UMA ABORDAGEM DISTRIBUÍDA PARA O SLAM VISUAL

LETÍCIA HELENA SILVA PORTO

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA DE SISTEMAS ELETRÔNICOS E AUTOMAÇÃO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

UMA ABORDAGEM DISTRIBUÍDA PARA O SLAM VISUAL

LETÍCIA HELENA SILVA PORTO

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE.

APROVADA POR:

GEOVANY ARAÚJO BORGES, Dr., ENE/UNB
(ORIENTADOR/PRESIDENTE)

MARIANA COSTA BERNARDES, Dr., FGA/UNB
(EXAMINADOR INTERNO)

CARLA MARIA CHAGAS E CAVALCANTE KOIKE, Dr., CIC/UNB
(EXAMINADOR INTERNO)

Brasília, 17 de dezembro de 2019.

FICHA CATALOGRÁFICA

SILVA PORTO, LETÍCIA HELENA,
UMA ABORDAGEM DISTRIBUÍDA PARA O SLAM VISUAL [Distrito Federal] 2019.
vii+65 p., 210 x 297 mm (ENE/FT/UnB, Mestre, Engenharia de Sistemas Eletrônicos e Automação, 2019).
DISSERTAÇÃO DE MESTRADO – Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. SLAM visual	2. Robótica móvel
3. ORB-SLAM	4. SLAM Distribuído
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

S. PORTO, LETICIA H. (2019). UMA ABORDAGEM DISTRIBUÍDA PARA O SLAM VISUAL, DISSERTAÇÃO DE MESTRADO, Publicação PPGEA.DM-735/19, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, vii+65.

Letícia Helena Silva Porto

Departamento de Eng. Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

À minha família e amigos.

AGRADECIMENTOS

Primeiramente, gostaria de agradecer aos meus orientadores, professor Geovany Borges e professor João Ishihara, por toda a ajuda e conhecimento, que foram essenciais para o desenvolvimento deste trabalho.

Além disso, gostaria de agradecer a todos os meus amigos que me acompanharam durante todos esses anos da minha vida acadêmica: Camila, Johnny, Alexandre, Artur, Ricardo, Thiago e Sarah. Vocês estavam presentes em todos esses anos de graduação e mestrado e sempre foram fonte de inspiração e motivação. Muito obrigada por terem feito dessa experiência a melhor possível!

Agradeço também ao meu marido, Rodrigo, por ter sido meu ponto de apoio durante todos esses anos e por ter vivenciado ao meu lado todas as frustrações e alegrias dos anos de graduação e mestrado. Obrigada por sempre me motivar a seguir em frente, por estar ao meu lado nos momentos de tristeza, por ter vibrado sempre com as minhas conquistas e por ser a minha inspiração para tentar sempre ser uma pessoa melhor.

Por fim, agradeço à minha família. Obrigada aos meus pais, Francisco e Janete, por todo o trabalho duro que vocês tiveram para que eu pudesse chegar até aqui, por terem sido sempre meu porto seguro e me dado apoio para vencer todas as dificuldades. Agradeço a minha irmã Ana Júlia, por ter sido sempre a minha fonte de inspiração de força e com quem eu sempre compartilhei as minhas vivências. Obrigado aos meus avós, tios e tias, por acreditarem em mim e por fazerem eu me sentir sempre capaz de seguir em frente. Não poderia deixar de agradecer especialmente a minha tia Aparecida, por ter me apoiado durante toda a minha trajetória acadêmica, sem você não teria sido possível chegar até aqui. Amo muito todos vocês.

RESUMO

Título: Uma abordagem distribuída para o SLAM visual
Autor: Letícia Helena Silva Porto
Orientador: Prof. Dr. Geovany Araújo Borges
Coorientador: Prof. Dr. João Yoshiyuki Ishihara

O presente trabalho propõe uma abordagem distribuída para o problema do SLAM visual. A arquitetura desenvolvida com uma abordagem de múltiplos mapas, baseada no ORB-SLAM2, pretende resolver o problema da perda de referência. Este problema ocorre durante o rastreamento quando o robô não consegue mais se localizar no ambiente e precisa esperar passar por um local já mapeado para voltar a executar o SLAM visual. Isso pode acontecer quando o robô muda abruptamente de direção e a nova imagem obtida não possui características visuais em comum com os locais mapeados anteriormente. Como, nesse caso, o único sensor que está obtendo informações do ambiente é a câmera, o robô não é mais capaz de se localizar ou de continuar o mapeamento, uma vez que não consegue encaixar a nova imagem capturada no mapa que está sendo construído. Com o intuito de solucionar esse problema, o sistema desenvolvido neste trabalho propõe a construção de uma nova instância de mapa quando ocorre a perda de referência durante o rastreamento. Além disso, o sistema apresentado está estruturado de modo que a execução do SLAM visual é realizada paralelamente ao gerenciamento das instâncias de mapas construídas. Ademais, a implementação do sistema proposto utilizando o ROS garantiu uma flexibilidade para configurar a execução do SLAM com a quantidade de robôs que o usuário desejar, sem que seja necessário realizar modificações significativas no algoritmo apresentado.

ABSTRACT

Title: A visual SLAM distributed approach
Author: Letícia Helena Silva Porto
Supervisor: Prof. Dr. Geovany Araújo Borges
Co-Supervisor: Prof. Dr. João Yoshiyuki Ishihara

The present work proposes a distributed approach to the visual SLAM problem. The developed architecture with a multi-map approach based on ORB-SLAM2 aims to solve the problem of reference loss. This problem occurs during tracking when the robot can no longer locate itself in the environment and needs to wait until it goes through a previously mapped location in order to run visual SLAM again. This happens when the robot abruptly changes direction and the new image obtained has no visual characteristics in common with previously mapped locations. That happens because in this case the only sensor that is getting environmental information is the camera and the robot is no longer able to locate itself or continue mapping since it cannot fit the new captured image into the map being built. In order to solve this problem, the system presented in this paper proposes the construction of a new map instance when reference loss occurs during tracking. In addition, the presented system is structured so that visual SLAM is executed in parallel with the management of the built map instances. Besides, the implementation of the proposed system using ROS provided flexibility to configure SLAM execution with as many robots as desired without significant modifications to the presented algorithm.

1 INTRODUÇÃO 1
1.1 CONTEXTUALIZAÇÃO 1
1.2 MOTIVAÇÃO 2
1.3 DEFINIÇÃO DO PROBLEMA E OBJETIVOS 3
1.4 REVISÃO BIBLIOGRÁFICA 4
1.5 APRESENTAÇÃO DO MANUSCRITO 5

2 ORB-SLAM2 6
2.1 INTRODUÇÃO 6
2.2 PRINCÍPIOS BÁSICOS DO SLAM VISUAL 6
2.3 ESTRUTURA DO ORB-SLAM2 11
2.3.1 EXTRAÇÃO DE *Features* 13
2.3.2 INICIALIZAÇÃO DO MAPA 16
2.3.3 RASTREAMENTO 19
2.3.4 RELOCALIZAÇÃO 22
2.3.5 MAPEAMENTO LOCAL 23
2.3.6 FECHAMENTO DE LAÇO 24

3 METODOLOGIA PROPOSTA 29
3.1 INTRODUÇÃO 29
3.2 ESTRUTURA DA SOLUÇÃO PROPOSTA 30
3.2.1 *Robot Operating System* (ROS) 30
3.2.2 ARQUITETURA DA IMPLEMENTAÇÃO 31
3.3 ABORDAGEM DE MÚLTIPLOS MAPAS PARA O SLAM VISUAL 31
3.3.1 ORB-SLAM2 MODIFICADO 31
3.3.2 CONVERSÃO E ENVIO DAS INSTÂNCIAS DE MAPAS 33
3.3.3 UNIFICA_MAPAS 35
3.4 ABORDAGEM DISTRIBUÍDA PARA O SLAM VISUAL 39

4 RESULTADOS 41
4.1 INTRODUÇÃO 41
4.2 AMBIENTE DE SIMULAÇÃO 41
4.3 ANÁLISE DOS RESULTADOS DA ABORDAGEM DE MÚLTIPLOS MAPAS 42
4.4 ANÁLISE DOS RESULTADOS DA ABORDAGEM DISTRIBUÍDA PARA O SLAM VISUAL 44
4.4.1 MAPEAMENTO COM DOIS ROBÔS 45
4.4.2 MAPEAMENTO COM TRÊS ROBÔS 47

5 CONCLUSÕES 60
5.1 CONSIDERAÇÕES FINAIS 60
5.2 PROPOSTAS PARA TRABALHOS FUTUROS 60

REFERÊNCIAS BIBLIOGRÁFICAS 61

Lista de Figuras

1.1 "Sandstorm" - um dos veículos participantes na competição de veículos autônomos promovida pelo DARPA (fonte: [1]). 2

2.1 Procedimento genérico de estimação da posição no SLAM visual. Figura traduzida de [2]. 8
2.2 Exemplo de situação na qual o BA é utilizado. Figura traduzida de [3]. 10
2.3 Esquemático genérico do funcionamento do SLAM visual. Figura traduzida de [4]. 10
2.4 Exemplos obtidos durante a execução do ORB-SLAM. A imagem (a) exibe os *keyframes* em azul, a câmera atual em verde e os pontos do mapa em preto e vermelho, sendo os pontos em vermelho pertencentes ao mapa local. (b) exibe o gafo de covisibilidade. (c) mostra em verde um exemplo da árvore de dispersão e (d) exibe o grafo essencial. (Fonte: [5]). 12
2.5 Diagrama de funcionamento do ORB-SLAM2. Imagem traduzida de [6]. 13
2.6 Pré processamento dos dados de entrada no ORB-SLAM2. Imagens traduzidas de [6]. 14
2.7 Uma pirâmide de imagens com quatro níveis e um fator de escala 2. Figura traduzida de [3]. 15
2.8 Exemplo da extração de cantos utilizando o FAST (fonte: [7]). 16
2.9 Exemplo do uso da matriz de homografia. A homografia relaciona os pontos das imagens que pertencem a projeção de um mesmo plano. Figura traduzida de [3]. 17
2.10 Exemplo de uso da matriz fundamental. Dada uma projeção $\mathbf{x}_A$ em uma imagem, o ponto correspondente na outra imagem é restrito a uma linha. Figura traduzida de [3]. 18
2.11 Exemplo de uma imagem estéreo sendo rastreada, com os pontos rastreados em destaque (fonte: [6]). 22
2.12 Exemplo de um fechamento de laço durante a execução do SLAM. (a) exibe quando ocorre a detecção do laço e (b) ilustra a correção desse laço (fonte: [5]). 26

3.1 Diagrama geral da estrutura de funcionamento da abordagem proposta em um exemplo para dois robôs. 32
3.2 Diagrama de blocos do ORB-SLAM2 modificado para a abordagem de múltiplos mapas. Em destaque em roxo estão as rotinas principais que executam em paralelo e em vermelho estão os blocos que foram modificados. 34
3.3 Esquemático do paradigma de publicar e assinar implementado para a troca de informações entre os nós do sistema. 34
3.4 Diagrama de blocos do nó "Unifica_Mapas". 36
3.5 Exemplo de um possível configuração da rede para um sistema de SLAM com três robôs. 40

4.1 Mapa do ambiente obtido com o ORB-SLAM2. 43
4.2 Resultado obtido com a abordagem de múltiplos mapas. 43
4.3 Exemplo de um dos pares de imagens disponibilizados pelo banco de dados da KITTI. 44
4.4 Configuração do sistema com dois robôs. 45
4.5 Resultado obtido para um sistema com dois robôs. 46
4.6 Mapa obtido com o algoritmo ORB-SLAM2 original. 46
4.7 Configuração do sistema com três robôs. 47
4.8 Instâncias de mapas geradas no experimento com a sequência de dados número 07. 48
4.9 Resultado obtido da união das duas primeiras instâncias de mapa. 49

4.10 Resultados obtidos da união das três instâncias de mapa geradas com a sequência de número 07. 49
4.11 Mapa obtido com o algoritmo ORB-SLAM2 original com a sequência de número 07..... 50
4.12 Resultados obtidos da união das instâncias de mapa 2 e 3..... 50
4.13 Instâncias de mapas geradas no experimento com a sequência de dados número 09. 51
4.14 Resultado obtido da união das duas primeiras instâncias de mapa. 52
4.15 Resultados obtidos da união das três instâncias de mapa geradas com a sequência de número 09. 53
4.16 Mapa obtido com o algoritmo ORB-SLAM2 original com a sequência de número 09..... 54
4.17 Resultados obtidos da união das instâncias de mapa 2 e 3 para a sequência de dados 09..... 54
4.18 Gráficos com as trajetórias percorridas pelos robôs em relação ao percurso do banco de dados.... 55
4.19 Instâncias de mapas geradas no experimento com a sequência de dados número 05. 56
4.20 Resultado obtido da união das duas primeiras instâncias de mapa geradas a partir da sequência de imagens 05. 57
4.21 Resultados obtidos da união das três instâncias de mapa geradas a partir da sequência de imagens 05, utilizando a abordagem distribuída. 57
4.22 Mapa obtido com o algoritmo ORB-SLAM2 original com a sequência de número 05..... 59
4.23 Resultados obtidos da união das instâncias de mapa 2 e 3 para a sequência de dados 05..... 59

Símbolos Latinos

$\mathbf{O}$	Sistema de coordenadas
d	Distância
$\mathbf{R}$	Matriz de Rotação
$\mathbf{T}$	Transformação de corpo rígido
$\mathbf{S}$	Matriz de transformação de similaridade
$\mathbf{X}$	Posição de um ponto tridimensional no mapa, que representa um ponto no ambiente
$\hat{\mathbf{X}}$	Valor estimado da posição do ponto tridimensional
$\mathbf{x}$	Posição de um *keypoint* em uma imagem
$\hat{\mathbf{x}}$	Estimação da posição de um *keypoint* em uma imagem
$\mathbf{P}$	Matriz da câmera
$\hat{\mathbf{P}}$	Valor estimado da Matriz da câmera
u_E	Coordenada horizontal do *keypoint* estéreo na imagem esquerda
v_E	Coordenada vertical do *keypoint* estéreo na imagem esquerda
u_D	Coordenada horizontal do *keypoint* na imagem direita
$\mathbf{H}$	Matriz de homografia
$\mathbf{F}$	Matriz fundamental
I	Imagem
R_H	Resultado da heurística de decisão do modelo geométrico
S_H	Pontuação calculada da homografia
S_F	Pontuação calculada da matriz fundamental
p	Ponto tridimensional do mapa
$\mathcal{P}$	Conjunto de pontos do mapa
K	Imagem chave, ou *Keyframe*
$\mathcal{K}$	Um conjunto de *keyframes*
$\mathbf{n}$	Vetor de direção de observação do ponto do mapa
$\mathbf{D}$	Descritor representativo armazenado nos pontos do mapa
$\mathbf{t}$	Vetor de translação
$\mathbf{v}$	Vetor de *bag of words*
$\mathbb{R}^2$	Conjunto bidimensional dos números reais
$\mathbb{R}^3$	Conjunto tridimensional dos números reais
$\mathbb{R}^+$	Conjunto dos números reais positivos
b	Linha de base dos sensores RGB-D e estéreo
f_x	Coordenada na abcissa da distância focal
f_y	Coordenada na ordenada da distância focal
c_x	Coordenada na abcissa do ponto principal
c_y	Coordenada na ordenada do ponto principal
l	Pixel
s	Pontuação de similaridade
C	Função de custo

$\mathbf{e}$	Erro
M	Mapa
m	Mapa serializado para o formato de mensagem do ROS
$\mathbb{N}$	Conjunto dos números naturais
$\mathbb{N}^+$	Conjunto dos números naturais positivos
J	Janela de busca

Símbolos Gregos

θ	Peso da aresta do grafo de covisibilidade
π	Funções de projeção
τ	Tópico do ROS
λ	Número de imagens de entrada até o início da construção de um novo mapa
Ω	Matriz de covariância
ρ	Função de custo de Huber
δ	Fator de escala
Λ	Matriz de informação da aresta

Grupos Adimensionais

i, j, k	Contador

Subscritos

ref	Referência
E	Esquerdo
D	Direito
A	Atual
C	Candidato
R	Referência
min	Valor mínimo
max	Valor máximo
m	Monocular
s	Estéreo
w	Coordenadas do mundo

Sobrescritos

T	Transposta da matriz
$\wedge$	Valor estimado

Siglas

SLAM	*Simultaneous Localization and Mapping*
DARPA	*Defense Advanced Research Projects Agency*
ORB	*Oriented Fast and Rotated Brief*
GPS	*Global Positioning System*
EKF	*Extended Kalman Filter*
PTAM	*Parallel Tracking and Mapping*
RGB	*Red, Green, Blue*
RGB-D	*Red, Green, Blue and Depth*
DDF-SLAM	*Decentralized Data Fusion SLAM*
DTAM	*Dense Tracking and Mapping*
DPPTAM	*Dense Piecewise Planar Tracking and Mapping*
FAST	*Features from Accelerated Segment Test*
BRIEF	*Binary Robust Independent Elementary Features*
SURF	*Speeded-Up Robust Features*
SIFT	*Scale Invariant Feature Transform*
SVO	*Semidirect Visual Odometry*
DT-SLAM	*Deferred Triangulation for robust SLAM*
LSD-SLAM	*Large-Scale Direct monocular SLAM*
BA	*Bundle Adjustment*
RANSAC	*Random Sample Consensus*
DTL	*Direct Linear Transformation*
PnP	*Perspective-n-Point*
P3P	*Perspective-3-Point*
VO	*Visual Odometry*
DoFs	*Degrees of Freedom*
ROS	*Robot Operating System*
XML	*Extensible Markup Language*
JSON	*JavaScript Object Notation*
DVO-SLAM	*Dense Visual SLAM*
TUM	*Technische Universität München*

Notação

Neste trabalho vetores são representados por letras minúsculas em negrito. Matrizes são representadas por letras maiúsculas em negrito. Já espaços e conjuntos em geral são representados por letras maiúsculas caligráficas.

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

A robótica é um ramo de estudo amplo que atraiu bastante atenção de pesquisadores ao redor do mundo nas últimas décadas, resultando em avanços importantes para a comunidade científica [8]. Um dos ramos de estudo da robótica móvel é o desenvolvimento de robôs autônomos, que são definidos como robôs capazes de navegar em um ambiente dinâmico não-estruturado sem a necessidade de uma intervenção humana [9].

As aplicações da robótica móvel são muito variadas e, por vezes, relacionadas a atividades que poderiam ser nocivas à saúde humana, por exemplo, no resgate de vítimas de acidentes, no transporte de substâncias perigosas ou em atividades de exploração solitárias ou cooperativas junto a outros veículos não tripulados [10, 11, 12]. Para que robôs possam servir de solução efetiva em aplicações reais, é esperado que eles sejam capazes de realizar as atividades necessárias ao mesmo tempo em que se locomovem em ambientes muitas vezes desconhecidos e de difícil acesso.

Um exemplo de aplicação de robôs móveis que vem sendo bastante aprimorado nas últimas décadas são os carros autônomos. A Figura 1.1 exibe o "Sandstorm", um dos participantes do primeiro grande desafio promovido pelo Departamento de Defesa dos Estados Unidos (DARPA), em 2004. Essa foi a primeira competição em larga escala de veículos terrestres autônomos, que buscava demonstrar a capacidade dos diversos métodos de pesquisa em direção autônoma executando em um ambiente real. Nessa competição, nenhum dos participantes foi capaz de completar o percurso, porém, ela serviu para impulsionar as pesquisas de tecnologias para veículos autônomos [1].

Desde então, nos últimos 15 anos, os avanços nos veículos autônomos e na robótica móvel foram significativos. Hoje em dia carros autônomos já são considerados uma possibilidade real, passando de uma mera fantasia de filmes de ficção científica para uma possibilidade em um futuro próximo. Os benefícios de tornar a frota atual de veículos terrestres em meios de transporte autônomos seriam variados, como, por exemplo, o aumento da segurança no transporte de passageiros, não sendo mais necessário garantir que os motoristas estejam sóbrios para guiar o veículo. Além do possível aumento de eficiência nos meios de transporte, com a possibilidade de evitar engarrafamentos.

Entretanto, para que a implementação de meios de transporte com esse nível de autonomia seja viável é necessário que esses tipos de robôs móveis tenham a capacidade de se localizar corretamente no ambiente. De modo que esses robôs sejam capazes de planejar e executar a sua movimentação no ambiente com êxito, mesmo que não possuam nenhuma informação prévia sobre ele.

Para que isto seja possível, é essencial que sejam empregados métodos que forneçam informações precisas da localização desses robôs no ambiente em que estão inseridos. Logo, o problema da localização e do reconhecimento do meio em que os robôs se encontram são uma questão central em aplicações da robótica móvel autônoma.

Figura 1.1: “Sandstorm” - um dos veículos participantes na competição de veículos autônomos promovida pelo DARPA (fonte: [1]).

1.2 MOTIVAÇÃO

A linha de pesquisa escolhida para o desenvolvimento deste trabalho de mestrado foi a robótica móvel, por ser um ramo de estudo que apresentou um salto em seu desenvolvimento nas últimas décadas e pela afinidade dos pesquisadores envolvidos com esse campo de estudo. Ademais, o desejo de atribuir uma certa inteligência a esses robôs, para que estes resolvam problemas de forma autônoma, torna desafiador o desenvolvimento de estratégias de localização e mapeamento de ambientes que não sejam previamente conhecidos.

O projeto de pesquisa para realização desse trabalho iniciou como uma extensão do trabalho realizado em [13], resultando em uma arquitetura modular para robôs móveis autônomos [14]. Durante o decorrer da pesquisa, notou-se que o problema da localização é uma questão central na robótica, para que os robôs possam planejar e executar a sua movimentação no ambiente com êxito.

A localização é um problema de estimar a posição local do robô em relação as coordenadas do mundo [15]. A dificuldade deste problema está no fato de que essa posição não pode ser medida diretamente, mas através de dados coletados do sensores, que normalmente possuem ruídos. Nas últimas décadas, diversas técnicas já foram empregadas visando à obtenção da localização precisa de robôs móveis, como, por exemplo, a utilização das informações da odometria e técnicas estocásticas. Uma das principais técnicas obtidas para resolver esse problema foi a invenção do SLAM, do inglês *Simultaneous Localization and Mapping* [16, 17].

O SLAM permite que um robô seja colocado em um ambiente desconhecido e seja capaz de construir um mapa de forma autônoma. Nas técnicas de SLAM, o robô móvel constrói um mapa de um ambiente ao mesmo tempo em que estima a sua localização. Nesse caso, tanto a trajetória percorrida quanto a localização de todos os pontos de referência do ambiente são estimadas em tempo real, sem a necessidade de um conhecimento *a priori* da localização [4].

As soluções para o problema de SLAM não são apenas avanços para a comunidade científica, elas também são importantes para auxiliar em problemas do mundo real. O SLAM permite que um robô explore de forma autônoma um ambiente nocivo ou inacessível para seres humanos. Além disso, essas técnicas podem ser

aplicadas em ambientes militares, exploração de minas, exploração espacial, entre outras inúmeras situações práticas nas quais um conjunto de robôs podem utilizar os métodos de SLAM para explorar o ambiente [17].

Para tanto, diversas soluções de SLAM foram propostas nas últimas décadas. As primeiras abordagens propuseram soluções utilizando filtragem estocástica para o SLAM. Porém, os resultados obtidos através das metodologias que utilizam a abordagem visual obtiveram avanços mais significativos nos últimos anos [18]. Entre as soluções desenvolvidas na literatura de SLAM visual, o ORB-SLAM2 [5, 6] é atualmente considerado o estado da arte para aplicações com um só robô.

É importante destacar que o ORB-SLAM2 é utilizado em aplicações com um único robô explorando o ambiente. Porém, em muitas aplicações o uso de um conjunto de robôs mapeando o ambiente de forma autônoma e cooperativa apresenta vantagens em relação ao uso de um único robô, uma vez que um sistema de múltiplos robôs permite uma operação mais rápida e com mais eficiência [19]. O sucesso da aplicação com múltiplos robôs irá depender do quão bem eles irão mapear e se localizar no ambiente durante a exploração enquanto eles mantêm a comunicação com os outros robôs da rede e trocam informações para melhorar o desempenho uns dos outros [20].

1.3 DEFINIÇÃO DO PROBLEMA E OBJETIVOS

Recentemente, técnicas de SLAM utilizando apenas os dados fornecidos por câmeras têm sido bastante difundidas e tornaram-se um ramo específico, chamado de SLAM visual [21]. O trabalho proposto é baseado no estado da arte dos métodos de SLAM visual, ORB-SLAM2, que propõe um algoritmo divido em três rotinas principais que executam paralelamente: rastreamento, mapeamento local e o fechamento de laço.

Porém, um dos problemas detectados nesse e em outros métodos de SLAM visual é a perda de referência durante o rastreamento e a posterior necessidade de relocalização do robô no ambiente. Como o robô recebe informações do ambiente ao seu redor apenas através de imagens, quando ocorre alguma interrupção nos dados obtidos ou quando a qualidade das imagens capturadas não é suficiente para o processamento das características do ambiente, o robô perde a sua referência no mapa que estava sendo construído e necessita se relocalizar no ambiente.

As técnicas de SLAM visual propostas na literatura que tem a capacidade de lidar com a perda do rastreamento, assim como o ORB-SLAM2, resolvem esse problema adicionando um módulo de relocalização [21, 2]. Dessa forma, é necessário que o robô volte a observar um local previamente mapeado para que ele seja capaz de calcular a sua posição e voltar a explorar o ambiente.

Porém, foi observado que essas técnicas presentes na literatura funcionam bem quando o robô está explorando ambientes pequenos, mas em ambientes maiores, externos, ele pode demorar até percorrer um local que já foi visitado. Enquanto isso, o sistema irá continuar sem a referência entre a sua posição atual e o mapa que já foi construído do ambiente, ficando impossibilitado de continuar a execução do SLAM.

A fim de solucionar esse problema, este trabalho propõe uma abordagem de múltiplos mapas para o SLAM visual, baseada no ORB-SLAM2. A solução proposta inicia a construção de um novo mapa quando o robô perde a referência durante o rastreamento. Posteriormente, esses mapas são unificados caso seja detectado que eles possuem uma região em comum. Além disso, dado que a metodologia é baseada na construção e fusão de múltiplas instâncias de mapas, ela é facilmente expandida para um sistema com múltiplos robôs.

Assim, este trabalho tem o intuito de desenvolver um algoritmo de SLAM visual distribuído que resolva os problemas de perda do rastreamento e mapeamento do ambiente durante a sua execução, problemas esses detectados em testes realizados com algoritmos já consagrados de SLAM visual. Ademais, pretende-se desenvolver uma abordagem de SLAM visual que seja flexível para a execução com qualquer quantidade de robôs que o usuário desejar utilizar para explorar o ambiente, possibilitando diferentes configurações da rede de comunicação entre eles.

1.4 REVISÃO BIBLIOGRÁFICA

Esta sessão tem como objetivo introduzir e discutir as abordagens encontradas na literatura, que foram fundamentais para a realização deste trabalho. Além disso, serão apresentados as soluções propostas mais relevantes para o avanço das pesquisas em SLAM nas últimas décadas.

Divulgado inicialmente em 1986, em [22], o SLAM aumentou a sua popularidade na década de 1990 com as publicações de trabalhos como [23] e [24]. Em abordagens clássicas de SLAM, o robô estima a sua posição no mundo ao mesmo tempo em que mapeia o ambiente, conforme ele se locomove [25, 26].

Ao longo dos anos, novas abordagens surgiram utilizando diferentes tipos de combinações de sensores, a saber: GPS, sensores inerciais, lasers, sensores de odometria, entre outros. Entretanto, recentemente, técnicas de SLAM utilizando apenas câmeras têm sido bastante difundidas e tornaram-se um ramo específico, chamado de SLAM visual [21].

As soluções propostas para o SLAM são geralmente baseadas em filtros Bayesianos ou em soluções de otimização [2]. Entre os métodos baseados em filtragem, duas metodologias são consideradas por [27, 3] como as principais soluções da chamada era clássica do SLAM:

- SLAM com Filtro de Kalman Estendido (EKF-SLAM) [28]: o EKF-SLAM foi uma das primeiras abordagens criadas e possui esse nome por utilizar o filtro EKF para o cálculo da estimação da localização do robô e dos pontos do mapa. A implementação dessa abordagem gera alguns problemas, como apontado por [4], entre eles, o fato que o custo computacional tende a aumentar com o aumento do mapa, sendo inviável a execução em ambientes muito grandes.
- FastSLAM [29]: Essa metodologia fatora o problema do SLAM em dois. Primeiramente, o problema da localização do robô é resolvido com um filtro de partículas. Depois, é realizada a estimação dos pontos de referência do ambiente.

Porém, apesar dessas duas metodologias serem bastante difundidas na comunidade de pesquisa, um dos problemas principais nessas abordagens é que os erros no processo de estimação são propagados tanto para a estimação da posição atual do robô quanto para a estrutura do mapa, gerando divergências no mapa que não são recuperadas facilmente. Em [18], o autor demonstra que soluções não baseadas em filtragem possuem um melhor desempenho em comparação com as que utilizam técnicas de filtragem. Desde então, a maioria das soluções propostas para o SLAM visual são abordagens que não são baseadas em filtros [2].

Uma das técnicas que revolucionou a forma de resolver o SLAM visual foi o PTAM, do inglês *Parallel Tracking and Mapping,* publicada no ano de 2007 em [30]. Ele foi um dos primeiros trabalhos a ter a ideia de dividir o rastreamento e o mapeamento do ambiente em rotinas que executam paralelamente [31]. Desde então,

diversos trabalhos foram divulgados apresentando variações ou modificações dessa técnica, como [32], [33] e [34].

Em 2015, foi proposto o ORB-SLAM [5] como uma solução eficiente e inovadora para o SLAM visual monocular. Assim como o PTAM, essa metodologia divide o processamento do sistema em rotinas paralelas. Entretanto, o ORB-SLAM possui três *threads* principais: uma para o rastreamento, uma para o mapeamento e uma terceira para a detecção de fechamentos de laços. As principais contribuições desse algoritmo é o uso em tempo real das *features* ORB [16], do inglês *Oriented Fast and Rotated Brief*, e a capacidade de processar a relocalização com invariância ao ângulo de observação do ambiente [5].

Em 2017, os autores expandiram a solução para o ORB-SLAM2 [6]. Nesse trabalho foi apresentado um sistema completo de SLAM visual, desenvolvendo a primeira solução de código livre que recebe imagens de entrada de uma câmera monocular, estéreo ou RGB-D.

Com os avanços das soluções para o SLAM visual, começaram a surgir trabalhos que utilizam essas metodologias em aplicações com múltiplos robôs. No DDF-SLAM, proposto em [35] no ano de 2010, os autores apresentam um método para distribuir as informações da geração do mapa no time de robôs, com o intuito de obter escalabilidade e um melhor custo de processamento.

Em [20], os autores propõem um sistema distribuído de múltiplos robôs para o aprendizado do ambiente. Esse sistema utiliza o SLAM baseado no EKF para cada robô da rede, que constrói um modelo tridimensional do ambiente. Além disso, cada robô compartilha o seu modelo de mapa com os outros da rede para a construção de um mapa global.

Entretanto, entre as soluções de SLAM distribuído encontradas na literatura, somente o trabalho realizado em [36] utiliza o ORB-SLAM para realizar a rotina do SLAM visual em cada robô da rede. Porém, nesse caso, cada robô compartilha os *keyframes* gerados com os outros da rede. O mapa construído pelos robôs será formado por todos os *keyframes* obtidos pelo conjunto de robôs.

1.5 APRESENTAÇÃO DO MANUSCRITO

Este trabalho está organizado em cinco capítulos. O primeiro capítulo foi composto por esta introdução ao trabalho e foi escrito com o intuito de apresentar o contexto e a motivação para a sua realização, além de conter uma revisão bibliográfica da literatura utilizada como base para o desenvolvimento deste projeto. No Capítulo 2 é apresentada a estrutura geral das metodologias de SLAM visual e o ORB-SLAM2, algoritmo no qual este trabalho foi baseado. Além disso, é detalhado o embasamento teórico utilizado para o desenvolvimento deste projeto. No Capítulo 3 é apresentada a solução de algoritmo desenvolvida durante o trabalho, descrevem-se as etapas e a estrutura de implementação da abordagem distribuída para o SLAM visual proposta. No Capítulo 4 são analisados os resultados obtidos nos testes realizados em ambiente de simulação para a validação da metodologia proposta. Por fim, o Capítulo 5 contém as conclusões deste trabalho e as sugestões de propostas para trabalhos futuros.

2 ORB-SLAM2

2.1 INTRODUÇÃO

Este capítulo irá, primeiramente, apresentar a estrutura genérica de uma solução para o problema do SLAM visual para sistemas com apenas um robô móvel, utilizando como base os métodos aplicados nas principais implementações encontradas na literatura. Em particular, será apresentado o ORB-SLAM2, método considerado atualmente o estado da arte e a solução mais completa para o SLAM visual monoagente [21], dando ênfase nos principais conceitos teóricos utilizados para o desenvolvimento da proposta apresentada neste trabalho.

2.2 PRINCÍPIOS BÁSICOS DO SLAM VISUAL

Essa seção tem o intuito de fornecer um melhor entendimento ao leitor sobre a estrutura geral de um algoritmo de SLAM visual, utilizada pelas implementações consideradas o estado da arte na literatura para sistemas com apenas um robô móvel [2]. O SLAM visual se refere ao problema de utilizar imagens como a única fonte de informação do ambiente para obter a posição do robô, veículo ou de uma câmera em movimento, em um ambiente desconhecido, e, ao mesmo tempo, construir uma representação do ambiente explorado [31].

De maneira geral, um algoritmo de SLAM visual funciona executando módulos que se repetem a cada ciclo de execução durante o processo de construção incremental do mapa [17, 21, 2].

A estrutura básica de um algoritmo genérico de SLAM visual é composta pelas etapas de (1) obtenção e associação dos dados de entrada, (2) inicialização do mapa, (3) estimação da posição e (4) atualização do mapa. Com o intuito de tornar o algoritmo mais robusto, em algumas técnicas também são utilizados os módulos de (5) otimização do mapa, (6) detecção de fechamento de laço e (7) relocalização [21, 2]. Uma descrição mais detalhada de cada etapa é dada a seguir:

1. Obtenção e associação dos dados de entrada

   O primeiro passo da execução consiste em obter dados do ambiente através das imagens capturadas pela câmera. As soluções existentes na literatura podem utilizar imagens capturadas por diferentes tipos de câmera, como monocular, estéreo ou RGB-D.

   Os métodos de SLAM visual podem ser divididos em diretos, indiretos ou híbridos [2]. Métodos diretos, como o DTAM [37] e o DPPTAM [38], exploram as informações disponíveis a cada pixel da imagem para estimar os parâmetros da posição da câmera.

   Os métodos indiretos, por sua vez, surgiram com o intuito de reduzir a complexidade computacional do processamento de cada pixel utilizando apenas as características mais marcantes da imagem ou pontos de referência do ambiente, que podem ser encontrados em imagens subsequentes. Esse conjunto de características é chamado de *features.*

   Na literatura, há décadas de pesquisa em diferentes tipos de algoritmos para a extração de *features* de imagens, sendo o FAST [39] um dos mais utilizados, e de algoritmos de descritores de *features.* Esses

dois algoritmos são utilizados em conjunto para que possa ser feita a associação dos dados. Entre os descritores desenvolvidos têm-se o BRIEF [40], SURF [41], SIFT [42], ORB [16], entre outros.

Métodos como o SVO [43], que utilizam uma combinação da metodologia direta para estabelecer a correspondência de *features* e da indireta para refinar a estimação da posição da câmera, são considerados híbridos.

A associação dos dados ocorre ao comparar as *features* atuais com as extraídas nas iterações anteriores. A partir desse processo é possível obter correspondências entre as diferentes imagens de entrada e calcular as posições dos pontos de referência observados e a posição do robô [2, 17].

Como este trabalho foi realizado com a abordagem baseada em *features*, a partir desse ponto serão considerados os métodos indiretos para a explicação dos próximos passos.

2. Inicialização do mapa

   Para inicializar o SLAM visual, é necessário definir o sistema de coordenadas para as estimações das posições da câmera e o sistema de coordenadas 3D que irá representar o ambiente. O objetivo dessa etapa é calcular a posição relativa entre duas imagens de entrada para obter um conjunto inicial de pontos do mapa e a posição inicial do robô [5].

3. Estimação da posição

   Para o cálculo da estimação da posição da câmera em um novo *frame* de entrada, a associação entre as *features* observadas na nova imagem de entrada com as observadas anteriormente deve ser computada. A partir dos dados relativos calculados entre o novo *frame* de entrada e um anterior, a posição da câmera é obtida, dada a posição conhecida do *frame* anterior.

   A maioria das soluções de SLAM visual assume um valor a priori para a posição de um novo *frame*, devido ao custo computacional da execução da associação de dados. A estimação do valor a priori é geralmente o primeiro passo para a estimação da posição e ele é utilizado para guiar e limitar a quantidade de processamento utilizada para a associação de dados [2].

   Um dos modelos de movimentação utilizados para a estimação do valor a priori da posição assume que a câmera está se movimento com uma velocidade constante no ambiente. Este modelo é utilizado por algoritmos como o PTAM [30], DT SLAM [44], ORB-SLAM e o DPPTAM.

   A Figura 2.1 exibe um esquemático genérico de como a estimação da posição funciona. O modelo de movimento é utilizado para calcular o a priori da posição, $\mathbf{O}_m$, do novo *frame*. Uma lista de pontos 3D, que representam os pontos de referência do mapa visíveis no novo *frame*, são projetados na imagem e a associação de dados executa em uma janela de busca J ao redor da localização das projeções dos pontos de referência. O sistema então procede para minimizar o erro d de reprojeção para obter a melhor estimativa de posição para o novo *frame* [2, 21].

4. Atualização do mapa

   O mapa representa o ambiente explorado como uma nuvem de pontos. Na imagem de entrada atual observada pelo robô, novos pontos de referência do ambiente podem ter sidos detectados, nesse caso, coordenadas 3D com a localização no sistema de coordenadas do mapa são calculadas para esses novos pontos de interesse do ambiente e são adicionadas ao mapa em construção [17].

   Na literatura, são utilizados majoritariamente dois tipos de mapas para representar o ambiente: mapas métricos e mapas topológicos. Os mapas métricos modelam o ambiente real através de uma representação contendo informações cartesianas do ambiente.

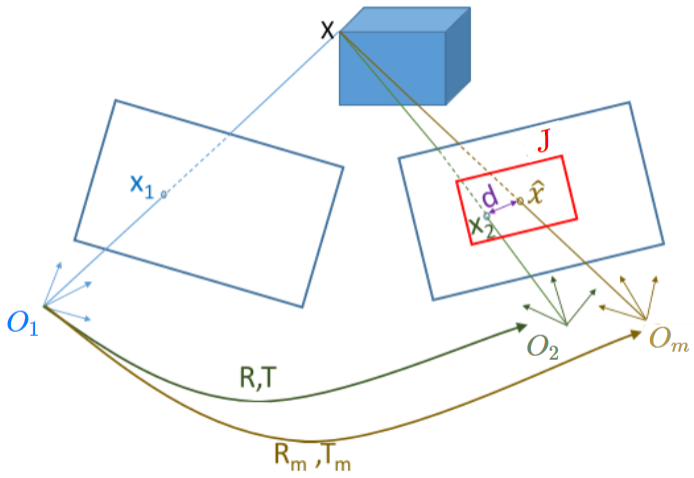


Figura 2.1: Procedimento genérico de estimação da posição no SLAM visual. Figura traduzida de [2].

Porém, para exploração em ambientes extensos, a utilização deste tipo de mapa pode se tornar inviável, dado que ele armazena muitas informações geométricas do ambiente [2].

Os mapas topológicos surgiram para tentar minimizar a quantidade de informações métricas armazenadas em favor de inserir informações de conectividade. Esses consistem em um mapa na forma de nós e arcos, semelhante à um grafo, em que cada nó representa uma característica relevante do ambiente e os arcos indicam um caminho entre essas características [2].

Apesar do uso de mapas topológicos na representação de ambientes grandes apresentar vantagens, também são necessárias informações métricas para que seja possível estimar a posição da câmera. Métodos recentes de SLAM visual, como o LSD-SLAM [45] e o ORB-SLAM, utilizam mapas híbridos, que são localmente métricos e globalmente topológicos.

O uso de um mapa híbrido permite que o sistema observe o mundo com um nível mais alto de abstração, o que permite soluções mais eficientes para fechamentos de laços e correções de falhas utilizando as informações topológicas. Ao mesmo tempo em que é possível melhorar a eficiência do cálculo da posição métrica limitando o escopo do mapa a uma região local ao redor da câmera [2].

5. Otimização do mapa

   A manutenção do mapa é responsável geralmente por sua otimização através do uso do algoritmo de BA [46], do inglês *Bundle Adjustment,* ou através de otimizações de grafos [47]. Durante a exploração do ambiente, novas coordenadas 3D dos pontos de referência do ambiente são obtidas por triangulação baseada no cálculo da estimação da posição da câmera. Porém, conforme o robô vai se movimentando no ambiente, o sistema começa a divergir no cálculo na posição da câmera, devido ao acúmulo dos erros na estimação de posições anteriores que foram utilizadas previamente para expandir o mapa [21, 17].

   O algoritmo de BA é conhecido por fornecer tanto uma estimação acurada da localização da câmera, bem como uma boa reconstrução geométrica do ambiente [46]. A projeção de um ponto tridimensional $\mathbf{X}$ do ambiente para o ponto bidimensional $\mathbf{x}$ correspondente em uma imagem pode ser realizada através de uma matriz da câmera $\mathbf{P}$, sendo $\mathbf{x} = \mathbf{PX}$. Entretanto, câmeras são sujeitas a ruído em uma situação real, o que faz com que essa relação não seja sempre satisfeita corretamente. Dado um cenário no qual tem-se um conjunto de câmeras, com as suas respectivas matrizes da câmera $\mathbf{P}^i$, que observam um conjunto de

pontos do ambiente $\mathbf{X}_j$, como exibido na Figura 2.2, o algoritmo de BA permite otimizar essa solução, assumindo um ruído Gaussiano [3].

O objetivo do BA é estimar as matrizes da câmera $\hat{\mathbf{P}}^i$ e os pontos tridimensionais $\hat{\mathbf{X}}_j$ cujas as projeções correspondentes $\hat{\mathbf{x}}^i_j$ satisfazem corretamente a relação: $\hat{\mathbf{x}}^i_j = \hat{\mathbf{P}}^i\hat{\mathbf{X}}_j$. Além disso, ele também minimiza o erro de reprojeção entre o ponto reprojetado e a medição $\mathbf{x}^i_j$ para cada imagem na qual ele é visualizado, como descrito em [48]. Logo, o algoritmo BA minimiza uma função de erro de reprojeção do tipo [3]

$$\min_{\hat{\mathbf{P}}^i,\hat{\mathbf{X}}_j} \sum_{i,j} d(\hat{\mathbf{P}}^i\hat{\mathbf{X}}_j, \mathbf{x}^i_j)^2, \tag{2.1}$$

na qual $d(\mathbf{x}_1, \mathbf{x}_2)$ representa a distância na imagem entre pontos homogêneos $\mathbf{x}_1$ e $\mathbf{x}_2$.

Porém, devido a sua complexidade computacional, esse algoritmo foi por muito tempo considerado inviável para aplicações em tempo real de SLAM visual [5]. A primeira aplicação de tempo real que utilizou o BA foi o trabalho de odometria visual em [49], seguida pelo método inovador de SLAM visual desenvolvido em [30], o PTAM [5]. Porém, o uso do BA no PTAM só foi possível devido ao uso do conceito de imagens chave, do inglês *keyframes*, em que somente imagens selecionadas como chave são repassadas para o algoritmo, já que seria inviável executá-lo em todas as imagens já mapeadas [2].

6. Fechamento de laço

   Um fechamento de laço ocorre quando o algoritmo detecta uma correspondência entre a imagem atual de entrada com uma mapeada anteriormente. Ou seja, quando o robô revisita um local do ambiente que já foi mapeado. Nesse caso, o erro acumulado devido a movimentação do robô pode ser corrigido [21].

7. Relocalização

   A relocalização é necessária quando o rastreamento falha devido a algum movimento rápido da câmera ou quando ocorre alguma falha na obtenção das imagens. Nesse caso, as *features* obtidas nas novas imagens de entrada não possuem correspondências com as observadas anteriormente, impossibilitando o cálculo da posição atual da câmera e, consequentemente, o robô para de atualizar o mapa.

   Para que seja possível obter uma nova posição para a câmera, o sistema busca comparar os pontos de referência observados nos novos *frames* com os já incorporados ao mapa. Isso ocorre quando o robô passa em um local do ambiente que já estava mapeado [21].

   Caso o módulo de relocalização não esteja incorporado, o sistema para de funcionar caso haja a perda do rastreamento.

A Figura 2.3 exibe um esquemático genérico de como um algoritmo de SLAM visual funciona. O robô, representado pelo triângulo branco, é capaz de estimar a sua posição, dada pelos triângulos cinzas. Além disso, ele é capaz de estimar a posição dos pontos de referência do ambiente, representados por estrelas, em que ele se movimenta [4].

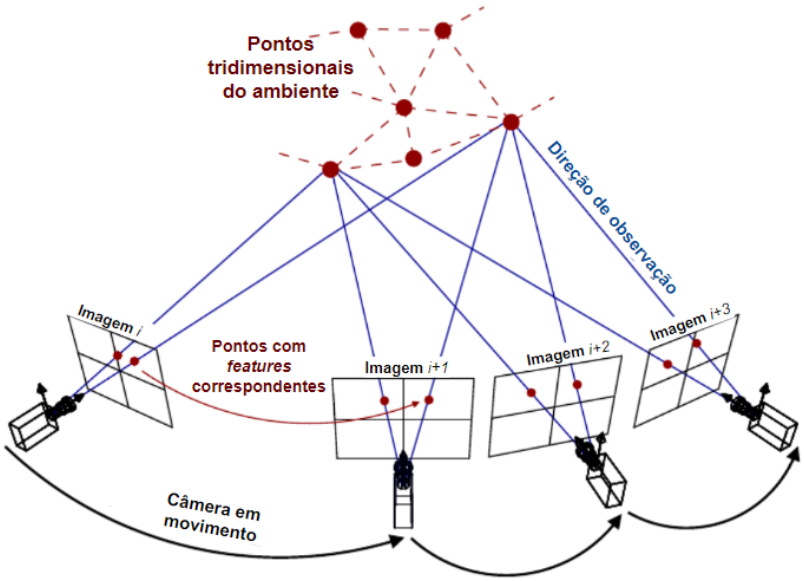


Figura 2.2: Exemplo de situação na qual o BA é utilizado. Figura traduzida de [3].

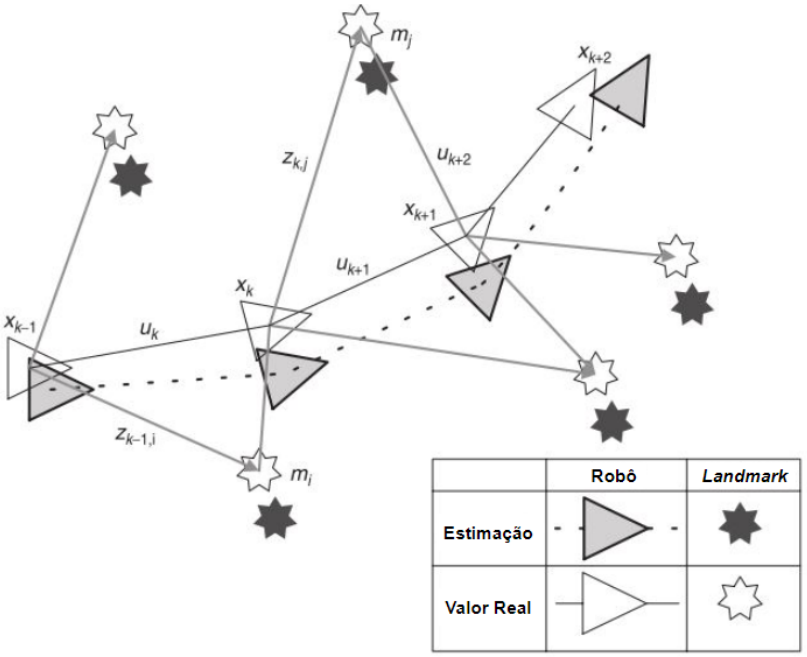


Figura 2.3: Esquemático genérico do funcionamento do SLAM visual. Figura traduzida de [4].

2.3 ESTRUTURA DO ORB-SLAM2

O ORB-SLAM2 [6] é um sistema completo de SLAM visual para sistemas monoagentes, sendo a primeira solução de código livre que permite imagens de entrada de uma câmera monocular, estéreo ou RGB-D, a Figura 2.5 exibe um diagrama do algoritmo. Esse sistema é uma expansão do trabalho anterior dos autores em [5], o ORB-SLAM, que propõe uma solução versátil e acurada para um sistema de SLAM visual monocular. Esse trabalho apresenta contribuições importantes para a literatura, como o uso das mesmas *features* para todas as tarefas. Ou seja, em todos os módulos são utilizadas as mesmas *features* para o processamento, fazendo com que o sistema seja mais simples, eficiente e confiável.

O ORB-SLAM é uma abordagem de SLAM visual monocular que tem o seu nome baseado no descritor de *features* ORB, proposto em [16] como uma alternativa eficiente ao SIFT, do inglês *Scale Invariant Feature Transform*, e ao SURF, do inglês *Speeded-Up Robust Features*. De acordo com os autores, as *features* ORB são mais robustas a variações de iluminação e ao ângulo de captura da imagem e permitem um processamento mais eficiente [5].

A solução proposta pelo ORB-SLAM2 é construída por cima do sistema desenvolvido em [5], na qual são utilizadas algumas técnicas já existentes na literatura, como a execução em paralelo dos módulos de localização e de mapeamento apresentada no PTAM [30]. Com a diferença de que o ORB-SLAM executa três rotinas principais em paralelo: rastreamento, mapeamento local e fechamento de laço, como pode ser visualizado na Figura 2.5.

A primeira etapa da solução é a inicialização do mapa, que consiste em um conjunto de pontos de mapa e imagens chave, chamadas de *keyframes*. Cada ponto de mapa consiste em uma representação 3D de uma coordenada do mundo e armazena informações sobre todas as imagens chave das quais esse ponto pode ser observado. As imagens chave armazenam informações sobre a posição e os parâmetros de configuração da câmera, além das ORB *features* extraídas da imagem [5].

A informação de covisibilidade entre os *keyframes* é bastante utilizada na solução proposta pelo ORB-SLAM2. Para tanto, o sistema utiliza os *keyframes* e os pontos do mapa para construir um grafo de covisibilidade. Nesse grafo, cada nó é formado por um *keyframe* e estes são conectados por arestas caso compartilhem pelo menos 15 observações do mesmo ponto de mapa. Além do grafo de covisibilidade, o sistema constrói uma estrutura de árvore de dispersão, do inglês *spanning tree.* Essa árvore inicia na primeira imagem chave e conecta somente as imagens chave que compartilharem o maior número de pontos de mapa entre elas.

A última estrutura de dados utilizada pelo algoritmo é um grafo essencial, uma versão reduzida do grafo de covisibilidade que contém os mesmos nós, porém considera somente as arestas que tiverem uma alta taxa de covisibilidade e as arestas de fechamento de laço. Isso promove uma otimização mais rápida do mapa global sem muitas perdas de acurácia. A Figura 2.4 exibe exemplos das estruturas utilizadas pelo sistema [6, 5, 3].

Como pode ser visto na Figura 2.5, o ORB-SLAM2 funciona executando três rotinas principais em paralelo. O módulo de rastreamento é responsável por rastrear a localização da câmera em cada imagem de entrada e pela decisão de quando inserir uma nova imagem chave no mapa. A rotina de mapeamento local processa as imagens chave dadas pelo rastreamento e as insere no mapa local, formado pelo conjunto de *keyframes* que compartilham pontos do mapa com a imagem de entrada atual; também é responsável por detectar e excluir as imagens redundantes do mapa local. Por fim, o módulo fechamento de laço procura por fechamento de laços a cada imagem chave recebida. Caso seja detectado um fechamento, é executado um algoritmo com o intuito de reduzir o erro acumulado durante a execução do laço.

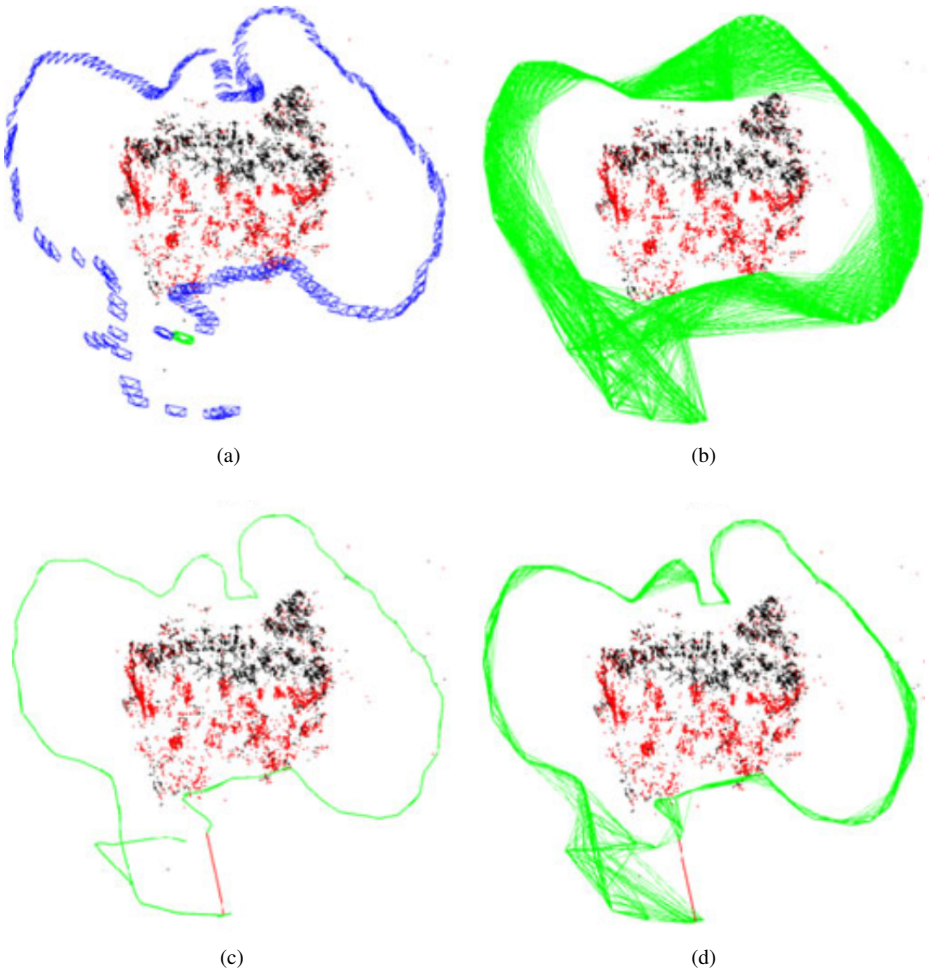


Figura 2.4: Exemplos obtidos durante a execução do ORB-SLAM. A imagem (a) exibe os *keyframes* em azul, a câmera atual em verde e os pontos do mapa em preto e vermelho, sendo os pontos em vermelho pertencentes ao mapa local. (b) exibe o gafo de covisibilidade. (c) mostra em verde um exemplo da árvore de dispersão e (d) exibe o grafo essencial. (Fonte: [5]).

Na Figura 2.5 também pode ser visualizado o módulo de reconhecimento de local que é utilizado para as rotinas de relocalização e fechamento de laço. Esse módulo contém salvas as representações de locais já mapeados em um banco de dados para correspondências posteriores [5, 3].

Nas sessões a seguir os módulos do algoritmo ORB-SLAM2 e as técnicas que são utilizadas por eles serão explicados de forma mais aprofundada.

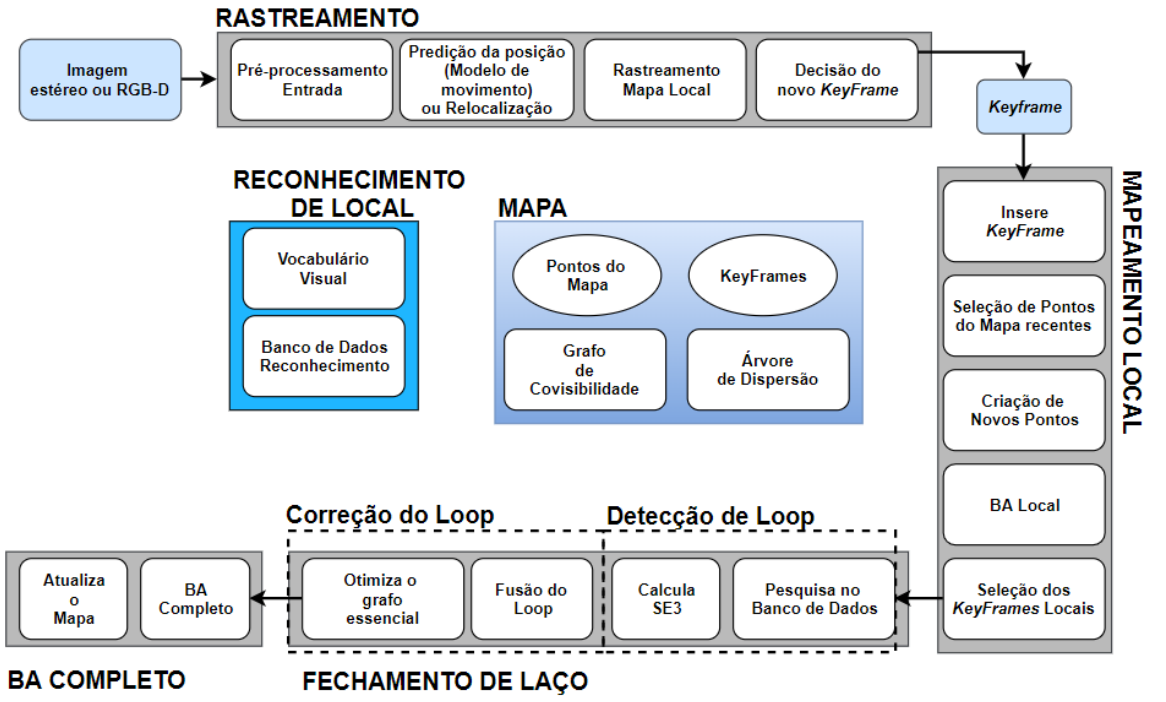


Figura 2.5: Diagrama de funcionamento do ORB-SLAM2. Imagem traduzida de [6].

2.3.1 Extração de *Features*

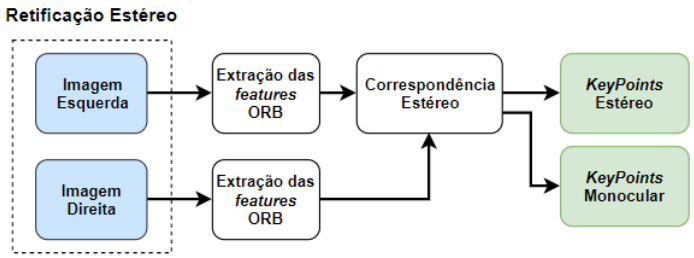
O ORB-SLAM2, assim como os outros métodos baseados em *features,* realiza um pré-processamento dos seus dados de entrada, como exibido nas imagens da Figura 2.6. As imagens de entrada são posteriormente descartadas e todas as operações realizadas no sistema são baseadas nas *features* obtidas, dessa forma, o sistema é independente do tipo de sensor utilizado. Os outros módulos do sistema podem funcionar com *keypoints* monoculares ou estéreo [6].

Os *keypoints* estéreo são definidos pelas coordenadas $\mathbf{x}_s = (u_E, v_E, u_D)$, sendo (u_E, v_E) as coordenadas na imagem da esquerda e u_D a coordenada horizontal na imagem da direita. No caso de câmeras estéreo, as *features* ORB são extraídas nas duas imagens e, para cada *feature* obtida na imagem da esquerda, é procurada uma correspondente na imagem da direita. Posteriormente, o *keypoint* estéreo é gerado com as coordenadas da *feature* da esquerda e a coordenada horizontal da *feature* correspondente da direita.

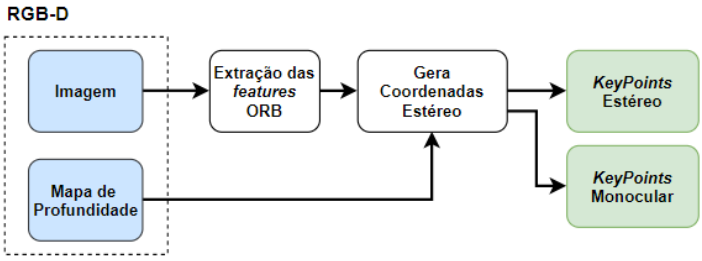
Quando são utilizadas câmeras RGB-D, são extraídas as *features* ORB na imagem RGB e, como proposto em [50], para cada *feature* com coordenada (u_E, v_E), o seu valor de profundidade, d, é transformado em uma coordenada u_D virtual a partir de

$$u_D = u_E - \frac{f_x b}{d}, \tag{2.2}$$

em que f_x representa a distância focal horizontal e b representa a linha de base do sensor RGB-D, dada pela distância entre o emissor de luz e o sensor de infravermelho. A incerteza do sensor de profundidade é representada pela incerteza na coordenada virtual. Desse modo, as *features* extraídas das imagens de entrada do tipo estéreo ou RGB-D são tratadas igualmente pelo resto do sistema [6].



(a) Pré-processamento realizado em imagens de entrada de uma câmera estéreo.



(b) Pré-processamento realizado em imagens de entrada de uma câmera RGB-D.

Figura 2.6: Pré processamento dos dados de entrada no ORB-SLAM2. Imagens traduzidas de [6].

Os *keypoints* estéreo podem ser classificados ainda como próximos ou afastados. Eles são considerados próximos quando a sua profundidade associada é menor que 40 vezes o valor da linha de base da câmera estéreo ou RGB-D, como sugerido em [51], caso contrário, eles são considerados afastados. A triangulação de pontos próximos a partir de uma só imagem pode ser facilmente estimada e são providenciadas informações de translação, rotação e escala. Por outro lado, para pontos afastados é possível estimar de forma acurada apenas a informação de rotação. Logo, a triangulação desses pontos é realizada somente quando eles são visualizados em múltiplas imagens [6].

Os *keypoints* monoculares são representados pelas coordenadas $\mathbf{x}_m = (u_E, v_E)$ e correspondem as *features* ORB obtidas a partir de imagens de entrada capturadas com câmeras monoculares. No caso de imagens de entrada do tipo estéreo ou RGB-D, esses *keypoints* indicam *features* cujas correspondências não foram encontradas na imagem direita ou que possuem um valor de profundidade inválido. Esses pontos são triangulados a partir de múltiplas visualizações e não fornecem informações de escala, apesar de contribuírem para a estimação de translação e rotação [6].

Para a extração de uma *feature* ORB da imagem, o primeiro passo realizado é a criação de uma pirâmide com a imagem de entrada contendo diferentes níveis de granularidade e escalas. Para criar uma pirâmide, é aplicado um filtro de *blur* e uma subamostragem na imagem original. A imagem resultante é adicionada em uma camada acima da imagem original. Assim, esse processo resulta em uma pirâmide com a base sendo a imagem original e camadas de imagens reduzidas acima. A Figura 2.7 exibe um exemplo de uma pirâmide com o fator de escala 2 [3].

A pirâmide permite a extração das *features* em diferentes níveis de escala. Devido a natureza de *features* de imagens como a FAST, um procedimento de extração pode perder dados significativos quando a busca é realizada somente em um nível de escala. Utilizar a pirâmide é uma forma de evitar que isso aconteça [3].

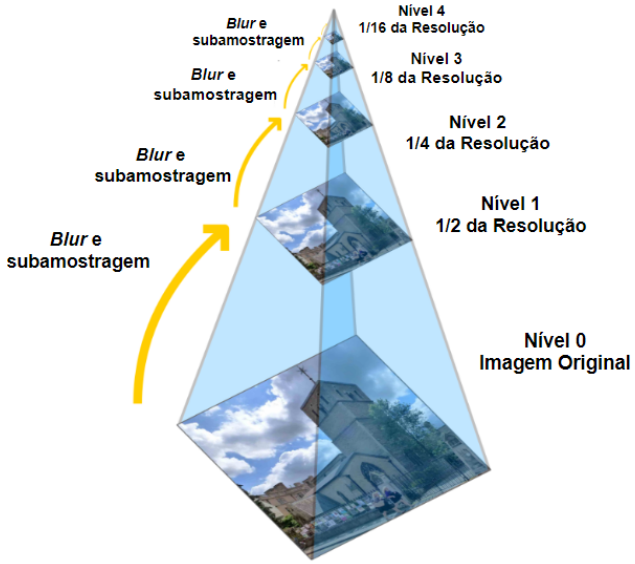


Figura 2.7: Uma pirâmide de imagens com quatro níveis e um fator de escala 2. Figura traduzida de [3].

Esse método também foi utilizado na solução proposta pelo PTAM [30], entretanto, ao invés de utilizar apenas quatro níveis de escala com o fator 2, como é feito no PTAM, os autores do ORB-SLAM escolheram utilizar uma pirâmide de oito níveis com um fator de escala de 1.2 [5, 3].

Após construída a pirâmide, o ORB-SLAM2 executa o algoritmo de extração de cantos FAST [7] em cada nível da pirâmide. A Figura 2.8 exibe um exemplo de extração de cantos utilizando o FAST, no qual considera-se um círculo de três *pixels* de raio para cada *pixel* l da imagem, um canto é detectado caso pelo menos nove *pixels* consecutivos do círculo são mais claros ou mais escuros que l somado a um limiar pré-definido. A vantagem da utilização desse algoritmo em relação aos outros já desenvolvidos na literatura é que ele executa em menor tempo. Uma comparação entre o FAST e outros métodos de detecção de cantos é dada em [7].

Com o intuito de garantir uma distribuição homogênea ao longo de toda a imagem, o ORB-SLAM2 divide as imagens de cada nível da pirâmide em células de 30 por 30 *pixels*. Então, o algoritmo tenta detectar pelo menos cinco cantos em cada uma delas, modificando os valores dos parâmetros do FAST de acordo com a dificuldade de detecção. Dessa forma, o algoritmo garante tratar de forma diferente as regiões distintas da imagem [5].

Após a detecção dos cantos, o próximo passo é o cálculo do descritor de *features* ORB, do inglês *Oriented Fast and Rotated Brief*. Como seu nome indica, ele é uma combinação do FAST e do BRIEF proposta em [16], que adiciona uma componente de orientação ao FAST e melhora a eficiência computacional das *features* BRIEF [16, 3]. Dessa forma, as *features* ORB garantem mais rapidez para serem processadas ao mesmo tempo em que apresentam uma boa invariância ao ângulo de captura, em [52] os autores demonstram a boa performance do ORB aplicado ao reconhecimento de padrões.

Os descritores ORB são obtidos a partir dos cantos detectados pelo FAST e são posteriormente utilizados para todas as execuções de correspondência de *features* do sistema [5]. Além disso, esses descritores são discretizados e armazenados em uma estrutura chamada de *bag of words,* apresentada pelo autor em [53] como uma árvore de vocabulário visual, que é utilizada para acelerar a execução dos algoritmos que procuram por correspondências de *features* [53, 3].

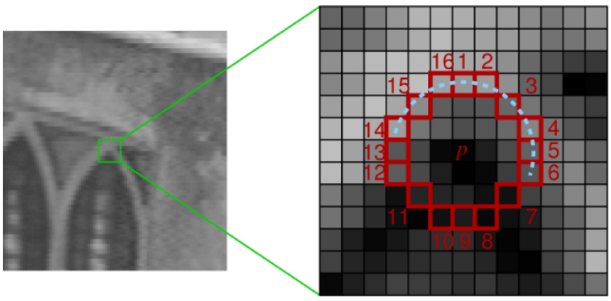


Figura 2.8: Exemplo da extração de cantos utilizando o FAST (fonte: [7]).

2.3.2 Inicialização do Mapa

O objetivo da inicialização do mapa é calcular a posição relativa entre duas imagens para triangular um conjunto inicial de pontos do mapa. Na solução monocular apresentada pelo ORB-SLAM, o mapa é inicializado de forma automática pelo algoritmo, ou seja, ele é independente do ambiente que está sendo mapeado e não requer nenhuma ação humana para selecionar uma boa configuração inicial [5].

Nas soluções de SLAM visual monocular é necessário um procedimento para inicializar o mapa, já que a informação de profundidade não pode ser obtida a partir de apenas uma imagem. Quando a inicialização é realizada, a primeira imagem é utilizada como a imagem inicial, a partir da qual os *frames* seguintes serão comparados.

No ORB-SLAM, a primeira etapa de inicialização do mapa consiste em procurar correspondências entre as imagens. Para isso, as ORB *features* são extraídas da imagem de referência, I_R, e da imagem de entrada atual, I_A, e é executada a rotina de comparação dessas *features*. Caso não sejam encontradas correspondências suficientes, outra imagem será considerada para ser a imagem de referência.

Posteriormente, são calculados dois modelos geométricos em paralelo: uma matriz de homografia $\mathbf{H}_{AR}$, que assume que a cena observada é plana; e uma matriz fundamental $\mathbf{F}_{AR}$, que assume que a cena observada é não plana. A homografia relaciona os pontos vistos em duas imagens distintas, como exibido na Figura 2.9.

A matriz $\mathbf{H}_{AR}$ relaciona os pontos que pertencem as projeções em cada imagem que representam vistas distintas de um mesmo plano através da equação

$$\mathbf{x}_A = \mathbf{H}_{AR}\mathbf{x}_R, \tag{2.3}$$

na qual $\mathbf{x}_R$ representa um ponto da imagem no sistema de coordenadas do *frame* inicial, ou de referência, e $\mathbf{x}_A$ representa um ponto da imagem no sistema de coordenadas do *frame* de entrada atual. A matriz de homografia $\mathbf{H}_{AR}$ mapeia diretamente de um ponto da imagem pertencente a projeção no sistema de coordenadas $\mathbf{O}_A$ para um ponto correspondente na imagem com o sistema de coordenadas $\mathbf{O}_R$, caso os parâmetros intrínsecos da câmera sejam os mesmos [3].

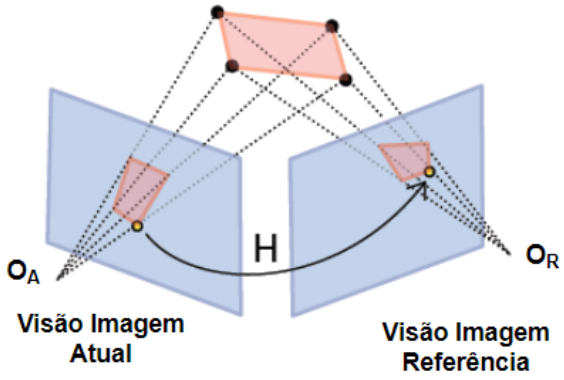


Figura 2.9: Exemplo do uso da matriz de homografia. A homografia relaciona os pontos das imagens que pertencem a projeção de um mesmo plano. Figura traduzida de [3].

A matriz fundamental $\mathbf{F}_{AR}$ também relaciona os pontos de duas imagens que exibem projeções da mesma cena, no entanto, ela não depende da cena ser plana. A Figura 2.10 exibe um exemplo de como a matriz fundamental funciona, $\mathbf{F}_{AR}$ é uma matriz que traz uma relação entre um ponto $\mathbf{x}$ e uma linha que contém a projeção 3D desse ponto, uma teoria mais aprofundada pode ser encontrada em [48]. A matriz fundamental relaciona esses pontos através da equação

$$\mathbf{x}_A^T \mathbf{F}_{AR} \mathbf{x}_R = 0. \tag{2.4}$$

Os dois modelos geométricos são estimados utilizando o algoritmo RANSAC, do inglês *Random Sample Consensus*, como descrito em [48] e [5]. O sistema calcula então uma pontuação baseada no erro de reprojeção para cada matriz obtida, a partir da projeção das múltiplas correspondências de uma imagem na outra. A decisão de qual modelo geométrico utilizar, matriz de homografia ou fundamental, é baseada na heurística

$$R_H = \frac{S_H}{S_H + S_F}, \tag{2.5}$$

na qual R_H é o resultado da heurística, S_H é a pontuação calculada da homografia e S_F é a pontuação calculada da matriz fundamental. Caso $R_H \geq 0,45$, a homografia é escolhida, caso contrário, é escolhida a matriz fundamental [3, 5].

Assim que o modelo geométrico é escolhido, o ORB-SLAM tenta recuperar a hipótese de movimento associada. No caso da homografia, oito hipóteses de movimento são obtidas através do processo explicado em [54]. Já no caso da matriz fundamental, ela é convertida para uma matriz essencial e são obtidas quatro hipóteses de movimento utilizando o método descrito em [48].

Em ambos os casos, todas as hipóteses obtidas são verificadas através da triangulação das *features* correspondentes nas duas imagens, que é feita utilizando o método DTL, do inglês *Direct Linear Transformation*, algoritmo descrito em [48]. Mais detalhes sobre esse processo podem ser obtidos em [5, 3].

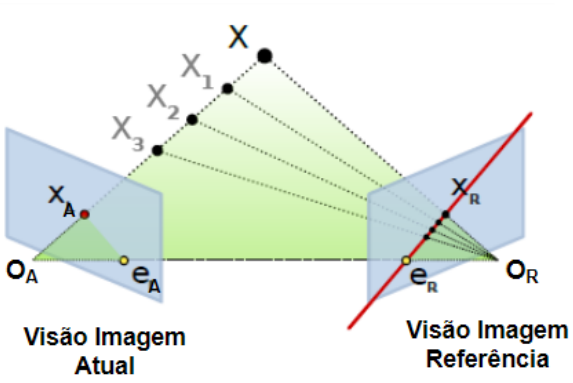


Figura 2.10: Exemplo de uso da matriz fundamental. Dada uma projeção $\mathbf{x}_A$ em uma imagem, o ponto correspondente na outra imagem é restrito a uma linha. Figura traduzida de [3].

Assim que o processo de inicialização é considerado satisfatório, um mapa inicial é criado utilizando a imagem de referência e a imagem atual como os dois primeiros *keyframes* e todas as correspondências que foram trianguladas são utilizadas como pontos do mapa. Cada ponto do mapa, p_i, armazena as seguintes informações:

1. Sua posição 3D, $\mathbf{X}_{w,i}$, no sistema de coordenadas do mundo;
2. a direção de observação $\mathbf{n}_i$, que é um vetor unitário que representa a média de todas as direções das visualizações;
3. um descritor representativo $\mathbf{D}_i$, que é o descritor ORB associado ao ponto cuja distância de hamming é mínima em relação aos outros descritores nos *keyframes* nos quais o ponto é observado;
4. a distância mínima, d_{min}, e máxima, d_{max}, da qual o ponto pode ser observado.

E cada *keyframe,* K_i, armazena os seguintes dados:

1. a posição da câmera, $\mathbf{T}_{iw} \in \mathrm{SE}(3)$, que é uma transformação de corpo rígido, que transforma pontos do mundo para o sistema de coordenadas da câmera;
2. as configurações intrínsecas da câmera, incluindo a distância focal e o ponto principal;
3. todas as *features* ORB extraídas da imagem, sendo estas associadas ou não com um ponto do mapa.

Nesse ponto, é executado o primeiro BA total no mapa, em que todos os *keyframes* e pontos do mapa são otimizados com exceção do primeiro, que permanece fixo como a origem. Caso não haja pelo menos 100 pontos no mapa após a inicialização, essa estrutura é descartada e o processo de inicialização é recomeçado [3].

Um dos benefícios do uso de câmeras estéreo ou RGB-D, inseridos no ORB-SLAM2, é que o conhecimento da informação de profundidade de apenas uma imagem de entrada retira a necessidade de uma estrutura específica para a inicialização do movimento, como no caso monocular. No início da execução do sistema é criado um *keyframe* a partir da primeira imagem. A sua posição é considerada a origem e um mapa inicial é criado a partir de todos os *keypoints* estéreo [6].

2.3.3 Rastreamento

Após uma inicialização bem-sucedida, a rotina de rastreamento é utilizada para estimar o movimento do robô e mapear as redondezas a cada imagem de entrada recebida. Como pode ser visualizado na Figura 2.5, o primeiro passo dessa rotina é o pré-processamento das imagens de entrada e a extração das *features* ORB, processo descrito na subseção 2.3.1.

A rotina de rastreamento do ORB-SLAM2 pode ser feita de duas formas, dependendo se o modelo de movimento será utilizado. Como mencionado anteriormente, o ORB-SLAM2 utiliza um modelo de movimento de velocidade constante. Quando o rastreamento é feito com sucesso no *frame* anterior, o sistema utiliza o modelo de movimento para prever a posição da câmera no *frame* atual e para realizar uma busca guiada dos pontos do mapa observados no *frame* anterior.

Nesse caso, o algoritmo executa a associação de dados assumindo que as *features* que foram mapeadas para um ponto do mapa na imagem anterior podem ser encontradas em uma área da imagem atual, que é determinada baseando-se no modelo de movimento. Se não forem encontradas correspondências suficientes entre a imagem atual e a rastreada anteriormente, o rastreamento é realizado sem utilizar o modelo de movimento.

Caso o rastreamento com o modelo de movimento falhe, o sistema irá usar um método diferente para o rastreamento. Ao invés de projetar as *features* para a imagem atual, as *features* da imagem atual são transformadas em um vetor de *bag of words,* que são posteriormente comparadas as *features* do último *keyframe* mapeado [5, 3].

Para isso, o sistema utiliza a biblioteca DBoW2, desenvolvida em [53]. O método *bag of words* é uma técnica que utiliza um vocabulário visual para converter uma imagem de modo que seja representada por um vetor numérico esparso, permitindo que o sistema administre grandes conjuntos de imagens. O vocabulário visual é criado de forma offline com os descritores ORB extraídos de um conjunto grande de imagens de ambientes internos e externos. Se esse vocabulário for generalista o suficiente, ele pode ser utilizado para a execução do algoritmo com uma boa performance em diferentes ambientes [52].

Utilizando os vetores de *bag of words*, a biblioteca cria um banco de dados hierárquico, que facilita a comparação para detectar se o robô está passando por um local já mapeado anteriormente [53]. O sistema constrói então um banco de dados de forma incremental que contém um índice inverso, que armazena para cada *visual word* do vocabulário em quais *keyframes* que ela foi vista, de modo que a busca no banco de dados seja feita de forma mais eficiente. Esse banco de dados é atualizado também quando um *keyframe* é apagado em algum dos módulos do sistema [5].

O sistema então faz uso desse banco de dados para procurar correspondências entre o *keyframe* mapeado com a imagem atual de entrada, para que seja possível estimar a posição atual da câmera. Caso não sejam encontradas pelo menos 15 correspondências, o rastreamento é finalizado e é iniciado o módulo de relocalização.

Em ambos os casos do rastreamento, com ou sem o modelo de movimento, é realizada uma otimização da posição estimada utilizando a versão do algoritmo BA chamada pelos autores do ORB-SLAM2 de "somente-movimento", para isso o sistema utiliza o método de Levenberg-Marquardt implementado na biblioteca g2o [47]. Esse algoritmo otimiza a orientação da câmera, $\mathbf{R} \in SO(3)$, e a posição, $\mathbf{t} \in \mathbb{R}^3$, com o intuito de minimizar o erro de reprojeção entre pontos 3D correspondentes em coordenadas do mundo, $\mathbf{X}^i \in \mathbb{R}^3$, e *keypoints,* $\mathbf{x}^i_{(.)}$ [6]. Estes podem ser monoculares, $\mathbf{x}^i_m \in \mathbb{R}^2$, ou estéreo, $\mathbf{x}^i_s \in \mathbb{R}^3$, com $i \in \mathcal{X}$ sendo o conjunto de todas as correspondências, de modo que

$$\{\mathbf{R},\mathbf{t}\} = \underset{\mathbf{R},\mathbf{t}}{\operatorname{argmin}} \sum_{i\in\mathcal{X}} \rho\left(\left\|\mathbf{x}^i_{(.)} - \pi_{(.)}(\mathbf{R}\mathbf{X}^i + \mathbf{t}\right\|^2_{\Sigma}\right), \tag{2.6}$$

em que ρ é a função de custo de Huber e Σ representa a matriz de covariância associada com a escala do *keypoint* [6]. As funções de projeção $\pi_{(.)}$, monocular π_m e estéreo π_s, são definidas como

$$\pi_m\left(\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}\right) = \begin{bmatrix} f_x \dfrac{X}{Z} + c_x \\ f_y \dfrac{Y}{Z} + c_y \end{bmatrix}, \tag{2.7}$$

$$\pi_s\left(\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}\right) = \begin{bmatrix} f_x \dfrac{X}{Z} + c_x \\ f_y \dfrac{Y}{Z} + c_y \\ f_x \dfrac{X-b}{Z} + c_x \end{bmatrix}, \tag{2.8}$$

em que a distância focal, (f_x, f_y), o ponto principal, (c_x, c_y), e a linha de base da câmera estéreo, dada pela distância entre os centros de projeção das câmeras, do inglês *baseline,* b, são os parâmetros conhecidos de calibração da câmera [6]. Nessa otimização, ou BA somente-movimento, todos os pontos são fixados e apenas a posição da câmera é otimizada.

Assim que é obtida uma estimação para a posição e um conjunto inicial de correspondências de *features,* o sistema é capaz de projetar o mapa na imagem e procurar por mais pontos de mapa correspondentes. Com o intuito de manter o sistema operando em um ambiente extenso sem precisar lidar com a complexidade de mapas grandes, um subconjunto do mapa global, chamado de mapa local, é utilizado para essa projeção. O mapa local é formado pelo conjunto de *keyframes* $\mathcal{K}_1$ que compartilham pontos do mapa com o *frame* atual; e pelo conjunto $\mathcal{K}_2$, formado pelos vizinhos dos *keyframes* pertencentes ao conjunto $\mathcal{K}_1$ no grafo de covisibilidade [5].

No ORB-SLAM2, os *keyframes* e os pontos do mapa são utilizados para construir um grafo de covisibilidade. A informação de covisibilidade entre os *keyframes* é bastante útil em múltiplas rotinas do sistema e é representada como um grafo ponderado não direcionado, como em [50]. Cada nó é formado por um *keyframe* e uma aresta entre dois *keyframes* existe caso eles compartilhem pelo menos 15 observações de pontos do mapa, sendo o peso da aresta, representado por θ, dado pelo número de pontos do mapa em comum [5].

O mapa local também possui um *keyframe* de referência, $K_{ref} \in \mathcal{K}_1$, que compartilha a maior quantidade de pontos do mapa com a imagem atual. Agora, o algoritmo possui uma estimação para a posição atual, assim como alguns pontos do mapa, o sistema irá então utilizar o mapa local para refinar essa estimação da posição. Isso é realizado utilizando os cinco passos a seguir, para cada ponto do mapa contido nos *keyframes* de $\mathcal{K}_1$ e $\mathcal{K}_2$ [5, 3]:

1. Calcula a projeção dos pontos do mapa na imagem atual.
2. Calcula o ângulo entre o raio de visualização atual, $\mathbf{v}$, e a média das direções de visualização do ponto, $\mathbf{n}$. Descarta se $\mathbf{v} \cdot \mathbf{n} < \cos(60°)$.

3. Calcula a distância d entre o ponto do mapa e o centro da câmera. Descarta se o valor estiver fora da região $d \notin [d_{min}, d_{max}]$.

4. Calcula a escala na imagem como d/d_{min}.

5. Compara o descritor representativo $\mathbf{D}$ do ponto do mapa com as *features* sem correspondências com pontos de mapa da imagem atual, na escala calculada, e associa o ponto do mapa com a melhor correspondência.

Assim que esse processo é finalizado para todos os pontos do mapa local, a posição é otimizada novamente utilizando as novas correspondências criadas. Caso sejam encontradas menos de 30 correspondências entre os pontos do mapa e as *features* da imagem atual, o rastreamento é interrompido e é inicializada a rotina de relocalização [5, 3].

Quando o rastreamento ocorre com sucesso até esse ponto, a última etapa dessa rotina é decidir se a imagem atual deve se tornar um *keyframe.* O ORB-SLAM2 segue o método introduzido pelo ORB-SLAM de inserir novos *keyframes* o mais rápido possível, pois isso faz com que o rastreamento seja mais robusto a pertubações na movimentação da câmera. Nesse sistema um *keyframe* não é imutável, ao invés disso, ele pode ser descartado caso ele seja redundante, ou seja, a área observada por ele for suficientemente representada por outros *keyframes*.

No caso monocular, para que um novo *keyframe* seja inserido, as quatro condições abaixo devem ser satisfeitas [5]:

1. A última relocalização deve ter sido realizada a mais de 20 imagens atrás;

2. a rotina de mapeamento local está em espera ou o último *keyframe* inserido foi a mais de 20 imagens atrás;

3. a imagem atual rastreia pelo menos 50 pontos;

4. a imagem atual rastreia menos que 90% dos pontos contidos no K_{ref}.

A diferença entre pontos próximos e afastados quando a câmera é estéreo ou RGB-D introduz uma outra condição para a inserção de um novo *keyframe,* que pode ser fundamental em ambientes nos quais uma grande parte dos pontos detectados são classificados como afastados. A Figura 2.11 exibe um exemplo de ambiente como esse. Nessa imagem os pontos destacados em verde são classificados como próximos e os pontos em azul são os afastados. Nesse tipo de ambiente o sistema necessita de uma quantidade suficiente de pontos próximos para poder estimar de forma acurada a translação. Portanto, se número de pontos próximos rastreados for menor que um limiar $\tau_t = 100$ e for possível obter pelo menos uma quantidade $\tau_t = 70$ de novos pontos próximos, o novo *keyframe* será inserido.

Ao invés de impor critérios de distância entre as imagens, como realizado no PTAM, no ORB-SLAM2 é utilizada uma regra de mudança visual mínima, como na condição 4. A condição 1 garante uma boa relocalização e a condição 3 um bom rastreamento. Se um *keyframe* for inserido enquanto a rotina de mapeamento local está em execução, um sinal é enviado para que o algoritmo de BA local seja interrompido e o novo *keyframe* seja processado [5].

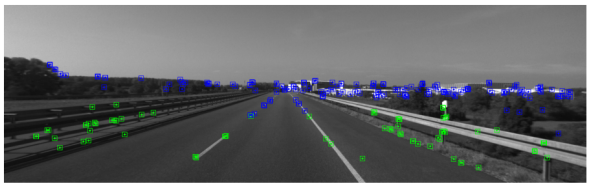


Figura 2.11: Exemplo de uma imagem estéreo sendo rastreada, com os pontos rastreados em destaque (fonte: [6]).

2.3.4 Relocalização

Nos casos em que a rotina de rastreamento falha, o ORB-SLAM2 utiliza o módulo de relocalização para tentar obter a posição da câmera. Nesse módulo, o sistema compara as imagens de entrada atuais com os *keyframes* já mapeados na tentativa de encontrar um *keyframe* que observe o mesmo ambiente visível na imagem atual.

Quando isso ocorre, o algoritmo não está mais executando a rotina de rastreamento, já que não consegue obter uma posição relativa as imagens já rastreadas, e, consequentemente, as rotinas de mapeamento local e fechamento de laço também permanecem em estado de espera até o robô percorrer um local já conhecido do ambiente [6, 5, 3].

O ORB-SLAM2 utiliza o método *bag of word* descrito na seção anterior para fazer a relocalização. Para tanto, a primeira etapa é converter a imagem de entrada para essa representação. Posteriormente, é realizada uma busca no banco de dados visando encontrar todos os *keyframes* similares a imagem atual.

Para encontrar esses candidatos, inicialmente, o sistema busca por todos os *keyframes* que compartilham pelo menos uma *visual word* com a imagem atual. Nessa etapa podem ser encontrados diversos *keyframes* candidatos para a relocalização, caso existam muitos lugares já mapeados que possuam a aparência similar a região observada pela imagem atual.

Posteriormente, são selecionados os candidatos que compartilham as maiores quantidades obtidas de *visual words,* para os quais será calculado um escore de similaridade entre a imagem atual e os candidatos selecionados, utilizando os vetores de *bag of words* que os representam. O escore de similaridade entre dois vetores de *bag of words,* $\mathbf{v}_1$ e $\mathbf{v}_2$, é calculado como [53]

$$s(\mathbf{v}_1, \mathbf{v}_2) = 1 - \frac{1}{2}\left|\frac{\mathbf{v}_1}{|\mathbf{v}_1|} - \frac{\mathbf{v}_2}{|\mathbf{v}_2|}\right|. \tag{2.9}$$

O sistema então busca pelo *keyframe* candidato com o maior escore de similaridade, s_{max}. A pontuação considerada para um *keyframe* é obtida somando o escore calculado para ele aos escores dos candidatos que estão conectados a ele no grafo de covisibilidade. Ao final, todos os *keyframes* que tiverem uma pontuação menor que $0.75 \cdot s_{max}$ são descartados.

Caso não seja encontrado nenhum candidato satisfatório, a busca é finalizada e o módulo de relocalização é executado novamente para a próxima imagem de entrada. Caso sejam encontrados candidatos satisfatórios, o sistema tenta calcular uma posição para a câmera a partir das correspondências encontradas.

Para os candidatos remanescentes, o algoritmo performa iterações do algoritmo RANSAC e tenta estimar a melhor posição utilizando o algoritmo PnP, como descrito em [55, 5]. Esse algoritmo permite a estimação da

posição da câmera a partir das correspondências 3D para 2D entre os pontos de mapa nos *keyframes* candidatos e as *features* da imagem atual. Um *keyframe* candidato é aceito para a relocalização somente se ao final desse procedimento houver pelo menos 50 correspondências entre os pontos de mapa rastreados por ele e as *features* da imagem atual [56, 5, 3].

Caso seja obtida uma posição para a câmera, o sistema retoma as rotinas de rastreamento, mapeamento local e fechamento de laço.

2.3.5 Mapeamento Local

Como mencionado anteriormente, o ORB-SLAM2 mantém um mapa local formado pelo conjunto de *keyframes* que compartilham pontos do mapa com o *keyframe* atual. A rotina de mapeamento local é responsável pelo processamento dos novos *keyframes* e pela atualização e constante otimização desse mapa local.

Inicialmente, a rotina de mapeamento local insere o novo *keyframe* no mapa local. Para tanto, o grafo de covisibilidade é atualizado, adicionando um novo nó para o novo *keyframe* recebido, K_i, e atualizando as arestas do grafo de acordo com os pontos de mapa compartilhados entre K_i e os outros *keyframes* pertencentes ao grafo. Depois, a árvore de dispersão é atualizada conectando K_i com o *keyframe* que ele possui mais pontos em comum. Por último, o sistema calcula a representação de K_i em *bag of words*, que irá auxiliar a etapa de associação de dados para a triangulação de novos pontos [5].

O próximo passo da rotina de mapeamento local é a seleção dos pontos do mapa adicionados recentemente. Para que esses pontos sejam mantidos no mapa, eles devem satisfazer testes restritivos para os três primeiros *keyframes* após a sua criação, que garantem que esses pontos serão rastreáveis e não serão erroneamente triangulados. Para que os pontos não sejam removidos do mapa, eles devem satisfazer as duas condições abaixo [5]:

- A rotina de rastreamento deve encontrar esses pontos em mais do que 25% das imagens em que seja previsto que eles sejam visíveis.
- Caso mais de um *keyframe* tenha sido adicionado ao mapa local após a inclusão do ponto no mapa, o ponto deve ser observado por pelo menos três *keyframes.*

Caso o ponto satisfaça esse teste, ele só poderá ser removido se em algum momento da execução ele for observado por menos que três *keyframes.* Isso pode acontecer quando *keyframes* redundantes são retirados do mapa e quando o BA local descarta observações discrepantes. É importante destacar que o ORB-SLAM2 não apaga de fato os pontos da memória do sistema, ele apenas marca esses pontos como inutilizados [5].

O terceiro passo da rotina de mapeamento local é a criação de novos pontos, que são criados através da triangulação das correspondências encontradas entre as *features* ORB de K_i com os *keyframes* conectados a K_i no grafo de covisibilidade, $\mathcal{K}_L$. O sistema calcula a triangulação apenas para as *features* que não são relacionadas ainda com um ponto do mapa. Nesse ponto, caso não haja nenhum novo *keyframe* processado pelo rastreamento esperando para ser inserido no mapa, o mapeamento local procura por pontos do mapa duplicados e funde esses pontos [5].

O próximo passo realizado na rotina de mapeamento local é a otimização, para tanto, o sistema executa um algoritmo de otimização que o autor denominou BA local. Esse algoritmo é utilizado para otimizar uma janela local de *keyframes* e pontos do mapa. O algoritmo de BA local otimiza um conjunto $\mathcal{K}_L$ de *keyframes*

que estão conectados com o K_i no grafo de covisibilidade e todos os pontos visíveis por eles, $\mathcal{P}_L$. Todos os outros *keyframes* $\mathcal{K}_F$ não pertencentes a $\mathcal{K}_L$ que observam pontos contidos em $\mathcal{P}_L$ contribuem para a função de custo, porém permanecem fixados na otimização.

Definindo $\mathcal{X}_k$ como o conjunto de correspondências entre os pontos em $\mathcal{P}_L$ e *keypoints* em um *keyframe* k, o problema de otimização é definido como [6]

$$\{\mathbf{X}^i, \mathbf{R}_l, \mathbf{t}_l \mid i \in \mathcal{P}_L, l \in \mathbf{K}_L\} = \underset{\mathbf{X}^i, \mathbf{R}_l, \mathbf{t}_l}{\operatorname{argmin}} \sum_{k \in \mathcal{K}_L \cup \mathcal{K}_F} \sum_{j \in \mathcal{X}_k} \rho(E_{kj}), \tag{2.10}$$

sendo

$$E_{kj} = \left\| \mathbf{x}^j_{(.)} - \pi_{(.)}(\mathbf{R}_k \mathbf{X}^j + \mathbf{t}_k \right\|^2_{\Sigma}. \tag{2.11}$$

Por fim, com o intuito de manter uma reconstrução compacta, a rotina de mapeamento local tenta detectar e remover *keyframes* redundantes. O sistema descarta todos os *keyframes* pertencentes a $\mathcal{K}_L$ que compartilharem pelo menos 90% dos seus pontos de mapa com pelo menos outros três *keyframes* [5].

2.3.6 Fechamento de Laço

O que diferencia o SLAM de outras abordagens de localização como a odometria visual (VO), do inglês *Visual Odometry*, é a criação de um mapa. Esse mapa não é criado somente com o intuito de mapear um ambiente, ele também permite o refinamento da estimação da posição de uma forma que as outras abordagens não permitem. Ao revisitar um local já mapeado e reconhecer que o robô já passou por aquele local, o algoritmo pode corrigir erros que foram acumulados com o tempo. Esse processo é conhecido como fechamento de laço [3].

A rotina de fechamento de laço do ORB-SLAM2 recebe o último *keyframe* processado pela rotina de mapeamento local, K_i, e executa em duas etapas. Inicialmente, um fechamento de laço deve ser detectado e validado, e, posteriormente, esse laço é corrigido através de uma otimização. O laço de execução da rotina de fechamento de laço está detalhado no Algoritmo 2.1.

O primeiro passo realizado pela rotina de fechamento de laço é a detecção dos *keyframes* candidatos para o fechamento de laço. Essa etapa é realizada caso o mapa possua pelo menos 10 *keyframes* e caso o último fechamento de laço tenha ocorrido a pelo menos 10 *keyframes* anteriores ao K_i. Inicialmente, o sistema calcula o escore de similaridade entre o vetor de *bag of words* de K_i e o de todos os seus vizinhos no grafo de covisibilidade ($\theta_{min} = 30$), utilizando a Equação 2.9, e armazena o menor escore calculado, s_{min}.

Posteriormente, é realizada uma busca do banco de dados para procurar por *keyframes* candidatos ao fechamento de laço entre as imagens rastreadas anteriormente, sendo descartados todos os *keyframes* cujos escores de similaridades são menores que s_{min}. Além disso, são descartados todos os *keyframes* conectados a K_i no grafo de covisibilidade.

Para que o sistema aceite um candidato de fechamento de laço que foi detectado no passo anterior, ele busca para cada um deles um conjunto consistente de três *keyframes* candidatos consecutivos que estejam conectados

Algoritmo 2.1 Laço de execução da rotina de Fechamento de Laço.

```
1: função FECHAMENTO_DE_LAÇO( )
2:     enquanto 1 faça
3:         se NovoKeyFrame então
4:             se DetectarLaço( ) então
5:                 se ComputarSIM3( ) então
6:                     CorrigirLaço( )
7:                 fim se
8:             fim se
9:         fim se
10:    fim enquanto
11: fim função
```

no grafo de covisibilidade. Durante a execução do ORB-SLAM2, podem existir vários candidatos para o fechamento de laço, caso existam muitos ambientes com a aparência similar a observada por K_i [5].

Em seguida, o sistema tenta validar a detecção do fechamento de laço com os *keyframes* candidatos encontrados. Inicialmente, o sistema computa as correspondências entre os descritores ORB associados com os pontos do mapa em K_i e os associados com os pontos do mapa nos *keyframes* candidatos, ou seja, aqui o sistema procura diretamente por correspondências 3D para 3D para cada candidato, diferentemente da correspondência realizada na rotina de relocalização.

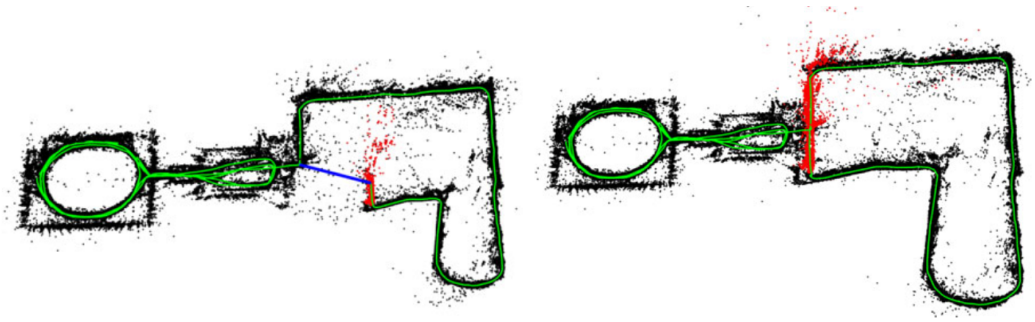
Quando o candidato apresenta uma quantidade satisfatória de correspondências, pelo menos 20, significa que o ORB-SLAM2 encontrou um par de pontos do mapa que correspondem ao mesmo ponto tridimensional do ambiente real. Porém, devido aos erros acumulados durante a execução, esses pontos provavelmente não estarão próximos no mapa construído apesar de representarem o mesmo ambiente. A Figura 2.12a ilustra essa situação, na qual a linha azul está conectado os pontos correspondentes encontrados.

No caso do SLAM monocular, existem sete graus de liberdade (DoFs), do inglês *Degrees of Freedom*, nos quais o mapa pode acumular erros com o passar do tempo: três de translação, três de rotação e um fator de escala [57]. Portanto, para fechar um laço, é necessário calcular a transformação de similaridade entre o *keyframe* atual, K_i, e um candidato, K_l, que irá fornecer os dados do erro acumulado durante a execução do laço. O cálculo da similaridade irá servir também como uma validação geométrica do laço [5].

A matriz de transformação de similaridade de um *keyframe* K_1 para um K_2, $\mathbf{S}_{K_1,K_2} \in \mathrm{Sim}(3)$, pode ser descrita como [57, 3]

$$\mathbf{S}_{K_1,K_2} = \begin{bmatrix} \delta_{K_1,K_2}\mathbf{R}_{K_1,K_2} & \mathbf{t}_{K_1,K_2} \\ 0 & 1 \end{bmatrix}, \tag{2.12}$$

em que $\delta_{K_1,K_2} \in \mathbb{R}^+$ é o fator de escala, $\mathbf{R}_{K_1,K_2} \in \mathrm{SO}(3)$ representa a matriz de rotação e $\mathbf{t}_{K_1,K_2} \in \mathbb{R}^3$ é o vetor de translação. $\mathrm{Sim}(3)$ e $\mathrm{SO}(3)$ são chamados de grupos de Lie, sendo $\mathrm{SO}(3)$ o grupo das rotações sobre a origem de um espaço tridimensional e $\mathrm{Sim}(3)$ o grupo das transformações de similaridade no espaço tridimensional, que é uma combinação de transformação de corpo rígido com um fator de escala [3, 5].



(a) (b)

Figura 2.12: Exemplo de um fechamento de laço durante a execução do SLAM. (a) exibe quando ocorre a detecção do laço e (b) ilustra a correção desse laço (fonte: [5]).

O ORB-SLAM calcula essa transformação de similaridade utilizando o método de Horn descrito em [58]. Esse método consegue obter a transformação entre dois sistemas de coordenadas dados pares de pontos correspondentes dos dois sistemas.

Posteriormente, o algoritmo verifica a transformação de similaridade. Ele checa o erro obtido quando os pontos são projetados no mapa após utilizá-la. Caso essa verificação encontre mais que 20 correspondências, a transformação é aceita.

Ao contrário do caso monocular, no qual erros de escala podem acontecer com o decorrer do tempo de execução [57], nos casos em que a câmera utilizada é estéreo ou RGB-D vão existir apenas seis DoFs. Nesses casos, a informação de profundidade torna a escala observável e a validação geométrica e a otimização do grafo de posições não precisam mais lidar com o desvio de escala e são baseadas em transformações de corpo rígido, em vez de similaridade [6]. Quando o sensor é estéreo ou RGB-D, o sistema considera que $\delta_{K_1,K_2} = 1$. Nesse caso, o grupo de transformações de corpo rígido no espaço tridimensional, $\mathrm{SE}(3)$, é representado por

$$\mathbf{T}_{K_1,K_2} = \begin{bmatrix} \mathbf{R}_{K_1,K_2} & \mathbf{t}_{K_1,K_2} \\ 0 & 1 \end{bmatrix} \in \mathrm{SE}(3). \tag{2.13}$$

O algoritmo então procede com uma busca guiada por mais correspondências entre pontos tridimensionais, com o auxílio da transformação de similaridade computada, e utiliza as correspondências encontradas para otimizar ainda mais a transformação. Dado um conjunto de n correspondências $i \Rightarrow j$ entre os *keypoints* associados a pontos tridimensionais do mapa observados pelo *keyframe* 1 e pelo *keyframe* 2, é desejado otimizar a transformação relativa $\mathbf{S}_{12}$ que minimiza o erro de reprojeção em ambas as imagens, definidos por [5]

$$\mathbf{e}_1 = \mathbf{x}_{1,i} - \pi_1(\mathbf{S}_{12}, \mathbf{X}_{2,j}), \tag{2.14}$$

$$\mathbf{e}_2 = \mathbf{x}_{2,j} - \pi_2(\mathbf{S}_{12}^{-1}, \mathbf{X}_{1,i}), \tag{2.15}$$

nos quais $\mathbf{X} \in \mathbb{R}^3$ representa a localização dos pontos tridimensionais do mapa e $\mathbf{x}$ representa os *keypoints.* A função de custo a ser minimizada é dada por [5]

$$C = \sum_{n} (\rho_h(\mathbf{e}_1^T \mathbf{\Omega}_{1,i}^{-1} \mathbf{e}_1) + \rho_h(\mathbf{e}_2^T \mathbf{\Omega}_{2,j}^{-1} \mathbf{e}_2)), \tag{2.16}$$

na qual $\mathbf{\Omega}_{1,i}$ e $\mathbf{\Omega}_{2,j}$ são as matrizes de covariância associadas com a escala na qual os *keypoints* foram detectados nas imagens 1 e 2 [5]. Caso pelo menos 20 correspondências sejam encontradas após a otimização, o candidato é aceito para uma última verificação [3].

Agora, o algoritmo executa outra busca por mais correspondências entre pontos do mapa. Isso é feito através da projeção dos pontos de mapa encontrados no fechamento de laço dos *keyframes* que estão conectados aos candidatos. Se após essa busca pelo menos 40 correspondências forem obtidas, esse candidato é aceito e a otimização dos outros candidatos é interrompida. Em seguida, o sistema procede para a correção do laço.

Nesse ponto, o fechamento de laço já foi detectado e validado, o último passo realizado pela rotina de fechamento de laço é a correção do laço. Nessa etapa são fundidos pontos do mapa duplicados e são inseridos novas conexões no grafo de covisibilidade que irão vincular o fechamento de laço. Para que isso seja feito, inicialmente, a rotina de mapeamento local é interrompida para que ela não modifique o mapa durante a otimização [3, 5].

Depois, as conexões do grafo de covisibilidade são atualizadas de forma que o K_i seja conectado aos *keyframes* que estão na localização ao redor do candidato ao fechamento de laço. O algoritmo então procede para a correção da posição. Primeiro, a posição de K_i é corrigida utilizando a matriz de similaridade calculada e essa correção é propagada para os *keyframes* que estão conectados ao K_i, de forma que os dois lados do laço sejam alinhados. Além disso, todos os pontos de mapa relacionados aos *keyframes* modificados são corrigidos [5].

Para realizar efetivamente o fechamento de laço, o próximo passo é a propagar a correção realizada através do grafo essencial, que distribui o erro do fechamento de laço ao longo do grafo. Nesse passo, o sistema executa uma otimização no grafo de posições sobre as transformações de similaridade para corrigir os desvios de escala, no caso monocular [5]. Dado um grafo de posições com arestas binárias, o erro em uma aresta está definido como [5]

$$\mathbf{e}_{i,j} = \log_{\mathrm{Sim}(3)}(\mathbf{S}_{ij} \mathbf{S}_{jw} \mathbf{S}_{iw}^{-1}), \tag{2.17}$$

no qual $\mathbf{S}_{ij} \in \mathrm{Sim}(3)$ é a transformação relativa entre os dois *keyframes,* calculada antes da otimização do grafo e com o fator de escala igual a 1. E o subíndice w representa as transformações em relação às coordenadas do mundo. No caso da aresta do fechamento do laço, essa relação é obtida através do método de Horn [5].

O objetivo é otimizar a as posições dos *keyframes*, sendo a função de custo dada por [5]

$$C = \sum_{i,j} (\mathbf{e}_{i,j}^T \mathbf{\Lambda}_{i,j} \mathbf{e}_{i,j}), \tag{2.18}$$

em que $\mathbf{\Lambda}_{i,j}$ é a matriz de informação da aresta, definida como uma matriz identidade, como feito em [59].

O último passo da rotina de fechamento de laço é a execução de uma otimização utilizando o BA. Para tanto, no ORB-SLAM2 foi incorporado uma otimização com um algoritmo denominado pelos autores de BA total. Este é um caso específico do BA local, descrito na subseção anterior, no qual todos os *keyframes* e pontos do mapa são otimizados, com exceção do *keyframe* da origem que é fixado com o intuito de obter uma solução

ótima. Essa otimização pode ser computacionalmente custosa e, portanto, ela é realizada em uma execução em paralelo, enquanto o sistema continua a criação do mapa e a detecção de novos fechamentos de laços [6].

Entretanto, isso gera o desafio de fundir o resultado da otimização com o estado atual do mapa. Caso uma nova detecção de fechamento de laço seja realizada enquanto a otimização ainda está em execução, essa execução é interrompida e o sistema procede para o novo fechamento de laço, que irá executar o algoritmo de otimização novamente.

Quando a execução do BA total termina, o sistema precisa fundir o subconjunto atualizado de *keyframes* e pontos do mapa que foram otimizados com os *keyframes* e pontos não atualizados que foram inseridos enquanto a otimização estava em execução. Isso é realizado através da propagação da correção realizada nos *keyframes* otimizados para os *keyframes* não atualizados através da árvore de dispersão. Pontos que não foram atualizados são transformados de acordo com a correção realizada no seu *keyframe* de referência [6].

3 METODOLOGIA PROPOSTA

3.1 INTRODUÇÃO

O capítulo anterior deste trabalho descreveu o ORB-SLAM2, apresentando a estrutura desse algoritmo de SLAM visual, ilustrada na Figura 2.5. Além disso, foram detalhados os conceitos teóricos utilizados na sua implementação. Devido aos bons resultados do ORB-SLAM2 encontrados na literatura [5, 6, 3, 21], ele é considerado atualmente o estado da arte e a solução mais completa baseada em *features* para o SLAM visual [21]. Por isso e devido ao fato desse algoritmo ser uma solução de código livre, ele foi escolhido como base para a implementação da proposta apresentada neste trabalho.

Apesar das vantagens e sucesso de aplicação do ORB-SLAM2 mencionados por [5, 6, 3, 21], um dos problemas em aberto das soluções de SLAM visual é a perda de referência durante a rotina de rastreamento devido a perda momentânea de qualidade dos dados de entrada [36, 5]. Isso ocorre quando o robô não é capaz de calcular sua posição, já que não consegue mais se localizar no ambiente.

Com o intuito de contornar esse problema, as soluções de SLAM visual presentes na literatura utilizam um módulo de reconhecimento de local para a relocalização no ambiente [21, 2]. Porém, para que seja possível prosseguir com o rastreamento, é necessário esperar uma imagem de entrada que observe um ambiente previamente mapeado para se relocalizar no mapa e voltar a executar as rotinas do algoritmo de SLAM visual.

Quando isso ocorre, caso ele não passe por um local já conhecido, o robô não consegue mais se posicionar no mapa já construído e nem retomar o mapeamento do ambiente. E no caso em que o robô consegue se relocalizar, ele deixa de mapear uma parte do ambiente.

Isso pode acontecer quando o robô se movimenta abruptamente e a nova imagem de entrada não possui *features* em comum com as mapeadas anteriormente. Como nesse caso o único sensor que está obtendo informações do ambiente é a câmera, e o robô não possui um conhecimento prévio do ambiente, ele não é mais capaz de se localizar ou de continuar o mapeamento, uma vez que, ele não é capaz de encaixar essa nova imagem no mapa que estava sendo construído.

Na nova abordagem proposta neste trabalho, pretende-se resolver o problema da perda do mapeamento do ambiente devido a perda momentânea de qualidade dos dados de entrada. Além desse problema, é proposta uma solução para o mapeamento colaborativo, ou seja, o problema de permitir que um conjunto de robôs possa realizar o mapeamento de um ambiente desconhecido de forma cooperativa. Para tanto, cada robô mapeia o ambiente executando o ORB-SLAM2 modificado proposto neste trabalho e compartilha suas instâncias de mapa com os outros robôs da rede. Ademais, cada robô executa um algoritmo para fazer a união das instâncias de mapas recebidas, caso estas possuam regiões em comum.

Nas seções seguintes deste capítulo, a solução proposta será apresentada de forma mais detalhada. Na seção 3.2 será descrita a estrutura da solução proposta. E nas seções posteriores, 3.3 e 3.4, será detalhada, respectivamente, a implementação da abordagem de múltiplos mapas baseada no ORB-SLAM2 e da expansão dessa abordagem para uma configuração distribuída para o SLAM visual.

3.2 ESTRUTURA DA SOLUÇÃO PROPOSTA

Para um melhor entendimento do leitor sobre a estrutura da solução apresentada neste trabalho, inicialmente, esta seção irá apresentar a ferramenta utilizada para a implementação do algoritmo desenvolvido. Posteriormente, a estrutura do algoritmo desenvolvido será explanada.

3.2.1 *Robot Operating System* (ROS)

Para a implementação da solução proposta foi utilizado o Sistema Operacional de Robôs (ROS), do inglês *Robot Operating System*. O ROS é um sistema flexível para o desenvolvimento de programas que disponibiliza bibliotecas e ferramentas que auxiliam os desenvolvedores na criação de aplicações para a robótica. Ele foi desenvolvido com o intuito de facilitar a integração das várias funcionalidades do robô e disponibiliza e reutiliza bibliotecas e simuladores, como o *Stage* e o Gazebo [60].

O ROS tem como objetivo ser uma plataforma ponto a ponto, baseada em ferramentas, multilíngue, leve, grátis e de código livre [60]. Um sistema construído utilizando o ROS consiste de um número de processos, que podem estar em diferentes hospedeiros, conectados em uma topologia ponto-a-ponto. Esse tipo de topologia requer algum tipo de mecanismo de administração que permita que os processos possam se localizar durante a execução. No ROS, esse mecanismo é denominado *master.*

Outros elementos fundamentais que compõem a estrutura do ROS são os pacotes, os nós, as mensagens e os tópicos [60]. O ROS organiza o *software* em pacotes, sendo que cada pacote é a menor unidade individual funcional que pode ser criada com o ROS e estes são compilados e processados individualmente. Um pacote é basicamente um diretório que possui um arquivo do tipo XML que o descreve. Esse diretório pode conter nós, bibliotecas, bancos de dados, arquivos de configuração ou qualquer outro tipo de arquivo que constituir um módulo útil.

Os nós são as unidades que executam alguma computação. O ROS foi desenvolvido para ser modular, de modo que um sistema é geralmente composto por vários nós. Sendo que esses nós executam de forma independente um dos outros. Além disso, eles tentam ocorrer de maneira paralela.

Na implementação deste trabalho foi utilizado o paradigma de publicar e assinar, do inglês *publisher e subscriber*, para a coordenação entre os nós. Neste paradigma, existe um nó que publica uma informação no formato de uma mensagem em um tópico. Se algum outro nó desejar obter aquela informação, ele deve assinar aquele tópico. Quando um nó é um assinante de um tópico, ele é informado sempre que uma mensagem nova é publicada nele.

As mensagens são configuradas como uma estrutura de dados pré-definida. Essas mensagens podem ser formadas por dados de um tipo padrão primitivo como um inteiro, ponto flutuante, booleanos, entre outros, além de vetores, outras mensagens e vetores de outras mensagens.

Os tópicos são as estruturas que armazenam as mensagens publicadas, eles são configurados pelo nome e pelo tipo de mensagem publicada. Um único nó pode publicar ou ser assinante em múltiplos tópicos. Além disso, podem existir múltiplos nós assinando ou publicando em um mesmo tópico. De maneira geral, os nós que publicam ou assinam um tópico não estão cientes da existência um dos outros [60].

3.2.2 Arquitetura da Implementação

A Figura 3.1 exibe um diagrama geral da estrutura da solução proposta. Nesse caso, a imagem ilustra uma situação exemplo na qual dois robôs estariam mapeando o ambiente. Como pode ser visualizado na figura, o sistema desenvolvido funciona com cada robô executando dois algoritmos principais, que foram denominados de "ORB-SLAM2 modificado" e de "Unifica_Mapas". Na arquitetura proposta, em cada um dos robôs, cada um desses algoritmos executa de forma paralela em um nó do ROS.

A separação dos dois algoritmos em nós diferentes foi realizada após ser observado que isso melhorava a eficiência do processamento do algoritmo. Essa melhoria foi verificada ao comparar a execução do sistema com essa estrutura em relação a uma implementação na qual a rotina de unificação dos mapas executasse em uma *thread* paralela no mesmo nó. Ademais, essa estrutura em dois nós facilitou expandir o sistema para a execução em múltiplos robôs.

Além disso, as imagens de entrada do sistema de cada um dos robôs e as instâncias de mapas, representados no diagrama da Figura 3.1, foram estruturadas como mensagens. Essas informações são enviadas e recebidas através de tópicos do ROS. Assim, esses dados podem ser acessados facilmente pelos nós que desejem obter essa informação.

Devido a sua estrutura de funcionamento, a abordagem proposta é bastante flexível, permitindo que o usuário configure o SLAM visual com quantos robôs desejar. Para isso, basta fazer com que os nós de mapeamento dos robôs publiquem as suas instâncias de mapas em tópicos, de modo que os outros robôs possam assiná-los com o intuito de receber essas informações para construir um mapa unificado. Na seção seguinte será descrito o funcionamento dos dois nós que formam a estrutura da solução proposta neste trabalho.

3.3 ABORDAGEM DE MÚLTIPLOS MAPAS PARA O SLAM VISUAL

Esta seção irá salientar os detalhes da abordagem de múltiplos mapas desenvolvida para o SLAM Visual. Os dois nós implementados para a construção da solução serão detalhados. Inicialmente, serão apresentadas as modificações realizadas no ORB-SLAM2. Em seguida, será explicado o nó "Unifica_mapas".

3.3.1 ORB-SLAM2 Modificado

A solução proposta neste trabalho para uma abordagem de múltiplos mapas do SLAM visual foi baseada no ORB-SLAM2. Para realizar o mapeamento do ambiente de maneira eficaz, foram mantidas as rotinas de rastreamento, mapeamento local e fechamento de laço executando em paralelo. Além disso, foi utilizada a mesma implementação para a relocalização. Para tanto, a implementação de código livre desses módulos fornecida pelos autores do ORB-SLAM2 foi utilizada, adaptando a implementação para ser executada no ROS. A estrutura dessas rotinas e os métodos utilizados por elas foram apresentadas de forma mais aprofundada no capítulo anterior.

A Figura 3.2 exibe o diagrama de blocos do algoritmo ORB-SLAM2 modificado. Como pode ser visualizado no diagrama, o robô irá capturar as imagens com a câmera, que serão passadas inicialmente para a rotina de rastreamento. Essa rotina, herdada do ORB-SLAM2, é responsável pela estimação da localização da câmera em cada imagem de entrada e pela decisão de quando inserir um novo *keyframe* no mapa, como esclarecido na

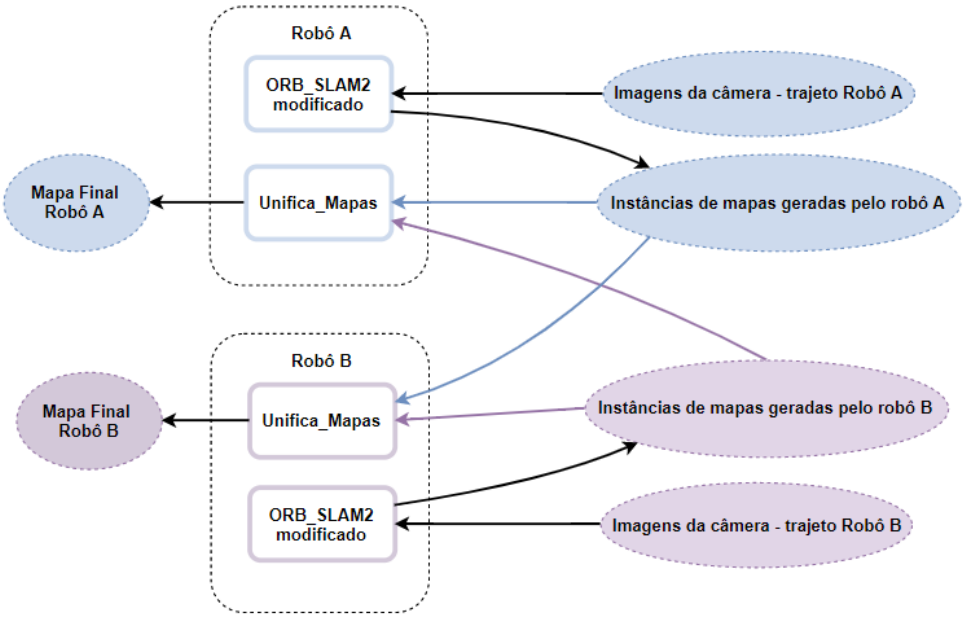


Figura 3.1: Diagrama geral da estrutura de funcionamento da abordagem proposta em um exemplo para dois robôs.

subseção 2.3.3.

Quando o rastreamento não é capaz de estimar a localização, o sistema tenta se relocalizar no mapa que ele já construiu do ambiente. Isso ocorre quando não forem detectadas *features* em comum entre a imagem atual de entrada e as mapeadas previamente. Nesse caso, a relocalização tenta estimar a posição do robô, verificando se ele já percorreu aquela região em algum dos lugares já mapeados.

A partir do rastreamento, o novo *keyframe* processado é repassado para a rotina de mapeamento local. O mapeamento local é responsável por inserir esses *keyframes* no mapa, criar novos pontos do mapa e otimizar o mapa local. Além disso, ele é responsável por detectar e excluir as imagens redundantes.

Ademais, têm-se a rotina de fechamento de laço, que procura por fechamento de laços a cada nova imagem chave recebida. Ela verifica se o *keyframe* atual observa uma região em comum com alguma outra imagem já mapeada anteriormente. Se for detectado um laço, é executado um algoritmo com o intuito de reduzir o erro acumulado durante a execução do SLAM entre os dois pontos do laço.

Uma das principais contribuições da solução com múltiplos mapas apresentada neste trabalho é o fato de que o robô não irá perder dados do ambiente. Isso ocorre quando ele muda de direção de maneira abrupta durante a sua locomoção e o rastreamento para de funcionar, problema já exposto em trabalhos anteriores [36].

Para tanto, a modificação realizada no ORB-SLAM2 foi que quando o rastreamento se perde e a relocalização falha em estimar a posição da câmera nas próximas λ imagens de entrada, com $\lambda \in \mathbb{R}^+$, o sistema para de tentar se relocalizar e começa a construir um novo mapa. Além disso, o mapa construído anteriormente, caso possua mais que 10 *keyframes*, é serializado e publicado em um tópico do ROS em forma de uma mensagem, como será esclarecido no tópico 3.3.2.

Essa modificação foi realizada pois, ao mapear ambientes extensos, quando o robô perde a sua referência

de posição, pode ser que ele precise percorrer uma parte considerável do ambiente até encontrar uma região que já foi mapeada anteriormente para conseguir se relocalizar. Além disso, em alguns casos, o robô não consegue mais retomar o mapeamento. E ele não considera todos os dados do ambiente que ele percorreu com o rastreamento desligado.

Na solução proposta, todo o ambiente percorrido pelo robô será mapeado, mesmo que ele perca o rastreamento. Por fim, caso as instâncias de mapa construídas possuam regiões em comum do ambiente, esses mapas serão unificados pelo outro nó que está executando no sistema.

3.3.2 Conversão e Envio das Instâncias de Mapas

As instâncias de mapas construídas pelo robô no decorrer da execução do SLAM visual serão armazenadas e gerenciadas pelo nó "Unifica_mapas", que será detalhado na subseção 3.3.3. Porém, para que esse nó obtenha as informações contidas nos mapas, foi utilizado o paradigma do ROS de publicar e assinar.

Na arquitetura do sistema implementado, o nó "ORB-SLAM2 modificado" inicializa a construção de um novo mapa sempre que o rastreamento se torna incapaz de estimar a localização. Quando isso ocorre, o mapa anterior não é descartado, já que o intuito deste sistema é evitar perder esses dados do ambiente. Ao invés disso, esse mapa é enviado para o outro nó que executa em paralelo, o "Unifica_mapas".

Para que ocorra o envio dos dados, as informações serializadas do mapa M_i, com $i \in \mathbb{N}^+$, são publicadas em um tópico τ_R no formato de uma mensagem do ROS pelo nó "OBR-SLAM2 modificado". E o nó "Unifica_mapas" deverá ser um assinante desse tópico, para que ele seja notificado sempre que um mapa i seja enviado por esse tópico.

Como mencionado anteriormente, as mensagens do ROS possuem uma estrutura pré-definida de dados. Para que o paradigma de publicar e assinar pudesse ser utilizado no sistema desenvolvido, a instância de mapa a ser enviada precisou ser serializada para que estivesse adequada ao formato de uma mensagem.

No contexto de armazenamento e envio de dados, serializar é o processo de traduzir estruturas de dados em um formato que possa ser armazenado, por exemplo, em um arquivo ou em um espaço de memória, e possam ser reconstruídos posteriormente. A operação oposta para obter esses dados novamente é denominada de desserialização [61].

Uma instância de mapa é um objeto formado por um conjunto de *keyframes* e de pontos do mapa que representam o ambiente. Para serializar esse dados em um formato de texto foi utilizado o padrão JSON [62, 63], do inglês *JavaScript Object Notation*. De modo que para transmitir as informações do mapa, apenas uma mensagem composta por um elemento do tipo *string* precisa ser publicada no tópico τ_R.

A Figura 3.3 exibe um esquemático de como está implementada a troca de informações entre os nós do sistema. Para que o algoritmo que executa o SLAM prossiga com o mapeamento, é criada temporariamente uma rotina em paralelo que transforma o mapa M_i em um objeto JSON equivalente. Esse objeto JSON é facilmente convertido para a *string* m_i, que armazena as informações do mapa. Após ser publicado, os dados do M_i não ficam mais disponíveis para o ORB-SLAM2 modificado.

Quando o nó "Unifica_Mapas" recebe a *string* m_i ele executa uma rotina para desserializar esses dados para o formato original. Inicialmente, m_i é convertida para um objeto JSON. Em seguida, é obtido o formato original de M_i, que é armazenado no formato de um objeto de uma classe, programado na linguagem C++ [64].

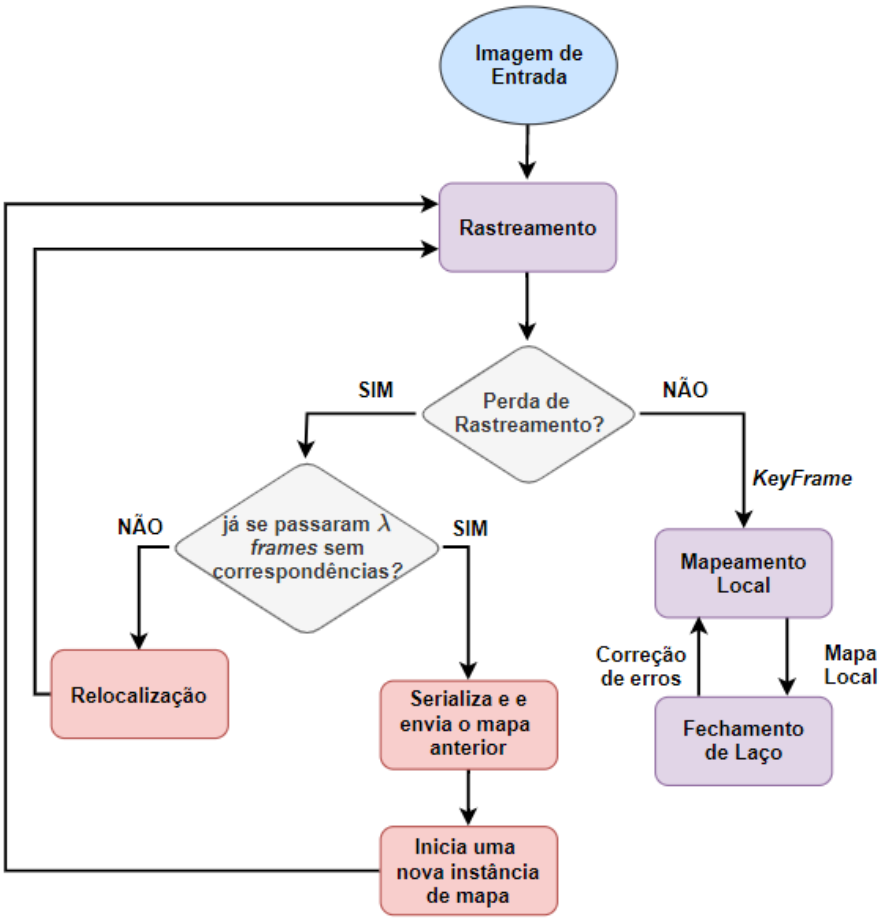


Figura 3.2: Diagrama de blocos do ORB-SLAM2 modificado para a abordagem de múltiplos mapas. Em destaque em roxo estão as rotinas principais que executam em paralelo e em vermelho estão os blocos que foram modificados.

Figura 3.3: Esquemático do paradigma de publicar e assinar implementado para a troca de informações entre os nós do sistema.

3.3.3 Unifica_Mapas

O outro algoritmo que executa no sistema, paralelamente ao SLAM visual, é o nó "Unifica_Mapas". Esse nó é responsável por armazenar e gerenciar as instâncias de mapas recebidas pelo robô através de um tópico τ_R. Esse algoritmo possui um ciclo de execução de cinco etapas principais, que aguarda o recebimento de um mapa para iniciar. O Algoritmo 3.1 descreve esse ciclo de execução cujo o digrama de blocos é exibido na Figura 3.4.

Algoritmo 3.1 Ciclo de execução do nó "Unifica_Mapas".

```
1:  função UNIFICA_MAPAS( )
2:      enquanto 1 faça
3:          se NovoMapa( ) então
4:              M_i ← DesserializaMapa(m_i)
5:              SalvaMapa(M_i)
6:              para k ← 0 até QUANTIDADE_MAPAS_SALVOS faça
7:                  se DetectarSobreposiçãoEntreMapas(M_i, M_k) então
8:                      se ComputarPosiçãoRelativa(M_i, M_k) então
9:                          UnificaMapas(M_i, M_k)
10:                     fim se
11:                 fim se
12:             fim para
13:         fim se
14:     fim enquanto
15: fim função
```

Como no decorrer da execução do SLAM visual é possível que sejam construídas algumas instâncias de mapas, esse segundo nó foi implementado para que o sistema fosse capaz de associar essas instâncias caso elas possuam regiões em comum. Dessa forma, essa nova proposta de SLAM é capaz de obter um mapa final que abrange uma região maior do ambiente percorrido pelo robô do que o que seria obtido com apenas a execução do ORB-SLAM2.

Como pode ser visualizado no diagrama da Figura 3.4, o primeiro passo do ciclo de execução desse nó é a desserialização do mapa M_i, após o recebimento dos dados de entrada m_i. O método utilizado nessa etapa foi explicado com mais detalhes no tópico 3.3.2. Após a obtenção de M_i no formato desejado, este é armazenado em um banco de dados para ser utilizado posteriormente.

As próximas três etapas do algoritmo consistem em detectar se o mapa de entrada atual M_i pode ser concatenado com algum outro já existente no banco de dados. Em caso afirmativo, tentar calcular a posição relativa entre os dois mapas. E por último, unificar os mapas. Esses próximos três passos executam em um ciclo para cada *keyframe* $K_A \in \mathcal{K}_I$, sendo $\mathcal{K}_I$ o conjunto de todos os *keyframes* que formam o mapa de entrada atual, M_i.

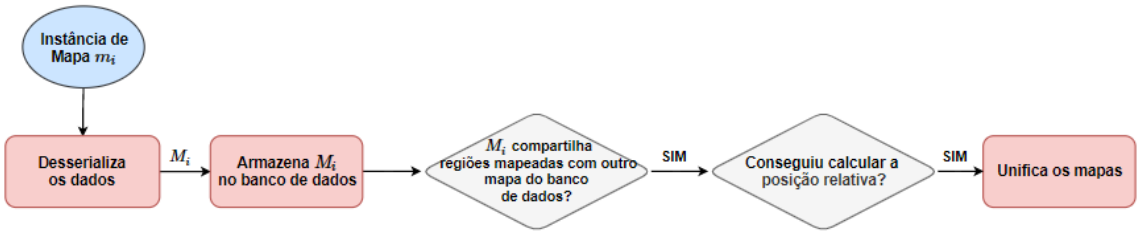


Figura 3.4: Diagrama de blocos do nó “Unifica_Mapas”.

Detecção de sobreposição entre dois mapas

Após a desserialização de M_i, caso o banco de dados já possua instâncias de mapas armazenadas, o sistema verifica se o mapa M_i compartilha regiões mapeadas com algum dos mapas recebidos anteriormente. É importante ressaltar que todos os mapas armazenados terão o seu próprio sistema de coordenadas.

Como todos os dados obtidos do ambiente foram através da câmera, os mapas são construídos através do cálculo da posição relativa de uma sequência de imagens. Quando o rastreamento se perde e inicia um novo mapa, este vai possuir um próprio sistema de coordenas, com o primeiro *keyframe* mapeado fixado na origem.

O próximo passo então é verificar se existe alguma sobreposição entre M_i e algum dos mapas do banco de dados, M_k, com $k \in \mathbb{N}$, $k \in [0, i-1]$. Inicialmente, o sistema confere se algum dos *keyframes* pertencentes a M_i observam uma região visualmente parecida com alguma região observada pelos *keyframes* de M_k.

Seja $\mathcal{K}_K$ o conjunto dos *keyframes* $K_C \in \mathcal{K}_K$ pertencentes ao mapa M_k que são considerados candidatos para serem o ponto de fusão entre os dois mapas. Inicialmente, esse conjunto é formado por todos os *keyframes* contidos em M_k.

Para cada *keyframe* K_A pertencente ao mapa recebido atual, o sistema percorre todos os elementos de $\mathcal{K}_K$ em busca de candidatos correspondentes a K_A. Isso é feito computando, para cada K_C, as correspondências entre os descritores ORB associados aos pontos do mapa observados por ele e os descritores ORB associados com os pontos do mapa observados por K_A.

Essa correspondência é obtida através da distância calculada entre os dois descritores ORB, caso essa distância esteja abaixo de um limiar pré-definido, os descritores são considerados correspondentes. Ou seja, o sistema procura por correspondências visuais entre os pontos tridimensionais do mapa associados aos *keyframes* de ambos os mapas.

Em seguida, um *keyframe* K_C continuará a ser considerado um candidato válido para a unificação dos mapas caso ele passe por um teste de consistência. Esse teste verifica se K_C pertence a um conjunto formado por pelo menos três *keyframes* seguidos de M_k, no qual cada *keyframe* possui no mínimo 35 pontos correspondentes com K_A. Caso ele não passe nessa verificação, ele é removido do conjunto $\mathcal{K}_K$. Se sobrar algum candidato válido, o sistema procede para o próximo passo: utilizar essas correspondências para tentar calcular uma posição relativa entre os mapas.

Cálculo da posição relativa entre dois mapas

O objetivo desta etapa é obter uma posição relativa entre algum dos candidatos validados no passo anterior e o *keyframe* K_A. A partir dessa relação, será possível unificar os mapas na etapa seguinte.

Para calcular uma posição relativa entre os mapas, o sistema tenta obter uma matriz de transformação de similaridade $\mathbf{S}_{K_A,K_C} \in \mathrm{Sim}(3)$ de K_A para cada candidato K_C. Essa matriz é calculada utilizando o método de Horn, descrito em [58].

A matriz de transformação de similaridade é dada pela equação 2.12, nesse caso:

$$\mathbf{S}_{K_A,K_C} = \begin{bmatrix} \delta_{K_A,K_C}\mathbf{R}_{K_A,K_C} & \mathbf{t}_{K_A,K_C} \\ 0 & 1 \end{bmatrix}, \tag{3.1}$$

em que $\delta_{K_A,K_C} \in \mathbb{R}^+$ é o fator de escala, sendo $\delta_{K_A,K_C} = 1$ quando as imagens forem capturadas por câmeras estéreo ou RGB-D. $\mathbf{R}_{K_A,K_C} \in \mathrm{SO}(3)$ representa a matriz de rotação e $\mathbf{t}_{K_A,K_C} \in \mathbb{R}^3$ é o vetor de translação.

No passo anterior, foram obtidas as correspondências entre as *features* dos pontos do mapa observados por K_A e por K_C. Logo, nesse ponto, o sistema possui pares correspondentes de pontos tridimensionais do mapa para cada candidato K_C. Para o cálculo da matriz de transformação entre esses pontos, o sistema performa iterações do algoritmo RANSAC e tenta calcular a melhor solução para resolver um problema P3P [65], do inglês *Perspective-3-Point*.

A partir de agora são considerados apenas os candidatos para os quais o sistema foi capaz de obter uma matriz $\mathbf{S}_{K_A,K_C}$. O algoritmo então procede para uma busca guiada de mais correspondências entre pontos tridimensionais do mapa entre K_A e os candidatos restantes. Para isso, ele utiliza a $\mathbf{S}_{K_A,K_C}$ obtida para cada candidato.

Por fim, o sistema otimiza a matriz $\mathbf{S}_{K_A,K_C}$ considerando as novas correspondências encontradas. Essa otimização é realizada com intuito de minimizar o erro de reprojeção desses pontos, definidos pelas Equações 2.14 e 2.15, com a função de custo dada pela Equação 2.16. Caso ao final da otimização permaneçam pelo menos 35 correspondências, o candidato é escolhido para a unificação dos mapas.

Ao final dessa etapa, caso ocorra com sucesso, o sistema possui uma matriz de transformação de similaridade $\mathbf{S}_{K_A,K_C}$ que relaciona um *keyframe* K_C com o K_A. Com essa matriz será realizada a unificação dos mapas na próxima etapa.

Unificação dos mapas

Nesse passo, o sistema irá, inicialmente, converter as posições dos *keyframes* que compõem o mapa de entrada atual do nó "Unifica_Mapas", M_i. Para isso, será utilizada a matriz de transformação de similaridade computada no passo anterior.

Esse nó armazena todas as instâncias de mapas recebidas desde o início da execução do sistema, organizando essas instâncias pela ordem na qual elas foram recebidas. Sempre que um novo mapa M_i é recebido, caso ele apresente uma região mapeada em comum com um outro mapa M_k recebido anteriormente, esses mapas serão unificados. Nesta abordagem foi definido que o mapa mais atual recebido M_i será convertido para o sistema de coordenadas de M_k para que eles possam ser unificados.

Desse modo, supondo que M_1 e M_2 sejam, respectivamente, o primeiro e o segundo mapa recebidos. Se eles forem unificados, eles formarão um novo mapa: $M_1' = M_1 \cup M_2'$, que está no sistema de coordenadas de M_1. Quando for recebido um terceiro mapa M_3, caso ele só possua correspondência com M_2, ele será unificado com M_1' e será convertido para o seu sistema de coordenadas. Logo, ao final da execução, todos os mapas estão no mesmo sistema de coordenadas e servirão como uma representação mais abrangente do ambiente mapeado pelo robô.

O primeiro passo para a conversão dos *keyframes* do mapa M_i é a obtenção da posição corrigida de K_A para o sistema de coordenadas do mapa M_k. Para isso, a posição de K_C é convertida de $\mathbf{T}_{K_C w_k} \in \mathrm{SE}(3)$ para a matriz de transformação de similaridade $\mathbf{S}_{K_C,w_k} \in \mathrm{Sim}(3)$. Sendo $\mathbf{T}_{K_C w_k} \in \mathrm{SE}(3)$ uma transformação de corpo rígido, que transforma do sistema de coordenadas do mapa M_k para o sistema de coordenadas da câmera, para o *keyframe* K_C. Essa conversão é realizada através da relação

$$\mathbf{S}_{K_C,w_k} = \begin{bmatrix} s_{K_C,w_k}\mathbf{R}_{K_C,w_k} & \mathbf{t}_{K_C,w_k} \\ 0 & 1 \end{bmatrix} \rightarrow \mathbf{T}_{K_C,w_k} = \begin{bmatrix} \mathbf{R}_{K_C,w_k} & \dfrac{\mathbf{t}_{K_C,w_k}}{s_{K_C,w_k}} \\ 0 & 1 \end{bmatrix}. \tag{3.2}$$

Sabendo que a matriz $\mathbf{S}_{K_A,K_C}$ fornece a posição relativa do *keyframe* K_A em relação ao K_C, para obter a posição corrigida de K_A para as coordenadas do mundo do mapa M_k, $\mathbf{S}_{K_A,w_k} \in \mathrm{Sim}(3)$, é realizada a seguinte operação:

$$\mathbf{S}_{K_A,w_k} = \mathbf{S}_{K_A,K_C} \cdot \mathbf{S}_{K_c,w_k}. \tag{3.3}$$

Em seguida, o sistema converte os *keyframes* $K_j \in \mathcal{K}_I$, com $j \neq A$, para o novo sistema de coordenadas. Para isso, primeiro, é computada a matriz $\mathbf{S}_{j,w_i} \in \mathrm{Sim}(3)$ a partir da transformação $\mathbf{T}_{j,w_i} \in \mathrm{SE}(3)$ de cada *keyframe* j, mantendo a rotação e a translação e fixando a escala em 1. Sendo $\mathbf{T}_{j,w_i} \in \mathrm{SE}(3)$ é a posição armazena em K_j, que representa uma transformação de corpo rígido que transforma pontos das coordenadas do mundo para as coordenadas da câmera, com w_i representando o subíndice que indica o sistema de coordenadas do mundo do mapa M_i.

As posições dos *keyframes* $K_j \in \mathcal{K}_I$ no novo sistema de coordenadas, $\mathbf{S}_{j,w_k} \in \mathrm{Sim}(3)$, são calculadas como

$$\mathbf{S}_{j,w_k} = \mathbf{S}_{j,K_A} \cdot \mathbf{S}_{K_A,w_k}, \tag{3.4}$$

com

$$\mathbf{T}_{j,K_A} = \mathbf{T}_{j,w_i} \cdot \mathbf{T}_{w_i,K_A} \tag{3.5}$$

e

$$\mathbf{T}_{w_i,K_A} = (\mathbf{T}_{w_i,K_A})^{-1}. \tag{3.6}$$

Por fim, a matriz $\mathbf{S}_{j,w_k} \in \mathrm{Sim}(3)$ é convertida para a posição $\mathbf{T}_{j,w_k} \in \mathrm{SE}(3)$, utilizando a relação dada pela Equação 3.2.

Após calculadas as novas posições dos *keyframes* pertencentes a $\mathcal{K}_I$, o sistema precisa corrigir as posições dos pontos tridimensionais do mapa associados a eles. Cada ponto do mapa, p_n, armazena a informação da sua

posição 3D nas coordenadas do mundo, $\mathbf{X}_{w_i,n}$. Além disso, eles são associados a um *keyframe* K_j, para o qual foi calculada uma posição $\mathbf{T}_{j,w_k} \in \mathrm{SE}(3)$. Para cada ponto, p_n, a conversão da posição para as coordenadas do mapa M_k é calculada como

$$\mathbf{X}_{w_k,n} = (S_{j,w_k})^{-1} \cdot \mathbf{T}_{j,w_k} \cdot \mathbf{X}_{w_i,n}. \tag{3.7}$$

Após o cálculo das novas posições dos pontos p_n, a instância de mapa M_i já está nas mesmas coordenadas do mapa M_k. Portanto, esses mapas podem ser unificados para uma representação mais abrangente do ambiente. O último passo para a concatenação dos mapas é a execução do algoritmo de otimização BA total na instância de mapa que teve as suas coordenadas modificadas. Esse algoritmo de otimização é o mesmo utilizado no ORB-SLAM2, que foi detalhado no capítulo anterior.

Caso o nó "Unifica_Mapas" ainda não tenha comparado o mapa atual de entrada M_i com todos os mapas armazenados no banco de dados, ele tentará executar novamente as etapas de detecção de sobreposição entre dois mapas, cálculo da posição relativa entre dois mapas e unificação dos mapas, para os demais mapas do banco de dados.

3.4 ABORDAGEM DISTRIBUÍDA PARA O SLAM VISUAL

Nas seções anteriores foi detalhada a arquitetura de um SLAM visual com uma abordagem de múltiplos mapas. Nessa abordagem, um robô é capaz de continuar mapeando um ambiente mesmo que em algum momento ele perca a referência de rastreamento. Isso é possível pois ele possui um módulo dedicado para armazenar e gerenciar as múltiplas instâncias de mapas que são construídas durante a execução do SLAM.

Além de resolver o problema exposto anteriormente, a solução proposta pode ser facilmente ampliada para uma abordagem distribuída. Assim, o sistema será capaz de funcionar com um conjunto de robôs agindo de forma cooperativa para o mapeamento de um ambiente desconhecido.

A implementação com múltiplos robôs é possível levando em consideração o fato de que o algoritmo desenvolvido para um robô já é baseado na existência de múltiplas instâncias de mapas, que são construídos pelo robô utilizando o algoritmo que executa no nó "ORB-SLAM2 modificado". E, se esses mapas possuírem regiões em comum, eles são unificados pelo nó "Unifica_Mapas". Além disso, a arquitetura implementada utilizando o paradigma de publicar e assinar do ROS torna o sistema flexível para que seja configurado com a quantidade de robôs desejada.

A Figura 3.1 exibe um exemplo da abordagem distribuída proposta para um sistema com dois robôs mapeando o ambiente de forma cooperativa. No exemplo da imagem, ambos os robôs enviam os mapas gerados para todos os da rede. Entretanto, o sistema poderia ser configurado de diferentes formas devido a sua flexibilidade de configuração.

Nesse exemplo, a comunicação é estruturada de modo que o $Robô_A$ publica os mapas gerados no formato de mensagens em um tópico τ_{R_A} que é assinando por ambos os robôs. Ao mesmo tempo em que o $Robô_B$ também publica as instâncias de mapas construídas em um tópico τ_{R_B}, que também é assinado por ambos os robôs. Nesse caso, como ambos os robôs irão receber todos os mapas construídos, é esperado que ao final da execução eles gerem mapas similares do ambiente.

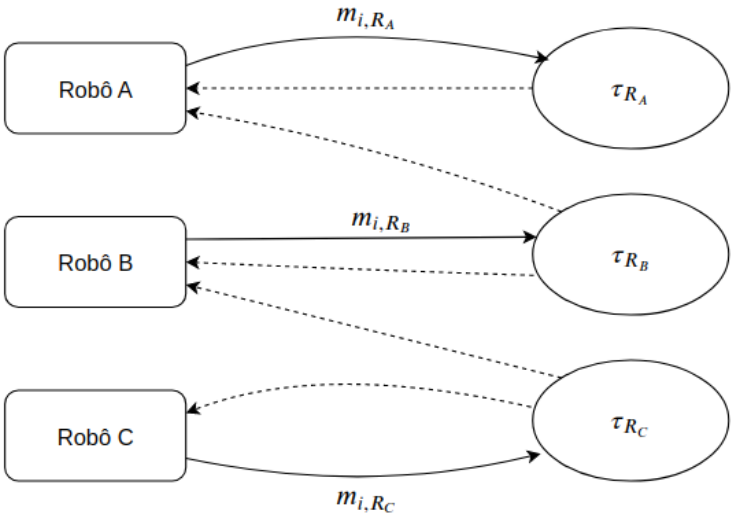


Figura 3.5: Exemplo de um possível configuração da rede para um sistema de SLAM com três robôs.

Quando os robôs recebem uma nova instância de mapa, independente da sua origem, eles executam as cinco etapas do nó “Unifica_Mapas” presente em cada um deles. Esse nó não precisou ser modificado para o funcionamento em uma arquitetura com múltiplos robôs e a sua implementação não depende da quantidade de robôs utilizados.

A única modificação necessária para configurar o sistema desejado com múltiplos robôs é a decisão de como será a troca de informações entre esses nós, ajustando o envio e o recebimento de mensagens de cada robô. Por exemplo, a Figura 3.5 ilustra uma situação na qual seriam utilizados três robôs e a configuração das trocas de informações entre eles.

Na imagem, as linha contínuas representam a ação de publicar uma mensagem em um tópico e as linhas pontilhadas estão conectando os tópicos aos nós assinantes. Como pode ser visualizado, os robôs deste exemplo recebem diferentes combinações das instâncias de mapa construídas durante a execução do SLAM visual. Nesse caso, cada robô irá finalizar a execução do sistema com um mapa final diferente.

A vantagem da abordagem distribuída desenvolvida é que ela permite a exploração de uma região maior de forma mais rápida e com menos suscetibilidade a erros. Como podem ser utilizados múltiplos agentes para a realização do SLAM visual, quando um desses agentes apresenta alguma falha, não é comprometido todo o mapeamento do ambiente.

4 RESULTADOS

4.1 INTRODUÇÃO

Este capítulo tem o intuito de apresentar os principais resultados obtidos através de experimentos realizados com a solução de SLAM visual proposta neste trabalho. A discussão dos resultados será dividida em duas seções principais, com o objetivo de verificar o desempenho do sistema desenvolvido em exemplos distintos de aplicações.

Inicialmente, serão analisados os resultados obtidos em testes nos quais apenas um robô está mapeando o ambiente, com o intuito de avaliar os prós e os contras da abordagem de múltiplos mapas implementada. Em seguida, será feita a análise da abordagem distribuída em situações em que o SLAM visual é executado com múltiplos robôs.

4.2 AMBIENTE DE SIMULAÇÃO

Como descrito no capítulo anterior, o algoritmo de SLAM foi implementado utilizando a ferramenta ROS, do inglês *Robot Operating System*. Os experimentos realizados para a validação do algoritmo foram executados *offline*. Ou seja, foram utilizados bancos de dados consagrados na literatura do SLAM visual para a execução do algoritmo em um computador, simulando um robô mapeando o ambiente.

Um dos bancos de dados utilizados para os experimentos foi o disponibilizado pelo grupo de visão computacional da TUM, apresentado em [66]. Esse banco de dados fornece algumas sequências de imagens RGB-D capturadas em ambientes internos. Ele é vastamente utilizado para a avaliação de métodos de SLAM e odometria visual em condições de diferentes iluminações, texturas e estruturas [6]. Ademais, em [6] é realizada uma comparação da acurácia dos métodos de SLAM visual ElasticFusion [67], Kintinuous [68], DVO-SLAM [69], RGB-D SLAM [70] e ORB-SLAM2 utilizando os dados desse banco de dados.

As imagens RGB-D do banco de dados da TUM foram obtidas com um sensor Microsoft Kinect. Os dados foram gravados com uma frequência de 30 Hz com uma resolução de 640×480. Além disso, ele fornece os dados da trajetória percorrida pela câmera para efeito de comparação [66].

Outro banco de dados importante para a comunidade que trabalha com SLAM e odometria visual é o fornecido pela KITTI [71, 72]. O banco de dados da KITTI contém algumas sequências de pares de imagens estéreo. Essas imagens foram capturadas por uma plataforma móvel, enquanto esta percorria algumas regiões da cidade de Karlsruhe, na Alemanha [72], a 80 km/h. O sensor estéreo utilizado possui uma linha de base de $b = 54$ cm e funciona a 10 Hz com uma resolução, após a retificação, de 1240×376 pixeis.

Esse banco de dados é bem desafiador devido a sua taxa de atualização das imagens, à velocidade do veículo durante a captura e dos cenários externos que foram capturados. Além disso, ele fornece 11 sequências de imagens com os valores das posições da câmera para a comparação dos resultados obtidos com os algoritmos de SLAM visual.

Os dados disponibilizados pela KITTI estão no formato de sequências de pares de imagens cruas, sendo o

par formado por uma imagem esquerda e uma direita. Assim, para a utilização desse banco de dados para a validação do sistema desenvolvido, foi implementado um outro nó do ROS que publica essas imagens em um tópico onde esses dados ficam disponíveis para a execução do algoritmo de SLAM.

Para a análise dos resultados gerados e a visualização dos mapas obtidos foi utilizado o pacote EVO, disponibilizado em [73].

4.3 ANÁLISE DOS RESULTADOS DA ABORDAGEM DE MÚLTIPLOS MAPAS

Os primeiros testes da abordagem proposta foram realizados com o intuito de validar a modificação realizada no ORB-SLAM2 e a estrutura do algoritmo implementado. Como detalhado no capítulo anterior, a solução apresentada de múltiplos mapas funciona com dois nós executando em paralelo no ROS. Um dos principais objetivos dessa estrutura é que o robô seja capaz de continuar o mapeamento caso ocorra alguma pertubação nos dados de entrada que interrompa o rastreamento.

Para realizar essa avaliação, foi utilizado o banco de dados de imagens RGB-D fornecido pela TUM. Pois, apesar do espaço percorrido ser interno e curto, as imagens apresentam bastante variação de iluminação. Além disso, no caso da sequência utilizada para esse teste em específico, *rgbd_dataset_freiburg2_pioneer_slam2,* a sequência de imagens apresentou alguns momentos de trepidação. Isso aconteceu porque o vídeo foi gravado com o sensor RGB-D embarcado em um robô Pioneer que estava sendo guiado por um controle remoto. Quando esse robô se locomove no ambiente muito rápido ou quando ele faz curvas, a qualidade das imagens capturadas diminui.

A curva em azul do gráfico exibido na Figura 4.1 representa o mapa construído com o algoritmo original ORB-SLAM2, apresentado em [6]. O gráfico ilustra o mapa final em comparação com o trajeto percorrido pela câmera, representado pela linha pontilhada. Esse experimento foi realizado com o ORB-SLAM2 executando em um nó do ROS e as imagens de entrada foram disponibilizadas como mensagens através de um tópico.

Como pode ser observado na imagem, a construção do mapa foi interrompida durante a execução da primeira curva pelo robô. Isso aconteceu pois, enquanto o robô se movimentava na curva, as imagens de entrada tornaram-se muito embaçadas e não foi possível manter o rastreamento.

O robô então continua a explorar o ambiente, porém, mantendo as rotinas que atualizam o mapa em espera até que ele encontre um ambiente já conhecido para se relocalizar. Isso ocorre somente ao final do percusso, como exibido no gráfico. Coincidentemente, o robô consegue se relocalizar ao passar pelo mesmo ponto em que o mapeamento tinha sido perdido anteriormente. O resultado gerado com o ORB-SLAM2 nessa sequência de imagens ilustra o tipo de situação que inspirou a proposta da abordagem de múltiplos mapas.

A Figura 4.2 exibe o resultado obtido com a abordagem de múltiplos mapas para a mesma sequência de imagens de entrada. Esse resultado foi gerado com a execução em conjunto dos nós “ORB-SLAM2 modificado” em conjunto com o “Unifica_Mapas”. Como pode ser observado, nesse caso o robô também perde a referência de rastreamento durante a execução da primeira curva, devido as pertubações nos dados de entrada. Porém, nesta abordagem, o sistema continua tentando se relocalizar apenas durante os primeiros 20 *frames* de entrada.

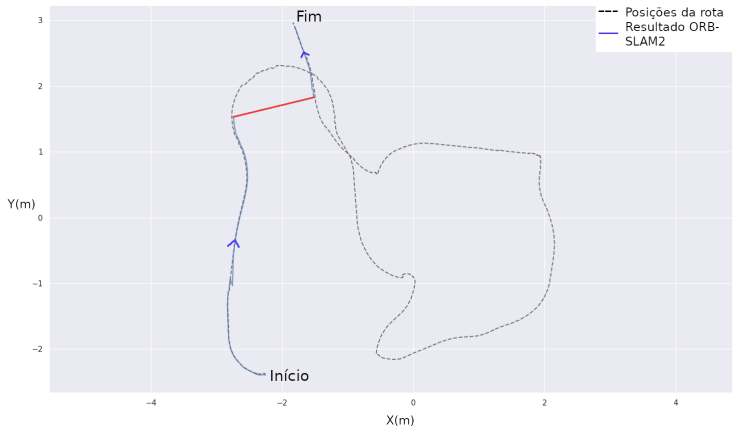
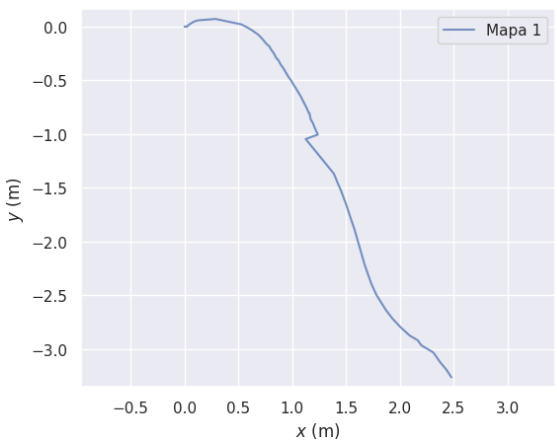
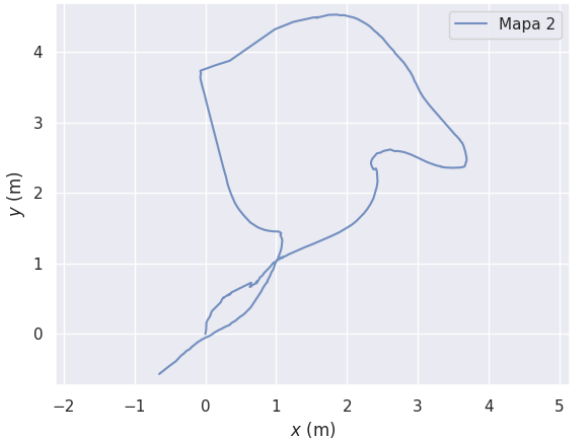


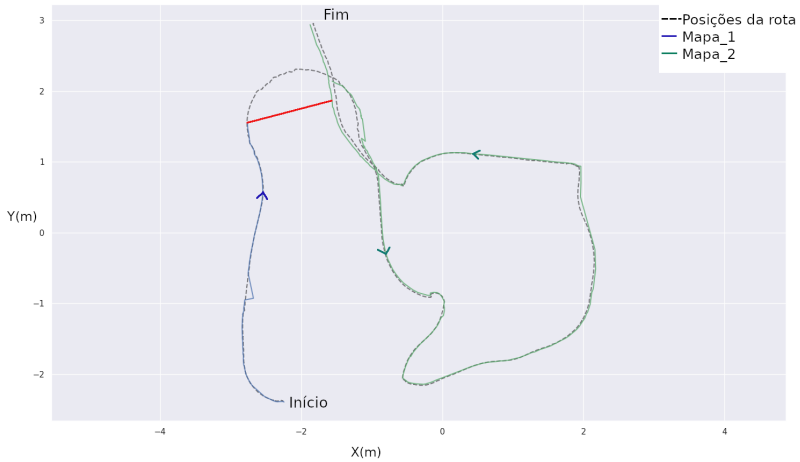
Figura 4.1: Mapa do ambiente obtido com o ORB-SLAM2.



(a) Primeiro mapa gerado.



(b) Segundo mapa construído.



(c) Mapa final gerado.

Figura 4.2: Resultado obtido com a abordagem de múltiplos mapas.

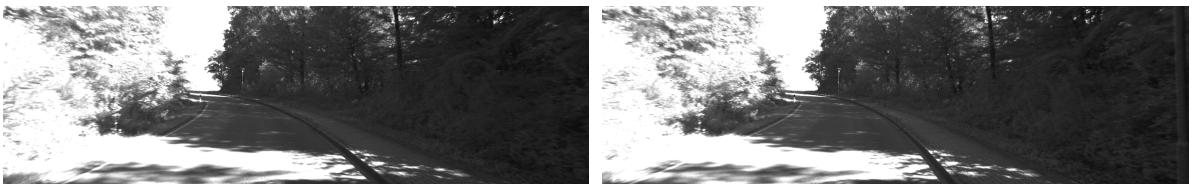
Em seguida, o primeiro mapa que estava em construção, ilustrado no gráfico da Figura 4.2a, foi enviado para o nó que gerencia as instâncias de mapa. Após o envio, um novo mapa foi inicializado, exibido na Figura 4.2b, e o sistema foi capaz de finalizar o mapeamento do ambiente. Dessa forma, os dados de entrada fornecidos enquanto o robô tentava se relocalizar não foram perdidos, como no caso anterior. Logo, foi possível mapear uma região maior do ambiente com a abordagem de múltiplos mapas proposta. Por fim, com os resultados obtidos nessa seção, é possível validar a solução de múltiplos mapas para o SLAM visual baseada no ORB-SLAM2.

4.4 ANÁLISE DOS RESULTADOS DA ABORDAGEM DISTRIBUÍDA PARA O SLAM VISUAL

Após a validação da estrutura do algoritmo de múltiplos mapas através dos experimentos discutidos na seção anterior, foram realizados testes para validar a abordagem distribuída para o SLAM visual proposta neste trabalho. Nesta seção, as simulações foram executadas com as sequências de imagens estéreo fornecidas pelo banco de dados da KITTI. Esse banco de dados foi escolhido pois os seus conjuntos de imagens possuem um número maior de dados para o experimento, já que foram capturadas em um ambiente externo.

Os dados fornecidos estão no formato de pares de imagens no formato PNG, como exibido na Figura 4.3. Sendo uma das imagens obtidas com o sensor da esquerda e a outra com o da direita. Para que esses dados fossem utilizados para o teste do algoritmo implementado, foi desenvolvido um outro nó do ROS que executa paralelamente ao sistema de SLAM visual. Esse novo nó é responsável por publicar essas imagens no formato de mensagens em um tópico a uma taxa de 6 Hz.

Devido a estrutura da implementação do algoritmo desenvolvido, essa solução de SLAM é bastante flexível para ser configurada com quantos robôs forem desejados. A seguir, serão analisados alguns experimentos realizados com diferentes conjuntos de imagens e configurações de robôs. Para a simulação de dois robôs percorrendo o ambiente, as sequências de imagens disponibilizadas foram divididas entre o número de agentes que irão "percorrer" o ambiente.



(a) Imagem gerada pelo sensor esquerdo.

(b) Imagem gerada pelo sensor direito.

Figura 4.3: Exemplo de um dos pares de imagens disponibilizados pelo banco de dados da KITTI.

4.4.1 Mapeamento com Dois Robôs

O primeiro experimento realizado simulou o ambiente sendo mapeado por um conjunto de dois robôs, com a estrutura de comunicação configurada conforme exibido no diagrama da Figura 4.4. Nesse caso, os bancos de dados utilizados para os testes foram divididos em duas partes, cada uma servindo como dados de entrada para um dos robôs.

O primeiro percurso utilizado para a validação foi o de número 03, que abrange um percurso de 560 metros e possui 801 pares de imagens estéreo. Para simular o ambiente sendo mapeado por dois robôs, foi executado em um mesmo computador os nós que seriam correspondentes a execução dos robôs e as trocas de mensagens ente eles. O $Robo_A$ recebeu como dados de entrada os pares de imagens de 0 até o 400, que correspondem a um percurso de 280 metros. E o $Robo_B$ recebeu os pares de 380 ao 801, para que houvesse uma região explorada em comum.

O sistema procede com a execução do algoritmo de SLAM "ORB-SLAM2 modificado" por ambos os robôs. As duas instâncias de mapa construídas pelo $Robo_A$ e pelo $Robo_B$ estão ilustradas, respectivamente, nas Figuras 4.5a e 4.5b. Primeiro, o $Robo_A$ finalizou a execução e serializou o mapa construído para enviar para os nós "Unifica_Mapas" que estivessem assinando o tópico τ_{R_A}. Em seguida, o $Robo_B$ compartilhou o mapa gerado no tópico τ_{R_B}.

Nesse experimento, ambos os robôs receberam os mapas construídos. Como os dados de entrada foram os mesmos e na mesma ordem, o resultado gerado na execução do nó "Unifica_Mapas" de cada robô foi o mesmo. O mapa obtido após a identificação da sobreposição entre os mapas e a correção das posições do segundo mapa recebido para o sistema de coordenadas do primeiro, pode ser visualizado no gráfico da Figura 4.5c. Na imagem, a curva em azul indica o caminho mapeado pelo primeiro robô e a em verde o caminho percorrido pelo segundo.

Além disso, a Figura 4.5d exibe um gráfico com a comparação entre o mapa final gerado e as posições da câmera fornecidas pelo banco de dados. Pode-se concluir que o resultado obtido nesse experimento com a abordagem distribuída foi de acordo com o esperado e possibilitou um mapeamento satisfatório do ambiente.

Por fim, para efeito de comparação com a abordagem proposta neste trabalho, também foi gerado o resultado com o ORB-SLAM2 original. O mapa obtido pode ser visualizado no gráfico da Figura 4.6. Observa-se que o resultado final foi bastante semelhante ao gerado com a abordagem distribuída proposta neste trabalho.

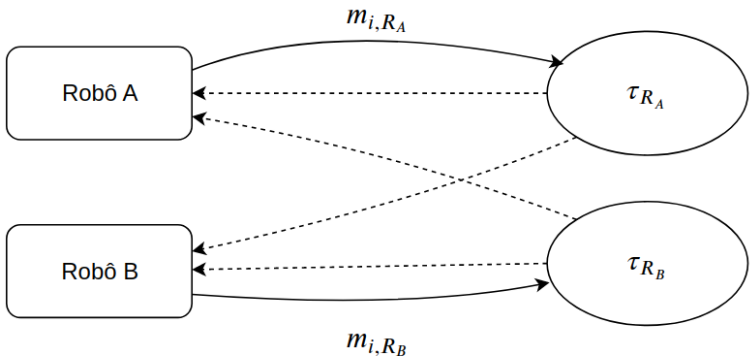
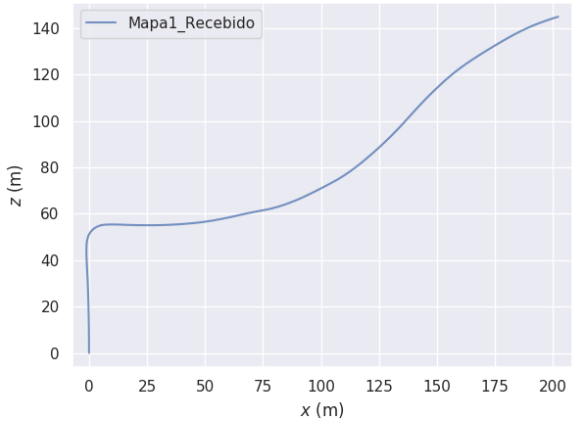
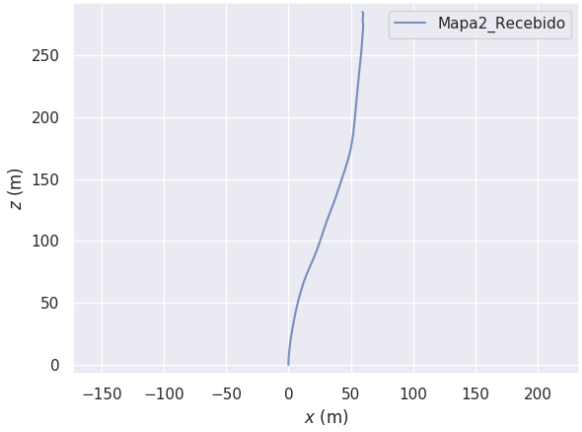


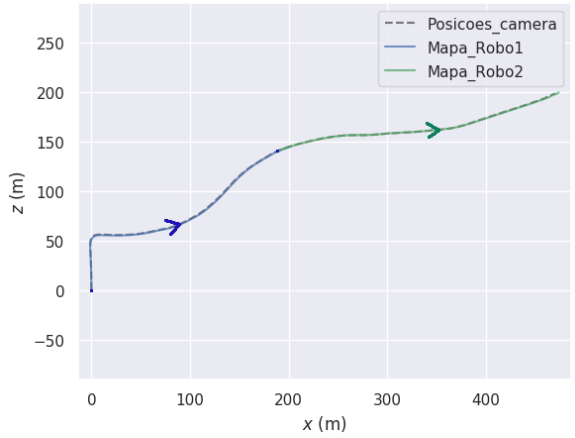
Figura 4.4: Configuração do sistema com dois robôs.



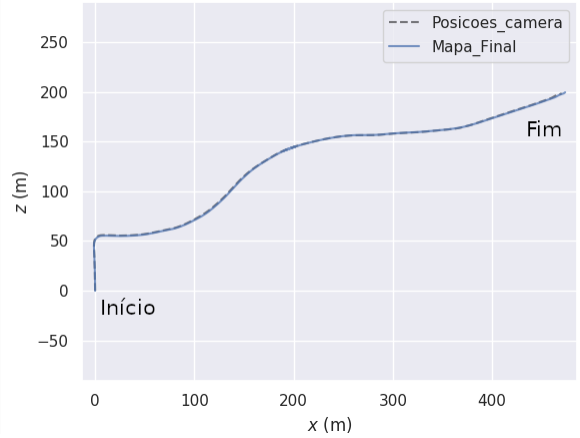
(a) Primeira instância de mapa recebida.



(b) Segunda instância de mapa recebida.



(c) Instâncias de mapas após a correções das posições do Mapa_Robo2.



(d) Mapa final gerado após a unificação.

Figura 4.5: Resultado obtido para um sistema com dois robôs.

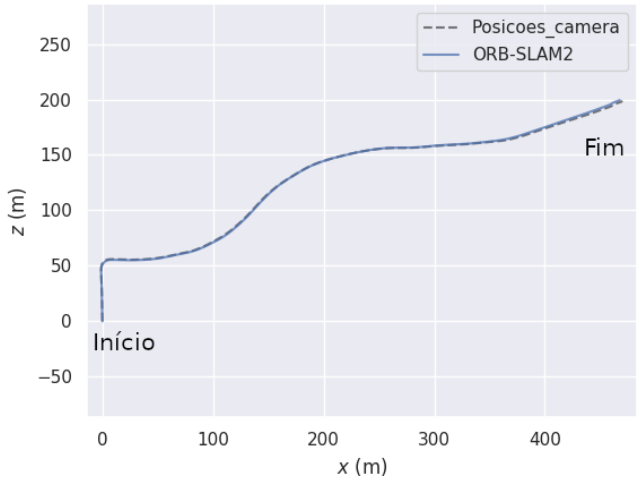


Figura 4.6: Mapa obtido com o algoritmo ORB-SLAM2 original.

4.4.2 Mapeamento com Três Robôs

Os outros testes realizados para validar a proposta da abordagem distribuída simularam o ambiente sendo mapeado por três robôs, com a estrutura de comunicação configurada de acordo com o esquemático ilustrado na Figura 4.7. Para a execução dos testes nesse sistema, os bancos de dados foram divididos em três grupos.

Resultados obtidos com a sequência de número 07

O primeiro percurso utilizado para a validação do sistema com três robôs foi o de número 07, que possui 1101 pares de imagens estéreo e abrange um percurso de 694 metros. Nesse caso, para simular o mapeamento feito por 3 robôs, o banco de dados foi dividido com o $Robo_A$ percorrendo os primeiros 256 metros, que correspondem aos pares de imagens de 0 a 400.

O $Robo_B$ mapeou 213 metros, recebendo os dados dos pares 380 ao 700. E o $Robo_C$ percorreu 236 metros, mapeando do par de imagens 680 ao 1101. Os pares de imagens dados em comum para os robôs gera propositalmente regiões mapeadas em comum.

A partir da execução do algoritmo de SLAM modificado nos robôs $Robo_A$, $Robo_B$ e $Robo_C$ foram obtidas, respectivamente, as instâncias de mapas ilustradas nas Figuras 4.8a, 4.8b e 4.8c. Como nesse experimento a comunicação foi estruturada de forma que o $Robo_A$ e o $Robo_B$ recebem as mesmas instâncias de mapa de entrada e na mesma sequência, o resultado gerado por eles será idêntico.

Inicialmente, esses dois robôs recebem o primeiro mapa gerado, ilustrado na Figura 4.8a. Em seguida, eles recebem o mapa exibido na Figura 4.8b. Após recebida a segunda instância de mapa, o nó "Unifica_Mapas" dos robôs detectou corretamente a sobreposição existente entre os dois mapas e recalculou o segundo mapa para o sistema de coordenadas no primeiro, como pode ser visualizado no gráfico da Figura 4.9a. A Figura 4.9b exibe o resultado final da unificação dos dois primeiros mapas $M_1' = M_1 \cup M_2$ em comparação com as posições percorridas pela câmera.

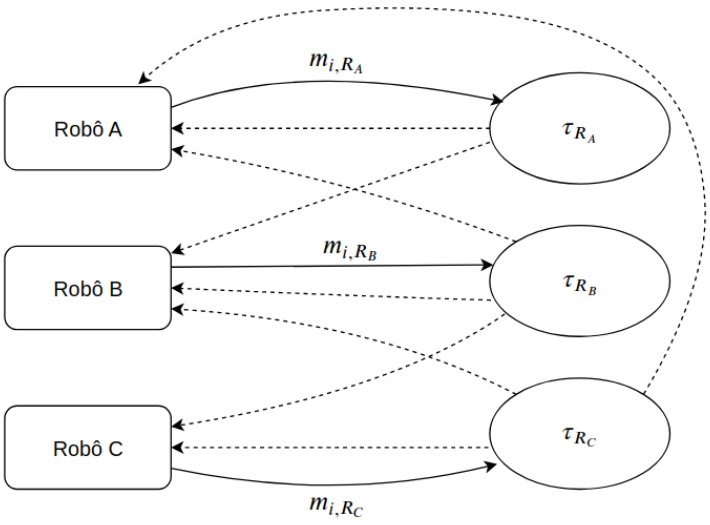
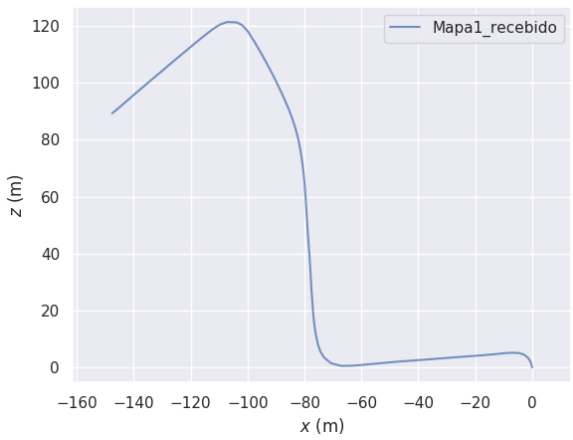
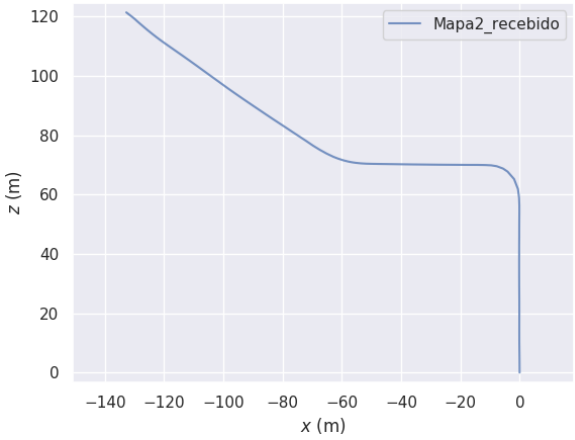


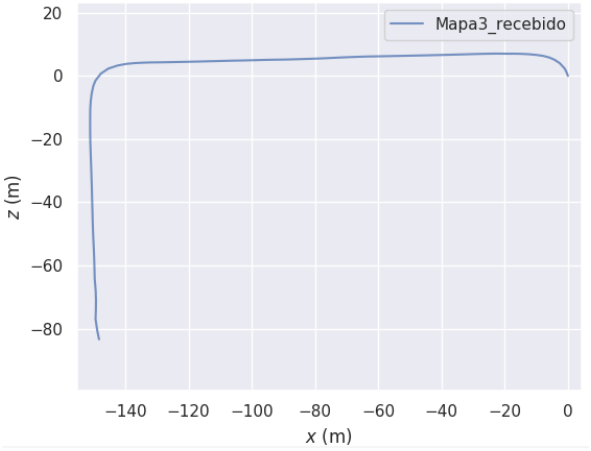
Figura 4.7: Configuração do sistema com três robôs.



(a) Instância de mapa gerada pelo Robô A.



(b) Instância de mapa gerada pelo Robô B.



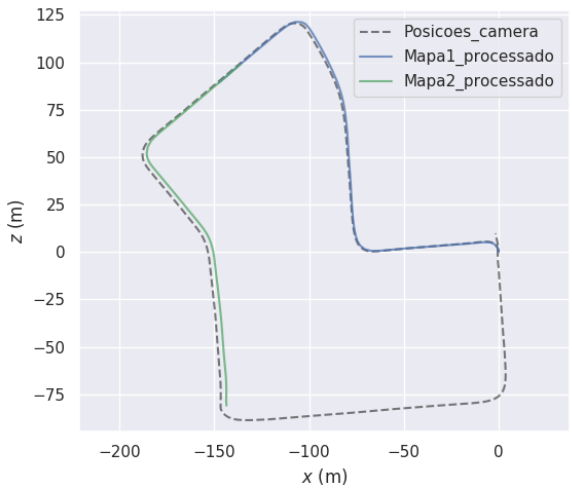
(c) Instância de mapa gerada pelo Robô C.

Figura 4.8: Instâncias de mapas geradas no experimento com a sequência de dados número 07.

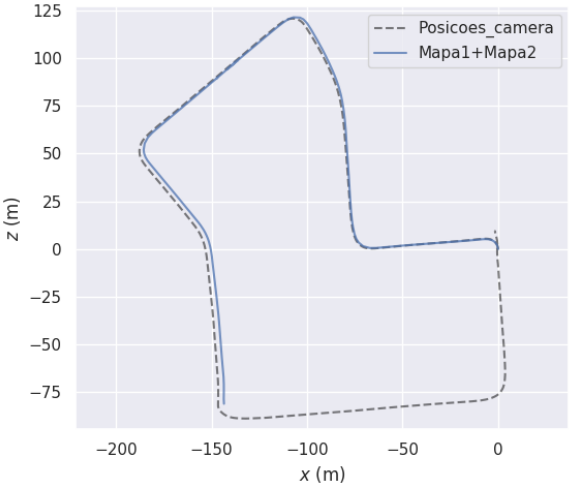
Em seguida, os robôs recebem a última instância de mapa gerada, representada no gráfico da Figura 4.8c. O nó "Unifica_Mapas" tenta detectar a sobreposição entre essa instância e o mapa já armazenado e calcular a distância relativa ele entres, para colocá-los no mesmo sistema de coordenadas. O resultado obtido pode ser visualizado no gráfico da Figura 4.10a.

A Figura 4.10b exibe o resultado final da unificação de todas as instância de mapa geradas pelos robôs em comparação com as posições percorridas pela câmera. Além disso, o gráfico da Figura 4.11 ilustra o resultado gerado com o algoritmo ORB-SLAM2 original.

Observa-se que nesse experimento o mapa final gerado com a abordagem distribuída diverge dos pontos da trajetória da câmera um pouco mais do que o gerado com o algoritmo original, executado só com um robô. Isso pode ser explicado, haja vista que o ORB-SLAM2 conseguiu reduzir os erros acumulados com o fechamento de laço que ocorre ao final da trajetória percorrida. Como a abordagem distribuída implementada só utiliza um ponto em comum entre os mapas para processar a unificação, o fechamento de laço não foi aproveitado para melhorar o mapa final.

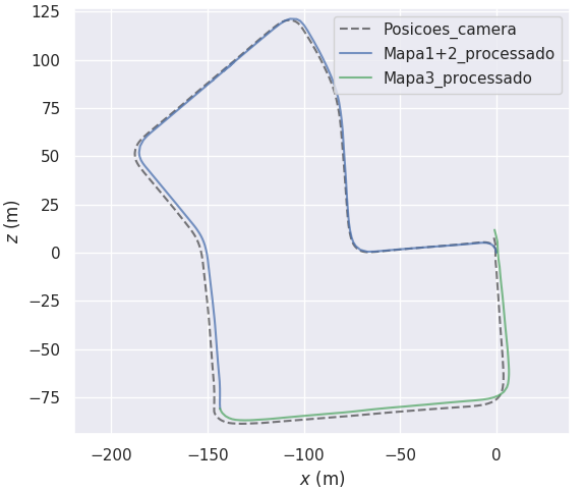


(a) Instâncias de mapas após a correções das posições do segundo mapa.

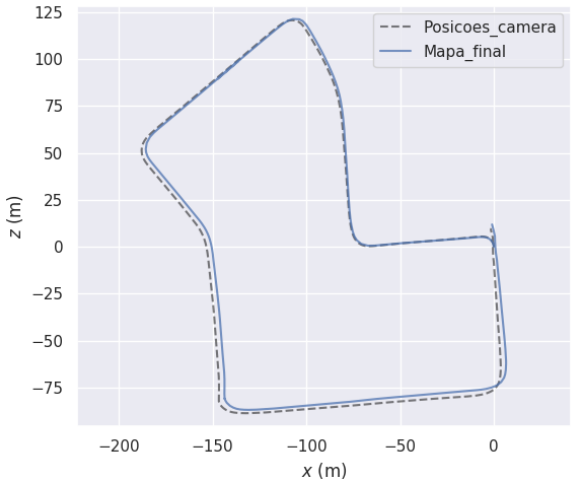


(b) Mapa final gerado após a unificação dos mapas 1 e 2.

Figura 4.9: Resultado obtido da união das duas primeiras instâncias de mapa.



(a) Instâncias de mapas após a correções das posições do terceiro mapa.



(b) Mapa final gerado após a unificação dos mapas.

Figura 4.10: Resultados obtidos da união das três instâncias de mapa geradas com a sequência de número 07.

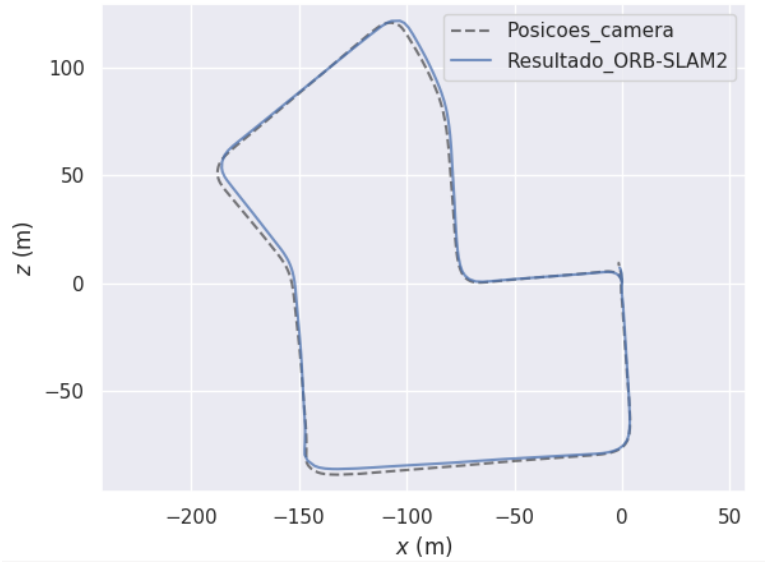
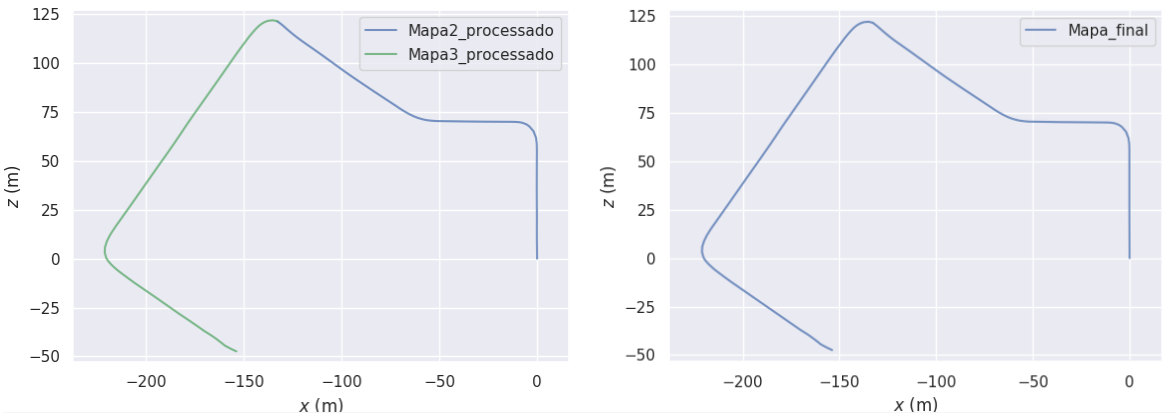


Figura 4.11: Mapa obtido com o algoritmo ORB-SLAM2 original com a sequência de número 07.

Um resultado final diferente foi gerado pelo $Robo_C$. Já que ele recebeu como entrada para o nó "Unifica_Mapas", respectivamente, os mapas 2 e 3, ilustrados nas Figuras 4.8b e 4.8c. Esse nó procede então com a detecção da região em comum entre os mapas e o cálculo da posição relativa entre eles para poder unificá-los.

A Figura 4.12a exibe o resultado após a correção das posições do mapa 3 para o sistema de coordenadas do mapa 2. O resultado final da unificação destas instâncias de mapas está ilustrado na Figura 4.12b. Nesse caso, o mapa final ficou em um sistema de coordenadas diferente do gerado pelos outros robôs da rede e abrange uma região percorrida menor.



(a) Instâncias de mapas após a correções das posições do terceiro mapa em relação ao segundo.

(b) Mapa final gerado após a unificação dos mapas 2 e 3.

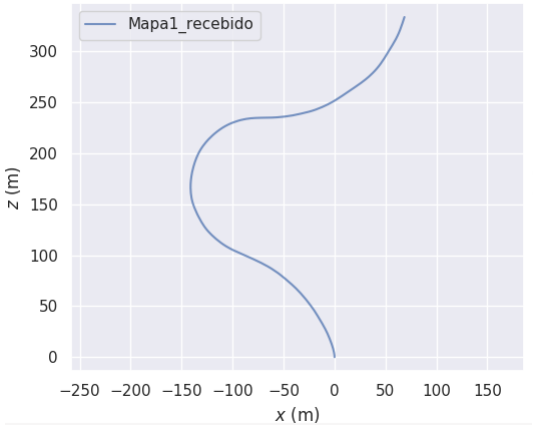
Figura 4.12: Resultados obtidos da união das instâncias de mapa 2 e 3.

Resultados obtidos com a sequência de número 09

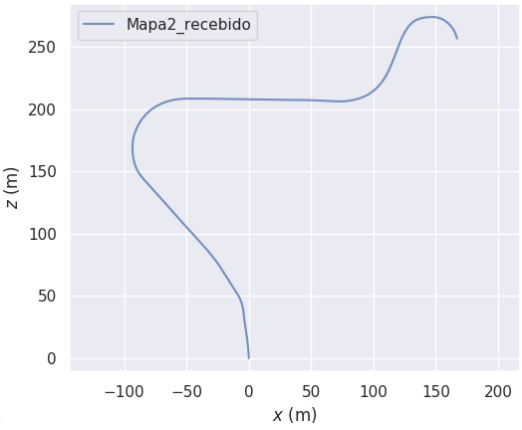
Um segundo experimento foi realizado no mesmo sistema com três robôs, exibido na Figura 4.7, no qual foi utilizada a sequência de imagens de número 09. Essa sequência possui um trajeto maior que a utilizada no experimento anterior, com 1591 pares de imagens estéreo e um percurso de 1705 metros.

Nesse teste, para simular o mapeamento feito pelos 3 robôs, esse banco de dados foi dividido com o $Robo_A$ percorrendo os primeiros 524 metros, que correspondem aos pares de imagens de 0 a 500. O $Robo_B$ mapeou 527 metros, recebendo os dados dos pares 480 ao 1000. E o $Robo_C$ percorreu 680 metros, mapeando do par de imagens 980 ao 1591. Assim como no exemplo anterior, os pares de imagens dados em comum para os robôs geram propositalmente regiões mapeadas em comum.

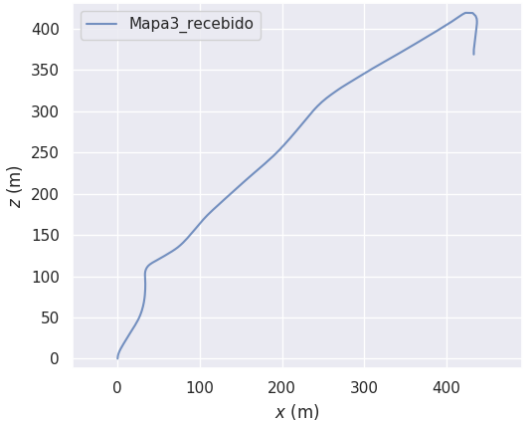
A partir da execução do algoritmo de SLAM modificado nos robôs $Robo_A$, $Robo_B$ e $Robo_C$ foram geradas, respectivamente, as instâncias de mapas ilustradas nas Figuras 4.13a, 4.13b e 4.13c. Nesse experimento, assim como no anterior, a comunicação foi estruturada de forma que o $Robo_A$ e o $Robo_B$ recebem as mesmas instâncias de mapa de entrada e na mesma sequência, logo, o resultado gerado por eles será idêntico.



(a) Instância de mapa gerada pelo Robô A.



(b) Instância de mapa gerada pelo Robô B.



(c) Instância de mapa gerada pelo Robô C.

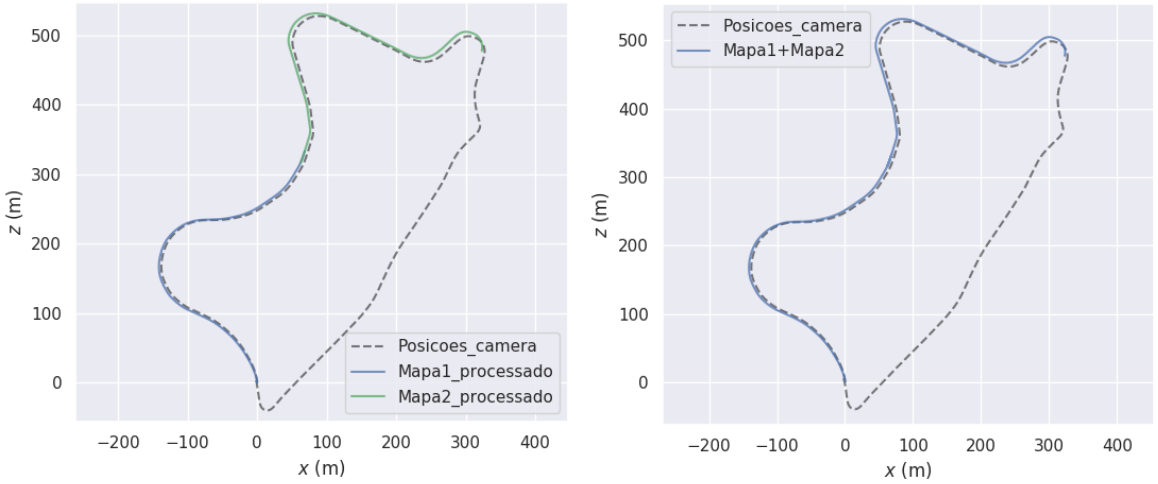
Figura 4.13: Instâncias de mapas geradas no experimento com a sequência de dados número 09.

Esses dois robôs recebem no nó “Unifica_Mapas”, respectivamente, o primeiro mapa gerado, ilustrado na Figura 4.13a, e o mapa exibido na Figura 4.13b. Em seguida, esse nó dos robôs verifica a sobreposição existente entre os dois mapas e modifica o segundo mapa para ficar no sistema de coordenadas do primeiro. O resultado obtido com esse processamento pode ser visualizado no gráfico da Figura 4.14a. E o gráfico da Figura 4.14b exibe o resultado final da unificação dos dois primeiros mapas $M_1' = M_1 \cup M_2$ em comparação com as posições percorridas pela câmera.

Posteriormente, os robôs recebem a última instância de mapa gerada, representada no gráfico da Figura 4.13c. O nó “Unifica_Mapas” tenta, então, detectar a sobreposição entre essa instância e o mapa armazenado anteriormente e calcular a distância relativa ele entres, para colocá-los no mesmo sistema de coordenadas. O resultado obtido pode ser visualizado no gráfico da Figura 4.15a, que ilustra todas as instâncias de mapas geradas no mesmo sistema de coordenadas.

O resultado da unificação de todos os mapas gerados pela rede de robôs pode ser visualizado no gráfico da Figura 4.15b. No gráfico é possível verificar a comparação do mapa final gerado com as posições percorridas pela câmera. Além disso, o gráfico da Figura 4.16 ilustra o resultado obtido com o algoritmo ORB-SLAM2 original em um só robô.

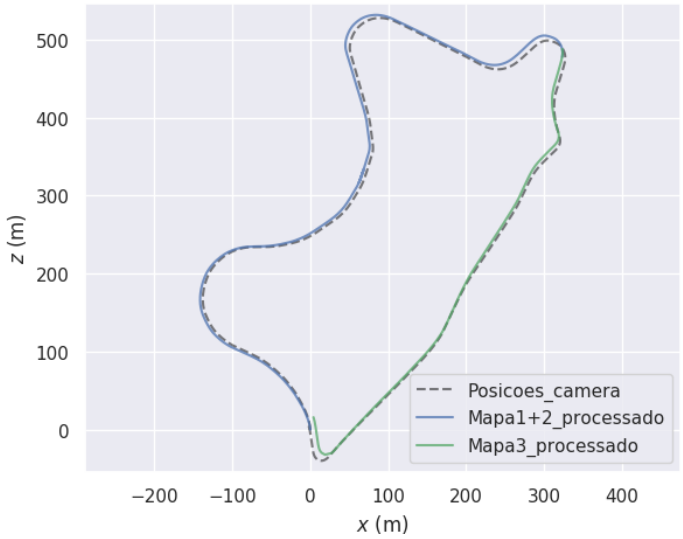
Nesse experimento, o mapa final gerado com a abordagem distribuída foi bastante similar ao resultado do algoritmo original executado só com um robô. Diferentemente do que ocorreu no experimento anterior, na sequência de dados 07, aqui o ORB-SLAM2 não foi capaz de detectar o fechamento de laço no final da execução, devido a quantidade muito reduzida de imagens que observavam a mesma região. Logo, ele não corrigiu os erros acumulados durante a execução do algoritmo de SLAM visual.



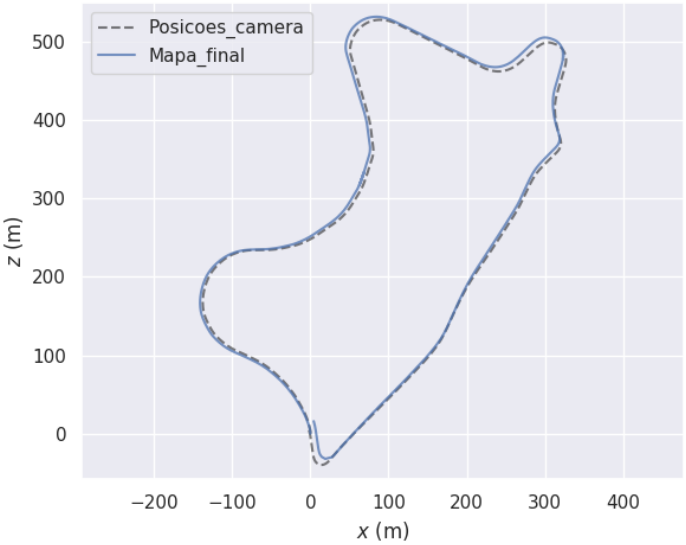
(a) Instâncias de mapas após a correções das posições do segundo mapa.

(b) Mapa final gerado após a unificação dos mapas 1 e 2.

Figura 4.14: Resultado obtido da união das duas primeiras instâncias de mapa.



(a) Instâncias de mapas após a correções das posições do terceiro mapa.



(b) Mapa final gerado após a unificação dos mapas.

Figura 4.15: Resultados obtidos da união das três instâncias de mapa geradas com a sequência de número 09.

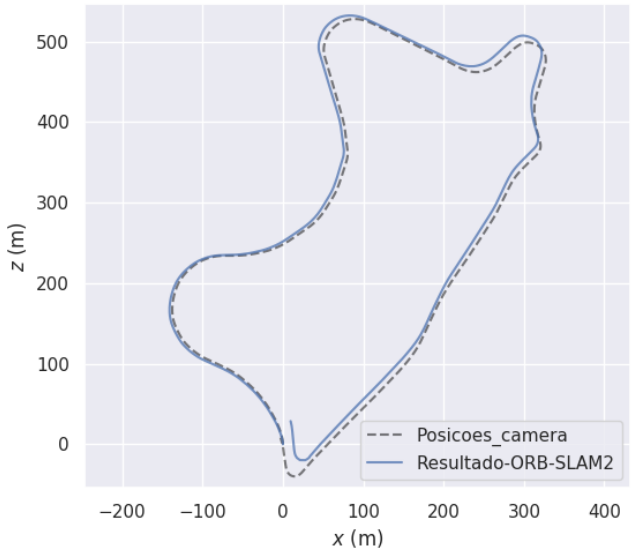
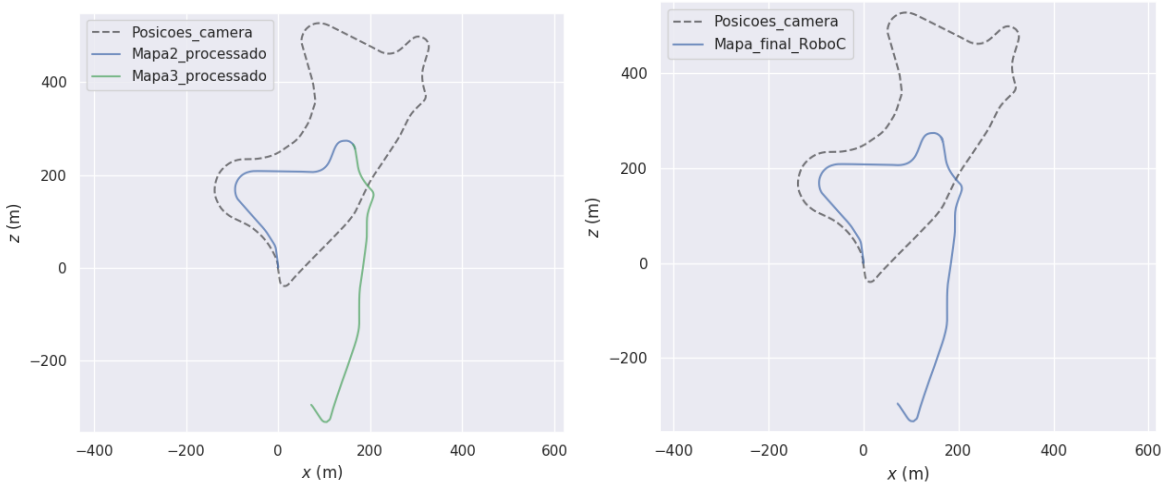


Figura 4.16: Mapa obtido com o algoritmo ORB-SLAM2 original com a sequência de número 09.

Assim como no experimento realizado anteriormente, um resultado final diferente foi construído pelo $Robo_C$; pois ele recebeu como entrada para o nó "Unifica_Mapas", respectivamente, apenas os mapas 2 e 3, ilustrados nas Figuras 4.13b e 4.13c. Esse nó procedeu então com a detecção da região em comum entre esses mapas e o cálculo da posição relativa entre eles para poder unificá-los.

O resultado obtido após a correção das posições do mapa 3 para o sistema de coordenadas do mapa 2 está representado no gráfico da Figura 4.17a. O resultado final da unificação destas instâncias de mapas está ilustrado na Figura 4.17b. Nesse caso, o sistema de coordenadas do mapa final é diferente do mapa resultante obtido pelos outros robôs da rede, além de abranger uma região mapeada menor.



(a) Instâncias de mapas após a correções das posições do terceiro mapa em relação ao segundo.

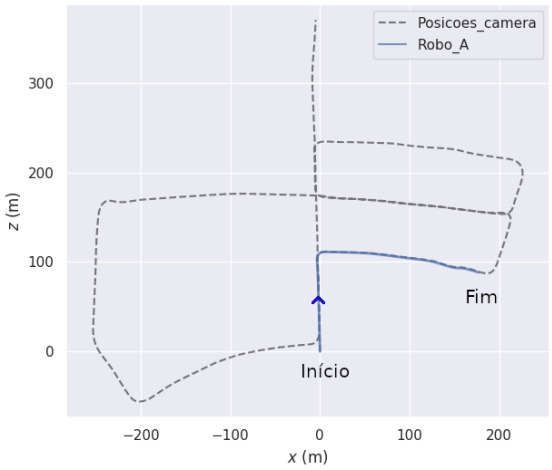
(b) Mapa final gerado após a unificação dos mapas 2 e 3.

Figura 4.17: Resultados obtidos da união das instâncias de mapa 2 e 3 para a sequência de dados 09.

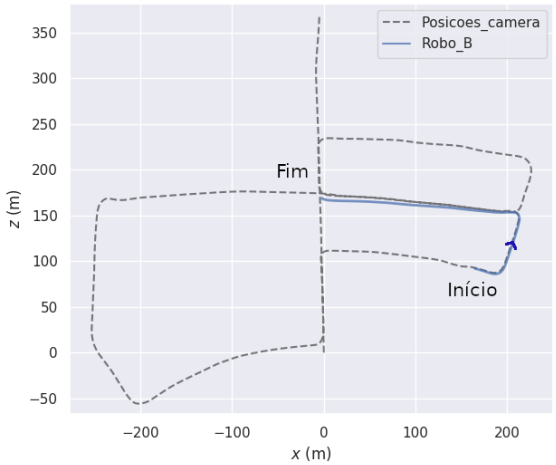
Resultados obtidos com a sequência de número 05

Por fim, foi realizado um experimento com mesmo sistema de três robôs, no qual foi utilizada a sequência de dados de número 05. Essa sequência de dados possui um percurso bem mais extenso, de 2205 metros, com 2761 pares de imagens estéreo. Esse teste foi realizado nesse trajeto maior com o intuito de modificar o percurso definido para os robôs. Para esse experimento não foram utilizados todos os dados fornecidos.

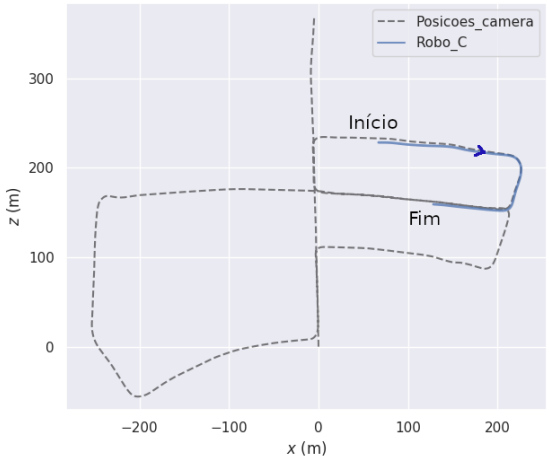
Nesse caso, o $Robo_A$ mapeou os primeiros 291 metros, que correspondem aos pares de imagens de 0 a 400. O $Robo_B$ moveu-se por 313 metros, recebendo as imagens dos pares 380 ao 800. E o $Robo_C$ percorreu 301 metros, mapeando do par de imagens 1000 ao 1400. As direções da movimentação dos robôs no ambiente nesse exemplo podem ser melhor visualizadas através dos gráficos das Figuras 4.18a, 4.18b e 4.18c. Assim como no exemplo anterior, os locais visitados em comum geram propositalmente regiões mapeadas em comum para possibilitar a unificação dos mapas.



(a) Trajetória percorrida pelo Robô A.



(b) Trajetória percorrida pelo Robô B.



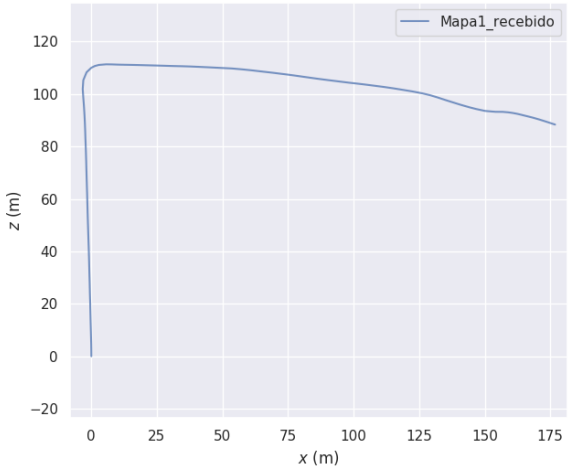
(c) Trajetória percorrida pelo Robô C.

Figura 4.18: Gráficos com as trajetórias percorridas pelos robôs em relação ao percurso do banco de dados.

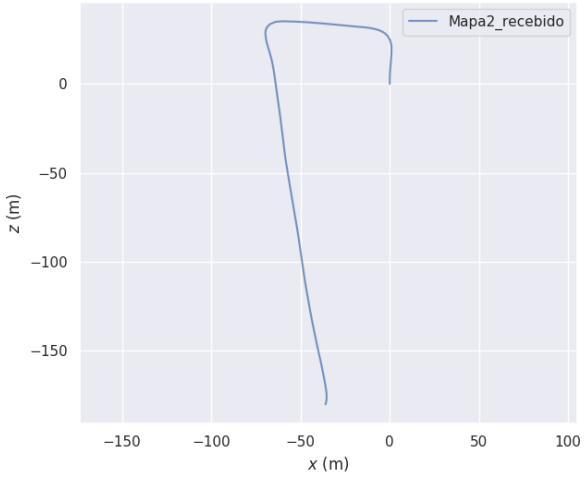
O algoritmo de SLAM modificado nos robôs $Robo_A$, $Robo_B$ e $Robo_C$ irá construir, respectivamente, as instâncias de mapas representadas nos gráficos das Figuras 4.19a, 4.19b e 4.19c. Nesse experimento, assim como nos anteriores, o $Robo_A$ e o $Robo_B$ irão gerar o mesmo mapa final.

O nó "Unifica_Mapas" desses dois robôs irá receber o mapa 1, ilustrado na Figura 4.19a. Posteriormente, será recebido o mapa 2, representado no gráfico da Figura 4.19b. Esse nó de ambos os robôs checa então se existe alguma sobreposição entre os mapas recebidos e, em caso afirmativo, modifica o segundo mapa para ficar no sistema de coordenadas do primeiro. O resultado obtido após esse processamento pode ser visualizado no gráfico da Figura 4.20a. E o resultado final da unificação dos dois primeiros mapas $M_1' = M_1 \cup M_2$ está ilustrado na Figura 4.20b.

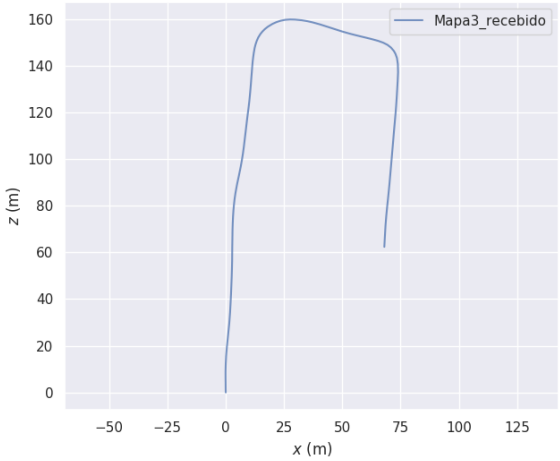
Em seguida, os robôs recebem a última instância de mapa gerada, representada no gráfico da Figura 4.19c. O nó "Unifica_Mapas" tenta detectar a sobreposição entre essa instância e o mapa armazenado e, em caso afirmativo, calcula a distância relativa entre eles. O gráfico da Figura 4.21a exibe todas as instâncias de mapas geradas no mesmo sistema de coordenadas, após realizado esse processamento.



(a) Instância de mapa gerada pelo Robô A.

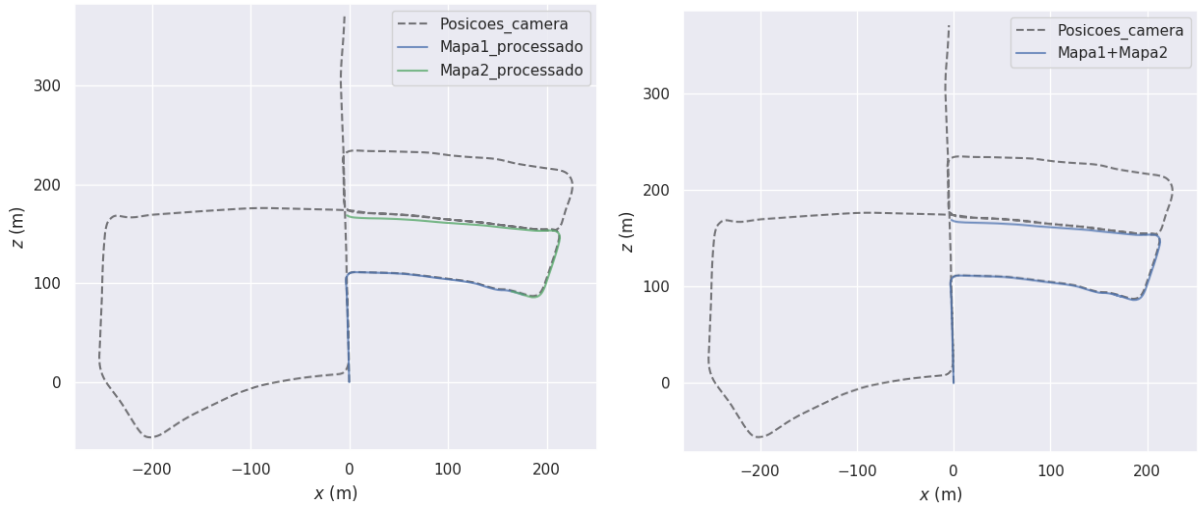


(b) Instância de mapa gerada pelo Robô B.



(c) Instância de mapa gerada pelo Robô C.

Figura 4.19: Instâncias de mapas geradas no experimento com a sequência de dados número 05.

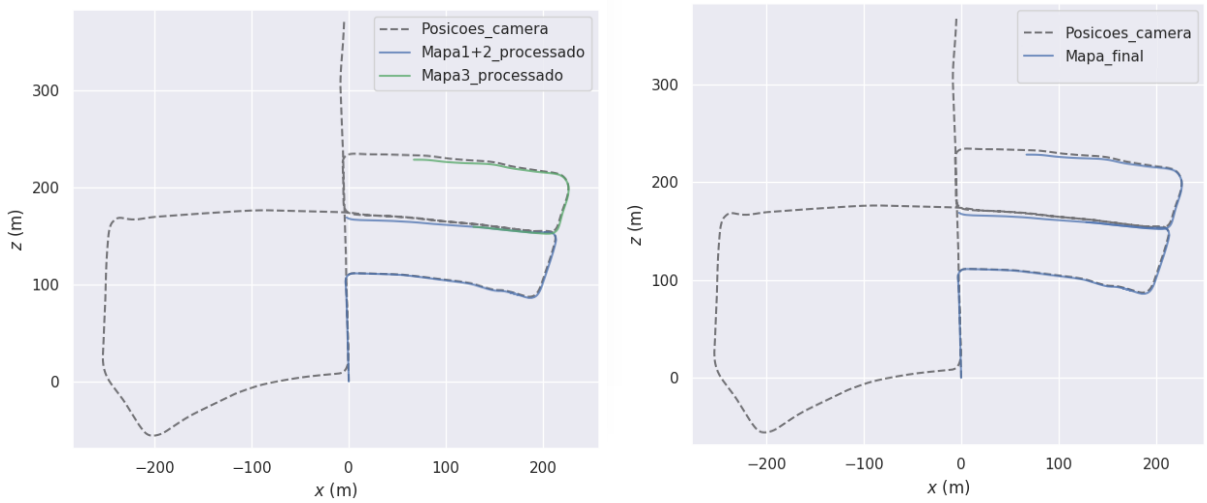


(a) Instâncias de mapas após a correções das posições do segundo mapa.

(b) Mapa final gerado após a unificação dos mapas 1 e 2.

Figura 4.20: Resultado obtido da união das duas primeiras instâncias de mapa geradas a partir da sequência de imagens 05.

A Figura 4.21b ilustra o resultado final obtido com a unificação de todas as instâncias de mapas recebidas. No gráfico dessa imagem é exibida a comparação entre o mapa final gerado pelo nó "Unifica_Mapas" com as posições da câmera fornecidas pelo banco de dados. Ademais, o gráfico da Figura 4.22 exibe o resultado obtido com um só robô executando o algoritmo ORB-SLAM2 original, recebendo como dados de entrada todas as imagens contidas na sequência 05 do bancos de dados da KITTI.



(a) Instâncias de mapas após a correções das posições do terceiro mapa.

(b) Mapa final gerado após a unificação das três instâncias de mapas.

Figura 4.21: Resultados obtidos da união das três instâncias de mapa geradas a partir da sequência de imagens 05, utilizando a abordagem distribuída.

Assim como no teste com a sequência 07, o resultado obtido nesse terceiro experimento diverge um pouco do resultado do algoritmo original. Do mesmo modo que no caso anterior, isso pode ser explicado devido aos fechamentos de laço que ocorrem durante a execução do ORB-SLAM2, que permitem que ele reduza o impacto dos erros acumulados com o tempo de execução do SLAM visual. Enquanto que no algoritmo distribuído, os mapas gerados por cada robô foram acumulando erros durante a execução do SLAM e eles não chegaram a percorrer o ambiente até um local de fechamento de laço.

O $Robo_C$, assim com nos casos anteriores, obteve um mapa final diferente, uma vez que ele recebeu, respectivamente, apenas os mapas 2 e 3 como dados de entrada para o nó "Unifica_Mapas". Em seguida, esse nó verifica a ocorrência de uma região em comum entre esses mapas e o cálculo da posição relativa entre eles para poder unificá-los.

O gráfico da Figura 4.23a exibe o resultado obtido após a correção das posições do mapa 3. O resultado final da unificação destas instâncias de mapas pode ser visualizado na Figura 4.23b. Assim como nas execuções anteriores, o sistema de coordenadas do mapa final gerado por esse robô é diferente do mapa obtido pelos outros.

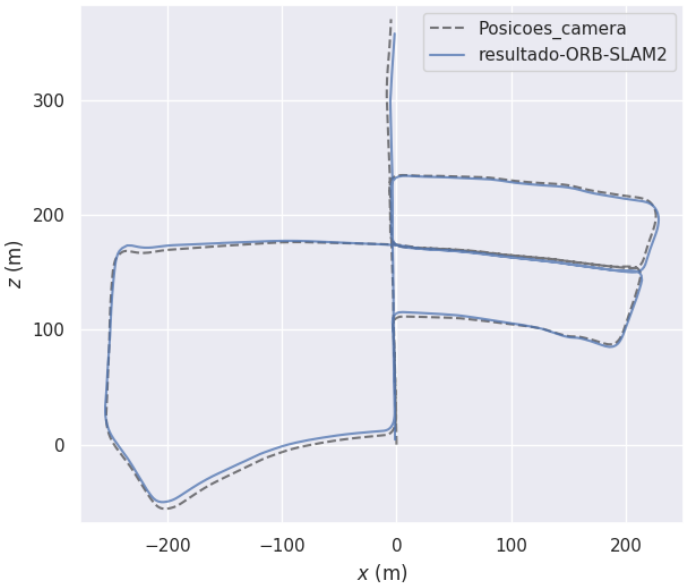
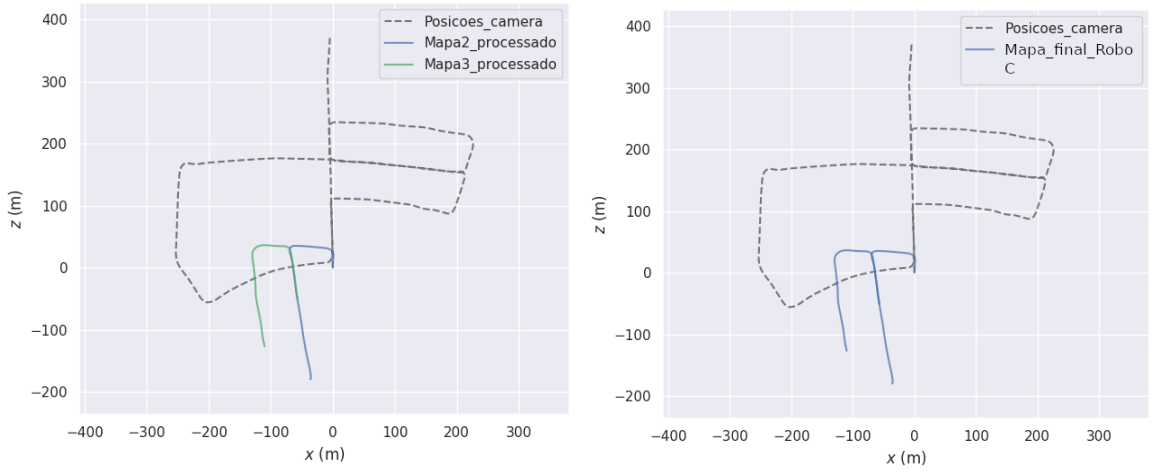


Figura 4.22: Mapa obtido com o algoritmo ORB-SLAM2 original com a sequência de número 05.



(a) Instâncias de mapas após a correções das posições do terceiro mapa em relação ao segundo.

(b) Mapa final gerado após a unificação dos mapas 2 e 3.

Figura 4.23: Resultados obtidos da união das instâncias de mapa 2 e 3 para a sequência de dados 05.

5 CONCLUSÕES

5.1 CONSIDERAÇÕES FINAIS

Neste trabalho, foi proposta uma abordagem distribuída para o SLAM visual. O sistema foi desenvolvido com o intuito de fornecer uma solução flexível que permita ser utilizada em um sistema com qualquer quantidade de robôs. Reduzido para o caso de apenas um robô, com relação à solução estado da arte para o SLAM visual, o ORB-SLAM2, o algoritmo proposto introduz uma solução de múltiplos mapas gerados. Como tal, mesmo para o caso monoagente, o algoritmo SLAM proposto apresenta melhorias. De fato, o algoritmo SLAM proposto resolve o problema da falha no rastreamento devido a perda momentânea de qualidade dos dados de entrada do ORB-SLAM2.

Os resultados obtidos através dos experimentos realizados para a validação deste projeto forneceram informações qualitativas e quantitativas a respeito do desempenho da metodologia de SLAM desenvolvida. Assim, foi possível verificar que a abordagem proposta de múltiplos mapas atingiu os objetivos estabelecidos no início do projeto, ou seja, ela foi capaz de evitar que o robô permanecesse se movimentando no ambiente com a rotina de rastreamento pausada. E, ao final da execução, essa metodologia conseguiu gerar um mapa do ambiente de forma satisfatória.

Além disso, com os experimentos realizados foi possível validar a proposta de abordagem distribuída feita neste trabalho e verificar a flexibilidade de configuração do sistema proposto. Já que o sistema foi capaz de funcionar com êxito em simulações com conjuntos de dois e três robôs trabalhando em cooperação.

Nesse caso, foi observado que o mapa final gerado, a partir da unificação dos mapas construídos individualmente, apresentou um resultado satisfatório. Entretanto, observou-se que em percursos longos, o SLAM vai acumulando erros durante a execução do algoritmo e esses erros interferem no resultado final do algoritmo distribuído implementado, quando comparado ao resultado obtido por um só robô que consegue otimizar o mapa em um ponto de fechamento de laço.

5.2 PROPOSTAS PARA TRABALHOS FUTUROS

Para trabalhos futuros, é proposto que o sistema seja executado e validado em um conjunto de robôs reais para a localização e mapeamento em um ambiente desconhecido. Dessa forma, ele poderá ser avaliado em uma aplicação em tempo real e com todas as dificuldades de um ambiente real.

Além disso, propõe-se implementar uma melhoria no nó “Unifica_Mapas” para que ele seja capaz de detectar mais de um ponto de sobreposição entre os mapas para a geração do mapa geral. Assim, possivelmente os resultados serão mais semelhantes com os obtidos com a execução do SLAM feita só com um robô.

Outra sugestão para possíveis trabalhos futuros seria tentar fazer outra modificação no algoritmo que executa a rotina de localização e mapeamento simultâneos, para que sejam considerados os dados de algum outro sensor embarcado no robô. Os dados desse sensor seriam utilizados para a predição da posição da câmera ao invés do uso do modelo de movimento de velocidade constante.

[1] R. Behringer, "The DARPA grand challenge-autonomous ground vehicles in the desert," *IFAC Proceedings Volumes*, vol. 37, no. 8, pp. 904–909, 2004.

[2] G. Younes, D. Asmar, and E. Shammas, "A survey on non-filter-based monocular visual SLAM systems," *arXiv preprint arXiv:1607.00470*, vol. 413, p. 414, 2016.

[3] C. Kahlefendt, "Implementation and evaluation of monocular SLAM for an underwater robot," Master's thesis, University of Western Australia Robotics and Automation Lab, 2018.

[4] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part I," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[5] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[6] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[7] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European conference on computer vision*. Springer, 2006, pp. 430–443.

[8] K. Yousif, A. Bab-Hadiashar, and R. Hoseinnezhad, "An overview to visual odometry and visual SLAM: Applications to mobile robotics," *Intelligent Industrial Systems*, vol. 1, no. 4, pp. 289–311, 2015.

[9] R. R. Murphy, *Introduction To AI Robotics*. Cambridge, Massachusetts: MIT, 2000.

[10] M. Bernard, K. Kondak, I. Maza, and A. Ollero, "Autonomous transportation and deployment with aerial robots for search and rescue missions," *Journal of Field Robotics*, vol. 28, no. 6, pp. 914–931, 2011.

[11] Y. Liu and G. Nejat, "Robotic urban search and rescue: A survey from the control perspective," *Journal of Intelligent & Robotic Systems*, vol. 72, no. 2, pp. 147–165, 2013.

[12] C. M. Humphrey and J. A. Adams, "Robotic tasks for chemical, biological, radiological, nuclear and explosive incident response," *Advanced robotics*, vol. 23, no. 9, pp. 1217–1232, 2009.

[13] L. H. S. Porto, "Controle de movimento de um robô móvel não-holonômico com tração diferencial," 2017. [Online]. Available: http://bdm.unb.br/handle/10483/19239

[14] L. H. Silva Porto, R. Werberich da Silva Moreira de Oliveira, R. Bauchspiess, C. Brito, L. F. da Cruz Figueredo, G. Araujo Borges, and G. Novaes Ramos, "An autonomous mobile robot architecture for outdoor competitions," in *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*, Nov 2018, pp. 141–146.

[15] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.

[16] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski, "ORB: An efficient alternative to SIFT or SURF." in *ICCV*, vol. 11, no. 1. Citeseer, 2011, p. 2.

[17] M. R. Naminski, "An analysis of simultaneous localization and mapping (SLAM) algorithms," 2013.

[18] H. Strasdat, J. M. Montiel, and A. J. Davison, "Visual SLAM: why filter?" *Image and Vision Computing*, vol. 30, no. 2, pp. 65–77, 2012.

[19] D. Fox, J. Ko, K. Konolige, B. Limketkai, D. Schulz, and B. Stewart, "Distributed multirobot exploration and mapping," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1325–1339, 2006.

[20] R. Chellali *et al.*, "A distributed multi robot SLAM system for environment learning," in *2013 IEEE Workshop on Robotic Intelligence in Informationally Structured Space (RiiSS)*. IEEE, 2013, pp. 82–88.

[21] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual SLAM algorithms: A survey from 2010 to 2016," *IPSJ Transactions on Computer Vision and Applications*, vol. 9, no. 1, p. 16, 2017.

[22] R. C. Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *The international journal of Robotics Research*, vol. 5, no. 4, pp. 56–68, 1986.

[23] J. J. Leonard and H. F. Durrant-Whyte, "Simultaneous map building and localization for an autonomous mobile robot," in *Proceedings IROS'91: IEEE/RSJ International Workshop on Intelligent Robots and Systems' 91*. Ieee, 1991, pp. 1442–1447.

[24] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *ICRA*, vol. 2, 1999, pp. 1322–1328.

[25] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (SLAM): Part II," *IEEE robotics & automation magazine*, vol. 13, no. 3, pp. 108–117, 2006.

[26] G. Bresson, Z. Alsayed, L. Yu, and S. Glaser, "Simultaneous localization and mapping: A survey of current trends in autonomous driving," *IEEE Transactions on Intelligent Vehicles*, vol. 2, no. 3, pp. 194–220, 2017.

[27] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[28] T. Bailey, J. Nieto, J. Guivant, M. Stevens, and E. Nebot, "Consistency of the EKF-SLAM algorithm," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2006, pp. 3562–3568.

[29] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit *et al.*, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," *Aaai/iaai*, vol. 593598, 2002.

[30] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*. IEEE Computer Society, 2007, pp. 1–10.

[31] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha, "Visual simultaneous localization and mapping: a survey," *Artificial Intelligence Review*, vol. 43, no. 1, pp. 55–81, 2015.

[32] R. Castle, G. Klein, and D. W. Murray, "Video-rate localization in multiple maps for wearable augmented reality," in *2008 12th IEEE International Symposium on Wearable Computers*. IEEE, 2008, pp. 15–22.

[33] S. Weiss, M. W. Achtelik, S. Lynen, M. C. Achtelik, L. Kneip, M. Chli, and R. Siegwart, "Monocular vision for long-term micro aerial vehicle state estimation: A compendium," *Journal of Field Robotics*, vol. 30, no. 5, pp. 803–831, 2013.

[34] G. Klein and D. Murray, "Improving the agility of keyframe-based SLAM," in *European Conference on Computer Vision*. Springer, 2008, pp. 802–815.

[35] A. Cunningham, M. Paluri, and F. Dellaert, "DDF-SAM: Fully distributed SLAM using constrained factor graphs," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 3025–3030.

[36] H. Zhang, X. Chen, H. Lu, and J. Xiao, "Distributed and collaborative monocular simultaneous localization and mapping for multi-robot systems in large-scale environments," *International Journal of Advanced Robotic Systems*, vol. 15, no. 3, p. 1729881418780178, 2018.

[37] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "DTAM: Dense tracking and mapping in real-time," in *2011 international conference on computer vision*. IEEE, 2011, pp. 2320–2327.

[38] A. Concha and J. Civera, "DPPTAM: Dense piecewise planar tracking and mapping from a monocular sequence," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 5686–5693.

[39] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European conference on computer vision*. Springer, 2006, pp. 430–443.

[40] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua, "BRIEF: Computing a local binary descriptor very fast," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 7, pp. 1281–1298, 2011.

[41] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," in *European conference on computer vision*. Springer, 2006, pp. 404–417.

[42] D. G. Lowe *et al.*, "Object recognition from local scale-invariant features." in *iccv*, vol. 99, no. 2, 1999, pp. 1150–1157.

[43] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast semi-direct monocular visual odometry," in *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2014, pp. 15–22.

[44] C. D. Herrera, K. Kim, J. Kannala, K. Pulli, and J. Heikkilä, "DT-SLAM: Deferred triangulation for robust SLAM," in *2014 2nd International Conference on 3D Vision*, vol. 1. IEEE, 2014, pp. 609–616.

[45] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *European conference on computer vision*. Springer, 2014, pp. 834–849.

[46] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment a modern synthesis," in *International workshop on vision algorithms*. Springer, 1999, pp. 298–372.

[47] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A general framework for graph optimization," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 3607–3613.

[48] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[49] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd, "Real time localization and 3D reconstruction," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 1. IEEE, 2006, pp. 363–370.

[50] H. Strasdat, A. J. Davison, J. M. Montiel, and K. Konolige, "Double window optimisation for constant time visual SLAM," in *2011 International Conference on Computer Vision*. IEEE, 2011, pp. 2352–2359.

[51] L. M. Paz, P. Piniés, J. D. Tardós, and J. Neira, "Large-scale 6-DOF SLAM with stereo-in-hand," *IEEE transactions on robotics*, vol. 24, no. 5, pp. 946–957, 2008.

[52] R. Mur-Artal and J. D. Tardós, "Fast relocalisation and loop closing in keyframe-based SLAM," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 846–853.

[53] D. Gálvez-López and J. D. Tardos, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012.

[54] O. D. Faugeras and F. Lustman, "Motion and structure from motion in a piecewise planar environment," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 2, no. 03, pp. 485–508, 1988.

[55] V. Lepetit, F. Moreno-Noguer, and P. Fua, "EPnP: An accurate O(n) solution to the PnP problem," *International journal of computer vision*, vol. 81, no. 2, p. 155, 2009.

[56] R. Mur-Artal and J. D. Tardós, "ORB-SLAM: Tracking and mapping recognizable features," in *Workshop on Multi View Geometry in Robotics (MVIGRO)-RSS*, 2014.

[57] H. Strasdat, J. Montiel, and A. J. Davison, "Scale drift-aware large scale monocular SLAM," *Robotics: Science and Systems VI*, vol. 2, no. 3, p. 7, 2010.

[58] B. K. Horn, "Closed-form solution of absolute orientation using unit quaternions," *Josa a*, vol. 4, no. 4, pp. 629–642, 1987.

[59] H. Strasdat, "Local accuracy and global consistency for efficient visual SLAM," Ph.D. dissertation, Department of Computing, Imperial College London, 2012.

[60] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[61] M. Cline, "What's this serialization thing all about?" *Serialization and Unserialization: C++ FAQ Lite*, vol. 2009, 1991.

[62] P. Bourhis, J. L. Reutter, F. Suárez, and D. Vrgoč, "JSON: data model, query languages and schema specification," in *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. ACM, 2017, pp. 123–135.

[63] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč, "Foundations of JSON schema," in *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2016, pp. 263–273.

[64] S. R. Schach, *Classical and Object-Oriented Software Engineering W/Uml and C++*. McGraw-Hill, Inc., 1998.

[65] J. Tang, W. Chen, and J. Wang, "A study on the P3P problem," in *International Conference on Intelligent Computing*. Springer, 2008, pp. 422–429.

[66] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[67] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger, "Elasticfusion: Real-time dense SLAM and light source estimation," *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1697–1716, 2016.

[68] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. J. Leonard, and J. McDonald, "Real-time large-scale dense RGB-D SLAM with volumetric fusion," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 598–626, 2015.

[69] C. Kerl, J. Sturm, and D. Cremers, "Dense visual SLAM for RGB-D cameras," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 2100–2106.

[70] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3-D mapping with an RGB-D camera," *IEEE transactions on robotics*, vol. 30, no. 1, pp. 177–187, 2013.

[71] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 3354–3361.

[72] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[73] M. Grupp, "evo: Python package for the evaluation of odometry and SLAM." https://github.com/MichaelGrupp/evo, 2017.