UMA ABORDAGEM PARA MAPEAMENTO E LOCALIZAÇÃO SIMULTÂNEOS UTILIZANDO INFORMAÇÃO TOPOLÓGICA

RAFAEL GONÇALVES COLARES

UMA ABORDAGEM PARA MAPEAMENTO E LOCALIZAÇÃO SIMULTÂNEOS UTILIZANDO INFORMAÇÃO TOPOLÓGICA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

ORIENTADOR: LUIZ CHAIMOWICZ

Belo Horizonte

Maio de 2011

Colares, Rafael Gonçalves
C683a Uma Abordagem para mapeamento e localização simultâneos utilizando informação topológica / Rafael Gonçalves Colares. — Belo Horizonte, 2011
xxii, 50 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais
Orientador: Luiz Chaimowicz

1. Robótica. 2. Mapeamento. 3. Planejamento. 4. Exploração. I. Título.

CDU 519.6*82.9(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Uma abordagem para mapeamento e localização simultâneos utilizando informação topológica

RAFAEL GONÇALVES COLARES

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. LUIZ CHAIMOWICZ - Orientador
Departamento de Ciência da Computação - UFMG

PROF. DENIS FERNANDO WOLF
ICMC - Universidade de São Paulo

PROF. MARCOS AUGUSTO MENEZES VIEIRA
Departamento de Ciência da Computação - UFMG

PROF. MARIO FERNANDO MONTENEGRO CAMPOS
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 08 de junho de 2011.

Aos meus pais e irmão.

Agradecimentos

Aos meus pais, Elton e Ioni, pela educação que me deram e por todo carinho e apoio durante todo o processo da dissertação. Ao meu irmão, Marcos, pelo apoio, companheirismo e por todas as conversas descontraídas.

Ao professor e orientador Luiz Chaimowicz, pela presença, ajuda e suporte constantes durante o desenvolvimento da dissertação.

Aos meus tios, Xikuta, Maguinho, Cleuminha, Su e Patrícia, que me receberam em Belo Horizonte e que até hoje me ajudam a me sentir em casa. E aos meus primos, sempre me convidando para os mais diversos programas.

Aos amigos Lucas, Pinta e Nick, por sempre apoiarem as minhas decisões, me incentivarem a continuar, e por sempre estarem disponíveis quando eu precisei deles.

Aos amigos da firma: Aaron Hengles, André "Zangado" Martins, Eduardo "Leitoso" Chielle, Jonata Tyska, Rodrigo Remor, Thiago "Moicano" Goulart, Vinícius "Vinni" Rodrigues e "Zinedine" Zenon Oliveira, pelos e-mails, palavras de apoios e eternas discussões sobre futebol.

Aos amigos que fiz em Belo Horizonte, Rafael Santin, Tiago Cunha, Zilton, Leandro, Thiago Pé-de-Pano e Dudu, pela amizade, companheirismo e por me mostrarem que posso sempre contar com eles.

Ao pessoal do Rebolation, Anderson, Rafael Vieira, Mano, Fillipe e Marcelo, pelos jogos e manhãs de sábado sempre agradáveis!

Ao amigos do Verlab, Armando, Douglas, Wolmar, Samuel, Moises, Elizabeth, Yuri, Ericson, Gabriel, Wilson, Renato e Villar, pelo apoio e ajuda.

Ao PPGCC-UFMG pela organização, competência e estrutura e ao CNPq pelo apoio financeiro.

Por fim, agradeço a todos aqueles que, sabendo ou não, direta ou indiretamente, me ajudaram no desenvolvimento desta dissertação.

"So you think you can tell Heaven from Hell?"
(Pink Floyd)

Resumo

A exploração de ambientes não conhecidos é um campo muito estudado na robótica móvel. Assumindo que o robô não está ciente da sua posição inicial no ambiente, se faz necessário que o mesmo seja capaz de locomover-se produzindo um mapa do ambiente ao mesmo tempo em que se localiza nesse mesmo mapa. Esse processo é geralmente chamado de SLAM (Simultaneous Localization and Mapping).

Esta dissertação apresenta uma nova abordagem para o mapeamento e a localização simultânea de robôs móveis em ambientes desconhecidos que utiliza planos simples de forma a guiar a exploração do ambiente. São utilizados mapas topológicos, gerados de forma dinâmica, em conjunto com uma função de utilidade para decidir quais ações o robô deve executar para melhorar a eficiência do mapeamento.

Foram realizados simulações e experimentos reais de forma a avaliar a abordagem proposta e os resultados apresentam um ganho significativo na eficiência do mapeamento, tanto em termos da qualidade do mapa, como em termos do desempenho de execução quando comparado com uma exploração aleatória. Ainda foram feitos experimentos para testar cada termo da função de utilidade, demonstrando a importância de cada um deles para a exploração e a qualidade do mapa.

Palavras-chave: Robotica, Mapeamento, Exploração, Planejamento.

Abstract

The exploration of unknown environments is a common topic in mobile robotics. Assuming that the robot does not know its initial position, it is necessary for the robot to be able to navigate through the environment generating a map while it localizes itself on this same map. This process is generally known as SLAM (Simultaneous Localization and Mapping).

This dissertation presents a novel approach for augmenting simultaneous localization and mapping (SLAM) with simple plans that guide the exploration of the environment. We use dynamically generated topological maps in conjunction with a utility function to decide which actions the robot should perform in order to improve mapping efficiency.

We execute a series of simulated and real experiments in order to evaluate the proposed approach and results show a significant improvement of mapping efficiency both in terms of map quality and execution performance when we compared with a random exploration. Also, we executed experiments to test each term of the utility function, showing the importance of each one to the exploration and the map quality.

Keywords: Robotic, Mapping, Planning, Exploring.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

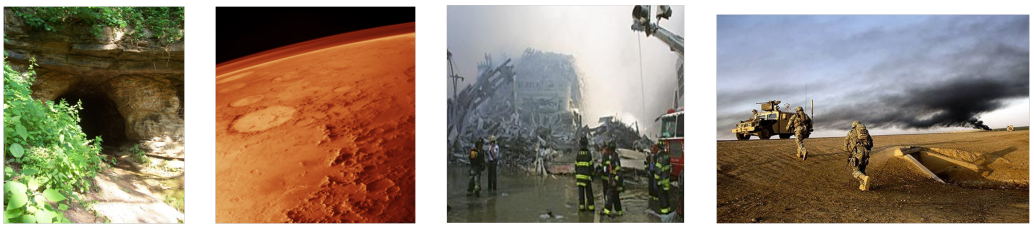
1.1 Motivação

A pesquisa em robótica tem crescido exponencialmente nos últimos anos. Uma área de grande interesse pela comunidade está relacionada à capacidade de um robô de agir de forma autônoma. Conseguir fazer o robô interagir com o ambiente em que ele está inserido, sendo capaz de tomar suas próprias decisões é um dos grandes objetivos da robótica. Esse objetivo é muito amplo e envolve diversos sub-problemas e áreas de pesquisas.

Um desses sub-problemas está relacionado ao conhecimento do robô sobre o ambiente ao seu redor. Um robô, para interagir de forma autônoma com o ambiente, precisa ser capaz de executar uma série de tarefas que o permita perceber o ambiente, tais como, localização, planejamento de trajetória, navegação, mapeamento. Realizar essas tarefas de forma isolada, por exemplo, localizar-se quando possui o mapa ou fazer o mapeamento do ambiente quando se conhece a localização são problemas que possuem algoritmos consolidados na literatura para a sua resolução.

A fim de evitar a necessidade de possuir o mapa previamente, é possível fazer com que o robô mapeie o ambiente enquanto navega pelo mesmo. Esse processo é geralmente chamado de SLAM (*Simultaneous Localization and Mapping*) [Smith et al., 1990]. Desse modo, o robô possui um mapa fiel do ambiente e ainda se localiza dentro dele. Com essa abordagem, surge a possibilidade de explorar ambientes antes desconhecidos e de interagir com esse novo ambiente. Isso nos permite enviar o robô para ambientes de difícil acesso ao ser humano. A Figura 1.1, apresenta alguns desses ambientes.

Sendo o robô capaz de mapear e se localizar dentro desse mapa, o próximo passo é fazê-lo capaz de explorar o ambiente desconhecido de forma eficiente. Para conseguir



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

Figura 1.1. Exemplos ambientes onde um robô explorador autônomo seria útil.

um alto nível de autonomia, o robô deve ser capaz de escolher de forma inteligente como ele irá se deslocar pelo ambiente. Apenas um deslocamento aleatório não permite uma exploração eficiente do ambiente por parte do robô. Para atingir esse objetivo é preciso que o robô seja capaz de planejar as suas ações ao mesmo tempo que mapeia o ambiente.

1.2 Exploração de Ambientes com Planejamento

A exploração de ambientes não conhecidos é um campo muito estudado na robótica móvel. Como explicado na Seção 1.1, nesse cenário, o robô desconhece o ambiente em que está inserido e tenta produzir um mapa desse lugar através das informações sensoriais que ele recebe. Assumindo que o robô não está ciente da sua posição inicial relativa ao ambiente, se faz necessário que o mesmo seja capaz de locomover-se produzindo um mapa do ambiente ao mesmo tempo em que se localiza nesse mesmo mapa. A esse problema se dá o nome de *Simultaneous Localization and Mapping*(**SLAM**).

O problema do SLAM pode ser descrito como um processo de estimação da posição do robô baseado nas últimas leituras dos seus sensores. Iterando todas as leituras em função do tempo, é possível descrever o movimento do robô e estimar um mapa do ambiente. Geralmente são utilizados pelo menos dois sensores para resolver o problema do SLAM. Usualmente é comum utilizar um sensor de odometria para descrever o deslocamento do robô e um sensor capaz de perceber características do ambiente para descrever o mapa e determinar a posição real do robô, geralmente um laser ou um sonar. A fusão das leituras dos dois ou mais sensores sensores permite uma confiança maior no mapa obtido e na posição relativa do robô.

Apesar do grande número de trabalhos sobre SLAM, apenas algumas abordagens preocupam-se explicitamente em como o robô deverá explorar o ambiente. Conhecidas por SPLAM - *Simultaneous Planning, Localization and Mapping* [Leung et al., 2006b], ou *Active SLAM* [Stachniss et al., 2004], essas abordagens utilizam as informações sensoriais não apenas para mapear o ambiente e localizar o robô, mas também para

planejar as suas ações.

A componente de planejamento em um sistema de SLAM deve ser capaz de analisar diversos fatores, tais como o mapa construído até o momento, as informações sensoriais atuais, o estado do robô e, baseado neles, decidir qual ação o robô deve executar. Essa ação pode possuir diferentes objetivos como melhorar a qualidade do mapa, explorar o ambiente ou reduzir a incerteza da localização do robô.

Ainda sobre a componente de planejamento, essa pode aparecer no sistema de SLAM de diferentes maneiras. Alguns trabalhos [Feder et al., 1999; Makarenko et al., 2002] procuram planejar a melhor trajetória para o robô com base nas informações atuais do mapa e dos sensores, mas sem se preocupar com um planejamento de mais alto nível, ou seja, apenas decidem a melhor ação para um dado momento, sem se preocupar com futuros planejamentos. Outros trabalhos [Leung et al., 2008] utilizam a idéia de comportamentos para controlar os robôs. Nesses trabalhos, geralmente existem 2 comportamentos: explorar ou melhorar a qualidade do mapa, e o robô pode mudar de um para o outro dependendo do estado que o sistema se encontra. Uma outra abordagem [Huang et al., 2005] diz respeito às ações que o robô deve tomar em um determinado momento. Essas ações podem tanto ser comandos diretos, como por exemplo, "seguir em frente" ou "virar à esquerda", como podem ser instruções mais abstratas, como por exemplo, mandar o robô para um determinado ponto do mapa ou encontrar um marco no ambiente. Essas abordagens serão discutidas em mais detalhes no Capítulo 2.

1.3 Definição do Problema e Objetivos

O problema abordado nessa dissertação pode ser definido como: dado um robô com pose inicial desconhecida, equipado com um sensor de odometria e um sensor laser em um ambiente que não se possui nenhuma informação prévia, mapear e explorar esse ambiente de forma eficiente. Por eficiente entende-se que o robô deve explorar o ambiente da forma mais rápida e direta possível, não desconsiderando a qualidade do mapa obtido. Mais especificamente, podemos dizer que o objetivo é aplicar uma componente de planejamento em um sistema robótico executando SLAM. Por planejamento considera-se a escolha de pontos do mapa que o robô deve visitar para uma exploração eficiente do ambiente.

De forma a solucionar esse problema, foi proposta uma nova abordagem de SPLAM que utiliza o *Distributed Particle - SLAM* (**DP-SLAM**) [Eliazar & Parr, 2003] como algoritmo base de SLAM. Sobre o mapa métrico fornecido pelo DP-SLAM,

é gerado de forma dinâmica um mapa topológico do ambiente mapeado até o momento. Com base nesse mapa topológico são obtidas informações necessárias ao planejamento, como a qualidade do mapa em determinados pontos ou as fronteiras entre o que já foi mapeado e o que é desconhecido ainda.

O planejamento é realizado através de um funcional de utilidade, chamado de função de utilidade para essa dissertação. Essa foi definida para balancear a exploração do ambiente com a melhoria da qualidade do mapa. Baseando-se nas informações que o mapa topológico armazena do mapa métrico, a função de utilidade seleciona qual ponto do mapa o robô deve visitar como uma próxima ação.

1.4 Organização da Dissertação

Este documento está organizado da seguinte forma: neste capítulo foi apresentado o tema a ser tratado na dissertação. O Capítulo 2 apresenta uma revisão de trabalhos que exploram o SPLAM. No Capítulo 3 é apresentado o sistema de SPLAM proposto, analisando todos os detalhes que o compõe. O Capítulo 4 apresenta os experimentos e os resultados obtidos e, no Capítulo 5 são apresentadas as conclusões e trabalhos futuros.

Capítulo 2

Revisão Bibliográfica

2.1 Estimação de Estados

Segundo Thrun et al. [2006], a estimação de estados, em robótica, trata do problema de estimar valores dos dados sensoriais que não são diretamente observáveis, mas podem ser inferidos. Um estado, a partir de agora denominado x, define as características do ambiente que são importantes para a solução do problema. O estado é composto por variáveis relativas ao robô e ao ambiente que precisam ser determinadas ao longo do tempo, para descrever como está o ambiente em um determinado tempo t. No problema do SLAM, é comum o estado s_t ser composto pela pose do robô $\mathbf{q}_t = [x_t, y_t, \theta_t]^\mathbf{T}$ e pelas pose de alguns objetos, "marcos", do ambiente $\mathbf{m}_t^i = [x_t, y_t, \theta_t]^\mathbf{T}$.

Para estimar o estado s_t de um robô a partir de suas ações e percepções ao longo do tempo é feito um processo chamado de filtragem. Em um mundo perfeito, sem erros de atuação, ao se aplicar uma ação, denominada lei de controle, u_t, espera-se que o estado do robô corresponda com o resultado da aplicação da lei de controle. Por exemplo, aplicar um de giro de 90°, deveria fazer com que a pose do robô em $t+1$ fique $\mathbf{q}_{t+1} = [x_t, y_t, \theta_t + 90°]^\mathbf{T}$. Mas, como existem erros nos atuadores, não é possível garantir que o robô tenha girado exatamente 90°. Então, após aplicar a lei de controle u_t, o robô deve fazer uma leitura dos dados dos sensores, z_t, para determinar a sua posição real.

No processo de filtragem, o estado s_t é dado por $p(s_t|s_{0:t-1}, z_{1:t-1}, u_{1:t})$. Considerando a suposição de Markov, no qual o estado s_t depende apenas do estado anterior e da lei de controle u_t, é possível descartar todas as leituras anteriores de $u_{1:t-1}$ e $z_{1:t-1}$ e utilizar apenas u_t para prever o novo estado: $p(s_t|s_{t-1}, u_t)$. Essa probabilidade é chamada de modelo de transição. Ainda, se o estado s_t for completo, podemos assumir que a observação z_t depende apenas do estado atual s_t, logo temos o modelo de observação

$p(z_t|s_t)$.

O modelo de transição e o modelo de observação são o estado de crença do robô. Este conceito reflete o conhecimento que o robô possui sobre o estado atual. Baseando-se apenas nas ações de controle e nas observações, o estado de crença procura prever o estado do sistema a partir da distribuição de probabilidade: $bel(x) = p(s_t|z_{1:t}, u_{1:t})$.

2.1.1 Filtros de Partículas

Uma forma de computar o estado de crença $bel(x)$ é utilizando Filtro de Partículas. A idéia principal do filtro de partícula é representar o estado de crença como um conjunto de amostras aleatórias do estado do sistema. Essa representação é aproximada, porém, como não é paramétrica, ela pode representar um número muito grande de distribuições. Será feita uma rápida introdução, mas informações mais detalhadas podem ser encontradas nos trabalhos de Doucet et al. [2001], Thrun [2000] e Thrun et al. [2006].

O Filtro de Partículas mantém um conjunto de partículas $S_t = \{S_t^1 ... S_t^n\}$, que representam uma hipótese de como seria o estado real do sistema no momento t. Cada partícula recebe um peso relacionado à qualidade da estimativa do estado sistema.

Ao aplicarmos em cada uma dessa novas partículas o modelo de transição de Markov $p(S_t|S_{t-1}, u_t)$, é gerado um novo conjunto de partículas $S_{t+1} = \{S_{t+1}^1, ..., S_{t+1}^n\}$. Então as partículas são avaliadas em função da observação z_t de acordo com o modelo de observação de Markov $p(z_t|S_t^i)$. Essa avaliação determina um peso normalizado para cada partícula, significando quão boa é a estimativa representada pela partícula. Na próxima iteração, apenas as partículas de melhor peso geram novas partículas, as outras são retiradas do sistema.

Filtros de Partículas podem ser usados para o problema da localização do robô, onde a posição do robô é o estado escondido, o modelo de transição representa o movimento do robô e o modelo de observação são as informações sensoriais. O modelo de transição é representado por uma função linear e pode ser descrito para cada partícula como:

$$S_t = A_t * S_{t-1} + B_t * u_t + N(0, \sigma), \tag{2.1}$$

onde A_t e B_t são argumentos lineares. É possível perceber que uma função de ruído aleatório de distribuição normal com média 0 e desvio padrão σ, $N(0, \sigma)$, foi aplicada nas equações. Essa função é utilizada para tentar modelar esses erros citados da odometria. Como não é possível descrever exatamente qual é esse erro, funções dessa forma são utilizadas para tentar reduzir os erros na localização.

2.2 SLAM

Como mencionado, o SLAM - *Simultaneous Localization and Mapping* é significativamente mais difícil que a localização em um mapa conhecido e significativamente mais difícil que o mapeamento com a localização do robô conhecida [Thrun et al., 2006]. A princípio, se os sensores e atuadores presentes no robô fossem perfeitos, o problema do SLAM seria trivial. Nesse cenário, seria necessário apenas integrar os valores de odometria do robô a cada instante, atualizando o mapa com os dados recebidos pelos sensores. O problema é que tanto os sensores, quanto os atuadores, sofrem problemas de ruídos fazendo com que os dados obtidos pelo robô possuam um alto grau de incerteza.

Para contornar esse problema, o SLAM segue um processo probabilístico de atualização da posição do robô e de construção do mapa. A base desse processo foi descrito por Smith et al. [1990] no seu trabalho sobre mapeamento estocástico. Nele, é assumido que, a partir dos dados fornecidos pelos sensores, é possível observar marcos distintos no ambiente. Ao se reobservar esses mesmos marcos ao longo do tempo, é possível estimar a localização do robô e a localização dos próprios marcos, de modo a construir o mapa. Dessa forma, em um determinado instante de tempo t, o estado do sistema s_t possuirá a pose do robô q_t e a estimativa de localização de cada marco m_t^i:

$$s_t = \begin{pmatrix} q_t \\ m_t^1 \\ \cdot \\ \cdot \\ m_t^i \end{pmatrix} \tag{2.2}$$

2.2.1 Tipos de SLAM

Existem diferentes tipos de algoritmos de SLAM, cada um com características próprias. Um tipo de algoritmo de SLAM muito comum é o baseado em características do ambiente, como o trabalho seminal [Smith et al., 1990] descrito acima. Nessa abordagem, o robô procura no ambiente por marcos que diferenciem a posição em que ele se encontra de outras as quais ele já passou. Dessa forma, ao perceber que ele está em uma certa posição relativa a um marco visto anteriormente, o robô consegue corrigir a sua posição em relação ao mapa estimado. Possivelmente o algoritmo mais utilizado com essa abordagem é o FastSlam [Montemerlo et al., 2002, 2003]. Esse algoritmo é baseado em filtro de partículas Rao-Blackwellized [Doucet et al., 2000], o que permite representar o mapa como um conjunto de distribuições gaussianas das posições dos marcos do ambiente, enquanto aplica diferentes hipóteses da posição do robô como se fosse um filtro de partículas simples. Outra abordagem bem comum são algoritmos que

poderiam ser chamados de EKF-SLAM [Dissanayake et al., 2001; Paskin, 2003] por se utilizarem do filtro de kalman extendido para tentar localizar o robô no mapa. Nessa estratégia de SLAM ainda pode ser citado o trabalho de Yuen & MacDonald [2004], que utiliza múltiplos filtros de Kalman para a realização do SLAM.

Uma alternativa aos algoritmos baseados em características do ambiente são os chamados SLAM Topológicos. Essa estratégia procura determinar um mapa mais simples, com apenas informações relevantes de lugares explorados pelo robô. Muitas vezes, o mapa topológico pode envolver informações de contexto, métricas ou apenas uma descrição do movimento do robô. Nessa área podem ser citados algoritmos que possuem o mapa topológico de forma explícita [Kuipers et al., 2004; Thrun et al., 1998] ou alguns que possuem ele de forma indireta [Gutmann & Konolige, 2000; Angeli et al., 2009]. O mapa topológico também pode ser utilizado como forma de diminuir a quantidade de informação no sistema. No trabalho de Victorino & Rives [2006], por exemplo, apenas regiões de interesse são realmente mapeadas, para situações em que o robô se encontra, por exemplo, em um corredor, simétrico, sem características marcantes, o mapeamento é feito apenas de forma topológica. Outra utilidade do mapa topológico é auxiliar na navegação do robô e na atualização da posição das marcos do mapa [Lee et al., 2007; Ko et al., 2004]. Nesse contexto, o mapa topológico auxilia na correção das posições dos marcos, armazenando informações relativas às posições dos mesmo.

Uma outra estratégia de SLAM são as que não utilizam nenhuma característica do ambiente para localizar e mapear. Essas abordagens procuram identificar mapas inicialmente locais e só depois transferem a informação ao âmbito global. Enquanto isso, a posição do robô é calculada com a premissa de que o mesmo deve estar dentro de algum ponto já mapeado, e iterando todas as posições anteriores, é possível prever a sua posição atual. O DP-SLAM [Eliazar & Parr, 2003, 2004] aplica essa ideia, utilizando filtro de partículas para guardar não um, mas vários possíveis mapas. Por ser a base da abordagem utilizada nesse trabalho, uma explicação mais detalhada sobre o funcionamento desse algoritmo será apresentada no Capítulo 3.

2.3 SLAM com Planejamento (SPLAM)

A maioria dos trabalhos relacionados com SLAM não se preocupam explicitamente com o controle do robô. Geralmente, a exploração do ambiente ocorre de maneira independente dos dados obtidos pelos sensores e não levam em conta o mapa construído pelo robô até o dado momento [Stachniss et al., 2004]. Isso se deve ao fato de que esses

trabalhos focam na melhoria de aspectos próprios do SLAM como a localização e o mapeamento dos ambientes ou encontrar métodos mais eficientes para esse problema, não se importando em como o robô se desloca no ambiente para obter os dados.

De fato, a maior dificuldade em se incluir um planejamento em SLAM se deve ao fato de que o próprio mapa do ambiente não é conhecido, não permitindo desse modo, um planejamento a longo prazo [Huang et al., 2005]. No entanto, para tarefas de exploração de ambientes desconhecidos, estratégias como caminhamento aleatório podem não ser opções favoráveis. Em um cenário como esse é interessante determinar estratégias mais eficientes de exploração, que levem em conta tanto a localização quanto o mapeamento do ambiente. A essa área de pesquisa dá-se o nome de Robótica Exploratória [Makarenko et al., 2002]. Essa área envolve três linhas de pesquisa: localização, mapeamento e determinação da trajetória . A determinação da trajetória é responsável pela tomada de decisão de qual ação deve ser executada pelo robô de forma a maximizar a informação no sistema, seja escolhendo um bom caminho de exploração do mapa, seja melhorando a informação já contida no mapa.

Alguns dos trabalhos envolvendo SPLAM baseiam-se em escolher a próxima ação do robô a ser executada para maximizar uma função de utilidade do tipo:

$$\mu = max(U), \tag{2.3}$$

onde U é a função de utilidade e pode variar dependendo das condições e das intenções do sistema. O primeiro trabalho a descrever uma estratégia de SPLAM foi Feder et al. [1999], onde a função de utilidade se resume a escolher a próxima ação com maior probabilidade de maximizar a informação existente no sistema.

Nesse trabalho, Feder et al. [1999] procurava estimar qual ação do robô provocaria um maior ganho na *Probability Density Function*(PDF) do sistema. A PDF é representada por uma matriz que possui informações acerca da incerteza entre as localizações do robô, de cada marco, de cada marco para o robô e de cada marco para um outro marco. Por se basear apenas na PDF do sistema, o robô ficava muito restrito a navegar apenas por lugares por onde ele já visitou, procurando melhorar a localização de marcos vistos previamente, ignorando a possibilidade de explorar o ambiente. E por apenas planejar a próxima ação, essa estratégia apresenta problemas de controle ao longo do tempo.

Para resolver os problemas citados, Bourgault et al. [2002] propôs uma nova função de utilidade (2.4), que procura englobar, tanto a exploração de novas áreas, quanto a melhoria da localização do robô. Cada um desses termos é multiplicado por um peso para ponderar entre exploração e aumento da informação do sistema. A função

proposta foi

$$\mu = max(w_I U_I + w_L U_L), \tag{2.4}$$

onde U_I privilegia a exploração de novas áreas, U_L privilegia a qualidade da localização do robô e w_I, w_L são os pesos de ponderação para cada termo da função.

Makarenko et al. [2002] adicionou à função de utilidade um terceiro termo para penalizar trajetórias muito longas. Como o mapa do ambiente é desconhecido, não é possível afirmar a priori que uma determinada trajetória é a melhor para o mapeamento. Dessa forma, uma trajetória mais curta permite uma melhor otimização da trajetória executada pelo robô.

As estratégias apresentadas anteriormente consideram o SPLAM para apenas uma ação. Huang et al. [2005] classificou essa estratégia como *single step look-ahead strategy*, pois planeja apenas a próxima ação do robô. Alguns trabalhos argumentam que planejando um conjunto de ações, ao invés de apenas uma ação produzem resultados melhores. Como não é possível prever se ocorrerá um ganho de informação no sistema, ou se novos marcos serão encontrados, pode-se supor para o planejamento que não haverá ganho de informação no sistema e que o mapa se manterá o mesmo até que se recalcule um novo plano.

Huang et al. [2005] mostrou que aplicando essa abordagem nas PDFs do sistema é possível obter resultados melhores que o planejamento de uma ação apenas, e denominou a estratégia de *Model Predictive Control* (MPC). No MPC, embora o planejamento seja feito para os próximos N passos, a trajetória é replanejada a cada passo, à medida que novos dados são inseridos na estimação do sistema. Apesar de apresentar resultados melhores, o MPC possui o mesmo problema do trabalho de Feder et al. [1999], ou seja, tende a navegar por áreas já exploradas.

Leung et al. [2006a] apresentou uma solução para esse problema utilizando uma máquina de estados que determina o comportamento do sistema entre 3 alternativas: exploração, melhoria da localização ou melhoria do mapa. Para não modificar o MPC, os autores utilizam alguns marcos falsos com alta incerteza posicionados em locais de interesse, levando o MPC a traçar trajetórias àquelas regiões. O trabalho de Leung et al. [2008] apresenta uma implementação prática para a metodologia apresentada.

Alguns trabalhos utilizam a noção de planejamento na forma de coordenação de grupos de robôs realizando SLAM. Esses trabalhos precisam coordenar as ações de cada robô para otimizar a exploração. O conceito de planejamento pode ficar implícito no sistema, mas ainda assim deve ser levado em conta.

No trabalho de Fox et al. [2006] a coordenação dos robôs é feita através de uma

política de comportamentos. O robô pode tanto explorar o ambiente, como pode tentar encontrar com outros robôs. Ao explorar o ambiente, os robôs utlizam uma estratégia conhecida por *frontier based exploration* proposta por Yamauchi [1997]. Essa estratégia procura por regiões do mapa onde exista a fronteira entre lugares mapeados e lugares desconhecidos e traça a trajetória do robô para navegar até essa região. Essa abordagem também pode ser vista em outros trabalhos com múltiplos robôs como por exemplo [Burgard et al., 2000; Yamauchi, 1998]

Como será descrito no próximo capítulo, este trabalho utiliza uma função de utilidade para decidir a próxima ação do robô como nos trabalhos de Feder et al. [1999]; Bourgault et al. [2002]; Makarenko et al. [2002], mas o algoritmo de SLAM utilizado não necessita de marcos no ambiente. Para suprir essa necessidade, foi utilizada a abordagem proposta por Ko et al. [2004] de gerar sobre o mapa métrico um mapa topológico para ser a base do nosso planejamento.

Capítulo 3

Metodologia

Como mencionado no Capítulo 1, o objetivo deste trabalho é propor uma política de navegação que possa ser integrada com o DP-SLAM de forma a obter uma exploração mais eficiente do ambiente e um mapa de melhor qualidade. Para tanto, foram criados mecanismos capazes de gerarem planos de navegação para o robô, enquanto o algoritmo de SLAM roda em paralelo. Foi construído um mapa topológico sobre o mapa métrico calculado pelo SLAM. Esse mapa topológico é uma abstração do mapa métrico, capaz de armazenar informações importantes do ambiente que são utilizadas por uma função de utilidade responsável por gerar os planos de como o ambiente deve ser explorado.

A Figura 3.1 apresenta a arquitetura do sistema. O sistema é composto de duas etapas: uma de planejamento e uma de navegação. A etapa de planejamento é dividida em 2 sub-etapas: o Mapa Topológico, responsável por analisar e armazenar informações do mapa gerado pelo SLAM; e a Função de Utilidade, que recebe as informações do mapa topológico e decide qual deve ser a próxima ação tomada pelo robô. A etapa de navegação também é dividida em 2 sub-etapas: o SLAM propriamente dito, responsável apenas pelo mapeamento do ambiente; e o Navegador, que recebe da função de utilidade uma região do mapa e tenta levar o robô até a região alvo de forma segura.

.

A Figura 3.1 ilustra o fluxo de funcionamento do sistema. Inicialmente, o robô permanece estático enquanto o SLAM gera o primeiro mapa. Esse primeiro mapa é passado para a etapa de planejamento. O Mapa Topológico gera as informações necessárias à função de utilidade, que calcula a melhor região a ser explorada e passa para o navegador. A partir desse momento, apenas a etapa de navegação permanece ativa, o planejamento se mantém estático até o robô chegar na região escolhida pela função de utilidade. Ao chegar na região, novamente o robô fica parado, esperando o SLAM terminar de mapear todas as informações coletadas até então, e o processo

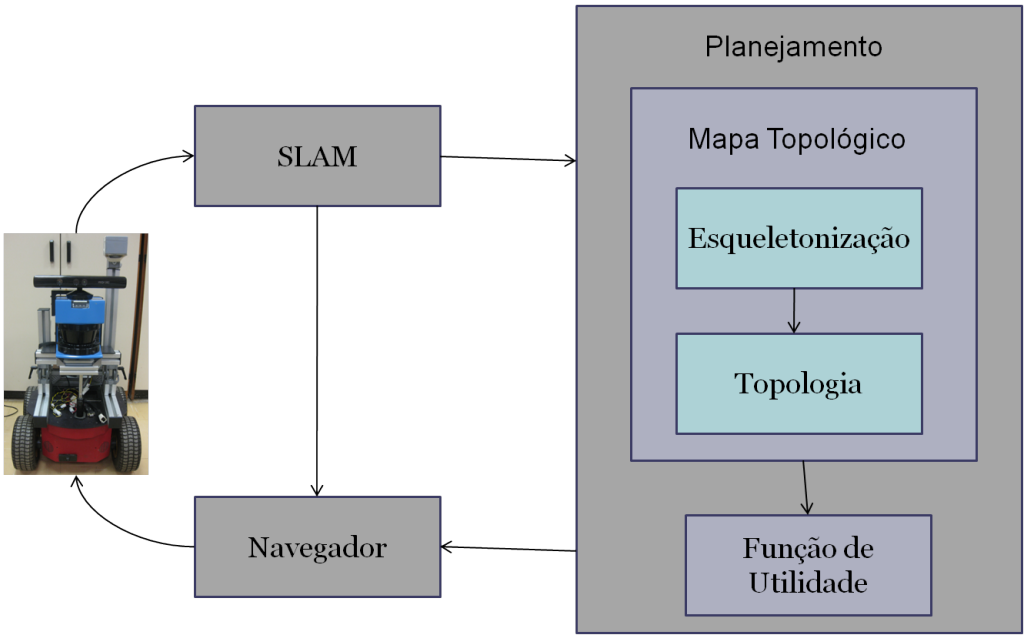


Figura 3.1. Os componentes principais do sistema.

inicial se repete. As próximas seções explicam cada um desses componentes em detalhes

3.1 DP-SLAM

Como algoritmo de SLAM foi utilizado o *Distributed Particle SLAM* (DP-SLAM), proposto nos trabalhos Eliazar & Parr [2003, 2004] e Eliazar [2005]. Ele foi escolhido como algoritmo base do nosso sistema por ser *open source* e possuir características favoráveis à nossa aplicação. A procura por algoritmos de SLAM foi baseada no site *www.openslam.org*, que é uma plataforma para que pesquisadores possam publicar seus algoritmos de SLAM e possui diversas implementações disponíveis. Mais especificamente, dentre as opções possíveis, o DP-SLAM foi selecionado por:

- **Implementação Genérica**: O DP-SLAM exige apenas informações de odometria e de um sensor de distância em um formato pré-definido, dessa forma é possivel adaptá-lo para o uso em praticamente qualquer situação;
- **Mapa Visual**: O DP-SLAM fornece como saída um mapa no formato ".png", o que nos permite realizar um planejamento em cima da imagem do mapa;
- **Fácil integração com o Player/Stage**: O código do DP-SLAM já possui funções próprias para adicionar dados provindos de algum robô, facilitando a

sua integração com a plataforma Player/Stage [Gerkey et al., 2003] usada nos experimentos;

- **Boa Documentação**: O DP-SLAM é bem documentado, tanto o seu algoritmo, como o seu funcionamento;
- **Execução On-Line**: Alguns algoritmos de SLAM funcionam apenas com logs de caminhos já feitos por robôs, o que impossibilitaria o seu uso para o nosso sistema. Já o DP-SLAM consegue receber as informações sensoriais do robô e processá-las em tempo real.

Muitos algoritmos de SLAM implementam filtros de partículas nos mapas e nas posições do robô, usando grades de ocupação para representar o mapa, para acompanhar as posições dos objetos no ambiente. O grande problema com essa abordagem é a sua complexidade, pois para cada partícula que será gerada em um dado instante, é necessário copiar todo o mapa. Esta cópia é um gargalo do sistema se levado em conta o processamento e um sério problema de uso de memória por parte do algoritmo. A grande contribuição do DP-SLAM é uma representação mais eficiente do mapa de forma a deixar mais rápida as cópias e reduzindo a quantidade de memória necessária para representar as grades de ocupação.

Para tentar resolver esse problema, o DP-SLAM utiliza a noção de *particle ancestry*, ou ancestral da partícula. Se uma partícula na iteração t gera novas partículas na iteração $t+1$, essas partículas são chamadas de "filhas" e a original de "pai". Duas partículas com o mesmo "pai", são chamadas de "irmãs". É fácil perceber que os mapas de partículas "filhas" diferenciam da partícula pai, apenas pela observação adicionada ao mapa no instante da sua geração. Apesar de parecer promissora a idéia de armazenar apenas as novas observações de cada mapa, isso pode causar um grande transtorno na hora de localizar o robô, já que a partícula não possui o mapa completo, e para obtê-lo seria preciso percorrer todo o caminho na árvore de partículas até a partícula inicial. Para resolver esse problema o DP-SLAM implementa o que os autores chamaram de *Distributed Particle Mapping* (**DP-Mapping**).

Cada nodo presente na árvore de ancestrais representa uma hipótese, uma possibilidade de mapa e de posição do robô. Na raiz da árvore se encontra a partícula inicial e todas as outras partículas são "descendentes" dela. Cada partícula possui um apontador para a sua partícula "pai", um ID único e uma lista de células da grade de ocupação que esta partícula alterou.

Para evitar que a árvore cresça muito e acabe causando problemas de espaço e tempo no sistema, a cada nova geração adicionada na árvore, ocorre uma poda em

folhas que se tornaram desnecessárias. Em cada nova geração, as partículas são avaliadas e apenas as melhores partículas são escolhidas para gerar novas partículas. Desse modo, aquelas partículas que não geraram filhos podem ser removidas da árvore. Isso pode ocasionar um efeito em cascata, removendo da árvore diversas partículas recursivamente. Outra poda que é feita, está relacionada com partículas que possuam apenas um filho. Nesse caso, as informações da partícula filha são adicionadas à partícula pai e a partícula filha é removida da árvore. Assim os autores conseguiram manter uma árvore relativamente pequena em consideração ao tamanho que ela poderia atingir.

A Figura 3.2 mostra as podas feitas na árvore em um dado instante de execução. A Figura 3.2(a) demonstra a árvore antes da poda e a Figura 3.2(c) a árvore após a poda. Na Figura 3.2(b), foram marcadas as podas que aconteceram. Inicialmente, são realizadas as podas de partículas que não geraram filhos. Essa podas foram marcadas com uma reta diagonal em vermelho. É interessante notar à direita da árvore que uma ramificação inteira de partículas foi cortada da árvore, pois as duas filhas não geraram filhos, deixando a partícula pai sem filhos também. Logo após essa poda, são verificadas partículas com apenas um filho. Nessa hora é possível ver que uma cadeia inteira de partículas seja reduzida a apenas uma partícula.

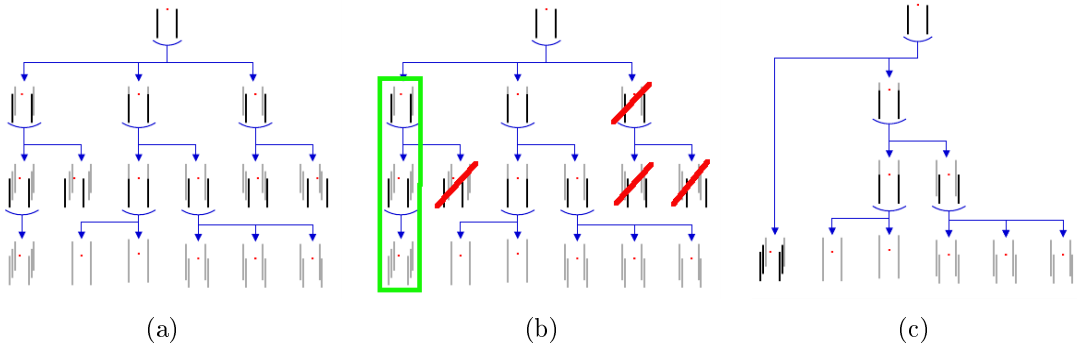


Figura 3.2. Processo de poda na árvore do DP-SLAM [Eliazar, 2005].

Apesar da poda da árvore conseguir reduzir consideravelmente o número de partículas no sistema, manter um mapa para cada partícula ainda é um processo custoso. Para resolver esse problema, o DP-SLAM utiliza apenas um mapa "real" e vários mapas "virtuais". O mapa é uma grade de ocupação, mas cada célula do mapa não representa ocupado ou livre. Ao invés disso, cada célula mantém uma árvore de observações que armazena o ID de cada partícula que já observou aquela célula e cada partícula possui uma lista com as células que ela observou. Dessa forma, se uma partícula precisa saber o status de uma célula, basta procurar na árvore de observação pelo ancestral que fez uma observação mais recente àquela célula.

É essa representação que permite a poda de partículas com um só filho. Dessa forma é possível reduzir a informação em cada célula. Para realizar a poda, a lista de observações da partícula filha é percorrida e todos os itens presentes recebem o ID da partícula pai; se a partícula pai possuía alguma observação nesses itens, pode-se removê-las e depois pode-se adicionar esses itens à lista de observações da partícula pai.

O DP-SLAM possui ainda uma outra característica chamada pelos autores de SLAM hierárquico. Ao se deslocar pelo ambiente o robô acumula erros de odometria. Para reduzir a influência desses erros ao mapear e localizar o robô, o DP-SLAM implementa um modelo desse erro em seu algoritmo. Primeiramente, é executado o SLAM, chamado de *low slam*, produzindo um "*low map*". As informações de trajetória do robô e de observações feitas pelo robô obtidas nessa fase são utilizadas na próxima fase chamada de *high slam*. Nesta fase, o DP-SLAM aplica o modelo de erro na posição do robô e gera a trajetória do robô com as informações obtidas no *low slam* utilizando as observações feitas como base. Dessa forma, é obtido o "*high map*". Nós chamamos o processo de obtenção dos "*low map*" e "*high map*" de *ciclo* de SLAM. É interessante notar que, enquanto o "*high map*" possui toda a informação já mapeada pelo robô, o "*low map*" possui informações apenas do ciclo atual. Os Algoritmos 1, 2 e 3 apresentam o funcionamento do DP-SLAM.

Algorithm 1 Algoritmo DP-SLAM

```
InitLowSlam();{Inicializa o mapa, variáveis,...}
InitHighSlam();{Inicializa o mapa, variáveis,...}
while continueSLAM do
  LowSlam();
  HighSlam();
end while
```

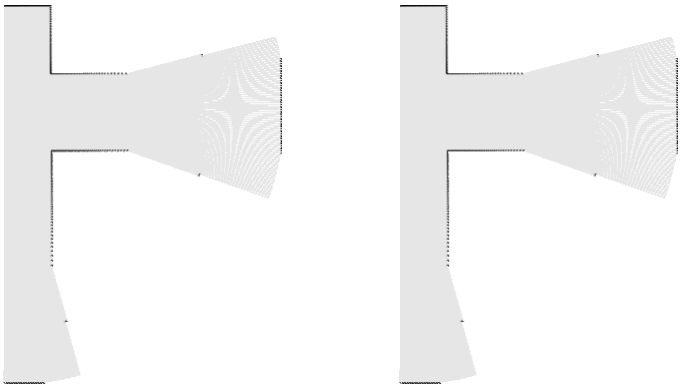
Algorithm 2 Low DP-SLAM

```
while lowCounter do
  GetSensation(); {Lê e armazena as informações dos sensores}
  Localize(); {Aplica a localização nas partículas, determinando as N melhores partículas}
  UpdateAncetry(); {Atualiza a árvore de ancestrais}
  LogSense(); {Gera um log das observações feitas pelo laser para o high SLAM}
  LogPath(); {Seleciona a melhor partícula e gera um log da odometria para o high SLAM}
  PrintMap(); {Gera o low map baseado na melhor partícula}
end while
```

Algorithm 3 High DP-SLAM

Require: *path*, *sense*; {Informações de trajetória e observação processadas pelo *low SLAM*}
HighLocalize(); {Aplica a localização nas partículas, determinando as N melhores partículas}
HighUpdateAncetry(); {Atualiza a árvore de ancestrais}
DefineBestParticle(); {Avalia as partículas e escolhe a melhor delas}
HighPrintMap(); {Gera o *high map* baseado na melhor partícula}

A Figura 3.3(a) ilustra o *low map* do 1º ciclo de um dos nossos experimentos e a Figura 3.3(b) ilustra o *high map* do mesmo ciclo. É possível notar que os dois mapas são idênticos, pois logo no primeiro ciclo, o *low map* é simplesmente copiado para o *high map*. A Figura 3.4(a) ilustra o *low map* do 2º ciclo. Como o robô alterou a sua orientação ao mapear essa região, a orientação do *low map* não continua a mesma do *high map*. Por esse motivo é preciso rotacionar o *low map*. A Figura 3.4(b) ilustra o resultado dessa rotação. Com o *low map* rotacionado, basta adicionar as suas informações ao *high map*. A Figura 3.5 ilustra essa o *high map* do 2º ciclo.



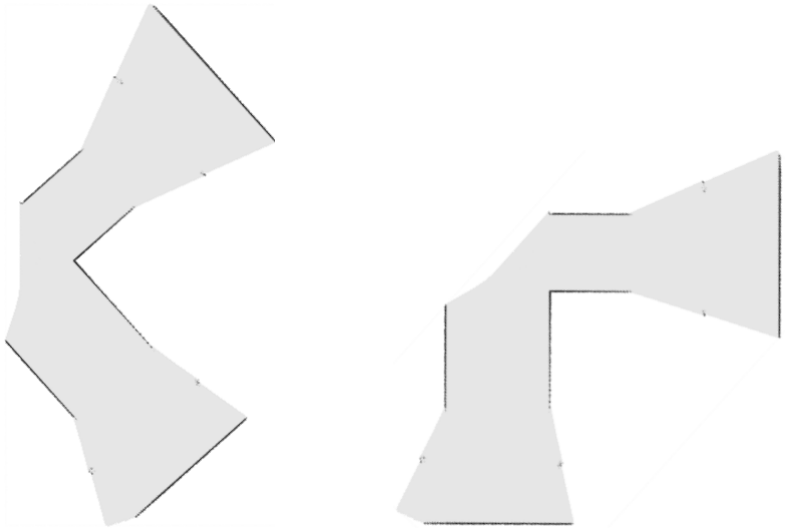
(a) *Low map* do 1º ciclo. (b) *High map* do 1º ciclo.

Figura 3.3. Mapas *low* e *high* do 1º ciclo

3.2 Mapa Topológico

Como mencionado anteriormente, o mapa topológico serve como a base para o planejamento. Como o DP-SLAM não utiliza marcos no ambiente, o mapa topológico foi utilizado para definir pontos de referência no mapa métrico.

O processo de gerar dinamicamente o mapa topológico é dividido em duas etapas. Inicialmente, um processo de **Esqueletonização** [Ballard & Brown, 1982] é aplicado ao mapa obtido pelo SLAM. O resultado desse processo, um conjunto de pontos, é então



(a) *Low map* do 2º ciclo.

(b) *Low map* do 2º ciclo rotacionado para a orientação do *High Map*.

Figura 3.4. *Low map* do 2º ciclo

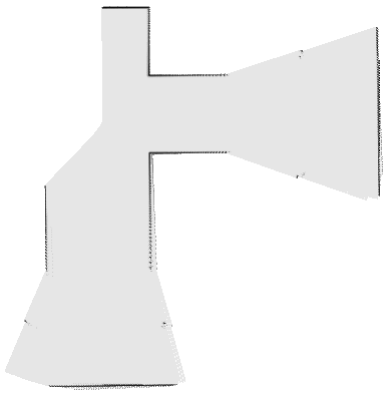


Figura 3.5. *High map* do 2º ciclo.

analisado e são criadas regiões circulares no mapa com centro em alguns desses pontos. Esses pontos são usados para gerar os nodos dos mapas topológicos. O funcionamento dessas duas etapas é explicado a seguir.

3.2.1 Esqueletonização

Essa etapa é baseada no trabalho de Ko et al. [2004] onde os autores utilizam um processo de esqueletonização para detectar o esqueleto do mapa obtido através do SLAM. Segundo o trabalho, o esqueleto de uma imagem é a menor representação possível da mesma. A Figura 3.6, retirada de Ko et al. [2004], demonstra o resultado obtido com o uso desse algoritmo.

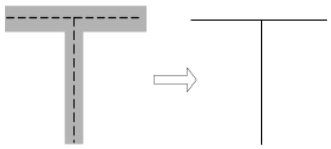


Figura 3.6. Resultado do processo de esqueletonização.

Como pode ser visto pela Figura 3.6, aplicar o processo de esqueletonização no mapa é muito parecido com o resultado obtido ao se computar um Diagrama de Voronoi Generalizado [Aurenhammer, 1991]. Dessa forma é possível obter uma representação do ambiente, demarcando os pontos mais distantes de todos os obstáculos. A escolha pela esqueletonização é baseada no nível menor de complexidade de sua aplicação.

A esqueletonização é realizada em imagens, então é necessário um mapa visual para ser utilizada. No início do processo, é preciso transformar o mapa obtido pelo SLAM até o momento em uma simples grade de ocupação binária, ou a célula está ocupada ou está livre. Para o nosso sistema, células ocupadas e células desconhecidas serão consideradas ocupadas e receberão valor 0 e células livres receberão valor 1.

Após a obtenção da grade de ocupação, uma janela 3X3, Figura 3.7, é deslizada pela imagem da grade. Sempre que a célula *p1* contiver um valor 1, ou seja, for uma célula livre, seus oito vizinhos são testados para ver se satisfazem certas condições:

p_9	p_2	p_3
p_8	p_1	p_4
p_7	p_6	p_5

Figura 3.7. A janela 3x3 usada para esqueletonização [Ko et al., 2004].

- 1º Passo:
  1. $2 \leq \mathrm{N}(p1) \leq 6$;
  2. $\mathrm{S}(p1) = 1$;
  3. $p2{*}p4{*}p6 = 0$;
  4. $p4{*}p6{*}p8 = 0$.
- 2º Passo:
  1. $2 \leq \mathrm{N}(p1) \leq 6$;
  2. $\mathrm{S}(p1) = 1$;

3. $p2*p4*p8 = 0$;
4. $p2*p6*p8 = 0$.

onde, N($p1$) = $p2 + p3 + ... + p8 + p9$ e S($p1$) é o número de mudanças de 0 para 1 na sequência $p2$, $p3$,...,$p8$, $p9$. Se alguma dessas condições for falsa, então $p1$ não é parte do esqueleto da imagem e é eliminado. Esse processo é repetido até que não haja mais pontos para serem eliminados. A Figura 3.8 demonstra o processo de *thinning*.

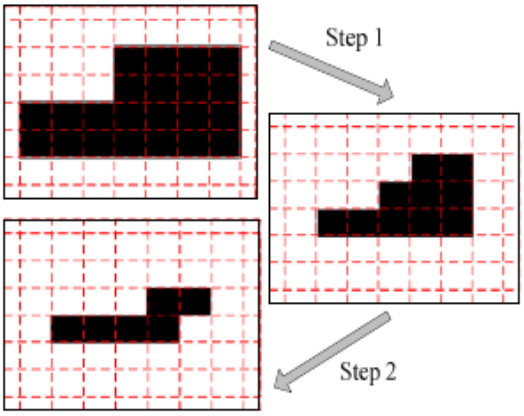


Figura 3.8. Aplicando as duas etapas de Thinning [Ko et al., 2004].

3.2.2 Topologia

Depois de obtido esqueleto do mapa, os pontos que o compõem são projetados no mapa do SLAM e o Algoritmo 4 é aplicado para definir regiões no mapa, sendo cada região representada por um nodo no mapa topológico. As regiões são definidas por áreas circulares com centros nas coordenadas definidas nos nodos e raio que pode ser ajustado para uma melhor representação do mapa. Esses pontos centrais serão considerados pela função de utilidade para definir a próxima ação do robô. Cada nodo possui 3 informações relevantes: (i) um ID, que serve como um identificador, diferenciando um nodo de outro, (ii) uma variável que contabiliza a incerteza retornada pelo DP-SLAM naquela região, (iii) as coordenadas da região, um par (x, y) nas coordenadas do mapa do SLAM, determinando a posição daquela região no mapa.

A primeira linha do Algoritmo 4 percorre todos os pontos do esqueleto da imagem. Se o mapa topológico estiver vazio, as linhas 2 e 3 inserem o ponto do esqueleto sendo analisado no mapa topológico. Se não estiver vazio, as linhas 5 a 9 verificam se existe algum outro nodo do mapa topológico próximo do ponto sendo analisado. Caso não exista, as linhas 12 e 13 adicionam esse ponto no mapa, senão o algoritmo recomeça

Algorithm 4 Geração do Mapa Topológico

```
Require: esqueleto[] {Os pontos do esqueleto.}
 1: for i = 1 to esqueleto.size() do
 2:   if mapa_topologico.is_empty() then
 3:     mapa_topologico.insert(esqueleto[i]);
 4:   else
 5:     for e = 1 to mapa_topologico.size() do
 6:       D1 = esqueleto[i];
 7:       D2 = mapa_topologico[e];
 8:       if distance(D1, D2) < DIST_MIN then
 9:         break;
10:       end if
11:     end for
12:     if e = mapa_topologico.size() then
13:       mapa_topologico.insert(esqueleto[i]);
14:     end if
15:   end if
16: end for
```

com o próximo ponto do esqueleto. Durante a navegação do robô, o Algoritmo 4 é executado sempre que o robô atinge o seu alvo. Isso se deve ao fato de que cada nova informação adicionada ao mapa do SLAM pode significativamente alterar o conteúdo do mapa em si, fazendo com que o o mapa topológico atual fique ultrapassado.

A Figura 3.9 mostra um mapa do DP-SLAM onde o esqueleto já foi calculado e os pontos projetados sobre o mapa. O Algoritmo 4 é então aplicado e a Figura 3.10 é o resultado da primeira iteração. Nesta figura um novo nodo foi adicionado ao mapa topológico com as coordenadas do primeiro ponto do esqueleto. Uma região circular é gerada com centro nesse ponto, como demonstrado na Figura 3.11, e todas as informações do mapa do DP-SLAM nessa região são armazenadas. Para encontrar o próximo ponto do mapa topológico, os pontos do esqueleto são verificados até que seja encontrado um ponto a uma distância mínima de todos os pontos que compõem o mapa topológico, como é possível ver na Figura 3.12. E na Figura 3.13 é possível ver o mapa topológico completo.

3.3 Função de utilidade

O planejamento consiste em decidir regiões para o robô visitar de forma sequencial. Como o mapa completo do ambiente não é conhecido, esse plano é montado dinamicamente, sendo a próxima região a ser visitada escolhida apenas quando o robô chega na região atual. O plano calculado pelo sistema possui algumas características:

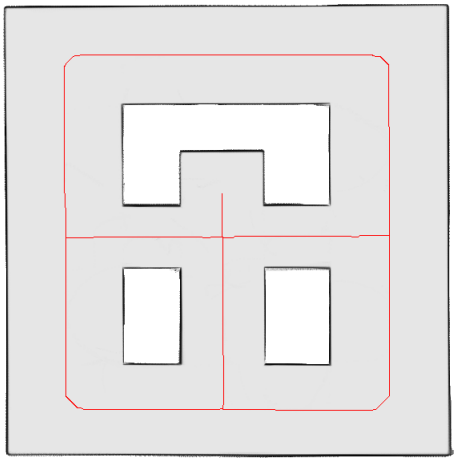


Figura 3.9. Esqueleto projetado sobre o mapa do DP-SLAM.

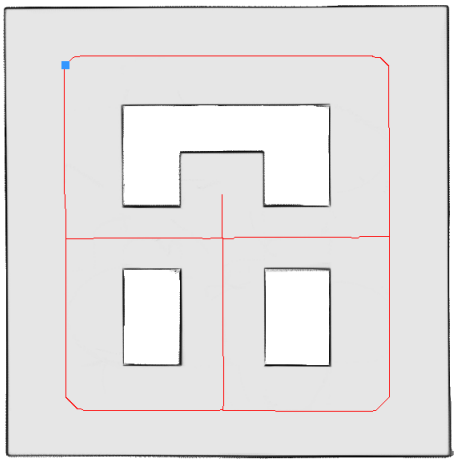


Figura 3.10. Nodo gerado no primeiro ponto do esqueleto.

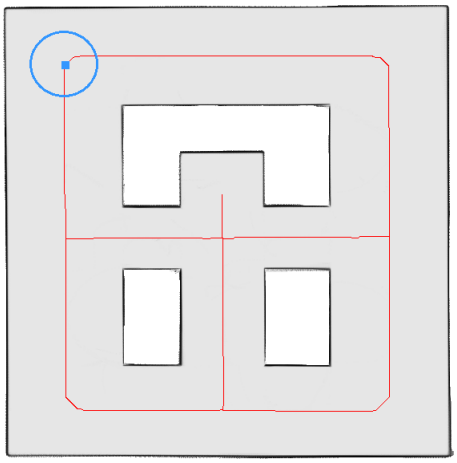


Figura 3.11. Região circular criada com centro no nodo do mapa topológico.

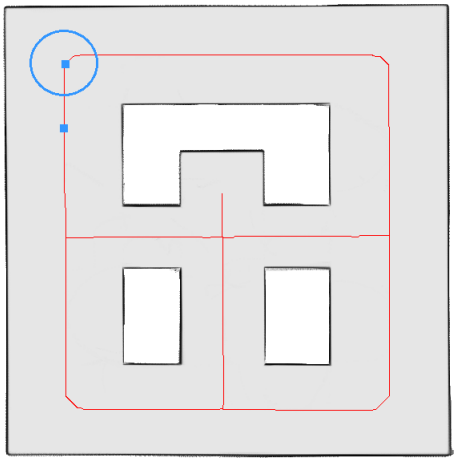


Figura 3.12. Próximo ponto escolhido para compor o mapa topológico.

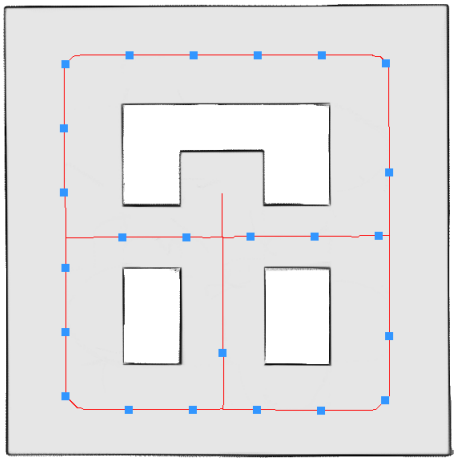


Figura 3.13. Mapa topológico completo.

- **As trajetórias executadas pelo robô não devem ser longas:** Como o mapa do ambiente não é conhecido, se faz necessário reavaliar o planejamento constantemente. Ao escolher visitar uma região mais próxima do robô, pode-se verificar mais rapidamente se a escolha foi boa ou não. Dessa forma, o sistema é capaz de se recuperar de uma decisão ruim em menos tempo.

- **Visitar a região escolhida deve melhorar o mapa:** Navegar até a região escolhida deve adicionar informação ao mapa, seja em forma de melhoria da qualidade ou exploração. O sistema é capaz de analisar informações sobre cada região do mapa e decidir qual região visitar, privilegiando a exploração do mapa, mas garantindo a qualidade do mapa.

- **A trajetória deve ser simples:** A trajetória da posição atual do robô até

a região escolhida deve ser de simples execução. Por simples, entende-se que o robô deve navegar preferencialmente em linha reta, evitando muitas curvas e, especialmente, evitando curvas muito fechadas, como curvas de 180 °. Como realizar curvas gera mais erros odométricos do que andar em linha reta, esse aspecto foi levado em conta quando o sistema decide a próxima região alvo.

De forma a obter essas características e baseando-se na informação contida no mapa topológico, a função de utilidade procura decidir qual região do mapa, definida pelos nodos do mapa topológico, o robô deve explorar. Para tanto, foi definida uma função de utilidade que procura explorar o ambiente eficientemente ao mesmo tempo em que se preocupa com a qualidade do mapa. Essa função de utilidade leva em conta uma série de fatores e pode ser descrita como:

$$U(x) = (D(x) + Err(x) - V(x)) * Q(x), \tag{3.1}$$

onde para cada região x, $U(x)$ é o valor de utilidade, $D(x)$ é a distância do posição atual do robô para o ponto central da região x, $Err(x)$ é a qualidade do mapa naquela região, $V(x)$ é a função que determina se o robô já passou por aquela região e $Q(x)$ é o fator do ângulo para aquela região. A região que o robô deve explorar como a próxima ação é aquela que retorna o maior valor de utilidade.

O primeiro fator, $Err(x)$, é a qualidade do mapa na região, relacionada com a incerteza do mapa naquela região. Para determinar esse valor, toda a região é analisada procurando por pontos na grade de ocupação em que a incerteza seja muito grande. Apesar de ser um fator importante para a construção de um bom mapa, ele não deve ser tratado com prioridade máxima, pois apenas essa informação não é determinante para um mapeamento eficiente, podendo fazer com que o robô escolha regiões que exijam um deslocamento complexo a partir da sua posição atual, ou ainda, que escolha alvos de forma a sempre voltar por caminhos já mapeados.

O segundo fator, $D(x)$, a ser levado em conta é a distância da posição atual do robô para cada ponto central das regiões presentes no mapa topológico, de forma a impedir que o robô explore alvos próximos à sua posição atual. Dessa forma tentamos incentivar a exploração do mapa. Entretanto, Makarenko et al. [2002] mostrou que evitar trajetórias muito longas produz melhores resultados no planejamento. Por este motivo, nós aplicamos um corte nas regiões que o robô pode explorar como próxima ação. Apenas regiões mais próximas ao robô serão analisadas pela função de utilidade.

Ainda no sentido de ampliar a exploração do mapa foi determinado um terceiro fator, $V(x)$: evitar as regiões já visitadas. Foi criada uma estrutura para acompanhar o movimento do robô por todo o mapa, de forma que, quando a possibilidade de visitar

uma região é calculada, verifica-se o robô já passou por aquela região. Esse fator é especialmente interessante em situações que o robô chega em uma encruzilhada que ele havia explorado previamente. Nesse caso, esse fator incentiva o robô a escolher um caminho que ele não tenha passado.

O último fator a ser levado em consideração, $Q(x)$, é relacionado orientação do robô. Basicamente, a intenção é evitar que o robô execute muitas curvas. Esse fator é importante pois erros de odometria ocorrem mais frequentemente em situações de rotação do que andando para frente. Por ser um fator que possui uma influência direta com a qualidade do mapa, esse fator foi considerado como crítico, e por isso, ele multiplica os outros três fatores.

Para determinar esse valor, inicialmente o mapa é dividido, a partir da posição atual do robô, em 4 setores. Em seguida, nós determinamos qual o ângulo que o robô está apontando e em quais setores se encontram as regiões do mapa topológico. O valor a ser multiplicado varia de acordo com o ângulo do robô e o setor da região em questão. A Figura 3.14, apresenta um apontando para o setor II. Nessa configuração temos:

- Regiões no setor II: fator = 1.0;
- Regiões nos setores I e III: fator = 0.5;
- Regiões no setor IV: fator = 0.0.

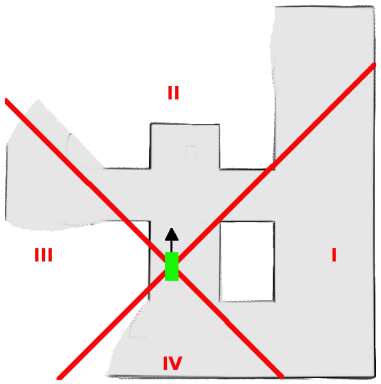


Figura 3.14. Mapa divido em 4 setores. O robô é representado pelo retângulo verde e as linhas vermelhas são as fronteiras dos setores.

3.4 Navegador ND

Para realizar a navegação do robô entre as regiões, nós usamos o *Nearness Diagram Navigation* (**ND**) [Minguez et al., 2004a,b], um algoritmo reativo para desvio de obstáculos para robôs navegando por ambientes complexos. O algoritmo procura identificar entidades no ambiente, tais como obstáculos e áreas livres, e tenta levar o robô de forma segura para a região escolhida. Essas entidades são utilizadas pelo algoritmo para definir um conjunto de situações em que o robô pode se encontrar e quais ações ele deve executar para contorná-las. As informações sensoriais obtidas pelo robô são utilizadas para identificar essas situações e as ações associadas a elas.

O navegador ND utiliza uma estratégia de divisão e conquista para identificar a situação atual do robô. Se o robô detecta algum obstáculo na sua zona de segurança ele entra no estado de *Low Safety*, senão ele entra no estado de High Safety. A zona de segurança é definida pela distância mínima que é permitido ao robô se aproximar de um obstáculo. O estado de *Low Safety* pode ser classificado em dois sub-estados: *Low Safety* 1, onde os obstáculos se encontram apenas em um dos lados da zona de segurança e *Low Safety* 2 quando existem obstáculos dos dois lados. O estado de High Safety também é classificado em sub-estados: *High Safety Goal in Region* quando o alvo está visível na área livre para o robô navegar, *High Safety Wide Region* quando a área para o robô navegar é ampla ou *High Safety Narrow Region* quando a área é estreita. Identificando em qual desses estados o robô se encontra, o navegador ND executa as ações associadas a cada situação.

Um ponto importante do navegador ND são os parâmetros. Para uma navegação suave do robô é importante escolher bem os parâmetros. Alguns parâmetros são relacionados com o robô, como as velocidades linear e angular máximas, formato do robô, etc. Outros parâmetros relacionam-se com questões de segurança, como o tamanho da zona de segurança ou tamanho mínimo para considerar uma área ampla ou estreito. Ainda existem alguns parâmetros de tempo que determinam por quanto tempo o robô deve executar uma determinada ação.

Depois que a função de utilidade escolhe para qual região o robô deve ir, essa informação é entregue ao navegador ND como um par (x, y) em coordenadas do mapa do SLAM. Essas coordenadas são então convertidas para coordenadas locais do robô e inseridas como objetivo para o navegador ND que fica responsável por levar o robô de forma segura ao objetivo.

3.5 Visão Geral do Sistema

Após a explicação de cada componente individualmente, o processo completo será mostrado, apresentando o resultado obtido com cada etapa do sistema. Na Figura 3.15 está o início do processo: o robô chegou na região alvo e terá que decidir uma nova região.

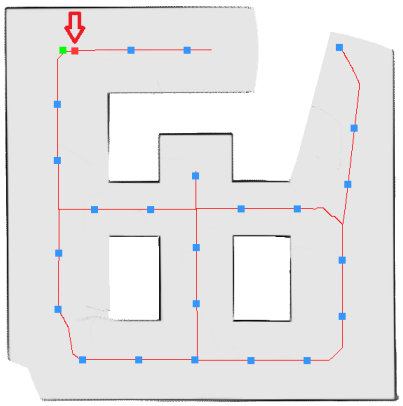


Figura 3.15. Situação do sistema quando o robô atinge a região alvo, indicada pela seta.

Nesse momento o mapa do DP-SLAM disponível para o sistema planejar a próxima ação está na Figura 3.16. O primeiro passo é obter o esqueleto do mapa, visível na Figura 3.17.

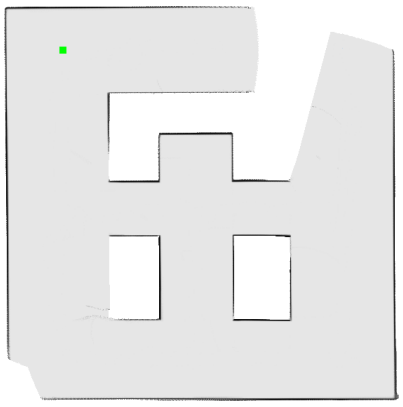


Figura 3.16. Mapa obtido pelo DP-SLAM, usado como base para planejar a próxima ação.

Ao obter o esqueleto, o próximo passo é aplicar o Algoritmo 4 nos pontos do esqueleto para gerar o mapa topológico. A Figura 3.18 demonstra o resultado dessa etapa. Com o mapa topológico gerado, a função de utilidade analisa as regiões de escolhe a melhor região a ser visitada (Figura 3.19).

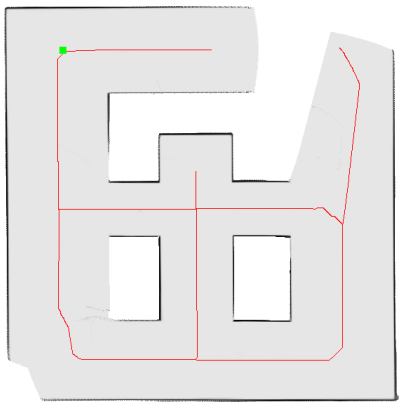


Figura 3.17. Esqueleto do mapa sobre o mapa do DP-SLAM.

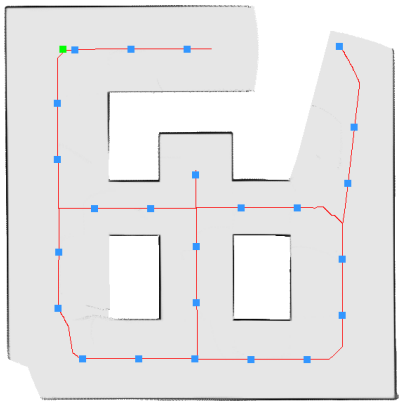


Figura 3.18. Mapa topológico obtido através do Algoritmo 4.

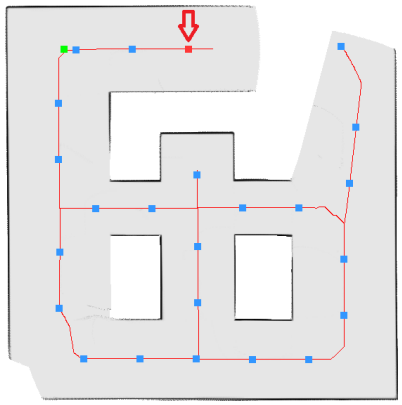


Figura 3.19. Região escolhida para ser explorada pelo robô, indicada pela seta.

Depois de escolhida qual região visitar, cabe ao navegador ND conduzir o robô até a região escolhida. A Figura 3.20 mostra a sequência de mapas gerados até que o robô atinja a região.

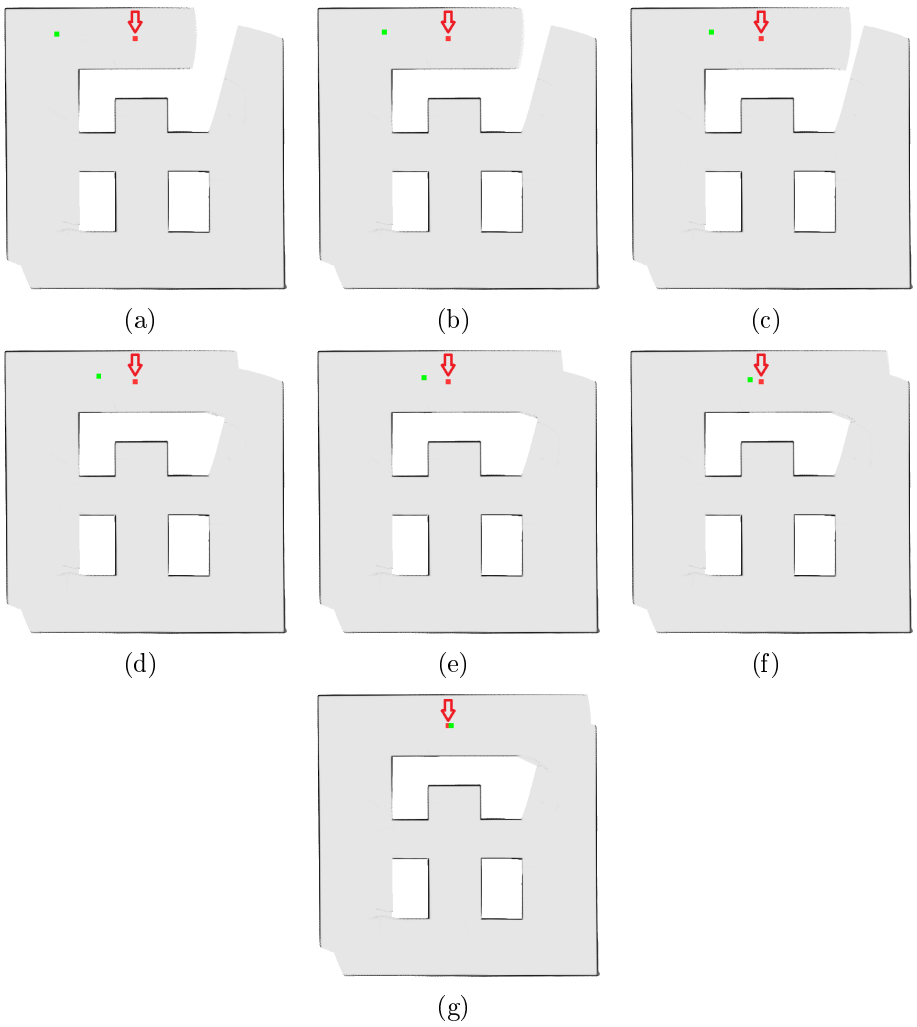


Figura 3.20. Sequência de mapas gerados enquanto o robô navegava para a região alvo.

Capítulo 4

Resultados

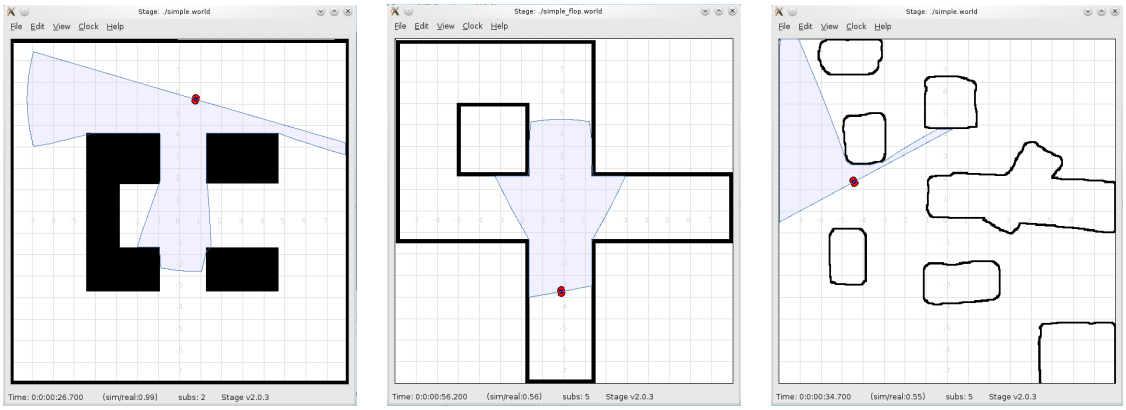
Para testar a eficiência do sistema, foram realizadas uma série de experimentos utilizando uma plataforma para simulações robóticas e foi testado o mesmo sistema em um robô real. O objetivo dos testes foi avaliar os benefícios de se usar uma política de navegação ao se mapear um ambiente genérico, sem marcos pré-definidos, quando comparada a uma navegação aleatória.

4.1 Experimentos Simulados

Os experimentos simulados foram realizados utilizando a plataforma Player/Stage [Gerkey et al., 2003]. Foram executados 20 testes usando a nossa política de navegação e 20 testes usando uma navegação aleatória. Nas simulações foi utilizado um Pioneer 2DX equipado com um laser Sick com orientação de 90° em relação à horizontal. Nas simulações foi aplicado uma função disponível no Player/Stage que simula erros de odometria. O erro simulado foi de três centímetros para as coordenadas (x, y) do robô e de cinco centímetros para a coordenada θ. Os ambientes utilizados para os testes simulados são apresentados na Figura 4.1.

A navegação aleatória foi implementada como um sistema reativo. O robô tenta apenas navegar em linha reta. Ao encontrar algum obstáculo, ele procura desviar do mesmo, de forma a não colidir. Quando se encontra a uma certa distância do obstáculo, o robô verifica qual lado está mais próximo ao obstáculo, e gira para o lado oposto. As constantes de velocidade e de desvio dos obstáculos foram as mesmas aplicadas para o Navegador ND, utilizado pela política de navegação da função de utilidade.

Para realizar a comparação entre a política de navegação e a navegação aleatória, foram analisadas a capacidade do robô de mapear o ambiente por completo, o número de ciclos do DP-SLAM necessários para mapear completamente o ambiente e



(a) (b) (c)

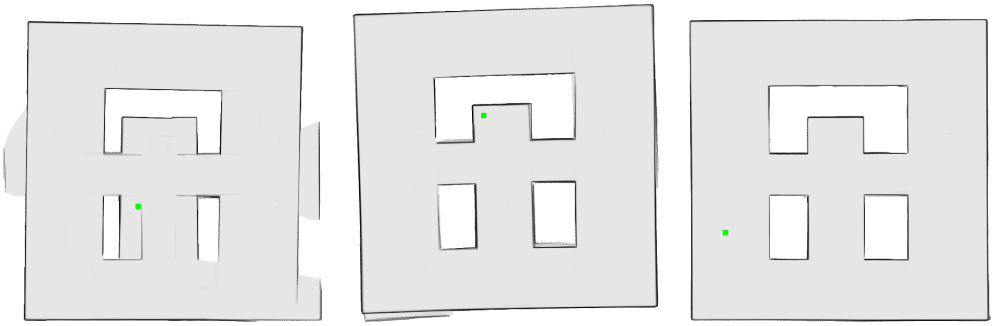
Figura 4.1. (a) Ambiente 1. (b) Ambiente 2. (c) Ambiente 3.

a qualidade do mapa obtido ao final do processo. A capacidade de mapear o ambiente por completo é auto-explicativa. Já o número de ciclos necessários para mapear o ambiente define quantos ciclos do DP-SLAM foram necessários para mapear todo o ambiente. Para esse teste, foi definido um limite máximo de 200 ciclos do DP-SLAM para o robô completar o mapa. Então, se o robô não conseguiu completar o mapa em um determinado teste, seu resultado nesse teste será 200.

A qualidade do mapa foi definida segundo a escala: "ruim" - "bom" - "excelente", segundo as quais os mapas são classificados empiricamente. Um mapa "ruim" define situações em que o robô não consegue se localizar corretamente no mapa, provocando situações em que paredes se sobrepõem. Já o mapa "bom" caracteriza situações em que paredes se sobrepõem no decorrer da exploração, mas mesmo assim o robô consegue se localizar e corrigir esses defeitos. Um mapa é denominado "excelente", quando do início ao fim da exploração o robô consegue se localizar e construir um mapa sem "defeitos". A Figura 4.2 ilustra esses três casos:

A Tabela 4.1 apresenta os resultados obtidos para as características de completude do mapa e do número de ciclos de DP-SLAM que foram necessários para completar os mapas, para a nossa política de navegação e para a navegação aleatória no Ambiente 1. Os valores na tabela representam a média obtida nas 20 execuções de cada navegação. É possível notar que o número de ciclos necessários para mapear todo o ambiente sofreu uma redução de 67% ao se utilizar a nosso política de navegação. Também é possível perceber que a nossa política de navegação conseguiu completar o mapa 16 vezes das 20 execuções, sendo que a navegação aleatória completou o mapa em apenas cinco das 20 vezes.

Os bons resultados podem ser explicados pela função de utilidade. Ela foi capaz



(a) (b) (c)

Figura 4.2. Alguns mapas gerados nos nossos experimentos: (a) mapa classificado como ruim, (b) mapa classificado como bom, (c) mapa classificado como excelente.

Tabela 4.1. Resultados da performance para o Ambiente 1.

	Ciclos do DP-SLAM	Mapas Completos
Política de Navegação	62,45 ± 10,27	16
Navegação Aleatória	186,3 ± 11,84	5

de analisar os mapas gerados e decidir qual a melhor opção de se visitar no momento para uma exploração eficiente do ambiente. Sem a função de utilidade, o robô navegava insistentemente por áreas já mapeadas do ambiente, na maioria das vezes não conseguindo completar o mapa. É importante ressaltar que como o sistema não possui um algoritmo de planejamento de caminhos completo, apenas um de navegação, em alguns casos o nosso sistema não foi capaz de fornecer uma boa trajetória para o robô. Por esse motivo, não foi possível atingir 100% de mapas completos usando a nossa política de navegação.

A Tabela 4.2 apresenta a classificação dos 20 mapas obtidos pelos dois sistemas de navegação. Pode ser observado que o nosso sistema de navegação foi capaz de produzir mapas de melhor qualidade em relação a navegação aleatória. Isso se deve ao fato de que a função de utilidade faz o balanço a exploração com a qualidade do mapa.

Tabela 4.2. Qualidade do Mapa para o Ambiente 1.

	Ruim	Bom	Excelente
Política de Navegação	2	8	10
Navegação Aleatória	7	6	7

É interessante notar que, apesar do uso da função de utilidade, nós ainda obtivemos mapas de qualidade ruim. Isso se deve ao fato de que, para executar o DP-SLAM

em tempo real foi necessário reduzir o número de partículas utilizadas pelo algoritmo. Nós utilizamos 100 partículas enquanto no trabalho original do DP-SLAM Eliazar & Parr [2003], os autores utilizaram 3000 partículas, permitindo a eles mapear ambientes maiores com melhor qualidade, fazendo o processamento de forma *off-line*.

Outra análise que foi realizada diz respeito à área mapeada pelos robôs em cada ciclo do DP-SLAM. Para isso, foi analisado em cada ciclo do DP-SLAM a quantidade de área livre encontrada pelo robô durante o mapeamento, e a partir destes dados foi calculada a média dos valores. O resultado dessa análise, para o Ambiente 1, pode ser vista na Figura 4.3.

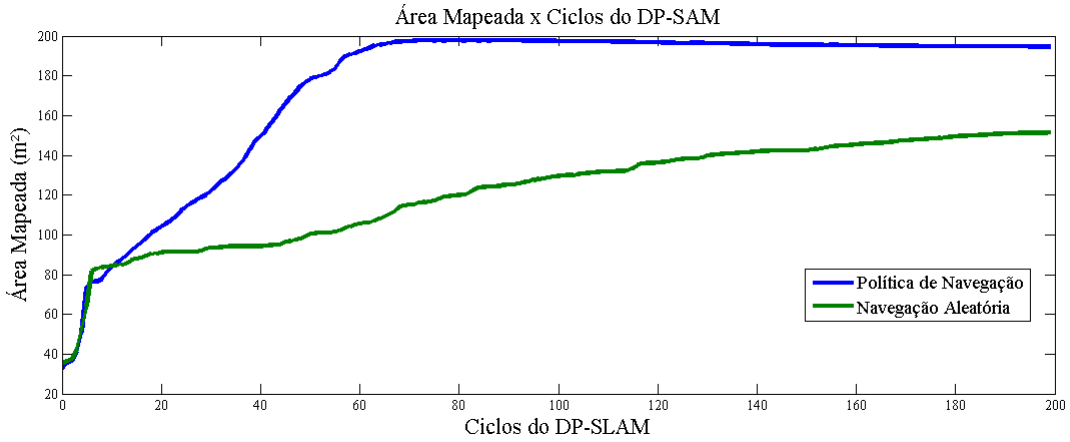


Figura 4.3. Área mapeada em metros quadrados pelo número de ciclos do DP-SLAM.

É possível perceber no gráfico que até o ciclo 10 a navegação aleatória possui uma ligeira vantagem sobre a nossa política de navegação. Como explicado no Capítulo 3, a nossa política de navegação mantém o robô estático enquanto calcula o mapa topológico enquanto que a navegação aleatória não possui essa necessidade. Por esse motivo, inicialmente, a navegação aleatória consegue mapear mais, mas logo em seguida, a nossa política de navegação consegue superar, mapeando mais rapidamente uma maior área do ambiente.

As Tabelas 4.3, 4.5, 4.4 e 4.6 fazem a comparação da política de navegação com a navegação aleatória para os Ambientes 2 e 3. Esses cenários apresentaram resultados similares aos do Ambiente 1. A política de navegação foi capaz de produzir mais mapas completos e com uma média de ciclos do DP-SLAM menor. A qualidade do mapa também foi superior para os dois ambientes.

Tabela 4.3. Resultados da performance para o Ambiente 2.

	Ciclos do DP-SLAM	Mapas Completos
Política de Navegação	115,9 ± 58,41	16
Navegação Aleatória	170,5 ± 59,39	7

Tabela 4.4. Qualidade do Mapa para o Ambiente 2.

	Ruim	Bom	Excelente
Política de Navegação	0	3	17
Navegação Aleatória	3	6	11

Tabela 4.5. Resultados da performance para o Ambiente 3.

	Ciclos do DP-SLAM	Mapas Completos
Política de Navegação	134,2 ± 32,41	11
Navegação Aleatória	196,45 ± 45,75	1

4.1.1 Função de Utilidade

Ainda foram realizados outros experimentos simulados, de forma a testar especificamente a função de utilidade. Para realizar esses testes a função de utilidade foi alterada de forma a apenas considerar cada componente dela de forma individual, e ainda algumas combinações entre os componentes. Ao total foram testadas nove diferentes funções de utilidade e em cada uma delas foram testadas as três características citadas anteriormente.

Para comprovar a eficiência da função de utilidade, foram realizados alguns testes utilizando apenas alguns termos da nossa função. Inicialmente, foi cortado apenas o limite de distância das regiões que podem ser escolhidas. Agora, qualquer região do mapa topológico pode ser escolhida para ser visitada. As Tabelas 4.7 e 4.8 comparam essa nova função com a política de navegação e com a navegação aleatória.

Ao realizar o corte de regiões muito afastadas do robô, o sistema está se baseando na idéia proposta por Makarenko et al, [2002], que aplicaram na sua função uma penalidade para trajetórias muito longas. Como nosso trabalho não envolve determinação

Tabela 4.6. Qualidade do Mapa para o Ambiente 3.

	Ruim	Bom	Excelente
Política de Navegação	3	7	10
Navegação Aleatória	5	5	10

Tabela 4.7. Resultados da performance sem o corte de regiões.

	Ciclos do DP-SLAM	Mapas Completos
Política de Navegação	62,45 ± 10,27	16
Navegação Aleatória	186,3 ± 11,84	5
Política sem Limite	138,8 ± 60,98	5

Tabela 4.8. Qualidade do Mapa sem o corte de regiões.

	Ruim	Bom	Excelente
Política de Navegação	2	8	10
Navegação Aleatória	7	6	7
Política sem Limite	5	8	7

de trajetórias, e sim escolha de regiões, o método de evitar longas trajetórias foi limitar o número de regiões para o planejador escolher. Ao analisar as duas tabelas, é possível perceber que limitar o número de regiões que o robô pode visitar é importante, tanto para a qualidade do mapa, como para a eficiência da exploração. Como explicado na Seção 3.3, isso acontece pois o sistema não possui o mapa completo do ambiente e este está em constante atualização. Ao escolher uma região muito distante do robô, ignora-se toda a informação que está entrando no sistema durante a navegação do robô. Por este motivo, ao decidir por regiões mais próximas do robô, estaremos constantemente reavaliando o nosso plano, podendo nos recuperar de uma ação errada mais rapidamente. Ainda há o problema do navegador, que para trajetórias muito longas, pode não guiar o robô por um caminho favorável. Por causa dos resultados obtidos neste teste, todos os outros de performance da função de utilidade foram realizados utilizando o limite de distância das regiões.

A função de utilidade foi determinada de forma empírica, com base em alguns trabalhos da literatura. Como explicado no Capítulo 2, para Feder et al.,[1999] o objetivo da função era basicamente escolher uma ação que aumentasse a quantidade de informação do sistema. Essa premissa foi a base da primeira versão da função de utilidade, que privilegiava apenas a qualidade do mapa: $U(x) = Err(x)$. Essa função, como pode ser visto na Tabela 4.9, possuía o mesmo problema do trabalho de Feder et al.,[1999], tende a navegar sempre por regiões já conhecidas, o que pode ser confirmado pelo fato de não completar nenhum mapa.

A idéia de Bourgault et al.,[2002] de incluir um termo para exploração tentava resolver o problema do robô navegar apenas em lugares já explorados apresentou resultados melhores para a exploração de ambientes. Para incluir esse termo na função de

utilidade foi utilizada a distância do robô para as regiões a serem exploradas. A função $U(x) = (D(x) + Err(x)) * Q(x)$ é uma adaptação da função utilizada por Bourgault et al.,[2002] para o sistema.

Diversos trabalhos na literatura utilizam uma técnica chamada *frontier based exploration* Burgard et al. [2000]; Yamauchi [1998]; Fox et al. [2006]. No caso dessa técnica, o robô procura sempre se locomover para regiões onde exista a fronteira entre áreas que ele conhece e áreas que ele não conhece. Basicamente, o algoritmo procura conduzir o robô para lugares que ele não explorou. Para tentar aproveitar essa idéia na função de utilidade, foi incluído o termo $V(x)$, e este em conjunto com o termo $D(x)$, é uma adaptação desse algoritmo para o sistema.

O último fator, relacionado com o ângulo do robô e a posição da região escolhida, foi proposto para resolver um problema que surgiu durante a realização dos testes da função. Notou-se que mapas de qualidade inferior eram gerados mais frequentemente nos testes onde o robô realizava muitas mudanças de direção do que nos testes onde o robô navegava a maior parte do tempo em linha reta.

Tabela 4.9. Resultados da performance para diferentes funções de utilidade.

	Ciclos do DP-SLAM	Mapas Completos
$U(x) = (D(x) + Err(x) - V(x)) * Q(x)$	62,45 ± 10,27	16
$U(x) = D(x)$	169,2 ± 31,83	3
$U(x) = Err(x)$	200 ± 0	0
$U(x) = V(x)$	180,8 ± 43,32	3
$U(x) = Q(x)$	138,05 ± 47	11
$U(x) = (D(x) + Err(x)) * Q(x)$	121,6 ± 53,51	13
$U(x) = (D(x) - V(x)) * Q(x)$	122,45 ± 56,60	12
$U(x) = (Err(x) - V(x)) * Q(x)$	182,3 ± 22,52	3
$U(x) = (D(x) + Err(x) - V(x))$	154,25 ± 47,75	5

Ao analisar as Tabelas 4.9 e 4.10 é possível notar algumas características do nosso sistema. A função completa, com todos os termos, consegue balancear a exploração do ambiente com a qualidade do mapa. Ao se retirar um termo da função pode-se perder esse equilíbrio. Por exemplo, a função $U(x) = (Err(x) - V(x)) * Q(x)$ consegue completar apenas 3 mapas e o número de ciclos do DP-SLAM aumenta consideravelmente, enquanto que na questão da qualidade do mapa, essa função possui um dos melhores desempenhos.

É interessante notar que alguns fatores influenciam mais que outros para certas características. A função $U(x) = Err(x)$, apresenta os melhores resultados de qualidade de mapa, mas por outro lado, não foi capaz de completar nenhum mapa

Tabela 4.10. Resultados da qualidade do mapa para diferentes funções de utilidade.

	Ruim	Bom	Excelente
$U(x) = (D(x) + Err(x) - V(x)) * Q(x)$	2	8	10
$U(x) = D(x)$	5	8	7
$U(x) = Err(x)$	2	4	14
$U(x) = V(x)$	6	7	7
$U(x) = Q(x)$	4	7	9
$U(x) = (D(x) + Err(x)) * Q(x)$	5	7	8
$U(x) = (D(x) - V(x)) * Q(x)$	6	9	5
$U(x) = (Err(x) - V(x)) * Q(x)$	4	6	10
$U(x) = (D(x) + Err(x) - V(x))$	3	9	8

completamente. A função $U(x) = Q(x)$ demonstra que o fator do ângulo é importante tanto para a exploração do ambiente como para a qualidade do mapa, enquanto que a função $U(x) = V(x)$ produziu os piores resultados no balanço das duas características.

4.2 Experimentos Reais

O sistema também foi testado em um robô real, um Pioneer P3-AT equipado com um Laser SICK (Figura 4.4). O sistema rodou em um processador Pentium M(1.73GHz) e 512MB de memória, com Ubuntu, para o primeiro teste e em um processador Core 2 Duo(2.GHz) e 2GB de memória para os outros testes. Os testes foram realizados nos corredores em frente ao nosso laboratório.

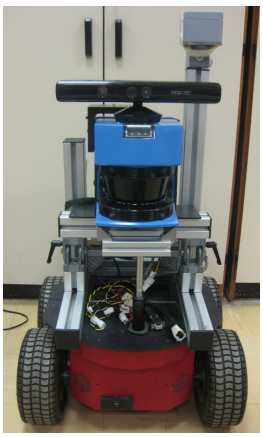


Figura 4.4. Pioneer P3-AT usado no experimento.

Para realizar os testes reais foram necessários alguns ajustes em constantes do sistema. Primeiramente foi preciso adaptar a velocidade do robô para se adequar ao

tempo de cálculo do DP-SLAM. Um parâmetro que precisou receber atenção especial foi a distância para o desvio de obstáculos. Nos testes reais a distância permitida ao robô de se aproximar dos obstáculos foi maior que nas simulações por questão de segurança. Outro parâmetro alterado foi a distância mínima para considerar que o robô chegou na região a ser explorada. Isso se deve pela variação dos tamanhos do robô real e do simulado.

Foram realizados 3 testes reais. As trajetórias percorridas pelo robô em cada teste podem ser vistas na Figura 4.5. A Tabela 4.11 apresenta os resultados para cada teste. A primeira coluna possui o número de ciclos do DP-SLAM que foram necessários para gerar o mapa do percurso. A segunda coluna apresenta a duração aproximada, em minutos, do percurso feito pelo robô e a terceira coluna a distância aproximada, em metros, percorrida pelo robô. A quarta coluna possui o número de vezes em que o robô calculou um novo alvo para visitar.

	Ciclos do DP-SLAM	Duração do percurso (min)	Distância percorrida (m)	Número de Planejamentos
Teste 1	18	≈ 18	≈ 15	3
Teste 2	68	≈ 60	≈ 35	9
Teste 3	48	≈ 50	≈ 30	8

Tabela 4.11. Resultados para os testes reais.

As Figuras 4.6, 4.7 e 4.8 apresentam uma sequência de mapas obtidos na realização dos testes reais. Pode-se observar que o robô é capaz de mapear o ambiente conforme esperado, se comportando como nos experimentos simulados.

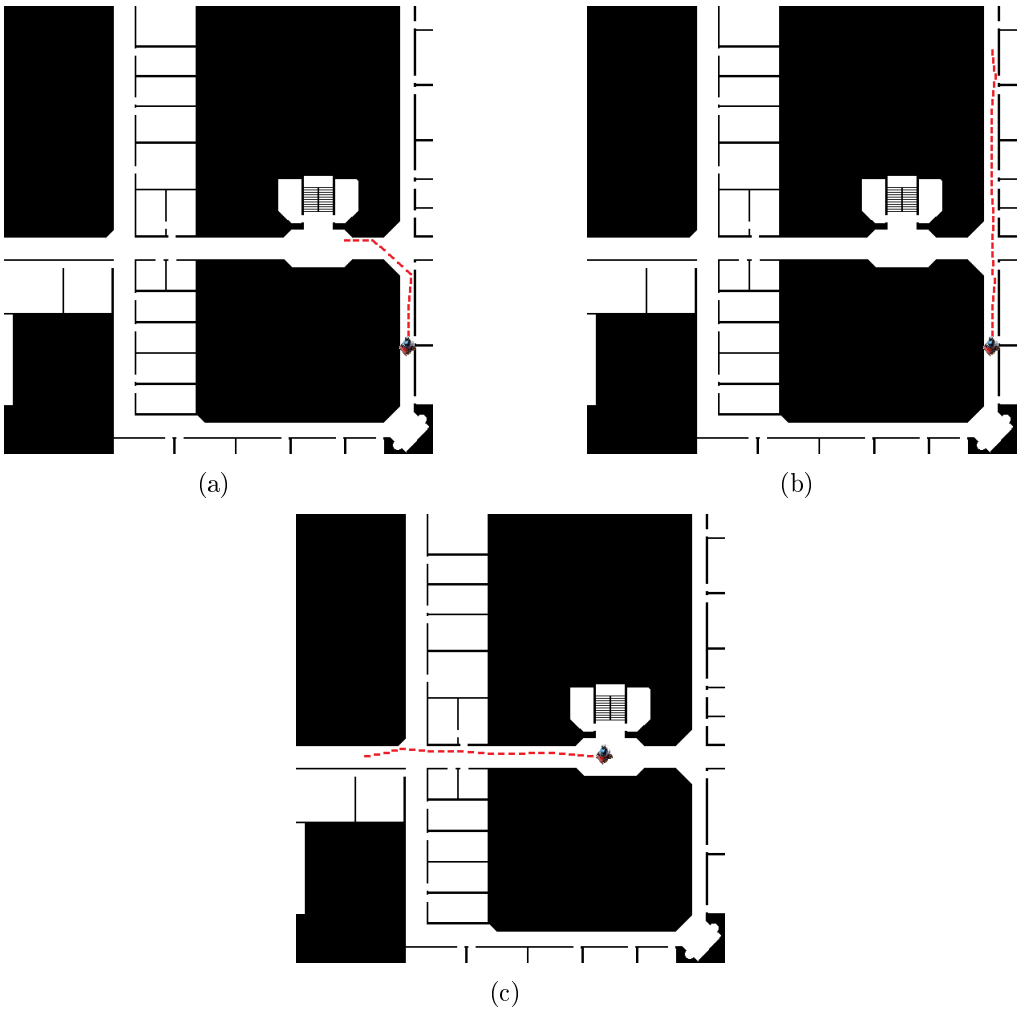


Figura 4.5. Trajetórias realizadas pelo robô. (a) Teste 1, (b) Teste 2, (c) Teste 3.

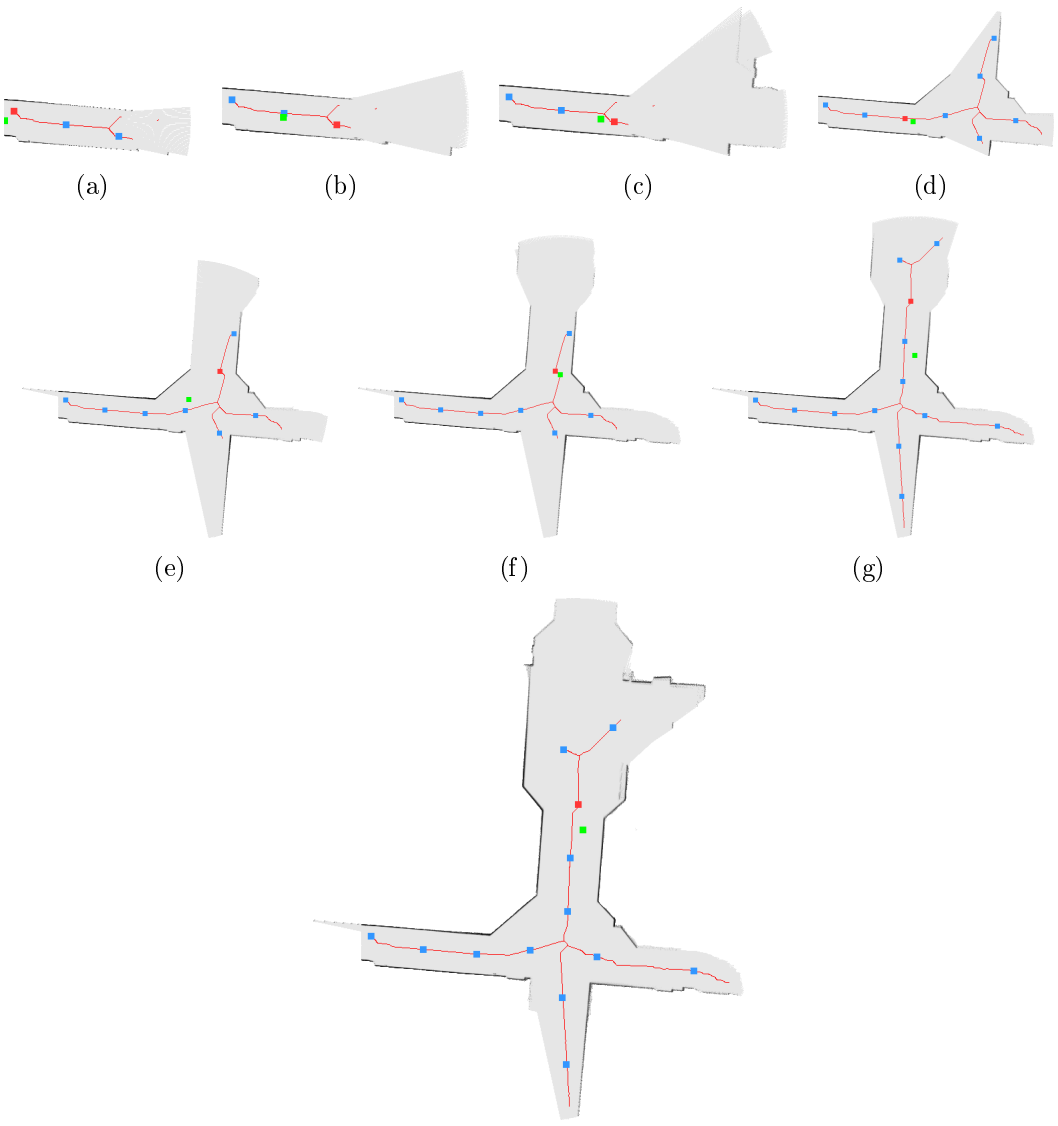
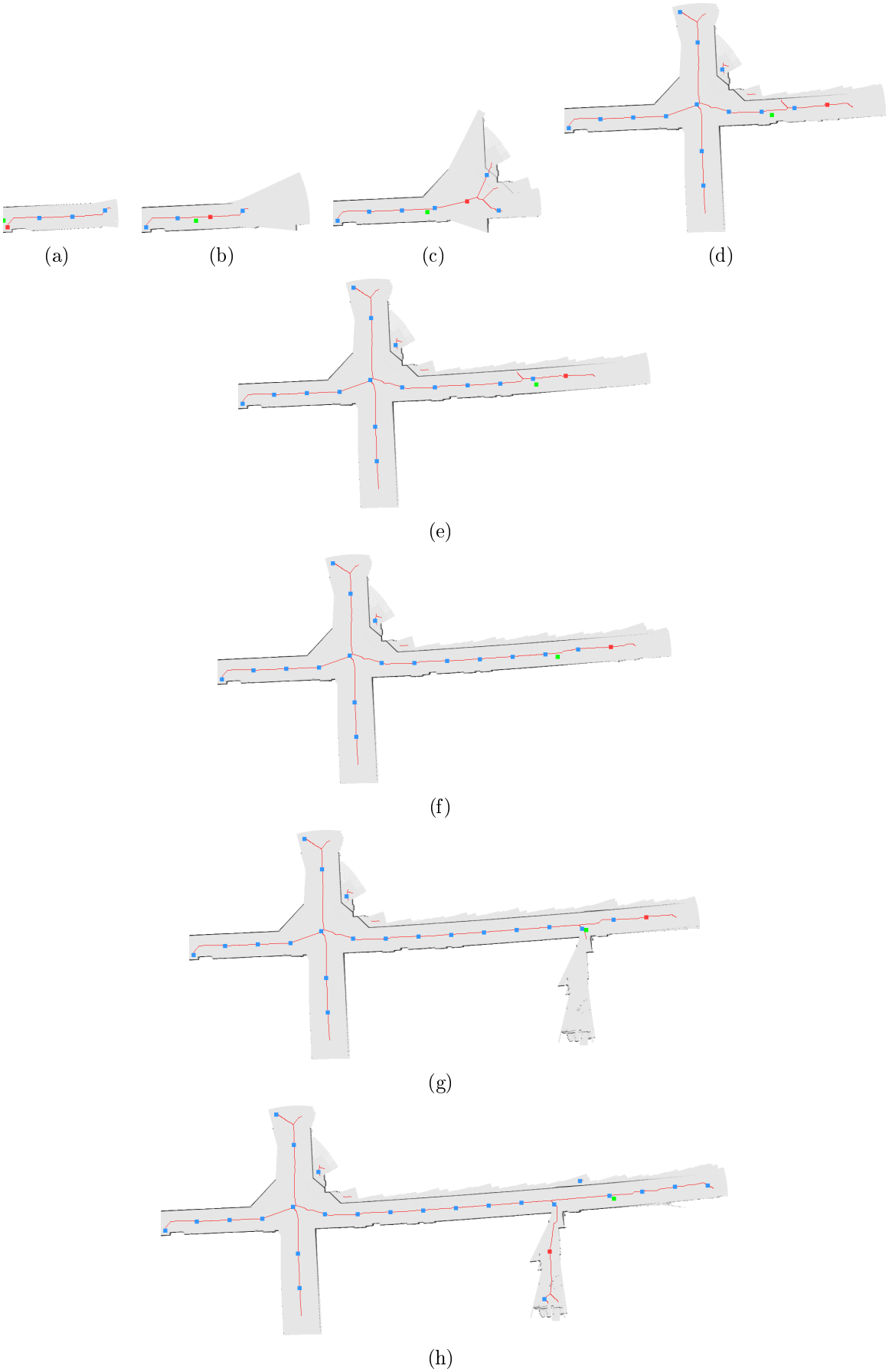


Figura 4.6. Sequência de mapas gerados no primeiro teste real.



(a) (b) (c) (d) (e) (f) (g) (h)

Figura 4.7. Sequência de mapas gerados no segundo teste real.

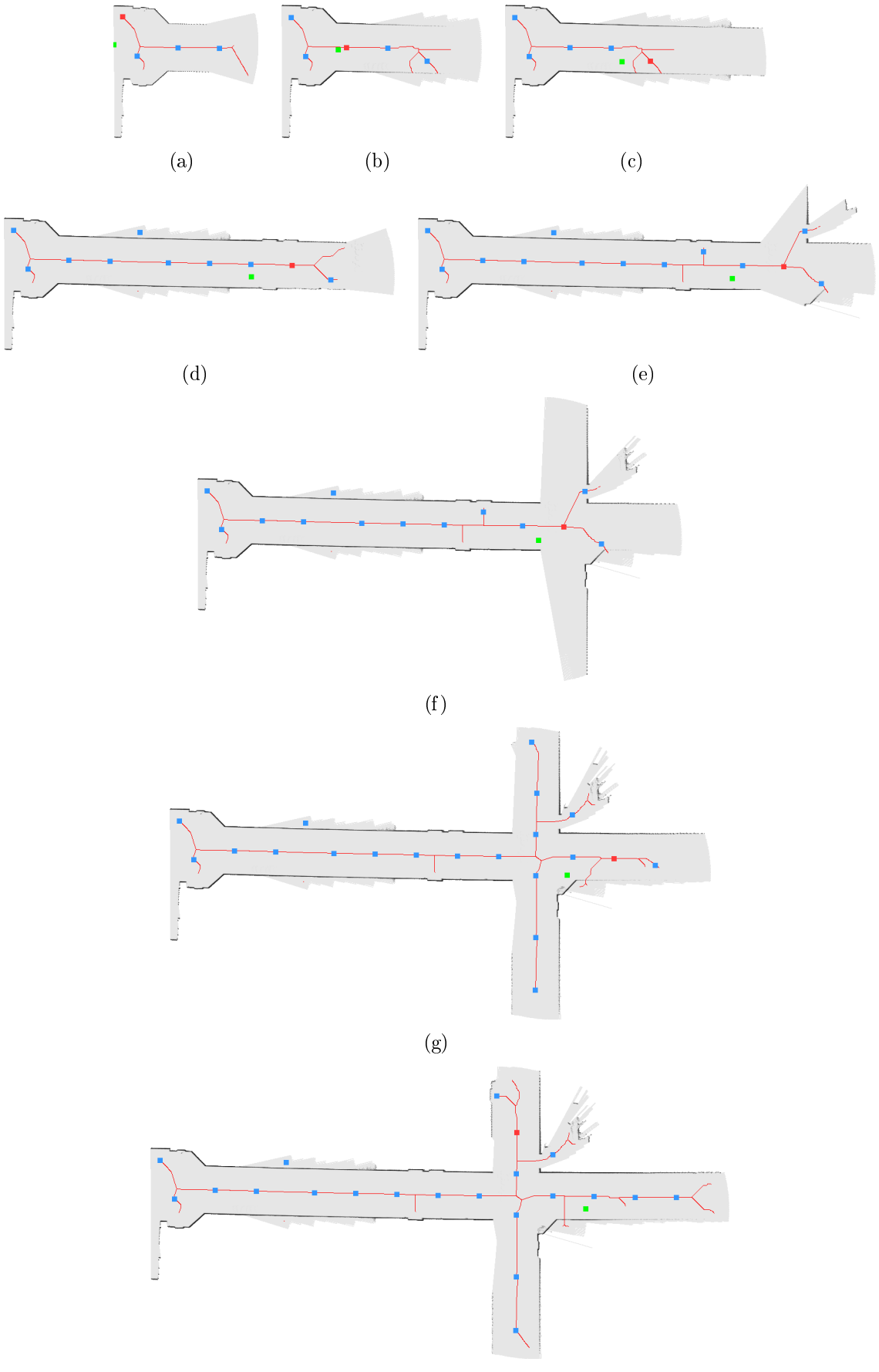


Figura 4.8. Sequência de mapas gerados no terceiro teste real.

Capítulo 5

Conclusões e Trabalhos Futuros

5.1 Conclusões

Esta dissertação apresentou uma nova abordagem para a exploração de ambientes desconhecidos. Através do desenvolvimento de uma política de navegação aplicada em conjunto com o DP-SLAM a nova abordagem foi capaz de gerar planos e guiar o robô pelo ambiente. A abordagem aplica mapas topológicos gerados dinamicamente sobre o mapa métrico do DP-SLAM, que servem como a base para o nosso planejamento. Foi desenvolvida uma função de utilidade que, baseada nas informações do mapa topológico, decide qual a melhor região do mapa o robô deve explorar.

Como foi demonstrado no Capítulo 4, a nossa abordagem se mostrou mais eficiente tanto na exploração como na qualidade do mapa quando comparada com uma navegação aleatória por parte do robô. Ainda foram testados diferentes versões da nossa função de utilidade, fazendo um paralelo com alguns trabalhos da literatura, em que é possível notar a importância de cada termo que a compõe.

O navegador do nosso sistema em alguns dos testes feitos não conseguiu conduzir o robô para as regiões escolhidas pela função de utilidade. Como o nosso navegador não planeja caminhos, apenas conduz o robô em segurança para a região escolhida, algumas vezes o robô não foi capaz de alcançar a região alvo.

Também foram realizados testes em robôs reais para verificar se o sistema poderia funcionar em situações reais. Com apenas alguns ajustes no algoritmo usado para as simulações, foi possível verificar que o sistema funcionando com um robô real se comportava como no robô simulado, possibilitando seu uso para situações reais.

Em suma, as principais contribuições desta dissertação são o uso de mapas topológicos sobre o mapa métrico do DP-SLAM para usar como fonte de informações sobre o estado do mapa e o desenvolvimento da função de utilidade para um algoritmo de

SLAM que não utiliza marcos.

5.2 Trabalhos Futuros

Existem várias possibilidades para o aprimoramento da abordagem descrita. Algumas sugestões são listadas abaixo:

- Alterar a obtenção de dados sensoriais por parte do DP-SLAM para tentar sincronizar a velocidade do processamento do mesmo com a velocidade de navegação do robô. Essa sincronização pode permitir o uso de mais partículas no DP-SLAM melhorando a qualidade do mapeamento.

- Implementar um algoritmo de detecção de *loop closure* para o DP-SLAM, para poder se recuperar melhor de erros no mapa e utilizar essa informação no planejamento do sistema.

- Utilizar um algoritmo de planejamento de caminhos para ser usado em conjunto com o navegador ND para melhorar o processo de navegação do robô. Esse algoritmo deve aumentar o número de mapas completados pelo robô.

- Estudar diferentes termos para a função de utilidade para tentar melhorar os resultados obtidos. Seria interessante tentar abstrair algumas estratégias da literatura, como a *frontier based exploration*, para o nosso sistema e testar os seus valores.

- Aplicar a abordagem descrita nesta dissertação para mais de um robô. Nesse caso haveria troca de informações entre os robôs podendo ser apenas localização dos robôs, troca de informações sobre o mapa, etc.

Referências Bibliográficas

Angeli, A.; Doncieux, S.; Meyer, J.-A. & Filliat, D. (2009). Visual topological slam and global localization. In *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, ICRA'09, pp. 2029--2034, Piscataway, NJ, USA. IEEE Press.

Aurenhammer, F. (1991). Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23:345--405.

Ballard, D. & Brown, C. (1982). *Computer Vision*. Prentice-Hall.

Bourgault, F.; Makarenko, A.; Williams, S.; Grocholsky, B. & Durrant-Whyte, H. (2002). Information based adaptive robotic exploration. In *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pp. 540 – 545.

Burgard, W.; Moors, M.; Fox, D.; Simmons, R. & Thrun, S. (2000). Collaborative multi-robot exploration. In *IEEE Int. Conf. Robotics Automation (ICRA)*.

Dissanayake, M.; Newman, P.; Clark, S.; Durrant-Whyte, H. F. & Csorba, M. (2001). A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on Robotics and Automation*, 17(3):229–241.

Doucet, A.; de Freitas, N. & Gordon, N. (2001). *Sequential Monte Carlo Methods in Practice*. Berlin: Springer-Verlag.

Doucet, A.; de Freitas, N.; Murphy, K. & Russell, S. (2000). Rao-blackwellised particle filtering for dynamic bayesian networks. In *Uncertainty in Artificial Intelligence (UAI)*.

Eliazar, A. (2005). *DP-Slam*. PhD thesis, Departament of Computer Science, Duke University.

Eliazar, A. & Parr, R. (2003). Dp-slam: Fast, robust simultaneous locazation and mapping without predetermined landmarks. In *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI)*, pp. 1135 – 1142.

Eliazar, A. & Parr, R. (2004). Dp-slam 2.0. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 2, pp. 1314 – 1320.

Feder, H. J. S.; Leonard, J. J. & Smith, C. M. (1999). Adaptive mobile robot navigation and mapping. *International Journal of Robotics Research*, 18(7):650 – 668.

Fox, D.; Ko, J.; Konolige, K.; Limketkai, B.; Schulz, D. & Stewart, B. (2006). Distributed multirobot exploration and mapping. *Proceedings of the IEEE*, 94(7):1325 –1339.

Gerkey, B. P.; Vaughan, R. & Howard, A. (2003). The player/stage project: Tools for multi-robot and distributed sensor systems. In *11th International Conference on Advanced Robotics*, pp. 317 – 323.

Gutmann, J. & Konolige, K. (2000). Incremental mapping of large cyclic environments. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*.

Huang, S.; Kwok, N.; Dissanayake, G.; Ha, Q. & Fang, G. (2005). Multi-step look-ahead trajectory planning in slam: Possibility and necessity. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1091 – 1096.

Ko, B.-Y.; Song, J.-B. & Lee, S. (2004). Real-time building of a thinning-based topological map with metric features. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 2, pp. 1524 –1529.

Kuipers, B.; Modayil, J.; Beeson, P.; MacMahon, M. & Savelli, F. (2004). Local metrical and global topological maps in the hybrid spatial semantic hierarchy. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Lee, Y.-J.; Kwon, T.-B. & Song, J.-B. (2007). Slam of a mobile robot using thinning-based topological information. In *International Journal of Control, Automation, and Systems*, volume 5, pp. 577 – 583.

Leung, C.; Huang, S. & Dissanayake, G. (2006a). Active slam using model predictive control and attractor based exploration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5026 – 5031.

Leung, C.; Huang, S. & Dissanayake, G. (2008). Active slam for structured environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1898 – 1903.

Leung, C.; Huang, S.; Kwok, N. M. & Dissanayake, G. (2006b). Planning under uncertainty using model predictive control for information gathering. *Robotics and Autonomous Systems*, pp. 898–910.

Makarenko, A. A.; Williams, S.; Bourgault, F. & Durrant-Whyte, H. (2002). An experiment in integrated exploration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1.

Minguez, J.; Member, A. & Montano, L. (2004a). Nearness diagram (nd) navigation: Collision avoidance in troublesome scenarios. *IEEE Transactions on Robotics and Automation*, 20.

Minguez, J.; Osuna, J. & Montano, L. (2004b). A divide and conquer strategy based on situations to achieve reactive collision avoidance in troublesome scenarios. In *IEEE International Conference on Robotics and Automation (ICRA 2004)*.

Montemerlo, M.; Thrun, S.; Koller, D.; & Wegbreit, B. (2003). Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pp. 1151 – 1156.

Montemerlo, M.; Thrun, S.; Koller, D. & Wegbreit, B. (2002). Fastslam: a factored solution to the simultaneous localization and mapping problem. In *Eighteenth national conference on Artificial intelligence*, pp. 593 – 598.

Paskin, M. (2003). Thin junction tree filters for simultaneous localization and mapping. In *Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*.

Smith, R.; Self, M. & Cheeseman, P. (1990). Estimating uncertain spatial relationships in robotics. In *Autonomous Robot Vehicles*, pp. 167 – 193.

Stachniss, C.; Hahnel, D. & Burgard, W. (2004). Exploration with active loop-closing for fastslam. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pp. 1505 – 1510.

Thrun, S. (2000). Probabilistic algorithms in robotics. *AI Magazine*, 21:93–109.

Thrun, S.; Burgard, W. & Fox, D. (2006). *Probabilistics Robotics*. The MIT Press, first edição.

Thrun, S.; Gutmann, J. S.; Fox, D.; Burgard, W. & Kuipers, B. (1998). Integrating topological and metric maps for mobile robot navigation: A statistical approach. In *Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pp. 989–995.

Victorino, A. & Rives, P. (2006). Slam with consistent mapping in an hybrid model. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 3575 –3580.

Yamauchi, B. (1997). A frontier-based approach for autonomous exploration. In *In Proceedings of the IEEE International Symposium on Computational Intelligence, Robotics and Automation*, pp. 146--151.

Yamauchi, B. (1998). Frontier-based exploration using multiple robots. In *Second Int. Conf. Autonomous Agents*.

Yuen, D. C. & MacDonald, B. A. (2004). Theoretical considerations of multiple particle filters for simultaneous localisation and map-building. In Negoita, M. G.; Howlett, R. J. & Jain, L. C., editores, *Knowledge-Based Intelligent Information and Engineering Systems*, volume 3213 of *Lecture Notes in Computer Science*, pp. 203–209. Springer Berlin / Heidelberg.