UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

DIEGO PITTOL

Exploração Autônoma utilizando SLAM Monocular Esparso

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Orientador: Prof. Dr. Edson Prestes e Silva Junior

Porto Alegre
2018

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Pittol, Diego

Exploração Autônoma utilizando SLAM Monocular Esparso / Diego Pittol. – Porto Alegre: PPGC da UFRGS, 2018.

85 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2018. Orientador: Edson Prestes e Silva Junior.

1. Exploração 3D, SLAM monocular, Nuvem de pontos. I. Silva Junior, Edson Prestes e. II. Título.

"O Universo [...] é um lugar desconcertantemente grande,
um fato que, para continuar levando uma vida tranquila,
a maioria das pessoas tende a ignorar."
— Douglas Adams

AGRADECIMENTOS

Aos meus pais, Valmor e Beatriz, por sempre valorizarem e enfatizarem a importância da educação, abrindo mão do seu conforto para me fornecer a oportunidade de chegar até aqui. À Vanessa, minha irmã, por ser meu exemplo, tanto na vida pessoal quanto na academia. À Anna Laura, por ser uma verdadeira companheira ao longo dessa e de todas jornadas.

Ao meu orientador, Edson, pela oportunidade, pelos ensinamentos e paciência. E a todos os meus colegas do *Phi-Group*, pelos conselhos, ajuda, almoços e risadas.

RESUMO

Nos últimos anos, observamos o alvorecer de uma grande quantidade de aplicações que utilizam robôs autônomos. Para que um robô seja considerado verdadeiramente autônomo, é primordial que ele possua a capacidade de aprender sobre o ambiente no qual opera. Métodos de SLAM (Localização e Mapeamento Simultâneos) constroem um mapa do ambiente por onde o robô trafega ao mesmo tempo em que estimam a trajetória correta do robô. No entanto, para obter um mapa completo do ambiente de forma autônoma é preciso guiar o robô por todo o ambiente, o que é feito no problema de exploração. Câmeras são sensores baratos que podem ser utilizadas para a construção de mapas 3D. Porém, o problema de exploração em mapas gerados por métodos de SLAM monocular, i.e. que extraem informações de uma única câmera, ainda é um problema em aberto, pois tais métodos geram mapas esparsos ou semi-densos, que são inadequados para navegação e exploração. Para tal situação, é necessário desenvolver métodos de exploração capazes de lidar com a limitação das câmeras e com a falta de informação nos mapas gerados por SLAMs monoculares. Propõe-se uma estratégia de exploração que utilize mapas volumétricos locais, gerados através das linhas de visão, permitindo que o robô navegue em segurança. Nestes mapas locais, são definidos objetivos que levem o robô a explorar o ambiente desviando de obstáculos. A abordagem proposta visa responder a questão fundamental em exploração: "Para onde ir?". Além disso, busca determinar corretamente quando o ambiente está suficientemente explorado e a exploração deve parar. A abordagem proposta é avaliada através de experimentos em um ambiente simples (i.e. apenas uma sala) e em um ambiente compostos por diversas salas.

Palavras-chave: Exploração 3D, SLAM monocular, Nuvem de pontos.

ABSTRACT

In recent years, we have seen the dawn of a large number of applications that use autonomous robots. For a robot to be considered truly autonomous, it is primordial that it has the ability to learn about the environment in which it operates. SLAM (Simultaneous Location and Mapping) methods build a map of the environment while estimating the robot's correct trajectory. However, to autonomously obtain a complete map of the environment, it is necessary to guide the robot throughout the environment, which is done in the exploration problem. Cameras are inexpensive sensors that can be used for building 3D maps. However, the exploration problem in maps generated by monocular SLAM methods (i.e. that extract information from a single camera) is still an open problem, since such methods generate sparse or semi-dense maps that are ill-suitable for navigation and exploration. For such a situation, it is necessary to develop exploration methods capable of dealing with the limitation of the cameras and the lack of information in the maps generated by monocular SLAMs. We proposes an exploration strategy that uses local volumetric maps, generated using the lines of sight, allowing the robot to safely navigate. In these local maps, objectives are defined to lead the robot to explore the environment while avoiding obstacles. The proposed approach aims to answer the fundamental question in exploration: "Where to go?". In addition, it seeks to determine correctly when the environment is sufficiently explored and the exploration must stop. The effectiveness of the proposed approach is evaluated in experiments on single and multi-room environments.

Keywords: 3D Exploration, Monocular SLAM, Point cloud.

LISTA DE ABREVIATURAS E SIGLAS

BA	*Bundle Adjustment*
BRIEF	*Binary Robust Independent Elementary Features*
DDA	*Digital Differential Analyzer*
GPU	*Graphics Processing Unit*
LPA*	*Lifelong Planning* A*
LSD-SLAM	*Large-Scale Direct Monocular SLAM*
NBV	*Next-Best-View*
ORB	*Oriented FAST and Rotated BRIEF*
PTAM	*Parallel Tracking and Mapping*
RANSAC	*Random Sample Consensus*
RGBD	*Red, Green, Blue, Depth*
RTA*	*Real-Time* A*
SLAM	*Simultaneous Localization and Mapping*
VANT	Veículo Aéreo Não Tripulado

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, observa-se o alvorecer de uma grande quantidade de aplicações que utilizam robôs autônomos nas mais diversas áreas, tais como aplicações humanitárias que visam salvar vidas e evitar riscos a estas (MADHAVAN et al., 2017; PRESTES et al., 2016; MADHAVAN et al., 2015; OKEREAFOR et al., 2013), aplicações militares (BHAT; MEENAKSHI, 2014; MAXWELL; LARKIN; LOWRANCE, 2013; NASKAR et al., 2011) e aplicações civis, como simples aspiradores de pó domésticos (HASAN; REZA et al., 2014). Ainda, os recentes avanços na área de carros autônomos, como os obtidos por Shim et al. (2015), Pettersson e Karlsson (2015) e Jo et al. (2015), possibilitam que em poucos anos tenhamos estes sendo utilizados em larga escala (ROSS, 2014).

Ter a capacidade de aprender sobre o ambiente no qual opera, sem interferência humana, é uma habilidade primordial para que robôs sejam considerados verdadeiramente autônomos. Segundo Oßwald et al. (2016) e Amigoni (2008), a aprendizagem de mapas do ambiente é um dos problemas fundamentais na robótica móvel, pois é necessário nas mais diversas áreas de aplicação que requerem que o robô conheça o ambiente em que está inserido, como transporte, limpeza, busca e resgate, localização e monitoramento. Em muitas aplicações, apesar de necessidade, o mapa do ambiente não está disponível previamente e os robôs devem ser capazes de contruí-lo de forma autônoma, em um processo de exploração. Inclusive, em diversas aplicações militares e civis, o objetivo do robô é explorar ativamente o ambiente e fornecer um mapa, podendo ser em 2D ou 3D, a um agente humano (CADENA et al., 2016).

Segundo Senarathne e Wang (2016), mapas 3D do ambiente são mais desejáveis e especialmente vantajosos para robôs que executarem tarefas complexas. Estes mapas 3D podem ser obtidos utilizando métodos monoculares de reconstrução de ambientes, com um custo (financeiro e computacional) mínimo de uma câmera simples, tornando-os um campo de interesse crescente (MOSTEGEL, 2013; SANTOSH; ACHAR; JAWAHAR, 2008). A ideia de robôs autônomos que utilizam apenas uma câmera simples não é recente e pode ser observada em trabalhos como Lewis (1998), Sobey (1994) e Nelson e Aloimonos (1989). Porém, sensores do tipo câmeras possuem um campo de visão limitado e segundo Shen (2014), um dos desafios da exploração de ambientes 3D com métodos monoculares (i.e. utilizando apenas uma

câmera) é que estes sensores não conseguem oferecer informações completas e precisas sobre o ambiente. Por este motivo, é necessário, segundo Kumar e Michael (2017), focar no desenvolvimento de novos métodos de exploração 3D que sejam capazes de lidar com as limitações das câmeras.

Esta dissertação aborda o problema de exploração autônoma de ambientes 3D utilizando apenas uma câmera monocular como sensor. Apesar de existirem diversos trabalhos na literatura que propõem estratégias de exploração, a maioria assume que o mapa do ambiente será denso. Porém, as técnicas de mapeamento monocular fornecem mapas esparsos (MUR-ARTAL; MONTIEL; TARDOS, 2015) ou semidensos (ENGEL; SCHÖPS; CREMERS, 2014; KLEIN; MURRAY, 2007). A fim de utilizar métodos de exploração já desenvolvidos, algumas abordagens (NUSKE et al., 2015; SANTOSH; ACHAR; JAWAHAR, 2008) simplificam o problema de exploração para 2D, enquanto outras (HOPPE et al., 2012; DUNN; FRAHM, 2009) tentam tornar o ambiente denso utilizando malhas, por exemplo. O trabalho de Stumberg et al. (2017) propõe um método de exploração 3D monocular, porém, este é altamente atrelado a técnica monocular utilizada no mapeamento e deixa diversos pontos abertos para melhorias, tais como a eficiência para permitir missões mais longas e a redução da necessidade de recursos computacionais permitindo explorações em ambientes grandes e complexos. Segundo Ström, Nenci e Stachniss (2015), a questão central na exploração é "Para onde ir?" e é esta questão que busca-se responder nesta dissertação.

1.1 Motivação

1.1.1 Como aprender o mapa de um ambiente?

Em geral, aprender o mapa de um ambiente requer a solução de três problemas: mapeamento, localização e planejamento de caminhos (STACHNISS, 2006). **Mapeamento** se refere ao problema de integrar as informações do ambiente obtidas pelos sensores do robô em uma representação estruturada, tendo como principais aspectos a representação do ambiente e a interpretação dos dados dos sensores. A **localização** é o problema de estimar a pose[1] do robô em relação a um mapa conhecido *a priori*. Este problema pode ser dividido em dois tipos, a localização

[1] Combinação entre a posição e a orientação do robô.

local, quando o estado inicial do robô é conhecido, e a localização global, quando o estado inicial é desconhecido. Por último, o **planejamento de caminhos** é o problema em que, dado um mapa do ambiente, deve-se guiar o robô do ponto em que ele se encontra até um destino, considerando a eficiência do trajeto de acordo com parâmetros definidos.

Segundo Stachniss (2006), estes três problemas não podem ser solucionados individualmente. Para realizar o mapeamento, por exemplo, o robô precisa associar suas leituras do ambiente ao mapa que está construindo e, para isso, precisa conhecer a pose, no mapa, que originou estas leituras. Da mesma forma, para determinar a pose correta de um robô é necessário, em geral, possuir um mapa do ambiente que possa ser comparado com o que o robô está observando. Os problemas de mapeamento e localização devem ser solucionados em paralelo, de forma que o mapa seja utilizado para localização ao mesmo tempo em que é construído (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011). Este problema é conhecido como **SLAM (*Simultaneous Localization and Mapping*)**. Técnicas de SLAM que utilizam apenas uma câmera para obter informações do ambiente, conhecidas como SLAM monocular, como as apresentadas por Mur-Artal, Montiel e Tardos (2015), Engel, Schöps e Cremers (2014) e Klein e Murray (2007) obtiveram sucesso em realizar o mapeamento de ambientes 3D. Porém, são dependentes de saliências na imagem para extrair a profundidade, fazendo com que gerem mapas esparsos ou semidensos, representados através de uma nuvem de pontos que contém apenas parte da informação sobre o ambiente.

Quando une-se planejamento de caminhos e mapeamento, temos o problema conhecido como **exploração**, que é o principal foco desta dissertação. Este problema consiste em planejar o movimento do robô de forma a obter mais informações do ambiente, podendo assim utilizá-las para construir o mapa. Segundo Meng et al. (2017), conforme aplicações com robôs que navegam autonomamente florescem, a exploração autônoma e reconstrução de ambientes têm se tornado uma tendência para a comunidade de pesquisadores. Todo sensor pode ser movido de forma autônoma permitindo que o ambiente seja mapeado, a principal questão é qual a melhor forma de movê-lo.

Apesar da definição de **exploração** ser a combinação de planejamento de caminhos e mapeamento, na prática, quando um robô deve explorar ambientes reais, o problema se torna uma combinação de **planejamento de caminhos** e **SLAM**.

Alguns autores, inclusive, dão diferentes nomes para esse problema, tais como Exploração Integrada ou SLAM Ativo (STACHNISS, 2006). Neste tipo de situação, muitas vezes as técnicas buscam guiar o robô por regiões que melhorem a qualidade do mapa, ao invés de simplesmente explorar o mais rápido possível.

Os movimentos do robô durante o processo de construção de mapas, quando não autônomo, são convencionalmente guiados por um operador humano e estão sujeitos à habilidade e subjetividade deste (MENG et al., 2017). Neste tipo de abordagem, a qualidade do mapa pode ser drasticamente afetada dependendo da movimentação feita pelo robô. Por exemplo, quando o robô permanece muito tempo andando por áreas com poucas características distinguíveis se torna muito difícil construir um mapa preciso. Na exploração, quando o processo de mapeamento é autônomo, o robô deve planejar seus movimentos para reduzir a incerteza sobre o mapa por meio de visitas a regiões desconhecidas. Espera-se que o robô inicie sem nenhum conhecimento prévio do ambiente e termine com este completamente mapeado, sem qualquer interferência de um operador.

1.1.2 Exploração autônoma monocular em ambientes 3D

Para que robôs possam executar tarefas em ambientes 3D complexos, é necessário que o mapa gerado também seja 3D (SENARATHNE; WANG, 2016). Técnicas de SLAM monoculares permitem a geração de mapas 3D através do uso de apenas uma câmera (MOSTEGEL, 2013; SANTOSH; ACHAR; JAWAHAR, 2008). Câmeras são sensores interessantes por permitirem o desenvolvimento de sistemas de baixo custo (SANTOSH; ACHAR; JAWAHAR, 2008). Além disso, segundo Engel, Sturm e Cremers (2014), câmeras são uma fonte rica de informação e, em comparação a sensores de profundidade (e.g. sonares e lasers), não possuem um intervalo mínimo e máximo de profundidades, permitindo seu uso em ambientes pequenos e grandes. Entretanto, como demonstrado a seguir, existem alguns desafios devido a incapacidade de mapear corretamente regiões sem saliências visuais utilizando apenas câmeras simples.

Para realizar uma exploração 3D, Song e Jo (2017) afirmam que os VANTs (Veículo Aéreo Não Tripulado) são os robôs ideais por possuírem grande manobrabilidade e poderem alcançar praticamente qualquer pose necessária. Shen (2014), na mesma linha, afirma que VANTs são a plataforma ideal para uma grande gama de

aplicações, tanto em ambientes internos quanto externos, devido a seu tamanho reduzido, grande mobilidade e capacidade de pairar. Ainda, segundo ele, é necessário que o VANT tenha capacidade de voar autonomamente, minimizando a necessidade de um operador. A exploração autônoma usando VANTs é claramente uma área rica em possibilidades de pesquisa (KUMAR; MICHAEL, 2017). Porém, segundo Heng et al. (2014), os VANTs são limitados quanto a sua capacidade de carga e bateria, o que torna as câmeras especialmente adequadas para realizar tarefas de exploração com estes robôs.

Mapas gerados utilizando apenas uma câmera, através de SLAMs monoculares, são mapas esparsos ou semidensos. A informação fragmentada destes mapas faz com que estratégias baseadas em fronteiras, muito tradicionais em exploração, falhem (SHEN; MICHAEL; KUMAR, 2012). Fronteiras são as bordas entre a região livre e a região desconhecida do mapa, e Yamauchi (1997) propôs a primeira abordagem de exploração utilizando-as. Diversas variações (COWLEY; TAYLOR; SOUTHALL, 2011; MOBARHANI et al., 2011; STACHNISS; MOZOS; BURGARD, 2008; VINCENT et al., 2008; MEI et al., 2006) foram propostas para aprimorar os resultados da exploração baseada em fronteiras. Todas estas, porém, são prejudicadas pela dificuldade na correta classificação das regiões em livres ou desconhecidas causada pelas informações incompletas em mapas esparsos ou semi densos (SHEN, 2014). Outro problema é que abordagens monoculares dependem fortemente de regiões salientes no ambiente para determinar corretamente a existência dos obstáculos (ROGGEMAN et al., 2017; SANTOSH; ACHAR; JAWAHAR, 2008). Dessa forma, é possível que obstáculos com poucas saliências visuais jamais sejam percebidos pelo SLAM. Neste caso, este obstáculo será sempre uma região desconhecida do mapa, e o robô pode ser, erroneamente, enviado para a fronteira entre o obstáculo não detectado e o espaço livre, causando uma colisão.

A maioria das abordagens de exploração atuais assumem que o mapa que está sendo construído é suficientemente consistente para ser utilizado no planejamento dos movimento do robô (STRÖM; BOGOSLAVSKYI; STACHNISS, 2017). O trabalho de Song e Jo (2017) utiliza uma câmera RGBD e inicialmente determina o objetivo com base no ganho de informação. Então, define um caminho até esse objetivo levando em consideração a cobertura da região ao longo do caminho. executando um algoritmo de inspeção para garantir que nenhuma lacuna (i.e. uma pequena falha em uma região de obstáculo) será deixada durante o trajeto. Shen,

Michael e Kumar (2012) simula a expansão de um sistema de partículas virtuais com dinâmica Newtoniana. O algoritmo determina como próxima fronteira a região onde o conjunto de partículas foi significativamente expandido. Ambas abordagens, porém, dependem de mapas densos e consistentes. Hoppe et al. (2012) e Dunn e Frahm (2009), por outro lado, buscam transformar o mapa esparso em denso, criando uma malha de triângulos com base na nuvem de pontos existente. Após, utilizam NBV[2] (*Next-Best-View*) para explorar, visando completar as lacunas da malha. Estas abordagens, porém, como as anteriores, executam a exploração sobre um mapa denso, gerado a partir da nuvem de pontos.

Outras abordagens simplificam o problema de mapeamento 3D tomando suas decisões sobre versões 2D do mapa. O trabalho de Fraundorfer et al. (2012) utiliza uma câmera estéreo e extrai uma fatia 2D do mapa 3D para executar a exploração, fixando a altura do voo na altura desta fatia. Então, sobre ela, extrai fronteiras como nos métodos tradicionais. Este trabalho foi posteriormente expandido por Heng et al. (2014) que melhorou alguns aspectos, dentre eles, o sistema de mapeamento. O trabalho proposto por Roggeman et al. (2017) utiliza uma câmera estéreo e visa evitar, durante a exploração, situações em que a localização possa ser perdida. Apesar de criar um mapa 3D, uma grade 2D é gerada removendo o solo, utilizando RANSAC (*Random Sample Consensus*), e projetando os obstáculos restantes sobre a grade. As decisões, então, são tomadas sobre esta. Chudoba et al. (2016) foca na exploração para construção de uma mapa que permita a localização de um VANT utilizando uma câmera apontada para o chão. Durante a exploração, o método prioriza a escolha de objetivos próximos, permitindo que a estimativa da pose seja melhorada constantemente através de retornos a áreas bem conhecidas do mapa. Porém, o mapa gerado é apenas um plano abaixo do robô. Estas abordagens são explorações 2D e não permitem que o robô modifique sua altura durante a exploração. Isto impede que ambientes 3D sejam explorados na vertical, não sendo possível obter um mapa realmente completo do ambiente.

[2]Técnicas baseadas em NBV buscam definir a próxima pose a ser atingida pelo robô que forneça o melhor ponto de vista do ambiente baseado em uma função de ganho definida.

1.1.3 Desafios da exploração monocular 3D

Vimos que para robôs serem realmente autônomos estes precisam possuir a capacidade de mapear o ambiente em que estão inseridos, através de uma exploração. Fica claro, também, que mapas 3D são necessários quando deseja-se que robôs operem em ambientes complexos (i.e. compostos por diversas salas e objetos). As câmeras são uma ótima opção de sensor para realizar o mapeamento 3D devido ao seu baixo custo e fácil acesso no mercado. Além disso, elas são o sensor ideal para utilização em VANTs (HENG et al., 2014), que são os robôs mais indicados para exploração 3D dado sua mobilidade (SONG; JO, 2017). Apesar de existirem algumas técnicas para realizar o SLAM utilizando apenas uma câmera monocular como sensor, ainda existe uma lacuna de abordagens de exploração que são verdadeiramente pensadas para o tipo de informação fornecida por estes SLAMs. O que vê-se são, na grande maioria, técnicas baseadas em fronteiras ou NBV, que presumem que o mapeamento será denso o suficiente para sua execução. Ambientes esparsos, assim como semidensos, são um desafio aos métodos de exploração por não fornecerem informações completas. A falta de informação pode acarretar, facilmente, na colisão do robô com um obstáculo não detectado no ambiente. Pode ainda, prendê-lo em uma busca incessante pelo fechamento de uma fronteira que não será fechada, uma vez que a região não possui saliências suficientes.

A grande motivação dessa dissertação é responder a questão central em exploração, como apontado por Ström, Nenci e Stachniss (2015): "Para onde ir?". Além desta pergunta, outros pontos importantes devem ser observados. É necessário que o robô saiba quando parar a exploração, ou seja, quando o ambiente está suficientemente mapeado. Também, espera-se que o robô não tome decisões que o levem a colidir com obstáculos. Então, o robô deve ser capaz de explorar o ambiente completamente, decidindo qual região visitar a cada momento, navegando de forma segura até esta região. Para isso, apresenta-se uma abordagem que determina a região livre do ambiente utilizando linhas de visão e define as regiões a serem visitadas utilizando os obstáculos encontrados.

1.2 Objetivos

O objetivo desta dissertação é propor um método de exploração 3D compatível com técnicas de SLAM monocular que geram representações do ambiente através de nuvem de pontos. O método deve permitir que robôs criem mapas 3D do ambiente em que estão inseridos de forma autônoma.

O método de exploração que se está propondo faz uso das linhas de visão formadas entre a pose que o robô estava quando detectou um ponto da nuvem, e a posição do próprio ponto visto, gerando um indício de que existe espaço livre ao longo dessa linha. É possível assumir então que uma região que possua diversos indícios de espaço livre pode ser atravessada pelo robô. Uma nuvem de pontos, porém, não é adequada para lidar com esse tipo de informação, pois não armazena informações sobre áreas livres. Por outro lado, uma grade de *voxels*[3] é adequada, já que é uma representação volumétrica do ambiente que representa tanto o espaço livre quanto ocupado e desconhecido. No entanto, gerar e armazenar uma representação volumétrica global utilizando uma grade de *voxels* demanda muitos recursos computacionais. Para contornar este problema, nossa abordagem propõe a utilização de um mapa volumétrico local durante a maior parte do processo de exploração. Este mapa local utiliza as informações da nuvem de pontos (i.e. do mapa global) para criar uma representação local do ambiente em uma grade de *voxels*. As linhas de visão são geradas para determinar os *voxels* livres e os pontos da nuvem são utilizados para determinar os *voxels* ocupados. Durante a grande maioria da exploração, é sobre este mapa local que a escolha da próxima região a ser visitada é tomada. O mapa global é utilizado, em termos de planejamento, apenas quando o mapa local não possuir regiões a serem visitadas. Então, o mapa global é utilizado para determinar se existe algum ponto, globalmente, que ainda tenha regiões em aberto.

Segundo Shen, Michael e Kumar (2012), o problema da exploração pode ser dividido em duas partes: definir as regiões que devem ser visitadas para estender o mapa e navegar de forma autônoma até estas regiões. Para navegar de forma segura pelo ambiente, o método proposto planeja caminhos que passem apenas pelos *voxels* livres do mapa local, utilizando o algoritmo A* (HART; NILSSON; RAPHAEL, 1968), propondo para este algumas modificações tornando o caminho gerado mais

[3] Um *voxel* representa um valor em uma grade regular 3D.

seguro. Para definir qual região visitar, segundo Shade e Newman (2011), um dos desafios é atribuir um valor numérico a cada região candidata, definindo qual é mais promissora. O método que se está propondo, visando evitar que o robô seja enviado para regiões totalmente desconhecidas, utiliza uma abordagem que o move pelo ambiente para regiões com maior probabilidade de serem alcançáveis. Para definir esta probabilidade, analisa-se a aglomeração de pontos da nuvem. A região mais atrativa é considerada aquela que, no mapa local, possui o maior aglomerado de pontos ainda não visitados. Um ponto é considerado visitado quando o robô chega a uma certa distância mínima deste ponto. Essa abordagem, porém, define na maioria dos casos o objetivo como sendo um *voxel* ocupado. Assim, o planejamento de caminhos não é feito até o objetivo em si, mas sim até uma região livre próxima a ele, dentro de uma margem de segurança. Durante a navegação, todos pontos que estiverem a uma distância do robô menor que essa margem de segurança são considerados visitados.

O ideal, segundo Mostegel (2013), para evitar que o robô colida durante a navegação, é que ele se mova apenas para frente, sempre olhando na direção que está indo. Porém, movendo-se apenas para frente não há um real ganho de informação, já que o sensor é uma câmera com campo de visão limitado, e esta só estará se aproximando e reduzindo a área capturada pela imagem. Ainda, mover-se apenas para a frente pode ser insuficiente para fornecer pontos de vista suficientemente diferentes que resultem em boas estimativas de profundidade dos pontos pelo SLAM monocular.

Assim, outro problema a ser abordado é para onde direcionar o sensor durante a navegação. Esse problema só deve ser respondido para a orientação em torno do eixo vertical, conhecida como *yaw*. No escopo deste trabalho, a possibilidade de o robô mover a câmera em torno dos outros eixos não é contemplada. A abordagem proposta visa determinar, primeiramente, qual é o ângulo em que o robô deve estar orientado no último passo do caminho para que esteja olhando diretamente para o objetivo. Então, determina-se que o robô chegue ao passo final olhando 90º para um dos lados em relação a este ângulo. O lado é definido através daquele que necessita da menor rotação para ser atingido, pois, movimentos de rotação tendem a prejudicar os métodos de SLAM monocular (MOSTEGEL, 2013). Então, pequenas rotações são distribuídas ao longo do caminho rotacionando o robô enquanto realiza movimentos de translação pelo ambiente.

Por fim, o método proposto define que a exploração termina quando uma quantidade miníma dos pontos do mapa for visitada. É necessário utilizar apenas uma parte para determinar a completude do processo, uma vez que os mapas gerados em SLAMs monoculares tendem a possuir muitos *outliers*. Estes são, normalmente, pontos que não representam um objeto real e encontram-se em posições inalcançáveis do ambiente. É possível que em determinado momento não existam novos objetivos para a exploração no mapa local. Então, o método analisa se existe algum ponto no mapa global que possua pontos em aberto suficientes para prosseguir com a exploração, e caso não haja, esta é encerrada.

Em suma, as principais contribuições dessa dissertação são:

- Uma forma de definir para onde ir em uma exploração 3D, utilizando apenas informações esparsas sobre o ambiente e evitando colocar o robô em situações de risco.
- Um critério de parada que evite que a exploração termine precocemente, mas também, consiga, na presença de *outliers*, determinar que o ambiente foi completamente explorado.
- Uma abordagem de exploração utilizando mapas locais na maioria do processo, evitando gerar dispendiosos mapas volumétricos globais, reduzindo assim o custo computacional do método.

1.3 Organização

Esta dissertação é dividida da seguinte forma. O Capítulo 2 contextualiza e formaliza os problemas fundamentais no escopo de exploração. Também, apresenta em detalhes as técnicas utilizadas como base para o trabalho proposto. Por fim, apresenta alguns trabalhos relacionados.

O Capítulo 3 aborda o método de exploração proposto. Inicialmente é apresentada a estrutura básica do sistema, seguido da explicação a respeito do uso das linhas de visão para gerar o mapa local. Então, cada etapa do processo de exploração é abordada de forma detalhada. Este capítulo encerra-se com a apresentação do planejamento de caminhos.

No Capítulo 4 são apresentados e discutidos os experimentos realizados. Inicialmente, compara-se a geração do mapa local utilizando apenas parte da informação

do mapa global. Então, são apresentados testes do sistema completo em um ambiente simples, composto por apenas uma sala. Na sequência, testes em um ambiente complexo, composto por múltiplas salas, é apresentado. Finalizando, o Capítulo 5 contém nossas conclusões, assim como o que pode ser feito para dar continuidade a este trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão contextualizados os problemas fundamentais no escopo de exploração, assim como as técnicas e métodos utilizados para solucioná-los. Inicialmente, na Seção 2.1 é apresentado o problema de Planejamento de Caminhos e o algoritmo utilizado como base para a solução proposta. Então, nas Seções 2.2 e 2.3 são apresentados os problemas de Mapeamento e Localização, respectivamente. Estes dois problemas são a base necessária para abordarmos o problema de SLAM, apresentado na Seção 2.4, juntamente com o método de SLAM Monocular utilizado neste trabalho. Na Seção 2.5 é apresentado o problema de exploração, foco principal desta dissertação. Por fim, na Seção 2.6 são apresentados alguns trabalhos relacionados.

A Tabela 2.1[1] apresenta um resumo da formalização matemática utilizada nesta dissertação.

Tabela 2.1: Nomenclaturas utilizadas.

Termo	Significado
$\boldsymbol{x}$	Pose do robô composta pela posição, em coordenadas cartesianas, e pela orientação em torno do eixo vertical (*yaw*). Quando associada a um instante t é descrita como $\boldsymbol{x}_t$, podendo ser agrupadas em um conjunto de poses discretizadas no tempo de 0 até t, formando uma trajetória $\boldsymbol{x}_{0:t} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_t\}$.
$\boldsymbol{X}$	Conjunto de todas possíveis poses $\boldsymbol{x}$.
$\boldsymbol{X}_{livre}$	Conjunto $\boldsymbol{X}_{livre} \subseteq \boldsymbol{X}$ formado pelas poses livres de obstáculo.
$\boldsymbol{v}_i$	O *i-ézimo* elemento de um mapa, sendo $\boldsymbol{v}_i, i = 1, 2, ..., n$, a célula centrada nas coordenadas cartesianas $(x, y)^T$ quando 2D, ou o *voxel* centrado nas coordenadas cartesianas $(x, y, z)^T$ quando 3D.
$\boldsymbol{V}$	Mapa $\boldsymbol{V} = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3, ..., \boldsymbol{v}_n\}$ do ambiente representado por uma grade regular, sendo n o número de elementos.
$\boldsymbol{V}_{livre}$	Conjunto $\boldsymbol{V}_{livre} \subseteq \boldsymbol{V}$ formado pela região livre.
$\boldsymbol{V}_{ocupado}$	Conjunto $\boldsymbol{V}_{ocupado} \subseteq \boldsymbol{V}$ formado pela região ocupada.
$\boldsymbol{V}_{desconhecido}$	Conjunto $\boldsymbol{V}_{desconhecido} \subseteq \boldsymbol{V}$ formado pela região desconhecida.

Continua na próxima página

[1] A notação que será utilizada nesta dissertação para se referir a componentes de variáveis ou vetores é: *componente*(***variável***). Por exemplo, o valor g do elemento $\boldsymbol{v}$ é dado por $g(\boldsymbol{v})$.

Tabela 2.1 – Continua na página anterior

Termo	Significado
$\boldsymbol{v}_p$, $\boldsymbol{v}_d$	Elementos do mapa representando origem e destino.
$g(\boldsymbol{v}_i)$	Custo para chegar em $\boldsymbol{v}_i$ partindo de $\boldsymbol{v}_p$.
$h(\boldsymbol{v}_i)$	Custo estimado para chegar em $\boldsymbol{v}_d$ partindo de $\boldsymbol{v}_i$.
$f(\boldsymbol{v}_i)$	$f(\boldsymbol{v}_i) = g(\boldsymbol{v}_i) + h(\boldsymbol{v}_i)$.
$d(\boldsymbol{v}, \boldsymbol{v}')$	Distância euclidiana entre $\boldsymbol{v}$ e $\boldsymbol{v}'$.
$u(\boldsymbol{v}_i)$	Custo de liberdade para navegação na vizinhança de $\boldsymbol{v}_i$.
$\boldsymbol{m}_l$	Posição do *l-ézimo landmark*. O conjunto de *landmarks* é definido por $\boldsymbol{m} = \{\boldsymbol{m}_1, \boldsymbol{m}_2, ..., \boldsymbol{m}_j\}$, sendo j o número total de *landmarks*.
$\boldsymbol{z}$	Leitura, composta por observações do ambiente, obtida pelo sensor do robô. No instante t, $\boldsymbol{z}_t = \{\boldsymbol{z}_{1,t}, \boldsymbol{z}_{2,t}, ..., \boldsymbol{z}_{j,t}\}$. É possível especificar uma leitura de um determinado *landmark* através da notação $\boldsymbol{z}_{l,t}$. As leituras podem ser agrupadas, gerando um conjunto $\boldsymbol{z}_{0:t} = \{\boldsymbol{z}_0, \boldsymbol{z}_1, \boldsymbol{z}_2, ..., \boldsymbol{z}_t\}$.
$\boldsymbol{u}_t$	Informação de controle de movimento que corresponde a mudança da pose do robô do instante t-1 para o instante t. A sequência destas informações é representada por $\boldsymbol{u}_{1:t} = \{\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3, ..., \boldsymbol{u}_t\}$.
$\boldsymbol{k}_a$	O *a-ézimo keyframe*.
$\boldsymbol{K}$	Conjunto $\boldsymbol{K} = \{\boldsymbol{k}_1, \boldsymbol{k}_2, \boldsymbol{k}_3, ..., \boldsymbol{k}_r\}$, sendo r o número de *keyframes*.
$\boldsymbol{T}$	Pose da câmera. Quando associada a um *keyframe* $\boldsymbol{k}_a$ é descrita por $\boldsymbol{T}_a$.
$\boldsymbol{p}_b$	O *b-ézimo* ponto de uma nuvem de pontos.
$\boldsymbol{P}$	Conjunto $\boldsymbol{P} = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, ..., \boldsymbol{p}_s\}$, sendo s o número de pontos.
$\hat{\boldsymbol{P}}_a$	Conjunto $\hat{\boldsymbol{P}}_a \subseteq \boldsymbol{P}$ de pontos vistos pelo *keyframe* $\boldsymbol{k}_a$.
$\hat{\boldsymbol{K}}_b$	Conjunto $\hat{\boldsymbol{K}}_b \subseteq \boldsymbol{K}$ de *keyframes* que conseguem ver o ponto $\boldsymbol{p}_b$.
θ	Peso de uma aresta no grafo de covisibilidade.
$\boldsymbol{k}_{ref}$	O *keyframe* de referência.
$\mathcal{K}_a$	Conjunto $\mathcal{K}_a \subseteq \boldsymbol{K}$ de vizinhos de $\boldsymbol{k}_a$ no grafo de covisibilidade.
$\hat{\mathcal{K}}$	Conjunto $\hat{\mathcal{K}} \subseteq \boldsymbol{K}$ de *keyframes* utilizados na geração do mapa local.
ξ	Tamanho da aresta de um *voxel* no mapa local.
κ	Quantidade mínima de *keyframes* para geração do mapa local.
$\boldsymbol{y}_i$	Posição em coordenadas cartesianas $(x, y, z)^T$ do centro do *voxel* $\boldsymbol{v}_i$.

Continua na próxima página

Tabela 2.1 – Continua na página anterior

Termo	Significado
β_i	Quantidade de indícios de que o *voxel* $\boldsymbol{v}_i$ está livre, limitado a $\ddot{\beta}$.
$\mathcal{P}_i$	Os pontos $\mathcal{P}_i \subseteq \boldsymbol{P}$ que estão dentro da área do *voxel* $\boldsymbol{v}_i$.
$\hat{\mathcal{P}}_i$	Os pontos $\hat{\mathcal{P}}_i \subseteq \mathcal{P}_i$ não visitados.
$e(\boldsymbol{v}_i)$	Retorna o estado do *voxel* $\boldsymbol{v}_i$.
λ	Mínimo $\lvert\mathcal{P}_i\rvert$ para considerar o *voxel* $\boldsymbol{v}_i$ ocupado.
μ	Valor mínimo de β_i para considerar o *voxel* $\boldsymbol{v}_i$ livre.
$m(\boldsymbol{k}_a)$	Retorna o estado do *keyframe* $\boldsymbol{k}_a$, pode ser *aberto* ou *fechado*.
ϱ	Percentual mínimo de pontos não visitados em $\hat{\boldsymbol{P}}_a$ para o *keyframe* $\boldsymbol{k}_a$ ser considerado aberto.
$a(\boldsymbol{v}_i)$	Retorna a atratividade do *voxel* $\boldsymbol{v}_i$.
$n(\boldsymbol{v}_i)$	Retorna se um *voxel* $\boldsymbol{v}_i$ é interessante ou não.
α	Mínimo $\lvert\hat{\mathcal{P}}_i\rvert$ para que $\boldsymbol{v}_i$ seja interessante.
ϑ	Proporção mínima entre $\lvert\hat{\mathcal{P}}_i\rvert$ e $\lvert\mathcal{P}_i\rvert$ para que $\boldsymbol{v}_i$ seja interessante.
ψ	Orientação em torno do eixo vertical (*yaw*).
$\Delta\psi$	Variação mínima em torno do eixo vertical (*yaw*).
η	Distância miníma para considerar um *voxel* visitado.
$\mathcal{L}_i$	Subconjunto $\mathcal{L}_i \subseteq \boldsymbol{V}$ formado pelos vizinhos do *voxel* $\boldsymbol{v}_i$.
ω	Variação aplicada sobre os índices de um *voxel* formando a vizinhança utilizada na distribuição da atração.
$\mathcal{C}$	Caminho $\mathcal{C} = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3, ..., \boldsymbol{v}_q\}$, sendo q o número de passos.

2.1 Planejamento de caminhos

Planejamento de caminhos é o problema que consiste em, eficientemente, guiar um robô da pose que ele se encontra até uma pose específica no ambiente (STACHNISS, 2006). Dado que uma pose do robô no ambiente é representada por $\boldsymbol{x}$, pode-se agrupar todas as possíveis poses do robô em $\boldsymbol{X}$. Então, o problema pode ser definido como encontrar um conjunto de poses que levem o robô de $\boldsymbol{x}_p$ até $\boldsymbol{x}_d$, sendo estas as poses de partida e destino, respectivamente.

Para resolver este problema, é necessário que o robô possua um mapa do ambiente *a priori*. Assumindo um mapa 2D discreto e regular, pode-se representá-

lo por $\boldsymbol{V} = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3, ..., \boldsymbol{v}_n\}$, onde n é o número de elementos do mapa, e $\boldsymbol{v}_i$, sendo $i = 1, 2, ..., n$, é o elemento do mapa nas coordenadas cartesianas $(x, y)^T$. Cada elemento $\boldsymbol{v}_i$ possui um estado, que pode ser: livre, ocupado ou desconhecido. Pode-se então definir um conjunto $\boldsymbol{V}_{livre} \subseteq \boldsymbol{V}$ composto pelos elementos de $\boldsymbol{V}$ que estejam livres e podem ser visitados pelo robô.

As possíveis poses $\boldsymbol{X}$ podem ser obtidas através de $\boldsymbol{V}_{livre}$, de maneira que cada $\boldsymbol{x} \in \boldsymbol{X}$ possua um equivalente $\boldsymbol{v} \in \boldsymbol{V}$ em termos de posição. Então, pode-se definir $\boldsymbol{X}_{livre}$ como o conjunto de poses geradas pelo conjunto $\boldsymbol{V}_{livre}$[2] de elementos livres do mapa.

Para solucionar o problema de planejamento de caminhos, pode-se criar uma representação do ambiente em forma de um grafo onde os nodos são os elementos de $\boldsymbol{V}_{livre}$ e uma aresta é criada entre $\boldsymbol{v}_i \in \boldsymbol{V}_{livre}$ e todos os seus vizinhos $\boldsymbol{v}_i' \in \boldsymbol{V}_{livre}$. Os pontos de partida $\boldsymbol{v}_p$ e destino $\boldsymbol{v}_d$ são definidos como nodos especiais do grafo. Então, deve-se encontrar o conjunto de nodos que levem do nodo de partida ao nodo destino.

2.1.1 Algoritmo A*

Esta subseção apresenta o algoritmo **A***, proposto por Hart, Nilsson e Raphael (1968), que é uma das soluções mais conhecidas para o problema de planejamento de caminhos sobre uma representação do ambiente em forma de grade. Conforme visto na Seção anterior, esta grade pode ser representada como um grafo. Este algoritmo, com algumas modificações descritas no Capítulo 3, foi utilizado na solução proposta nesta dissertação.

O Algoritmo 1 apresenta o funcionamento do A*. Nele, é necessário manter duas estruturas globais:

- $\boldsymbol{V}_{aberto}$ é o conjunto de nodos $\boldsymbol{v}_i \in \boldsymbol{V}_{livre}$ a serem analisados.
- $\boldsymbol{V}_{fechado}$ é o conjunto de nodos $\boldsymbol{v}_i \in \boldsymbol{V}_{livre}$ que foram analisados anteriormente.

Além disso, para cada $\boldsymbol{v}_i \in \boldsymbol{V}_{livre}$ são utilizados alguns valores:

- $g(\boldsymbol{v}_i)$: custo do caminho para chegar em $\boldsymbol{v}_i$ partindo de $\boldsymbol{v}_p$.
- $h(\boldsymbol{v}_i)$: custo estimado por uma heurística qualquer para chegar em $\boldsymbol{v}_d$ partindo de $\boldsymbol{v}_i$. Normalmente, utiliza-se a distância euclidiana entre $\boldsymbol{v}_i$ e $\boldsymbol{v}_d$ como heu-

[2] Diferentes poses podem ser mapeadas para um mesmo elemento $\boldsymbol{v}$.

Algoritmo 1: Algoritmo A* para planejamento de caminhos.

Entrada: $\boldsymbol{v}_p$, $\boldsymbol{v}_d$; // Pontos de partida e destino
Saída: *Caminho* ; // ∅ caso não exista um caminho

$\boldsymbol{V}_{aberto} = \emptyset$;
$\boldsymbol{V}_{fechado} = \emptyset$;
$g(\boldsymbol{v}_p) = 0$;
$h(\boldsymbol{v}_p) = \mathit{Heurística}(\boldsymbol{v}_p, \boldsymbol{v}_d)$;
$f(\boldsymbol{v}_p) = g(\boldsymbol{v}_p) + h(\boldsymbol{v}_p)$;
$\mathit{pai}(\boldsymbol{v}_p) = \boldsymbol{v}_p$;
Insere $\boldsymbol{v}_p$ em $\boldsymbol{V}_{aberto}$;
enquanto $\boldsymbol{V}_{aberto} \neq \emptyset$ **faça**
    $\boldsymbol{v}_c$ = Elemento $\boldsymbol{v}$ de $\boldsymbol{V}_{aberto}$ que minimize $f(\boldsymbol{v})$;
    se $\boldsymbol{v}_c = \boldsymbol{v}_d$ **então**
        retorna *RastrearCaminho*();
    Remove $\boldsymbol{v}_c$ de $\boldsymbol{V}_{aberto}$;
    Insere $\boldsymbol{v}_c$ em $\boldsymbol{V}_{fechado}$;
    para *cada vizinho* $\boldsymbol{v}_a \in \boldsymbol{V}_{livre}$ *de* $\boldsymbol{v}_c$ **faça**
        se $\boldsymbol{v}_a \notin \boldsymbol{V}_{fechado}$ **então**
            se $\boldsymbol{v}_a \notin \boldsymbol{V}_{aberto}$ **então**
                $g(\boldsymbol{v}_a) = \infty$;
                Insere $\boldsymbol{v}_a$ em $\boldsymbol{V}_{aberto}$;
            $\mathit{custo} = g(\boldsymbol{v}_c) + d(\boldsymbol{v}_c, \boldsymbol{v}_a)$;
            se $\mathit{custo} < g(\boldsymbol{v}_a)$ **então**
                $g(\boldsymbol{v}_a) = \mathit{cost}$;
                $\mathit{pai}(\boldsymbol{v}_a) = \boldsymbol{v}_c$;
                $h(\boldsymbol{v}_a) = \mathit{Heurística}(\boldsymbol{v}_a, \boldsymbol{v}_d)$;
                $f(\boldsymbol{v}_a) = g(\boldsymbol{v}_a) + h(\boldsymbol{v}_a)$;
    fim
fim
retorna ∅ ;
Função *RastrearCaminho*():
    $\mathit{Caminho} = \emptyset$;
    $\boldsymbol{v}_c = \boldsymbol{v}_d$;
    Insere $\boldsymbol{v}_c$ em *Caminho* ;
    enquanto $\mathit{pai}(\boldsymbol{v}_c) \neq \boldsymbol{v}_c$ **faça**
        $\boldsymbol{v}_c = \mathit{pai}(\boldsymbol{v}_c)$;
        Insere $\boldsymbol{v}_c$ em *Caminho* ;
    fim
retorna *Caminho* ;

rística. É importante que essa heurística nunca superestime a real distância.

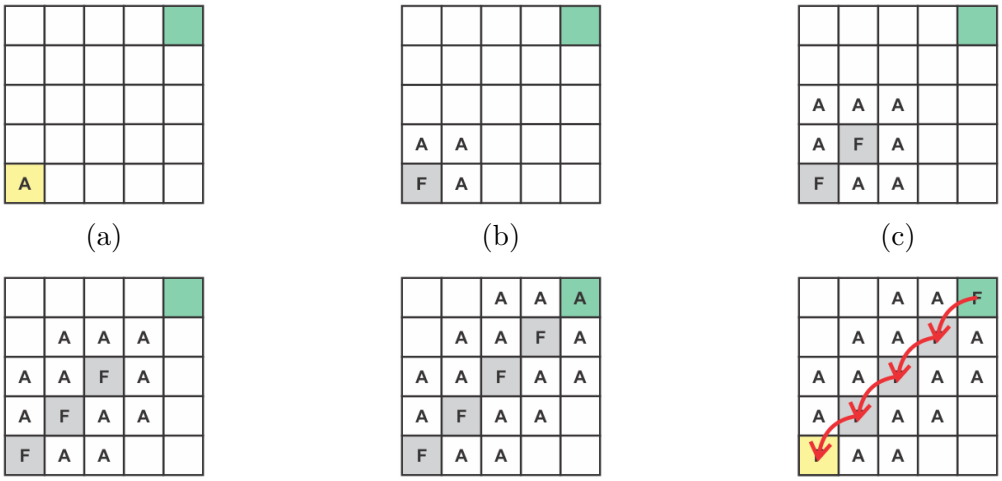
- $f(\boldsymbol{v}_i)$: custo hipotético para chegar em $\boldsymbol{v}_d$, através de $\boldsymbol{v}_i$, partindo de $\boldsymbol{v}_p$. É composto pelo custo para ir de $\boldsymbol{v}_p$ até $\boldsymbol{v}_i$ e pelo custo estimado entre $\boldsymbol{v}_i$ e $\boldsymbol{v}_d$, sendo $f(\boldsymbol{v}_i) = g(\boldsymbol{v}_i) + h(\boldsymbol{v}_i)$
- $pai(\boldsymbol{v}_i)$ é o elemento $\boldsymbol{v}_i' \in \boldsymbol{V}_{livre}$ pai de $\boldsymbol{v}_i$. Nele é armazenado o melhor elemento anterior, ou seja, o $\boldsymbol{v}_i'$ que minimiza $g(\boldsymbol{v}_i)$.

O algoritmo, inicialmente, insere $\boldsymbol{v}_p$ em $\boldsymbol{V}_{aberto}$, sendo $g(\boldsymbol{v}_p) = 0$ e $pai(\boldsymbol{v}_p) = \boldsymbol{v}_p$, linhas 3 a 7. O valor de $h(\boldsymbol{v}_p)$ é obtido através de uma heurística que busca acelerar o processo, permitindo determinar qual elemento de $\boldsymbol{V}_{aberto}$ é mais promissor. Normalmente é utilizado como heurística a distância euclidiana até $\boldsymbol{v}_d$. O algoritmo itera sobre $\boldsymbol{V}_{aberto}$ até encontrar um caminho que leve a $\boldsymbol{v}_d$ ou $\boldsymbol{V}_{aberto} = \emptyset$. É extraído, a cada iteração, o $\boldsymbol{v}_c$ de $\boldsymbol{V}_{aberto}$ que minimize $f(\boldsymbol{v}_c)$, linhas 8 e 9. Para cada $\boldsymbol{v}_c$ são executados os seguintes passos: se $\boldsymbol{v}_c$ for igual a $\boldsymbol{v}_d$, então retorna o caminho e encerra. Caso contrário, remove $\boldsymbol{v}_c$ de $\boldsymbol{V}_{aberto}$ e o insere em $\boldsymbol{V}_{fechado}$, uma vez que ele está sendo analisado. Então, para cada vizinho $\boldsymbol{v}_a \in \boldsymbol{V}_{livre}$ de $\boldsymbol{v}_c$ que não esteja em $\boldsymbol{V}_{fechado}$ são executados os seguintes passos: se $\boldsymbol{v}_a \notin \boldsymbol{V}_{aberto}$, então ele é inserido e são determinados seus custos e atribuído $pai(\boldsymbol{v}_a) = \boldsymbol{v}_c$, linhas 19 a 24. Se $\boldsymbol{v}_a \in \boldsymbol{V}_{aberto}$, linha 20, e o caminho até $\boldsymbol{v}_a$ gerado por $\boldsymbol{v}_c$ minimizar $g(\boldsymbol{v}_a)$, então os custos de $\boldsymbol{v}_a$ são atualizados e $\boldsymbol{v}_c$ é atribuído como seu novo pai. Por fim, uma nova iteração começa analisando um novo elemento de $\boldsymbol{V}_{aberto}$. Caso um caminho seja encontrado, este pode ser recuperado apenas observando o pai de cada elemento partindo de $\boldsymbol{v}_d$, linhas 26 a 33.

A Figura 2.1 apresenta um exemplo de execução do algoritmo. O ponto de partida é representado pela célula amarela e o destino pela célula verde. Inicialmente, na Figura 2.1(a) o ponto de partida é colocado na lista $\boldsymbol{V}_{aberto}$, representado pelo "A" sobre a célula. Então, na Figura 2.1(b), o melhor elemento (i.e. que minimiza $f(\boldsymbol{v})$) é retirado de $\boldsymbol{V}_{aberto}$ e colocado em $\boldsymbol{V}_{fechado}$, representado pelo "F" sobre a célula, assim como o tom cinza do fundo. Então, todos os vizinhos são colocados em $\boldsymbol{V}_{aberto}$. Na Figura 2.1(c) e Figura 2.1(d) o processo é repetido, com um novo melhor elemento de $\boldsymbol{V}_{aberto}$ sendo escolhido a cada passo, sendo retirado de $\boldsymbol{V}_{aberto}$ e colocado em $\boldsymbol{V}_{fechado}$ e seus vizinhos sendo colocados em $\boldsymbol{V}_{aberto}$ caso não estejam em $\boldsymbol{V}_{fechado}$. Na Figura 2.1(e), após repetir mais uma vez o processo, o destino é colocado em $\boldsymbol{V}_{aberto}$. Por fim, na Figura 2.1(f) o destino é selecionado como melhor elemento de $\boldsymbol{V}_{aberto}$, encerrando o processo. O caminho, representado em vermelho,

é recuperado indo recursivamente, de pai em pai, começando pelo pai do destino.

Figura 2.1: Exemplo de execução do algoritmo A*.



2.1.2 Variações do A*

Diversas variações ao algoritmo A* estão disponíveis na literatura. Koenig e Likhachev (2002b) propõe uma versão incremental do A*, denominada *Lifelong Planning* A* (LPA*), que permite replanejar o caminho, quando o ambiente sofre uma alteração, de forma mais eficiente. Sempre que necessário, um novo caminho que leve da origaem ao destino é gerado. Para tal, utiliza uma lista de elementos a serem atualizados, fazendo com que apenas alguns elementos de todo mapa sejam reprocessador, utilizando nestas atualizações as informações previamente obtidas no planejamento inicial. Em outro trabalho, Koenig e Likhachev (2002a) propõe o D*Lite, também visando acelerar o replanejamento de caminhos em ambientes dinâmicos. Neste, porém, o caminho é gerado sempre entre o elemento do mapa correspondente a posição do robô e o destino, de forma diferente ao LPA*, que sempre considera a origem, mesmo nos replanejamentos. Esta abordagem procura um caminho na direção oposta, ou seja, partindo do destino. Assim, o custo $g(\boldsymbol{v}_i)$ apresentado anteriormente passa a ser a distância até o objetivo.

O trabalho de Daniel et al. (2010) visa reduzir o tamanho do caminho gerado pelo A*. Quando aplicado sobre grades, o A* considera apenas movimentos em ângulos de 45°, ou seja, movimentos para o centro de um de seus oito vizinhos. Daniel et al. (2010) propõe considerar, também, movimentos que sejam realizados em

qualquer ângulo, tornando possível encontrar caminhos menores. Para tal, é permitido que o pai de um elemento seja qualquer outro elemento que possua uma linha de visão direta com ele, e não apenas um de seus oito vizinhos. O trabalho de Korf (1990), denominado *Real-time* A* (RTA*), busca resolver o planejamento em tempo-real, ou seja, em que o robô possua uma restrição de tempo e deve definir uma ação a ser realizada ao final de cada período. Para tal, a quantidade de possíveis caminhos considerados é reduzida e uma ação é escolhida em um intervalo fixo de tempo. Caso um caminho ainda não tenha sido encontrado, como ação, será escolhido o melhor resultado obtido até então. Sefer, Kuter e Nau (2009) propôs modificações ao RTA* evitando que seja escolhida uma ação que coloque o robô em uma situação de falha.

2.2 Mapeamento

No problema de **mapeamento**, assume-se que o robô possui conhecimento preciso de sua pose, porém, não possui nenhum conhecimento sobre o ambiente ao seu redor, ou seja, não possui um mapa. O robô deve, então, construir uma representação do ambiente que contenha as informações dele, tais como obstáculos, região livre, e outras informações que possam ser relevantes (e.g. regiões perigosas, objetos, etc). A construção do mapa deve ser realizada pelo robô através das informações obtidas por seus sensores exteroceptivos[3], que podem ser câmeras, sonares, lasers, entre outros.

Para a tarefa de mapeamento, independente do sensor que o robô possua, cada leitura $\boldsymbol{z}_t$ obtida pelo sensor está associada a pose $\boldsymbol{x}_t$ em que o robô estava quando a obteve. Temos, então, um conjunto de poses assumidas pelo robô discretizadas no tempo de 0 até t, representadas por $\boldsymbol{x}_{0:t} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_t\}$, e um conjunto com suas respectivas leituras $\boldsymbol{z}_{0:t} = \{\boldsymbol{z}_0, \boldsymbol{z}_1, \boldsymbol{z}_2, ..., \boldsymbol{z}_t\}$. É necessário, agora, representar o ambiente, organizando as informações obtidas pelas leituras $\boldsymbol{z}_{0:t}$ nas poses $\boldsymbol{x}_{0:t}$ em um mapa.

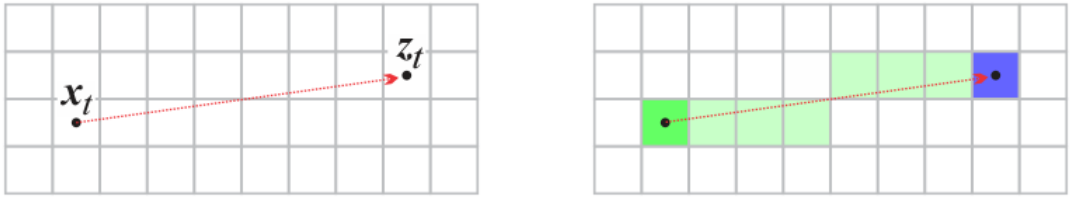
As leituras $\boldsymbol{z}_{0:t}$ podem ser mapeadas em dois tipos de representação do ambiente: mapas métricos e mapas topológicos. **Mapas métricos** representam as propriedades do ambiente de forma geométrica e geralmente discretizam o ambiente,

[3]Sensores exteroceptivos adquirem informações sobre a área de atuação do robô, como medidas de distância, luminosidade do ambiente, etc.

podendo ser em uma grade 2D ou um mapa volumétrico 3D. **Mapas topológicos** representam regiões do ambiente e suas conexões, geralmente através de um grafo, onde os nodos são os locais do ambiente (i.e. as poses do robô) e as arestas, a forma com que eles se conectam. As representações topológicas são uma forma eficiente, em termos de memória, de representar o ambiente, permitindo seu uso em ambientes grandes e complexos. No entanto, estas não representam diretamente os espaços livres e desconhecidos do ambiente.

Diferente dos mapas topológicos, que não armazenam informações sobre regiões livres ou desconhecidas, mapas métricos discretizados, como uma **grade de ocupação** 2D, permitem representar este tipo de informação. Cada célula que compõe a grade tem uma probabilidade de estar ocupada. Grades de ocupação são muito utilizadas com robôs equipados com sensores de alcance, do tipo sonar ou laser. Assumindo um mapa $\boldsymbol{V}$, representado por uma grade 2D, e um sensor laser, é necessário associar cada leitura $\boldsymbol{z}_t$ do laser com as células $\boldsymbol{v}_i \in \boldsymbol{V}$ por ela afetadas (i.e. células onde o raio do laser incide). Deve-se associar a pose do robô $\boldsymbol{x}_t$ com a célula $\boldsymbol{v}_i$ correspondente, diminuindo a probabilidade desta célula estar ocupada. Então, as células correspondentes a cada obstáculo observado pela leitura $\boldsymbol{z}_t$ devem ter suas probabilidade de ocupação aumentada. Por fim, linhas devem ser traçadas partindo de $\boldsymbol{x}_t$ até a posição de cada obstáculo e as células no caminho devem ter sua probabilidades de ocupação reduzidas. As células $\boldsymbol{v}_i \in \boldsymbol{V}$ transpassadas por essa linha podem ser obtidas utilizando o Algoritmo 2, conhecido como **DDA (*Digital Differential Analyzer*)** (WATT, 1999). Apesar de representarem diretamente as regiões livres e ocupadas, os mapas em grade de ocupação, especialmente 3D, são pouco eficientes para representar ambientes grandes e complexos, uma vez que sua resolução se mantém constante. A Figura 2.2 ilustra o processo de mapeamento de uma leitura de um sensor laser sobre uma grade de ocupação.

Figura 2.2: Exemplo de geração de uma grade de ocupação. Em (a) a pose $\boldsymbol{x}_t$ e a leitura $\boldsymbol{z}_t$ sobrepostas a grade, com a linha de visão em vermelho. Em (b) a grade alterada, quanto mais verde menor a probabilidade ocupação, quando mais azul, maior.



(a) (b)

Algoritmo 2: Algoritmo DDA para linhas 2D

Entrada: $\boldsymbol{v}_1$, $\boldsymbol{v}_2$; // Células de origem e destino
1 $\delta x = x(\boldsymbol{v}_2) - x(\boldsymbol{v}_1)$;
2 $\delta y = y(\boldsymbol{v}_2) - y(\boldsymbol{v}_1)$;
3 **se** $\delta x > \delta y$; // Define os passos pela maior variação
então
4 | $passos = |\delta x|$;
5 **senão**
6 | $passos = |\delta y|$;
fim
7 $\Delta x = \delta x / passos$; // Calcula incremento em x
8 $\Delta y = \delta y / passos$; // Calcula incremento em y
9 $\boldsymbol{v}_a = \boldsymbol{v}_1$; // Variável auxiliar
10 **para** *cada passo de passos* **faça**
11 | $x(\boldsymbol{v}_a) = x(\boldsymbol{v}_a) + \Delta x$; // Incrementa coordenada x
12 | $y(\boldsymbol{v}_a) = y(\boldsymbol{v}_a) + \Delta y$; // Incrementa coordenada y
13 | **Ação** em $\boldsymbol{v}_a$; // Ação desejada
fim

2.3 Localização

A **localização** de robôs móveis é, segundo Thrun, Burgard e Fox (2005), o problema que consiste em estimar, através da leitura $\boldsymbol{z}_t$, a pose $\boldsymbol{x}_t$ de um robô em relação a um mapa $\boldsymbol{V}$ do ambiente previamente conhecido. Este problema pode ser considerado, de certa forma, o oposto do problema de mapeamento, já que na localização o mapa do ambiente é previamente conhecido, mas deseja-se estimar a pose do robô. Existem variações deste problema que estão associadas a diferentes fatores.

O primeiro fator é o ambiente em que o robô vai operar, podendo este ser um ambiente estático ou dinâmico. Quando está inserido em um ambiente estático o robô é o único elemento em movimento. Este problema é mais simples, podendo ser dificultado caso o mapa esteja desatualizado. Por outro lado, em ambientes dinâmicos outros elementos também estão em movimento. As mudanças causadas pelo movimento dos outros elementos torna o problema mais complexos do que em ambiente estáticos. Ainda, estas mudanças podem ser permanentes, gerando uma discrepância entre o mapa e o ambiente real, dificultando o processo de localização.

O segundo fator é o conhecimento que o robô possui sobre sua pose inicial, ge-

rando duas variações do problema de localização com distintos níveis de dificuldade. Caso o robô conheça sua pose inicial, o problema a ser solucionado é o de **localização local**, que é o mais simples, pois necessita apenas que o os movimentos do robô sejam rastreados de acordo com suas leituras. Uma versão mais complexa do problema, conhecido como **localização global**, é quando o robô não conhece sua pose inicial. Neste, o robô é inserido em um ambiente sem qualquer informação prévia sobre sua pose e deve, analisando seu entorno, estimar a sua localização. Existe ainda uma terceira variação, conhecida como **problema do robô raptado**, que não está relacionada diretamente com o conhecimento do robô sobre sua pose inicial. Nesta variação, enquanto o robô se move pelo ambiente, influências externas podem deslocá-lo para outro lugar sem que ele perceba. O robô deve identificar que sua estimativa, que até então era gerada por uma localização local, está errada e iniciar um processo de localização global.

2.4 SLAM

Na maioria das aplicações reais em robótica móvel, o robô não conhece a sua pose e não possui um mapa do ambiente *a priori*. Para determinar sua pose, ele precisa realizar um processo de localização, que depende de um mapa preciso. Por outro lado, para obter um mapa do ambiente, ele precisa realizar um processo de mapeamento, que depende de uma localização precisa. Isso gera um problema do tipo "ovo e galinha", em que a localização depende do mapeamento e o mapeamento depende da localização. Este problema é conhecido como **SLAM (*Simultaneous Localization and Mapping*)** e para solucioná-lo é necessário estimar a pose do robô ao mesmo tempo em que o mapa do ambiente é construído.

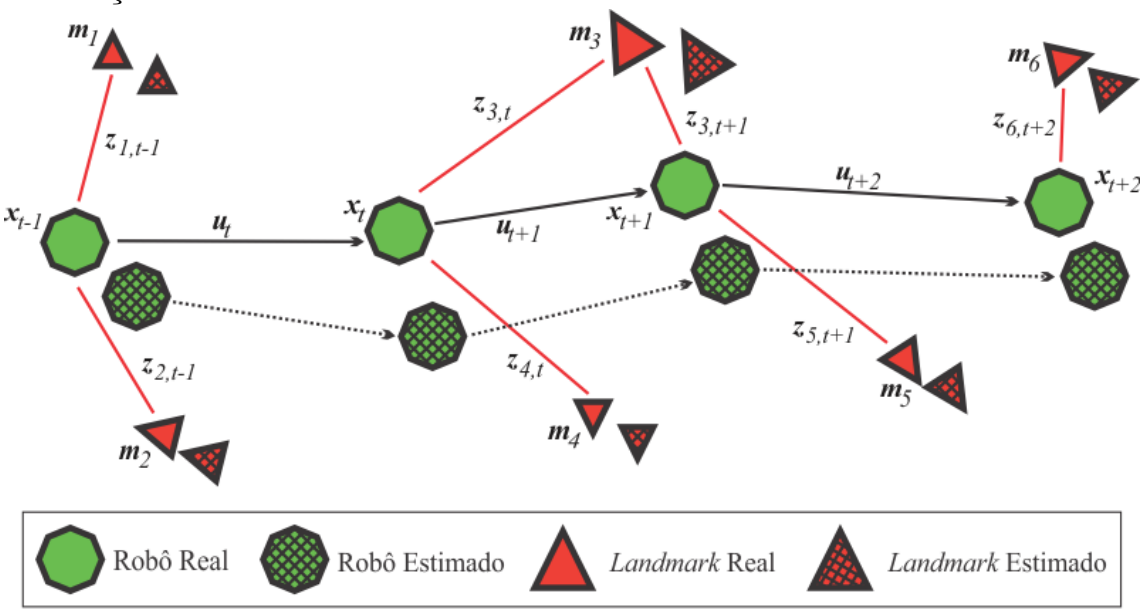
É necessário introduzir alguns conceitos para formalizar o problema de SLAM:

- $\boldsymbol{u}_t$ é a informação de controle de movimento que corresponde a mudança da pose do robô no instante $t-1$ para o instante t, normalmente dada por medidas de odometria. A sequência destas informações é representada por $\boldsymbol{u}_{1:t} = \{\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3, ..., \boldsymbol{u}_t\}$.
- $\boldsymbol{m}_l$ é a posição do *l-ézimo landmark*. O conjunto de *landmarks* é definido por $\boldsymbol{m} = \{\boldsymbol{m}_1, \boldsymbol{m}_2, ..., \boldsymbol{m}_j\}$, sendo j o número destes.
- $\boldsymbol{z}_{l,t}$ é a observação do *landmark* $\boldsymbol{m}_l$ feita pelo robô no instante de tempo t. O

conjunto de observações feitas pelo robô no instante de tempo t é representado por $\boldsymbol{z}_t = \{\boldsymbol{z}_{1,t}, \boldsymbol{z}_{2,t}, ..., \boldsymbol{z}_{j,t}\}$. As leituras podem ser agrupadas gerando um conjunto representado por $\boldsymbol{z}_{0:t} = \{\boldsymbol{z}_0, \boldsymbol{z}_1, \boldsymbol{z}_2, ..., \boldsymbol{z}_t\}$.

A Figura 2.3 apresenta um exemplo do problema. O robô no tempo $t-1$ observa os *landmarks* $\boldsymbol{m}_1$ e $\boldsymbol{m}_2$. Então, as posições de $\boldsymbol{m}_1$ e $\boldsymbol{m}_2$ são estimadas com base nas leituras $\boldsymbol{z}_{1,t-1}$ e $\boldsymbol{z}_{2,t-1}$. Em seguida, a ação $\boldsymbol{u}_t$ move o robô da pose $\boldsymbol{x}_{t-1}$ para $\boldsymbol{x}_t$. O processo é repetido nos passos seguintes. Os *landmarks* $\boldsymbol{m}_3$ e $\boldsymbol{m}_4$ são observados no instante t. No instante $t+1$, o robô observa $\boldsymbol{m}_5$ e, novamente, $\boldsymbol{m}_3$. Por fim, $\boldsymbol{m}_6$ é observado no instante $t+2$.

Figura 2.3: Exemplo do problema de SLAM. A pose $\boldsymbol{x}$ real do robô e a posição dos *landmarks* $\boldsymbol{m}$ do ambiente são desconhecidas. Ambas são estimadas através das informações de controle $\boldsymbol{u}$ e das leituras $\boldsymbol{z}$.

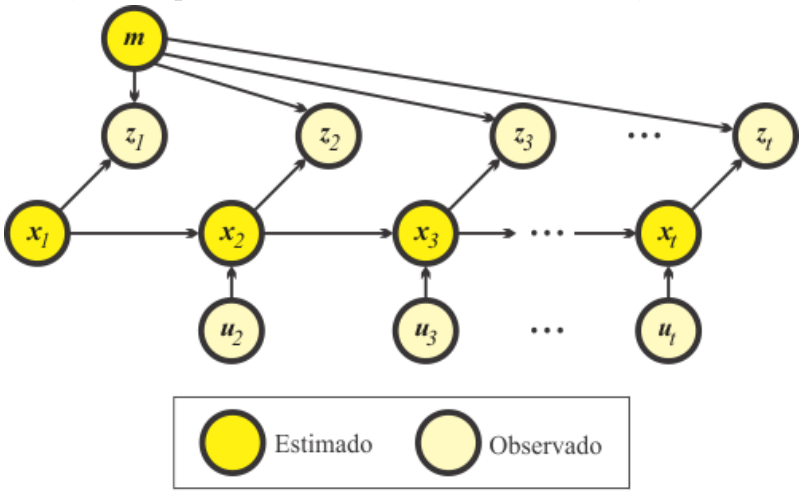


Pode-se então, formular o problema de SLAM como uma Rede Bayesiana Dinâmica, apresentada na Figura 2.4. As variáveis observadas são as informações de ações $\boldsymbol{u}_{1:t}$ e as leituras $\boldsymbol{z}_{1:t}$ obtidas pelo robô, enquanto as variáveis que devem ser estimadas são as poses $\boldsymbol{x}_{1:t}$ e a posição do conjunto de *landmarks* $\boldsymbol{m}$.

Um fator importante a ser observado em métodos de SLAM é o **fechamento de *loops***[4]. Quando um robô retorna a uma região já visitada, é possível corrigir erros acumulados ao longo do trajeto. Porém, o SLAM precisa perceber que um *loop* foi fechado, pois houve uma revisita. Um SLAM que não feche *loops* não tira proveito da informação de revisita para corrigir o mapa. Além disso, está assumindo que o

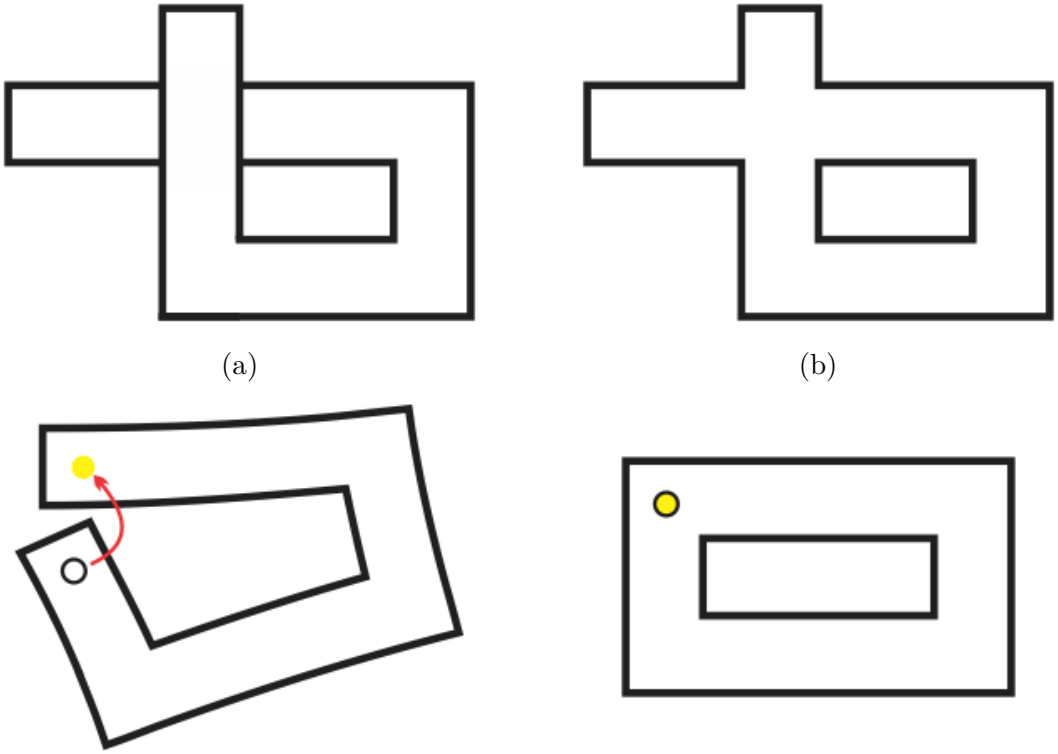
[4]Um *loop* é um laço formado quando o robô sai de um ponto do ambiente, se desloca e então retorna ao ponto de partida.

Figura 2.4: Formulação do SLAM como uma Rede Bayesiana Dinâmica. As variáveis observadas, $\boldsymbol{z}$ e $\boldsymbol{u}$, são dependentes das variáveis estimadas, $\boldsymbol{x}$ e $\boldsymbol{m}$.



ambiente é um "corredor infinito", não percebendo as interconexões entre as regiões. A Figura 2.5 exemplifica a importância do fechamento de *loop*.

Figura 2.5: Exemplo da importância do fechamento de *loop* para evitar o problema do "corredor infinito" e corrigir o mapa. Em (a) o mapa com um "corredor infinito" e em (b) o mapa correto após o fechamento de um *loop*. Em (c) o mapa com o erro dos sensores acumulado ao longo do trajeto e em (d) o mapa corrigido.



(a) (b)

(c) (d)

2.4.1 SLAM Visual e SLAM Monocular

Um **SLAM Visual** utiliza apenas sensores passivos[5], como uma câmera, para capturar as informações do ambiente. A utilização de uma câmera como sensor possui vantagens como o baixo custo, a riqueza de informações e a já citada passividade. Se a câmera utilizada for simples, ou seja, que não capture estimativas de profundidade, apenas uma imagem, então o SLAM Visual pode ser chamado de **SLAM Monocular**.

É possível dividir os SLAMs Monoculares em duas categorias: os métodos baseados em *features*[6] e os métodos diretos. Os **métodos baseados em *features*** primeiro extraem das imagens um conjunto de primitivas geométricas (e.g. cantos, retas, pontos, etc) e depois as associam de forma robusta em imagens consecutivas. As *features* extraídas são os dados de entrada para o processo de SLAM. Já os **métodos diretos** utilizam diretamente a intensidade dos *pixels* da imagem como entrada para o SLAM, e possuem a vantagem de utilizar toda, ou quase toda, informação da imagem, mesmo onde não existem primitivas geométricas.

Umas das primeiras abordagens bem sucedidas de SLAM Monocular foi o **PTAM (*Parallel Tracking and Mapping*)**, proposto por Klein e Murray (2007), que utiliza *features* em um processo que trata localização e mapeamento paralelamente, gerando um mapa esparso. Mais recentemente, foi proposto por Engel, Schöps e Cremers (2014) o **LSD-SLAM (*Large-Scale Direct Monocular SLAM*)**, uma técnica direta que obtém bons resultados, gerando um mapa semidenso. Essa técnica, porém, necessita que a câmera mova-se lateralmente, limitando as ações que o robô pode realizar, impedindo que ele ande olhando para frente, por exemplo. Já a abordagem proposta por Mur-Artal, Montiel e Tardos (2015), conhecida como **ORB-SLAM**, não possui as mesmas restrições de movimento apresentadas no LSD-SLAM. Essa abordagem utiliza *features* ORB (*Oriented FAST and Rotated BRIEF*) e também obtém bons resultados. Sua desvantagem é gerar um mapa esparso que contém menos informações do ambiente. Sua vantagem, além de não possuir tantas restrições de movimento, é utilizar as mesmas *features* em todas tarefas do SLAM, reduzindo assim o custo computacional, permitindo que ele seja executado em *hardwares* com menos capacidade.

[5]Sensores passivos não emitem energia, apenas recebem energia do ambiente.
[6]Uma *feature* é uma características geométrica do ambiente, como um ponto, uma reta, etc.

2.4.2 ORB-SLAM

Esta subseção apresenta o **ORB-SLAM**, um SLAM Monocular proposto por Mur-Artal, Montiel e Tardos (2015), que pertence a categoria de métodos baseados em *features*, extraindo *features* ORB (RUBLEE et al., 2011) das imagens para criar um mapa. Este SLAM é considerado estado da arte em SLAM Monocular, tendo como principais características a capacidade de operar *online* sem a necessidade de uma GPU (*Graphics Processing Unit*) e a utilização das mesmas *features* em todos subprocessos do SLAM.

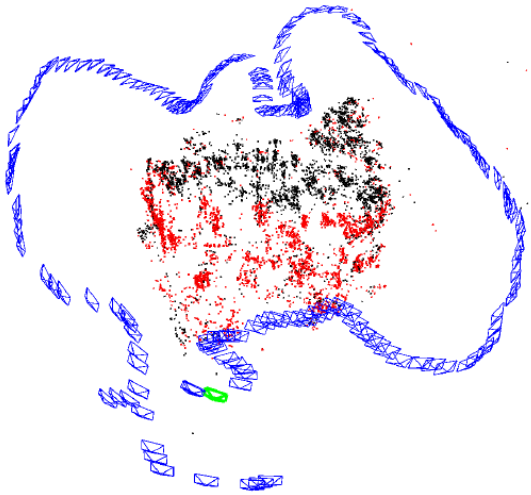
O ORB-SLAM cria um mapa topológico representado através de uma nuvem de pontos. A estrutura deste mapa é formada por um conjunto de *keyframes* e um conjunto de pontos. Os *keyframes* são representados por $\boldsymbol{K} = \{\boldsymbol{k}_1, \boldsymbol{k}_2, ..., \boldsymbol{k}_r\}$, sendo r o número de *keyframes*. Cada *keyframe* $\boldsymbol{k}_a$ armazena todas as *features* extraídas do *frame*[7] que o gerou e a pose da câmera $\boldsymbol{T}_a$. Os pontos, também chamados de pontos do mapa, são representados por $\boldsymbol{P} = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, ..., \boldsymbol{p}_s\}$, sendo s o número de pontos do mapa. Cada ponto $\boldsymbol{p}_b$ armazena sua posição no ambiente 3D através das coordenadas cartesianas $(x, y, z)^T$ e o descritor da *feature* ORB que o gerou. Os pontos do mapa estão associados com os *keyframes* que conseguem os observar, sendo $\hat{\boldsymbol{P}}_a \subseteq \boldsymbol{P}$ os pontos de $\boldsymbol{P}$ visíveis para o *keyframe* $\boldsymbol{k}_a$.

Adicionalmente ao mapa, o ORB-SLAM constrói um **grafo de covisibilidade**. Este é um grafo não direcionado ponderado onde cada *keyframe* é um nodo e uma aresta entre dois *keyframes* existe se estes observam pontos em comum, sendo θ o peso da aresta representando a quantidade de pontos em comum. Através deste grafo de covisibilidade as tarefas de localização e mapeamento são restritas apenas a uma área local de covisibilidade, independente do tamanho do mapa. Devido a este grafo poder conter muitas arestas, a otimização do mapa após o fechamento de um *loop* é realizada sobre outro grafo, denominado **grafo essencial**, que mantém todos os *keyframes*, porém diminui o número de arestas mantendo apenas as arestas com maiores θ, algumas arestas de fechamento de *loop*[8] e uma **árvore geradora** (*spanning tree*) que representa um subgrafo com o número mínimo de arestas. A estrutura do mapa e estes grafos podem ser observados na Figura 2.6.

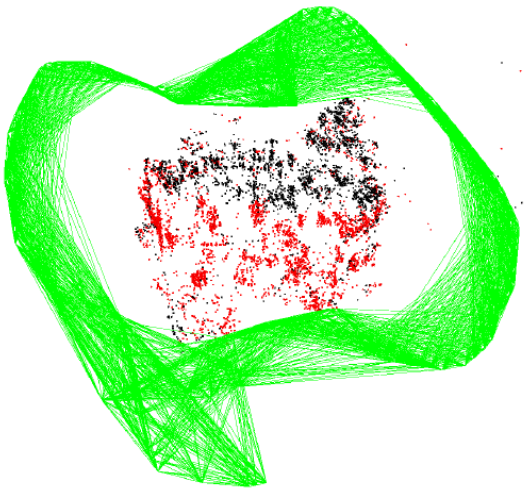
[7] Um *frame* é um quadro de um video.

[8] Uma aresta de fechamento de *loop* conecta dois nodos do grafo que foram detectados como iguais, ou seja, representam a mesma região do ambiente.

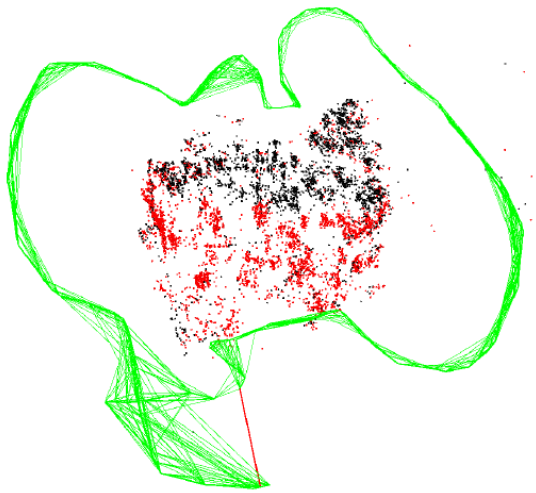
Figura 2.6: Mapa em (a), grafo de covisibilidade em (b), grafo essencial em (c) e árvore geradora em (d). Adaptado de (MUR-ARTAL; MONTIEL; TARDOS, 2015).



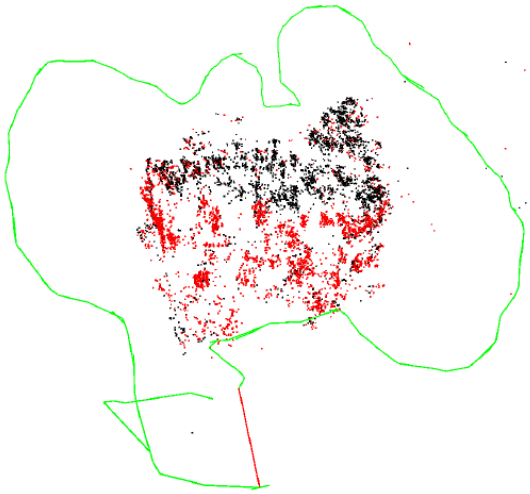
(a) *Keyframes* em azul, pose da câmera atual em verde, pontos da nuvem em vermelho e preto, pontos do mapa local em vermelho.



(b) Grafo de covisibilidade com arestas em verde.



(c) Grafo essencial com arestas de covisibilidade em verde e aresta de *loop* em vermelho.



(d) árvore geradora com arestas de covisibilidade em verde e aresta de *loop* em vermelho.

2.4.2.1 Estrutura geral

A estrutura geral do ORB-SLAM é composta por três *threads*: Uma para a localização, uma para o mapeamento e uma para o fechamento de *loops*. Estas três tarefas ocorrem em paralelo. A *thread* de localização é a responsável por localizar a câmera (i.e. robô) a cada novo *frame* capturado e decidir quando um novo *keyframe* deve ser inserido em $\boldsymbol{K}$. Inicialmente, uma estimativa da pose é obtida através de um modelo de movimento de velocidade constante. Então, as *features* ORB do novo *frame* são extraídas e casadas com as dos pontos do mapa presentes no *frame*

anterior. Através dos casamentos é realizada uma otimização da pose utilizando *Bundle Adjustment* (BA) (TRIGGS et al., 1999). Se a localização é perdida, por causa de uma oclusão ou movimento brusco, então um processo de relocalização global é realizado utilizando *Bag of Words* (GÁLVEZ-LÓPEZ; TARDOS, 2012) para comparar o *frame* atual com todos os *keyframes* de $\boldsymbol{K}$.

Uma vez que uma estimativa inicial da pose da câmera e do casamento das *features* é obtida, a *thread* de localização obtém um mapa de visibilidade local utilizando o grafo de covisibilidade. O *keyframe* do mapa local que possuir maior similaridade com o *frame* é o *keyframe* referência, representado por $\boldsymbol{k}_{ref}$. Então os pontos do mapa local são reprojetados sobre o *frame* e a pose é otimizada usando os casamentos obtidos entre os pontos reprojetados e as *features*. Por fim, a *thread* de localização decide se o *frame* atual será inserido como um novo *keyframe* em $\boldsymbol{K}$. Para que o *frame* se torne um *keyframe* as seguintes condições devem ser atendidas: um mínimo de 20 *frames* processados desde a última inserção; uma quantidade miníma de 50 pontos rastreados no *frame*, garantindo assim que ele seja rastreável; um máximo de 90% dos pontos rastreados no *frame* em comum com os pontos $\hat{\boldsymbol{P}}_{ref}$ de $\boldsymbol{k}_{ref}$, garantindo que houve uma mudança visual.

A *thread* de mapeamento é responsável por processar cada novo *keyframe* $\boldsymbol{k}_a$ inserido pela *thread* de localização e realizar ajustes locais para atingir uma reconstrução ótima da região próxima a pose da câmera $\boldsymbol{T}_a$. Inicialmente, o grafo de covisibilidade é atualizado adicionando um novo nodo para $\boldsymbol{k}_a$, assim como as arestas entre ele e seus vizinhos, representados por $\mathcal{K}_a$. Então, a árvore geradora é atualizada, criando uma aresta entre $\boldsymbol{k}_a$ e o *keyframe* de $\mathcal{K}_a$ com mais pontos em comum (i.e. maior θ). Uma representação de $\boldsymbol{k}_a$ através de *Bag of Words* é extraída para auxiliar na associação de dados durante a triangularização de novos pontos. Antes de criar novos pontos, uma política para remover pontos de baixa qualidade é aplicada, mantendo apenas os pontos de alta qualidade. Para ser mantido no mapa, o ponto deve cumprir duas condições: ser rastreado em pelo menos 25% dos *frames* em que ele deveria estar visível (i.e. obtidos logo após sua criação); ser rastreado em pelo menos três *keyframes*, se mais de um *keyframe* foi inserido após sua criação. Este processo garante que os pontos que permanecerem no mapa são rastreáveis e foram corretamente triangularizados.

Então, a *thread* de mapeamento cria novos pontos do mapa utilizando *features* ORB não casadas (i.e. que não correspondem a um ponto $\boldsymbol{p}_b \in \boldsymbol{P}$) de $\boldsymbol{k}_a$ e

as *features* não casadas de todos seus vizinhos do grafo de covisibilidade, representados por $\mathcal{K}_a$. Para cada casamento obtido, um novo ponto do mapa é triangularizado e sua integridade é checada através da profundidade, do erro de reprojeção e da consistência na escala. Se este ponto for considerado íntegro, ele é inserido em $\boldsymbol{P}$. Um novo ponto do mapa é obtido utilizando apenas $\boldsymbol{k}_a$ e um *keyframe* de $\mathcal{K}_a$, porém, após ser inserido, ele é reprojetado sobre o resto dos *keyframes* vizinhos. Um processo de otimização é realizado sobre $\boldsymbol{k}_a$, $\mathcal{K}_a$ e todos os pontos vistos por estes. É responsabilidade da *thread* de mapeamento, também, remover os *keyframes* redundantes afim de manter uma representação compacta do ambiente. Um *keyframe* $\boldsymbol{k}_a$ é descartado se $\mathcal{K}_a$ possuir pelo menos três *keyframes* que compartilham 90% ou mais dos pontos $\hat{\boldsymbol{P}}_a$ de $\boldsymbol{k}_a$.

Por fim, a *thread* de fechamento de *loops* é responsável por determinar se existem *loops* a cada novo *keyframe* inserido no mapa. A busca de *loops* é realizada sobre o último *keyframe* $\boldsymbol{k}_a$ processado pela *thread* de mapeamento. Inicialmente um conjunto de candidatos é obtido. Para tal, a menor similaridade entre $\boldsymbol{k}_a$ e seus vizinhos em $\mathcal{K}_a$ é calculada utilizando *Bag of Words*. Então, todos *keyframes* $\boldsymbol{k} \in \boldsymbol{K} \wedge \boldsymbol{k} \notin \mathcal{K}_a$ que possuírem uma similaridade maior que a miníma são considerados candidatos. Para efetivamente aceitar um *keyframe* como candidato a *loop*, a *thread* de fechamento de *loops* deve encontrar três candidatos a *loop* consecutivos que estão conectados entre si através do grafo de covisibilidade.

Após determinar os candidatos a *loop*, é necessário estimar a transformação de similaridade entre $\boldsymbol{k}_a$ e cada *keyframe* candidato, representado por $\boldsymbol{k}_c$, para determinar o erro acumulado entre estes. Inicialmente, é encontrada a correspondência entre os pontos do mapa $\hat{\boldsymbol{P}}_a$ e $\hat{\boldsymbol{P}}_c$. Com esta correspondência, é realizado um processo de RANSAC para estimar a transformação de similaridade. Se uma transformação for suportada por pontos suficientes do mapa, então um *loop* é efetivamente fechado. Conhecendo o erro acumulado entre $\boldsymbol{k}_a$ e $\boldsymbol{k}_c$ os dois *keyframes* são alinhados, os pontos do mapa duplicados são fundidos e novas arestas são inseridas no grafo de covisibilidade. Por fim, uma otimização é realizada sobre o grafo essencial para manter o mapa globalmente consistente, distribuindo o erro acumulado entre $\boldsymbol{k}_a$ e $\boldsymbol{k}_c$ ao longo do grafo.

2.4.2.2 Inicialização do mapa

Os processos realizados pelo ORB-SLAM descritos até aqui dependem de uma estimativa inicial do mapa para serem executados. A localização, por exemplo, utiliza os pontos do mapa presentes no último *frame* processado para estimar a pose da câmera, que é então otimizada utilizando o grafo de covisibilidade. É necessário estimar um mapa inicial para poder realizar estas tarefas, assim como todas as outras que dele dependam. A solução não deve depender das características do ambiente, assim como não deve necessitar de intervenção humana para selecionar dois bons pontos de vista (i.e. com suficiente paralaxe entre si).

Para inicializar o mapa, o ORB-SLAM estima, em paralelo, dois modelos geométricos diferentes: uma homografia assumindo uma cena planar e uma matriz fundamental assumindo uma cena não planar. Para estimar ambos modelos, inicialmente são extraídas as *features* ORB do *frame* atual e anterior. Se existirem casamentos suficientes entre estas, a homografia e a matriz fundamental são estimadas utilizando RANSAC com quatro e oito pontos, respectivamente. A melhor homografia e a melhor matriz fundamental são obtidas através de uma pontuação baseada no erro de reprojeção. Caso as melhores não possuam pontos suficientes que as sustentem, o processo é reiniciado utilizando um novo *frame*. Caso elas possuam, uma análise é realizada escolhendo a homografia quando a cena for planar ou houver pouca paralaxe, e escolhendo a matriz fundamental quando a cena não for planar. Para definir qual das duas escolher, é analisada a mesma pontuação utilizada durante o processo de RANSAC. Após escolher qual das soluções utilizar, hipóteses do movimento realizado entre as imagens são obtidas e analisadas para evitar que uma inicialização ocorra com uma paralaxe muito pequena. Se a transformação for considerada suficientemente boa, o mapa inicial é otimizado utilizando BA.

Esse processo de inicialização utiliza RANSAC, o que gera um fator aleatório. Por isso, diferentes execuções sobre uma mesma sequência de imagens geram mapas diferentes, variando em escala e qualidade. Isso dificulta a repetição de testes, pois diferentes cenários serão gerados a cada nova execução. Também, não permite definir uma conversão entre a escala do mapa e uma escala métrica compatível com o ambiente, pois, a cada execução a escala do mapa varia, necessitando de um novo fator de conversão.

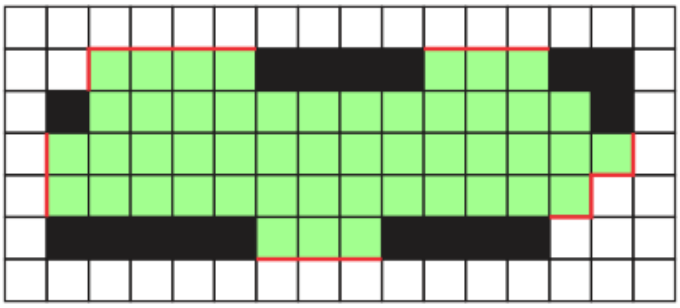
2.5 Exploração

A exploração autônoma de ambientes, ou simplesmente **exploração**, é a união dos problemas de mapeamento e de planejamento de caminhos. O mapeamento é responsável por transformar as informações coletadas pelo sensor do robô em um mapa. Para tal, é necessário que o robô se desloque pelo ambiente para coletar novas leituras, expandindo seu conhecimento. Em outras palavras, é necessário controlar os movimentos do robô para que ele maximize seu conhecimento sobre o ambiente. Surge então a principal questão que define o problema de exploração: "Para onde ir?".

Alguns requisitos devem ser observados em uma abordagem de exploração. É necessário que o robô explore totalmente o ambiente, evitando encerrar o processo precocemente. Também, espera-se que o robô não colida com obstáculos enquanto navega pelo ambiente. Atentando-se a estes dois requisitos, a abordagem deve definir um conjunto de poses $\boldsymbol{x}$ que o robô deve assumir para explorar o ambiente.

Muitas abordagens propõem utilizar fronteiras para resolver o problema de exploração. Assumindo um mapa $\boldsymbol{V}$ em forma de grade, uma fronteira é uma borda entre as células de $\boldsymbol{V}$ classificadas como livres, representadas por $\boldsymbol{V}_{livre}$, e as classificadas como desconhecidas, representadas por $\boldsymbol{V}_{desconhecido}$. A Figura 2.7 apresenta um exemplo de fronteiras em uma grade 2D. As abordagens que utilizam fronteiras identificam, através delas, as regiões a serem exploradas e determinam que a exploração foi completada, em geral, quando não existem mais fronteiras.

Figura 2.7: Exemplo de fronteiras em um mapa 2D. Em verde as células livres, em preto as ocupadas e em branco as desconhecidas. Em vermelho as fronteiras entre as células livres e desconhecidas.



Em ambientes 3D, no entanto, métodos de exploração baseado em fronteiras obtém resultados fracos devido ao fato dos sensores adquirirem informações incompletas sobre o seu entorno, dificultando a correta classificação de espaços como des-

conhecidos ou livres (SHEN, 2014). Além disso, armazenar mapas volumétricos que contêm a representação de espaço livre e ocupado em grandes ambientes 3D pode tornar-se computacionalmente intratável devido à grande necessidade de memória.

Abordagens baseadas em fronteiras podem ser consideras um subconjunto de uma gama de solução baseadas no ganho de informação. Essas abordagens levam em consideração que na maioria dos métodos de mapeamento o robô mantém uma crença sobre o mapa, e no caso do SLAM, é mantida também uma crença sobre a pose do robô. Então, é possível selecionar regiões para serem exploradas que minimizem a incerteza sobre o mapa, quando apenas mapeamento, e também a pose do robô, quando em um processo de SLAM.

Assim, o problema pode ser formulado como encontrar, a cada instante de tempo t, a próxima pose $\boldsymbol{x}_{t+1}$ a ser assumida. Durante o deslocamento entre a pose $\boldsymbol{x}_t$ e a pose $\boldsymbol{x}_{t+1}$ o robô obtém uma sequência de observações $\boldsymbol{z}_{t:t+1}$. Abordagens como as propostas por Song e Jo (2017), Adler, Xiao e Zhang (2014), Vallvé e Andrade-Cetto (2014) e Quin et al. (2013), buscam encontrar a pose $\boldsymbol{x}_{t+1}$ que gera as leituras $\boldsymbol{z}_{t:t+1}$ que maximizem o ganho de informação. Porém, determinar o ganho de informação pode ser intratável devido a complexidade do ambiente. Geralmente, são utilizadas aproximações, e a maneira de obter a aproximação depende de diversos fatores como o modelo do ambiente e o tipo de dado obtido pelo sensor.

Uma extensão das abordagens que utilizam o ganho de informação são as abordagens que utilizam funções de utilidade, com as propostas por Ström, Bogoslavskyi e Stachniss (2017), Meng et al. (2017), Chudoba et al. (2016), Tabib et al. (2016) e Heng et al. (2015). Essas abordagens criam funções para relacionar o ganho de informação com o custo para obtê-la. Uma escolha comum para estimar o custo é utilizar a distância que o robô deve percorrer. Então, o problema passa a ser encontrar a pose que maximize o ganhe do informação minimizando o custo para obtê-la.

2.6 Trabalhos relacionados

O escopo desta dissertação é o problema de exploração de ambientes 3D utilizando apenas uma câmera como fonte de informação. Busca-se propor uma abordagem que permita utilizar as técnicas estado da arte de SLAM monocular, acoplando a estas um algoritmo de exploração. Estes métodos têm por caracterís-

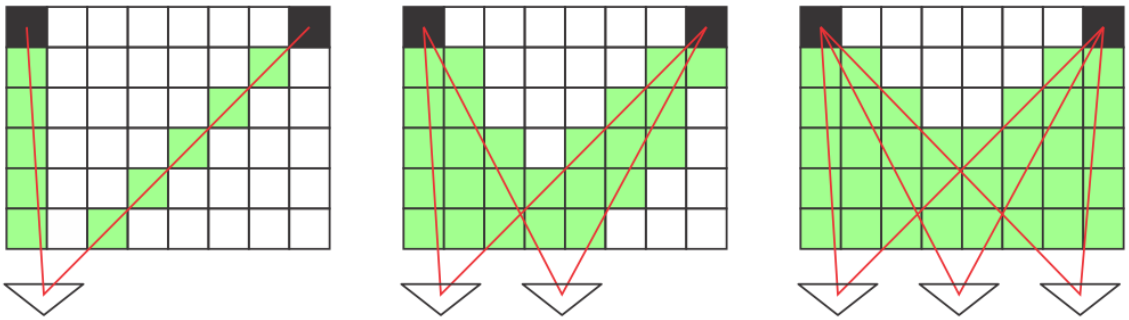
tica a necessidade de saliências visuais no ambiente para que ele seja corretamente mapeado. Regiões do ambiente com baixa saliência visual podem não ser mapeadas, impedindo que abordagens que utilizem fronteiras sejam utilizadas, uma vez que uma região de baixa saliência seria considerada desconhecida e poderia formar uma fronteira. Deslocar-se para esta fronteira não faria com que a região fosse mapeada, apenas causaria uma colisão do robô com o obstáculo que não foi detectado.

O trabalho de Stumberg et al. (2017) aproxima-se do proposto nesta dissertação, pois também realiza um processo de exploração utilizando um SLAM monocular, o LSD-SLAM (ENGEL; SCHÖPS; CREMERS, 2014). O mapa gerado por esse SLAM é semi-denso e representado por uma nuvem de pontos, com estrutura semelhante ao ORB-SLAM descrito na Subseção 2.4.2. Para explorar, o trabalho de Stumberg et al. (2017) propõe gerar um mapa de ocupação volumétrico representado por uma *octree*. Cada *keyframe* $\boldsymbol{k}_a \in \boldsymbol{K}$ é processado para que os *voxels* ocupados sejam obtidos através dos pontos $\hat{\boldsymbol{P}}_a$. Os *voxels* livres são obtidos traçando linhas de visão que partem de $\boldsymbol{k}_a$ e chegam em cada $\boldsymbol{p}_b \in \hat{\boldsymbol{P}}_a$. Cada *voxel*, ao final do processo, possui uma probabilidade de estar ocupado e seu estado é definido utilizando limiares sobre esta. Este mapa volumétrico deve ser atualizado para incluir as novas informações obtidas ou corrigir as informações quando o SLAM fecha um *loop*. Porém, por se tratar de um mapa global, este processo pode demorar e durante este tempo o robô fica parado esperando.

Como a reconstrução do ambiente é semi-densa, geralmente não é possível conhecer completamente os obstáculos do ambiente. Também não é possível assumir que as regiões do ambiente sem saliências são planares. É possível, no entanto, aumentar o conhecimento sobre a região livre através de movimentos laterais da câmera, como ilustrado na Figura 2.8. Com base no ganho obtido por estes movimentos, a proposta de Stumberg et al. (2017) é gerar um padrão de movimentos laterais, formando uma estrela, que passa sempre pela mesma origem. Através desse padrão, o robô aumenta seu conhecimento local sobre o ambiente. Além disso, essa abordagem gera movimentos de paralaxe que são ideais para o LSD-SLAM. Durante o deslocamento, a câmera é movida lateralmente, formando um ângulo de 90º com a direção do movimento do robô.

Após a execução da exploração local (i.e. o padrão em estrela), um processo de exploração global é realizado, determinando qual a próxima região onde deve ocorrer uma exploração local. Para determinar esta nova região, os *voxels* que esti-

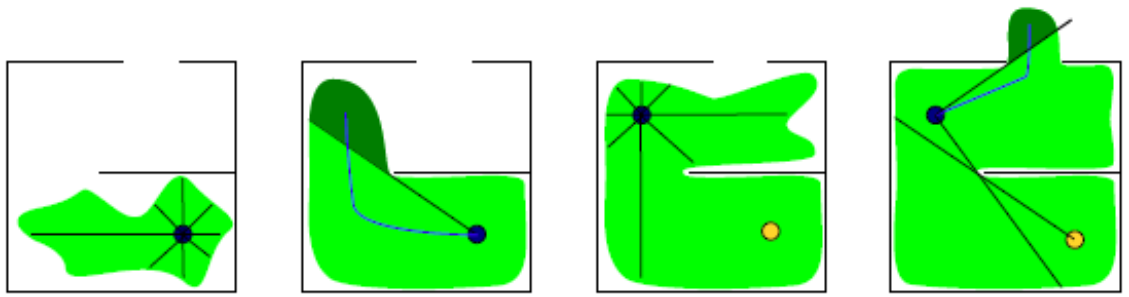
Figura 2.8: Exemplo do ganho de informação sobre o espaço livre através de movimentos laterais. Os dois obstáculos, em preto, são observados em (a) de uma pose, gerando as primeiras regiões livres, em verde. Através de deslocamentos laterais mais regiões livres são observadas. Em (b) e (c) as áreas livres após movimentações.



(a) (b) (c)

verem livres e possuírem uma linha de visão direta para qualquer origem das estrelas executadas anteriormente são marcados como visitados. Então, aqueles *voxels* livres que não foram marcados com visitados são marcados como interessantes. Os *voxels* marcados como interessantes são agrupados por proximidade formando blocos. Por fim, um *voxel* do maior bloco é escolhido como origem para a próxima estrela. Um exemplo do processo completo pode ser visto na Figura 2.9.

Figura 2.9: Exemplo do método de exploração proposto por Stumberg et al. (2017). Em verde claro a região livre, em verde escuro a região de interesse, em azul o robô e em amarelo a origem anterior da estrela. Adaptado de (STUMBERG et al., 2017)



(a) Estrela inicial.

(b) Região livre e região de interesse após estrela inicial.

(c) Nova estrela centrada na região de interesse.

(d) Região livre e nova região de interesse.

Os testes realizados por Stumberg et al. (2017) utilizam um VANT como robô. O primeiro teste demonstra o ganho de informação obtido ao executar a exploração local através do padrão de estrela. O segundo demonstra que o robô, após executar a exploração local, identifica corretamente uma nova região a ser visitada. Porém, este teste foi finalizado antes que o VANT executasse a segunda exploração local devido ao término da bateria. Apesar de mostrar-se promissora, esta abordagem

está limitada pela necessidade de um mapa volumétrico global. Gerar este mapa é um processo computacionalmente custoso e torna-se impraticável para ambientes grandes e complexos. Além disso, a estratégia de exploração local utilizando a estrela pode levar o robô a movimentar-se de forma pouco eficiente. Ao entrar em um corredor, por exemplo, o robô pode acabar navegando por toda extensão do corredor, indo e voltando.

O trabalho proposto por Mostegel (2013) também utiliza um SLAM monocular, o PTAM (KLEIN; MURRAY, 2007). Esta abordagem visa levar um VANT até uma posição objetivo, navegando por um ambiente desconhecido (i.e. sem um mapa *a priori*). O principal foco da abordagem é evitar que o robô perca a sua localização. Para tal, é realizada uma avaliação da qualidade na localização fornecida por cada pose, evitando poses que não favoreçam o processo localização.

Cada ponto $\boldsymbol{p}_b \in \boldsymbol{P}$ da nuvem de pontos (i.e. mapa) gerada pelo PTAM é avaliado para determinar uma pontuação referente a sua qualidade geométrica. Essa pontuação é baseada em dois critérios. O primeiro analisa indiretamente a incerteza sobre a posição do ponto. Como a profundidade dos pontos é extraída com base em uma triangularização, aqueles que possuírem um ângulo[9] de triangularização muito pequeno são penalizados. O segundo critério analisa a chance de um ponto $\boldsymbol{p}_b$ ser um *outlier*, observando quantos *keyframes* de $\boldsymbol{K}$ conseguem o observar, representados por $\hat{\boldsymbol{K}}_b \subseteq \boldsymbol{K}$. Os pontos que foram vistos por poucos *keyframes* possuem uma chance maior de serem *outliers* e por isso são penalizados. A pontuação de qualidade geométrica de um ponto $\boldsymbol{p}_b$ é obtida multiplicando as pontuações obtidas em cada critério, de forma que um ponto só obtém uma pontuação máxima se foi visto por *keyframes* suficientes e se possui um ângulo de triangularização suficientemente grande.

Além da qualidade geométrica dos pontos, é analisada também a probabilidade de um ponto $\boldsymbol{p}_b$ ser reconhecido em uma imagem obtida em uma pose $\boldsymbol{x}$ qualquer. É possível determiná-la analisando dois fatores, um relacionado ao ângulo de visão e outro a escala. O ângulo de visão é analisado, para um ponto $\boldsymbol{p}_b$, utilizando um *keyframe* $\boldsymbol{k}_a \in \hat{\boldsymbol{K}}_b$ como referência. Então, calcula-se o ângulo de triangularização de $\boldsymbol{p}_b$ utilizando $\boldsymbol{k}_a$ e a pose $\boldsymbol{x}$. Poses que geram ângulos grandes possuem baixa probabilidade, pois quanto maior o ângulo, menor a chance do ponto ser reconhecido. A escala é analisada utilizando novamente um *keyframe* $\boldsymbol{k}_a \in \hat{\boldsymbol{K}}_b$

[9]Em um triângulo formando pelo ponto e dois *keyframes* que o veem, o ângulo de triangularização é o ângulo que fica no ponto.

como referência. Calcula-se a distância do ponto $\boldsymbol{p}_b$ para $\boldsymbol{k}_a$ e para a pose $\boldsymbol{x}$, atribuindo a probabilidade de acordo com a diferença nestas distâncias. A probabilidade de reconhecimento do ponto é então obtida multiplicando a probabilidade associada ao ângulo e a probabilidade associada a escala.

Então, o *frame* é dividido em 8x8 partes e cada parte recebe uma qualidade de localização associada aos pontos do mapa que estão dentro dela. Movimentos que levem o VANT para regiões com baixa qualidade de localização são evitados. Outro ponto que Mostegel (2013) cita como importante é a análise do quanto uma região é promissora para gerar novos pontos. Para tal, as parte do *frame* que já possuem muitos pontos (i.e. pontos da nuvem nessa região da imagem) são consideradas mapeadas. Sobre as partes restantes, é feita uma análise das *features* rastreadas mas que ainda não tornaram-se pontos do mapa. Busca-se poses que maximizem o ângulo de triangularização destas *features*, respeitando a probabilidade de rastreamento dela na nova pose, como descrito anteriormente.

Os resultados obtidos por Mostegel (2013) demonstram que a abordagem é bem sucedida em levar o VANT da pose que ele se encontra até uma pose pré determinada, mapeando o ambiente durante o trajeto e mantendo baixa a incerteza sobre a localização do robô. A análise é feita sempre sobre o *frame* de entrada e pequenos passos (i.e. movimentos) são calculados a cada instante. Porém, esta não é uma abordagem de exploração real, pois não responde a principal questão, que é para onde ir globalmente. Uma pose destino tem que ser definida previamente por um operador, cabendo a solução proposta por Mostegel (2013) apenas levar o robô até esta pose.

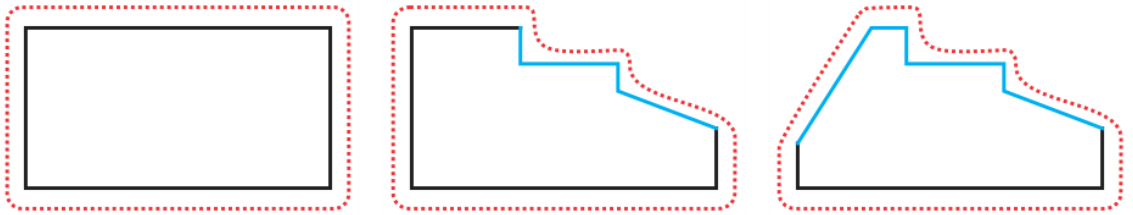
A abordagem proposta por Palazzolo e Stachniss (2017) realiza uma exploração 3D que pode ser monocular. Nela, é criado um mapa do ambiente representado por uma *octree*. Os autores assumem que existe uma *bounding box*[10] conhecida ao redor do objeto a ser mapeado. A cada iteração, é definida uma pose $\boldsymbol{x}$ que o robô deve assumir considerando o ganho de informação esperado, a distância até esta nova pose e a suavidade no caminho para alcançá-la.

Inicialmente é definida uma casca que cerca esta *bounding box* e durante a exploração é ajustada conforme informações sobre o objeto são obtidas. A Figura 2.10 apresenta esta casca no momento inicial, em (a), e durante a exploração, em (b) e (c). Então, um conjunto reduzido de poses candidatas é gerado sobre esta casca e

[10]Uma *bounding box* é uma caixa delimitadora, em uma tradução literal. É uma caixa que delimita uma região em torno de um objeto de interesse.

sempre olhando para o objeto (i.e. centro da casca). Assim, a cada iteração cem candidatos são gerados.

Figura 2.10: Exemplo da casca utilizada por Palazzolo e Stachniss (2017). Em (a) a casca, em vermelho, cercando a *bounding box*, em preto. Em (b) e (c) a casca ajustada após parte do ambiente ser mapeado, em azul.



(a) (b) (c)

Cada pose candidata é avaliada através do ganho de informação e de uma função de custo. O ganho de informação de uma pose é obtido de acordo com a sua capacidade de reduzir a incerteza sobre os pontos do mapa. Essa redução é estimada com base no ângulo da triangularização gerado pela nova pose para cada ponto. A função de custo avalia a distância a ser percorrida para que o VANT chegue até a pose e a variação na orientação necessária. Além disso, a função de custo também leva em consideração fatores de segurança que priorizam: caminhos próximos ao ponto de partida, facilitando o pouso; caminhos que não passam sobre obstáculos, facilitando pousos de emergência; caminhos próximos ao solo, evitando um grande impacto caso a bateria acabe. É escolhida como destino a pose que maximize o ganho de informação minimizando o custo.

Os resultados obtidos por Palazzolo e Stachniss (2017) mostram que a abordagem consegue mapear um ambiente com precisão e de forma rápida. Porém, essa abordagem assume que existe uma *bounding box* que cerque o objeto ou construção a ser mapeada. Mais do que isso, assume que existe uma casca cercando esta *bounding box* na qual o robô pode navegar livremente, sem colidir, e gera sobre esta casca poses candidatas sempre olhando para o centro dela. Isso impede que a abordagem seja utilizada em ambientes internos e complexos, permitindo apenas mapear o exterior de um prédio, por exemplo.

As três abordagens apresentadas possuem limitações. No trabalho de Stumberg et al. (2017), é necessário gerar um mapa volumétrico global, através de um processo que pode ser longo e faz o VANT ficar pairando no ar até que o mapa seja gerado. O trabalho de Mostegel, Wendel e Bischof (2014) propõe uma forma interessante de evitar que o VANT perca a estimativa de sua pose. Porém, esta não

é realmente uma exploração, pois é necessário pré-definir um objetivo global. Por fim, o trabalho de Palazzolo e Stachniss (2017) assume a existência de uma *bounding box* ao redor do objeto a ser mapeado e gera uma casca livre para navegar, a priori, cercando esta *bounding box*. Além disso, define poses para que o robô sempre olhe para o centro da casca, impedindo que a exploração seja executada em ambiente internos e complexos. A abordagem proposta nesta dissertação busca resolver o problema de exploração monocular 3D sem estas limitações.

3 EXPLORAÇÃO UTILIZANDO LINHAS DE VISÃO

Neste capítulo apresenta-se a estratégia de exploração de ambientes 3D utilizando SLAM monocular proposta. É importante ressaltar que tais métodos de SLAM geram mapas esparsos ou semi-densos, pois dependem de saliências visuais do ambiente. O ORB-SLAM foi o SLAM monocular escolhido para a prova de conceito. Os objetivos da abordagem proposta, como já definidos na Seção 1.2, são: definir para onde o robô deve ir a cada etapa da exploração, permitindo que ele mapeie todo o ambiente; definir um critério de parada para que o robô seja capaz de estimar quando o ambiente está suficientemente explorado; evitar que o robô colida com obstáculos enquanto se desloca. Um fator que deve ser levado em consideração é que regiões com baixa saliência visual podem não ser mapeadas, fazendo com que o mapa possua lacunas associadas a estas. Assume-se que o robô possui as mesmas capacidades de movimentação de um VANT, pois estes são ideais para exploração de ambientes 3D complexos.

O método proposto extrai um mapa volumétrico local da nuvem de pontos gerada pelo ORB-SLAM, estimando neste mapa a região livre pela qual o robô pode navegar. Depois, tenta-se extrair objetivos locais dentro do mapa local. Quando não existirem objetivos locais, uma exploração global é realizada definindo objetivos fora da região do mapa local. Na Seção 3.1 é apresentada a estrutura básica da exploração. A geração do mapa local é apresentada na Seção 3.2. A movimentação do robô é abordada na Seção 3.3. O planejador e a lógica de exploração, responsáveis por definir para onde ir e quando a exploração acabou, são explicados em detalhes na Seção 3.4. Por fim, na Seção 3.5 é apresentado a variação do algoritmo A* utilizada para planejar caminhos.

3.1 Estrutura básica da exploração

O processo de exploração proposto utiliza uma série de estados que se relacionam conforme o diagrama apresentado na Figura 3.1. Uma descrição detalhada dos processos internos de cada estado, assim como as regras de transição, são apresentadas ao longo desde capítulo. Os estados da exploração são:

- *Aguardando*: Estado inicial do sistema, aguarda até que o SLAM inicialize. Este é também o estado para o qual a exploração retornará caso o SLAM perca a estimativa da pose.
- *Atualizando Mapa Local*: Estado responsável por manter o mapa local.
- *Planejando*: Estado onde um planejador decide qual ação o robô deve realizar para prosseguir explorando. O planejador pode definir uma posição alvo para o robô ou um movimento circular para que o mesmo olhe ao seu redor. Além disso, o planejador é o responsável por determinar quando a exploração está completa.
- *Navegando*: Estado responsável por rastrear o movimento do robô marcando as regiões visitadas do mapa.
- *Olhando ao Redor*: Estado responsável por rastrear a rotação do robô.
- *Finalizado*: Estado final da exploração, alcançado quando o planejador define que a exploração está completa.

Figura 3.1: Relações entre os estados da exploração.

O ciclo de estados da exploração pode ser interrompido a qualquer momento de acordo com o que está acontecendo no SLAM. Se o SLAM perder a estimativa sobre a pose do robô, a exploração retorna, não importando em qual estado ela esteja, para o estado *Aguardando* até que uma estimativa volte a ser obtida. Também, quando um *loop* é fechado pelo SLAM, e consequentemente a nuvem de pontos é corrigida, a exploração retorna para o estado *Atualizando Mapa Local*. Um fechamento de *loop* torna o mapa local obsoleto e pode causar grande alteração na estimativa sobre a pose do robô, o que causaria um comportamento incorreto do sistema. Além disso, caso o robô saia da região representada no mapa local, a exploração retorna para o estado *Atualizando Mapa Local*.

3.2 Linhas de visão e mapa local

O ORB-SLAM gera um mapa esparso do ambiente, representado por uma nuvem de pontos, assim como a maioria dos SLAMs monoculares. Ele é composto por um conjunto de *keyframes* $\boldsymbol{K}$, associados às poses $\boldsymbol{x} \in \boldsymbol{X}$ que a câmera (i.e. o robô) assumiu, e um conjunto de pontos $\boldsymbol{P}$, associados aos obstáculos observados. Esse mapa não representa o espaço livre do ambiente e não está em um formato adequado para navegação e exploração.

Propõe-se utilizar um mapa local gerado a partir deste mapa global. O mapa local, representado por $\boldsymbol{V} = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3, ..., \boldsymbol{v}_n\}$, é uma grade regular 3D que contém $n = 100^3$ *voxels*. Cada *voxel* contém um estado referente à região que ele engloba, de tamanho ξ^3, podendo este estado ser: *livre*, *ocupado* ou *desconhecido*. Cada *voxel* $\boldsymbol{v}_i \in \boldsymbol{V}$ armazena as seguintes informações:

- $\boldsymbol{y}_i$: Posição em coordenadas cartesianas $(x, y, z)^T$ do centro do *voxel*.
- β_i: Quantidade de indícios de que o *voxel* está livre, limitado a $\ddot{\beta}$.
- $\mathcal{P}_i$: Pontos $\mathcal{P}_i \subseteq \boldsymbol{P}$ que estão dentro da área discretizada pelo *voxel*.

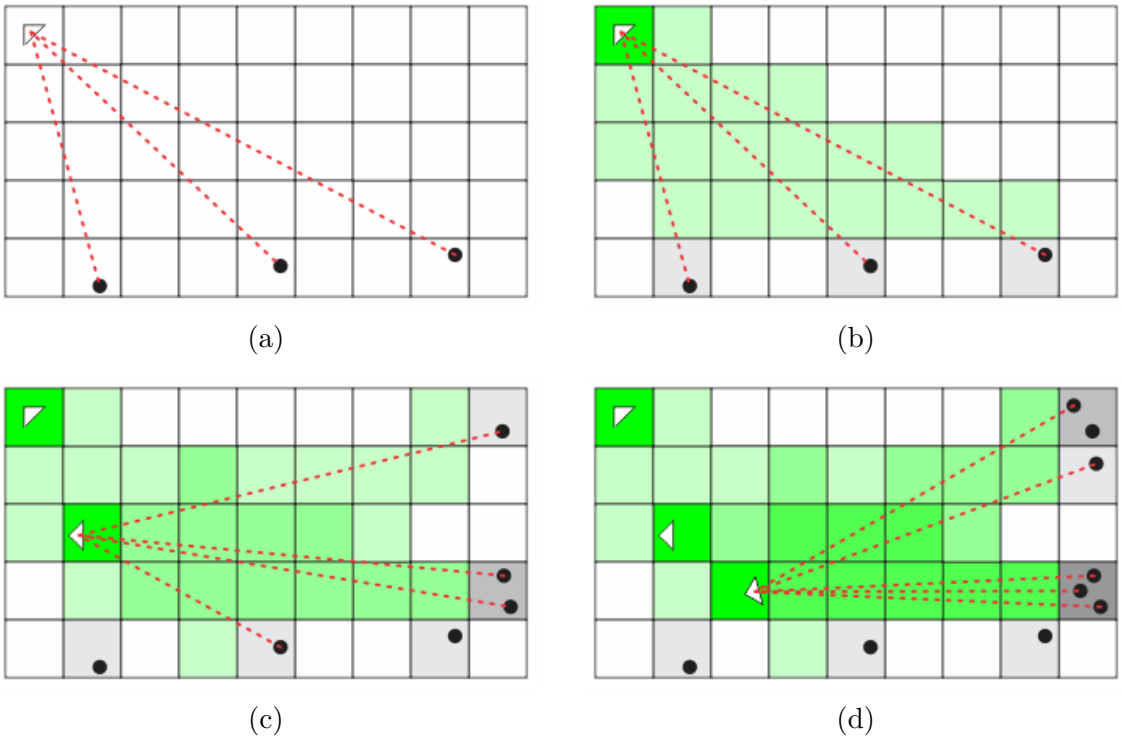
Antes de definir como o mapa local é gerado, é necessário esclarecer que o ORB-SLAM fornece, a todo momento, um *keyframe* de referência $\boldsymbol{k}_{ref}$. Este é o *keyframe* mais similar ao *frame* atual. O $\boldsymbol{k}_{ref}$ é utilizado como base para a geração do mapa local.

Quando o estado da exploração passa a ser *Atualizando Mapa Local*, é iniciado o processo de geração do mapa local. Inicialmente, é formado um conjunto de *keyframes* vizinhos a $\boldsymbol{k}_{ref}$, representado por $\hat{\mathcal{K}} \subseteq \boldsymbol{K}$. Ele é obtido através do grafo de covisibilidade e é composto por uma quantidade miníma κ de *keyframes*. Todos os vizinhos de $\boldsymbol{k}_{ref}$ no grafo de covisibildiade, representados por $\mathcal{K}_{ref}$, são colocados em $\hat{\mathcal{K}}$. Enquanto o número de elementos em $\hat{\mathcal{K}}$ for menor que κ, o melhor vizinho $\boldsymbol{k}_a \in \mathcal{K}_{ref}$ é escolhido, e todos os *keyframes* de $\mathcal{K}_a$ são inseridos em $\hat{\mathcal{K}}$. Repete-se esse processo utilizando o melhor vizinho de $\boldsymbol{k}_a$ até existirem pelo menos κ elementos em $\hat{\mathcal{K}}$ ou não existirem mais *keyframes* a serem colocados.

Após definir $\hat{\mathcal{K}}$, um conjunto de processos é aplicado para cada $\boldsymbol{k}_a \in \hat{\mathcal{K}}$. Inicialmente, o *voxel* que engloba a posição do *keyframe* $\boldsymbol{k}_a$ é considerado livre. Então, cada ponto $\boldsymbol{p}_b \in \hat{\boldsymbol{P}}_a$ que esteja dentro da área do mapa local é associado ao *voxel* $\boldsymbol{v}_i$ que engloba sua posição e é inserido em $\mathcal{P}_i$. Por fim, uma linha de visão

é traçada entre $\boldsymbol{k}_a$ e $\boldsymbol{p}_b$ utilizando o algoritmo DDA. Cada *voxel* na linha de visão recebe um indício de que está livre, aumentando seu valor β. A Figura 3.2 apresenta um exemplo do processo de criação do mapa local, em 2D para melhor visualização.

Figura 3.2: Exemplo de geração do mapa. Em (a) encontra-se sobre o mapa o primeiro *keyframe*, os pontos por ele visto e as linhas de visão, representados respectivamente pelo triangulo, pelas bolas e pelas linhas vermelhas. Em (b) os *voxels* processados. Quanto mais indícios de que um *voxel* está livre, mais verde. Da mesma forma, quanto maior o número de pontos associados, mais preto. Em (c) e (d) o estado do mapa após alguns *keyframes* serem processados.



(a) (b) (c) (d)

Após processar todos *keyframes* de $\hat{\mathcal{K}}$ o mapa local está completo. Então, é possível determinar o estado de cada *voxel* $\boldsymbol{v}_i \in \boldsymbol{V}$ através de

$$e(\boldsymbol{v}_i) = \begin{cases} \textit{ocupado} & \text{, se } \ |\mathcal{P}_i| \geq \lambda \\ \textit{livre} & \text{, se } \ \beta_i \geq \mu \ \wedge \ |\mathcal{P}_i| < \lambda \\ \textit{desconhecido} & \text{, caso contrário} \end{cases} , \tag{3.1}$$

sendo $|\mathcal{P}_i|$[1] a quantidade de pontos em $\mathcal{P}_i$, λ o número mínimo de pontos para considerar o *voxel* ocupado, β_i a quantidade de indícios de livre e μ o número mínimo de indícios para considerar o *voxel* livre.

[1]A notação que será utilizada nesta dissertação para se referir a quantidade de elementos em conjuntos é: $|conjunto|$. Por exemplo, a quantidade de elementos em $\mathcal{P}_i$ é dada por $|\mathcal{P}_i|$.

Como cada *voxel* representa um volume do ambiente, é possível que um *voxel* $\boldsymbol{v}_i$ possua diversos pontos em $\mathcal{P}_i$ e diversos indícios de liberdade. Por isso, para que ele seja considerado livre é preciso que $|\mathcal{P}_i| < \lambda$ seja verdadeiro. São utilizados dois acumuladores diferentes, um para ocupado e outro para livre, pois em casos específicos a quantidade de indícios de livre pode ser suficientemente maior que o $|\mathcal{P}_i|$, o que tornaria o *voxel* livre erroneamente, caso apenas um acumulador fosse utilizado.

3.3 Navegando e olhando ao redor

Quando a exploração encontra-se nos estados *Navegando* ou *Olhando ao Redor*, o sistema passa a rastrear a movimentação do robô conforme estimada pelo ORB-SLAM, analisando se ele já chegou no objetivo. Não está no escopo desta dissertação o controle do robô para que ele execute a rotação do estado *Olhando ao Redor* ou o deslocamento do estado *Navegando*. Além de rastrear o movimento, o sistema mantém atualizado o conjunto $\mathcal{P}_i$ de cada *voxel* $\boldsymbol{v}_i \in \boldsymbol{V}$, adicionando a ele os novos pontos da nuvem descobertos durante a movimentação. Também, durante a movimentação, os pontos da nuvem que se encontrarem a uma distância menor que η do robô são marcados como visitados. Essa análise é feita utilizando os *voxels* do mapa local $\boldsymbol{V}$. A pose $\boldsymbol{x}_t$ do robô é associada a um *voxel* $\boldsymbol{v}_i \in \boldsymbol{V}$ pela posição em coordenadas cartesianas. Então, os pontos $\mathcal{P}_c$ de cada *voxel* $\boldsymbol{v}_c \in \boldsymbol{V}$ em que $d(\boldsymbol{v}_i, \boldsymbol{v}_c) < \eta$ são marcados como visitados.

Se o estado da exploração for *Olhando ao Redor*, o objetivo do robô é apenas atingir uma variação de orientação em torno do eixo vertical, representada por $\Delta\psi$. Cabe ao sistema apenas detectar quando o robô rotacionou o suficiente. Já quando o estado for *Navegando*, o planejador, que será abordado na Seção 3.4, define um *voxel* destino para o robô, assim como um caminho para que ele o alcance. Cabe ao sistema rastrear os movimentos do robô, detectando quando ele chega ao destino. Quando o robô atinge o objetivo, esteja ele *Olhando ao Redor* ou *Navegando*, o estado da exploração passa a ser *Atualizando Mapa Local*.

3.4 Planejador e a lógica de exploração

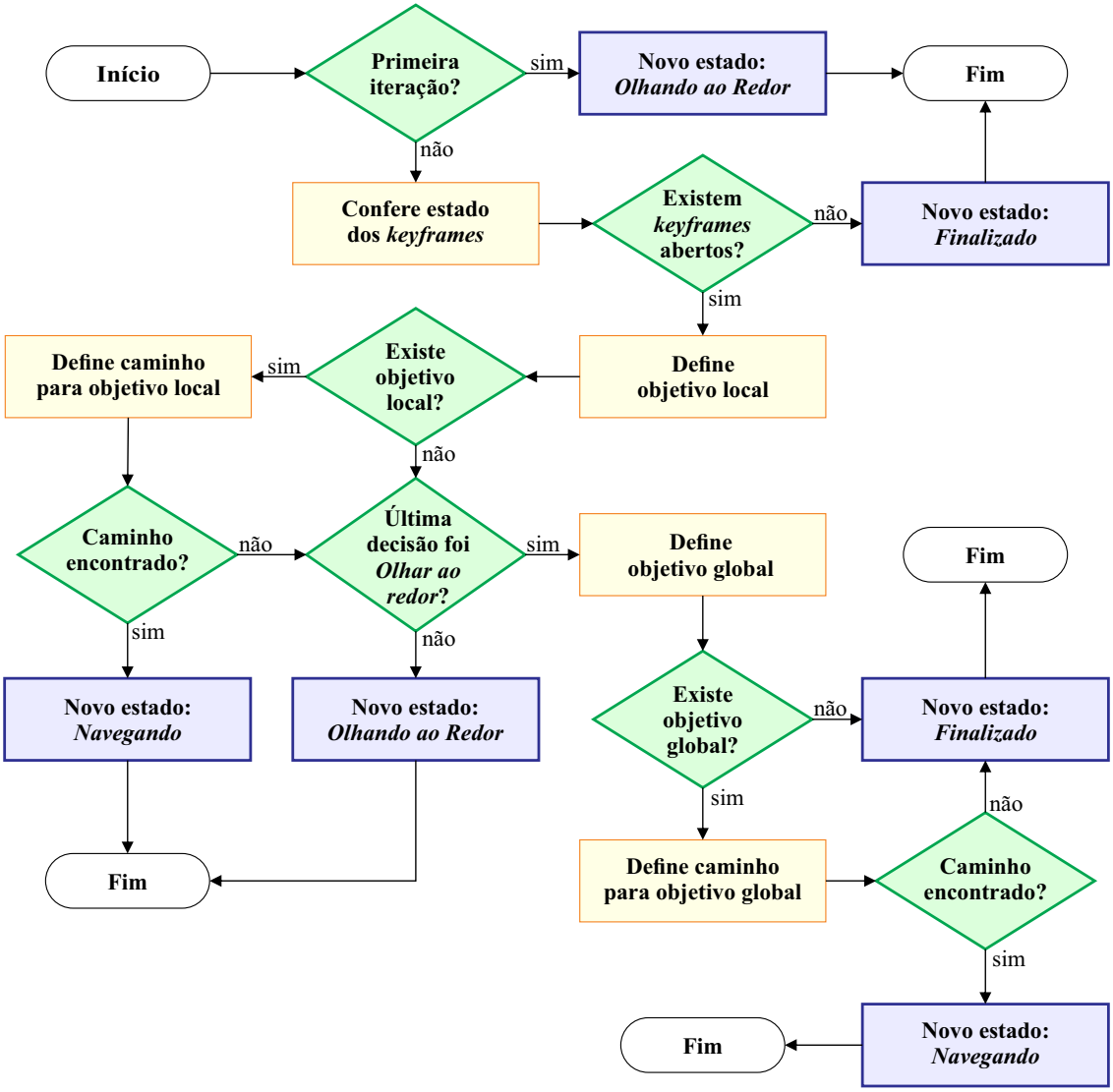
O planejador é o elemento central do processo de exploração proposto nesta dissertação. Cabe a ele, a cada iteração, decidir qual é a melhor ação a ser tomada para prosseguir explorando. Além disso, é sua função decidir se a exploração está completa. Quando o estado da exploração passa para *Planejando*, um fluxograma é executado para definir o próximo estado da exploração (i.e. a próxima ação). As decisões são tomadas baseadas na decisão anterior, no mapa local e, quando necessário, no mapa global. O resultado do planejador é sempre um novo estado para a exploração. Quando o resultado final for o estado *Navegando*, também fica definido o objetivo e o caminho a ser seguido. Já quando for *Olhando ao Redor*, fica definido para qual direção rodar. Por fim, se o resultado for o estado *Finalizado* o processo de exploração é finalizado e nenhum novo ciclo é executado.

A Figura 3.3 apresenta o fluxograma do planejador. A cor verde representa as decisões que compõem a lógica de exploração, a cor amarela representa os processos que suportam essa lógica e em azul estão representadas as mudanças de estado. Nesta seção será abordado em detalhes os processos que suportam a lógica de exploração, representados em amarelo no fluxograma.

3.4.1 Confere estado dos *keyframes*

No inicio do processo de decisão, todos os *keyframes* $\boldsymbol{k}_a \in \boldsymbol{K}$ são analisados de acordo com o conjunto de pontos $\hat{\boldsymbol{P}}_a$ visíveis de $\boldsymbol{k}_a$. Cada *keyframe* $\boldsymbol{k}_a \in \boldsymbol{K}$ pode estar *aberto*, caso $\hat{\boldsymbol{P}}_a$ possua um percentual mínimo ϱ de pontos não visitados, ou *fechado* caso não possua. É considerado um percentual mínimo devido ao ruido presente no mapa, o qual gera pontos inalcançáveis, que impediriam a exploração de finalizar caso fosse exigido que todos pontos do mapa fossem visitados. Os pontos da nuvem são marcados como visitados quando eles se encontram a uma distância menor ou igual a η do robô, conforme abordado na Seção 3.3. Após avaliar o estado dos *keyframes*, se todos eles estiverem *fechados* o processo de exploração está completo e o planejador define o estado da exploração como *Finalizado*. Caso pelo menos um *keyframe* esteja *aberto*, o processo de decisão prossegue.

Figura 3.3: Fluxograma do Planejador. Em verde as decisões que compõem a lógica, em amarelo os processos que suportam essa lógica e em azul as mudanças de estado.



3.4.2 Define objetivo local

Enquanto existirem *keyframes abertos*, o planejador busca por objetivos para prosseguir explorando. A primeira tentativa é realizada sobre o mapa local $\boldsymbol{V}$. Gera-se uma vizinhança composta por todos *voxels* que estão ω-distantes de $\boldsymbol{v}_i$, representada por $\mathcal{L}_i \subseteq \boldsymbol{V}$. A atratividade de um *voxel* $\boldsymbol{v}_i$ é obtida através de

$$a(\boldsymbol{v}_i) = \sum_{\boldsymbol{v}_c \in \mathcal{L}_i} \frac{a^*(\boldsymbol{v}_c)}{1 + d(\boldsymbol{v}_i, \boldsymbol{v}_c)} \quad , \tag{3.2}$$

sendo $d(\boldsymbol{v}_i, \boldsymbol{v}_c)$ a distância euclidiana entre dois *voxels* e $a^*(\boldsymbol{v}_c)$ a atratividade individual de um *voxel* $\boldsymbol{v}_c$, obtida através de

$$a^*(\boldsymbol{v}_c) = \sum_{\boldsymbol{p}_b \in \hat{\mathcal{P}}_c} |\hat{\boldsymbol{K}}_b| \quad , \tag{3.3}$$

sendo $\hat{\mathcal{P}}_c \subseteq \mathcal{P}_c$ os pontos da nuvem associados a $\boldsymbol{v}_c$ que ainda não foram visitados e $|\hat{\boldsymbol{K}}_b|$ a quantidade de *keyframes* que observam um ponto $\boldsymbol{p}_b$.

A atratividade é calculada utilizando uma vizinhança para agrupar os *voxels* que são atrativos individualmente e estão próximos. Assim, o objetivo tende a ser um ponto intermediário que englobe alguns *voxels* atrativos. Para que um *voxel* $\boldsymbol{v}_i \in \boldsymbol{V}$ seja um candidato a objetivo, ele deve ser, além de atrativo, interessante. Um *voxel* $\boldsymbol{v}_i$ é definido como interessante através de

$$n(\boldsymbol{v}_i) = \begin{cases} \textit{verdadeiro} & \text{, se } |\hat{\mathcal{P}}_i| > \alpha \quad \wedge \quad \frac{|\hat{\mathcal{P}}_i|}{|\mathcal{P}_i| - |\hat{\mathcal{P}}_i|} > \vartheta \\ \textit{falso} & \text{, caso contrário} \end{cases} \quad , \tag{3.4}$$

sendo α a quantidade mínima de pontos não visitados e ϑ a proporção mínima entre pontos visitados e não visitados. Assim, o *voxel* que maximiza a atração, dentre aqueles que são interessantes, é definido como o objetivo $\boldsymbol{v}_d$. Se um objetivo local for encontrado o planejador busca então um caminho para ele. Se não existir um objetivo local, ele analisa sua última decisão e caso ela tenha sido *Olhar ao Redor*, ele busca definir um objetivo global. Caso contrário, ele passa a exploração para o estado *Olhar ao Redor*, visando aumentar as informações locais.

3.4.3 Define caminho para objetivo local

Se o planejador encontrar um objetivo local, ele analisa então se existe um caminho possível que leve o robô até este objetivo. O algoritmo utilizado para o planejamento de caminhos é uma versão modificada do A* (HART; NILSSON; RAPHAEL, 1968) e é apresentado em detalhes na Seção 3.5. Neste ponto, é importante adiantar apenas que a busca por um caminho para o objetivo $\boldsymbol{v}_d$ é na verdade a busca por um caminho que leve o robô até uma distância mínima η de $\boldsymbol{v}_d$. Essa é a mesma distância utilizada para determinar que um ponto foi visitado, como abordado na Seção 3.3. Isso visa evitar que o robô seja levado até um obstáculo, uma

vez que os objetivos locais são extraídos com base em pontos não visitados associados a *features* do ambiente, ou seja, obstáculos. Se um caminho for encontrado, então, o planejador passa o estado da exploração para *Navegando*. Se não existir um caminho, o planejador analisa sua última decisão e caso ela tenha sido *Olhar ao Redor*, ele busca definir um objetivo global. Caso contrário, ele passa a exploração para o estado *Olhar ao Redor*, visando aumentar as informações locais.

3.4.4 Define objetivo global

Quando não existe um caminho até o objetivo local, ou não existe o próprio objetivo, o planejador inicialmente define que o robô deve olhar ao seu redor, buscando aumentar seu conhecimento sobre o ambiente. Após finalizar a ação de olhar ao redor, o ciclo da exploração inicia novamente, um novo mapa local é gerado e uma nova decisão é tomada pelo planejador. Se, mesmo tendo olhado ao seu redor, não for possível encontrar um objetivo local ou um caminho até ele, então o planejador busca por um objetivo global.

Inicialmente, busca-se um *keyframe* $\boldsymbol{k}_a \in \boldsymbol{K}$, dentre todos que ainda estejam *abertos*, para ser o objetivo global. Caso existam diversos *keyframes abertos* é escolhido aquele que está mais próximo do *keyframe* de referência $\boldsymbol{k}_{ref}$. Após definir o *keyframe* $\boldsymbol{k}_a$ que será objetivo global, é necessário encontrar um caminho para aproximar-se dele. Se posição de $\boldsymbol{k}_a$ estiver dentro do mapa $\boldsymbol{V}$ basta encontrar um caminho até o *voxel* correspondente.

Em determinados momentos o objetivo global pode estar fora da região coberta pelo mapa local $\boldsymbol{V}$. Isso ocorre, por exemplo, quando uma sala foi completamente explorada mas outra sala visitada anteriormente ainda possui pontos a serem explorados. Como a navegação do robô é realizada através do mapa local $\boldsymbol{V}$, é necessário encontrar um *keyframe* intermediário que esteja dentro dele. Para tal, um caminho sobre o grafo de covisibilidade é gerado entre $\boldsymbol{k}_{ref}$ e $\boldsymbol{k}_a$. Esse caminho é obtido utilizando a versão original do algoritmo A*, pois busca apenas determinar o conjunto de passos sobre os *keyframes* que conectam $\boldsymbol{k}_{ref}$ a $\boldsymbol{k}_a$.

Uma vez conhecendo esse caminho, basta seguí-lo de trás para frente, ou seja, partindo de $\boldsymbol{k}_a$, analisando cada *keyframe* até encontrar um que esteja dentro do mapa local $\boldsymbol{V}$. Assim, é obtido aquele mais distante no caminho até $\boldsymbol{k}_a$ que ainda esteja dentro de $\boldsymbol{V}$. Este então passa a ser o novo objetivo global $\boldsymbol{k}_a$. No caso pouco

provável de não existir um *keyframe* intermediário, a exploração é encerrada. Esse caso é pouco provável pois o mapa local cobre uma área do ambiente suficientemente grande para englobar a posição de diversos *keyframe*.

3.4.5 Define caminho para objetivo global

Após obter o objetivo global $\boldsymbol{k}_a$, o planejador busca então um caminho no mapa local até ele. O algoritmo utilizado é a mesma versão modificada do algoritmo A* utilizada para encontrar o caminho até um objetivo local, apresentado em detalhes na Seção 3.5. Porém, existe uma diferença em relação a busca de caminho para o objetivo local, abordada na Subseção 3.4.3. Aqui, não utiliza-se uma margem de segurança η e o caminho deve levar o robô até o *voxel* correspondente a $\boldsymbol{k}_a$. Se existir um caminho, o planejador define o próximo estado da exploração como *Navegando*. Caso contrário, o estado da exploração passa a ser *Finalizado* e ela é encerrada.

3.5 Caminhos no mapa local

O planejamento de caminhos sobre o mapa local é realizado utilizando uma versão modificada do algoritmo A* (HART; NILSSON; RAPHAEL, 1968), apresentado na Subseção 2.1.1. O objetivo é encontrar um conjunto de poses $\boldsymbol{x} \in \boldsymbol{X}$ que levem o robô do *voxel* em que ele se encontra ao *voxel* destino, representados por $\boldsymbol{v}_p$ e $\boldsymbol{v}_d$ respectivamente. Note que o planejamento é feito sobre os *voxels* do mapa $\boldsymbol{V}$ e não sobre todas possíveis poses do robô, representadas por $\boldsymbol{X}$. Dessa forma, o processo de geração de caminhos é simplificado a encontrar um conjunto de *voxels* que levem o robô até o destino, representado por $\mathcal{C} = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3, ..., \boldsymbol{v}_q\}$, sendo q o número de passos (i.e. *voxels*) do caminho. Após encontra esse caminho, uma pose $\boldsymbol{x} \in \boldsymbol{X}$ pode ser extraída a partir das coordenadas do centro de cada *voxel* de $\mathcal{C}$, associando a cada passo um angulo ψ que é a orientação em torno do eixo vertical (*yaw*).

Para gerar caminhos seguros, apenas os *voxels* $\boldsymbol{v} \in \boldsymbol{V}_{livres}$ são considerados candidatos ao caminho. No A* original, apresentado no Algoritmo 1, o custo do caminho é definido apenas pela distância euclidiana entre os *voxels* $\boldsymbol{v} \in \mathcal{C}$. A distância

euclidiana entre dois *voxels* $\boldsymbol{v}$ e $\boldsymbol{v}'$ é obtida através de

$$d(\boldsymbol{v}, \boldsymbol{v}') = \sqrt{(x - x') + (y - y') + (z - z')}, \tag{3.5}$$

então, o custo do caminho até um *voxel* $\boldsymbol{v}_i \in \mathcal{C}$ por ser obtido através de

$$g(\boldsymbol{v}_i) = d(\boldsymbol{v}_1, \boldsymbol{v}_2) + d(\boldsymbol{v}_2, \boldsymbol{v}_3) + ... + d(\boldsymbol{v}_{i-1}, \boldsymbol{v}_i), \tag{3.6}$$

que pode ser reescrito como

$$g(\boldsymbol{v}_i) = g(\boldsymbol{v}_{i-1}) + d(\boldsymbol{v}_{i-1}, \boldsymbol{v}_i) \tag{3.7}$$

Essa abordagem encontra o menor caminho, em termos de deslocamento, entre a origem e o destino. Porém, o menor caminho não será, necessariamente, o caminho mais seguro. Propõe-se nessa dissertação acrescentar um novo custo a este cálculo, considerando a liberdade para navegar na vizinhança dos *voxels*. Gera-se uma vizinhança $\mathcal{L}_i \subseteq \boldsymbol{V}$ composta por todos *voxels* que estão δ-distantes de $\boldsymbol{v}_i$, conforme já abordado na Subseção 3.4.2. O custo de liberdade de navegação de um *voxel* $\boldsymbol{v}_i$, dado a vizinhança $\mathcal{L}_i$, é obtido através de

$$u(\boldsymbol{v}_i) = \zeta \sum_{\boldsymbol{v}_c \in \mathcal{L}_i} \frac{u^*(\boldsymbol{v}_c)}{1 + d(\boldsymbol{v}_i, \boldsymbol{v}_c)} \quad , \tag{3.8}$$

sendo ζ um fator normalizador e

$$u^*(\boldsymbol{v}_c) = \begin{cases} 2\ddot{\beta} & , \text{se } \; e(\boldsymbol{v}_c) = \textit{ocupado} \\ \ddot{\beta} & , \text{se } \; e(\boldsymbol{v}_c) = \textit{desconhecido} \\ \ddot{\beta} - \beta_c & , \text{se } \; e(\boldsymbol{v}_c) = \textit{livre} \end{cases} , \tag{3.9}$$

sendo $\ddot{\beta}$ o limite máximo de indícios de livre que um *voxel* pode ter, β_c a quantidade de indícios de livre e $e(\boldsymbol{v}_c)$ o estado do *voxel* $\boldsymbol{v}_c$. A Figura 3.4 apresenta um exemplo do custo de liberdade em um mapa 2D, para melhor visualização.

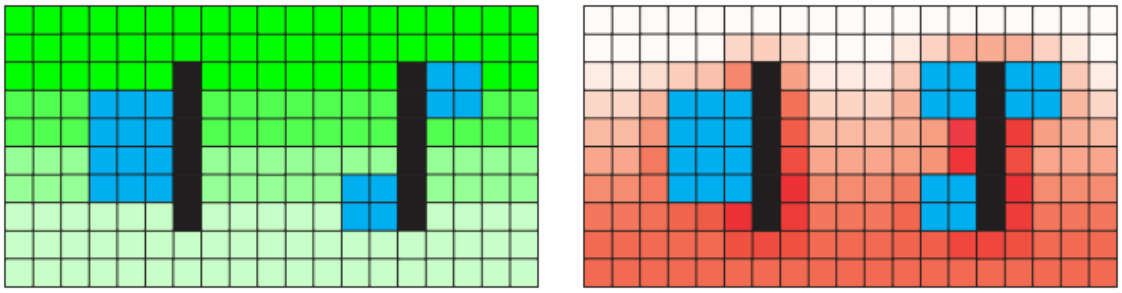
Este novo custo é somado ao custo da distância euclidiana utilizado na versão original do A*. Então, o custo do caminho até um *voxel* $\boldsymbol{v}_i \in \mathcal{C}$ apresentado na

Equação 3.7 passa agora a ser obtido através de

$$\underbrace{g(\boldsymbol{v}_i) = g(\boldsymbol{v}_{i-1}) + d(\boldsymbol{v}_{i-1}, \boldsymbol{v}_i)}_{\text{original}} + \underbrace{u(\boldsymbol{v}_i)}_{\text{modificação}} \tag{3.10}$$

Assim, a versão modificada prioriza caminhos que possuam *voxels* com mais indícios de liberdade e que estejam distantes de obstáculos e regiões desconhecidas, sem deixar de considerar a distância a ser percorrida.

Figura 3.4: Exemplo do custo de liberdade proposto para o A*. Em (a) um mapa com os *voxels* ocupados em preto, desconhecidos em azul e livres em verde, sendo mais verde os *voxels* com mais indícios de liberdade. Em (b) o custo de liberdade representado em vermelho, sendo mais vermelho os *voxels* com maior custo.



(a) (b)

Conforme abordado na Seção 3.4, o planejador gera caminhos em dois casos distintos. O primeiro caso é quando uma região do mapa local é atrativa o suficiente para continuar a exploração, gerando um *voxel* destino. O segundo caso é quando um objetivo global é gerado para que o robô retorne a uma região que ficou em aberto, gerando um *keyframe* destino. Para o primeiro caso, propõe-se nessa dissertação mais uma modificação ao A* original. Uma vez que consideramos para determinar a atratividade de um *voxel* a quantidade de pontos não visitados associados a ele, nosso objetivo $\boldsymbol{v}_d$ será provavelmente um *voxel* ocupado, ou próximo a *voxels* ocupados. Não seria sensato criar um caminho que leve o robô a esse $\boldsymbol{v}_d$. Então, propõe-se encontrar um caminho que leve o robô até uma distância miníma η de $\boldsymbol{v}_d$. O Algoritmo 3 apresenta as modificações propostas, ressaltando que as modificações nas linhas 7, 11 a 13, 29 e 30 só são utilizadas quando planeja-se um caminho para uma região atrativa do mapa local.

Após obter um caminho $\mathcal{C}$ até o destino $\boldsymbol{v}_d$, é necessário definir uma orientação em torno do eixo vertical (*yaw*) para cada $\boldsymbol{v}_i \in \mathcal{C}$, representada por ψ_i. Olhar na direção do movimento não é uma boa escolha quando o objetivo é aumentar o conhecimento sobre o ambiente. Para tal, é preciso olhar para outras regiões. Uma

Algoritmo 3: Algoritmo A* modificado. Em vermelho as linhas modificadas em relação a versão original, apresentada no algoritmo 1.

Entrada: $\boldsymbol{v}_p$, $\boldsymbol{v}_d$; `// Pontos de partida e destino`
Saída: *Caminho* ; `// ∅ caso não exista um caminho`

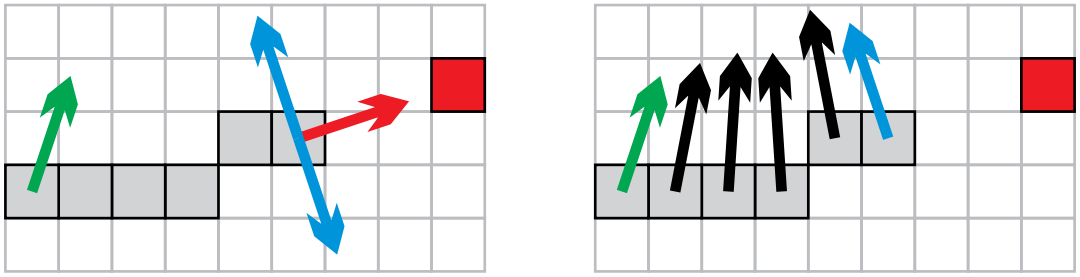
1 $\boldsymbol{V}_{abertos} = \emptyset$;
2 $\boldsymbol{V}_{fechados} = \emptyset$;
3 $g(\boldsymbol{v}_p) = 0$;
4 $h(\boldsymbol{v}_p) = Heurística(\boldsymbol{v}_p, \boldsymbol{v}_d)$;
5 $f(\boldsymbol{v}_p) = g(\boldsymbol{v}_p) + h(\boldsymbol{v}_p)$;
6 $pai(\boldsymbol{v}_p) = \boldsymbol{v}_p$;
7 $g(\boldsymbol{v}_d) = \infty$;
8 Insere $\boldsymbol{v}_p$ em $\boldsymbol{V}_{abertos}$;
9 **enquanto** $\boldsymbol{V}_{abertos} \neq \emptyset$ **faça**
10 $\quad \boldsymbol{v}_c$ = Elemento $\boldsymbol{v}$ de $\boldsymbol{V}_{abertos}$ que minimize $f(\boldsymbol{v})$;
11 $\quad$ **se** $d(\boldsymbol{v}_c, \boldsymbol{v}_d) \leq \eta \wedge g(\boldsymbol{v}_c) < g(\boldsymbol{v}_d)$ **então**
12 $\quad\quad g(\boldsymbol{v}_d) = g(\boldsymbol{v}_c)$;
13 $\quad\quad pai(\boldsymbol{v}_d) = \boldsymbol{v}_c$;
14 $\quad$ Remove $\boldsymbol{v}_c$ de $\boldsymbol{V}_{abertos}$;
15 $\quad$ Insere $\boldsymbol{v}_c$ em $\boldsymbol{V}_{fechados}$;
16 $\quad$ **para** *cada vizinho* $\boldsymbol{v}_a \in \boldsymbol{V}_{livres}$ *de* $\boldsymbol{v}_c$ **faça**
17 $\quad\quad$ **se** $\boldsymbol{v}_a \notin \boldsymbol{V}_{fechados}$ **então**
18 $\quad\quad\quad$ **se** $\boldsymbol{v}_a \notin \boldsymbol{V}_{abertos}$ **então**
19 $\quad\quad\quad\quad g(\boldsymbol{v}_a) = \infty$;
20 $\quad\quad\quad\quad$ Insere $\boldsymbol{v}_a$ em $\boldsymbol{V}_{abertos}$;
21 $\quad\quad\quad custo = g(\boldsymbol{v}_c) + d(\boldsymbol{v}_c, \boldsymbol{v}_a) + u(\boldsymbol{v}_c)$;
22 $\quad\quad\quad$ **se** $custo < g(\boldsymbol{v}_a)$ **então**
23 $\quad\quad\quad\quad g(\boldsymbol{v}_a) = cost$;
24 $\quad\quad\quad\quad pai(\boldsymbol{v}_a) = \boldsymbol{v}_c$;
25 $\quad\quad\quad\quad h(\boldsymbol{v}_a) = Heurística(\boldsymbol{v}_a, \boldsymbol{v}_d)$;
26 $\quad\quad\quad\quad f(\boldsymbol{v}_a) = g(\boldsymbol{v}_a) + h(\boldsymbol{v}_a)$;
$\quad\quad$ **fim**
$\quad$ **fim**
27 **retorna** *RastrearCaminho*() ;
28 **Função** *RastrearCaminho*`()`**:**
29 $\quad$ **se** $g(\boldsymbol{v}_d) == \infty$ **então**
30 $\quad\quad$ **retorna** $\emptyset$;
31 $\quad Caminho = \emptyset$;
32 $\quad \boldsymbol{v}_c = \boldsymbol{v}_d$;
33 $\quad$ Insere $\boldsymbol{v}_c$ em *Caminho* ;
34 $\quad$ **enquanto** $pai(\boldsymbol{v}_c) \neq \boldsymbol{v}_c$ **faça**
35 $\quad\quad \boldsymbol{v}_c = pai(\boldsymbol{v}_c)$;
36 $\quad\quad$ Insere $\boldsymbol{v}_c$ em *Caminho* ;
$\quad$ **fim**
37 **retorna** *Caminho* ;

alternativa custosa seria analisar a situação atual da nuvem e definir a orientação ψ_i que minimiza a incerteza sobre o mapa. Como deseja-se uma abordagem simples, propõe-se definir uma orientação $\hat{\psi}$ final e associar a cada passo um ângulo intermediário que leve o robô da orientação em que ele se encontra à orientação final. Novamente existem diferenças entre o planejamento realizado para uma região atrativa ou para um *keyframe* do mapa global. Se o destino for um *keyframe*, o $\hat{\psi}$ é a orientação deste.

Quando o destino é um *voxel* atrativo do mapa local, primeiro, encontra-se a orientação que faça o robô olhar diretamente para $\boldsymbol{v}_d$, lembrando que o último passo do caminho não será o próprio *voxel* $\boldsymbol{v}_d$, mas sim um *voxel* a uma distância máxima η dele. Após encontrar essa orientação, são criadas duas orientações candidatas que ficam paralelas a esta. A Figura 3.5(a) mostra a orientação para olhar diretamente para $\boldsymbol{v}_d$ e as duas candidatas. É escolhida como $\hat{\psi}$ a candidata que estiver mais próxima da orientação atual do robô, evitando assim grandes rotação.

Após definir $\hat{\psi}$, é necessário calcular a orientação ψ_i intermediária associada a cada $\boldsymbol{v}_i \in \mathcal{C}$. Para tal, é necessário dividir a diferença entre $\hat{\psi}$ e orientação atual do robô em $|\mathcal{C}| - 1$ passos criando uma variação. Então, ao primeiro passo é associada a orientação atual do robô e para cada passo subsequente é adicionada a variação, fazendo com que o último passo seja associado a $\hat{\psi}$. Assim, ao longo do caminho o robô fará uma transição suave da orientação em que se encontra para a orientação destino. A Figura 3.5(b) apresenta um exemplo da orientação ao longo do caminho.

Figura 3.5: Processo para definir a orientação ao longo do caminho. O *voxel* vermelho é $\boldsymbol{v}_d$, o caminho $\mathcal{C}$ em cinza, a orientação atual em verde, os candidatos a $\hat{\psi}$ são representados pelas setas azuis e as setas pretas são as orientações ao longo do caminho.



(a) (b)

4 EXPERIMENTOS E DISCUSSÃO

Neste capítulo serão discutidos os testes que foram realizados para avaliar o método de exploração proposto que utiliza linhas de visão e mapas locais. Inicialmente, na Seção 4.1, será discutida a geração do mapa local utilizando apenas alguns *keyframes* do mapa global, avaliando a possibilidade de utilizar todos os *keyframes*. Em seguida, na Seção 4.2, são demonstrados os testes do processo completo de exploração em um ambiente simples (i.e. apenas uma sala). São discutidos, em detalhes, pontos relevantes da exploração como: os mapas gerados; o efeito do ruído no mapa do SLAM; a trajetória; a efetividade do critério de parada e o tempo de processamento. Por fim, na Seção 4.3, são demonstrados os experimentos em um ambiente com múltiplas salas, avaliando a capacidade da exploração de levar robô em segurança de um ambiente para o outro. Todos os testes foram realizados utilizando os parâmetros apresentados na Tabela 4.1.

Tabela 4.1: Parâmetros utilizados nos testes.

Parâmetro	**Valor**	**Descrição**
κ	30	Quantidade mínima de *keyframes* para gerar o mapa local.
λ	5	Mínimo $\lvert\mathcal{P}_i\rvert$ para considerar o *voxel* $\boldsymbol{v}_i$ ocupado.
μ	5	Valor mínimo de β_i para considerar o *voxel* $\boldsymbol{v}_i$ livre.
ϱ	20%	Percentual mínimo de pontos não visitados em $\hat{\boldsymbol{P}}_a$ para o *keyframe* $\boldsymbol{k}_a$ ser considerado aberto.
α	5	Mínimo $\lvert\hat{\mathcal{P}}_i\rvert$ para que $\boldsymbol{v}_i$ seja interessante.
$\ddot{\beta}$	10	Quantidade máxima de indícios de livre que um *voxel* pode possuir.
ϑ	1.05	Proporção mínima entre $\lvert\hat{\mathcal{P}}_i\rvert$ e $\lvert\mathcal{P}_i\rvert$ para que $\boldsymbol{v}_i$ seja interessante.
$\Delta\psi$	90°	Variação mínima em torno do eixo vertical (*yaw*).
η	4	Distância miníma para considerar um *voxel* visitado.
ω	1	Variação aplicada sobre os índices de um *voxel* formando a vizinhança utilizada na distribuição da atração.
δ	2	Variação aplicada sobre índices de um *voxel* formando a vizinhança utilizada no planejamento de caminhos.
ξ	0.15	Tamanho da lateral de um *voxel* do mapa local, sendo igual valor para altura e profundidade.

4.1 Tempo para gerar o mapa local

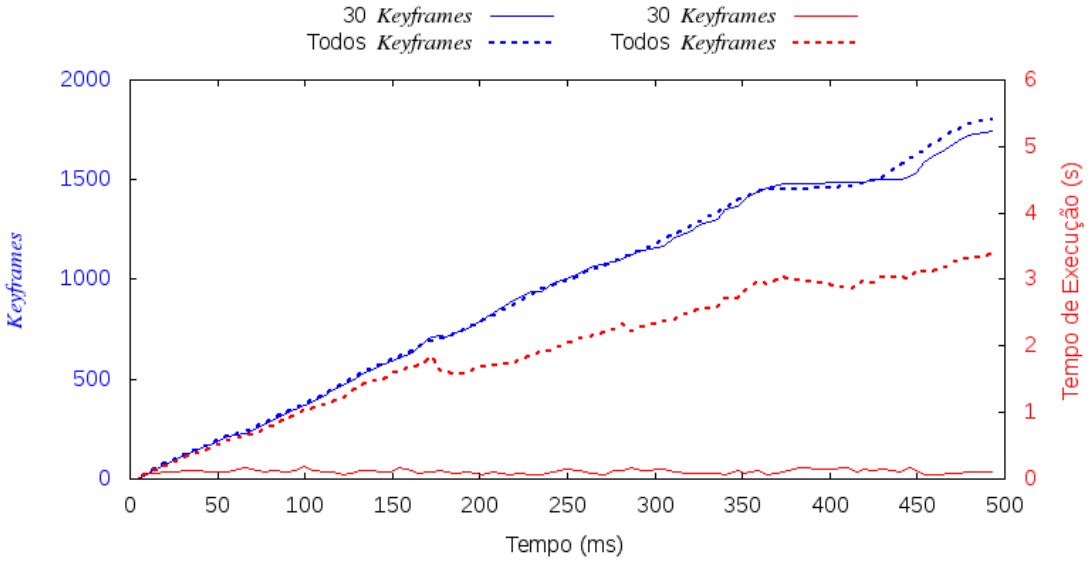
O primeiro experimento visa analisar a viabilidade de processar todos os *keyframes* que compõem o mapa global na geração do mapa local. Conforme abordado na Seção 3.2, durante o processo de geração do mapa local apenas uma parte dos *keyframes* são utilizados. Como consequência, há possivelmente menos informação disponível no mapa local, mas, por outro lado, isso permite que ele seja gerado de forma mais rápida. O experimento foi realizado utilizando a sequência 00 do *dataset* KITTI Monocular (GEIGER; LENZ; URTASUN, 2012), em um computador equipado com 16GB de memória RAM e um processador Intel Core i7-6700 3.4GHz.

A inicialização do mapa gerado pelo ORB-SLAM possui elementos aleatórios, conforme abordado na Subseção 2.4.2.2. Por isso, não é possível repetir exatamente o mesmo experimento, uma vez que o mapa global muda a cada execução do ORB-SLAM. Então, o experimento foi realizado executando duas vezes o processo de exploração. Na primeira execução o mapa local foi gerado utilizando todos os *keyframes*, enquanto na segunda foi utilizado um número mínimo de κ *keyframes* conforme descrito na Seção 3.2. Por utilizar um *dataset*, apenas o processo de geração do mapa local é avaliado, uma vez que o robô não responde as ações definidas pela exploração. Sempre que o robô sai da área representada pelo mapa local, um novo mapa é gerado.

A Figura 4.1 apresenta os resultados obtidos. Em azul, observa-se a quantidade de *keyframes* que compõem o mapa global e em vermelho o tempo gasto para gerar o mapa local. Observa-se que a quantidade de *keyframes* totais pode sofrer pequenas variações devido ao processo de inicialização do mapa. Em termos de performance, o tempo gasto para gerar o mapa local, quando todos os *keyframes* são utilizados, cresce conforme o número de *keyframes* aumenta, chegando a levar 3,36 segundos. Por outro lado, quando apenas uma parte desses *keyframes* é utilizada, o tempo permanece estável, com média 0,11 $\pm$ 0,033 segundos, tendo seu máximo em 0,189 segundos.

Fica evidenciado que utilizar todos os *keyframes* acarreta em um atraso no processo de exploração devido ao tempo de processamento, que tende a crescer conforme o mapa global aumenta. Assim, como o processo de exploração demanda decisões *online*, a manutenção de um mapa voumétrico local usando todos os *keyframes* se torna proibitiva. Por isso, utilizar uma quantidade κ de *keyframes*, é

Figura 4.1: Comparativo do tempo de execução para gerar o mapa local utilizando 30 *keyframes* e todos *keyframes*. Em azul o número de *keyframes* no mapa global e em vermelho o tempo de execução.



uma solução interessante que mantém o tempo de processamento dentro de uma margem conhecida, independente do tamanho do mapa global. Quanto maior for κ, mais informação será utilizada, porém, mais tempo será gasto na geração do mapa. É importante relembrar que o mapa não será gerado necessariamente com apenas κ *keyframes*, mas sim com um valor levemente maior ou igual a κ, conforme descrito na Seção 3.2.

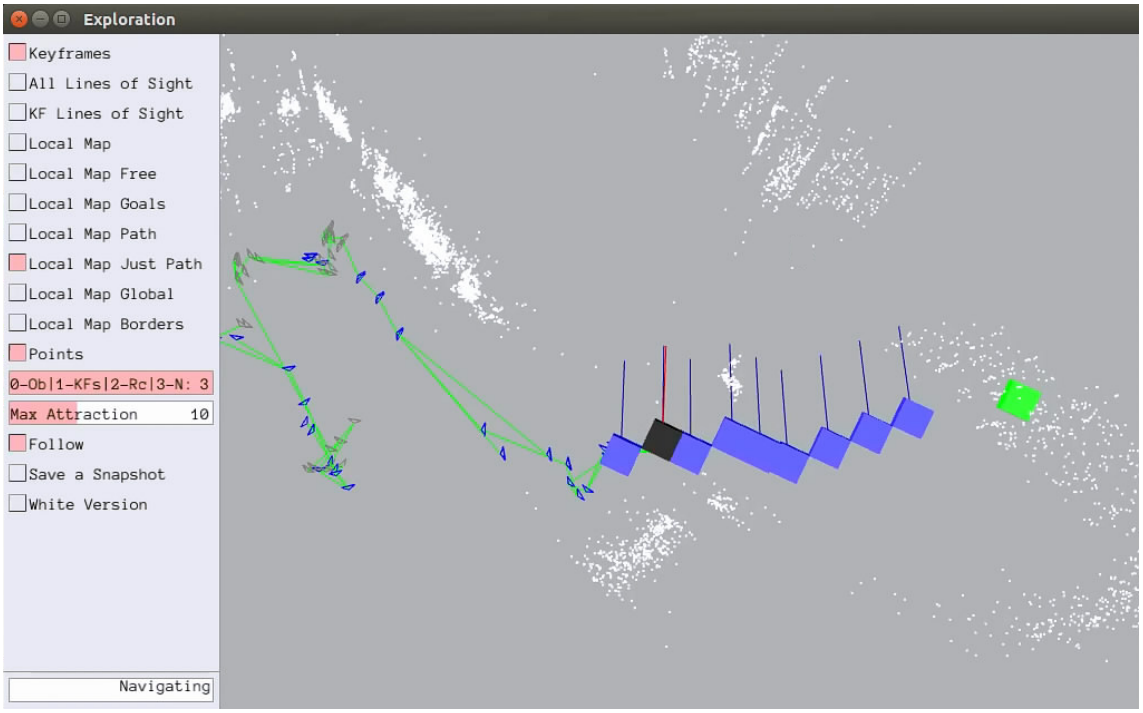
4.2 Exploração em ambiente simples

A utilização das linhas de visão para gerar mapas locais e assim explorar um ambiente foi avaliada, inicialmente, utilizando um ambiente simples, contendo apenas uma sala. Neste experimento, as imagens foram capturadas com uma câmera modelo Logitech C270[1] e o processamento foi realizado em um notebook Samsung Expert X51[2], equipado com 8GB de memória RAM e um processador Intel Core i7-7500U 2.70GHz. Nestes primeiros testes para avaliar a proposta, o método não foi aplicado diretamente sobre um VANT autônomo situado em um ambiente *indoor*, reduzindo-se assim os riscos e a complexidade do processo experimental. Como o objetivo principal era validar o funcionamento da estratégia de exploração, a movimentação da câmera foi realizada por uma pessoa, respeitando as ações indicadas

[1]http://www.logitech.com/product/hd-webcam-c270
[2]http://www.samsung.com/br/pc/notebook-expert-x51-np500r5m-xw2br

pelo sistema de decisão. A tela vista pelo operador da câmera é apresentada na Figura 4.2. Nesta, apenas o mapa global e o caminho no mapa local são demonstrados, evitando despender recursos para desenhar o mapa local por completo.

Figura 4.2: Tela vista pelo operador. A esquerda, na parte superior, um menu contendo diversas configurações e, na parte inferior, o estado da exploração. Ao centro/direita está o mapa global, o caminho em azul, o *voxel* destino em verde e o *voxel* em que a câmera se encontra em preto. As linhas azuis apontam a direção em que a câmera deve olhar, enquanto a linha vermelha aponta a direção em que a câmera está olhando.



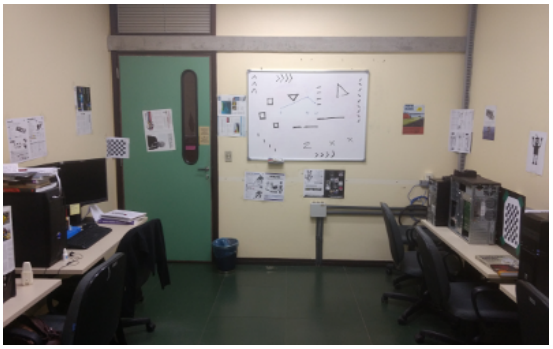
O operador da câmera deve observar alguns indicativos de movimento disponíveis no sistema de exploração durante a navegação. Quando o estado da exploração for *Olhar ao Redor*, o movimento deve ser feito com um certo balanço, evitando girar sobre o eixo da câmera, pois rotações puras são ruins para SLAMs Monoculares (MOSTEGEL, 2013). Durante a navegação, se o ORB-SLAM perder a estimativa sobre a pose da câmera, o operador deve voltar para a última pose em que a estimativa era conhecida.

O ambiente foi preparado para facilitar o funcionamento do ORB-SLAM, com a inserção de diversas marcações nas regiões sem saliência já que este SLAM depende de *features* e seu ponto fraco são as regiões homogêneas. A Figura 4.3 apresenta fotos do ambiente explorado. Foram realizadas quatro execuções de exploração no ambiente, denominadas: *Lab-R1*, *Lab-R2*, *Lab-R3* e *Lab-R4*[3]. Cada execução foi realizada

[3]Vídeos: *Lab-R1*, *Lab-R2*, *Lab-R3* e *Lab-R4*.

até a exploração estar completa, ou o ORB-SLAM falhar. O experimento *Lab-R1* foi executado com sucesso até o final, diferente dos experimentos *Lab-R2* e *Lab-R3*, que foram finalizados pois o ORB-SLAM estimou pontos do mapa em posições tão incorretas que perdeu a capacidade de detectar o movimento realizado pela câmera. O experimento *Lab-R4* foi encerrado no momento em que a exploração definiu como a próxima região a ser explorada um *voxel* que se encontrava fora do perímetro real do ambiente. Isso ocorreu devido a pontos ruidosos presentes no mapa global que estavam erroneamente posicionados fora do perímetro.

Figura 4.3: Fotos da sala utilizada nos experimentos em ambiente simples.



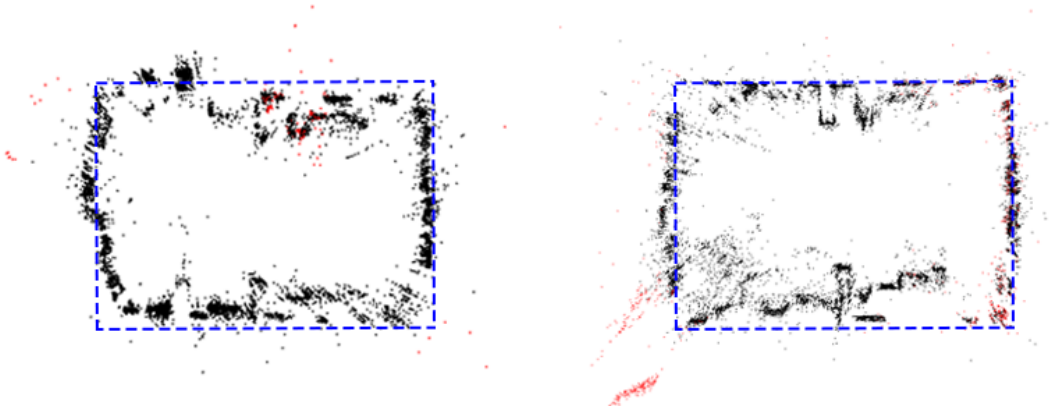
4.2.1 Mapas gerados

O resultado esperado, após um processo de exploração, é um mapa completo do ambiente. No entanto, não é possível analisar, de forma quantitativa, a completude de mapas esparsos, pois regiões de baixa saliência visual permanecerão desconhecidas e não existirão no mapa. Portanto, considera-se um mapa completo aquele que contemple o máximo possível de informação sobre o ambiente.

A Figura 4.4 apresenta o mapa final gerado nos quatro experimentos de exploração, sobrepostos pela planta baixa do ambiente. É possível observar que os mapas gerados representam o ambiente de forma satisfatória. Na Figura 4.4(a), observa-se o mapa gerado pelo experimento *Lab-R1*, que apesar de completo, detalha pouco as formas dos objetos. Este foi o experimento mais curto dos quatro, o que explica o menor detalhamento do ambiente. Na Figura 4.4(b) e Figura 4.4(d) observa-se os mapas gerados nos experimentos *Lab-R2* e *Lab-R4*, respectivamente. Apesar dos mapas estarem mais detalhados, estes contêm uma quantidade maior de ruído fora dos limites reais do ambiente. Não seria possível visitar os que estão mais distantes, o que impede a exploração de prosseguir corretamente. Na Figura 4.4(c) é possível

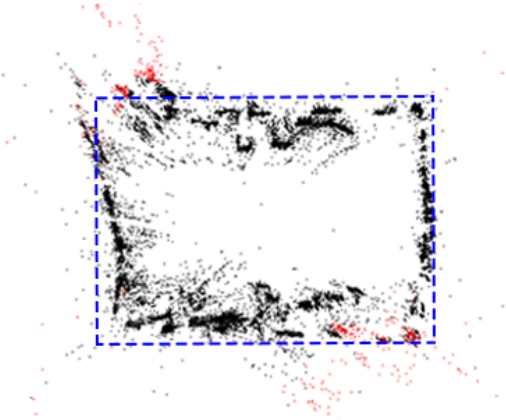
observar o mapa mais distorcido dentre os quatro, gerado no experimento *Lab-R3*. Essas variações obtidas no mapa final dependem de diversos fatores, dentre eles a trajetória desenvolvida pelo robô, conforme mostrado a seguir.

Figura 4.4: Mapas gerados nos quatro experimentos realizados em um ambiente simples, contendo apenas uma sala. A linha tracejada em azul representa as paredes da sala, ignorando os móveis internos. Em preto, os pontos visitados e, por último, os pontos em vermelho correspondem aos pontos abertos.

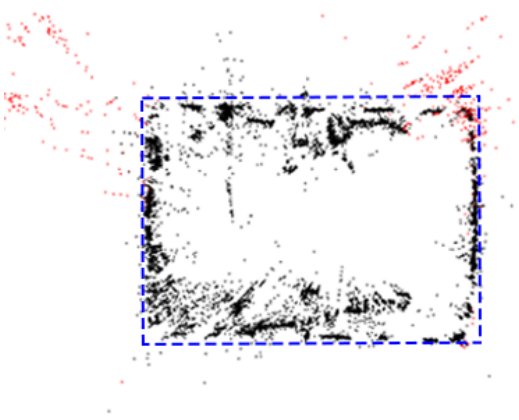


(a) *Lab-R1*

(b) *Lab-R2*



(c) *Lab-R3*



(d) *Lab-R4*

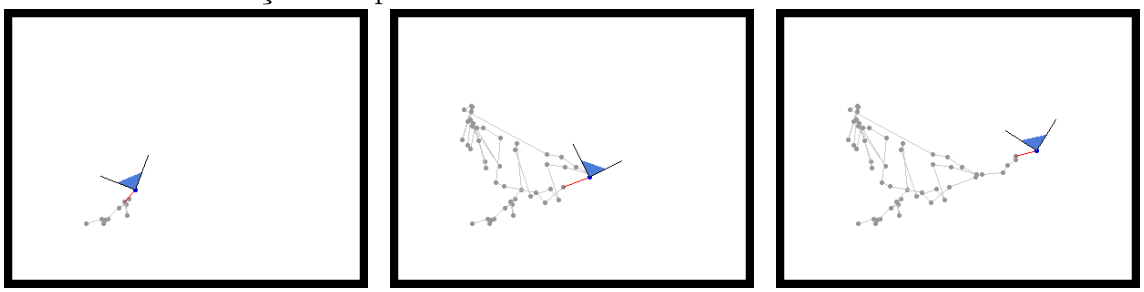
4.2.2 Trajetória

Os mapas finais dos quatro experimentos diferem entre si, pois foram gerados utilizando diferentes trajetórias. A cada execução, pequenos detalhes iniciais como a pose inicial da câmera e a inicialização do mapa geram diferentes resultados nas decisões do planejador. Como não possuímos um *ground truth* da trajetória para comparação, será utilizado a trajetória em *keyframes* gerada pelo próprio ORB-SLAM durante as execuções. Esta trajetória apresenta apenas a pose dos *keyframes*

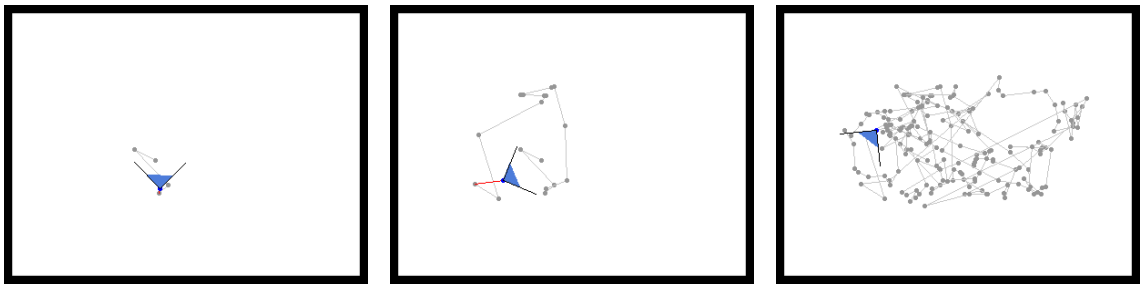
que compõem o mapa ao final da exploração, ordenados pela sua ordem de inserção. Não é armazenado o deslocamento realizado pela câmera entre a obtenção destes *keyframes*.

As Figuras 4.5 e 4.6 apresentam uma comparação entre os experimentos. Cada linha destas figuras corresponde a um experimento e cada coluna a um instante de tempo. Assim, é possível observar as diferenças nas trajetórias ao longo da exploração. Os experimentos *Lab-R1* e *Lab-R2*, apresentados na Figura 4.5, iniciam em poses similares, porém, após 20 segundos de exploração, já encontram-se em situações distintas. O mesmo comportamento é observado nos experimentos *Lab-R3* e *Lab-R4*, apresentados na Figura 4.6.

Figura 4.5: Trajetória dos *keyframes* nos experimentos *Lab-R1* e *Lab-R2*. Cada linha corresponde a um experimento e cada coluna a um instante de tempo. Os pontos são os *keyframe* já visitados. As linhas são apenas um guia para a sequência de passos, não representando o real deslocamento realizado. Em azul o *keyframe* mais atual e a direção em que ele olha.



(a) *Lab-R1* aos 20 segundos. (b) *Lab-R1* aos 90 segundos. (c) *Lab-R1* ao final.

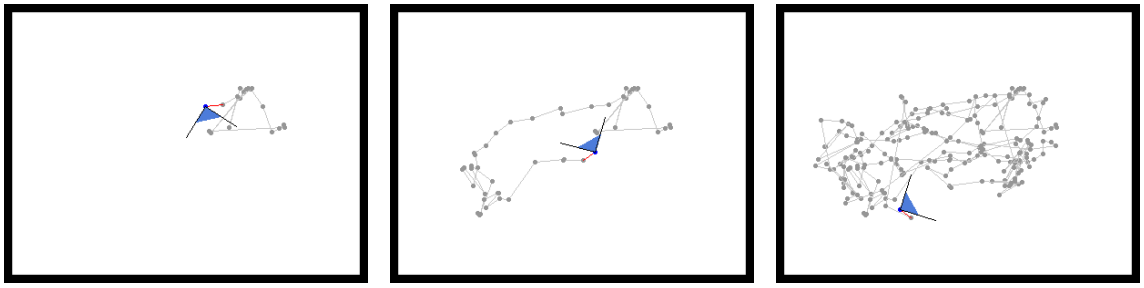


(d) *Lab-R2* aos 20 segundos. (e) *Lab-R2* aos 90 segundos. (f) *Lab-R2* ao final.

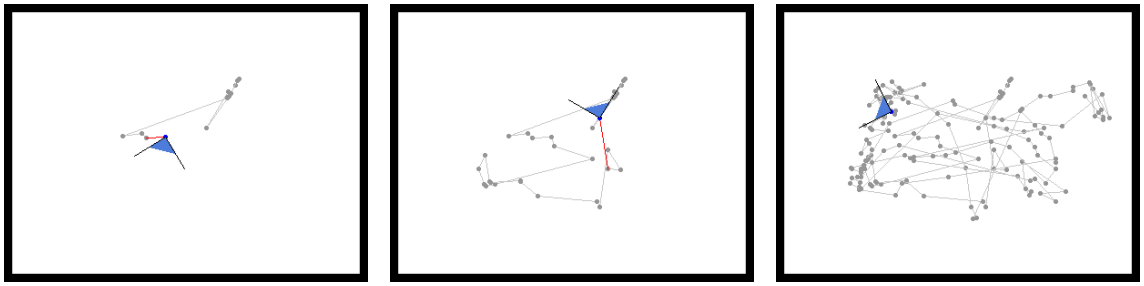
4.2.3 Critério de parada

Um ponto importante para a exploração é a capacidade de determinar quando o ambiente foi completamente explorado, para então encerrar o processo. Para o sistema de exploração proposto, o critério de parada está relacionado com a quantidade de *keyframes* abertos, sendo o encerramento determinado quando todos *keyframes*

Figura 4.6: Trajetória dos *keyframes* nos experimentos *Lab-R3* e *Lab-R4*. Cada linha corresponde a um experimento e cada coluna a um instante de tempo. Os pontos são os *keyframe* já visitados. As linhas são apenas um guia para a sequência de passos, não representando o real deslocamento realizado. Em azul o *keyframe* mais atual e a direção em que ele olha.



(a) *Lab-R3* aos 20 segundos. (b) *Lab-R3* aos 90 segundos. (c) *Lab-R3* ao final.



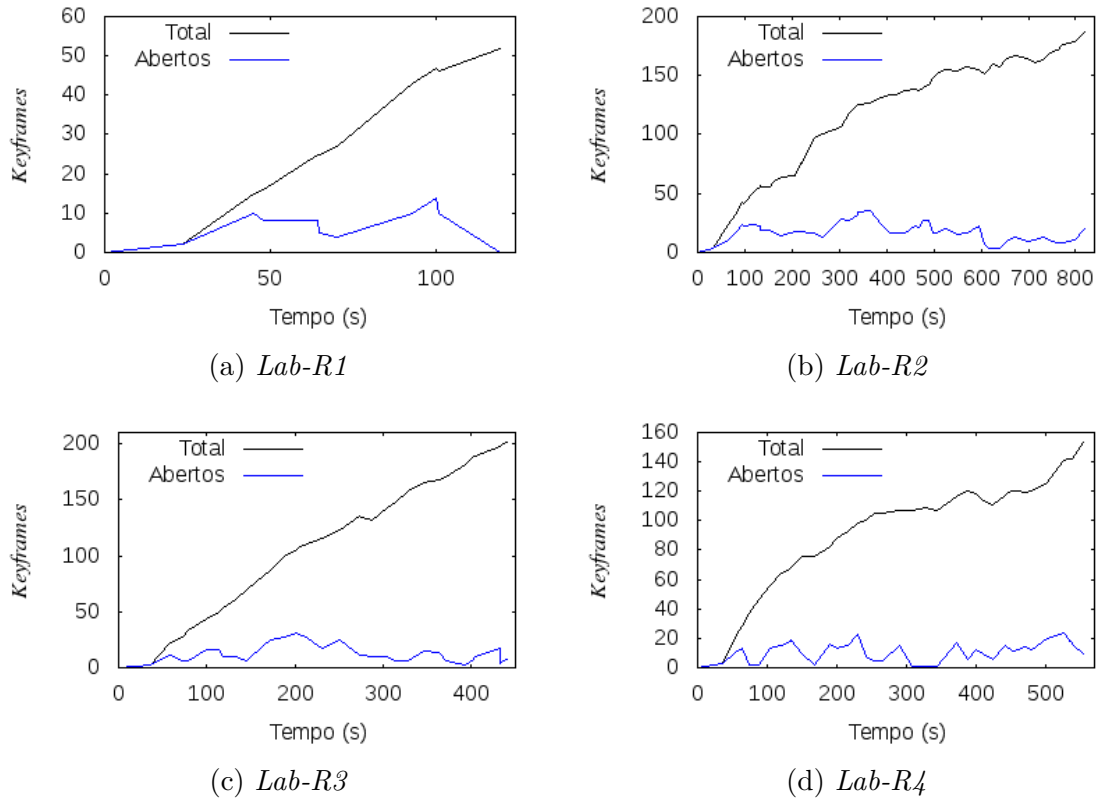
(d) *Lab-R4* aos 20 segundos. (e) *Lab-R4* aos 90 segundos. (f) *Lab-R4* ao final.

estiverem fechados. Para que um *keyframe* seja considerado fechado, é necessário que, no mínimo, 80% dos pontos que ele observa sejam visitados, o que acontece quando a câmera se aproxima destes pontos. Se esta porcentagem não for atingida, o *keyframe* é dito aberto e ainda é necessário executar o sistema de exploração. Esta margem busca lidar com o ruído presente no mapa, conforme observado na Figura 4.4, permitindo que o processo encerre mesmo que alguns pontos não sejam visitados, o que acontece com os pontos *outliers* que são inseridos do lado externo das paredes. Porém, a margem não deve ser tão grande que encerre o processo antes do ambiente estar completamente explorado.

A Figura 4.7 apresenta a quantidade total de *keyframes* e quantos estão abertos. Na Figura 4.7(a), é mostrado o experimento *Lab-R1*, que foi o único experimento a finalizar através do critério de parada. O experimento *Lab-R2*, apresentado na Figura 4.7(b), chegou próximo de encerrar, tendo apenas quatro *keyframes* abertos do instante 617 a 638 segundos. Na Figura 4.7(c), observa-se o experimento *Lab-R3*, que chegou a ter apenas seis *keyframes* abertos entre 314 e 328 segundos. Este experimento novamente chegou próximo do final aos 392 segundos, possuindo apenas três *keyframes* abertos. Por fim, na Figura 4.7(d), observa-se o experimento *Lab-R4*,

que chegou a ter apenas um *keyframe* aberto entre 307 e 344 segundos.

Figura 4.7: Evolução na quantidade de *keyframes* totais e abertos.



(a) *Lab-R1* (b) *Lab-R2*

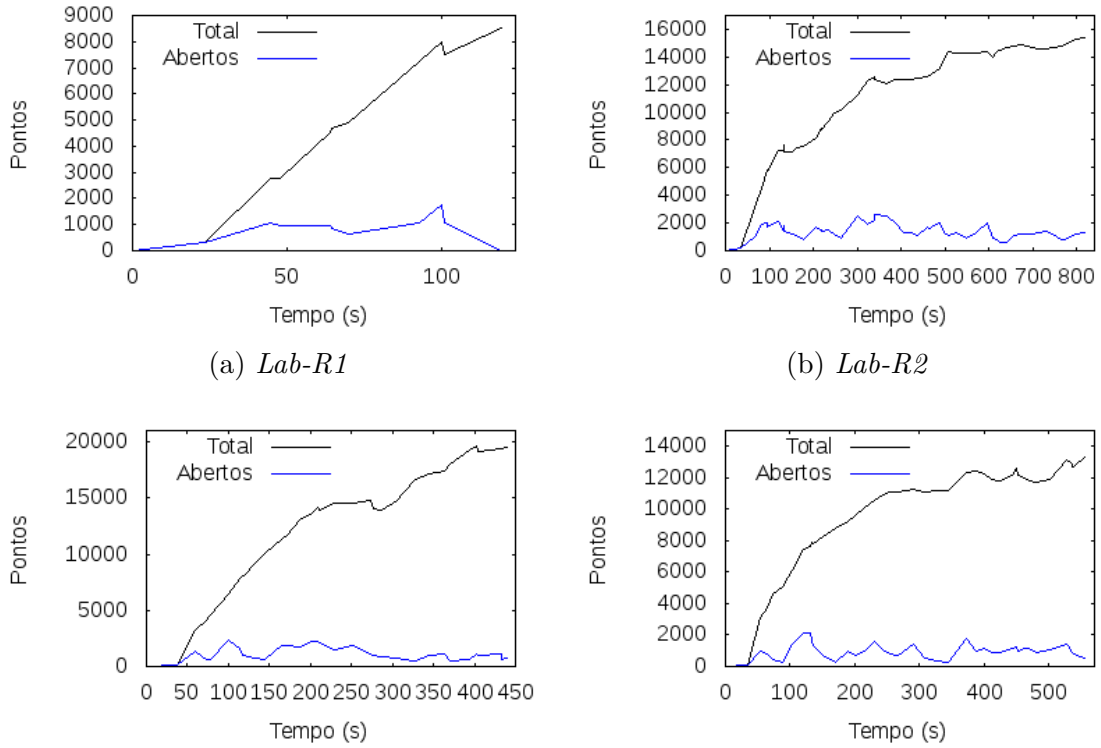
(c) *Lab-R3* (d) *Lab-R4*

Os testes *Lab-R2*, *Lab-R3* e *Lab-R4* aproximaram-se do necessário para o critério de parada decidir encerrar a exploração, porém, seguiram adiante pois novos pontos surgiram. Na sequência, o ORB-SLAM acabou falhando e impedindo que os novos pontos fossem visitados e a exploração encerrasse. Em um primeiro momento, a ideia de aumentar o percentual mínimo de pontos não visitados para definir um *keyframe* como aberto pode parecer interessante. Porém, observa-se que no experimento *Lab-R4*, aos 74 e 168 segundos, apenas dois *keyframes* estavam abertos, quase encerrando, precocemente, o processo de exploração. Isso demonstra que uma margem maior pode ocasionar o encerramento da exploração antes do ambiente estar completamente explorado.

É importante destacar que o critério de parada proposto utiliza o percentual de pontos abertos sobre os pontos que cada *keyframe* observa e não sobre o total de pontos do mapa global. Isso justifica-se pois se fosse utilizado um percentual sobre o total, assim que o mapa crescesse a exploração acabaria sendo encerrada pela quantidade de pontos fechados, ignorando o fato de existir uma região com uma concentração de pontos abertos. A Figura 4.8 apresenta a quantidade total

de pontos, assim como quantos encontram-se abertos, ao longo dos experimentos. Observa-se, em todos os casos, que a quantidade de pontos abertos mantém-se baixa mesmo com o aumento na quantidade total ao longo da exploração.

Figura 4.8: Evolução na quantidade de pontos totais e abertos (i.e. não visitados).



(a) *Lab-R1*

(b) *Lab-R2*

(c) *Lab-R3*

(d) *Lab-R4*

4.2.4 Tempo de processamento

A abordagem proposta utiliza mapas locais visando reduzir o tempo de processamento, permitindo que a aplicação seja executada *online*. As tarefas mais complexas dessa abordagem são: gerar o mapa local, planejar a próxima ação e definir o caminho no mapa local. A Tabela 4.2 apresenta o tempo de processamento médio destas três tarefas nos quatro experimentos. É possível observar que os experimentos obtiveram resultados similares. O experimento *Lab-R1* apresenta a menor média tanto para geração do mapa local, quanto para o planejamento. Nele, devido a inicialização, cada *voxel* acabou representando um volume maior do ambiente, o que simplificou parte das tarefas. Além disso, é o experimento com menor duração, sendo completado com 124 segundos, enquanto que *Lab-R2*, *Lab-R3* e *Lab-R4* encerraram aos 821, 442 e 558 segundos, respectivamente.

Tabela 4.2: Tempo de processamento das tarefas mais complexas.

Experimento	Tempo em Milissegundos					
	Mapa Local		Planejador		Caminho	
	Média	D. P.	Média	D. P.	Média	D. P.
Lab-R1	97	32	145	56	67	13
Lab-R2	125	33	197	43	64	15
Lab-R3	131	41	202	47	60	13
Lab-R4	137	44	208	57	69	22

O planejamento no experimento *Lab-R4* levou, em média, 208 ± 57 milissegundos para decidir a próxima ação, sendo esta a maior média dentre os experimentos. É importante ressaltar que o tempo necessário para gerar o caminho encontra-se incluído no cálculo do tempo de planejamento, pois gerar caminhos é uma subtarefa solicitada pelo planejador. Seria necessário, então, menos de meio segundo para gerar o mapa local e planejar a próxima ação. Isto permite que o robô, assim que complete o objetivo atual, defina rapidamente o próximo objetivo, evitando ficar parado aguardando o processamento.

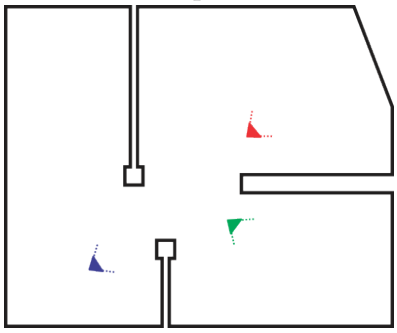
4.3 Exploração em ambiente com múltiplas salas

Para analisar a capacidade do método de explorar ambientes mais desafiadores, foram realizados experimentos em um ambiente com múltiplas salas. Nestes, as imagens foram capturadas utilizando a mesma câmera modelo Logitech C270 e o processamento foi realizado em um notebook Dell Vostro 3550 equipado com 4GB de memória RAM e um processador Intel Core i5-2410M 2.30GHz. A câmera foi movimentada por uma pessoa, seguindo o mesmo procedimento apresentado na Seção 4.2.

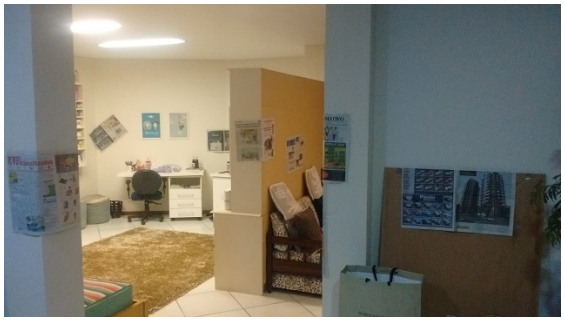
O ambiente foi preparado inserindo diversas marcações nas regiões sem saliência, a Figura 4.9 apresenta o mapa e fotos do ambiente explorado. Foram realizadas três rodadas de exploração no ambiente, denominadas: *Multi-R1*, *Multi-R2* e *Multi-R3*[4]. Durante os três experimentos o método guiou o movimento da câmera pelas diferentes sala, mas não completou a exploração adequadamente de acordo com o critério de parada. Nos três casos, o processo foi interrompido após o ORB-SLAM não conseguir estimar corretamente o movimento do robô, ou criar novos pontos do mapa. Por isso, não foi possível analisar a eficácia do critério de parada.

[4]Vídeos: *Multi-R1*, *Multi-R2* e *Multi-R3*.

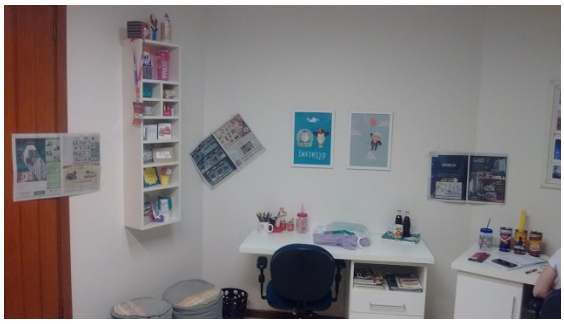
Figura 4.9: Planta baixa do ambiente em (a). Fotos do ambiente em (b), (c) e (d), obtidas através das poses destacadas na planta baixa.



(a) Planta baixa do ambiente.



(b) Foto obtida na pose marcada em (a) na cor azul.



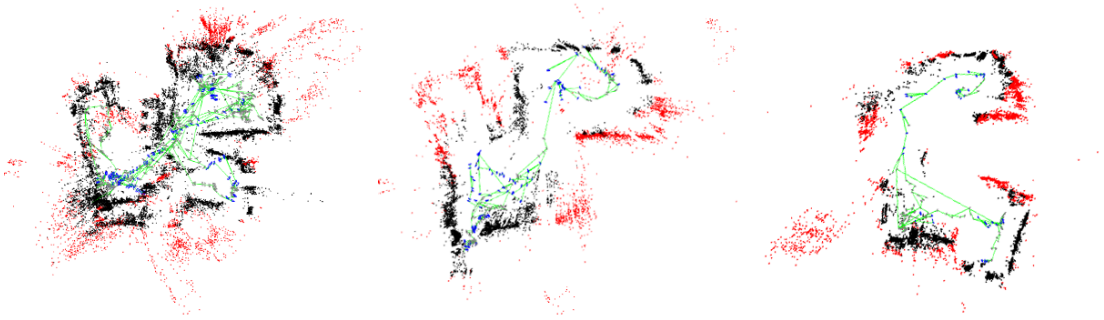
(c) Foto obtida na pose marcada em (a) na cor vermelha.

(d) Foto obtida na pose marcada em (a) na cor verde.

Durante toda execução, os experimentos foram comprometidos pelo desempenho abaixo do esperado do ORB-SLAM. Este desempenho é diretamente influenciado pela qualidade de câmera, uma vez que é necessário extrair e casar *features* do ambiente através das imagens. No decorrer da exploração, em diversos momentos, o SLAM não foi capaz de casar corretamente as *features*, consequentemente não incluindo novos pontos no mapa, acarretando na perda da localização. Devido a estes problemas, os mapas estimados, apresentados na Figura 4.10, também tiveram sua qualidade comprometida. Os três mapas ficaram bastante deformados não representando corretamente o ambiente. Ainda, no caso dos experimentos *Multi-R2* e *Multi-R3*, os mapas estão incompletos pois, o ORB-SLAM falhou antes de todas as salas serem exploradas.

Apesar dos mapas ruins, a exploração mostrou-se capaz de navegar entre as salas do ambiente. A Figura 4.11 apresenta a trajetória do experimento *Multi-R1*. O desenho desta trajetória foi reconstruído manualmente, uma vez que não possuímos o *ground truth*. Observa-se, na Figura 4.11(a) que a exploração iniciou navegando pela sala superior direita, e depois dirigiu-se a sala inferior direita. Na Figura 4.11(b), a segunda sala foi explorada e a câmera retornou para a primeira

Figura 4.10: Mapas gerados no ambiente com múltiplas salas. Em preto os pontos visitados, em vermelho os pontos não visitados, em azul e cinza os *keyframes* abertos e fechados, respectivamente. Em verde, a conexão entre os *keyframes* e seus melhores vizinhos.

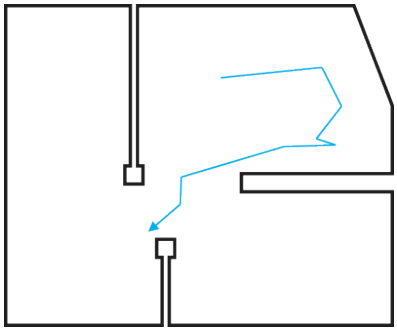


(a) *Multi-R1* (b) *Multi-R2* (c) *Multi-R3*

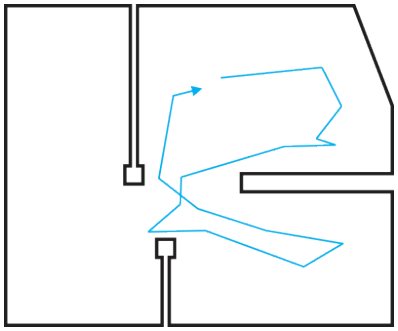
sala. Na Figura 4.11(c) a sala restante, à esquerda, está sendo explorada. Por fim, na Figura 4.11(d) observa-se a câmera retornando para a sala inicial, após explorar as três salas do ambiente. Este experimentos demonstram que, apesar da baixa qualidade na informação gerada pelo ORB-SLAM, a exploração permanece capaz de navegar por todo ambiente. Observa-se no mapa final, apresentado em na Figura 4.10(a), que uma grande quantidade de pontos não visitados (em vermelho) são ruído e estão em pontos inalcançáveis do ambiente, o que provavelmente impediria a exploração de finalizar corretamente.

Uma amostra das trajetórias nos experimentos *Multi-R2* e *Multi-R3* é apresentada na Figura 4.12. É possível observar, novamente, que a trajetória entre os experimentos varia, devido a pose inicial e a inicialização do mapa. Na Figura 4.12(a) e Figura 4.12(c) observa-se a trajetória após cinco minutos de exploração de *Multi-R2* e *Multi-R3*, respectivamente. Enquanto *Multi-R2* ainda explorava a primeira sala, *Multi-R3* já encontrava-se na segunda sala. Na Figura 4.12(b) e Figura 4.12(d) observa-se a trajetória após dez minutos, com cada experimento explorando uma sala diferente. Estes experimentos foram encerrados antes de explorarem todas as salas do ambiente, devido ao desempenho baixo do SLAM, porém, é possível observar que eles estavam navegando entre estas.

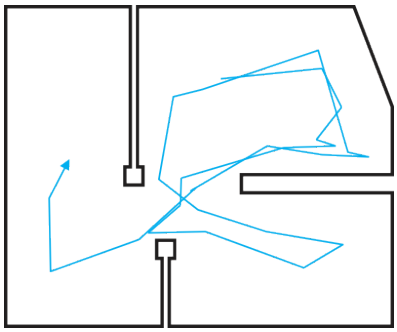
Figura 4.11: Trajetória, em azul, do experimento *Multi-R1* em diferentes instantes de tempo. A seta não representa a direção em que a câmera está olhando e deve ser utilizada apenas para compreender a sequência de passos.



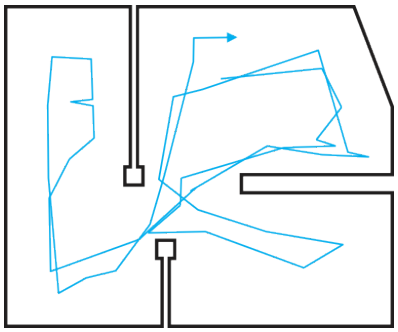
(a) *Multi-R1* aos 5 minutos.



(b) *Multi-R1* aos 10 minutos.

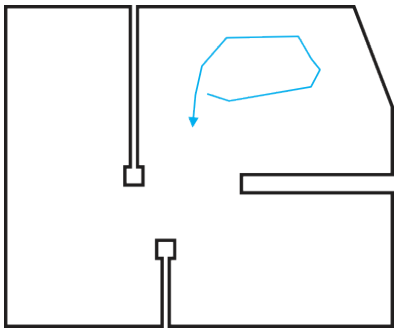


(c) *Multi-R1* aos 15 minutos.

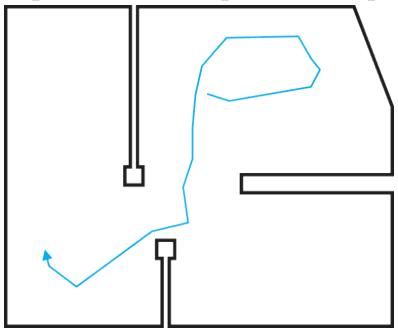


(d) *Multi-R1* aos 20 minutos.

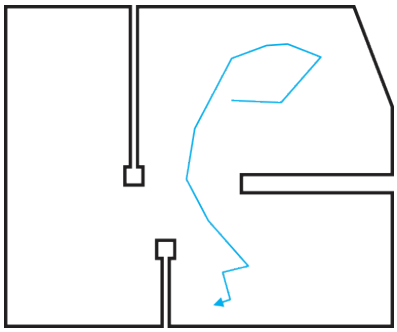
Figura 4.12: Trajetória, em azul, do experimento *Multi-R2*, em (a) e (b), e do experimento *Multi-R3*, em (c) e (d). A seta não representa a direção em que a câmera está olhando e deve ser utilizada para compreender a sequência de passos.



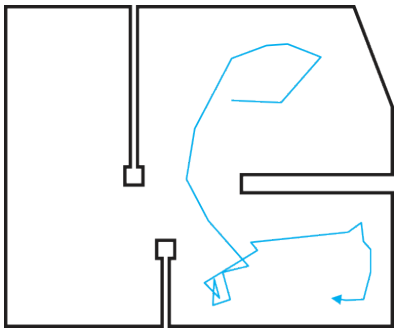
(a) *Multi-R2* aos 5 minutos.



(b) *Multi-R2* aos 10 minutos.



(c) *Multi-R3* aos 5 minutos.



(d) *Multi-R3* aos 10 minutos.

5 CONCLUSÃO

Neste trabalho, foi proposta uma estratégia de exploração monocular 3D utilizando SLAM monocular. Estes métodos de SLAM geram mapas esparsos ou semi-densos, geralmente representados por nuvens de pontos, que não são adequadas para exploração e navegação. Para lidar com estes mapas, propõe-se a geração de mapas volumétricos locais a partir da nuvem de pontos, utilizando as linhas de visão para estimar a região livre. Sobre este mapa local são definidos os objetivos da exploração. Caso não exista um objetivo local, um objetivo global é definido. A proposta foi executada por meio de testes em um ambiente simples, com apenas uma sala, e um ambiente mais complexo, composto por diversas salas.

Inicialmente, testou-se a necessidade de utilizar apenas parte dos *keyframes* para gerar o mapa local. Observou-se que a utilização de todos os *keyframes* impede o sistema de escalar juntamente com o mapa, pois quanto maior o mapa global, mais tempo é necessário para o processamento. Assim, definiu-se que utilizar apenas um conjunto reduzido de *keyframes* permite que o sistema opere também em ambientes grandes e mais estruturados sem consumir muito tempo de processamento.

Em seguida, quatro experimentos com o sistema de exploração completo foram realizados em um ambiente simples (i.e. apenas uma sala). De início, analisou-se os mapas gerados, observando que apesar de deformações e da presença de ruído, estes representavam bem o ambiente. Então, analisou-se as trajetórias de cada experimento, observando que devido a variação na pose inicial e na inicialização do mapa, cada experimento desloca-se pelo ambiente de maneira diferente, gerando assim, mapas diferentes. Em seguida, foi analisado o critério de parada proposto. Apenas em um experimento este efetivamente encerrou a exploração, sendo que nos outros três foi necessário encerrar o sistema por falha do ORB-SLAM. Porém, foi possível observar que o critério é capaz de determinar que a exploração está completa, dado os critérios definidos, desde que o SLAM funcione por tempo suficiente. Também, observou-se que utilizar um percentual de pontos não visitados menos rígido pode acarretar no encerramento precoce da exploração. Por fim, o tempo necessário para processar as tarefas complexas foi analisado, concluindo que, dado o *hardware* utilizado, é possível realizar todo ciclo de tomada de decisão em menos de meio segundo, evitando que o robô fique parado aguardando.

Um último experimento foi realizado para observar o comportamento da es-

tratégia de exploração em um ambiente com múltiplas salas. Através das trajetórias obtidas nas três rodadas de exploração deste experimento, observa-se que a exploração é capaz de navegar entre as salas do ambiente, levando o robô a mapeá-lo por completo. Infelizmente, os resultados do ORB-SLAM foram abaixo do esperado, não sendo possível realizar uma análise sobre o critério de parada.

De maneira geral, os resultados obtidos foram satisfatórios, demonstrando que a estratégia consegue deslocar o robô pelo ambiente em segurança, descobrindo novas regiões a serem exploradas. Um problema nos experimentos foi a qualidade do ORB-SLAM, que está diretamente associada com a qualidade das imagens capturadas. A baixa qualidade nos mapas gerados prejudicou o processo de exploração, impedindo-o de prosseguir até encerrar. Além disso, em diversos momentos a dificuldade do ORB-SLAM em mapear determinadas regiões fez com que ele perdesse a estimativa sobre a pose do robô.

Ao analisar os trabalhos relacionados, que já foram discutidos na Seção 2.6, é possível observar algumas vantagens da abordagem proposta nesta dissertação. No trabalho de Stumberg et al. (2017), é necessário gerar um mapa volumétrico global, através de um processo que demora e faz o VANT ficar pairando no ar até que mapa seja gerado. Isto não é necessário em nossa abordagem, devido a utilização de mapas locais. Já o trabalho de Mostegel, Wendel e Bischof (2014) propõe uma forma interessante de evitar que o VANT perca a estimativa de sua pose. Porém, esta não é realmente uma exploração, pois é necessário pré-definir um objetivo global. Nossa proposta não necessita de objetivos pré-definidos e determina como continuar explorando a cada iteração. Por fim, o trabalho de Palazzolo e Stachniss (2017) assume a existência de uma *bounding box* livre para navegar ao redor do objeto a ser mapeado. Além disso, define poses para que o robô sempre olhe para o centro desta *bounding box*. Isso impede a exploração de ser executada em ambiente internos e complexos. Nossa abordagem, por outra lado, permite explorar qualquer tipo de ambiente, estimando em tempo de execução as áreas livres para navegar.

Apesar de termos uma prova de conceito para a estratégia de exploração proposta, existem pontos em abertos e possibilidades de melhora. O impacto de cada um dos parâmetros do sistema, apresentados na Tabela 4.1, não foi avaliado individualmente e uma configuração paramétrica diferente pode obter melhores resultados. Outro ponto que deve ser analisado em detalhes é o cálculo da atratividade de um *voxel* utilizando uma vizinhança. É possível que este não proporcione um

ganho que compense seu custo. Se este for o caso, o cálculo pode ser simplificado considerando apenas o próprio *voxel*.

Uma questão importante a ser desenvolvida é a inicialização da escala. Neste trabalho, um valor de escala foi obtido empiricamente. Porém, é necessário utilizar alguma técnica para estimá-lo. No entanto, para tal, é necessário utilizar algum outro sensor além da câmera. Além desta questão, outro ponto não abordado é como lidar com uma situação em que o mapa local seja pequeno de mais para englobar um *keyframe* e alguns de seus pontos abertos. Apesar de pouco provável, neste caso, não seria possível gerar um caminho para estes pontos. Na proposta atual, o sistema não possui ferramentas para lidar com essa situação.

Apesar de simples e funcional, a definição da direção para qual o robô deve olhar, apresentado na Seção 3.5, mostrou-se um pouco limitada. Em diversos momentos dos experimento, a câmera acabou olhando em direção a uma parede muito próxima a ela, limitando seu campo de visão. É necessário um estudo de outras formas de definir esta direção, talvez baseando-se em ideias como as do trabalho de Palazzolo e Stachniss (2017), que utiliza uma função de ganho. Seria possível utilizar uma função de ganho para olhar na direção com mais espaço livre, ou algo do gênero, evitando focar em objetos próximos. Outra desvantagem observada na abordagem proposta para definir a orientação é associar, inicialmente, a direção em que a câmera deve olhar com cada passo do caminho. Nos momentos em que o SLAM não é capaz de gerar novos pontos do mapa, continuar mudando a orientação pode causar a perda da estimativa da pose. Uma solução seria analisar a qualidade de localização na imagem, como proposto por Mostegel, Wendel e Bischof (2014). Porém, isso torna o método mais acoplado ao SLAM escolhido.

A câmera foi movida por um operador humano nos testes realizados. É necessário testar essa abordagem utilizando um robô, preferencialmente um VANT, pela sua capacidade de movimentação. Para tal, as correções acima citadas devem ser realizadas, permitindo que o mesmo voe com mais segurança. É interessante, também, realizar alguns testes utilizando outro SLAM monocular, como o PTAM ou o LSD-SLAM.

REFERÊNCIAS

ADLER, B.; XIAO, J.; ZHANG, J. Autonomous exploration of urban environments using unmanned aerial vehicles. **Journal of Field Robotics**, Wiley Online Library, v. 31, n. 6, p. 912–939, 2014.

AMIGONI, F. Experimental evaluation of some exploration strategies for mobile robots. Em: IEEE. **Robotics and Automation (ICRA), IEEE International Conference on**. [S.l.], 2008. p. 2818–2823.

BHAT, S.; MEENAKSHI, M. Vision based robotic system for military applications–design and real time validation. Em: IEEE. **Signal and Image Processing (ICSIP), Fifth International Conference on**. [S.l.], 2014. p. 20–25.

CADENA, C. et al. Simultaneous localization and mapping: Present, future, and the robust-perception age. **CoRR, vol. abs/1606.05830**, 2016.

CHUDOBA, J. et al. Exploration and mapping technique suited for visual-features based localization of mavs. **Journal of Intelligent & Robotic Systems**, Springer, v. 84, n. 1-4, p. 351–369, 2016.

COWLEY, A.; TAYLOR, C. J.; SOUTHALL, B. Rapid multi-robot exploration with topometric maps. Em: IEEE. **Robotics and Automation (ICRA), IEEE International Conference on**. [S.l.], 2011. p. 1044–1049.

DANIEL, K. et al. Theta*: Any-angle path planning on grids. **Journal of Artificial Intelligence Research**, v. 39, p. 533–579, 2010.

DUNN, E.; FRAHM, J.-M. Next best view planning for active model improvement. Em: **BMVC**. [S.l.: s.n.], 2009. p. 1–11.

ENGEL, J.; SCHÖPS, T.; CREMERS, D. Lsd-slam: Large-scale direct monocular slam. Em: SPRINGER. **European Conference on Computer Vision**. [S.l.], 2014. p. 834–849.

ENGEL, J.; STURM, J.; CREMERS, D. Scale-aware navigation of a low-cost quadrocopter with a monocular camera. **Robotics and Autonomous Systems**, Elsevier, v. 62, n. 11, p. 1646–1656, 2014.

FRAUNDORFER, F. et al. Vision-based autonomous mapping and exploration using a quadrotor mav. Em: IEEE. **Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on**. [S.l.], 2012. p. 4557–4564.

GÁLVEZ-LÓPEZ, D.; TARDOS, J. D. Bags of binary words for fast place recognition in image sequences. **IEEE Transactions on Robotics**, IEEE, v. 28, n. 5, p. 1188–1197, 2012.

GEIGER, A.; LENZ, P.; URTASUN, R. Are we ready for autonomous driving? the kitti vision benchmark suite. Em: IEEE. **Computer Vision and Pattern Recognition (CVPR), IEEE Conference on**. [S.l.], 2012. p. 3354–3361.

HART, P. E.; NILSSON, N. J.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. **IEEE transactions on Systems Science and Cybernetics**, IEEE, v. 4, n. 2, p. 100–107, 1968.

HASAN, K. M.; REZA, K. J. et al. Path planning algorithm development for autonomous vacuum cleaner robots. Em: IEEE. **Informatics, Electronics & Vision (ICIEV), International Conference on**. [S.l.], 2014. p. 1–6.

HENG, L. et al. Efficient visual exploration and coverage with a micro aerial vehicle in unknown environments. Em: IEEE. **Robotics and Automation (ICRA), IEEE International Conference on**. [S.l.], 2015. p. 1071–1078.

HENG, L. et al. Autonomous visual mapping and exploration with a micro aerial vehicle. **Journal of Field Robotics**, Wiley Online Library, v. 31, n. 4, p. 654–675, 2014.

HOPPE, C. et al. Photogrammetric camera network design for micro aerial vehicles. Em: **Computer vision winter workshop (CVWW)**. [S.l.: s.n.], 2012. v. 8, p. 1–3.

JO, K. et al. Development of autonomous car—part ii: A case study on the implementation of an autonomous driving system based on distributed architecture. **IEEE Transactions on Industrial Electronics**, IEEE, v. 62, n. 8, p. 5119–5132, 2015.

KLEIN, G.; MURRAY, D. Parallel tracking and mapping for small ar workspaces. Em: IEEE. **Mixed and Augmented Reality (ISMAR), 6th IEEE/ACM International Symposium on**. [S.l.], 2007. p. 225–234.

KOENIG, S.; LIKHACHEV, M. Dˆ* lite. **AAAI/IAAI**, v. 28, 2002.

KOENIG, S.; LIKHACHEV, M. Incremental a*. Em: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2002. p. 1539–1546.

KORF, R. E. Real-time heuristic search. **Artificial intelligence**, Elsevier, v. 42, n. 2-3, p. 189–211, 1990.

KUMAR, V.; MICHAEL, N. Opportunities and challenges with autonomous micro aerial vehicles. Em: **Robotics Research**. [S.l.]: Springer, 2017. p. 41–58.

LEWIS, M. A. Visual navigation in a robot using zig-zag behavior. Em: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 1998. p. 822–828.

MADHAVAN, R. et al. The 2016 humanitarian robotics and automation technology challenge [competitions]. **IEEE Robotics & Automation Magazine**, IEEE, v. 24, p. 127–129, 2017.

MADHAVAN, R. et al. 2015 humanitarian robotics and automation technology challenge [humanitarian technology]. **IEEE Robotics & Automation Magazine**, IEEE, v. 22, n. 3, p. 182–184, 2015.

MAXWELL, P.; LARKIN, D.; LOWRANCE, C. Turning remote-controlled military systems into autonomous force multipliers. **IEEE Potentials**, IEEE, v. 32, n. 6, p. 39–43, 2013.

MEI, Y. et al. Energy-efficient mobile robot exploration. Em: IEEE. **Robotics and Automation (ICRA), IEEE International Conference on**. [S.l.], 2006. p. 505–511.

MENG, Z. et al. A two-stage optimized next-view planning framework for 3-d unknown environment exploration, and structural reconstruction. **IEEE Robotics and Automation Letters**, IEEE, v. 2, n. 3, p. 1680–1687, 2017.

MOBARHANI, A. et al. Histogram based frontier exploration. Em: IEEE. **Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on**. [S.l.], 2011. p. 1128–1133.

MOSTEGEL, C. **Active Monocular Localization: Towards Autonomous Monocular Exploration for Quadrotor MAVs**. Tese (PhD) — Graz University of Technology, 2013.

MOSTEGEL, C.; WENDEL, A.; BISCHOF, H. Active monocular localization: Towards autonomous monocular exploration for multirotor mavs. Em: IEEE. **Robotics and Automation (ICRA), IEEE International Conference on**. [S.l.], 2014. p. 3848–3855.

MUR-ARTAL, R.; MONTIEL, J. M. M.; TARDOS, J. D. Orb-slam: a versatile and accurate monocular slam system. **IEEE Transactions on Robotics**, IEEE, v. 31, n. 5, p. 1147–1163, 2015.

NASKAR, S. et al. Application of radio frequency controlled intelligent military robot in defense. Em: IEEE. **Communication Systems and Network Technologies (CSNT), International Conference on**. [S.l.], 2011. p. 396–401.

NELSON, R. C.; ALOIMONOS, J. Obstacle avoidance using flow field divergence. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 11, n. 10, p. 1102–1106, 1989.

NUSKE, S. et al. Autonomous exploration and motion planning for an unmanned aerial vehicle navigating rivers. **Journal of Field Robotics**, Wiley Online Library, v. 32, n. 8, p. 1141–1162, 2015.

OKEREAFOR, D. et al. Improving security and emergency response through the use of unmanned vehicles. Em: IEEE. **Emerging & Sustainable Technologies for Power & ICT in a Developing Society (NIGERCON), IEEE International Conference on**. [S.l.], 2013. p. 263–269.

OSSWALD, S. et al. Speeding-up robot exploration by exploiting background information. **IEEE Robotics and Automation Letters**, IEEE, v. 1, n. 2, p. 716–723, 2016.

PALAZZOLO, E.; STACHNISS, C. Information-driven autonomous exploration for a vision-based mav. **ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences**, p. 59–66, 2017.

PETTERSSON, I.; KARLSSON, I. M. Setting the stage for autonomous cars: a pilot study of future autonomous driving experiences. **IET Intelligent Transport Systems**, IET, v. 9, n. 7, p. 694–701, 2015.

PRESTES, E. et al. The 2016 humanitarian robotics and automation technology challenge [competitions]. **IEEE Robotics & Automation Magazine**, IEEE, v. 23, n. 3, p. 23–24, 2016.

QUIN, P. et al. Efficient neighbourhood-based information gain approach for exploration of complex 3d environments. Em: IEEE. **Robotics and Automation (ICRA), IEEE International Conference on**. [S.l.], 2013. p. 1343–1348.

ROGGEMAN, H. et al. Autonomous exploration with prediction of the quality of vision-based localization. **IFAC-PapersOnLine**, Elsevier, v. 50, n. 1, p. 10274–10279, 2017.

ROSS, P. E. Robot, you can drive my car. **IEEE Spectrum**, IEEE, v. 51, n. 6, p. 60–90, 2014.

RUBLEE, E. et al. Orb: An efficient alternative to sift or surf. Em: IEEE. **Computer Vision (ICCV), IEEE international conference on**. [S.l.], 2011. p. 2564–2571.

SANTOSH, D.; ACHAR, S.; JAWAHAR, C. Autonomous image-based exploration for mobile robot navigation. Em: IEEE. **Robotics and Automation (ICRA), IEEE International Conference on**. [S.l.], 2008. p. 2717–2722.

SEFER, E.; KUTER, U.; NAU, D. Real-time a* search with depth-k lookahead. Em: **Proceedings of the International Symposium on Combinatorial Search**. [S.l.: s.n.], 2009.

SENARATHNE, P.; WANG, D. Towards autonomous 3d exploration using surface frontiers. Em: IEEE. **Safety, Security, and Rescue Robotics (SSRR), IEEE International Symposium on**. [S.l.], 2016. p. 34–41.

SHADE, R.; NEWMAN, P. Choosing where to go: Complete 3d exploration with stereo. Em: IEEE. **Robotics and Automation (ICRA), IEEE International Conference on**. [S.l.], 2011. p. 2806–2811.

SHEN, S. **Autonomous navigation in complex indoor and outdoor environments with micro aerial vehicles**. [S.l.]: University of Pennsylvania, 2014.

SHEN, S.; MICHAEL, N.; KUMAR, V. Autonomous indoor 3d exploration with a micro-aerial vehicle. Em: IEEE. **Robotics and Automation (ICRA), IEEE International Conference on**. [S.l.], 2012. p. 9–15.

SHIM, I. et al. An autonomous driving system for unknown environments using a unified map. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, v. 16, n. 4, p. 1999–2013, 2015.

SIEGWART, R.; NOURBAKHSH, I. R.; SCARAMUZZA, D. **Introduction to autonomous mobile robots**. [S.l.]: MIT press, 2011.

SOBEY, P. Active navigation with a monocular robot. **Biological Cybernetics**, Springer, v. 71, n. 5, p. 433–440, 1994.

SONG, S.; JO, S. Online inspection path planning for autonomous 3d modeling using a micro-aerial vehicle. Em: IEEE. **Robotics and Automation (ICRA), IEEE International Conference on**. [S.l.], 2017. p. 6217–6224.

STACHNISS, C. **Exploration and mapping with mobile robots.** Tese (PhD) — University of Freiburg, 2006.

STACHNISS, C.; MOZOS, O. M.; BURGARD, W. Efficient exploration of unknown indoor environments using a team of mobile robots. **Annals of Mathematics and Artificial Intelligence**, Springer, v. 52, n. 2-4, p. 205–227, 2008.

STRÖM, D. P.; BOGOSLAVSKYI, I.; STACHNISS, C. Robust exploration and homing for autonomous robots. **Robotics and Autonomous Systems**, Elsevier, v. 90, p. 125–135, 2017.

STRÖM, D. P.; NENCI, F.; STACHNISS, C. Predictive exploration considering previously mapped environments. Em: IEEE. **Robotics and Automation (ICRA), IEEE International Conference on**. [S.l.], 2015. p. 2761–2766.

STUMBERG, L. von et al. From monocular slam to autonomous drone exploration. Em: IEEE. **Mobile Robots (ECMR), European Conference on**. [S.l.], 2017. p. 1–8.

TABIB, W. et al. Computationally efficient information-theoretic exploration of pits and caves. Em: IEEE. **Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on**. [S.l.], 2016. p. 3722–3727.

THRUN, S.; BURGARD, W.; FOX, D. **Probabilistic Robotics**. 1st. ed. Cambridge, MA: The MIT Press, 2005.

TRIGGS, B. et al. Bundle adjustment—a modern synthesis. Em: SPRINGER. **International workshop on vision algorithms**. [S.l.], 1999. p. 298–372.

VALLVÉ, J.; ANDRADE-CETTO, J. Dense entropy decrease estimation for mobile robot exploration. Em: IEEE. **Robotics and Automation (ICRA), IEEE International Conference on**. [S.l.], 2014. p. 6083–6089.

VINCENT, R. et al. Distributed multirobot exploration, mapping, and task allocation. **Annals of Mathematics and Artificial Intelligence**, Springer, v. 52, n. 2, p. 229–255, 2008.

WATT, A. H. **3D Computer Graphics**. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 1999.

YAMAUCHI, B. A frontier-based approach for autonomous exploration. Em: IEEE. **Computational Intelligence in Robotics and Automation (CIRA), IEEE International Symposium on**. [S.l.], 1997. p. 146–151.