribuída, em máquinas com desempenho aproximado, percebe-se que ele apresenta como diferença apenas a diminuição do tempo da imagem no *pipeline*, quando comparado à situação onde se usa máquinas de menor desempenho. No caso em que executou completamente na mesma máquina, ele apresentou desempenho superior à versão original seguido por um aumento pequeno do tempo no *pipeline* quando comparado com a versão não modificada.

A divisão das imagens não apresenta-se como uma boa estratégia como mostra a métrica do número de perdas de rastreo. Como dito no texto, isto ocorre devido a perda dos pontos de interesse que se encontram sobre a borda da divisão da imagem. Assim, um uso desta abordagem onde se deseja diminuir o número de perdas deve levar isto em consideração utilizando, por exemplo, uma borda repetida nas imagens divididas.

Quanto às trajetórias registradas, nota-se que elas não se sobrepõem à versão original do ORB-SLAM. Isto é causado, possivelmente, pela mudança na lógica de inserção de *keyframes*. Uma abordagem para a solução pode ser algo como a otimização por janela dupla, como a proposta por (STRASDAT et al., 2011) no ScaViSLAM. Também é possível explorar o tempo que o sistema de otimização de poses e pontos permanece ocioso devido à demora que pode ocorrer de forma aleatória na entrega dos dados pela rede. Para isso, pode-se empregar uma segunda janela de otimização que insere mais *keyframes* no processo. Esta pode ser uma boa estratégia para melhorar os resultados que deve ser explorada em trabalhos futuros.

Ao considerar o emprego da técnica na criação de um *software* de SLAM para operação na nuvem, deve-se levar em consideração que este trabalho não analisa de nenhuma maneira os tempos necessários para a transmissão das imagens. Dessa forma, as conclusões são válidas apenas para o processamento das imagens, considerando que o problema de transmissão foi resolvido por algum outro meio. A partir dos resultados do segundo e terceiro experimentos, pode-se concluir que o *software* de SLAM pode apresentar um melhor desempenho nos tempos necessários para estimação, desde que existam recursos para processamento paralelo em diversas máquinas. Este é um cenário animador porque a disponibilidade deste tipo de recurso é uma das características dos sistemas modernos de infraestrutura em nuvem.

Por fim, após a conclusão dos experimentos, parece que a heurística usada na adição de *keyframes* é muito restritiva, pois ela ignora completamente a existência de intervalo de tempo entre as imagens. Isto pode deixar o sistema mais robusto, como verificou-se pela diminuição do número de perdas de rastreo, mas desperdiça intervalos de tempo que podem ser usados para melhorar a otimização do mapa. Nos trabalhos futuros pretende-se reim-

plementar as heurísticas de adição de *keyframes* e o mecanismo de ajuste de poses e pontos levando estas observações em consideração.

Referências Bibliográficas

- ABSIL, P.-A.; MAHONY, R.; SEPULCHRE, R. *Optimization Algorithms on Matrix Manifolds*. Princeton, NJ, USA: Princeton University Press, 2007. ISBN 0691132984, 9780691132983.
- ACHARYA, K. A.; BABU, R. V.; VADHIYAR, S. S. A real-time implementation of sift using gpu. *Journal of Real-Time Image Processing*, p. 1–11, 2014. ISSN 1861-8219. Disponível em: <<http://dx.doi.org/10.1007/s11554-014-0446-6>>.
- AGUILAR-GONZÁLEZ, A.; ARIAS-ESTRADA, M. Towards a smart camera for monocular slam. In: *Proceedings of the 10th International Conference on Distributed Smart Camera*. New York, NY, USA: ACM, 2016. (ICDSC '16), p. 128–135. ISBN 978-1-4503-4786-0. Disponível em: <<http://doi.acm.org/10.1145/2967413.2967441>>.
- ALCANTARILLA JESUS NUEVO, A. B. P. Fast explicit diffusion for accelerated features in nonlinear scale spaces. In: *Proceedings of the British Machine Vision Conference*. [S.l.]: BMVA Press, 2013.
- BAY, H. et al. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, v. 110, n. 3, p. 346 – 359, 2008. ISSN 1077-3142. Similarity Matching in Computer Vision and Multimedia. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S1077314207001555>>.
- BLACKWELL, D. Conditional expectation and unbiased sequential estimation. *Ann. Math. Statist.*, The Institute of Mathematical Statistics, v. 18, n. 1, p. 105–110, 03 1947. Disponível em: <<http://dx.doi.org/10.1214/aoms/1177730497>>.
- BLANCO, J. L. *A tutorial on SE(3) transformation parameterizations and on-manifold optimization*. 2014.
- BOOTHBY, W. *An Introduction to Differentiable Manifolds and Riemannian Geometry*. Elsevier Science, 1975. (Pure and Applied Mathematics). ISBN 9780080873794. Disponível em: <<https://books.google.com.br/books?id=oWLJ6zaSwyIC>>.
- BROWN, D. C. *A solution to the general problem of multiple station analytical stereo triangulation*. Patrick Airforce Base, Florida, 1958.
- CALONDER, M. et al. Brief: Binary robust independent elementary features. In: *Proceedings of the 11th European Conference on Computer Vision: Part IV*. Berlin, Heidelberg: Springer-Verlag, 2010. (ECCV'10), p. 778–792. ISBN 3-642-15560-X,

- 978-3-642-15560-4. Disponível em: <<http://dl.acm.org/citation.cfm?id=1888089.1888148>>.
- CAO, J.; CHEN, T.; FAN, J. Landmark recognition with compact bow histogram and ensemble elm. *Multimedia Tools and Applications*, v. 75, n. 5, p. 2839–2857, 2016. ISSN 1573-7721. Disponível em: <<http://dx.doi.org/10.1007/s11042-014-2424-1>>.
- CHIUSO, A. et al. Structure from motion causally integrated over time. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 24, n. 4, p. 523–535, Apr 2002. ISSN 0162-8828.
- CIVERA, J.; DAVISON, A. J.; MONTIEL, J. M. M. Inverse depth parametrization for monocular slam. *IEEE Transactions on Robotics*, v. 24, n. 5, p. 932–945, Oct 2008. ISSN 1552-3098.
- CSORBA JEFFREY K. UHLMANN, H. F. D.-W. M. *New approach to simultaneous localization and dynamic map building*. 1996. 2738 - 2738 - 11 p. Disponível em: <<https://doi.org/10.1117/12.241084>>.
- CSORBA, M. *Simultaneous Localisation and Map Building*. Tese (Doutorado) — University of Oxford, 1997.
- CSURKA, G. et al. Visual categorization with bags of keypoints. In: *In Workshop on Statistical Learning in Computer Vision, ECCV*. [S.l.: s.n.], 2004. p. 1–22.
- DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. [S.l.: s.n.], 2005. v. 1, p. 886–893 vol. 1. ISSN 1063-6919.
- DAVISON, A. J. Real-time simultaneous localisation and mapping with a single camera. In: *Proceedings Ninth IEEE International Conference on Computer Vision*. [S.l.: s.n.], 2003. p. 1403–1410 vol.2.
- DAVISON, A. J. et al. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 29, n. 6, p. 1052–1067, June 2007. ISSN 0162-8828.
- DELLAERT, F. *Visual SLAM Tutorial: Bundle Adjustment*. [S.l.], 2014. Disponível em: <<https://pdfs.semanticscholar.org/6081/417b95bec070fb842a704044def427f8ef69.pdf>>.
- DHIMAN, N. K.; DEODHARE, D.; KHEMANI, D. Where am i? creating spatial awareness in unmanned ground robots using slam: A survey. *Sadhana*, v. 40, n. 5, p. 1385–1433, 2015. ISSN 0973-7677. Disponível em: <<http://dx.doi.org/10.1007/s12046-015-0402-6>>.
- DIEBEL, J. *Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors*. 2006.
- DOUCET, A. et al. Rao-blackwellised particle filtering for dynamic bayesian networks. In: *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.,

2000. (UAI '00), p. 176–183. ISBN 1-55860-709-9. Disponível em: <<http://dl.acm.org/citation.cfm?id=647234.720075>>.
- DURRANT-WHYTE, H.; BAILEY, T. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, v. 13, n. 2, p. 99–110, June 2006. ISSN 1070-9932.
- DURRANT-WHYTE, H.; RYE, D.; NEBOT, E. Localization of autonomous guided vehicles. In: GIRALT, G.; HIRZINGER, G. (Ed.). *Robotics Research*. London: Springer London, 1996. p. 613–625. ISBN 978-1-4471-0765-1.
- EADE, E.; DRUMMOND, T. Scalable monocular slam. In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. [S.l.: s.n.], 2006. v. 1, p. 469–476. ISSN 1063-6919.
- ENGEL, J.; KOLTUN, V.; CREMERS, D. Direct sparse odometry. Apr 2017.
- ENGEL, J.; SCHÖPS, T.; CREMERS, D. LSD-SLAM: Large-scale direct monocular SLAM. In: . [S.l.: s.n.], 2014.
- ENGEL, J.; STURM, J.; CREMERS, D. Semi-dense visual odometry for a monocular camera. In: . Sydney, Australia: [s.n.], 2013.
- FEI-FEI, L.; PERONA, P. A bayesian hierarchical model for learning natural scene categories. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. [S.l.: s.n.], 2005. v. 2, p. 524–531 vol. 2. ISSN 1063-6919.
- FERNÁNDEZ-MADRIGAL, J.-A.; CLARACO, J. L. B. *Simultaneous Localization and Mapping for Mobile Robots: Introduction and Methods*. 1st. ed. Hershey, PA, USA: IGI Global, 2012. ISBN 1466621044, 9781466621046.
- FISCHLER, M. A.; BOLLES, R. C. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, ACM, New York, NY, USA, v. 24, n. 6, p. 381–395, jun. 1981. ISSN 0001-0782. Disponível em: <<http://doi.acm.org/10.1145/358669.358692>>.
- FORSTER, C.; PIZZOLI, M.; SCARAMUZZA, D. SVO: Fast semi-direct monocular visual odometry. In: *IEEE International Conference on Robotics and Automation (ICRA)*. [S.l.: s.n.], 2014.
- FRESE, U.; SCHRODER, L. Closing a million-landmarks loop. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. [S.l.: s.n.], 2006. p. 5032–5039. ISSN 2153-0858.
- FUENTES-PACHECO, J.; RUIZ-ASCENCIO, J.; RENDÓN-MANCHA, J. M. Visual simultaneous localization and mapping: a survey. *Artificial Intelligence Review*, v. 43, n. 1, p. 55–81, 2015. ISSN 1573-7462. Disponível em: <<http://dx.doi.org/10.1007/s10462-012-9365-8>>.
- GABRIEL, E. et al. Open MPI: Goals, concept, and design of a next generation MPI implementation. In: *Proceedings, 11th European PVM/MPI Users' Group Meeting*. Budapest, Hungary: [s.n.], 2004. p. 97–104.

- GALVEZ-LÓPEZ, D.; TARDOS, J. D. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, v. 28, n. 5, p. 1188–1197, Oct 2012. ISSN 1552-3098.
- GRISSETTI, G. et al. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, v. 2, n. 4, p. 31–43, winter 2010. ISSN 1939-1390.
- GUCLU, O.; CAN, A. B. Fast and effective loop closure detection to improve slam performance. *Journal of Intelligent & Robotic Systems*, Oct 2017. ISSN 1573-0409. Disponível em: <<https://doi.org/10.1007/s10846-017-0718-z>>.
- GUIBAS, L. J.; SEDGEWICK, R. A dichromatic framework for balanced trees. In: *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 1978. (SFCS '78), p. 8–21. Disponível em: <<http://dx.doi.org/10.1109/SFCS.1978.3>>.
- HARTLEY, R. I. In defense of the eight-point algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 19, n. 6, p. 580–593, Jun 1997. ISSN 0162-8828.
- HARTLEY, R. I.; ZISSERMAN, A. *Multiple View Geometry in Computer Vision*. Second. [S.l.]: Cambridge University Press, ISBN: 0521540518, 2004.
- HELMKE, U. et al. Essential matrix estimation using gauss-newton iterations on a manifold. *International Journal of Computer Vision*, v. 74, n. 2, p. 117–136, Aug 2007. ISSN 1573-1405. Disponível em: <<https://doi.org/10.1007/s11263-006-0005-0>>.
- ITSEEZ. *Open Source Computer Vision Library*. 2015. <<https://github.com/itseez/opencv>>.
- JIN, H.; FAVARO, P.; SOATTO, S. A semi-direct approach to structure from motion. *The Visual Computer*, v. 19, n. 6, p. 377–394, Oct 2003. ISSN 1432-2315. Disponível em: <<https://doi.org/10.1007/s00371-003-0202-6>>.
- KARLSSON, N. et al. The vslam algorithm for robust localization and mapping. In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. [S.l.: s.n.], 2005. p. 24–29. ISSN 1050-4729.
- KIM, J. et al. A fast feature extraction in object recognition using parallel processing on cpu and gpu. In: *2009 IEEE International Conference on Systems, Man and Cybernetics*. [S.l.: s.n.], 2009. p. 3842–3847. ISSN 1062-922X.
- KITTI. *The KITTI Vision Benchmark Suite*. 2016. <<http://www.cvlibs.net/datasets/kitti/>>. Accessed: 2016-07-22. Disponível em: <<http://www.cvlibs.net/datasets/kitti/>>.
- KLEIN, G.; MURRAY, D. Parallel tracking and mapping for small ar workspaces. In: *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*. [S.l.: s.n.], 2007. p. 225–234.
- KUMAR, V. *Introduction to Parallel Computing*. 2nd. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0201648652.

- KÜMMERLE, R. et al. G2o: A general framework for graph optimization. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. [S.l.: s.n.], 2011. p. 3607–3613. ISSN 1050-4729.
- LATEGAHN, H.; GEIGER, A.; KITT, B. Visual slam for autonomous ground vehicles. In: *2011 IEEE International Conference on Robotics and Automation*. [S.l.: s.n.], 2011. p. 1732–1737. ISSN 1050-4729.
- LEE, P. Y.; MOORE, J. B. Pose estimation via gauss-newton-on-manifold. In: . [S.l.: s.n.], 2004.
- LEUTENEGGER, S.; CHLI, M.; SIEGWART, R. Y. Brisk: Binary robust invariant scalable keypoints. In: *2011 International Conference on Computer Vision*. [S.l.: s.n.], 2011. p. 2548–2555. ISSN 1550-5499.
- LOWE, D. G. Object recognition from local scale-invariant features. In: *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*. [S.l.: s.n.], 1999. v. 2, p. 1150–1157 vol.2.
- LOWE, D. G. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, v. 60, n. 2, p. 91–110, 2004. ISSN 1573-1405. Disponível em: <<http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94>>.
- LU, F.; MILIOS, E. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, v. 4, n. 4, p. 333–349, 1997. ISSN 1573-7527. Disponível em: <<http://dx.doi.org/10.1023/A:1008854305733>>.
- MA, Y.; KOSECKÁ, J.; SASTRY, S. Optimization criteria and geometric algorithms for motion and structure estimation. *International Journal of Computer Vision*, v. 44, p. 219–249, 1999.
- MA, Y. et al. *An Invitation to 3-D Vision*. [S.l.]: Springer, 2004.
- MATARIĆ, M. *The Robotics Primer*. Cambridge University Press, 2007. (Intelligent robotics and autonomous agents). ISBN 9780262633543. Disponível em: <<https://books.google.com.br/books?id=WWJPjgz-jgEC>>.
- MONTEMERLO, M. et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. In: *Eighteenth National Conference on Artificial Intelligence*. Menlo Park, CA, USA: American Association for Artificial Intelligence, 2002. p. 593–598. ISBN 0-262-51129-0. Disponível em: <<http://dl.acm.org/citation.cfm?id=777092.777184>>.
- MONTEMERLO, M. et al. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In: *IJCAI. Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*. Acapulco, Mexico, 2003.
- MOURAGNON, E. et al. Real time localization and 3d reconstruction. In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. [S.l.: s.n.], 2006. v. 1, p. 363–370. ISSN 1063-6919.

- MOUTARLIER, P.; CHATILA, R. Stochastic multisensory data fusion for mobile robot location and environment modeling. In: *International Symposium of Robotics Research*. [S.l.: s.n.], 1989.
- MOUTARLIER, P.; CHATILA, R. An experimental system for incremental environment modelling by an autonomous mobile robot. In: HAYWARD, V.; KHATIB, O. (Ed.). *Experimental Robotics I*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990. p. 327–346. ISBN 978-3-540-46917-9.
- MUR-ARTAL, R. *Real-Time SLAM for Monocular, Stereo and RGB-D Cameras, with Loop Detection and Relocalization Capabilities*. 2015. Disponível em: <https://github.com/raulmur/ORB_SLAM2>.
- MUR-ARTAL, R.; MONTIEL, J. M. M.; TARDÓS, J. D. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, v. 31, n. 5, p. 1147–1163, Oct 2015. ISSN 1552-3098.
- MUR-ARTAL, R.; TARDÓS, J. D. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, v. 33, n. 5, p. 1255–1262, 2017.
- NETO, O. de A. R. *Fix a bug in DistributOctTree algorithm of ORBextractor*. 2018. Disponível em: <https://github.com/raulmur/ORB_SLAM2/pull/600>.
- NEWCOMBE, R. A.; LOVEGROVE, S. J.; DAVISON, A. J. Dtam: Dense tracking and mapping in real-time. In: *2011 International Conference on Computer Vision*. [S.l.: s.n.], 2011. p. 2320–2327. ISSN 1550-5499.
- NISTER, D. An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 26, n. 6, p. 756–770, June 2004. ISSN 0162-8828.
- NISTER, D.; NARODITSKY, O.; BERGEN, J. Visual odometry. In: *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*. [S.l.: s.n.], 2004. v. 1, p. I–652–I–659 Vol.1. ISSN 1063-6919.
- NISTER, D.; STEWENIUS, H. Scalable recognition with a vocabulary tree. In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. [S.l.: s.n.], 2006. v. 2, p. 2161–2168. ISSN 1063-6919.
- PIRE, T. et al. S-PTAM: Stereo Parallel Tracking and Mapping. *Robotics and Autonomous Systems (RAS)*, v. 93, p. 27 – 42, 2017. ISSN 0921-8890.
- PIRE, T. et al. Stereo Parallel Tracking and Mapping for robot localization. In: *Proc. of the International Conference on Intelligent Robots and Systems (IROS)*. [S.l.: s.n.], 2015. p. 1373–1378.
- PIZZOLI, M.; FORSTER, C.; SCARAMUZZA, D. Remode: Probabilistic, monocular dense reconstruction in real time. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. [S.l.: s.n.], 2014. p. 2609–2616. ISSN 1050-4729.

- PRINCE, S. *Computer Vision: Models Learning and Inference*. [S.l.]: Cambridge University Press, 2012.
- QU, Y. et al. Evaluation of local features and classifiers in bow model for image classification. *Multimedia Tools and Applications*, v. 70, n. 2, p. 605–624, 2014. ISSN 1573-7721. Disponível em: <<http://dx.doi.org/10.1007/s11042-012-1107-z>>.
- RAO, C. R. Information and accuracy attainable in the estimation of statistical parameters. *Bull Calcutta. Math. Soc.*, v. 37, p. 81–91, 1945. Disponível em: <<http://ci.nii.ac.jp/naid/10006656681/en/>>.
- ROLAND, S.; REZA, N. I.; DAVIDE, S. *Introduction to autonomous mobile robots*. [S.l.]: MIT, 2011. Intelligent robotics and autonomous agents.
- ROSTEN, E.; PORTER, R.; DRUMMOND, T. Faster and better: A machine learning approach to corner detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 32, n. 1, p. 105–119, Jan 2010. ISSN 0162-8828.
- RUBLEE, E. et al. Orb: An efficient alternative to sift or surf. In: *2011 International Conference on Computer Vision*. [S.l.: s.n.], 2011. p. 2564–2571. ISSN 1550-5499.
- SCARAMUZZA, D.; FRAUNDORFER, F. Visual odometry [tutorial]. *IEEE Robotics Automation Magazine*, v. 18, n. 4, p. 80–92, Dec 2011. ISSN 1070-9932.
- SHARMA, K. High performance gpu based optimized feature matching for computer vision applications. *Optik - International Journal for Light and Electron Optics*, v. 127, n. 3, p. 1153 – 1159, 2016. ISSN 0030-4026. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0030402615015892>>.
- SILVA, B.; BURLAMAQUI, A.; GONÇALVES, L. On monocular visual odometry for indoor ground vehicles. In: *2012 Brazilian Robotics Symposium and Latin American Robotics Symposium*. [S.l.: s.n.], 2012. p. 220–225.
- SINHA, N. et al. Gpu-based video feature tracking and matching. 01 2006.
- SIVIC, J. et al. Discovering objects and their localization in images. In: *ICCV*. [S.l.]: IEEE Computer Society, 2005. p. 370–377.
- SIVIC, J.; ZISSERMAN, A. Video google: a text retrieval approach to object matching in videos. In: *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*. [S.l.: s.n.], 2003. p. 1470–1477 vol.2.
- SMITH, R.; SELF, M.; CHEESEMAN, P. Estimating uncertain spatial relationships in robotics. In: _____. *Autonomous Robot Vehicles*. New York, NY: Springer New York, 1990. p. 167–193. ISBN 978-1-4613-8997-2. Disponível em: <https://doi.org/10.1007/978-1-4613-8997-2_14>.
- SMITH, R. C.; CHEESEMAN, P. On the representation and estimation of spatial uncertainty. *Int. J. Rob. Res.*, Sage Publications, Inc., Thousand Oaks, CA, USA, v. 5, n. 4, p. 56–68, dez. 1986. ISSN 0278-3649. Disponível em: <<http://dx.doi.org/10.1177/027836498600500404>>.

- STRASDAT, H. et al. Double window optimisation for constant time visual slam. In: *2011 International Conference on Computer Vision*. [S.l.: s.n.], 2011. p. 2352–2359. ISSN 1550-5499.
- STRASDAT, H.; MONTIEL, J.; DAVISON, A. J. Visual slam: Why filter? *Image and Vision Computing*, v. 30, n. 2, p. 65 – 77, 2012. ISSN 0262-8856. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0262885612000248>>.
- TAKETOMI, T.; UCHIYAMA, H.; IKEDA, S. Visual slam algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, v. 9, n. 1, p. 16, Jun 2017. ISSN 1882-6695. Disponível em: <<https://doi.org/10.1186/s41074-017-0027-2>>.
- THRUN, S.; BURGARD, W.; FOX, D. *Probabilistic robotics*. Cambridge (Mass.) (London): the MIT Press, 2005. (Intelligent robotics and autonomous agents). ISBN 0-262-20162-3. Disponível em: <<http://opac.inria.fr/record=b1102986>>.
- TRIGGS, B. et al. Bundle adjustment — a modern synthesis. In: TRIGGS, B.; ZISSERMAN, A.; SZELISKI, R. (Ed.). *Vision Algorithms: Theory and Practice*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000. p. 298–372. ISBN 978-3-540-44480-0.
- WAN, J. et al. Cloud robotics: Current status and open issues. *IEEE Access*, v. 4, p. 2797–2807, 2016. ISSN 2169-3536.
- WILLIAMS, B. et al. A comparison of loop closing techniques in monocular {SLAM}. *Robotics and Autonomous Systems*, v. 57, n. 12, p. 1188 – 1197, 2009. ISSN 0921-8890. Inside Data Association. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0921889009000876>>.
- YANG, N. et al. Challenges in monocular visual odometry: Photometric calibration, motion bias, and rolling shutter effect. *IEEE Robotics and Automation Letters*, v. 3, n. 4, p. 2878–2885, Oct 2018.
- YANG, X.; CHENG, K.-T. Ldb: An ultra-fast feature for scalable augmented reality on mobile devices. In: *2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. [S.l.: s.n.], 2012. p. 49–57.

APÊNDICE A

Algoritmos

Algoritmo A.1: Estima rotação e translação

Entrada: F_1, F_0 ; Quadros contento a pose, conjunto de pontos de interesse e descritores dos pontos

Entrada: A ; Conjunto com a associação de pontos entre duas imagens

Entrada: Λ ; Matriz de parâmetros intrínsecos da câmera

Saída: Ω, τ ; Componentes da matriz $\in SE(3)$ que define a transformação entre poses

```

1  $W \leftarrow \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix};$ 
2  $L' \leftarrow \text{diagonal}[1, 1, 0];$ 
3  $F \leftarrow \text{oito pontos}(F_1, F_0, A, \Lambda);$ 
4 se  $F \neq \text{singular}$  então
5    $A \leftarrow \text{refina a associação}(F_1, F_0, A);$ 
6    $F \leftarrow \text{oito pontos}(F_1, F_0, A, \Lambda);$ 
7    $E \leftarrow \Lambda^T F \Lambda;$ 
8    $ULV^T \leftarrow \text{SVD}(E);$ 
9    $\tau_{\times 1} \leftarrow UL'WU^T;$ 
10   $\tau_{\times 2} \leftarrow -\tau_{\times 1};$ 
11   $\Omega_1 \leftarrow UW^{-1}V^T;$ 
12   $\Omega_2 \leftarrow UWV^T;$ 
13   $\omega_1 \leftarrow 3D(\begin{bmatrix} I_{3 \times 3} & 0 \end{bmatrix}, \begin{bmatrix} \Omega_1 & \tau_1 \end{bmatrix}, x_1, x_0);$ 
14   $\omega_2 \leftarrow 3D(\begin{bmatrix} I_{3 \times 3} & 0 \end{bmatrix}, \begin{bmatrix} \Omega_1 & \tau_2 \end{bmatrix}, x_1, x_0);$ 
15   $\omega_3 \leftarrow 3D(\begin{bmatrix} I_{3 \times 3} & 0 \end{bmatrix}, \begin{bmatrix} \Omega_2 & \tau_1 \end{bmatrix}, x_1, x_0);$ 
16  se  $\omega_1$  na frente de pose( $\begin{bmatrix} I_{3 \times 3} & 0 \end{bmatrix}$ ) e pose( $\begin{bmatrix} \Omega_1 & \tau_1 \end{bmatrix}$ ) então
17     $\begin{bmatrix} \Omega & \tau \end{bmatrix} \leftarrow \begin{bmatrix} \Omega_1 & \tau_1 \end{bmatrix};$ 
18  senão se  $\omega_2$  na frente de pose( $\begin{bmatrix} I_{3 \times 3} & 0 \end{bmatrix}$ ) e pose( $\begin{bmatrix} \Omega_1 & \tau_2 \end{bmatrix}$ ) então
19     $\begin{bmatrix} \Omega & \tau \end{bmatrix} \leftarrow \begin{bmatrix} \Omega_1 & \tau_2 \end{bmatrix};$ 
20  senão se  $\omega_3$  na frente de pose( $\begin{bmatrix} I_{3 \times 3} & 0 \end{bmatrix}$ ) e pose( $\begin{bmatrix} \Omega_2 & \tau_1 \end{bmatrix}$ ) então
21     $\begin{bmatrix} \Omega & \tau \end{bmatrix} \leftarrow \begin{bmatrix} \Omega_2 & \tau_1 \end{bmatrix};$ 
22  senão
23     $\begin{bmatrix} \Omega & \tau \end{bmatrix} \leftarrow \begin{bmatrix} \Omega_2 & \tau_2 \end{bmatrix};$ 
24 senão
25   $\begin{bmatrix} \Omega & \tau \end{bmatrix} \leftarrow \text{homografia}(F_1, F_0, A, \Lambda);$ 

```

Algoritmo A.2: Algoritmo de *front-end* do SLAM Visual Monocular

```

1  enquanto verdadeiro faça
2       $I_i \leftarrow$  leia imagem;
3       $C_i \leftarrow$  pontos de interesse( $I_i$ );
4       $D_i \leftarrow$  descritores( $C_i$ );
5       $F_i \leftarrow$  frame(pose( $\begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{0} \end{bmatrix}$ ),  $C_i$ ,  $D_i$ );
6      se estado = INICIALIZANDO então
7          se  $F_{i-1} \neq \text{NULO}$  então
8               $A \leftarrow$  associação( $F_i$ ,  $F_{i-1}$ );
9               $\begin{bmatrix} \Omega & \tau \end{bmatrix} \leftarrow$  estima rotação e translação( $F_i$ ,  $F_{i-1}$ ,  $A$ ,  $\Lambda$ );
10              $\mathbf{p} \leftarrow$  pose( $\begin{bmatrix} \Omega & \tau \end{bmatrix}$ );
11              $P_{3D} \leftarrow$  triangula pontos( $F_i$ ,  $F_{i-1}$ ,  $A$ );
12              $F_i.\mathbf{p} \leftarrow \mathbf{p}$ ;
13              $T_{3D} \leftarrow F_{i-1} \cup F_i$ ;
14              $j_i \leftarrow i - 1$ ;
15              $j_f \leftarrow i$ ;
16             estado  $\leftarrow$  MAPEANDO;
17         senão
18              $F_{i-1} \leftarrow F_i$ ;
19     senão
20          $\mathbf{p} \leftarrow T_{3D}.F_{i-1}.\mathbf{p} \ominus T_{3D}.F_{i-2}.\mathbf{p}$ ;
21          $F_i.\mathbf{p} \leftarrow T_{3D}.F_{i-1} \oplus \mathbf{p}$ ;
22          $A \leftarrow$  associação( $F_i$ ,  $F_{i-1}$ );
23         se  $A.tamanho < \text{limiar para keyframes}$  então
24              $P_{3D} \leftarrow P_{3D} \cup$  triangula pontos( $F_i$ ,  $F_{i-1}$ ,  $A$ );
25              $T_{3D} \leftarrow T_{3D} \cup F_i$ ;
26              $A \leftarrow$  associação( $F_i$ ,  $F_{j_i}$ );
27             enquanto  $A.tamanho < \text{limiar para janela}$  e  $j_i < j_f$  faça
28                  $j_i \leftarrow j_i + 1$ ;
29                 se  $j_i \neq j_f$  então
30                      $A \leftarrow$  associação( $F_i$ ,  $F_{j_i}$ );
31              $j_f \leftarrow i$ ;

```

Algoritmo A.3: Algoritmo Levenberg-Marquardt para matrizes esparsas e otimização com o cálculo do incremento para poses usando variedades lineares .

Entrada: $\bar{\mathbf{p}}_0$; Valores iniciais para a pose

Entrada: $\bar{\mathbf{p}}_f$; Poses fixas

Entrada: $\bar{\Upsilon}$; Conjunto das arestas na janela de otimização

Entrada: $\bar{\mathbf{X}}_0$; Valores iniciais para os pontos de referência

Entrada: $\bar{\mathbf{X}}_f$; Pontos de referência fixos no espaço

Saída: $\bar{\mathbf{p}}^*$, $\bar{\mathbf{X}}^*$

```

1  $k \leftarrow 1$ ;
2  $\{\mathbf{h}_k, \mathbf{H}_k\} \leftarrow \text{SparseLinearSystem}(\bar{\mathbf{p}}_{k-1}, \bar{\mathbf{p}}_f, \bar{\Upsilon}, \bar{\mathbf{X}}_{k-1}, \bar{\mathbf{X}}_f)$ ;
3  $\nu \leftarrow \tau \max(\text{diag}(\mathbf{H}_k))$ ;
4  $v \leftarrow 2$ ;
5  $\text{feito} \leftarrow \|\mathbf{h}_k\|_\infty < \tau_1$ ;
6 enquanto  $k < k_{\max}$  e  $\neg \text{feito}$  faça
7    $\Delta\Theta_k^* \leftarrow \text{SparseCholenskySolver}((\mathbf{H}_k + \nu \mathbf{I})\Delta\Theta_k^* = -\mathbf{h}_k)$ ;
8   se  $|\Delta\Theta_k^*| < \tau_2(|\Delta\Theta_k^*| + \tau_2)$  então
9      $\text{feito} \leftarrow \text{verdadeiro}$ ;
10  senão
11     $\{\bar{\mathbf{p}}_k, \bar{\mathbf{X}}_k\} \leftarrow \{\bar{\mathbf{p}}_{k-1}, \bar{\mathbf{X}}_{k-1}\} \boxplus \Delta\Theta_k^*$ ;
12     $l \leftarrow \frac{F([\bar{\mathbf{p}}_{k-1}, \bar{\mathbf{X}}_{k-1}]) - F([\bar{\mathbf{p}}_k, \bar{\mathbf{X}}_k])}{\frac{1}{2}\Delta\Theta_k^{*T}(\nu\Delta\Theta_k^* - \mathbf{h}_k)}$ ;
13    se  $l > 0$  então
14       $k \leftarrow k + 1$ ;
15       $\{\mathbf{h}_k, \mathbf{H}_k\} \leftarrow \text{SparseLinearSystem}(\bar{\mathbf{p}}_{k-1}, \bar{\mathbf{p}}_f, \bar{\Upsilon}, \bar{\mathbf{X}}_{k-1}, \bar{\mathbf{X}}_f)$ ;
16       $\text{feito} \leftarrow \|\mathbf{h}_k\|_\infty < \tau_1$ ;
17       $\nu \leftarrow \nu \max(\frac{1}{3}, 1 - (2l - 1)^3), v \leftarrow 2$ ;
18    senão
19       $\bar{\mathbf{p}}_k \leftarrow \bar{\mathbf{p}}_{k-1}$ ;
20       $\bar{\mathbf{X}}_k \leftarrow \bar{\mathbf{X}}_{k-1}$ ;
21       $\nu \leftarrow \nu v, v \leftarrow 2v$ ;
22       $k \leftarrow k + 1$ ;
23       $\{\mathbf{h}_k, \mathbf{H}_k\} \leftarrow \{\mathbf{h}_{k-1}, \mathbf{H}_{k-1}\}$ ;
24  $\bar{\mathbf{p}}^* \leftarrow \bar{\mathbf{p}}_{k-1}$ ;
25  $\bar{\mathbf{X}}^* \leftarrow \bar{\mathbf{X}}_{k-1}$ ;

```

(FERNÁNDEZ-MADRIGAL; CLARACO, 2012)

Algoritmo A.4: Algoritmo para cálculo do gradiente e Hessiano para o sistema linear esparso de um sistema de SLAM Visual.

Entrada: $\bar{p}$; Poses na janela de otimização

Entrada: $\bar{p}_f$; Poses fixas na janela de otimização

Entrada: $\bar{\Upsilon}$; Conjunto de arestas na janela de otimização

Entrada: $\bar{X}$; Pontos de referência na janela de otimização

Entrada: $\bar{X}_f$; Pontos de referência fixos na janela de otimização

Saída: $\tilde{h}$, $\tilde{H}$

```

1  $\tilde{h} \leftarrow \mathbf{0}_{Nm \times 1}$ ;
2  $\tilde{H} \leftarrow \text{MatrizEsparsaZeros}(Nm \times Nm)$ ;
3 para cada vértice  $n$  aparecendo em  $\bar{\Upsilon}$  faça
4    $E_n \leftarrow \left. \frac{\partial \{\bar{p}, \bar{X}\}^{\oplus \epsilon_n}}{\partial \epsilon_n} \right|_{\epsilon_n=0}$ ;
5 para cada aresta  $\Upsilon_{i,j} \in \bar{\Upsilon}$  faça
6    $r_{i,j} \leftarrow h_{i,j}(\mathbf{p}_i, \mathbf{X}_j) \ominus \Upsilon_{i,j}$ ;
7    $L_{i,j} \leftarrow \left. \frac{\partial \ln(y)}{\partial y} \right|_{y=r_{i,j}}$ ;
8    $J_{i,j}^i \leftarrow \left. \frac{\partial r_{i,j}(\check{\mathbf{p}}_i, \mathbf{X}_j)}{\partial \check{\mathbf{p}}_i} \right|_{\check{\mathbf{p}}_i=\mathbf{p}_i}$   $J_{i,j}^j \leftarrow \left. \frac{\partial r_{i,j}(\mathbf{p}_i, \check{\mathbf{X}}_j)}{\partial \check{\mathbf{X}}_j} \right|_{\check{\mathbf{X}}_j=\mathbf{X}_j}$ ;
9    $\tilde{J}_{j,j}^i \leftarrow L_{i,j} J_{i,j}^i \mathbf{E}_i$ ,  $\tilde{J}_{i,j}^j \leftarrow L_{i,j} J_{i,j}^j \mathbf{E}_j$ ;
10   $\tilde{r}_{i,j} \leftarrow \ln(r_{i,j})$ ;
11   $\tilde{\Lambda}_{i,j} \leftarrow (\mathbf{L}_{i,j} \Sigma_{i,j} \mathbf{L}_{i,j}^T)^{-1}$ ;
12   $\tilde{h}(i) \leftarrow \tilde{h}(i) + \tilde{J}_{i,j}^T \tilde{\Lambda}_{i,j} \tilde{r}_{i,j}$ ,  $\tilde{h}(j) \leftarrow \tilde{h}(j) + \tilde{J}_{i,j}^T \tilde{\Lambda}_{i,j} \tilde{r}_{i,j}$ ;
13   $\tilde{H}(i,i) \leftarrow \tilde{H}(i,i) + \tilde{J}_{i,j}^T \tilde{\Lambda}_{i,j} \tilde{J}_{i,j}^i$ ,  $\tilde{H}(i,j) \leftarrow \tilde{H}(i,j) + \tilde{J}_{i,j}^T \tilde{\Lambda}_{i,j} \tilde{J}_{i,j}^j$ ;
14   $\tilde{H}(j,i) \leftarrow \tilde{H}(j,i) + \tilde{J}_{i,j}^T \tilde{\Lambda}_{i,j} \tilde{J}_{i,j}^j$ ,  $\tilde{H}(j,j) \leftarrow \tilde{H}(j,j) + \tilde{J}_{i,j}^T \tilde{\Lambda}_{i,j} \tilde{J}_{i,j}^j$ ;

```

(FERNÁNDEZ-MADRIGAL; CLARACO, 2012)