UNIVERSIDADE FEDERAL DA BAHIA
ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM
ENGENHARIA ELÉTRICA

JOÃO LUIZ CARNEIRO CARVALHO

CONTRIBUIÇÕES AO PROBLEMA DE LOCALIZAÇÃO PARA VEÍCULOS TERRESTRES NÃO TRIPULADOS

Salvador
2023

UFBA

ESCOLA POLITÉCNICA

CONTRIBUIÇÕES AO PROBLEMA DE LOCALIZAÇÃO PARA VEÍCULOS TERRESTRES NÃO TRIPULADOS

TESE DE DOUTORADO

João Luiz Carneiro Carvalho

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, PPGEE, da Universidade Federal da Bahia, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia Elétrica.

Orientadores: Paulo C. M. de Abreu Farias
Eduardo F. de Simas Filho

Salvador
Maio de 2023

Carvalho, João Luiz Carneiro

CONTRIBUIÇÕES AO PROBLEMA DE LOCALIZAÇÃO PARA VEÍCULOS TERRESTRES NÃO TRIPULADOS/João Luiz Carneiro Carvalho. – Salvador: UFBA/PPGEE, 2023.

XIX, 161 p.: il.; 29, 7cm.

Orientadores: Paulo C. M. de Abreu Farias

Eduardo F. de Simas Filho

Tese (doutorado) – UFBA/PPGEE/Programa de Engenharia Elétrica, 2023.

Referências Bibliográficas: p. 134 – 142.

I. Farias, Paulo C. M. de Abreu *et al.*. II. Universidade Federal da Bahia, PPGEE, Programa de Engenharia Elétrica. III. Título.

JOÃO LUIZ CARNEIRO CARVALHO

CONTRIBUIÇÕES AO PROBLEMA DE LOCALIZAÇÃO PARA VEÍCULOS TERRESTRES NÃO TRIPULADOS

Tese apresentada ao Programa de Pós-graduação em Engenharia Elétrica da Universidade Federal da Bahia (PPGEE/UFBA) como parte dos requisitos necessários para obtenção do grau de Doutor em Engenharia Elétrica.

Salvador, 31 de Maio de 2023.

Examinada por:

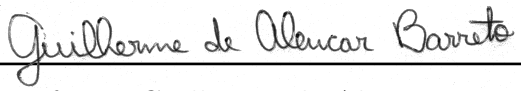
Prof. Dr. Paulo César Machado de Abreu Farias
Orientador (PPGEE/UFBA)

Documento assinado digitalmente
gov.br TIAGO TRINDADE RIBEIRO
Data: 05/06/2023 15:21:54-0300
Verifique em https://validar.iti.gov.br

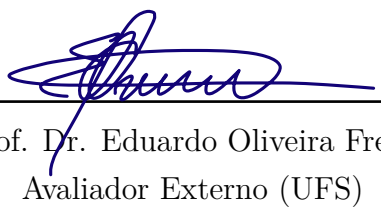
Prof. Dr. Tiago Trindade Ribeiro
Avaliador Interno (UFBA)

Documento assinado digitalmente
gov.br ANTONIO CARLOS LOPES FERNANDES JUI
Data: 05/06/2023 11:54:54-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Antônio Carlos Lopes Fernandes Júnior
Avaliador Interno (UFBA)



Prof. Dr. Guilherme de Alencar Barreto
Avaliador Externo (UFC)



Prof. Dr. Eduardo Oliveira Freire
Avaliador Externo (UFS)

É preciso estudar muito para saber um pouco.

Barão de Montesquieu

Resumo da Tese apresentada ao PPGEE/UFBA como parte dos requisitos necessários para a obtenção do grau de Doutor em Engenharia Elétrica (D. Eng.)

CONTRIBUIÇÕES AO PROBLEMA DE LOCALIZAÇÃO PARA VEÍCULOS TERRESTRES NÃO TRIPULADOS

João Luiz Carneiro Carvalho

Maio/2023

Orientadores: Paulo C. M. de Abreu Farias
Eduardo F. de Simas Filho

A localização de robôs móveis é uma tarefa complexa, principalmente em ambientes *indoor* não estruturados, devido à ruídos de medição e associações incorretas entre a percepção e o mapa do ambiente. Portanto, a quantificação da incerteza sobre a pose constitui uma parte fundamental dos métodos de localização. O procedimento de localização torna-se crítico quando o robô possui baixa confiança sobre sua última estimativa de pose, situação que requer um procedimento de localização global. Uma abordagem intuitiva para resolver o Problema da Localização Global (PLG) é distribuir várias hipóteses de pose ao longo do mapa e selecionar a mais provável de acordo com alguma heurística de otimização, como o método de Monte Carlo, Inteligência de Enxame ou Algoritmo Evolutivo, por exemplo. No entanto, as limitações de hardware e as características do ambiente podem afetar a eficácia da localização. Além disso, a literatura recente dispõe de poucos estudos explorando a eficácia e o custo computacional de diferentes métodos de localização sob cenários diversos, como escritórios, corredores e grandes armazéns, apenas para citar alguns exemplos. Neste contexto, este trabalho propõe duas contribuições ao algoritmo de localização Perfect Match (PM): aprimoramento da estimação da incerteza sobre a pose e incorporação do PLG. O PM é um algoritmo de rastreamento de pose que utiliza a abordagem de *scan-to-map matching* e se destaca pelo seu custo-benefício, pois apresenta alta precisão e baixo custo computacional. Entretanto, devido a natureza do algoritmo, a localização global não tem o mesmo desempenho que o rastreamento de pose. Além disso, a estimação da incerteza sobre a pose poderia ser aprimorada, visto que baseia-se apenas em características do mapa. A magnitude do erro de *matching*, informação relevante para indicar a qualidade da pose estimada, não é levada em consideração pelas implementações do PM disponíveis na literatura. Portanto, os resultados apresentados neste trabalho mostram que, nos cenários selecionados, a quantificação da incerteza sobre a pose pelo método proposto neste trabalho sugere ser mais adequada do que o PM com seu método original.

Em relação ao PLG, diferentes heurísticas de otimização baseadas em Algoritmos Evolucionários e Inteligência de Enxame foram utilizadas de forma colaborativa com o PM, sendo elas: Particle Swarm Optimization (PSO), Differential Evolution (DE) e Genetic Algoritm (GA). Usando simulações e experimentos reais, foram obtidos a taxa de sucesso e custo de computação usando diferentes tamanhos da população de partículas. Resultados mostram que os métodos propostos apresentam desempenhos diferentes para cenários distintos, mas aqueles baseados em GA e PSO apresentaram uma taxa média de sucesso acima de 83%, enquanto outros métodos não atingiram 80%.

Abstract of Thesis presented to PPGEE/UFBA as a partial fulfillment of the requirements for the degree of Doctor of Electrical Engineering (D. Eng.)

CONTRIBUTIONS TO THE LOCALIZATION PROBLEM FOR UNMANNED GROUND VEHICLES

João Luiz Carneiro Carvalho

May/2023

Advisors: Paulo C. M. de Abreu Farias
Eduardo F. de Simas Filho

Mobile robot localization is a complex task, specially in unstructured indoor environments, due to measurement noises and wrong scan-to-map association. Therefore, the quantification of uncertainty constitutes a important part of localization methods. The localization procedure becomes critical when the vehicle has low confidence about its last pose estimate, situation that requires a global localization procedure. An intuitive approach to solve the Global Localization Problem (GLP) is to distribute several pose hypotheses all over the map and select the most likely one according to an optimization heuristic such as Monte Carlo, Swarm Intelligence or Evolutionary Algorithm. However, hardware limitations and environment characteristics may affect the localization efficacy. In addition, the recent literature has few studies exploring the effectiveness and computing cost of different location methods under distinct scenarios, such as offices, corridors and large warehouses, for example. In this context, this work proposes two contributions to the Perfect Match (PM) localization algorithm: improvement of the uncertainty estimation about the pose and incorporation of the GLP. PM is a pose tracking algorithm that uses the *scan-to-map maching* approach and stands out for its cost-effectiveness, as it presents high accuracy and low computational cost. However, due to the kind of the algorithm, the global localization does not perform as well as the pose tracking. Furthermore, the estimation of the pose uncertainty could be improved, since it is based only on map features. The magnitude of the matching error, relevant information to indicate the quality of the estimated pose, is not taken into account by the PM implementations available in the literature. Therefore, the results presented in this work show that, in the selected scenarios, the quantification of the uncertainty about the pose by the proposed method suggests to be more adequate than the PM in its original form. Regarding the GLP, different optimization heuristics based on Evolutionary Algorithms and Swarm Intelligence were used collaboratively with the PM, such

as: Particle Swarm Optimization (PSO), Differential Evolution (DE) e Genetic Algoritm (GA). Using simulations and real experiments, success rate and computing cost using different population sizes were measured. Results show that the proposed methods present different performances for different scenarios, but those based on Genetic Algorithm and Particle Swarm Optimization presented an average success rate above 83%, while other methods did not reach 80%.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos e Abreviaturas

Símbolos

θ_t	Orientação no instante t
$\mathbf{p}_t$	Pose em 2D no instante t
${}^A\mathbf{p}_B$	Pose do *frame* B em relação ao *frame* A
${}^A\mathbf{R}_B$	Matriz de rotação do *frame* A para o *frame* B
${}^A\mathbf{T}_B$	Matriz de transformação do *frame* A para o *frame* B
σ_x^2	Variância de x
$\mathbf{\Sigma}$	Matriz de covariância de uma pose
v_t^o	Velocidade linear dada pela odometria no instante t
ω_t^o	Velocidade angular dada pela odometria no instante t
z_t^k	alcance obtido da k-ésima amostra do sensor no instante t
z_t^{k*}	alcance esperado da k-ésima amostra do sensor no instante t
$P(z_t^k\|\mathbf{p}_t)$	Probabilidade de ocorrer z_t^k, dado $\mathbf{p}_t$
$\mathbf{s}_k$	k-ésimo indivíduo da população
E	Erro de *matching*
$d_{\min}$	Deslocamento linear mínimo para iniciar uma nova iteração
$a_{\min}$	Deslocamento angular mínimo para iniciar uma nova iteração
$f_{\max}$	Número de partículas removidas da população por iteração
$P_{\max}$	Tamanho de população inicial
$P_{\min}$	Tamanho mínimo de população
L_{dev}	Desvio linear máximo para inicialização do Perfect Match
A_{dev}	Desvio angular máximo para inicialização do Perfect Match
$i_{\max}$	Número máximo de iterações para realização da localização global
i_{hit}	Iteração na qual a localização bem-sucedida é confirmada
Grad_x	Gradiente médio de E na direção x

Abreviaturas

AE Algoritmo Evolucionário

AG Algoritmo Genético

AMCL *Adaptive Monte Carlo Localization*

CIR Centro Instantâneo de Rotação

DE *Differential Evolution*

DEGL *Differential Evolution Global Localization*

EKF *Extended Kalman Filter*

FDP Função Densidade de Probabilidade

GAGL *Genetic Algorithm Global Localization*

GPS *Global Positioning System*

IE Inteligência de Enxame

IMU *Inertial Measurement Unit*

KLD *KullbackLeibler Divergence*

LaR Laboratório de Robótica da UFBA

LiDAR *Light Detection And Ranging*

LF *Likelihood Field*

MCL *Monte Carlo Localization*

MCU Movimento Circular Uniforme

MRU Movimento Retilíneo Uniforme

PLG Problema da Localização Global

PM *Perfect Match*

PMCov Proposta de Método para Cálculo da Covariância

PMGL *Perfect Match Global Localization*

PSL *Particle Swarm Localizer*

PSGL *Particle Swarm Global Localization*

PSO *Particle Swarm Optimization*

ROS *Robot Operating System*

RPROP *Resilient Backpropagation*

SLAM *Simultaneous Localization and Mapping*

TDE Transformada de Distância Euclideana

UGV *Unmanned Ground Vehicle*

Capítulo 1

Introdução

Localização é uma tarefa realizada pelo robô para identificar sua posição e orientação em relação ao espaço no qual ele está inserido. Em grande parte das aplicações, um robô móvel é caracterizado como um corpo rígido que se move pelo espaço e que precisa saber sua posição e orientação em relação a um sistema de coordenadas do espaço [1, p. 20]. Mais especificamente, robôs que se movem sobre uma superfície, como robôs com rodas, por exemplo, dependem de suas coordenadas x e y e sua rotação θ em relação à superfície. A tripla $\mathbf{p} = (x, y, \theta)$ que define a localização em 2D se chama Pose, e esta denominação será utilizada ao longo deste texto.

Para se mover e executar tarefas de forma segura e eficiente, o robô móvel deve manter uma estimativa de sua pose com reduzida incerteza. Entretanto, para se localizar de forma autônoma, ou seja, sem depender de um sistema de localização externo, o robô precisa conhecer o seu estado e compreender o ambiente ao seu redor. Desta forma, os algoritmos de localização precisam guardar em memória alguma representação virtual do ambiente, para que seja possível associar a percepção do ambiente em tempo real com algum mapa capaz de representar o ambiente digitalmente e com fidelidade adequada. Uma das representações mais comuns em localização em 2D é a grade de ocupação (ou *occupancy grid*, em inglês), que é frequentemente utilizada em robôs móveis de superfície que navegam em ambientes *indoor* [2].

De modo geral, os algoritmos de localização podem apenas manter uma pose estimada anterior e atualizá-la baseando-se nas mais recentes leituras dos sensores e do seu estado, processo denominado rastreamento de pose (ou *pose tracking*, em inglês). No entanto, em alguns casos a pose anterior é desconhecida ou de baixa confiabilidade devido a ausência de estimações ou medições dos sensores corrompidas.

Geralmente, existem duas circunstâncias comuns em que isto acontece: quando o robô está sendo inicializado (e não lhe foi informado qual pose inicial deve considerar)

ou na situação conhecida como Problema do Sequestro [3, p. 194], quando ele é reposicionado de maneira forçada e inadvertida. Também pode ser considerada como Problema do Sequestro aquela situação em que as medições dos sensores são obstruídas ou corrompidas por um longo período, comprometendo a percepção do robô. Consequentemente, o sistema de localização não é capaz de estimar a pose atual a partir da pose anterior, e portanto deve interromper o rastreamento de pose para iniciar algum processo de recuperação da localização. As situações descritas anteriormente caracterizam o Problema da Localização Global (PLG) e o robô deve considerar que sua localização pode estar em qualquer região do mapa. A partir de então, os dados dos sensores exteroceptivos são utilizados para encontrar a pose mais provável a partir de várias hipóteses. Porém, esta é uma tarefa sensível, pois a percepção do robô é limitada e a representação do ambiente pode conter ruídos e diversas ambiguidades.

Sendo assim, é comum a utilização de um ou mais sensores para aumentar o volume de informações usadas para a localização. Quando há mais de um sensor, as informações de diferentes fontes podem ser combinadas para aumentar a qualidade da estimação, uma vez que os sensores estão sujeitos a ruídos e limites de precisão próprios. Estes sensores podem produzir medições relativas, ou seja, pontos em relação ao robô, ou absolutas, onde as medições se referem a um sistema de coordenadas global – utilizando o Sistema de Posicionamento Global (*Global Positioning System* – GPS), por exemplo, é possível calcular suas coordenadas em relação à superfície terrestre por trilateração a partir das distâncias até pelo menos três satélites que têm posições conhecidas [4, p. 251].

Em algumas aplicações, a localização se baseia na detecção de pontos conhecidos, chamados de *landmarks* [5]. Estes podem ser objetos artificiais colocados especialmente para auxiliar na localização, como balizas, refletores ou códigos gráficos[1] [6] ou atributos naturais como árvores, gôndolas ou padrões geométricos no teto [7, 8], por exemplo. Entretanto, estas abordagens demandam a estruturação do local (ex.: instalação de *landmarks*) ou processamento adicional para extração e identificação de atributos do ambiente.

Por outro lado, a localização baseada em contornos pode utilizar dados brutos dos sensores e prescinde de um ambiente estruturado [9, 10] Geralmente, um mapa composto por uma grade de ocupação (*occupancy grid*), que representa o ambiente em pequenas células em 2D (x e y no plano da superfície) e indica quais estão ocupadas por um obstáculo e quais estão livres para serem atravessadas. Durante a localização, o robô tenta estabelecer uma correspondência entre as coordenadas de uma nuvem de pontos obtida dos sensores com as células ocupadas do mapa,

[1] QR Code é um exemplo de código gráfico, conhecido como *fiducial marker*, em inglês.

mecanismo conhecido como *scan-to-map matching* [9].

Entre outros métodos de localização, o *Perfect Match* (PM) [11] é um algoritmo de localização que realiza o *scan-to-map matching*. Nos trabalhos disponíveis na literatura [11–14], o PM apresentou baixo um erro de estimação da reduzido e um baixo custo computacional. Por isso, apesar de ainda não ser amplamente adotado no meio científico e comercial, este algoritmo tem potencial de se tornar um dos principais recursos disponíveis para localização de robôs.

Por outro lado, a primeira versão do PM, apresentada por Lauer *et al.* PM não é capaz de lidar com o PLG porque sempre depende de uma pose estimada anterior. Posteriormente, outros autores [15] introduziram um mecanismo de localização global no PM, porém utilizam uma abordagem onde o custo computacional é relativamente alto. Isto expõe um ponto de atenção neste algoritmo porque quando o robô está sendo inicializado ou é sequestrado, a localização global é uma tarefa crítica.

Além disso, a localização é ainda mais complexa em aplicações de robótica móvel que não dispõem, por exemplo, de odometria ou medição inercial, baseando-se apenas em uma nuvem de pontos e no *Occupancy Grid*. Devido à simetria de contornos, situação frequente na maioria dos ambientes urbanos e *indoor*, as nuvens de pontos próximas a diferentes cantos de uma sala ou diferentes corredores, por exemplo, podem ser bastante similares.

Além das limitações do PM em relação ao PLG, a estimação da incerteza sobre a pose também foi pouco aprofundada nas implementações estudadas. Uma das maneiras de determinar essa incerteza é através do cálculo das covariâncias das coordenadas da pose. Estas covariâncias dependem das características do algoritmo e dos dados recebidos pelos sensores, portanto de alguma forma a incerteza depende da percepção do robô. No caso do algoritmo PM, os autores descrevem a incerteza a partir de características do mapa do ambiente. Conforme discutido nos próximos capítulos, esta abordagem possui limitações, e portanto este trabalho propõe um aprimoramento do mecanismo de estimação da incerteza sobre a pose no PM.

1.1 Motivação

Trabalhos anteriores [9, 13, 14, 16] mostraram que o PM possui bom desempenho quando aplicado em robôs equipados com câmera ou sensores do tipo *Light Detection And Ranging* (LiDAR). Por isso, o PM tem sido investigado e aprimorado por pesquisadores, especialmente aqueles da Faculdade de Engenharia da Universidade do Porto (FEUP) [12, 17–19] e do Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência (INESC TEC) [9, 15, 16, 20], ambos na cidade do Porto,

em Portugal.

Além disso, no âmbito do projeto *Flexible and Autonomous Manufacturing Systems for Custom-Designed Products* (FASTEN)[2], pesquisadores do Laboratório de Robótica da UFBA (LaR) desenvolveram sistemas para construção de um robô móvel autônomo aplicado à indústria 4.0. Sendo assim, o PM foi selecionado para compor o sistema de localização deste robô, de modo que este trabalho apresenta as contribuições para o aprimoramento do PM no sentido de torná-lo um método de localização robusto e eficiente para a aplicação desejada.

Além dos aprimoramentos citados anteriormente, na localização global e na estimação da incerteza sobre a pose, este trabalho também foi motivado pelo reduzido número de publicações que investigam o desempenho e o custo computacional dos algoritmos de localização sob diferentes condições. Por exemplo, há interesse em avaliar o desempenho relativo de diferentes métodos de localização, em especial aqueles baseados em múltiplas hipóteses, em relação à quantidade dessas hipóteses no início da localização global, bem como em relação às características do ambiente.

1.1.1 Justificativa para a escolha do *Perfect Match*

Conforme citado anteriormente, o baixo custo computacional e o bom desempenho de estimação da pose fez do PM um método de localização prominente. Por outro lado, até o momento não se verificou um número expressivo de publicações com contribuições ao método PM. As principais iniciativas nesse sentido partiram dos pesquisadores da FEUP e do INESC TEC que, devido às parcerias com pesquisadores da UFBA, autorizaram o compartilhamento do código-fonte do PM desenvolvido. Entretanto, conforme apontado ao longo deste trabalho, algumas possibilidades de aprimoramento do PM ainda não haviam sido exploradas.

Durante o desenvolvimento do projeto FASTEN, o PM foi testado em experimentos preliminares e também apresentou resultados promissores. Sendo assim, o cenário se tornou favorável para a escolha do *Perfect Match*, visto que: (a) o PM apresenta um desempenho comparável com métodos de localização populares (b) havia a disponibilidade do código-fonte do PM com um certo grau de maturidade; (c) as possibilidades de aprimoramento do PM foram pouco exploradas.

1.1.2 Justificativa para a escolha dos métodos de otimização

A literatura apresenta diversos métodos para realizar a Localização Global de

[2]Informações sobre este projeto podem ser encontradas em `http://www.fastenmanufacturing.eu/` e em `https://cordis.europa.eu/project/id/777096`

Robôs móveis. Curiosamente, há uma maior concentração dos métodos baseados em multi-hipóteses, de modo que há uma população de indivíduos, que também são chamados de partículas, ou ainda, candidatos, onde cada uma representa uma hipótese de solução para o problema da localização. O método desse tipo que mais se destaca entre os desenvolvedores de *software* para robôs móveis por possuir código-fonte aberto e ter sua versão oficial disponibilizada facilmente na Internet é denominado *Adaptive Monte Carlo Localization* (AMCL). Obviamente, há exceções a este modelo como o Filtro de Histogramas [3, p. 86] ou abordagens menos convencionais, como a de Wang *et al* que utiliza redes neurais [21], apenas para citar dois exemplos.

No caso dos métodos baseados em multi-hipóteses, praticamente todos eles consistem em uma heurística de otimização que move e filtra os indivíduos em busca da melhor solução global. Por isso, diversas heurísticas podem ser testadas para o problema da busca pela pose correta do robô no mapa. Por outro lado, considerando que um dos objetivos deste trabalho é solucionar o PLG para o PM, algumas premissas foram estabelecidas para a escolha da heurística para realização da Localização Global:

- Que seja possível distribuir uma quantidade aleatória de hipóteses no espaço de busca da localização global, para que seja possível compará-lo com o AMCL;
- Que possa acomodar funções com múltiplos mínimos ou máximos globais;
- Que seja possível se sujeitar à uma função de *fitness* própria e adaptada do PM
- Que apresente uma forma de atualização da população com baixo custo computacional;
- Que seja possível interromper ou controlar a execução de cada iteração conforme o deslocamento do robô, conforme descrito no Capítulo 4 deste trabalho;

As primeiras quatro condições citadas anteriormente são comuns à maioria das heurísticas de otimização consultadas, enquanto que a última pode resultar em alguma adaptação da heurística. No entanto, houve a necessidade de limitar o número de heurísticas utilizadas neste trabalho, considerando o tempo para a implementação de cada uma delas. Ressalta-se também que a quantidade de heurísticas estudadas afeta o tempo dos experimentos realizados, visto que os testes dos cinco métodos de localização ocorreram durante centenas de horas de simulação e reprodução[3].

A primeira heurística avaliada, preliminarmente, foi o *Particle Swarm Optimization* (PSO), a partir de experiências compartilhadas entre os estudantes do Pro-

[3]Os ensaios com cada método em cada ambiente duraram entre 10 e 12 horas.

grama de Pós-Graduação em Engenharia Elétrica da UFBA, onde este método era usado para treinamento de redes neurais. As primeiras impressões indicaram que o PSO atendia bem o problema da Localização Global em ambientes pequenos. Então, os testes foram estendidos e os resultados satisfatórios foram confirmados, conforme publicação de Carvalho *et al* [22]. Em seguida, foram selecionadas heurísticas uma característica simular ao PSO: o comportamento de uma partícula é influenciado por outras partículas. Neste caso, observou-se que as heurísticas Algoritmo Genético (AG) e *Differential Evolution* (DE) possuíam essa característica e foram incluídas neste trabalho.

Além disso, os métodos propostos precisavam ser comparados com métodos de referência. Neste trabalho, estabeleceu-se o AMCL como um método de referência devido às suas características já citadas anteriormente, e também um método de Localização Global já proposto por Pinto *et al* mas que possui alto custo computacional, o *Initial Position Computation* [15]. Portanto, este trabalho envolveu a implementação de quatro heurísticas de multi-hipóteses e a utilização de um método de referência.

1.2 Objetivo Geral

Este trabalho consiste no estudo e aprimoramento do algoritmo de Localização *Perfect Match*. Dentro deste escopo, destacam-se os dois objetivos principais do trabalho: incorporar a localização global ao PM e aprimorar seu método de estimação da incerteza sobre a pose. Nesse sentido, esta tese apresenta propostas de métodos computacionais que, nos cenários analisados, melhoram o desempenho e a robustez do PM quando submetido à condições críticas.

No âmbito do primeiro objetivo, foi realizada uma avaliação detalhada do desempenho de métodos de otimização baseados em Algoritmo Evolucionário (AE) e Inteligência de Enxame (IE) aplicados no PLG em robótica móvel. Mais especificamente, foram utilizadas as heurísticas de otimização PSO, DE e AG. Estes métodos foram analisados de acordo com a quantidade de hipóteses e as características do ambiente. Do ponto de vista computacional, foram analisados os tempos decorridos em cada iteração do algoritmo e esforço até encontrar a pose correta.

Para alcançar o segundo objetivo, um novo mecanismo de cálculo de covariâncias para as coordenadas x, y e θ da pose foi proposto. Neste caso, o método se baseia nos gradientes do mapa de ocupação do ambiente e no erro de *matching* entre a nuvem de pontos do sensor a laser e os contornos do mapa. Para ambos os objetivos, utilizou-se o AMCL [23], um sistema de localização *open source* frequentemente utilizado pela literatura, como referência de desempenho.

1.3 Objetivos Específicos

Os objetivos específicos deste trabalho foram os desdobramentos dos objetivos gerais apresentados anteriormente, e estão relacionados a seguir:

1. Propor e desenvolver métodos de Localização Global baseados em AE ou IE acoplados ao PM;
2. Realizar uma análise comparativa dos diferentes métodos de localização global;
3. Confeccionar ambientes de simulação e reprodução de ensaios que expõem os métodos de localização global à condições operacionais críticas;
4. Avaliar o custo computacional relativo dos métodos de localização global propostos através da aferição do tempo de execução;
5. Apresentar uma análise detalhada do desempenho e do custo computacional dos métodos de localização global estudados;
6. Propor e desenvolver um novo método de estimação da incerteza sobre a pose que leva em consideração a magnitude do erro de matching;
7. Avaliar a viabilidade do novo método de estimação da incerteza em ensaios reais.
8. Propor um método de localização robusto e eficiente para o robô móvel autônomo utilizado neste trabalho

1.4 Principais contribuições desta tese

As principais contribuições desta tese podem ser identificadas a partir dos objetivos específicos deste trabalho. Portanto, este trabalho apresenta as principais contribuições:

1. Proposição de métodos eficientes para solução do PLG para o algoritmo PM.
2. Análise detalhada do desempenho e do custo computacional de diferentes heurísticas de otimização aplicadas ao PLG.
3. Aprimoramento do cálculo da incerteza sobre a pose para o PM;
4. Publicação dos resultados experimentais em ambientes reais e de simulação para diferentes condições de teste.

1.5 Organização do documento

Esta tese está organizada da seguinte forma: o Capítulo 2 discorre sobre os aspectos mais relevantes da localização em robótica móvel, que foram pertinentes a este trabalho; o Capítulo 3 descreve os principais algoritmos de localização encontrados na literatura, incluindo aquele que é objeto de estudo deste trabalho, o PM; o Capítulo 4, por sua vez, detalha as contribuições ao PM realizadas neste trabalho, conforme introduzido anteriormente, bem como a metodologia utilizada para desenvolvimento e realização dos experimentos. Os resultados obtidos são apresentados no Capítulo 5, enquanto o Capítulo 6 apresenta a conclusão e as perspectivas futuras deste trabalho.

Capítulo 2

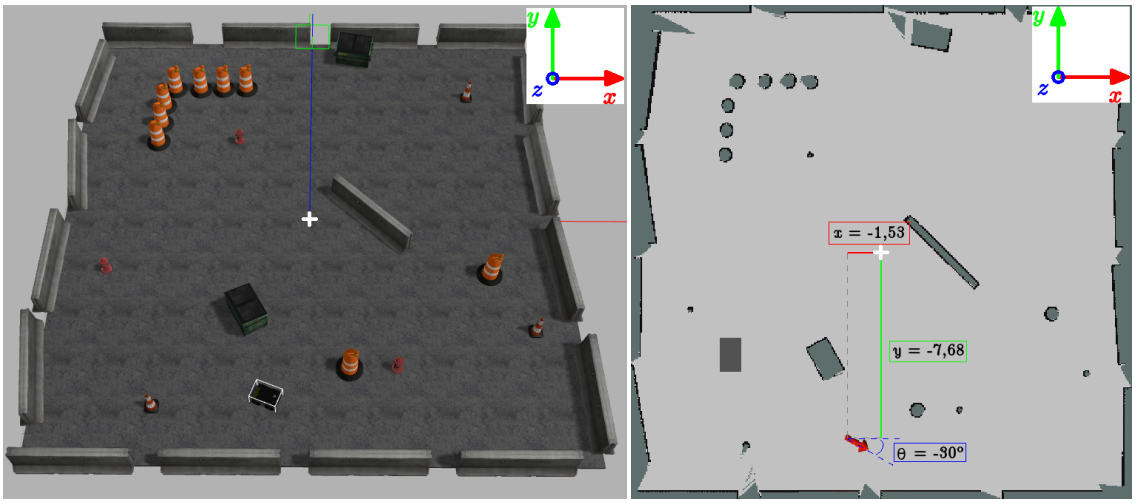
Localização em 2D

A posição e a orientação em um espaço 2D é estimada a partir do estado do robô, que guarda a dinâmica do seu movimento, e da percepção do ambiente em sua volta. Juntamente com um mapa, o estado e a percepção são utilizados para estimar a pose $\mathbf{p}_t = (x, y, \theta)$ no instante t em um sistema de coordenadas de duas dimensões. A Fig. 2.1 ilustra o modelo virtual de um Veículo terrestre não tripulado (*Unmanned Ground Vehicle* – UGV) em ambiente de simulação. O veículo se encontra na parte inferior da Fig. 2.1a, cujo espaço compreende aproximadamente 20×20 m. A seta vermelha na Fig. 2.1b ilustra a pose do veículo no respectivo mapa com origem $\mathbf{o} = (0, 0)$, bem como as coordenadas no plano, em metros, e a orientação, em graus, para melhor compreensão do leitor[1]. Portanto, neste exemplo, tem-se $\mathbf{p} = (-1.53, -7.68, -30)$.

A localização de um robô móvel é estimada porque suas coordenadas são calculadas a partir da medição de outras váriáveis, principalmente dos dados dos sensores de percepção e do estado do robô. Além disso, a estimação sempre está associada a algum grau de incerteza devido a ruídos, limitações dos sensores, ambiguidades e outros fatores imprevisíveis. Portanto, a pose fornecida pelo sistema de localização é uma aproximação da pose verdadeira, que por sua vez é chamada de *ground thruth*. Em simulação, é comum utilizar o *ground thruth* como a pose de referência para avaliar o desempenho do algoritmo de localização ou para substituí-lo. Por outro lado, em experimentos reais, esta variável não está disponível, e neste caso costuma-se utilizar algum sistema de localização auxiliar, como *Global Positioning System* (GPS), por exemplo.

Além disso, para realizar o rastreamento da pose, ou seja, atualizar $\mathbf{p}_t$ de acordo com seu estado e sua percepção ao longo do tempo, é necessário ter uma pose anterior conhecida $\mathbf{p}_{t-1}$ com reduzido grau de incerteza. Entretanto, nem sempre uma pose anterior é conhecida ou confiável (como no instante $t = 0$, por exemplo), e neste

[1] A representação convencional da rotação é dada em radianos



(a) Modelo virtual do robô em ambiente de simulação na parte inferior da imagem

(b) Posição do robô no respectivo mapa, indicado pela composição dos vetores em vermelho e verde a partir da origem, e orientação a partir da posição apontada pela seta em vermelho; na imagem estão especificadas o valor das coordenadas (x, y) e a orientação θ em relação à origem do mapa

Figura 2.1: Representação virtual de um robô de superfície. Os eixos x e y são mostrados em vermelho e verde, respectivamente, paralelos ao plano da imagem, como mostra a ilustração no canto superior direito de cada figura. O eixo z está ortogonal ao plano da imagem, saindo dele, e a origem do sistema de coordenadas está representado pela cruz branca

caso é necessário realizar um procedimento chamado de localização global.

Este capítulo discorre sobre os principais aspectos relacionados a localização que foram utilizados neste trabalho. Os sistemas de coordenadas utilizados na localização e, também, na caracterização do veículo, são apresentados na seção 2.1. A forma de representação do erro sobre a estimação das coordenadas do robô é apresentada na seção 2.2. A seção 2.3 introduz os modelos de movimento básicos que descrevem o estado do veículo, enquanto que a representação do mapa utilizado na localização é apresentado na seção 2.4. Finalmente, a percepção através de sensores é introduzida na seção 2.5.

2.1 Descrição da pose por sistema de coordenadas

Uma forma genérica e frequentemente utilizada para localizar um objeto no espaço é através da caracterização de um corpo rígido no sistema tridimensional. Além

de sua posição em 3D, também devem ser representadas quaisquer inclinações em relação às três dimensões, definindo portanto a sua pose no espaço. Sendo assim, para descrever a pose $\mathbf{p}_t$ de um veículo em 3D, são necessárias 6 dimensões [1, p. 17]: três para a posição (x, y, z) e três para a sua orientação $(\beta_{(x)}, \psi_{(y)}, \theta_{(z)})$.

As três dimensões da orientação se referem às rotações em relação a cada um dos eixos. A variável $\beta_{(x)}$ também é chamada de *roll*, em inglês, e se refere à rotação em relação ao eixo x. A variável $\psi_{(y)}$ representa o *pitch*, que define a rotação em torno do eixo y. Por último, o $\theta_{(z)}$) consiste na rotação em torno do eixo z e é chamado de *yaw*. No entanto, embora o sistema tridimensional seja ideal para representação de objetos no mundo físico, no caso de veículos que navegam em superfície, dimensões em 2D são suficientes pois geralmente são consideradas as seguintes premissas:

- a distância entre o robô e a superfície no qual ele navega é aproximadamente constante, geralmente nula;
- o robô rotaciona somente em relação ao eixo perpendicular à superfície (ou seja, ele não deve tombar).

Sendo assim, para localizar um veículo terrestre, a coordenada z e as rotações β e ψ podem ser suprimidas [1, p. 22], restando apenas um ponto ${}^{W}(x, y)$[2] do sistema de coordenadas global $\{W\}$ e um ângulo ${}^{W}\theta$ em relação ao eixo x, conforme ilustrado na Fig. 2.1b. Esta redução do número de dimensões permite a simplificação de transformações lineares.

Este trabalho utiliza, majoritariamente, a representação em 2D para todos os elementos do espaço, visto que são abordados apenas os robôs móveis de superfície. Porém, este capítulo também aborda o sistema cartesiano completo (em 3D), visto que eventualmente há um ponto de interface entre os algoritmos baseados em 2D com bibliotecas de programação, *softwares* de simulação e ferramentas de suporte ao desenvolvimento que utilizam a descrição em 3D.

Também é necessário destacar que este trabalho adota a convenção da literatura [1, 24–26] que representa o sistema com coordenadas x e y – e em alguns casos, z, obedecendo à regra da mão direita, conforme ilustra a Fig. 2.2a, onde os dedos apontam a direção positiva de cada eixo. As rotações positivas em torno de cada eixo também obedecem à mesma regra [27, p. 12], como mostra a Fig. 2.2b. O triedro positivo pode ser definido a partir dos versores das direções dos sistemas de coordenadas por intermédio do produto vetorial:

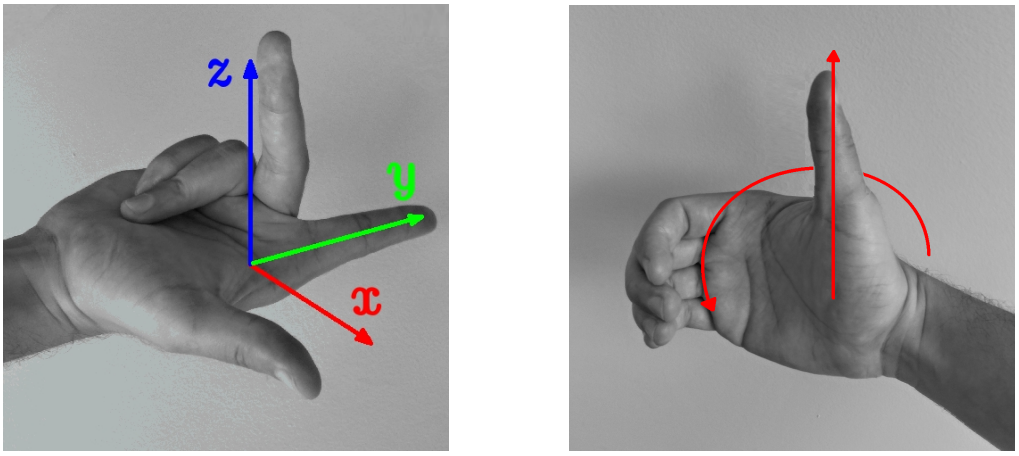
$$\hat{z} = \hat{x} \times \hat{y},\ \hat{x} = \hat{y} \times \hat{z},\ \hat{y} = \hat{z} \times \hat{x}, \tag{2.1}$$

onde $\hat{x}$, $\hat{y}$ e $\hat{z}$ são vetores unitários paralelos aos eixos do sistema de coordenadas em 3D. Além disso, fica estabelecido ao longo deste trabalho que os eixos x, y e

[2] A notação sobrescrita é similar à utilizada por Corke (2017) [1].

z são desenhados nas cores vermelho, verde e azul, respectivamente, quando há necessidade de distingui-los.

No sistema de coordenadas global, ou seja, no sistema cartesiano com origem no centro do ambiente, o eixo z sempre será perpendicular ao plano sobre o qual o robô navega (Fig. 2.1a). Estas mesmas convenções são adotadas por sistemas que operam no Sistema Operacional do Robô (*Robot Operating System* – ROS) [28], um *framework* para sistemas voltados à robótica que é apresentado na subseção 4.3.1.



(a) Regra da mão direita para direção positiva dos eixos

(b) Regra da mão direita para rotações positivas em torno de um dos eixos

Figura 2.2: Convenções baseadas na regra da mão direita

2.1.1 Pose em 2D

Para caracterizar a pose relativa de um corpo rígido em 2D, é comum atribuir à ele um sistema de coordenadas (também chamado de *frame*, em inglês). Uma vez que um robô costuma ser montado com diversos elementos em seu chassi, como sensores, rodas e manipuladores, entre outros, cada um destes elementos pode manter um *frame* próprio [25, p. 43]. Portanto, este robô pode ser caracterizado por um conjunto de *frames* relacionados entre si.

A Fig. 2.3 ilustra os sistemas de coordenadas 2D atribuídos a um veículo hipotético V montado com um sensor S que navega em um ambiente qualquer W. Os *frames* $\{W\}$, $\{V\}$ e $\{S\}$ estão fixados no centro do ambiente, do veículo e do sensor, respectivamente, de forma que $\{W\}$ constitui o sistema de coordenadas global. As coordenadas x e y do robô e do sensor estão descritas na ilustração, de forma que termos sobrescritos nas variáveis indicam o *frame* de referência, enquanto os termos subscritos indicam qual elemento está sendo descrito. Por exemplo, a variável ${}^{W}y_{V}$ representa a coordenada y do *frame* do veículo V em relação ao *frame* $\{W\}$. A Fig. 2.3 também mostra um ponto $\mathbf{q}$ representando algum objeto no espaço.

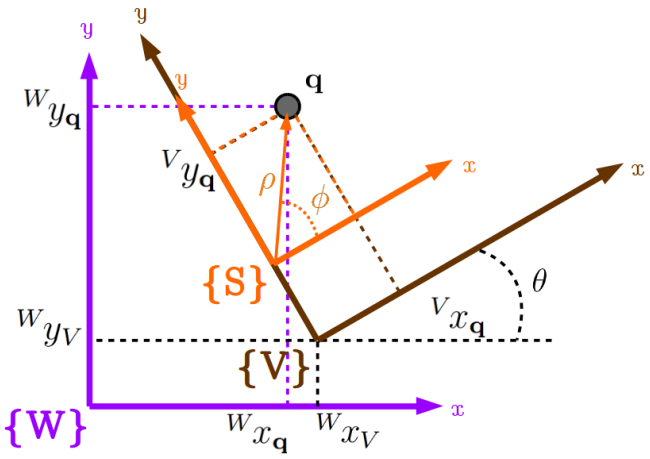


Figura 2.3: Sistemas de coordenadas (*frames*) que representam um ambiente W, um veículo V e um sensor S. Cada *frame* da figura se refere à origem dos vetores base dos frames

Na Fig. 2.3, o sensor detecta um ponto de interesse $\mathbf{q}$ (um obstáculo ou um *landmark*), localizado a uma distância $\rho \geq 0$ de sua origem e a um ângulo $\phi \in (-\pi, \pi]$ em relação ao seu eixo x. Neste caso, as coordenadas do ponto $\mathbf{q}$ em relação ao sensor S podem ser obtidas por:

$$^S\mathbf{q} = (\rho \cos\phi,\ \rho \sin\phi). \tag{2.2}$$

A localização de $\mathbf{q}$ em relação ao veículo constitui uma informação importante, visto que, para interagir de forma segura e eficiente, o robô deve perceber o ambiente em sua volta, evitando, principalmente, colisões com obstáculos durante a navegação. Consequentemente, a posição de $\mathbf{q}$ deve ser transformada para o sistema de coordenadas do veículo $\{V\}$. Dado que os eixos x e y dos *frames* $\{V\}$ e $\{S\}$ estão respectivamente paralelos, a transformação linear envolve apenas uma adição vetorial:

$$^V\mathbf{q} = (^V x_{\mathbf{q}}, ^V y_{\mathbf{q}}) = \ ^S\mathbf{q} + (^V x_S,\ ^V y_S), \tag{2.3}$$

onde $^V x_S$ e $^V y_S$ são as coordenadas de S no *frame* $\{V\}$.

Por outro lado, para que o robô possa se localizar, navegar e realizar tarefas, também é necessário estabelecer a correspondência entre os obstáculos detectados pelo sensor e suas respectivas localizações no mapa. Portanto, estes pontos devem ser transformados também para o sistema de coordenadas $\{W\}$. Entretanto, no exemplo da Fig. 2.3, os *frames* $\{W\}$ e $\{V\}$ não coincidem e sequer são ortogonais. Esta é uma situação típica, visto que o robô percorre o ambiente assumindo diferentes poses ao longo do tempo. Por isso, a transformação do ponto $\mathbf{q}$ de $\{V\}$ para $\{W\}$ não é tão simples como na Eq. (2.3).

A Fig. 2.4 mostra o mesmo *frame* $\{V\}$ com um novo *frame* auxiliar $\{U\}$ ortogonal a $\{W\}$ e com origem em $\{V\}$. Ao considerar os eixos de ambos os *frames* como

vetores unitários, $(\hat{\mathbf{x}}_V, \hat{\mathbf{y}}_V)$ e $(\hat{\mathbf{x}}_U, \hat{\mathbf{y}}_U)$, nota-se que é possível transformar o *frame* $\{U\}$ em $\{V\}$ através de uma rotação:

$$(\hat{\mathbf{x}}_V, \hat{\mathbf{y}}_V)^T = (\hat{\mathbf{x}}_U, \hat{\mathbf{y}}_U)^T \underbrace{\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}}_{{}^U\mathbf{R}_V(\theta)} \tag{2.4}$$

Na Eq. (2.4), ${}^U\mathbf{R}_V(\theta)$ é a matriz de rotação usada para transformar pontos de $\{U\}$ para $\{V\}$ [1, p. 24]. Então, nota-se que para transformar pontos de $\{S\}$ para $\{V\}$, sendo ambos os *frames* ortogonais, tem-se uma rotação com ângulo nulo:

$${}^S\mathbf{R}_V(\alpha) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \text{ onde } \alpha = 0. \tag{2.5}$$

Portanto, agregando a translação e a rotação, a Eq. (2.3) por ser generalizada por:

$$\begin{aligned} {}^V\mathbf{q} &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \underbrace{\begin{pmatrix} {}^Sx_\mathbf{q} \\ {}^Sy_\mathbf{q} \end{pmatrix}}_{{}^S\mathbf{q}} + \begin{pmatrix} {}^Vx_S \\ {}^Vy_S \end{pmatrix} \\ &= \underbrace{\begin{pmatrix} \cos\alpha & -\sin\alpha & {}^Vx_S \\ \sin\alpha & \cos\alpha & {}^Vy_S \end{pmatrix}}_{{}^V\mathbf{T}_S} \begin{pmatrix} {}^Sx_\mathbf{q} \\ {}^Sy_\mathbf{q} \\ 1 \end{pmatrix}, \ \alpha = 0, \end{aligned} \tag{2.6}$$

de forma que ${}^V\mathbf{T}_S$ representa a matriz de transformação que realiza transformações de S para V. De modo geral, portanto, uma matriz ${}^A\mathbf{T}_B$, também referida como movimento de corpo rígido [3, p. 27], realiza a translação e rotação necessárias para mapear pontos B em A. Sendo assim, é possível realizar transformações entre os *frames* S, V e W da Fig. 2.3 utilizando a matriz de transformação.

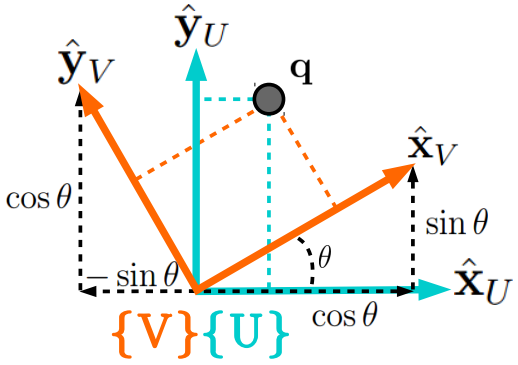


Figura 2.4: Sistema de Coordenadas temporário $\{U\}$ com origem em $\{V\}$

Nota-se que uma matriz de transformação ${}^W\mathbf{T}_V$ contém as coordenadas de V em W, $({}^Wx_V, {}^Wy_V)$, e a sua respectiva orientação ${}^W\theta_V$. Assume-se ainda que a

Eq. (2.6) descreve os movimento de rotação e translação de um corpo rígido em um plano. Portanto, pode-se descrever a pose $\mathbf{p}_V$ de um veículo de superfície através de ${}^{W}\mathbf{T}_V$.

2.1.2 Pose em 3D

Conforme citado no início deste capítulo, os sistemas de coordenadas em 3D possuem uma dimensão a mais do que aqueles em 2D. Além disso, a pose representada em 3D possui uma coordenada adicional e mais duas rotações: $\mathbf{p} = (x, y, z, \beta, \psi, \theta)$. A equação de translação de uma pose é similar à Eq. (2.3), porém com uma coordenada a mais:

$$ {}^{A}\mathbf{q}' = ({}^{A}x_{\mathbf{q}}, {}^{A}y_{\mathbf{q}}, {}^{A}z_{\mathbf{q}}) = {}^{B}\mathbf{q}' + {}^{A}\mathbf{o}_B \ , \qquad \mathbf{q}' \in \mathbb{R}^3, \tag{2.7} $$

onde ${}^{A}\mathbf{o}_B \in \mathbb{R}^3$ é a origem do *frame* $\{B\}$ em $\{A\}$. Por outro lado, dado que a orientação é definida por três ângulos[3], a rotação em 3D é representada por três matrizes de rotação:

$$ \mathbf{R}_\beta = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & \sin\beta \\ 0 & -\sin\beta & \cos\beta \end{pmatrix} \tag{2.8} $$

$$ \mathbf{R}_\psi = \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \ , \tag{2.9} $$

$$ \mathbf{R}_\theta = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix} \tag{2.10} $$

onde β, ψ e θ são os ângulos referentes aos eixos X, Y e Z, respectivamente.

A ordem na qual estas rotações são aplicadas influenciam diretamente na orientação final do robô. Existem seis possíveis sequências para realizar uma rotação em 3D (XYZ, XZY, YZX, YXZ, ZXY e ZYX), porém as mais citadas são XYZ [29, 30] e ZYX [1, 4]. Adotando-se a convenção ZYX [4, p. 210], tem-se que a matriz de rotação em três dimensões ${}^{A}\mathbf{R}_B \in \mathbb{R}^{3\times3}$ é dada por:

$$ \begin{aligned} {}^{A}\mathbf{R}_B &= \mathbf{R}_\beta \mathbf{R}_\psi \mathbf{R}_\theta \\ &= \begin{pmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta & -\sin\psi \\ \sin\beta\sin\psi\cos\theta - \cos\beta\sin\theta & \sin\beta\sin\psi\sin\theta + \cos\beta\cos\theta & \sin\beta\cos\psi \\ \cos\beta\sin\psi\cos\theta + \sin\beta\sin\theta & \cos\beta\sin\psi\sin\theta - \sin\beta\cos\theta & \cos\beta\cos\psi \end{pmatrix} \end{aligned} \tag{2.11} $$

[3] Considerando a notação de ângulos de Euler.

Sendo assim, a transformação homogênea de pontos de $\{B\}$ para $\{A\}$ é obtida por:

$$\begin{aligned}\begin{pmatrix} {}^{A}\mathbf{p} \\ 1 \end{pmatrix} &= \begin{pmatrix} {}^{A}\mathbf{R}_B & {}^{A}\mathbf{o}_B^T \\ 0\ 0\ 0 & 1 \end{pmatrix} \begin{pmatrix} {}^{B}\mathbf{p}^T \\ 1 \end{pmatrix} \\ &= {}^{A}\mathbf{T}_B \begin{pmatrix} {}^{B}\mathbf{p}^T \\ 1 \end{pmatrix}\end{aligned}, \tag{2.12}$$

onde ${}^{A}\mathbf{o}_B^T \in \mathbb{R}^{3\times1}$ e ${}^{B}\mathbf{p}^T \in \mathbb{R}^{3\times1}$ constituem matrizes transpostas e ${}^{A}\mathbf{T}_B \in \mathbb{R}^{4\times4}$ é a matriz de transformação homogênea em 3D. Como ${}^{A}\mathbf{T}_B$ contém translação e rotações, este pode representar a pose de B em $\{A\}$.

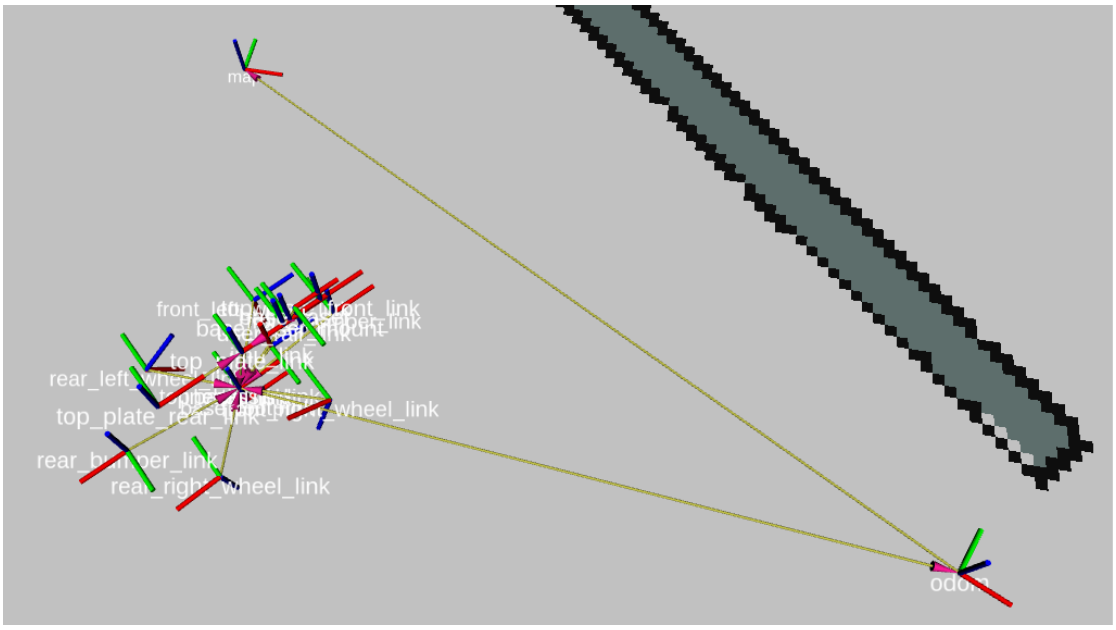
As transformações homogêneas são utilizadas para transformar pontos entre sistemas de coordenadas e representar poses de corpos presentes no espaço. Por exemplo, a Fig. 2.5 mostra um robô em ambiente virtual sendo representado por um modelo gráfico (Fig. 2.5a) e um conjunto de *frames* que estabelecem uma relação entre os elementos do espaço [28]. Conforme mostra a Fig. 2.5b, cada *frame* está atribuído a um dos elementos, como o mapa (`map`), a pose estimada pelo sistema de odometria (`odom`), as rodas do robô (`rear_right_wheel_link` e outros) e o para-choque traseiro (`rear_bumper_link`), apenas para citar alguns exemplos. Estes *frames* estão relacionados na forma de uma árvore, tipicamente chamada de árvore de transformada (*transform tree*, em inglês, ou simplesmente TF).

Foote [28] desenvolveu uma biblioteca para padronizar a representação e a transformação entre *frames*, denominando-a *tf library*. Os *frames* são relacionados através de um grafo direcionado que se assemelha a uma estrutura do tipo "pai-filhos", de forma que é possível transformar pontos entre quaisquer *frames*, desde que ambos estejam representados dentro do mesmo grafo. Portanto, é comum que o primeiro pai do todos os frames seja o mapa do ambiente (`map`), porque este abrange todos os elementos do espaço. Cada nó do grafo representa a pose do respectivo *frame*, ou ainda, a transformação homogênea relativa a este. A Fig. 2.5b foi concebida a partir da árvore gerada com a *tf library*, no mesmo instante de simulação da Fig. 2.5a, e o grafo com todos os *frames* do sistema está ilustrado na Fig. 2.6.

Nota-se na Fig. 2.6 que entre o mapa (`map`) e a base do robô (`base_link`) há um *frame* intermediário (`odom`). Ocorre que, neste exemplo, o sistema de odometria do veículo calcula a pose baseando-se no movimento das rodas, conforme será descrito na seção 2.3.2. Se este fosse um sistema perfeito os *frames* `map` e `odom` deveriam sempre coincidir, enquanto o *frame* do robô (`base_link`) seria atualizado de acordo com a pose calculada a partir dos dados da odometria. Ou seja, o erro em relação a pose do robô aumenta à medida que este se move. Então, neste caso, o robô utiliza um segundo sistema de localização baseado em sensor a laser que, a partir da percepção do ambiente, agrega erro do *frame* `odom` à árvore.

Em outras palavras, de acordo com a diferença entre a pose informada pela odometria e a pose estimada a partir do laser, o sistema de localização altera a pose do frame `odom` em relação ao sistema de coordenadas global para manter a pose do robô `base_link` mais próxima da correta. Esta diferença pode ser observada na Fig. 2.5, onde `map` e `odom` apresentam uma nítida divergência causada pela atualização do sistema de localização a laser, com o objetivo de manter a correta localização do robô (`base_link`). Convenientemente, esta mesma árvore de transformada mostrada nas Figs. 2.5 e 2.6 é utilizada pelo robô durante os testes conduzidos neste trabalho e discutidos a partir da seção 4.3.

(a) Modelo virtual do robô em ambiente de simulação.



(b) Árvore de sistemas de coordenadas

Figura 2.5: Robô virtual durante simulação. Modelo: Clearpath Husky A200; Ambiente de simulação: Clearpath Playpen

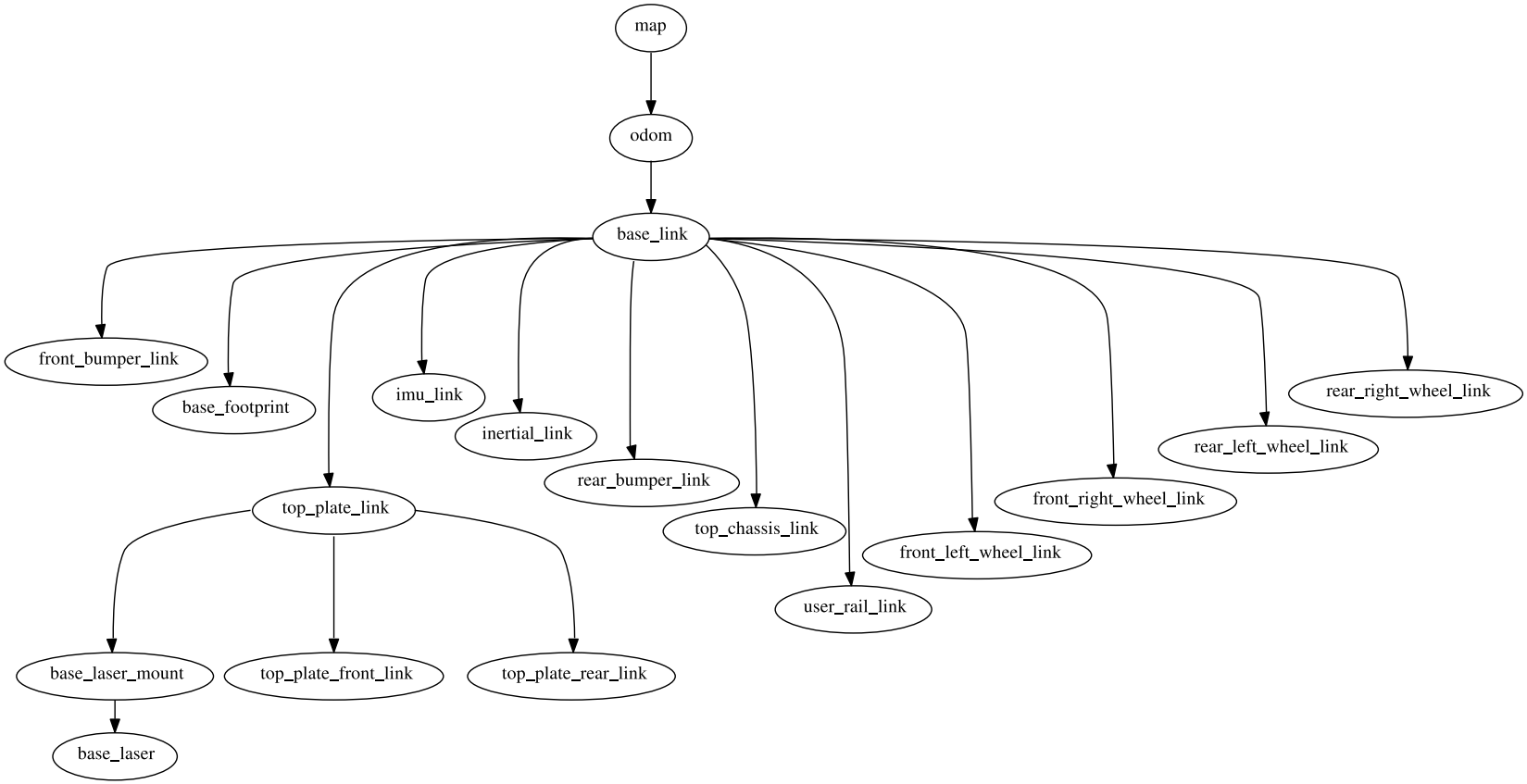


Figura 2.6: Grafo da arvore TF do robô Husky A200

2.1.3 Quatérnios

A matriz de rotação em 3D apresentada anteriormente traz alguns fatores indesejáveis, como a grande quantidade de operacões trigonométricas – Eq. (2.11), e o problema da singularidade (ou *gimbal lock*), no qual um grau de liberdade é anulado quando dois de seus eixos ficam alinhados. Alternativamente, a rotação de vetores em 3D pode ser compactamente representada através de quatérnios, que contorna estes problemas, como será visto a seguir.

Um quatérnio consiste em uma extensão de números complexos $q = q_0 + q_1 i + q_2 j + q_3 k$, onde $i^2 = -1$, $j^2 = -1$ e $k^2 = -1$ definem as diferentes partes imaginárias de q. A rigor, um quatérnio representa um ponto (q_0, q_1, q_2, q_3) no espaço complexo de quatro dimensões. Para caracterizar uma rotação sobre um vetor unitário, este quatérnio deve estar normalizado, isto é, $q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1$.

Por definição, as rotações em torno dos eixos x, y e z, dadas por $\mathbf{R}_\beta$, $\mathbf{R}_\psi$ e $\mathbf{R}_\theta$ nas Eqs. (2.8), (2.10) e (2.9), respectivamente, podem ser representadas em quatérnios por [27, p. 45]:

$$\begin{aligned} q_\beta &= \cos\frac{\beta}{2} + \sin\frac{\beta}{2}k \\ q_\psi &= \cos\frac{\psi}{2} + \sin\frac{\psi}{2}j \\ q_\theta &= \cos\frac{\theta}{2} + \sin\frac{\theta}{2}i \ . \end{aligned} \tag{2.13}$$

Multiplicações de quatérnios correspondem à combinação de rotações sucessivas [31, p. 497]. Portanto, a orientação do corpo rígido B em relação a $\{A\}$, composta pelas rotações em cada um dos três eixos, conforme apresentado na Eq. (2.11), seria definida em quatérnios por ${}^A q_B = q_\beta \ q_\psi \ q_\theta$. As regras fundamentais que permitem a multiplicação de quatérnios com a propriedade distributiva foram introduzidas por Hamilton [32]:

$$\begin{aligned} i^2 = j^2 = k^2 = ijk &= -1 \\ i = jk &= -kj \\ j = ki &= -ik \\ k = ij &= -ji \ . \end{aligned} \tag{2.14}$$

Nota-se, no entanto, que a propriedade comutativa não é válida em quatérnios ($jk \neq kj$). Sendo assim, a composição de duas rotações $q_x = (x_0, x_1, x_2, x_3)$ e

$q_y = (y_0, y_1, y_2, y_3)$ é dada por:

$$\begin{aligned} q_x q_y &= (x_0, x_1 i, x_2 j, x_3 k)(y_0, y_1 i, y_2 j, y_3 k) \\ &= x_0 y_0 + x_0 y_1 i + x_0 y_2 j + x_0 y_3 k \\ &\quad - x_1 y_1 + x_1 y_0 i - x_1 y_3 j + x_1 y_2 k \\ &\quad - x_2 y_2 + x_2 y_3 i + x_2 y_0 j - x_2 y_1 k \\ &\quad - x_3 y_3 - x_3 y_2 i + x_3 y_1 j + x_3 y_0 k \ . \end{aligned} \tag{2.15}$$

Fica evidente que, diferentemente da Eq. (2.11) que envolve operações trigonométricas, a Eq. (2.15) envolve apenas operações básicas de adição e multiplicação. Esta característica torna o quatérnio um tipo de dado mais apropriado para operações computacionais referentes à rotação de corpos rígidos em 3D.

Apesar de sua aptidão para operações numéricas, a compreensão de um quatérnio no espaço tridimensional não é trivial. Além disso, em aplicações onde há interesse em apenas uma das três orientações (como na localização em 2D, por exemplo), as vantagens do quatérnio se tornam pouco relevantes. Portanto, é comum a conversão entre quatérnios e matrizes de rotação ao longo do processo. A conversão do primeiro tipo para o segundo é dada por [4, p. 212]:

$$\mathbf{R} = \begin{pmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_0 q_3 + q_1 q_2) & 2(q_1 q_3 - q_0 q_2) \\ 2(q_1 q_2 - q_3 q_0) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_0 q_1 + q_2 q_3) \\ 2(q_0 q_2 + q_1 q_3) & 2(q_2 q_3 - q_0 q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{pmatrix} , \tag{2.16}$$

bem como é possível a recuperação de uma das orientações da pose:

$$\begin{aligned} \beta &= \arctan \frac{2(q_1 q_0 + q_2 q_3)}{q_0^2 - q_1^2 - q_2^2 + q_3^2} \\ \psi &= -\arcsin \left[2(q_1 q_3 - q_2 q_0)\right] \\ \theta &= \arctan \frac{2(q_3 q_0 + q_1 q_2)}{q_0^2 + q_1^2 - q_2^2 + q_3^2} \ . \end{aligned} \tag{2.17}$$

2.2 Variância e covariância

Dado que a pose do robô é um vetor de variáveis estimadas a partir de dados indiretos, sujeitos a ruídos e aproximações, há necessidade de representar algum erro em relação a estas variáveis. Portanto, uma das formas utilizadas para representar a pose é através de variáveis aleatórias gaussianas, como por exemplo:

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \sim N(0, \mathbf{\Sigma}_{2\mathrm{D}}),\ \mathbf{\Sigma}_{2\mathrm{D}} = \begin{pmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{pmatrix} , \tag{2.18}$$

onde $N(0, \mathbf{\Sigma}_{2\mathrm{D}})$ representa uma distribuição normal multivariada e $\mathbf{\Sigma}_{2\mathrm{D}}$ é uma matriz de covariância das variáveis independentes da pose em 2D (x, y, θ) com variâncias

σ_x^2, σ_y^2 e σ_θ^2. A justificativa para modelar a pose através de uma distribuição normal multivariada é que, em grande parte das aplicações, a incerteza sobre a pose pode ser aproximada por uma função gaussiana. Além disso, funções gaussianas são apropriadas para utilização no Filtro de Bayes, modelo matemático no qual muitos algoritmos de localização se baseiam. Por outro lado, na prática estas variáveis aleatórias não são independentes, de modo que os elementos fora da diagonal principal podem assumir valores diferentes de zero, caracterizando as covariâncias entre cada par de variáveis aleatórias. É importante ressaltar que, no caso da representação da pose em 3D com a rotação descrita através de quatérnios, a matriz de covariância terá tamanho 6×6, devido à composição das covariâncias das três coordenadas (x, y, z) e as rotações em relação a cada um dos eixos(β, ψ, θ). Portanto, a matriz de covariância de uma pose em 3D é caracterizada da seguinte forma:

$$\mathbf{\Sigma}_{3\mathrm{D}} = \begin{pmatrix} \mathrm{cov}_{x,x} & \mathrm{cov}_{x,y} & \mathrm{cov}_{x,z} & \mathrm{cov}_{x,\beta} & \mathrm{cov}_{x,\psi} & \mathrm{cov}_{x,\theta} \\ \mathrm{cov}_{y,x} & \mathrm{cov}_{y,y} & \mathrm{cov}_{y,z} & \mathrm{cov}_{y,\beta} & \mathrm{cov}_{y,\psi} & \mathrm{cov}_{y,\theta} \\ \mathrm{cov}_{z,x} & \mathrm{cov}_{z,y} & \mathrm{cov}_{z,z} & \mathrm{cov}_{z,\beta} & \mathrm{cov}_{z,\psi} & \mathrm{cov}_{z,\theta} \\ \mathrm{cov}_{\beta,x} & \mathrm{cov}_{\beta,y} & \mathrm{cov}_{\beta,z} & \mathrm{cov}_{\beta,\beta} & \mathrm{cov}_{\beta,\psi} & \mathrm{cov}_{\beta,\theta} \\ \mathrm{cov}_{\psi,x} & \mathrm{cov}_{\psi,y} & \mathrm{cov}_{\psi,z} & \mathrm{cov}_{\psi,\beta} & \mathrm{cov}_{\psi,\psi} & \mathrm{cov}_{\psi,\theta} \\ \mathrm{cov}_{\theta,x} & \mathrm{cov}_{\theta,y} & \mathrm{cov}_{\theta,z} & \mathrm{cov}_{\theta,\beta} & \mathrm{cov}_{\theta,\psi} & \mathrm{cov}_{\theta,\theta} \end{pmatrix}. \tag{2.19}$$

As formas de calcular a matriz de covariância dependem do algoritmo utilizado, sendo que o Capítulo 3 descreve algumas delas. O PM, por exemplo, se baseia em características do mapa para calcular a matriz de covariância no formato descrito pela Eq. (2.18) e a converte para o formato da Eq. (2.19) apenas quando é necessário transformar para o formato compatível à outros sistemas ou bibliotecas.

De certa forma, a matriz de covariância representa uma margem de erro sobre as coordenadas da pose, portanto pode ser útil para avaliar a qualidade da localização, ou ainda, verificar se o robô está perdido. Além disso, a grade de ocupação utilizada pelo método de localização está sujeita a erros de aproximação e outros fatores imprevisíveis durante a sua elaboração, o sistema de odometria está sujeito à falhas de tração das rodas e erros de quantização, e os sensores fazem medições do ambiente em sua volta que também estão sujeitas a ruídos, sensibilidade, precisão e, ainda, interferências externas. Estes fatores afetam diretamente a estimação da pose, e portanto, de alguma forma, os métodos de localização utilizam a matriz de covariância para representar a incerteza sobre a pose estimada.

A literatura apresenta algumas estratégias para utilizar a incerteza estimada como forma de detectar se o algoritmo de localização está "perdido", como a proposta de Farias *et al* [16], por exemplo, que utiliza dois algoritmos de localização distintos para detectar a situação de sequestro quando as poses estimadas por ambos divergem. Por outro lado, apesar do amplo o alcance da revisão da literatura

que envolve localização de robôs terrestres equipados com sensores do tipo *Range Finder*, realizada neste trabalho, não foi identificado um consenso ou um método de referência para indicar as condições nas quais o robô está "perdido" a partir da covariância da pose.

2.3 Modelos de movimento

Ao longo das últimas décadas, diversos tipos de robôs de superfície foram desenvolvidos, desde aqueles sobre rodas [4], com pernas (ou patas) [33], ou ainda robôs que rastejam [34]. Entretanto, os veículos com rodas predominam na maioria das aplicações e há, inclusive, variações dentro desta categoria quanto à dinâmica de movimento.

Robôs que possuem por rodas convencionais, ou seja, rodas que giram em torno de um eixo fixo à plataforma do robô, podem ser representados por modelos cinemáticos semelhantes a meios de locomoção muito comuns, como bicicleta, triciclo ou carro [4, pp. 16-26]. Assim, o seu funcionamento não difere muito daquele de veículos terrestres convencionais.

Uma diferença importante entre robôs sobre rodas e outros sistemas mecânicos robóticos são as restrições não holonômicas, que constituem restrições cinemáticas entre a roda e o solo. Por exemplo, o veículo autônomo CaRINA 2 [35] mostrado na Fig. 2.7 (um carro de passeio adaptado) não é capaz de se mover lateralmente, girar ou realizar curvas muito fechadas. Restrições não holonômicas são relações cinemáticas entre velocidades pontuais e velocidades angulares que não podem ser integradas na forma de relações algébricas entre variáveis de deslocamento translacional e rotacional. O resultado dessa falta de integrabilidade leva à falta de uma relação um-para-um entre as variáveis cartesianas e as variáveis conjuntas [36].

Portanto, na prática, os robôs com tração diferencial ou com rodas omnidirecionais são mais comuns, principalmente aqueles de pequeno porte [4, p. 15] que trafegam em ambientes *indoor*, visto que o espaço para manobras geralmente é reduzido nesses espaços. Além disso, a modelagem da navegação também se torna mais simples quando são utilizados robôs diferencial ou com rodas omnidirecionais.

Um veículo com tração diferencial como o da Fig. 2.8a, por exemplo, é capaz de girar (rotacionar) mantendo sua posição. Ainda assim, o movimento lateral (perpendicular à sua direção) não pode ser realizado devido à restrições cinemáticas. O veículo com rodas omnidirecionais (Fig. 2.8b), por sua vez, é capaz de realizar quaisquer movimentos de rotação e translação, inclusive simultaneamente. Esta liberdade de movimento é possível graças às rodas de construção complexa conhecidas como *Mekanum wheels* [36, p. 457].

Figura 2.7: Veículo Autônomo CaRINA 2 (cortesia do Laboratório de Robótica Móvel - ICMC/USP)

2.3.1 Robôs diferenciais

Neste trabalho, utilizou-se o robô do tipo *skid-steer* exibido na Fig. 2.8a, o Husky, devido a sua robustez em diversas superfícies e suporte ao desenvolvimento de novas aplicações. Neste veículo, as rodas dianteiras e traseiras de cada lado estão ligadas mecanicamente para girar na mesma velocidade, e cada um dos lados é acionado com velocidades distintas. Isso é feito com duas transmissões separadas e independentes: uma para as rodas do lado esquerdo e outra para as rodas do lado direito [4, p. 15].

Apesar do robô do tipo *skid-steer* possuir uma modelagem cinemática particular [37], é possível aproximá-lo a um robô com rodas diferenciais baseando-se apenas na velocidade das rodas. Um sistema de odometria realiza a leitura da velocidade angular das rodas em tempo real para obter as velocidades lineares de cada lado do Husky no instante t:

$$v_{l,t} = \pi D \cdot \omega_{l,t} \tag{2.20}$$

$$v_{r,t} = \pi D \cdot \omega_{r,t}, \tag{2.21}$$

onde D é o diâmetro das rodas do Husky (0,33 m), $\omega_{l,t}$ e $\omega_{r,t}$ são as velocidades angulares das rodas do lado esquerdo e direito, respectivamente.

A partir das velocidades de cada lado do Husky, é possível obter, de maneira aproximada, a taxa na qual o robô se desloca ou rotaciona em cada instante t, ou seja, suas velocidades linear v_t^o e angular ω_t^o, através das equações recomendadas pelo fabricante [38]:

$$v_t^o = \frac{v_{r,t} + v_{l,t}}{2} \tag{2.22}$$

(a) Robô com tração diferencial: Clearpath Husky A200

(b) Robô com rodas omnidirecionais: Highly maneuverable FIRST® Robot[4], por Dave Lundy[5]

Figura 2.8: Robôs de pequeno porte com rodas diferenciais e omnidirecionais.

$$\omega_t^o = \frac{v_{r,t} - v_{l,t}}{\mathrm{w}}, \tag{2.23}$$

onde w é a distância entre os centros das rodas de cada lado, que no Husky é igual a $0,555$ m. A odometria, que no contexto deste trabalho consiste no subsistema do robô que calcula a distância percorrida em um intervalo de tempo a partir de sensores de movimento instalados nos eixos das rodas, incorpora as Eqs. (2.20) a (2.23).

As velocidades v_t^o e ω_t^o são sempre aproximadas porque diversos fatores podem afetar a acurácia da odometria, sendo que a calibragem dos pneus, as irregularidades no terreno, as eventuais derrapagens das rodas e a imprecisão do odômetro, são os principais deles. Além disso, os erros de aproximação são propagados ao longo do tempo, de modo que a incerteza sobre a pose sempre aumenta. Por isso, embora seja um mecanismo simples e de baixo custo, a odometria não costuma ser o único recurso utilizado no processo de localização.

Deve-se ressaltar que, alternativamente, é possível utilizar uma Unidade de Medição Inercial, ou *Inertial Measurement Unit* (IMU), dispositivo constituído por uma combinação de giroscópio e acelerômetro, para obtenção direta do valor de ω_t^o, e indireta do valor de v_t^o a partir da aceleração. Entretanto, o sistema de odometria baseado em IMU também está sujeito à mesma propagação de erro.

As informações extraídas da odometria são utilizadas para prever a pose do veículo a partir da última pose estimada. Basicamente, existem dois modelos matemáticos que podem ser utilizados para calcular a pose a partir da odometria: baseado

[4]Fotografia sob licença Creative Commons (CC BY-ND 2.0)

[5]`https://www.flickr.com/photos/lundyd/47832359782`

em deslocamento ou baseado em velocidade [3, pp. 121-139].

2.3.2 Modelo de movimento baseado em deslocamento

No modelo de movimento baseado em deslocamento, as variações linear $\delta_{d,t}$ e angular $\delta_{\theta,t}$ do veículo no intervalo Δt são utilizadas para calcular a nova pose $\mathbf{p}_t$ partindo da pose anterior $\mathbf{p}_{t-1}$, conforme ilustra a Fig. 2.9. Os valores de $\delta_{d,t}$ e $\delta_{\theta,t}$ podem ser diretamente obtidos do sistema de odometria ou indiretamente a partir das velocidades v_t^o e ω_t^o das Eqs. (2.22) e (2.23). Portanto, o rastreamento da pose baseado no modelo de movimento por deslocamento é dado pela Eq. (2.24):

$$\mathbf{p}_t = \mathbf{p}_{t-1} + \begin{pmatrix} \delta_{d,t} \cos \theta_{t-1} \\ \delta_{d,t} \sin \theta_{t-1} \\ \delta_{\theta,t} \end{pmatrix} \tag{2.24}$$

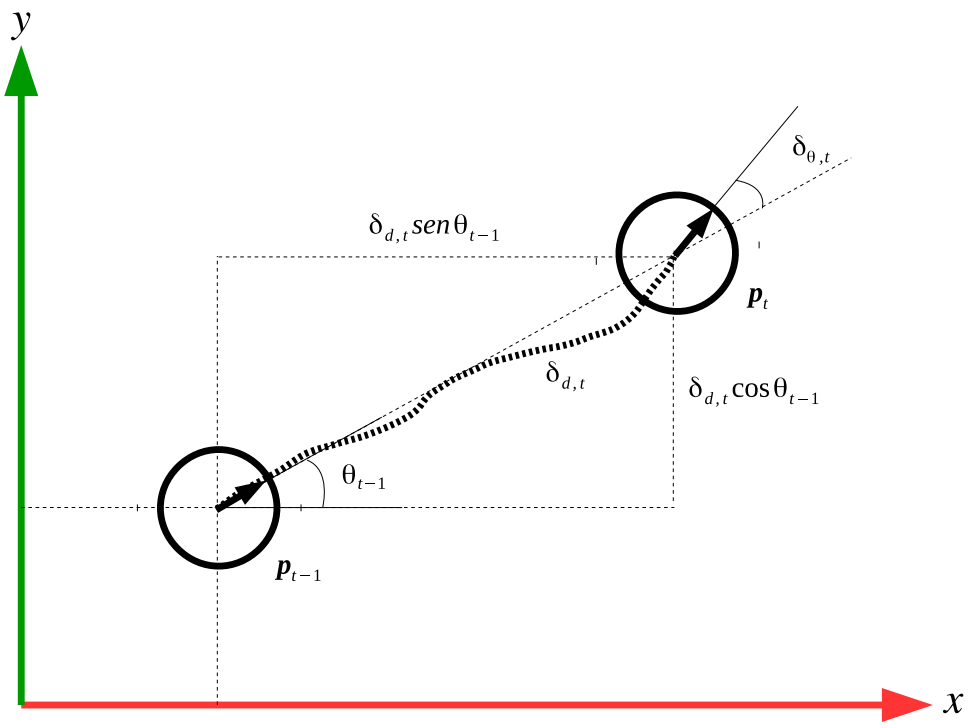


Figura 2.9: Modelo de movimento por deslocamento.

2.3.3 Modelo de movimento baseado em velocidade

Na representação da odometria baseada em velocidade, o sistema de odometria fornece os valores de v_t^o e ω_t^o no intervalo de tempo Δt. Quando o veículo se movimenta de forma retilínea ($\omega_t^o = 0$) a nova pose é obtida através da Eq. (2.24), assumindo

$\delta_{\theta,t} = 0$ e obtendo o deslocamento linear $\delta_{d,t}$ pela equação de Movimento Retilíneo Uniforme (MRU):

$$\delta_{d,t} = v_t^o \Delta t \tag{2.25}$$

Por outro lado, quando v_t^o e ω_t^o são não-nulos, o robô segue uma trajetória circular sobre uma circunferência de raio r:

$$r = \left| \frac{v_t^o}{\omega_t^o} \right| \tag{2.26}$$

no intervalo Δt e Centro Instantâneo de Rotação (CIR) em (x_c, y_c), como mostra a Fig. 2.10. No Movimento Circular Uniforme (MCU), as coordenadas do CIR podem ser obtidas a partir de qualquer pose $\mathbf{p}_{t-1}$ sobre esta circunferência:

$$\begin{pmatrix} x_c \\ y_c \end{pmatrix} = \begin{pmatrix} x_{t-1} - \frac{v_t^o}{\omega_t^o} \sin\theta_{t-1} \\ y_{t-1} + \frac{v_t^o}{\omega_t^o} \cos\theta_{t-1} \end{pmatrix}. \tag{2.27}$$

Portanto, após o intervalo Δt o robô terá a orientação $\theta_t = \theta_{t-1} + \omega_t^o \Delta_t$ e as novas coordenadas:

$$\begin{pmatrix} x_t \\ y_t \end{pmatrix} = \begin{pmatrix} x_c + \frac{v_t^o}{\omega_t^o} \sin\theta_t \\ y_c - \frac{v_t^o}{\omega_t^o} \cos\theta_t \end{pmatrix}. \tag{2.28}$$

Substituindo (2.27) em (2.28) obtém-se a equação de modelo de movimento baseado em velocidade [3, p. 128]:

$$\mathbf{p}_t = \mathbf{p}_{t-1} + \begin{pmatrix} -\frac{v_t^o}{\omega_t^o} \sin\theta_{t-1} + \frac{v_t^o}{\omega_t^o} \sin\left(\theta_{t-1} + \omega_t^o \Delta t\right) \\ \frac{v_t^o}{\omega_t^o} \cos\theta_{t-1} - \frac{v_t^o}{\omega_t^o} \cos\left(\theta_{t-1} + \omega_t^o \Delta t\right) \\ \omega_t^o \Delta t \end{pmatrix}, \ \omega_t^o \neq 0. \tag{2.29}$$

Os modelos de movimento baseados em deslocamento e velocidade determinam a pose *a priori* $\mathbf{p}_t$ a partir da pose anterior e da odometria referente ao intervalo Δ_t. Porém, a odometria por deslocamento (Eq. 2.24) assume que o veículo faz primeiro uma translação retilínea seguida de uma rotação no final. Neste caso, o modelo não prevê os movimentos de translação e rotação simultâneos. Na odometria por velocidade (Eq. 2.29), o modelo também é aproximado, uma vez que o robô normalmente não mantém suas velocidades linear e angular constantes por muito tempo e nem as modifica de maneira abrupta. Em outras palavras, o modelo não prevê a aceleração e desaceleração. Por isso, em ambos os casos o intervalo Δ_t deve ser o curto, de forma a limitar a divergência entre a trajetória real e o movimento aproximado pelo modelo matemático.

2.4 Mapa do ambiente

As seções anteriores mostraram que a pose do robô costuma ser dada em relação a um sistema de coordenadas global que representa o ambiente no qual ele está

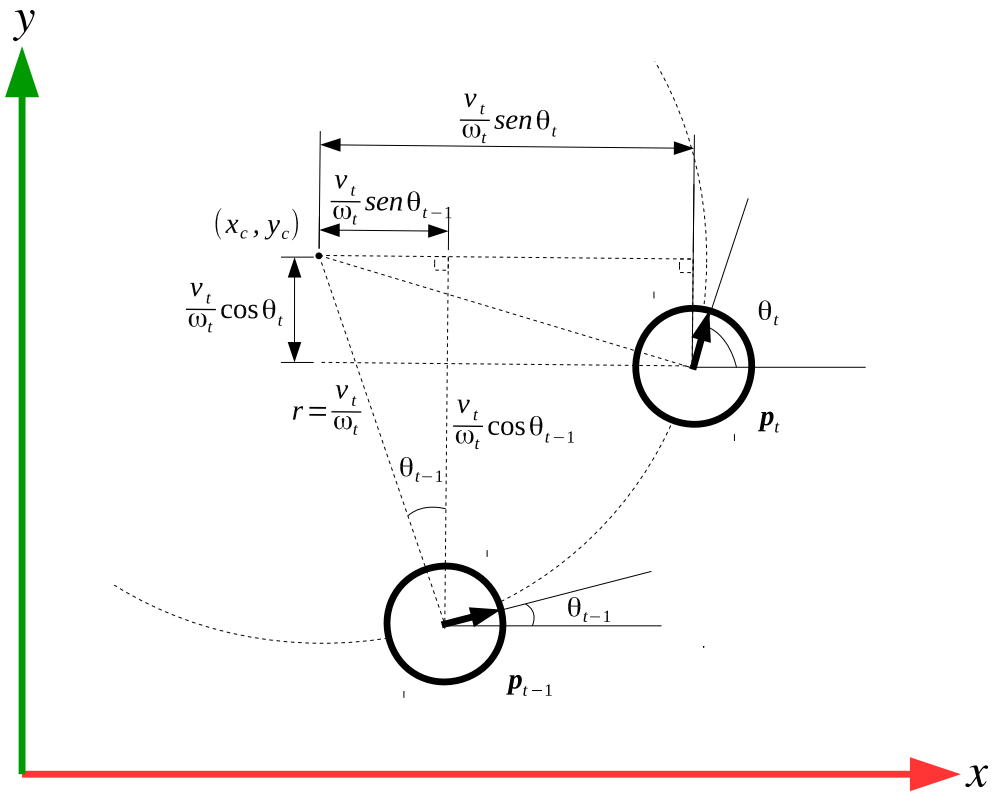


Figura 2.10: Modelo de movimento por velocidade.

inserido. Entretanto, não foi discutido como a pose do robô em relação ao ambiente pode ser estimada. Este é o papel dos algoritmos de localização descritos no capítulo 3 e, conforme será apresentado, todos eles são baseados em um mapa previamente construído. Esta seção descreve a representação de mapa utilizada por estes algoritmos.

Em robótica móvel, a navegação baseada apenas em sensores proprioceptivos (como o odômetro, por exemplo), abordagem conhecida como navegação estimada (*dead reckoning*, em inglês), está limitada a aplicações restritas. O principal motivo é que este método está sujeito a um elevado grau de incerteza sobre a pose após um longo percurso porque o erro associado a este tipo de informação é cumulativo [1, p. 155-160]. Por isso, é comum que os robôs móveis dependam também de alguma percepção do ambiente para navegar, interagir e se localizar. Neste caso, é necessário, também, dispor de um mapa do ambiente para que possa estabelecer uma relação entre o que se vê e o que está contido no mapa. Desta forma, é possível estimar sua pose no espaço.

Nos últimos anos, tem sido bastante estudada a técnica de Localização e Mapeamento Simultâneos, ou *Simultaneous Localization and Mapping* (SLAM), onde o robô explora o espaço sem conhecê-lo, mas constrói o seu mapa dinamicamente à medida em que navega e realiza a percepção do ambiente [7, 39–42]. Ainda assim,

em diversas aplicações [9, 22, 43–49] o robô navega dentro de um espaço conhecido e razoavelmente estático, de forma que o mapa pode ser previamente elaborado automaticamente utilizando o SLAM ou manualmente.

Existem diferentes formas de representar um mapa digital, porém a mais comum em robótica móvel de superfície é ilustrada na Fig. 2.11. O espaço é representado em 2D e dividido em pequenas células, onde aquelas que abrangem pelo menos uma parte dos obstáculos em azul são marcadas como ocupadas (em preto), enquanto as demais representam espaço livre (em branco). Fica evidente, portanto, que a fidelidade do mapa é diretamente proporcional à granularidade das células.

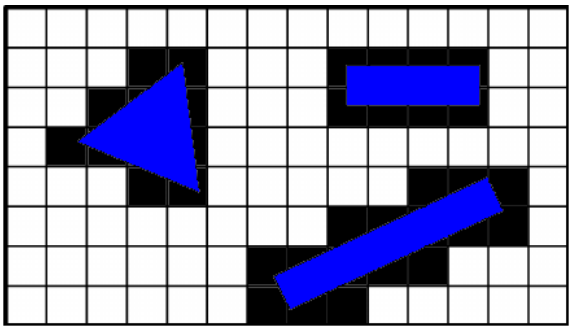


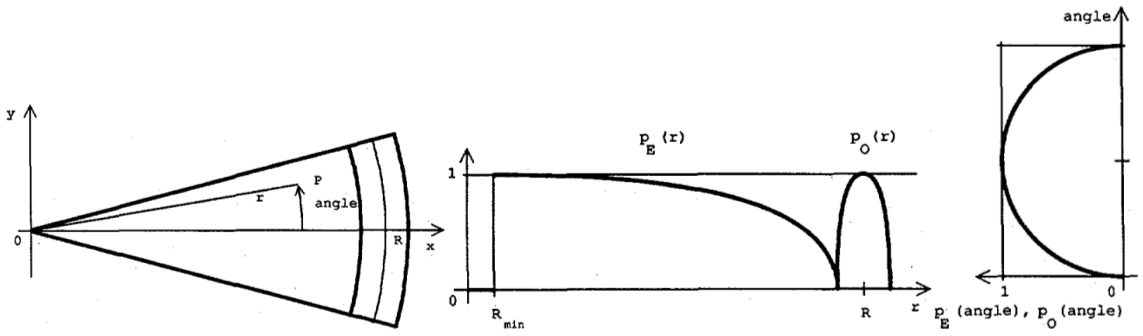
Figura 2.11: Grade de ocupação com decomposição morfológica. Os polígonos em azul constituem os obstáculos detectados, enquanto as células em preto correspondem aos respectivos contornos destes obstáculos; as células em branco representam a região livre.

Na década de 80, Elfes [50] observou que decompor o ambiente em uma matriz de células que descrevem áreas ocupadas ou livres é suficiente para permitir que o robô navegue em ambientes não-estruturados. Em seus experimentos, esta matriz foi concebida de maneira automatizada por meio da combinação de várias leituras de um anel de sonares processadas através de funções de densidade de probabilidade. Esta matriz, também chamada de grade de ocupação (ou *occupancy grid*, em inglês), tem sido amplamente utilizada desde então.

No trabalho de Elfes, a leitura de um sonar que detecta um obstáculo encontrado a uma distância $\mathtt{R}$ do robô é caracterizada por um feixe em formato de uma "fatia de pizza", como mostra a Fig. 2.12a. De certa forma, este feixe expõe a probabilidade $\mathtt{p_E(r)}$ de qualquer ponto $\mathtt{P}$ a uma distância $\mathtt{r} \in [\mathtt{R_{min}}, \mathtt{R}]$, dentro do espaço compreendido entre o alcance mínimo do sensor ($\mathtt{R_{min}}$) e a frente do feixe em $\mathtt{R}$, estar livre. Também é possível deduzir a probabilidade $\mathtt{p_O(r)}$ de algum ponto $\mathtt{P}$ estar em uma região ocupada, provavelmente próxima de $\mathtt{R}$. Neste caso, deve-se levar em consideração o erro máximo da leitura ϵ, de forma que, ao detectar um obstáculo, este provavelmente se encontre entre $\mathtt{R}-\epsilon$ e $\mathtt{R}+\epsilon$ (conforme mostra a "borda" da fatia de pizza na Fig. 2.12a). A Fig. 2.12b mostra a sobreposição dos perfis das

probabilidades $p_E(r)$ e $p_O(r)$ propostas por Elfes, evidenciando que a probabilidade de medições antes de R_{min} é nula.

As probabilidades $p_E(r)$ e $p_O(r)$ de cada ponto P também dependem do ângulo entre P e o eixo x do feixe (angle na Fig. 2.12a). Conforme mostra a Fig. 2.12c, estas probabilidades são iguais, visto que a incerteza sobre a medição nas bordas da fatia é menor do que em seu eixo central.



(a) Abrangência de um feixe emitido pelo sonar

(b) Perfil de probabilidade para r

(c) Perfil de probabilidade para angle

Figura 2.12: Caracterização da probabilidade da detecção de um obstáculo pelo sonar. Figura extraída de Elfes (1987) [50]

A partir da sobreposição de medições consecutivas, o valor final atribuído a cada célula cell da grade é dado pela comparação entre os valores de $p_E(cell)$ e $p_O(cell)$. A quantidade de células atualizadas depende do tipo de sensor, como mostra a Fig. 2.13, com exemplos de um sonar e um sensor a laser com tecnologia *Light Detection And Ranging* (LiDAR). No mapeamento com sonar (à esquerda), que possui um ângulo de abertura maior, deve-se atualizar todas as células dentro do cone de leitura (Fig. 2.12a). Por outro lado, no mapeamento a laser (à direita), que apresenta três feixes independentes bem focados, pode-se atualizar apenas as células que são atravessadas pelo feixe [1, p. 181] utilizando, por exemplo, o algoritmo de Bresenham (introduzido na subseção 2.5.2).

A grade de ocupação é uma matriz $M \times N$, onde cada célula representa uma pequena área do ambiente com dimensões $\Delta \times \Delta$. O valor de Δ, portanto, estabelece a resolução do mapa, que é de 5 cm na maioria dos casos [9, 49, 51, 52], mas pode assumir diferentes valores, como $2,5$ cm [22,53] ou até maiores que 10 cm [46,47,50], por exemplo. A resolução deste mapa normalmente está associada ao tamanho do ambiente e à memória do robô – quanto maior a resolução, melhor a representação do espaço e mais espaço será ocupado na memória.

Uma parte da literatura [1,3] considera que a grade de ocupação utiliza valores binários para especificar células vazias (0) ou ocupadas (1). Entretanto, na prática

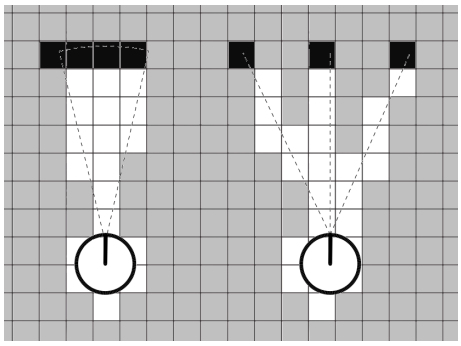


Figura 2.13: Exemplos ilustrativos de mapeamento por grade de ocupação utilizando sonar (esq.) e LiDAR (dir.)

[4, p. 471] a grade de ocupação é armazenada em uma imagem monocromática, onde cada *pixel* indica uma célula obstruída (em preto), livre (em branco) ou desconhecida (em cinza), como mostra a Fig. 2.14a, que mapeou o LaR em novembro de 2021. Sendo assim, é comum que alguns algoritmos de localização necessitem decompor a grade de ocupação em uma matriz binária, como mostra a Fig. 2.14b que foi obtida utilizando a biblioteca OpenCV[6].

(a) Mapa monocromático (b) Decomposição binária

Figura 2.14: Grade de ocupação do LaR (em nov/2021): cada *pixel* em (a) constitui uma célula de 25 cm^2; *pixels* em branco são células livres e *pixels* em preto são obstáculos; regiões não exploradas são marcadas em cinza. Em (b), apenas os contornos são representados (em preto). Mapa monocromático construído utilizando um sensor com tecnologia LiDAR e o algoritmo Hector SLAM [39] e a decomposição binária utilizando a biblioteca OpenCV. As imagens foram recortadas na largura para melhor visualização

Geralmente, a grade de ocupação é construída iterativamente à medida que o

[6]Disponível em `https://opencv.org`

robô se move, sendo imprescindível que, de alguma forma, a pose seja conhecida durante do mapeamento. O algoritmo de SLAM GMapping [54], por exemplo, utiliza a odometria para atualizar a pose, enquanto o Hector SLAM [39] se baseia apenas na correspondência entre nuvens de pontos consecutivas para estimar o deslocamento – e consequentemente atualizar a pose.

Inicialmente, todas as células da grade de ocupação são marcadas em cinza (desconhecidas). A partir da primeira leitura, as células dentro do campo de visão do sensor são atualizadas, de forma que aquelas que coincidem com a direção e o alcance medidos pelo sensor são marcadas em preto, e aquelas entre o sensor e os pontos são marcadas em branco – assume-se que correspondem à região livre. Ao combinar as evidências de dezenas ou centenas de leituras à medida que o robô se move, a área reconhecida como livre é expandida. As regiões possivelmente ocupadas também aumentam, enquanto a imprecisão dessas regiões diminui. Aquelas regiões que não foram alcançadas pelo sensor são mantidas em cinza.

A grade de ocupação é muito utilizada devido à sua boa relação de compromisso entre complexidade e fidelidade. Enquanto outras abordagens normalmente consistem em uma elaboração manual ou assistida, as grades de ocupação podem ser feitas de modo automático enquanto o robô navega. Além disso, a resolução da grade pode ser adaptada para obter o melhor custo-benefício entre fidelidade do mapa e desempenho do algoritmo de localização. Por outro lado, ressalta-se que a grade de ocupação constitui apenas uma seção transversal dos objetos encontrados na mesma altura do sensor, de modo que obstáculos acima ou abaixo deste campo de visão não são representados no mapa. No caso da Fig. 2.14, a grade do ocupação representa os obstáculos a uma altura aproximada de 40 cm.

2.5 Percepção do ambiente

Praticamente todos os robôs móveis que executam tarefas dependem de sensores para reconhecer o ambiente em sua volta, identificar obstáculos, interagir com objetos e navegar, entre outros objetivos. Estes sensores podem estar acoplados ao robô ou instalados no ambiente, embora o primeiro caso seja mais comum. O importante, contudo, é que sejam produzidas informações relevantes para que o robô consiga determinar, em relação a si, a posição daquilo que foi capturado pelo sensor (utilizando as transformações descritas na seção 2.1).

De modo geral, os sensores montados em um robô se dividem entre proprioceptivos e exteroceptivos [55, p. 101]. O primeiro tipo é dedicado à observação de variáveis internas, como aquelas apresentadas na seção 2.3 (velocidade das rodas), ou ainda outras medições de natureza mecânica (abertura de uma válvula) ou elé-

tricas (tensão aplicada ao motor), por exemplo. Estas variáveis constituem o estado do robô, como descrito no início deste capítulo, e são especialmente úteis para os modelos de movimento.

Os sensores exteroceptivos, por sua vez, realizam a leitura de variáveis externas que oriundas do espaço no qual ele se encontra. Existe uma grande variedade de sensores que fazem leituras do ambiente, podendo capturar medidas de distância, luminosidade, pressão, campo magnético, sinais de radiofrequência (RF) e acústicos, apenas para citar alguns exemplos genéricos. Entretanto, os mais comuns em robótica móvel são aqueles que produzem informações úteis no processo de navegação e localização, principalmente a posição relativa dos objetos ou sinais do ambiente. A bússola, por exemplo é um sensor magnético relevante na localização, contribuindo para a estimação da orientação do veículo em relação à direção do campo magnético da terra [1, p. 5]. A tabela 2.1 reúne alguns dos sensores mais utilizados em robótica móvel e estão classificados de acordo com o tipo de informação que coletam.

Tabela 2.1: Classificação dos sensores típicos em robôs móveis. Adaptado de Siegwart *et al.* [55, p. 104]

Tipo	Classificação geral	Tecnologia
Proprioceptivos	Sensores de posição/ movimento	Potenciômetro
		Encoder de roda
	Sensores inerciais	Giroscópio
		Acelerômetro
Exteroceptivos	Sensores táteis	Interruptor de contato
		Sensor de proximidade
	Sensores de orientação	Magnetômetro
		Inclinômetro
	Sistemas de sinalização	GPS
		Baliza luminosa
		Baliza de ultrassom/RF
	Sensores de alcance	Sensor ultrassônico
		Laser-scanner
	Sensores visuais	Câmera CCD/CMOS
		Câmera RGB-D

Os sensores exteroceptivos são essenciais para a percepção do robô, e portanto a confiabilidade da localização está diretamente relacionada à qualidade das informações produzidas por eles. Dentre os sensores exteroceptivos da Tabela 2.1, o *Laser-scanner* é um dos mais utilizados [27, p. 105] devido a sua baixa latência e

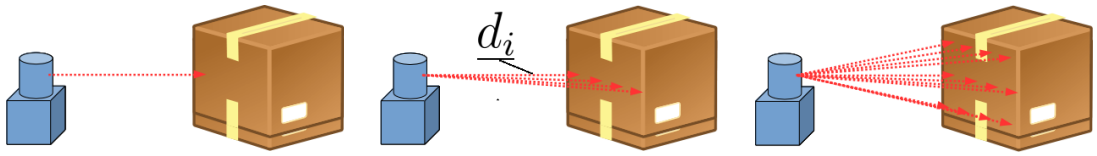
alta precisão. Além disso, o fato de produzir dados métricos, ao contrário da câmera convencional que coleta imagens, por exemplo, torna mais simples a conversão para os sistemas de coordenadas [3, p. 179]. Devido a estas duas principais vantagens, o *Laser-scanner* é o sistema de percepção preferencial para veículos autônomos [30, p. 222], ainda que, por outro lado, as câmeras apresentem outras vantagens que se destacam quando há limitações de custo e de estruturação – câmeras convencionais geralmente são mais baratas e capturam mais informações do ambiente do que o laser, ao custo de fornecer menor precisão e demandar processamento de imagem. Este trabalho se concentra, portanto, na localização baseada em varredura a laser.

2.5.1 Varredura a laser

Os sensores de varredura a laser modernos utilizados em robótica móvel possuem a tecnologia LiDAR e fornecem um conjunto ordenado de medições, frequentemente denominado nuvem de pontos (ou *point cloud*, em inglês). Cada medição é representada por um dado estruturado, composto pela distância até o objeto, a respectiva direção (ou ângulo) da medição e, em alguns casos, a intensidade da luz refletida. Dentre outras vantagens desta tecnologia, destacam-se a precisão espacial, alta taxa de transferência e relativa independência de fatores externos (prescinde a iluminação do ambiente, por exemplo).

De modo geral, os sensores LiDAR que medem a distância direta em direção até um anteparo são conhecidos como unidimensionais, ou sensor 1D. Se o emissor do feixe de laser estiver girando em um eixo, múltiplas medições podem ser obtidas em diferentes direções. Cada medição é constituída por uma distância e um ângulo, de forma que um conjunto dessas medições formam uma nuvem de pontos em sobre os contornos de anteparos próximos. Sensores deste tipo são denominados 2D *Laser-scanners*.

Por último, há sensores que fornecem uma nuvem de pontos em 3D, onde cada medição k representa um ponto com coordenadas x_k, y_k e z_k. A maioria dos sensores deste tipo utilizam múltiplas camadas de medições em 2D para construir uma nuvem de pontos em 3D com campo de visão limitado. Na prática, a leitura equivale àquela oriunda de um conjunto de sensores 2D posicionados na mesma origem mas com uma pequena diferença angular entre si. Ainda que a nuvem de pontos seja representada em 3D, estes sensores são frequentemente chamados de "2.5D" [56] devido ao fato de que apenas as faces frontais dos objetos podem ser capturados, de forma que as faces ocultas, ou ainda, demais objetos localizados por trás, ficam invisíveis na nuvem. A Fig. 2.15 ilustra os diferentes tipos de sensores LiDAR quanto ao campo de detecção.



(a) 1D (b) 2D (c) 3D (ou 2.5D)

Figura 2.15: Diferentes tipos de sensores LiDAR

A Fig. 2.15b exibe um um sensor 2D *Laser-scanner* que mede a distância livre d_i à sua frente para cada feixe i. As medições estão ordenadas da esquerda para direita, e igualmente espaçadas entre si por um ângulo α. Sendo assim, a posição relativa da i-ésima medição no *frame* do sensor é obtida através da eq. (2.2), dados $\rho = d_i$ e $\phi = (i\alpha + q)$, onde q é o ângulo da primeira medição (ou seja, para $i = 0$). As coordenadas do ponto detectado em relação ao robô são obtidas através da Eq. (2.6), e de maneira análoga obtêm-se as coordenadas no mapa $({}^W x_i, {}^W y_i)$ utilizando a transformada homogênea ${}^W T_V$.

Neste trabalho, utilizou-se o LiDAR 2D SICK LMS10x[7]. Este sensor possui um campo de visão de 270° e produz uma nuvem de até 720 pontos e, de acordo com o fabricante, o erro de medição está limitado a 30 mm, dependendo das condições do ambiente [57]. O alcance deste sensor depende da proporção de luz que é refletida difusamente pelo anteparo em relação na proporção que seria refletida em uma superfície branca. A Fig. 2.16 demonstra, em rosa claro, o alcance do sensor LMS10x para superfícies com até 10% de reflectância (até 18 metros), e em rosa escuro para as demais superfícies (até 20 metros).

2.5.2 Modelos de percepção

Os sensores que medem o alcance até alguma superfície, como laser e sonar, são os mais populares em robótica. Entretanto, é necessário lidar com as limitações inerentes a este tipo de medição já citadas anteriormente, como ruídos, presença de *outliers* e falhas de medição. Por isso, é comum a utilização de modelos matemáticos de percepção que incorporam estas limitações ou que calculam o grau de afinidade entre a medição e o mapa. Esta subseção discorre sobre as principais estratégias utilizadas para modelar a percepção do robô.

Cálculo do alcance esperado

Ao considerar uma medição de alcance, normalmente assume-se que a pose do sensor no mapa é conhecida. Portanto, a partir destas informações, o mecanismo de *ray tracing* permite prever a distância esperada em cada direção a partir do sensor.

[7] `https://www.sick.com`

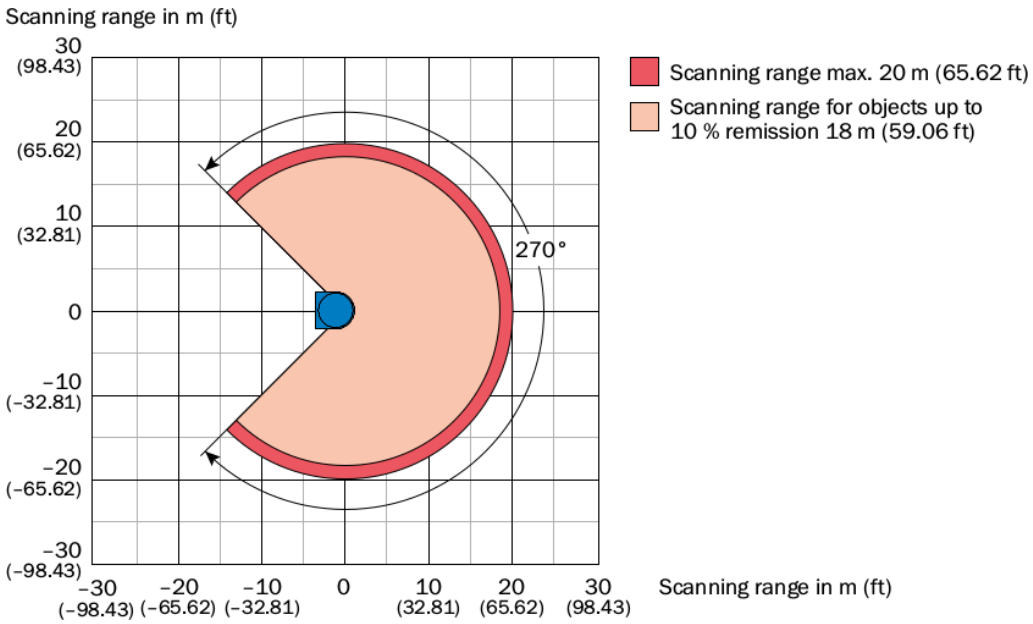


Figura 2.16: Alcance do sensor SICK LMS10x. Os eixos correspondem à distância, em metros, desde a origem do sensor. A regiões coloridas delimitam o campo de visão, de forma que a faixa em rosa escuro determina o alcance apenas para superfícies com reflectância superior a 10%. Ilustração extraída do manual de informações do produto [57]

Utilizando uma variação do algoritmo de Bresenham [58], por exemplo, é possível realizar o *ray tracing* em todas as direções a partir da pose, como ilustra a Fig. 2.17, e portanto prever a nuvem de pontos esperada por um sensor montado neste robô.

O algoritmo de Bresenham foi um dos primeiros desenvolvidos para computação gráfica, originalmente concebido para permitir que o computador controle um tipo de impressora digital, especialmente para desenhar linhas. Uma vez que utiliza apenas operações de adição, subtração e deslocamento de bits em números inteiros, o algoritmo é leve e por este motivo tem sido amplamente utilizado desde então em diversas áreas da computação. Em robótica móvel, este algoritmo pode ser utilizado para calcular o alcance esperado do sensor a partir de sua posição na grade de ocupação.

De forma sucinta, o algoritmo original aplicado à impressora assume que o espaço disponível para realizar o traçado é composto por células de tamanho fixo cujo centro de cada uma delas pode ser visitado pela cabeça de impressão. Então, o algoritmo recebe os pontos inicial $(x_0, y_0) \in \mathbb{Z}^2$ e final $(x_1, y_1) \in \mathbb{Z}^2$ da reta desejada e orienta a cabeça de impressão para visitar (e "pintar") todas as células mais próximas que são cruzadas pela reta.

Este mesmo método é utilizado na localização em robótica para realizar o *ray casting* da posição do sensor, ou seja, o cálculo do alcance esperado. Mais especifi-

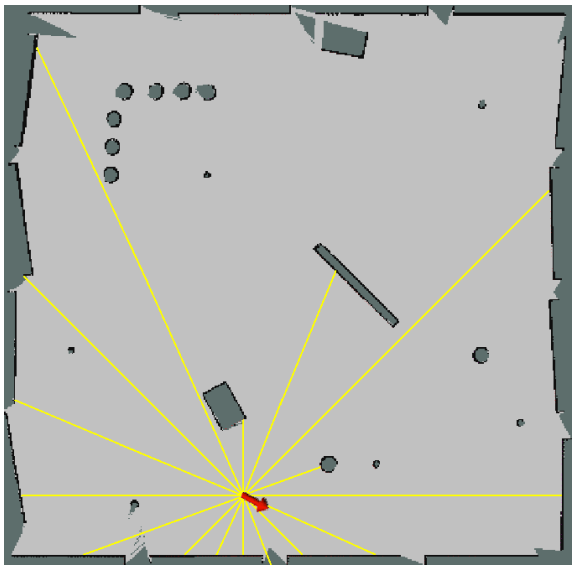


Figura 2.17: *Ray tracing* a partir de uma pose. Os raios em amarelo ilustram os alcances esperados por um sensor a laser a partir de sua pose (seta em vermelho)

camente, a partir da posição do sensor (x_0, y_0) e utilizando a grade de ocupação, é possível determinar quais os alcances esperados para cada feixe k utilizando o algoritmo de Bresenham. A equação geral deste algoritmo é definida como uma linha entre os dois pontos recebidos, dada por:

$$\frac{y - y_0}{y_1 - y_0} = \frac{x - x_0}{x_1 - x_0}, \tag{2.30}$$

onde $x, y \in \mathbb{R}$ são coordenadas intermediárias da reta.

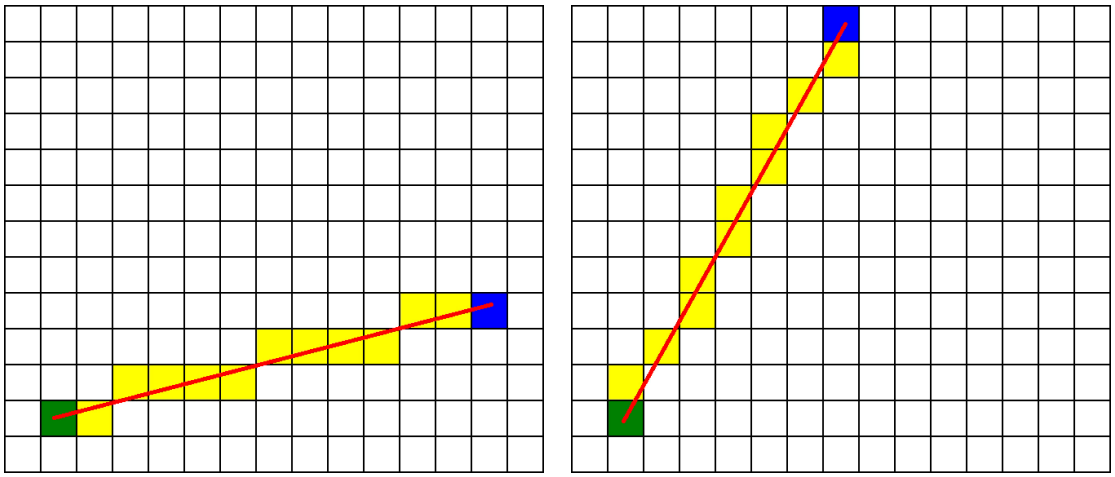
Logo no início, o algoritmo calcula a inclinação desta reta para determinar o eixo primário, ou seja, aquele eixo que terá incrementos unitários. A Fig. 2.18 mostra duas retas com inclinações diferentes, onde ambas iniciam na célula em verde e terminam na célula em azul. Observa-se que na Fig. 2.18a o eixo x é primário, pois em cada coluna há apenas uma célula visitada (em amarelo), enquanto na Fig. 2.18b, de forma análoga em relação às linhas, o eixo y é primário.

Esta inclinação é obtida pela razão entre as distancias na vertical $\Delta y = (y_1 - y_0)$ e horizontal $\Delta x = (x_1 - x_0)$ dos pontos terminais – neste caso em especial, assume-se que $(0 \leq x_0 < x_1)$ e $(0 \leq y_0 < y_1)$. Têm-se x como eixo primário quando $\frac{\Delta y}{\Delta x} \leq 1$, e y caso contrário. No primeiro caso, o valor de x é conhecido devido aos incrementos unitários ($x \in [x_0, x_0 + 1, x_0 + 2, ...]$) e portanto o valor de y é dado pelo inteiro mais próximo:

$$y = \left\lfloor \Delta y \frac{x - x_0}{\Delta x} + y_0 + \frac{1}{2} \right\rfloor. \tag{2.31}$$

De forma análoga, no segundo caso, y é conhecido e o valor de x é obtido através de:

$$x = \left\lfloor \Delta x \frac{y - y_0}{\Delta y} + x_0 + \frac{1}{2} \right\rfloor. \tag{2.32}$$



(a) Traçado quando x é o eixo primário (b) Traçado quando y é o eixo primário

Figura 2.18: Retas traçadas pelo algoritmo de Bresenham em uma grade de células de largura unitária

Na aplicação de traçado de raio do sensor, tomando como exemplo o caso onde x é o eixo primário, o algoritmo de Bresenham deve interromper os incrementos quando uma das coordenadas $(x_i, \lfloor y_i \rfloor)$ ou $(x_i, \lceil y_i \rceil)$ caírem sobre uma célula ocupada da grade de ocupação. Neste caso, tem-se que o alcance esperado do sensor para aquela abertura angular k no instante t é $z_t^{k*} = (x_i, y_i)$. Por outro lado, o alcance esperado pode nunca atingir uma célula ocupada, e neste caso o algoritmo deve interromper no alcance máximo nominal do sensor $z_{\max}$. Neste caso, $z_t^{k*} = (x_i, y_i)$ sempre que $\sqrt{{x_i}^2 + {y_i}^2} \geq z_{\max}$.

Modelo probabilístico da medição de alcance

Conforme discutido por Elfes [50] e Thrun *et al.* [3], os sensores exteroceptivos estão sujeitos a problemas inerentes (sensibilidade e precisão limitadas, falsas medições devido à reflexões do sinal, interferências externas, entre outros) que corrompem a leitura direta. Nesse contexto, o modelo probabilístico da medição de alcance é utilizado para incorporar explicitamente os erros causados pelos fatores citados. De acordo com este modelo, existem quatro fatores que afetam a medição do alcance do sensor [3, pp. 153-157]:

- precisão da medição, de forma que o alcance medido pode ser ligeiramente maior ou menor que o valor correto;
- obstrução por objetos próximos (*outliers*) que fazem o alcance medido ser menor do que o correto;
- ausência de reflexão do sinal, de modo que o sensor reporta o alcance máximo $z_{\max}$;

- medições aleatórias causadas por múltiplas reflexões ou interferência externa (*crosstalk* com outros sensores, por exemplo).

Os quatro fatores citados anteriormente são modelados através de funções de densidade de probabilidade, $\mathrm{p}_{\mathrm{hit}}(z_t^k, x_t)$, $\mathrm{p}_{\mathrm{short}}(z_t^k, x_t)$, $\mathrm{p}_{\mathrm{max}}(z_t^k, x_t)$ e $\mathrm{p}_{\mathrm{rand}}(z_t^k, x_t)$, nesta ordem, onde z_t^k é a medição em 1D obtida do k-ésimo feixe no instante t e x_t é o estado do robô neste instante – especificamente sua pose. Estas expressões são interpretadas como "a probabilidade de ocorrer a medição z_t^k a partir da pose definida por x_t". Visto que x_t é o termo comum, o mesmo será ocultado nas equações a seguir para facilitar a legibilidade, restando, portanto, $\mathrm{p}_{\mathrm{hit}}(z_t^k)$, $\mathrm{p}_{\mathrm{short}}(z_t^k)$, $\mathrm{p}_{\mathrm{max}}(z_t^k)$ e $\mathrm{p}_{\mathrm{rand}}(z_t^k)$.

A descrição da percepção através de funções de densidade de probabilidade é uma forma de incorporar ao modelo as incertezas inerentes ao processo de sensoriamento. Aliás, os algoritmos de localização probabilísticos adotam esta forma de representação da percepção. Portanto, alguns conceitos pertinentes sobre probabilidade serão discutidos no Capítulo 3. Ressalta-se que as medições se limitam à faixa de operação do sensor, portanto, as funções de probabilidade da percepção normalmente são determinadas para a condição $0 \leq z_t^k \leq z_{\mathrm{max}}$.

O primeiro fator (hit) estabelece que a probabilidade de uma medição se referir ao obstáculo esperado é dada por uma distribuição gaussiana com média z_t^k e variância σ_{hit}^2:

$$\mathrm{p}_{\mathrm{hit}}(z_t^k) = \frac{1}{\sqrt{2\pi\sigma_{hit}^2}} e^{-\frac{1}{2}\frac{(z_t^k - z_t^{k*})^2}{\sigma_{hit}^2}} \quad . \tag{2.33}$$

A medição curta causada por *outliers* (short), por sua vez, é modelada por uma função exponencial, de modo que a probabilidade desta medição ocorrer diminui à medida que o alcance aumenta. Porém, apenas *outliers* que estão à frente do obstáculo esperado são detectados, de modo que este fator define probabilidade nula para medições além do esperado, ou seja, $\mathrm{p}_{\mathrm{short}}(z_t^k) = 0$, $\forall\ z_t^k \geq z_t^{k*}$, e uma função exponencial para medições mais próximas:

$$\mathrm{p}_{\mathrm{short}}(z_t^k) = \frac{1}{1 - e^{-\lambda_{\mathrm{short}} z_t^{k*}}} \lambda_{\mathrm{short}} e^{-\lambda_{\mathrm{short}} z_t^k} \ , \forall\ z_t^k < z_t^{k*} \ , \tag{2.34}$$

onde λ_{short} é um parâmetro da função exponencial.

Nos casos onde o obstáculo está além do alcance do sensor (max), ou a reflexão do sinal não é capturada por este, os sensores costumam reportar o alcance máximo do mesmo, z_{max}. Portanto, o modelo apresenta uma função massa de probabilidade no ponto de alcance máximo do sensor:

$$\mathrm{p}_{\mathrm{max}}(z_t^k) = \begin{cases} 1 \text{ se } z_t^k = z_{\mathrm{max}} \\ 0 \text{ caso contrário.} \end{cases} \tag{2.35}$$

Finalmente, as medições aleatórias (rand) por motivos indeterminados apresentam uma distribuição de probabilidade uniforme ao longo da faixa de alcance do sensor:

$$\mathrm{p}_{\mathrm{rand}}(z_t^k) = \frac{1}{z_{\max}} \ , \ \forall \ z_t^k < z_{\max} \tag{2.36}$$

Os gráficos das Eqs. 2.33, 2.34, 2.35 e 2.36 estão apresentados na Fig. 2.19a. A equação geral para a percepção, portanto, pode ser dada como a probabilidade conjunta destas quatro distribuições. Se a pose do sensor $\mathbf{p}_t$ e o mapa m são conhecidos, o modelo da medição de alcance, portanto, é dado pela probabilidade de z_t^k, dados $\mathbf{p}_t$ e m, ou seja $\mathrm{p}(z_t^k|\mathbf{p}_t, m)$. Entretanto, por ser estático, m é frequentemente ocultado da equação geral:

$$\mathrm{p}(z_t^k|\mathbf{p}_t) = z_1\mathrm{p}_{\mathrm{hit}}(z_t^k) + z_2\mathrm{p}_{\mathrm{short}}(z_t^k) + z_3\mathrm{p}_{\max}(z_t^k) + z_4\mathrm{p}_{\mathrm{rand}}(z_t^k) \ , \tag{2.37}$$

onde $z_i, i = 1:4$ são os parâmetros de ponderação dos fatores de modo que $z_1 + z_2 + z_3 + z_4 = 1$. Portanto, a Eq. (2.37) representa a distribuição de probabilidade da medição ao longo de cada k-ésimo feixe, de modo que os algoritmos de localização a utilizam para avaliar a aptidão de sua pose.

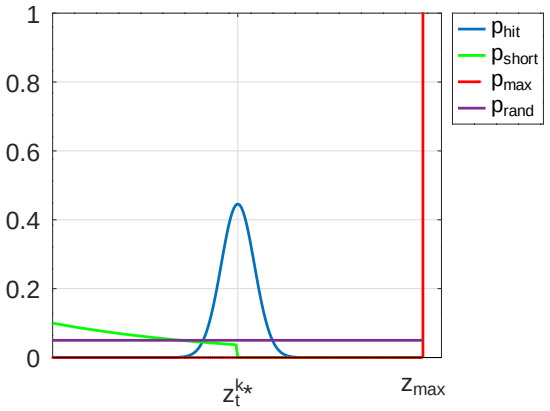
Os parâmetros σ_{hit}^2, λ_{short} e z_i devem ser ajustados conforme o tipo de sensor, as características do ambiente e a qualidade do mapa. A título de exemplo a Tabela 2.2 mostra três configurações para estes parâmetros de acordo com diferentes condições. A última linha desta tabela mostra a configuração típica utilizada para sensores do tipo LiDAR no algoritmo de localização AMCL, que por sua vez será apresentado na Seção 3.2. O gráfico das funções de probabilidade com cada uma das configurações são mostradas nas Figs. 2.19b, 2.19c e 2.19d.

Tabela 2.2: Exemplos de configurações para os parâmetros do modelo probabilístico da medição de alcance

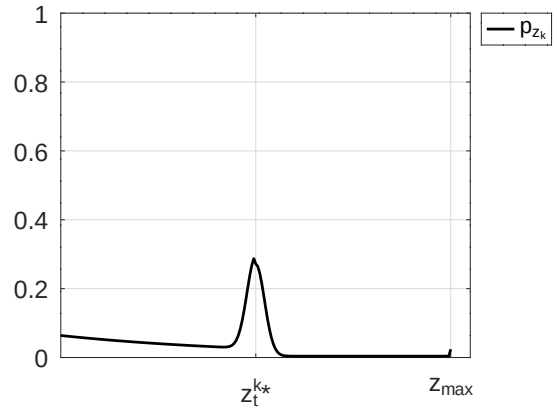
Configuração	σ_{hit}^2	λ_{short}	z_1	z_2	z_3	z_4
1. Ambiente com muitos outliers	0,2	0,1	0,3	0,6	0,02	0,08
2. Mapa com baixa fidelidade	0,8	0,1	0,45	0,05	0,05	0,45
3. Configuração típica do AMCL (laser)	0,2	0,1	0,95	0,1	0,05	0,05

Finalmente, a probabilidade de uma nuvem de pontos do sensor z_t representar a percepção correta é dada pela multiplicação das probabilidades individuais de cada medição:

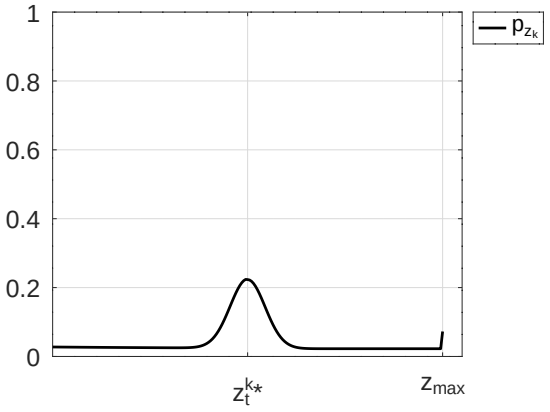
$$\mathrm{p}(z_t|\mathbf{p}_t) = \prod_k \mathrm{p}(z_t^k|\mathbf{p}_t) \tag{2.38}$$



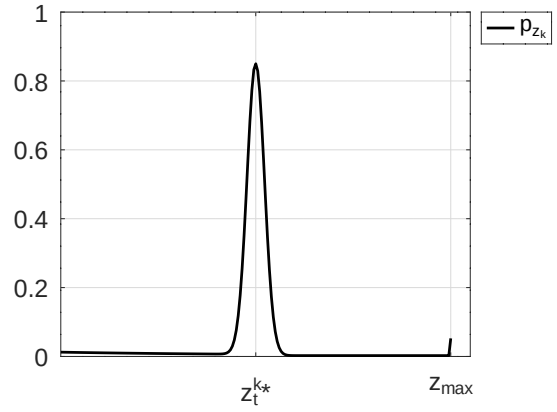
(a) FDP das Eqs. (2.33), (2.34), (2.35) e (2.36). Parâmetros utilizados: $\sigma^2_{hit} = 0,8$, $\lambda_{\text{short}} = 0,1$



(b) Configuração 1



(c) Configuração 2



(d) Configuração 3

Figura 2.19: Gráfico do modelo probabilístico da medição de alcance com diferentes configurações (conforme Tabela 2.2)

Campo de probabilidade

Em princípio, o modelo probabilístico da medição de alcance se baseia na determinação de quais obstáculos são visíveis a partir da pose do robô. Isso implica em realizar o *ray casting* para cada direção k a partir do sensor, que podem chegar às centenas. Para evitar esta operação de alto custo computacional, diferentes autores [59, 60] apresentaram a estratégia de Campo de Probabilidade (*Likelihood Field* – LF)[8].

O *Likelihood Field* (LF) é definido como uma função $l(z_t^k, m)$ que expressa a probabilidade de detecção de um obstáculo:

$$l(z_t^k, m) = e^{-||z_t^k - \text{dist}(z_t^k, m)||^2} \tag{2.39}$$

onde $\text{dist}(z_t^k, m)$ é uma função que retorna a distância euclidiana de z_t^k até a célula

[8]não confundir com Campo Potencial (*Potential Field*), abordagem utilizada em estratégias de navegação.

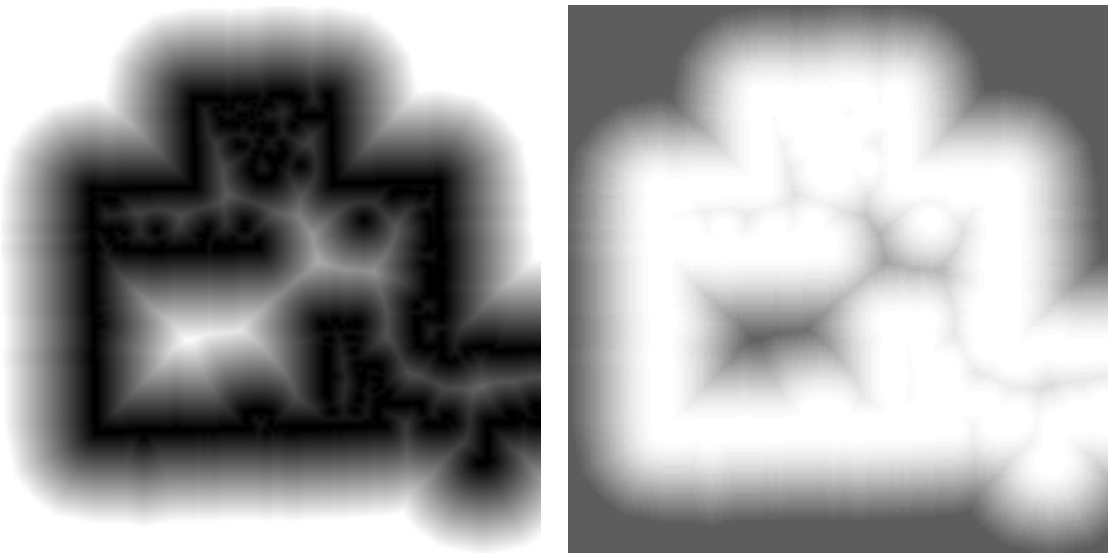
ocupada de $m_{M\times N}$ mais próxima (deve-se recordar que, conforme dito na seção 2.4, o mapa m é representado por uma matriz de células de tamanho $M \times N$). Dado que todos os valores de $\text{dist}(\cdot, m)$ dependem apenas de m que foi previamente construído, pode-se pré-computar uma nova matriz $m^{\text{dist}}_{M\times N}$ onde cada célula contém a distância até a célula ocupada mais próxima.

A aplicação da função $\text{dist}(\cdot, m)$ em todas as células de m, que produz m^{dist}, é chamada de Transformada de Distância Euclideana (TDE). A Fig. 2.20a mostra a TDE obtida a partir da decomposição binária do mapa da Fig. 2.14, enquanto a Fig. 2.20b representa o campo de probabilidade m^{LF}. Portanto, o cálculo de $l(z^k_t, m)$ se reduz à leitura de uma célula de m^{LF}.

Finalmente, a probabilidade de uma nuvem de pontos do sensor z_t representar a percepção correta é dada pela multiplicação dos campos de probabilidade de cada medição:

$$\text{p}(z_t|\mathbf{p}_t) = \prod_k \eta \; l(z^k_t, m) \tag{2.40}$$

onde η é o fator de normalização [60], a fim de manter $\text{p}(z_t|\mathbf{p}_t) < 1$.



(a) Transformada de distância euclidiana (TDE)

(b) Campo de probabilidade (LF)

Figura 2.20: Transformadas do mapa do LaR: (a) TDE e (b) LF. Pixels mais escuros representam valores menores.

Map matching

Os dois modelos de percepção discutidos anteriormente se baseiam na projeção da nuvem de pontos do laser sobre o mapa. Por outro lado, a estratégia de *map matching*, conforme apresentado a seguir, se baseia na correlação entre o mapa global

e um mapa local construído a partir da nuvem de pontos. Esta abordagem permite que não apenas a posição dos obstáculos seja utilizado como informação para a percepção, mas também o espaço livre entre o sensor e o obstáculo.

Conforme visto na seção 2.4, utilizando *ray casting* é possível determinar a área livre à frente do sensor até uma fronteira delimitada por um obstáculo. Da mesma forma, um mapa local (nas proximidades do robô) com células livres, ocupadas e desconhecidas pode ser inferido a partir de tais medições. A Fig. 2.21 mostra diferentes mapas locais construídos utilizando medições realizadas por um arranjo de sonares. Deste modo, pode-se comparar cada mapa local com algum trecho do mapa global, de modo que a similaridade entre eles sugere uma afinidade entre a pose estimada do sensor e a pose correta. A probabilidade da percepção estar correta, portanto, é dada por uma função de correlação entre o mapa global m e o mapa local m_{local} de dimensões $U \times V$ [3, p. 175]:

$$p(m_{local}|m) = \max \left\{ \frac{\sum_{x,y}(m_{x,y} - \bar{m})(m_{x,y,local} - \bar{m})}{\sqrt{\sum_{x,y}(m_{x,y} - \bar{m}\bar{m})^2 \sum_{x,y}(m_{x,y,local} - \bar{m})^2}}, 0 \right\} , \qquad (2.41)$$

onde $\bar{m}$ é o valor médio das células de um mapa intermediário calculado sobre a projeção de m_{local} em m:

$$\bar{m} = \frac{1}{2UV} \sum_{x,y} (m_{x,y} + m_{x,y,local}) \ . \qquad (2.42)$$

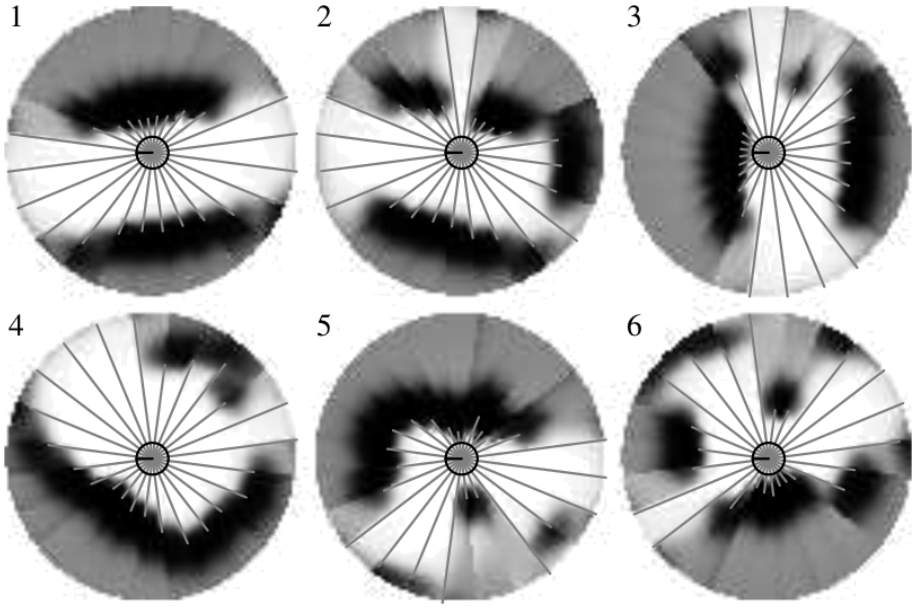


Figura 2.21: Mapas locais construídos a partir de um arranjo de sonares. 1. corredor, 2. corredor com porta aberta. 3. corredor com uma pessoa próxima, 4. canto de uma sala, 5. canto de uma sala com obstáculo, 6. vários obstáculos. Figura extraída de Thrun (1993) [61]

Capítulo 3

Algoritmos de Localização: Estado da Arte

Vários métodos de localização para robôs móveis têm sido propostos nas últimas décadas, mas a abordagem tradicional dos algoritmos consiste em estimar a densidade de probabilidade ao longo do espaço de possíveis localizações baseando-se nos modelos de movimento e de percepção. Portanto, tratam-se de algoritmos fundamentalmente probabilísticos. Sob a propriedade de Markov [3], a confiança sobre um estado esperado (ou a pose, no contexto deste trabalho) $bel(\mathbf{p}_t)$ depende do estado anterior $\mathbf{p}_{t-1}$ e das informações mais recentes sobre o movimento e a percepção (u_t, z_t), de modo que é normalmente descrito pelo *framework* do Filtro de Bayes:

$$bel(\mathbf{p}_t) = \eta \underbrace{p(z_t|\mathbf{p}_t)}_{g(\cdot)} \int \underbrace{p(\mathbf{p}_t|u_t, \mathbf{p}_{t-1})}_{f(\cdot)} bel(\mathbf{p}_{t-1}) d\mathbf{p}_{t-1} \ , \tag{3.1}$$

onde η é um parâmetro de normalização, $g(\cdot)$ é a probabilidade de ocorrer z_t, dado $\mathbf{p}_t$, e $f(\cdot)$ é a probabilidade de ocorrer $\mathbf{p}_t$, dados u_t e $\mathbf{p}_{t-1}$. A função $bel(\mathbf{p}_t)$ modela uma Função Densidade de Probabilidade (FDP), e a incerteza sobre a pose $\mathbf{p}_t$ pode ser inferida a partir das covariâncias calculadas.

Métodos de localização como o Filtro de Kalman [62] e o Filtro de Partículas [63] são implementações do Filtro de Bayes, embora eles assumam diferentes premissas em relação ao problema. O Filtro de Kalman implica que $f(\cdot)$ e $g(\cdot)$ sejam definidas por funções Gaussianas, $p(\cdot) \sim \mathcal{N}(\mu, \Sigma)$, onde μ é o estado (a média) e Σ é a matriz de covariância das variáveis aleatórias do estado. Conforme será mostrado na seção 3.1, esta premissa reduz o problema de estimação recursiva do estado a uma média ponderada de dois valores cujos pesos são inversamente proporcionais às respectivas covariâncias. Nos casos onde não é conveniente considerar que os modelos de percepção e de transição de estado não se aproximam à uma sistema linear, o *Extended Kalman Filter* (EKF) [44] realiza a linearização local para aproximar funções não-lineares.

Em outra abordagem, o Filtro de Partículas utiliza um conjunto de M hipóteses com respectivas importâncias para representar distribuições de probabilidade arbitrárias. Inicialmente, o Filtro de Partículas realiza a amostragem de hipóteses a partir da função de distribuição de probabilidade que modela a transição de estado, $f(.)$. Portanto, a expectativa $p(\mathbf{p}_x)$, $0 \leq x < M$, de que uma hipótese $\mathbf{p}_x$ seja amostrada é proporcional a $p(\mathbf{p}_x|u_t, \mathbf{p}_{t-1})$. O AMCL é uma instância do Filtro de Partículas, porém, ao invés de realizar a amostragem por $f(.)$, este método utiliza uma função refinada que modela o erro aleatório individualmente para cada variável do estado.

Este capítulo discorre sobre alguns algoritmos de localização que são pertinentes a este trabalho. Os algoritmos Filtro de Kalman e AMCL estão descritos nas seções 3.1 e 3.2, respectivamente. Já o PM, que se baseia em uma abordagem de *scan-to-map* matching, é apresentado na seção 3.3.

3.1 Filtro de Kalman

O Filtro de Kalman [64] é um algoritmo de estimação de uma variável não observável $\mathbf{x}_t \in R^n$ a partir de variáveis observáveis cujos valores são afetados por $\mathbf{x}_t$. Esta técnica utiliza apenas a média e variância (ou covariância para $n > 0$) de uma distribuição normal para estimar recursivamente o estado. Portanto, o Filtro de Kalman é uma implementação do Filtro de Bayes. Dado que o Filtro de Kalman assume que a função de transição de estado é aproximada por um sistema linear e que os ruídos envolvidos são gaussianos, o estado é definido por uma função gaussiana com média μ e uma variância σ^2 ou matriz de covariância $\mathbf{\Sigma}$, de modo que:

$$f(x_t) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x_t-\mu}{\sigma}\right)^2} \text{ , para o estado } x_t \text{ escalar, ou} \tag{3.2}$$

$$f(\mathbf{x}_t) = \frac{1}{(2\pi)^n \sqrt{\det(\mathbf{\Sigma})}} e^{-\frac{1}{2}(\mathbf{x}_t-\boldsymbol{\mu})^T \mathbf{\Sigma}^{-1}(\mathbf{x}_t-\boldsymbol{\mu})} \text{ , para o estado } \mathbf{x}_t \in \mathbb{R}^{1\times n} \tag{3.3}$$

O Filtro de Kalman implementa o cômputo da probabilidade para estados contínuos, e portanto não é diretamente aplicável a estados discretos. Neste caso, ao invés de utilizar a função contínua da Eq. (3.3), o Filtro de Kalman utiliza os parâmetros desta gaussiana, ou seja, a média μ e a covariância Σ, conforme descrito a seguir. Em aplicações onde o estado é necessariamente discreto, o Filtro de Bayes discreto, também chamado de Filtro de Histograma, e aplicável [3, p. 86].

A Fig. 3.1 ilustra o Filtro de Kalman realizando a estimação recursiva do estado de uma variável escalar x_t baseando-se em alguma função de propagação de estado (de x_{t-1} para x_t e em z_t (uma variável indireta afetada por x_t). Na prática, este seria

um exemplo da localização de um robô sobre trilhos em um espaço 1D utilizando a odometria (propagação do estado) e medições de sensores (z_t).

A Fig. 3.1a ilustra a posição inicial do robô x_0^+, também chamado de estado *a priori*. Esta posição é descrita por uma gaussiana com média $\mu = 3,45$ e variância $\sigma^2 = 2,25$. O gráfico mostra a probabilidade de x_t no eixo y (*Belief*). A base larga da função mostra que há uma elevada incerteza sobre o estado esperado. Em seguida, a Fig. 3.1b mostra a a probabilidade em vermelho a partir da medição z_0 – obtida por algum modelo de percepção, como o da Eq. (2.38). Finalmente, a Fig. 3.1c mostra a aplicação do Filtro de Kalman para estimar o estado x_0 a partir de x_0^+ e z_0.

O estado x_0 aparece como uma média ponderada das médias de x_0^+ e z_0, onde os pesos são as respectivas variâncias. Por outro lado, a variância de x_0 é obtida a partir das variâncias de x_0^+ e z_0, de modo que a incerteza sobre o novo estado é sempre menor do que as variâncias utilizadas – ou seja, novas informações, frequentemente chamadas de novidades, sempre contribuem para reduzir a incerteza sobre o estado. Este é o princípio do Filtro de Kalman, que se repete nas próximas estimações.

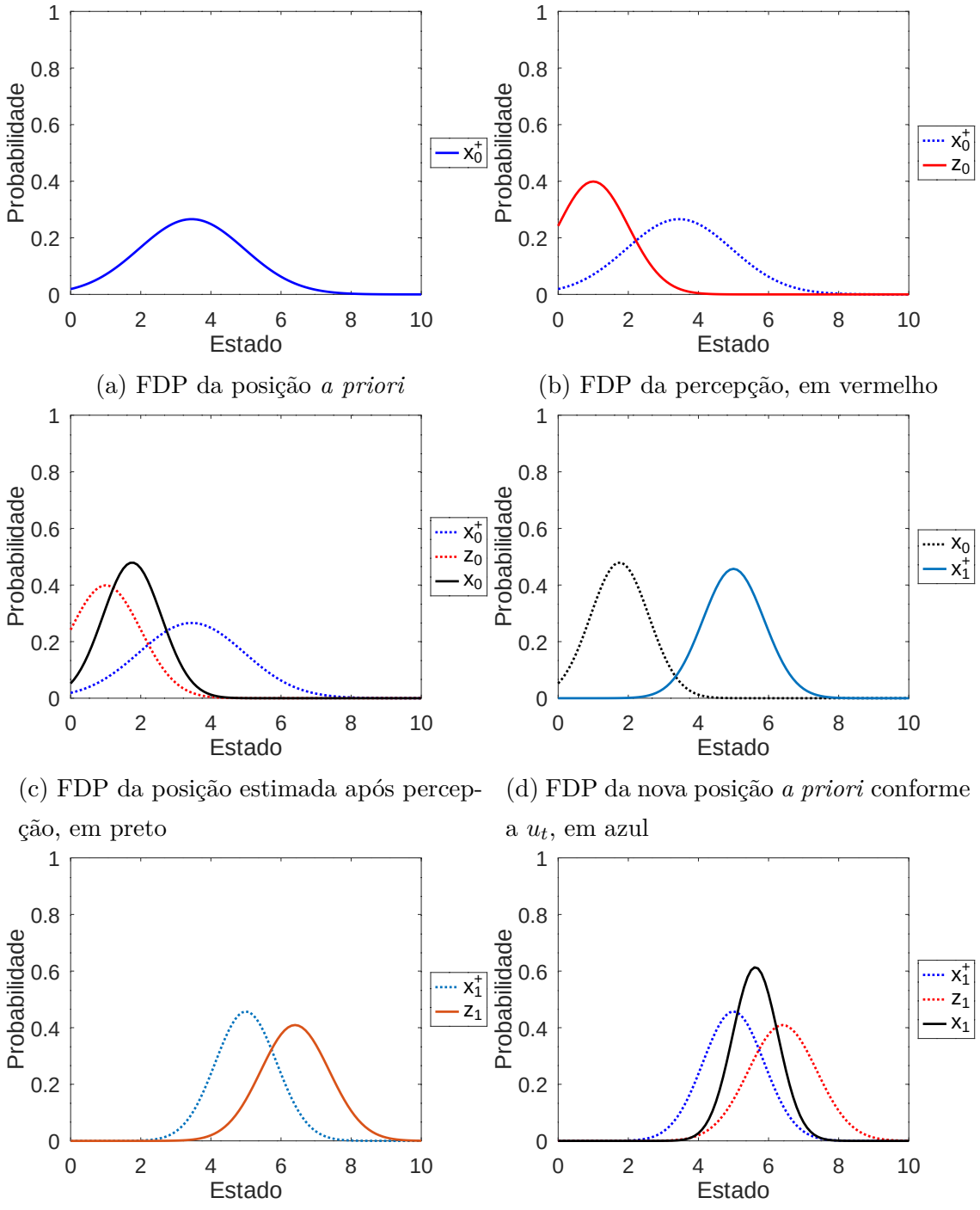
Em seguida, a propagação do estado x_t acontece (conforme a odometria, por exemplo), de modo a deslocar o gráfico para uma nova posição (Fig. 3.1d) e, como normalmente acontece, a incerteza sobre a posição cresce (ou seja, o valor de σ^2 aumenta) devido a ruídos de sistema. No caso da localização do robô, o novo estado esperado após a inclusão da odometria seria descrito por $\mu = 5$ e $\sigma^2 = 0,76$. As Figs. 3.1e e 3.1f continuam com a estimação recursiva do estado a partir da posição *a priori* x_1^+ e da percepção z_1, resultando em um novo estado x_1.

Para variáveis vetoriais $\mathbf{x}_t \in \mathbb{R}^n$, $n > 1$, como no caso da pose de um robô em 2D $\mathbf{p}_t = (x, y, \theta)$ o Filtro de Kalman utiliza a distribuição normal multivariada – Eq. (3.3). Neste caso, em localização de um robô terrestre, o estado dado por $\mathbf{p}_t$ é estimado em duas etapas: previsão da pose *a priori* e correção utilizando a percepção. Apesar de seu profundo embasamento matemático [3, pp. 45-53], serão mostradas a seguir apenas as equações de Kalman que constituem o algoritmo. Na etapa de previsão, um modelo de movimento (Seção 2.3) é usado, cuja equação é dada por:

$$\mathbf{p}_t^+ = \mathbf{F}_t \mathbf{p}_{t-1} + \mathbf{u}_t \ , \tag{3.4}$$

onde $\mathbf{u}_t$ corresponde ao segundo termo das Eqs. (2.24) e (2.29) e o vetor de coeficientes $\mathbf{F}_t \in \mathbb{R}^{3\times3}$ descreve o comportamento dinâmico da pose. Por representarem modelos simplificados do comportamento dinâmico do robô, para as equações de odometria (2.24) e (2.29), tem-se $\mathbf{F} = \mathbf{I}^{3\times3}$. A matriz de covariância *a priori* $\mathbf{\Sigma}_t^+ \in \mathbb{R}^{3\times3}$ é dada por:

$$\mathbf{\Sigma}_t^+ = \mathbf{F}_t \mathbf{\Sigma}_{t-1} \mathbf{F}_t^T + \mathbf{V} \ , \tag{3.5}$$



(a) FDP da posição *a priori*

(b) FDP da percepção, em vermelho

(c) FDP da posição estimada após percepção, em preto

(d) FDP da nova posição *a priori* conforme a u_t, em azul

(e) FDP da nova percepção, em vermelho

(f) FDP da posição estimada após percepção, em preto

Figura 3.1: Ilustração da estimação da posição com o Filtro de Kalman.

onde $\mathbf{V} \in \mathbb{R}^{3\times3}$ representa a incerteza sobre o modelo de movimento na forma de matriz de covariância. Observa-se, portanto que a incerteza na previsão cresce proporcionalmente à incerteza do modelo de movimento, como ilustra a Fig. 3.1d.

Na etapa de correção, a pose e a matriz de covariância são atualizadas com base nas informações obtidas pela percepção:

$$\mathbf{p}_t = \mathbf{p}_t^+ + \mathbf{K}(\mathbf{z}_t - \mathbf{H}\mathbf{p}_t^+) \tag{3.6}$$

onde $\mathbf{z}_t$ é a medição dada pelo sensor e $\mathbf{H}\mathbf{p}_t^+$, $\mathbf{H} \in \mathbb{R}^{3\times3}$, é o estado esperado obtido através de modelos de percepção como os das Eqs. (2.38), (2.40) e (2.41), ou seja, $\mathbf{H}\mathbf{p}_t^+$ modela a medição esperada a partir da pose *a priori* $\mathbf{p}_t^+$. Portanto, o termo $(\mathbf{z}_t - \mathbf{H}\mathbf{p}_t^+)$ caracteriza a correção da localização utilizando informações dos sensores. O termo $\mathbf{K}$ refere-se ao ganho de Kalman que minimiza o erro quadrático médio estimado [65, p. 146], dado por:

$$\mathbf{K} = \mathbf{\Sigma}_t^+\mathbf{H}^T(\mathbf{H}\mathbf{\Sigma}_t^+\mathbf{H}^T + \mathbf{W})^{-1} \tag{3.7}$$

onde $\mathbf{W} \in \mathbb{R}^{3\times3}$ define a matriz de covariância estimada referente aos sensores. Por fim, calcula-se a incerteza sobre a pose estimada $\mathbf{p}_t$:

$$\mathbf{\Sigma}_t = \mathbf{\Sigma}_t^+ - \mathbf{K}\mathbf{H}\mathbf{\Sigma}_t^+ \tag{3.8}$$

O ganho de Kalman tem um papel fundamental no algoritmo, uma vez que ele controla a influência da estimação da odometria e dos sensores na pose final estimada baseando-se na incerteza sobre cada um. Por exemplo, quando a incerteza sobre os sensores é muito baixa, a os elementos da matriz de covariância $\mathbf{W}$ se aproximam de 0 e o valor do ganho tende a se aproximar de $\mathbf{K} = \mathbf{H}^{-1}$. Por consequência, a Eq. (3.6) é reduzida a:

$$\begin{aligned} \mathbf{p}_t &\simeq \mathbf{p}_t^+ + \mathbf{I}\mathbf{H}^{-1}(\mathbf{z}_t - \mathbf{H}\mathbf{p}_t^+) \\ &\simeq \mathbf{p}_t^+ + \mathbf{H}^{-1}\mathbf{z}_t - \mathbf{I}\mathbf{p}_t^+ \\ &\simeq \mathbf{H}^{-1}\mathbf{z}_t \ , \end{aligned} \tag{3.9}$$

ou seja, a pose estimada é predominantemente baseada na percepção, visto que a qualidade da informação é muito alta. De forma análoga, quando a incerteza sobre a pose estimada pela odometria é muito baixa, o valor do ganho se aproxima de $\mathbf{K} = 0$ e por consequência a pose estimada se reduz a:

$$\begin{aligned} \mathbf{p}_t &\simeq \mathbf{p}_t^+ + 0(\mathbf{z}_t - \mathbf{H}\mathbf{p}_t^+) \\ &\simeq \mathbf{p}_t^+ \ , \end{aligned} \tag{3.10}$$

ou seja, a informação dos sensores é praticamente desprezada devido à alta qualidade da odometria. Embora estes sejam exemplos virtualmente inconcebíveis, servem para ilustrar a influência do ganho de Kalman na estimação do estado.

É importante destacar que o algoritmo de Kalman, apesar de robusto, depende de uma pose anterior $\mathbf{p}_{t-1}$, e portanto se aplica apenas em *Pose Tracking*. Por outro lado, existem extensões do Filtro de Kalman para acomodar situações em que múltiplas hipóteses de poses são consideradas, como a abordagem *Multi-Hypothesis Tracking* (MHT) descrita por Thrun *et al.* [3, pp. 218-220]. Além disso, as equações de Kalman descritas anteriormente são apropriadas para aplicações cujos modelos são descritos por funções lineares. Por isso, o Filtro de Kalman Estendido (*Extended Kalman Filter* – EKF) foi introduzido como uma adaptação do Filtro de Kalman para lidar com sistemas não lineares, tendo sido descrito por diferentes autores [1, 3, 55]. Neste algoritmo, os modelos não lineares de transição de estado e de percepção são linearizados localmente utilizando matrizes jacobianas [66].

3.2 Localização de Monte Carlo

O método de localização descrito anteriormente consiste basicamente na propagação de uma distribuição normal ao longo do espaço caracterizado por um sistema de coordenadas global, ou seja, o mapa. Porém, na maioria dos casos, a distribuição de probabilidade da pose de um robô não pode ser aproximada por uma simples gaussiana, visto que na maioria das vezes esta distribuição é multimodal, ou seja, pode apresentar mais de um pico de probabilidade. Portanto, o método de Kalman não é capaz de representar ambiguidades, ou seja, situações onde as informações disponíveis sugerem que o robô pode estar em mais de uma posição no mapa (tais situações são citadas no Capítulo 4). Ao invés de descrever uma FDP gaussiana, o método apresentado nesta seção representa a própria distribuição de probabilidade através de um conjunto de partículas discretas amostradas desta distribuição.

A Fig. 3.2 mostra uma ilustração frequentemente utilizada na literatura para demonstração de métodos de localização baseados em filtro de partículas, como é o caso da Localização de Monte Carlo (*Monte Carlo Localization* – MCL). O cenário apresentado compreende um robô que precisa se localizar em um espaço 1D utilizando apenas a odometria e um sensor capaz detectar a presença de uma porta. O ambiente contém três portas idênticas e ordenadas, 1, 2 e 3, da esquerda para a direita. As distâncias entre as portas 1-2 e 2-3 são de aproximadamente 1 m e 4 m, respectivamente.

Inicialmente, o robô se encontra em frente à primeira porta (Fig. 3.2a). Visto que a localização não é conhecida, um conjunto de hipóteses aleatórias distribuídas ao longo do espaço 1D é criado, como mostra a Fig. 3.2b com $N = 100$ partículas representadas por prismas azuis. Neste momento, todas as partículas possuem a mesma importância inicial $w_{ini} = \frac{1}{N}$. Portanto, a população S_t constitui o conjunto

de N partículas com suas respectivas posições x_t e importâncias w_t:

$$S_t = \{x_t^1, w_t^1, x_t^2, w_t^2, ..., x_t^N, w_t^N\} \ , \tag{3.11}$$

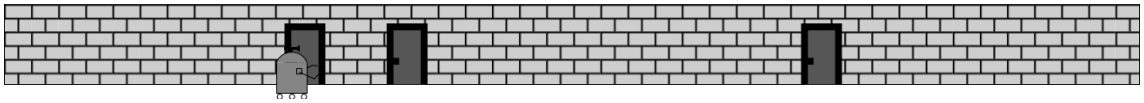
de modo que $w_t^1 + w_t^2 + ... + w_t^N = 1$.

Em seguida, o sensor informa que detectou uma porta. Para fins de simplificação, considera-se que a posição do sensor coincide com a do robô. O modelo de percepção (Fig. 3.2c) caracterizado por uma função contínua $p(z_0)$ mostra que há três regiões com alta probabilidade de localização do robô, enquanto as demais regiões apresentam baixa probabilidade. Nota-se que o modelo inclui uma variância (incerteza) em $p(z_0)$ nas regiões de alta probabilidade e uma probabilidade residual no restante do espaço. Isso se deve a possíveis ruídos e falhas de detecção inerentes ao sensor, que normalmente são configurados conforme a aplicação.

A função $p(z_0)$ é utilizada para atribuir importâncias w_0 a cada partícula, como mostra a Fig. 3.2d. A altura de cada importância caracteriza a probabilidade da partícula ser amostrada pelo algoritmo na próxima geração da população. A amostragem é realizada através do método da roda de roleta, também chamado de *Importance Sampling*. Este método, inspirado no item presente nas mesas de cassino, realiza a seleção aleatória de uma das N casas da roleta, de modo que cada casa possui probabilidade $\frac{1}{N}$. No caso do *Monte Carlo Localization* (MCL), a importância define a quantidade de casas da roleta que a partícula deverá ocupar.

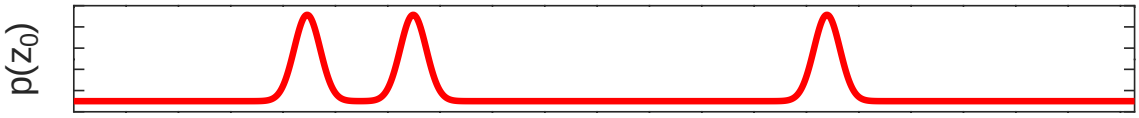
Nesta abordagem, no instante $t = 0$, cada k-ésima partícula, $0 < k < N$, recebe uma importância cumulativa $w_C^k = w_C^{k-1} + w^k$, onde $w_C^{k=0} = w^{k=0}$. Então, um número aleatório $r \in \mathbb{R}$, $0 \leq r \leq w_C^N$, é utilizado para selecionar a partícula n que irá compor a próxima geração, de modo que $w_C^n > r > w_C^{n+1}$. Este processo se repete N vezes, formando a nova geração de N indivíduos. É importante destacar que indivíduos com alta importância têm maior chance de serem selecionados, porém não serão réplicas idênticas devido a etapa de reamostragem a partir do modelo da odometria, conforme descrito a seguir.

As partículas recém-amostradas são deslocadas conforme algum modelo de odometria, de modo a acomodar a mudança relativa na posição do robô. Entretanto, devido à natureza estocástica do modelo, uma perturbação aleatória e limitada é incluída no deslocamento de cada partícula, de forma a caracterizar a incerteza do movimento. Uma vez que os modelos descritos nas Eqs. (2.24) e (2.29) não incorporam alguma incerteza, Thrun *et al* [67] propõem um modelo de amostragem para odometria adequado à este procedimento, onde recebe uma hipótese x_{t-1} e a informação de odometria u_t e retorna uma pose *a priori* x'_t aleatoriamente distribuída conforme uma distribuição de probabilidade $p(x'_t|x_{t-1}, u_t)$. Internamente, este modelo inclui algumas das equações citadas e adiciona um ruído aleatório, geralmente gaussiano, à nova pose obtida.

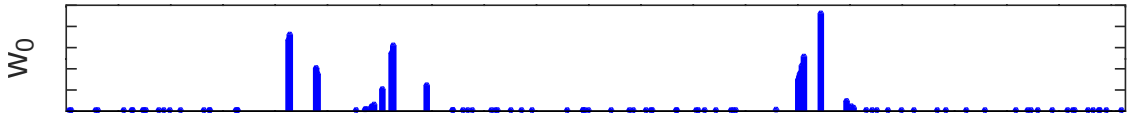


(a) Situação inicial do robô, detectando uma porta

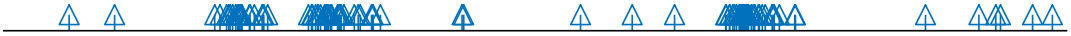
(b) Partículas que compõem a população inicial



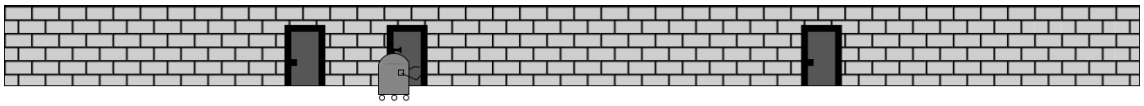
(c) Distribuição de probabilidade da detecção de uma porta



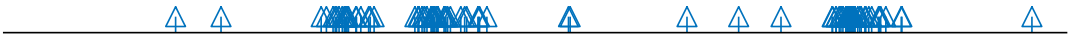
(d) Importâncias das partículas obtidas a partir de $p(z_0)$



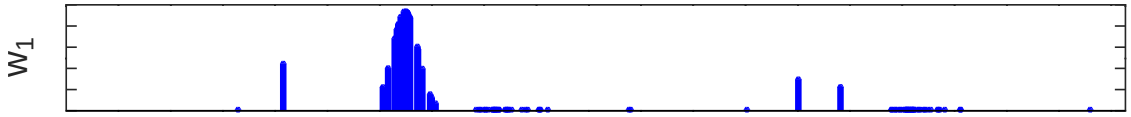
(e) População após amostragem por importância e reamostragem



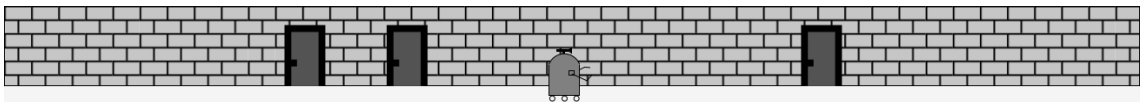
(f) Situação seguinte do robô, detectando uma porta



(g) Propagação das partículas de conforme a odometria

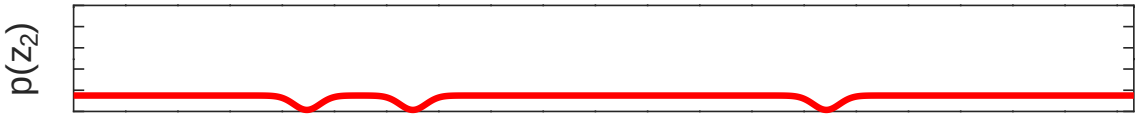


(h) Importâncias das partículas obtidas a partir de $p(z_1)$ – semelhante a $p(z_0)$



(i) Situação seguinte do robô, detecta ausência de uma porta

(j) População após amostragem, reamostragem e propagação conforme a odometria



(k) Distribuição de probabilidade de ausência da detecção de uma porta

Figura 3.2: Demonstração da Localização de Monte Carlo em um espaço 1D com 100 partículas; imagens (a) (f) e (i) adaptadas de Thrun *et al* [3]

Sendo assim, após mover-se aproximadamente 1 m, uma nova população é formada no instante $t = 1$ conforme mostra a Fig. 3.2g. Em seguida, as importâncias são atribuídas as partículas conforme a percepção e, como o robô detecta novamente a presença de uma porta, a distribuição de probabilidade da medição z_1 é semelhante à Fig. 3.2c. Observa-se, então, importâncias maiores em pontos próximos à porta e a concentração de partículas em torno da segunda porta.

O robô continua a mover-se e, em $t = 2$, após o novo *Importance Sampling*, desloca-se por aproximadamente $2,5$ m, e as partículas formam a distribuição apresentada na Fig. 3.2j. Neste ponto, o robô detecta a ausência de uma porta, e portanto a distribuição de probabilidade da percepção apresenta a forma da Fig. 3.2k. Portanto, $p(z_2)$ definirá as novas importâncias das partículas no instante $t = 2$, de modo que partículas próximas das portas terão maior chance de serem excluídas. Finalmente, nota-se que, apesar da população inicial apresentar baixa densidade de partículas próximas da posição correta (Fig. 3.2b), o algoritmo foi capaz de concentrar hipóteses próximas ao robô após algumas iterações (Fig. 3.2j).

3.2.1 População com tamanho adaptativo

Pela descrição do método de Monte Carlo, pode-se inferir que o esforço computacional está diretamente relacionado ao tamanho da população, visto que os modelos de movimento e de percepção são usados para atualizar cada hipótese x^i individualmente. Desta forma, quando o método converge para uma região (Fig. 3.2j), muitas partículas se concentram em torno de uma pose, tornando uma parte delas redundante. Por isso, Fox [68] introduziu o KLD-Sampling, sugerindo adaptar o número de partículas de forma proporcional à incerteza da pose. O autor propôs que, ao invés de gerar N novas hipóteses a cada iteração, o número de partículas geradas durante o *Importance Sampling* deve ser proporcional à divergência entre as distribuições de probabilidade das etapas de predição e correção. Na prática, a quantidade de indivíduos se torna proporcional à dispersão das partículas e à qualidade destas, conforme detalhado a seguir.

Em cada iteração, o método de KLD-Sampling determina a quantidade mínima de partículas para tornar menor que ϵ a divergência entre a distribuição real e a distribuição representada pelas partículas da população. Esta divergência é conhecida como *KullbackLeibler Divergence* (KLD), ou KL-distance. Dado que não é possível obter a distribuição real, o método estima com probabilidade $p \leq (1 - \delta)$ que KL-distance $< \epsilon$. Essa abordagem é empregada no AMCL, método de localização global utilizado neste trabalho.

Desta forma, o autor apresenta o cálculo da quantidade mínima de partículas

M_x iterativamente através da seguinte aproximação:

$$M_x = \frac{k-1}{2\epsilon}\left(1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}}z_{1-\delta}\right)^3 , \tag{3.12}$$

onde k é a quantidade de barras do histograma não-nulas até a iteração atual e $z_{1-\delta}$ é um parâmetro estatístico obtido da tabela de distribuição padrão [68].

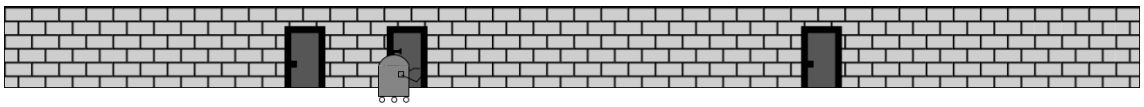
As Figs. 3.3, 3.4 e 3.5 ilustram três diferentes condições onde o método de KLD-Sampling modifica o tamanho da população em função do KL-distance. Em todas as ilustrações foi estabelecida a população inicial com 100 indivíduos e os parâmetros $\epsilon = 0,05$ e $z_{1-\delta} = 0,99$.

Na primeira condição (3.3), o método de localização está sendo inicializado e portanto as partículas estão distribuídas com probabilidade uniforme ao longo de todo o espaço de busca (3.3a). A Fig. 3.3b mostra as importâncias das hipóteses, evidenciando que há três regiões com maior probabilidade, dadas as informações dos sensores. Ainda na Fig. 3.3, as ilustrações seguintes mostram o progresso da nova população sendo criada dentro de uma iteração, mostrando a posição das partículas e os histogramas para os instantes onde a população possui 50, 100, 200 e 325 indivíduos, nesta ordem. Os histogramas apresentam 50 barras e, como há uma grande quantidade de barras não-nulas nos primeiros instantes, o método determinou, iterativamente, o tamanho mínimo $M_x = 325$.

A Fig. 3.4 mostra uma segunda condição onde o robô está na mesma posição apresentada na Fig. 3.3a, mas suas hipóteses estão distribuídas através de uma gaussiana em torno da segunda porta da esquerda para a direita. As importâncias destas hipóteses são mostradas na Fig. 3.4b. Neste caso, a concentração de partículas amostradas iterativamente (Figs. 3.4c, 3.4e, 3.4g e 3.4i) implica em uma quantidade menor de barras atualizadas no histograma em comparação à condição anterior, de modo que o tamanho da população resultante ($M_x = 125$) também é menor que o da condição anterior.

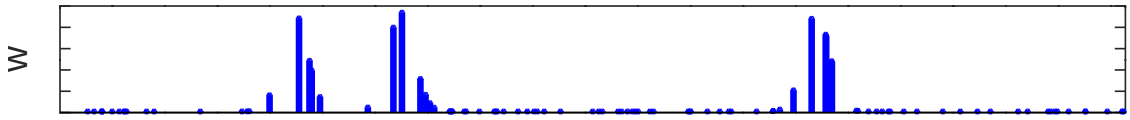
Finalmente, a Fig. 3.5 ilustra uma terceira condição o robô se encontra em frente à parede e seu método de localização apresenta uma distribuição normal de partículas próxima à terceira porta. Nota-se que as importâncias de algumas partículas próximas à média da gaussiana é menor (Fig. 3.5c), visto que o modelo de percepção entende que há menor probabilidade do robô estar em frente a uma porta. Portanto, apesar de apresentar uma distribuição de hipóteses com variância semelhante à segunda condição (Fig. 3.4), a relativa uniformidade das importâncias de maior valor implica em um maior número de barras atualizadas no histograma, e portanto um valor de M_x maior.

O método de KLD-Sampling atualiza o valor de M_x à medida que novas hipóteses são geradas pelo *Importance Sampling*. A Fig. 3.6 mostra a determinação dinâmica

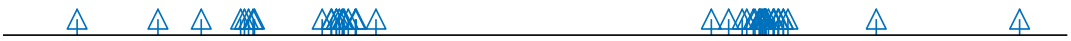


(a) Posição real do robô

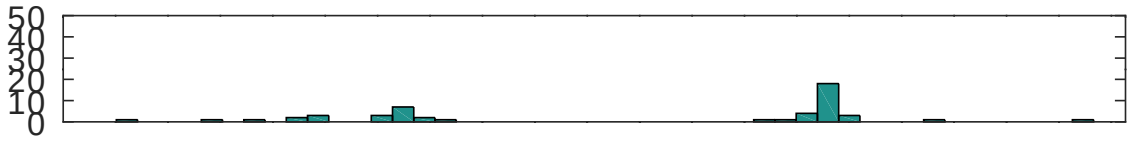
(b) Hipóteses aleatórias representando uma distribuição de probabilidade uniforme



(c) Respectivas importâncias das hipóteses em (b) após percepção

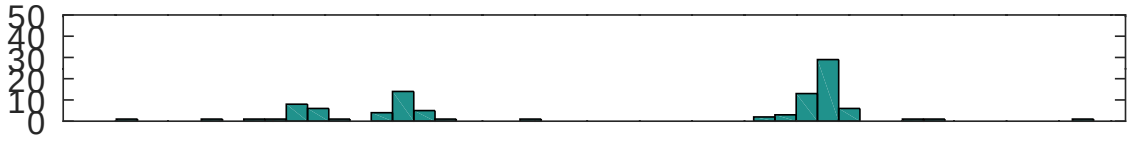


(d) Distribuição da população durante o KLD-samping para $P_{\text{size}} = 50$



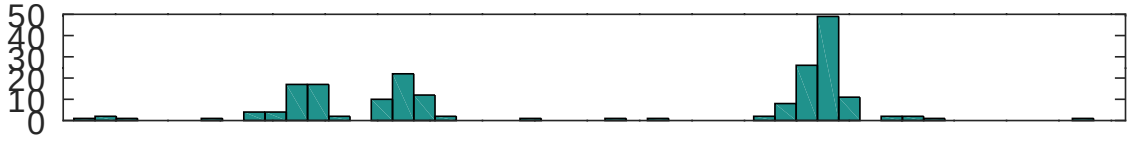
(e) Histograma da distribuição das hipóteses para $P_{\text{size}} = 50$

(f) Distribuição da população durante o KLD-samping para para $P_{\text{size}} = 100$

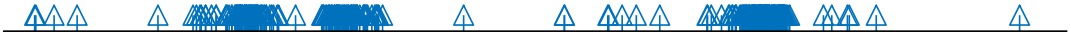


(g) Histograma da distribuição das hipóteses para $P_{\text{size}} = 100$

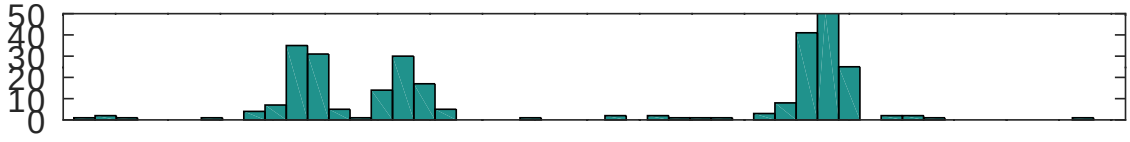
(h) Distribuição da população durante o KLD-samping para para $P_{\text{size}} = 200$



(i) Histograma da distribuição das hipóteses para $P_{\text{size}} = 200$



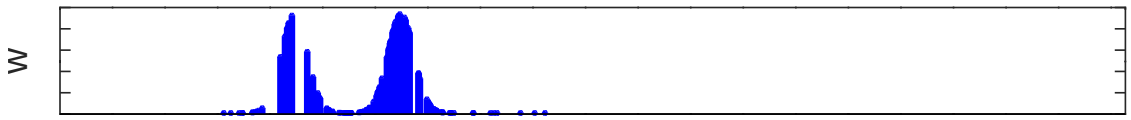
(j) Distribuição da população durante o KLD-samping para para $P_{\text{size}} = 325$



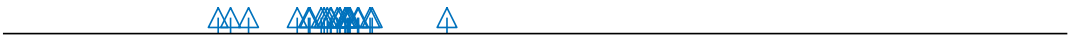
(k) Histograma da distribuição das hipóteses para $P_{\text{size}} = 325$

Figura 3.3: KLD-Sampling em uma iteração da primeira situação descrita no texto; os prismas em azul representam os indivíduos amostrados e os histogramas com 50 barras em verde representam a concentração para cada instante; P_{size} representa o tamanho instantâneo da população

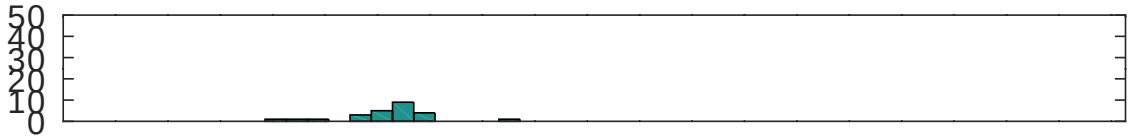
(a) Hipóteses aleatórias representando uma distribuição de probabilidade gaussiana



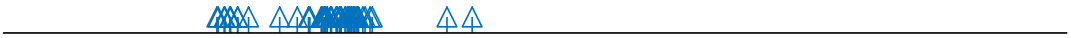
(b) Respectivas importâncias das hipóteses em (a) após percepção



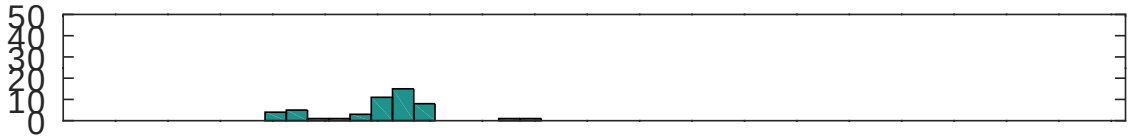
(c) Distribuição da população durante o KLD-samping para para $P_{\text{size}} = 25$



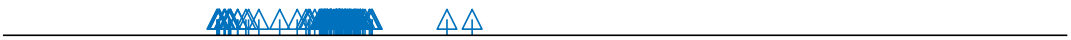
(d) Histograma da distribuição das hipóteses para $P_{\text{size}} = 25$



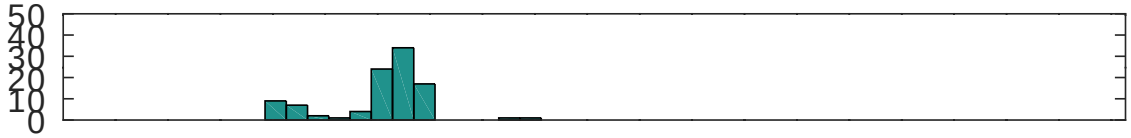
(e) Distribuição da população durante o KLD-samping para para $P_{\text{size}} = 50$



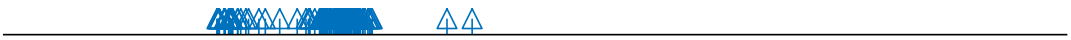
(f) Histograma da distribuição das hipóteses para $P_{\text{size}} = 50$



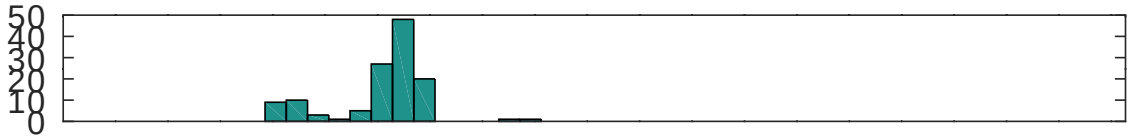
(g) Distribuição da população durante o KLD-samping para para $P_{\text{size}} = 100$



(h) Histograma da distribuição das hipóteses para $P_{\text{size}} = 100$

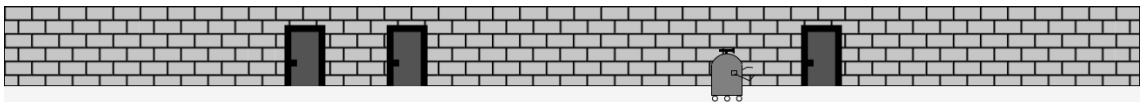


(i) Distribuição da população durante o KLD-samping para para $P_{\text{size}} = 125$

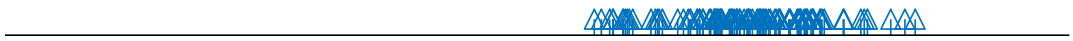


(j) Histograma da distribuição das hipóteses para $P_{\text{size}} = 125$

Figura 3.4: KLD-Sampling em uma iteração da segunda situação descrita no texto; os prismas em azul representam os indivíduos amostrados e os histogramas com 50 barras em verde representam a concentração para cada instante; P_{size} representa o tamanho instantâneo da população; a posição real do robô é mesma da Fig. 3.3a

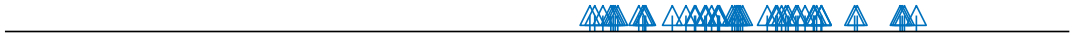


(a) Posição real do robô

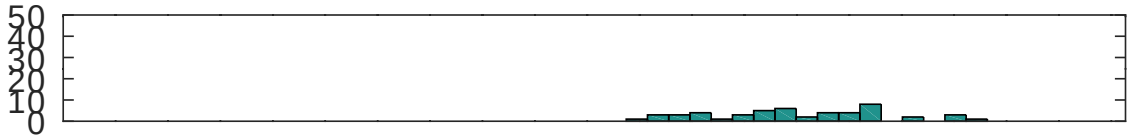


(b) Hipóteses aleatórias representando uma distribuição de probabilidade gaussiana

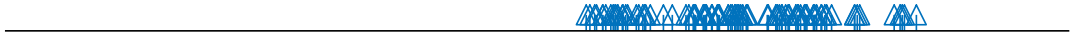
(c) Respectivas importâncias das hipóteses em (b) após percepção



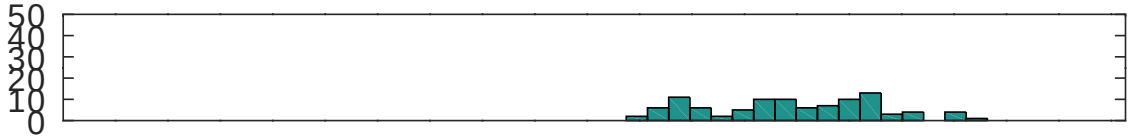
(d) Distribuição da população durante o KLD-samping para para $P_{\text{size}} = 50$



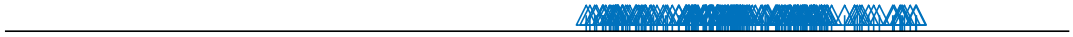
(e) Histograma para $P_{\text{size}} = 50$



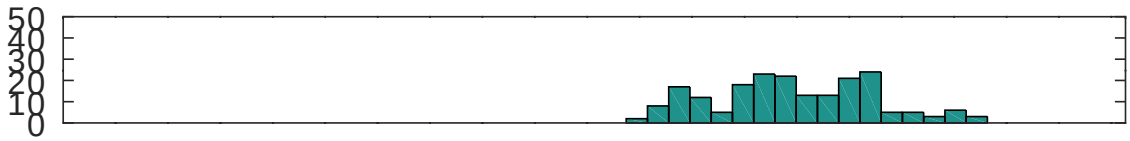
(f) Distribuição da população durante o KLD-samping para para $P_{\text{size}} = 100$



(g) Histograma para $P_{\text{size}} = 100$



(h) Distribuição da população durante o KLD-samping para para $P_{\text{size}} = 200$



(i) Histograma para $P_{\text{size}} = 200$

Figura 3.5: KLD-Sampling em uma iteração da terceira situação descrita no texto; os prismas em azul representam os indivíduos amostrados e os histogramas com 50 barras em verde representam a concentração para cada instante; P_{size} representa o tamanho instantâneo da população

do valor de M_x para as três condições citadas anteriormente, na mesma ordem. Nota-se que o método inicia com $M_x = 0$, portanto é necessário estabelecer um tamanho de população mínimo $P_{\min}$, independentemente do KL-distance, a fim de garantir que haja iterações suficientes para obter um valor de M_x coerente. Além disso, o método AMCL, que também possui o parâmetro de configuração $P_{\min}$, estabelece um tamanho máximo configurável $P_{\max}$, a fim de permitir que o usuário possa estabelecer limites para garantir a exequibilidade computacional.

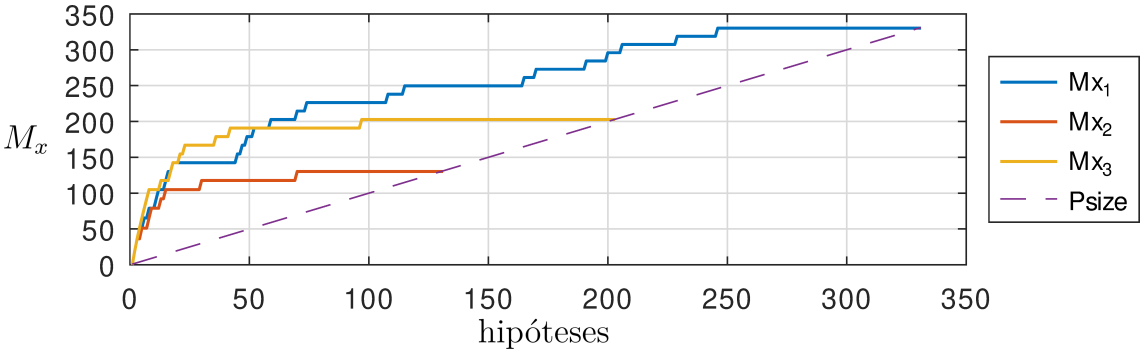


Figura 3.6: Valores de M_x ao longo de uma iteração para as situações descritas anteriormente; a linha tracejada representa o tamanho instantâneo da população

3.3 Perfect Match

Os métodos descritos anteriormente se baseiam em modelos probabilísticos que caracterizam o estado do robô e as informações dos sensores. Esta abordagem permite que o comportamento estocástico do sistema seja representado pelo algoritmo. Por outro lado, existem métodos [20] em que as variáveis aleatórias não são explicitamente modeladas, mas que apresentam desempenho similar (ou superior em alguns casos [16]) ao AMCL e ao Filtro de Kalman, por exemplo, como o algoritmo de localização *Perfect Match* (PM) [11].

O PM é um algoritmo de rastreamento de pose com baixo custo computacional que utiliza uma abordagem de *scan-to-map matching*. Foi introduzido por Lauer, Lange e Riedmiller [11] para localizar um agente de futebol de robôs em um campo de futebol com linhas brancas utilizando uma câmera RGB omnidirecional. Gouveia *et al.* [12] e Sobreira *et al.* [14] aplicaram o PM na localização de robôs autônomos em um mapa de ocupação 2D utilizando sensores do tipo *Range Finder* (aqueles que medem a distância livre em uma direção). M. Pinto *et al.* [13] adaptaram o PM para localização com mapas 3D utilizando sensores a laser 3D. A subseção a seguir descreve a abordagem do PM.

3.3.1 Localização baseada em contornos

Uma parte considerável dos robôs móveis apresentados na literatura utilizam sensores do tipo *Range Finder*, que informam a distância livre ρ na direção dada por um ângulo ϕ. A coordenada $\mathbf{q} \in \mathbb{R}^2$ corresponde a um ponto da superfície de algum obstáculo na respectiva direção e pode ser obtida através da transformação linear dada pela Eq. (2.2).

A Fig. 3.7 apresenta uma situação onde a abordagem de *scan-to-map matching* é utilizada. O ambiente fictício é constituído por um robô que dispõe de um sensor LiDAR similar àquele descrito na subseção 2.5.1 e encontra-se perante paredes representadas por três linhas assimétricas. O robô também mantém sua localização aproximada $\mathbf{p}_t^+ = (x^+, y^+, \theta^+)$, chamada de pose estimada. Na Fig. 3.7a, o robô recebe do LiDAR uma nuvem de pontos, onde cada um deles é definido pela distância ρ e ângulo ϕ a partir da posição real do robô.

Neste exemplo, o algoritmo de localização projeta a nuvem de pontos a partir da pose estimada $\mathbf{p}_t^+$, obtendo um conjunto de coordenadas dos pontos $\mathbf{Q}_t = \{\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_n\}$ que caracteriza uma amostragem dos contornos do ambiente. Nota-se que a nuvem de pontos não se ajusta sobre os contornos da parede, de modo que cada ponto $\mathbf{q}_i$ está a uma distância d_i do contorno mais próximo. Portanto, a imagem sugere que há uma discrepância considerável entre a pose estimada e a pose real e, para N pontos, pode-se definir o erro da nuvem de pontos em relação ao mapa como o somatório de todos d_i, $i \in [0, 1, ...N]$:

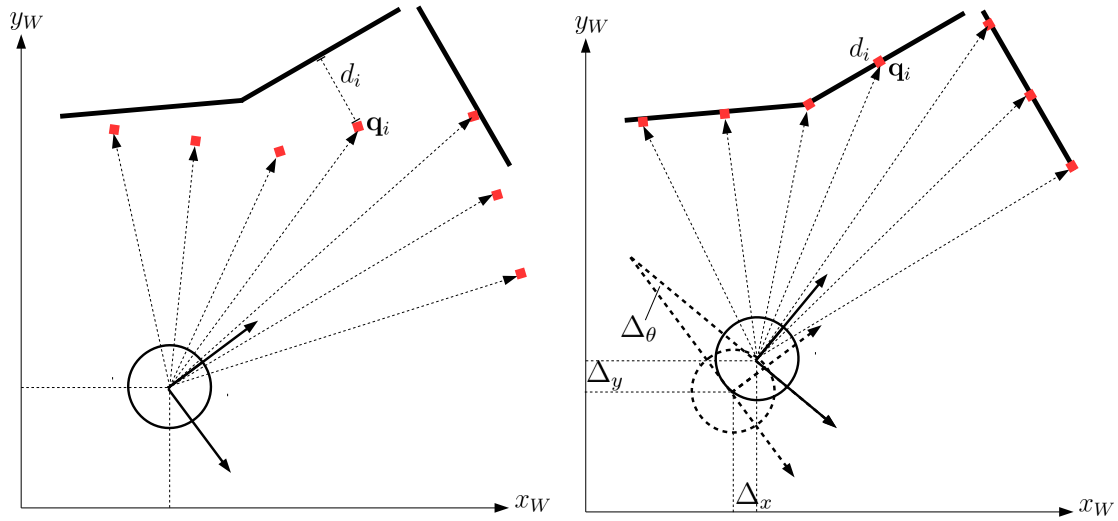
$$E = \sum_{i=1}^{N} \text{dist}({}^W\mathbf{T}_R \, {}^R\mathbf{q}_i) \ , \tag{3.13}$$

onde $({}^W\mathbf{T}_R \, {}^R\mathbf{q}_i)$ é a transformação homogênea do ponto ${}^R\mathbf{q}_i$ representado no sistema de coordenadas do robô R para o sistema de coordenadas global W conforme Eq. (2.6), e dist() é a distância d_i é obtida através de uma TDE no ponto ${}^W\mathbf{q}_i$ do mapa.

A Fig. 3.7b, por sua vez, mostra outra pose estimada $\mathbf{p}_t$ (enquanto $\mathbf{p}_t^+$ está representada por traços pontilhados) de modo que a mesma nuvem de pontos da Fig. 3.7a, agora projetadas em relação a $\mathbf{p}_t$, se ajusta aos contornos do mapa. Desta forma, o erro E é zero, ou muito próximo disso, e a imagem sugere que $\mathbf{p}_t = \mathbf{p}_t^+ + (\Delta_x, \Delta_y, \Delta_\theta)$ constitui a pose correta. Portanto, o rastreamento da pose baseada em *scan-to-map matching* consiste em encontrar a translação (Δ_x,Δ_y) e a rotação Δ_θ sobre a pose $\mathbf{p}_t^+$ que minimiza o erro de *matching* E.

3.3.2 Erro de *matching*

Os algoritmos descritos por Lauer *et al.* e Sobreira *et al.* [11, 14] computam cada nova pose a partir da minimização da função de erro de *matching* E, também cha-



(a) Distribuição da nuvem de pontos antes da otimização da pose

(b) Distribuição da nuvem de pontos após otimização da pose

Figura 3.7: Ilustração da otimização da pose orientada pelo erro de *matching* da nuvem de pontos

mada de função de *fitness*. Por outro lado, os autores apontam a necessidade de reduzir a influência em E das amostras com alto d_i, visto que, frequentemente, estas caracterizam *outliers*, que são medições inesperadas. Portanto, ambos os trabalhos propõem a utilização da função de minimização descrita por:

$$\underset{\Delta_x,\Delta_y,\Delta_\theta}{\text{minimizar}}\ E = \sum_{i=0}^{N} 1 - \frac{L_c^2}{L_c^2 + d_i^2}\ . \tag{3.14}$$

onde d_i é a TDE para um ponto $\mathbf{q}_i$ obtido a partir da pose $\mathbf{p}_t = \mathbf{p}_t^+ + (\Delta_x, \Delta_y, \Delta_\theta)$ e o parâmetro L_c, conforme descrito por Sobreira *et al.* [14], é usado para limitar a influência de pontos do laser muito distantes dos contornos do mapa (d_i elevado). Dessa forma, pontos do laser que destoam exageradamente dos contornos são interpretados como ruídos ou *outliers* e sua influência no cálculo de E é atenuada.

Sobreira *et al.* [14] também definem um teto para o valor de d_i, de forma que, para valores maiores que um parâmetro empírico A, o erro de *matching* deste ponto seria nulo ($d_i = 0, \forall\ d_i > A$). Em termos práticos, isso significa remover pontos da nuvem que destoam significativamente dos contornos. A Fig. 3.8 mostra o gráfico da Eq. (3.13) com a aplicação do limitador A para diferentes valores de L_c.

Nas implementações do Perfect Match de Lauer *et al.* [11], Sobreira *et al.* [14] e Pinto *et al.* [13], a grade de ocupação é binarizada (Fig. 3.9a) e utilizada para construção de uma matriz de distâncias pré-computada que armazena a TDE (Fig. 3.9b). Desta forma, o custo computacional para obter d_i para qualquer ponto $\mathbf{q}_i = (x_{q,i}, y_{q,i})$ é reduzido a uma leitura na matriz de distâncias. Esta característica

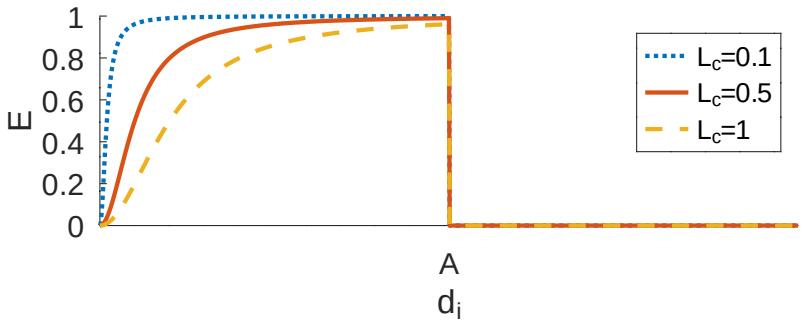


Figura 3.8: Séries com erros de *matching* em função da distância d_i (eixo x), com aplicação do limitador A para diferentes valores de L_c: $0,1$, $0,5$ e $1,0$

contribui para a redução do custo computacional do cálculo de E.

3.3.3 Otimização da pose por RPROP

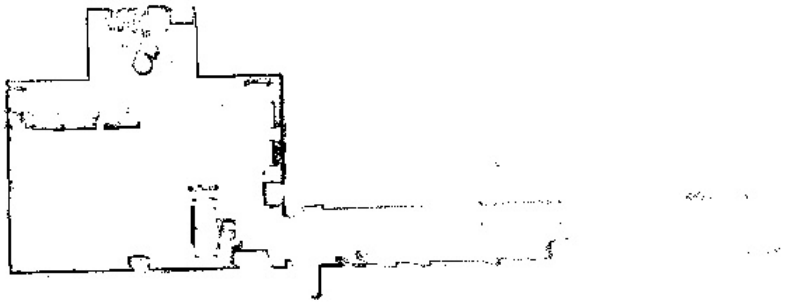
A partir de uma pose $\mathbf{p}_t^+$ e uma nuvem de pontos $\mathbf{Q}_t$, o PM utiliza o algoritmo denominado *Resilient Backpropagation* (RPROP) para minimizar E e obter a pose otimizada $\mathbf{p}_t$. O RPROP consiste em heurística inicialmente proposta por Riedmiller e Braun [69] para aprendizado em redes neurais, mas tem sido aplicado desde então em problemas de otimização. De acordo com Lauer *et al* [11], o RPROP foi originalmente concebido para ser utilizado no aprendizado de redes neurais, mas também pode ser aplicado a outros tipos de problemas de otimização sem restrições.

O objetivo do algoritmo RPROP é alterar o valores dos parâmetros da variável independente de maneira iterativa a fim de encontrar o mínimo local da função. Para isso, o RPROP se baseia apenas no sinal das derivadas parciais da função para incrementar ou decrementar os parâmetros. No caso do PM, o RPROP realiza a minimização descrita pela Eq. (3.14) alterando o valor do parâmetro Δ_x em incrementos η_x^+ ou η_x^-, a depender do sinal da respectiva derivada parcial. O mesmo procedimento ocorre para os demais parâmetros Δ_y e Δ_θ com incrementos η_y^+ ou η_y^- e η_θ^+ ou η_θ^-, respectivamente.

Dado que a derivada do somatório é igual ao somatório da derivada, ou seja, $(\sum E)' = \sum(E')$, os gradientes das matrizes de TDE nas direções x e y (Fig. 3.10) são utilizados para obter as derivadas parciais em x e y da função de erro E, Eq. (3.14). Estes gradientes representam a taxa de variação do erro em cada uma das direções e são obtidas através da aplicação do Filtro de Sobel [70] na matriz de TDE (Fig. 3.9b). No caso do parâmetro θ, a derivada parcial é dada por:

$$\frac{\partial E}{\partial \theta} = \frac{\partial E}{\partial d_i}\left[\frac{\partial \text{dist}(\cdot)}{\partial x_q(\cdot)}\frac{\partial x_q(\cdot)}{\partial \theta} + \frac{\partial \text{dist}(\cdot)}{\partial y_q(\cdot)}\frac{\partial y_q(\cdot)}{\partial \theta}\right] . \tag{3.15}$$

Para que o procedimento de otimização descrito anteriormente possa convergir para uma boa estimativa, a última pose estimada deve estar relativamente próxima

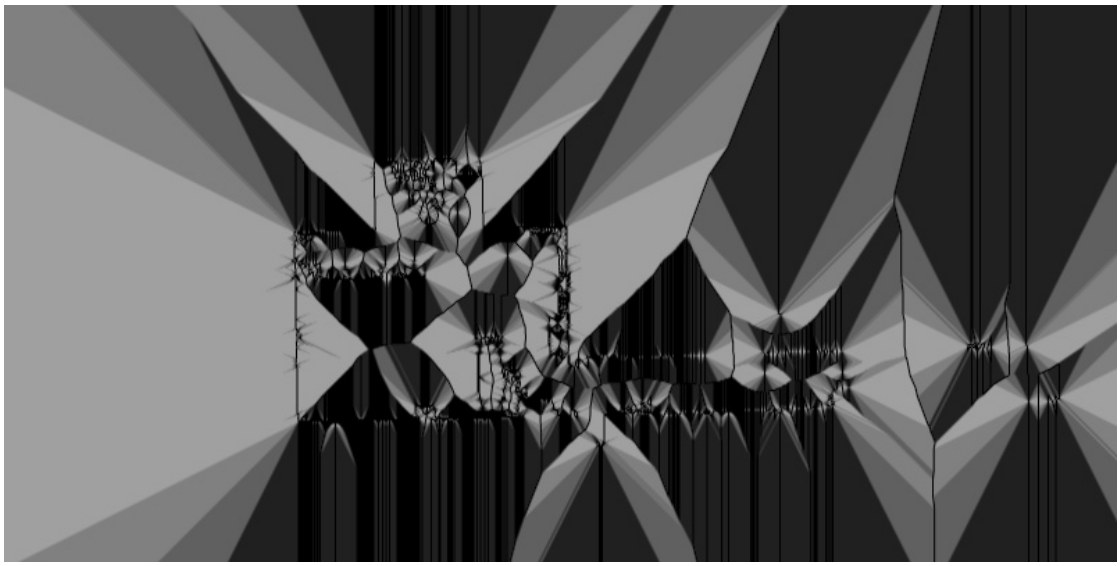


(a) Decomposição binária do mapa representando os contornos do espaço; os *pixels* em preto representam obstáculos, enquanto os *pixels* constituem um espaço livre

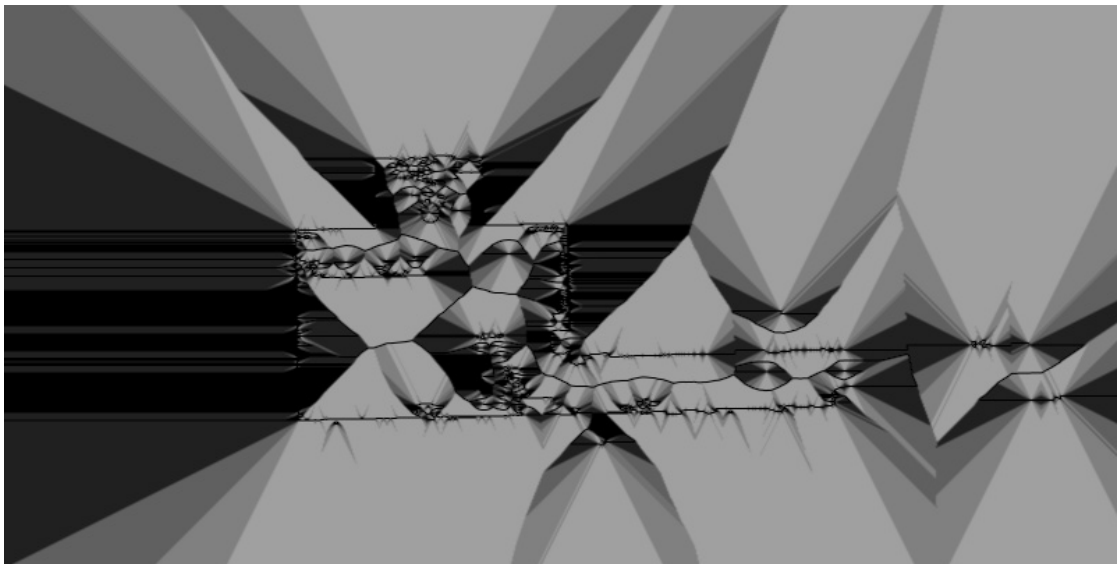
(b) Transformada de distâncias a partir da imagem (a); *pixels* mais escuros representam distâncias menores do contorno mais próximo

Figura 3.9: Imagens pré-computadas a partir da grade de ocupação do LaR mapeado em novembro de 2021

ao *ground truth*. As implementações de Lauer *et al* e Sobreira *et al* executam um número finito de passos do RPROP, embora o último apresente um número variável. Por esse motivo, o PM não se comporta de maneira satisfatória sem uma pose inicial conhecida. Se a última pose estimada está muito distante ou ainda muito rotacionada em comparação com o *ground truth*, a etapa de otimização com o RPROP pode não convergir para uma pose estimada aceitável, ainda que execute por várias iterações.



(a) Gradiente da transformada de distâncias na direção x



(b) Gradiente da transformada de distâncias na direção y

Figura 3.10: Gradientes pré-computados a partir da transformada de distâncias da Fig. 3.9b; *pixels* mais claros representam gradientes maiores (em valor absoluto) na respectiva direção

3.3.4 Suavização da trajetória

O PM é um algoritmo de *pose tracking*, por isso ele deve estimar uma trajetória em conformidade com o movimento do robô, ou seja, sem deslocamentos abruptos. Por outro lado, nas situações em que os dados do sensor oscilam muito (quando ocorre uma vibração no corpo do sensor, por exemplo) a pose estimada pode oscilar na mesma proporção, o que prejudica a qualidade do rastreamento.

Por este motivo, o PM estabelece uma dependência temporal entre as poses como forma de suavizar as oscilações da trajetória. No trabalho de Lauer *et al.* [11] essa

dependência utiliza o modelo de movimento da Eq. (2.29) para estimar a posição $\hat{\mathbf{p}}^+ \in \mathbb{R}^{1\times 2}$ e orientação $\hat{\phi}^+$ *a priori* com dados de *encoders*. Em seguida, o algoritmo estima a nova pose através da média ponderada de $(\hat{\mathbf{p}}^+, \hat{\phi}^+)$ e a pose estimada com o sensor exteroceptivo $(\hat{\mathbf{p}}, \hat{\phi})$:

$$\mathbf{p} = \frac{\mathbf{V}_{(\hat{\mathbf{p}}^+)}\hat{\mathbf{p}} + \mathbf{V}_{(\hat{\mathbf{p}})}\hat{\mathbf{p}}^+}{\mathbf{V}_{(\hat{\mathbf{p}}^+)} + \mathbf{V}_{(\hat{\mathbf{p}})}} \tag{3.16}$$

$$\phi = \frac{\sigma_{\hat{\phi}}^2\hat{\phi}^+ + \sigma_{\hat{\psi}}^2\hat{\phi}}{\sigma_{\hat{\phi}}^2 + \sigma_{\hat{\psi}}^2} \tag{3.17}$$

onde $\mathbf{V}$ é a matriz de covariância da posição e σ^2 é a variância da orientação.

As variâncias e covariâncias foram determinadas a partir da diferença entre sucessivas estimações ajustada por um parâmetro empírico $\alpha > 0$, que segundo os autores controla a precisão do movimento.

Sobreira *et al.* [14], por outro lado, utilizam o EKF para realizar a fusão das poses obtidas a partir da odometria $(\hat{\mathbf{p}}^+, \hat{\phi}^+)$ e do *map matching* $(\hat{\mathbf{p}}, \hat{\phi})$, como mostra o diagrama de blocos da Fig. 3.11. As informações de odometria são usadas para garantir uma suavização da trajetória, visto que o rastreamento com apenas o algoritmo de *map matching*, embora eficaz, resultaria em uma trajetória com "saltos". O bloco em amarelo da Fig. 3.11 mostra que, na etapa de Previsão o EKF usa a pose anterior e as informações de odometria. Na etapa de Correção a pose obtida pelo *map matching* corrige a estimativa obtida a partir da odometria.

Nos trabalhos de Pinto *et al* e Sobreira *et al* [13, 14], a matriz de covariância $\mathbf{\Sigma}_{Match} \in \mathbb{R}^{3\times 3}$ usada no Filtro de Kalman é computada usando a segunda derivada da função de erro (Eq. 3.13 modificada), que segundos os autores reflete a confiança sobre a solução obtida pelo algoritmo de *map matching*.

O diagrama de blocos da Fig. 3.11 também mostra todos os componentes do PM e suas respectivas entradas e saídas no procedimento de *scan-to-map matching*. Antes da otimização por RPROP, há um filtro de *outliers* (Filtro de erro máximo) que remove amostras $\mathbf{q}_i$ onde $d_i > A$. Nota-se que este procedimento é realizado duas vezes, assumindo portanto que há uma otimização intermediária.

3.3.5 Incerteza sobre a pose

O PM estima a incerteza através da determinação da estrutura da função de erro nas proximidades da pose otimizada. Na prática, quando pontos do laser se encontram no eixo de uma parede longa e paralela ao eixo x, por exemplo, não há um mínimo distintivo na direção x e o valor da segunda derivada parcial $\frac{\partial^2 E}{(\partial x)^2}$ é pequeno. Portanto, é possível deduzir se cada uma das variáveis da pose (x, y, θ) estão na região de um mínimo local. Lauer, Lange e Riedmiller [11] sugerem que as

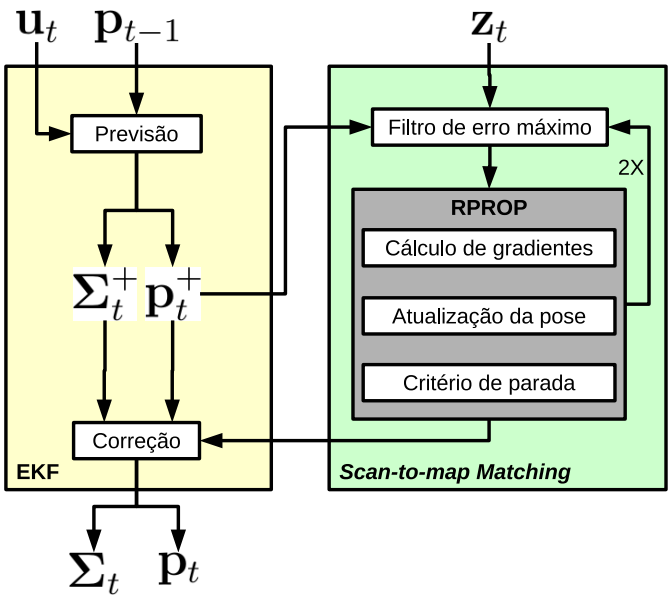


Figura 3.11: Diagrama de blocos do PM na versão utilizada neste trabalho; uma iteração do PM inicia (à esquerda) buscando a pose anterior $\mathbf{p}_{t-1}$ e dados da odometria $\mathbf{u}_t$ para realizar a previsão da pose $\mathbf{p}_t^+$ e da incerteza $\boldsymbol{\Sigma}_t^+$; $\mathbf{p}_t^+$ é usada para projetar a nuvem de pontos $\mathbf{z}_t$, e os pontos com erro de *matching* muito grande são filtrados da nuvem; à direita, o RPROP realiza a otimização, produzindo uma pose *a posteriori* que é utilizada na etapa de correção do EKF; figura adaptada de Sobreira [17].

derivadas parciais de segunda ordem $\frac{\partial^2 E}{(\partial x)^2}$, $\frac{\partial^2 E}{(\partial y)^2}$ e $\frac{\partial^2 E}{(\partial \theta)^2}$ de E poderiam ser utilizadas como uma medida de incerteza sobre x, y e θ, respectivamente. Porém, devido à limitações da aplicação que envolve o estudo (detecção de linhas de um campo de futebol de robôs), os autores utilizam a função de erro simplificada $E = \frac{1}{2}(\frac{d_i}{L_c})^2$ ao invés da Eq. (3.13) e ignoram *outliers*.

As implementações do PM por Sobreira *et al.* e Pinto *et al.* também utilizam a segunda derivada da função de erro, porém há uma fusão com a incerteza da odometria, desde que disponível, pois essas implementações possuem um Filtro de Kalman incorporado para realizar tal fusão. Nota-se que, em ambos os casos a estrutura da função de erro de *matching* é usada para calcular a incerteza, mas o próprio valor do erro não é considerado (embora seja utilizado na etapa de otimização da pose). Deste modo, deduz-se que um baixo valor de E não necessariamente consiste em uma alta confiança sobre a pose.

3.4 Localização global

A seção anterior mostrou que o Filtro de Kalman e o *Perfect Match* dependem de uma pose anterior conhecida e próxima do *ground truth*. O AMCL, por sua vez, é robusto o suficiente para prosseguir com o rastreamento de pose mesmo que a pose anterior seja desconhecida, desde que haja partículas nas proximidades do *ground truth* – mas ainda assim não há garantia de que o algoritmo irá convergir para a pose correta. Portanto, quando não há confiança sobre a pose anterior o robô se encontra em um caso especial da localização, denominado Problema da Localização Global.

O PLG é um tema bem conhecido na área de robótica móvel, entretanto diversas abordagens ainda são objetos de pesquisa. Recentemente Campbell *et al.* [71] e Wu *et al.* [72] introduziram e classificaram estratégias típicas de localização em robótica. Para limitar o escopo da revisão da literatura, este trabalho se concentrou nos estudos envolvendo veículos terrestres não tripulados (*Unmanned Ground Vehicle*s – UGVs) que utilizam uma grade de ocupação previamente construída e navegam em ambientes *indoor* não estruturados. Portanto, estudos sobre localização baseada em *landmarks*, que demandam extração de características das medições dos sensores, detecção e associação de *landmarks*, não foram incluídos na revisão dos trabalhos relacionados. Portanto, este trabalho está restrito a métodos que utilizam uma nuvem de pontos em 2D que são periodicamente produzidos por uma sistema de varredura a laser.

Muitos pesquisadores têm se dedicado particularmente a métodos relacionados aos problemas do rastreamento de pose [11, 44, 73, 74] e SLAM [40–42, 75, 76] porque em muitas aplicações a pose inicial é fornecida ao robô ou o mesmo está apenas explorando o ambiente e construindo o mapa iterativamente (SLAM). Porém, para executar tarefas em ambientes críticos de maneira autônoma, o robô deve ser capaz de recuperar sua pose sem se basear em poses anteriores – situação conhecida como Localização Global.

A principal estratégia para resolver o PLG é distribuir hipóteses de pose aleatórias ao longo do mapa e utilizar um método de filtro de partículas, como o MCL. Ao longo dos últimas anos, diversos estudos propuseram variações do MCL [23,45,46,77] ou se dedicaram a aprimorar e avaliar o desempenho do método amplamente utilizado, o AMCL [53, 78, 79].

Por outro lado, a revisão da literatura realizada neste trabalho identificou poucas publicações que exploram o PLG utilizando métodos baseados em heurísticas de IE e AE. Algumas iniciativas incluem o DE, que foi estudado no contexto de localização baseada em contornos [47, 52], bem como a Otimização por Enxame de Partículas (*Particle Swarm Optimization* – PSO) [48,49,80] ou ainda uma combinação de DE e

PSO [81]. Existem ainda implementações com outros métodos baseados em IE como o Algoritmo do Morcego [82], Colônia de Abelhas Artificial (*Artificial Bee Colony* – ABC) e o Algoritmo do Vagalume (*Firefly Algorithm* – FA) [51]. No entanto, não foram encontrados na literatura trabalhos que explorassem o desempenho de tais métodos de maneira exaustiva (ou seja, realizando de dezenas de ensaios em condições distintas) e comparativa (analisando a eficácia e o custo computacional relativos). Há ainda a possibilidade de estudar outras heurísticas não citadas anteriormente no contexto da localização de robôs, como o Simulated Annealing e Covariance matrix adaptation evolution strategy (CMA-ES), que não foram incluídos por limitação de tempo de realização deste trabalho.

Este trabalho propõe demonstrar que, ao lado do AG [83], MCL, PSO e DE possuem uma característica conveniente para o PLG em robôs móveis: uma boa relação entre custo computacional e habilidade para explorar os mínimos locais e o mínimo global. Entretanto, foram encontrados poucos estudos que analisam o desempenho destas heurísticas no contexto do PLG em condições heterogêneas. Por exemplo, este trabalho se concentra na avaliação do desempenho das heurísticas em situações críticas como ambientes de alta complexidade [49,53], corredores [10,46,84] e lugares com contornos simétricos [72].

Além disso, as heurísticas apresentadas neste trabalho focam na implementação acoplada com o PM, como forma de viabilizar um sistema de localização robusto, conforme descrito no Capítulo 4. É válido destacar que, em experimentos preliminares que precederam este trabalho, foi desenvolvido um método de localização global baseado no PSO e no PM [22]. Naquele trabalho, parte do PM foi utilizado como uma função de *fitness* para o PSO. Os resultados mostrados na subseção 4.1.3 indicam que esta solução apresentou um desempenho satisfatório em experimentos com dados reais.

O PM também foi investigado no contexto do problema do sequestro e do PLG. Farias *et al.* [16] apresentaram o PM e o AMCL executando de forma concorrente e monitorados por um sistema supervisório, com o objetivo de trazer as melhores características de cada método para um sistema de localização híbrido. Nos testes que representavam cenários críticos, especialmente quando um dos métodos começa a desviar da pose correta, o sistema supervisório foi capaz de identificar e corrigir a estimação incorreta de um método utilizando a estimação da outra parte, baseando-se em indicadores de qualidade do rastreamento de ambos.

Capítulo 4

Metodologia Proposta

Este capítulo apresenta as propostas desenvolvidas neste trabalho para aprimorar o algoritmo de localização *Perfect Match*. Estas contribuições se dividem em duas vertentes: Localização Global (seção 4.1) e incerteza sobre a pose (seção 4.2). Ao final, a metodologia de desenvolvimento dos experimentos é apresentada na seção 4.3.

4.1 Localização Global para o Perfect Math

Uma abordagem intuitiva para realizar a localização global de um robô móvel é distribuir uma grande quantidade de hipóteses de poses ao longo do mapa, aleatoriamente ou uniformemente, e escolher aquela mais provável baseando-se em alguma métrica de desempenho (uma função de *fitness*, por exemplo). Esta métrica deve levar em conta as informações trazidas pelos sensores e o atual estado do robô. Para resolver o PLG com o PM, Pinto *et al.* [15] propôs um método para otimizar e filtrar várias hipóteses de pose progressivamente. Apesar de seu alto custo computacional, visto que o algoritmo PM é executado para cada hipótese, os autores reivindicam que o método é eficiente em dois cenários distintos: no futebol de robôs (*RoboCup Middle Size League* – MSL) utilizando visão computacional e em um ambiente *indoor* típico usando nuvens de pontos em 3D obtidas a partir de um sensor LiDAR.

Por outro lado, soluções que demandam alto custo computacional normalmente não são bem vindas porque os recursos computacionais são limitados, especialmente quando se trata de *hardware* embarcado no robô. Sendo assim, soluções que entregam resultados satisfatórios com menor volume de processamento são preferidos, especialmente em aplicações de tempo real. Entretanto, o procedimento de localização global requer uma quantidade razoável de leitura dos sensores, a fim de aumentar a variedade de informações produzidas coletadas. [3, p. 245]. Por exemplo, em casos onde o ambiente possui muitas ambiguidades, o método de localização pode se con-

fundir com locais parecidos. Neste caso, o robô não deveria se limitar à percepção apenas em um local fixo, e portanto é desejável que ele se desloque no espaço para aumentar a diversidade de informações coletadas pelos sensores.

Sendo assim, se o robô está em movimento, diferentes leituras são obtidas dos sensores devido a mudança da pose relativa dos obstáculos em volta. Por exemplo, se o robô está posicionado perante um canto de uma sala quadrada, a nuvem de pontos produzida pelo laser pode ser bem parecida com a nuvem esperada se estivesse de frente a algum dos outros três cantos. A mesma ambiguidade pode ocorrer no meio de um corredor longo, que produz uma nuvem de pontos que desenha duas retas paralelas, similar à nuvens de pontos esperadas em outras partes do corredor. Para fugir destas ambiguidades, o robô pode se mover pelas redondezas a fim de capturar contornos singulares e portanto descartar hipóteses improváveis. Desta forma, conforme o robô se move, as hipóteses devem ser atualizadas com os mesmos deslocamentos.

A partir desta ideia, os métodos de Localização Global apresentados neste trabalho são capazes de realizar a localização à medida que o robô se move. Na maioria dos robôs, é comum a presença de um sistema de odometria que fornece periodicamente as velocidades linear v_t^o e angular ω_t^o do veículo extraídas dos seus sensores. Neste trabalho, v_t^o e ω_t^o são utilizados para atualizar a pose de cada hipótese $s_k = (x_k, y_k, \theta_k)$ dos algoritmos de localização global após o intervalo entre iterações consecutivas Δt_k. A pose no instante t é determinada conforme o modelo de movimento baseado em velocidade descrito pela Eq. (2.29), porém quando $\omega_t^o = 0$ a Eq. (2.24) de MRU é utilizada no lugar.

Neste capítulo, a partir da subseção 4.1.2, serão apresentados os métodos de localização global para o PM baseados em IE ou AE. Porém, antes de serem descritos individualmente, a subseção 4.1.1 descreve as características dos algoritmos que são comuns entre si. Sendo assim, os métodos são diferentes apenas no que se refere à heurística utilizada para atualização das hipóteses, o que implica em diferentes dinâmicas de evolução da população. Estas dinâmicas estão ilustradas no Apêndice C

4.1.1 Características compartilhadas entre os algoritmos

Os algoritmos de localização descritos no restante da seção implementam os conceitos abordados anteriormente. Todos os algoritmos dependem das seguintes informações para operar:

- Mapa de grade de ocupação;
- Nuvem de pontos em 2D que representam contornos de obstáculos em volta;

- Velocidades linear e angular instantâneas fornecidas por algum sistema de odometria.

A Fig. 4.1 ilustra o fluxograma genérico dos métodos de localização global, de forma que os algoritmos descritos nas seções 4.1.2, 4.1.3, 4.1.4 e 4.1.5 diferem na etapa destacada em cinza. Os métodos possuem pelo menos quatro parâmetros configuráveis compatíveis com o AMCL: deslocamentos linear e angular mínimos para iniciar uma iteração ($d_{\min}$ e $a_{\min}$, respectivamente) e tamanhos de população máximo e mínimo, $P_{\max}$ e $P_{\min}$, respectivamente. Neste trabalho $P_{\max}$ é equivalente à população inicial P_{init}, visto que o algoritmo sempre inicia com o tamanho máximo da população. Esta compatibilidade permite que todos os algoritmos de localização possuam equivalência no grau de diversidade das medições dos sensores e iniciem com o mesmo número de hipóteses.

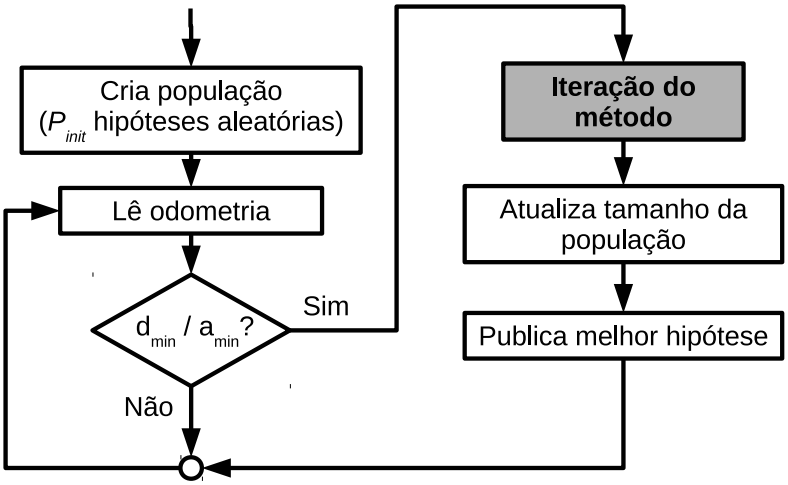


Figura 4.1: Fluxograma do método de localização global. O retângulo em cinza representa o passo no qual os métodos diferem entre si

Os algoritmos de localização global desenvolvidos neste trabalho são apresentados nas seções seguintes. A seção 4.1.2 apresenta uma implementação enxuta de um algoritmo de localização multi-hipóteses, o *Perfect Match Global Localization* (PMGL), baseado na abordagem proposta por Pinto *et al.* [15]. Em seguida, os métodos baseados em inteligência de enxame *Particle Swarm Global Localization* (PSGL) e *Differential Evolution Global Localization* (DEGL) são descritos nas seções 4.1.3 e 4.1.4, respectivamente. Por fim, o *Genetic Algorithm Global Localization* (GAGL), método evolucionário baseado em AG, é apresentado na seção 4.1.5.

Deslocamento mínimo

Para evitar a utilização de nuvens de pontos muito parecidas obtidas de poses consecutivas, os algoritmos demandam de, pelo menos, um deslocamento maior que $d_{\min}$ ou uma mudança de orientação maior que $a_{\min}$ para iterar (ou seja, atualizar as hipóteses). Portanto, em cada iteração, as hipóteses são atualizadas pela Eq. (2.29) e as respectivas importâncias usando a equação de *fitness* descrita a seguir. Porém, cada algoritmo utiliza uma heurística própria para mover ou substituir hipóteses ao longo da evolução de sua população.

Tamanho dinâmico da população

Em todos os algoritmos propostos neste trabalho, a população inicial é constituída por P_{init} hipóteses distribuídas aleatoriamente sob uma distribuição de probabilidade uniforme. Entretanto, apenas hipóteses situadas em células livres do mapa são consideradas. Portanto, para formar a população inicial, os métodos geram indivíduos aleatoriamente até que haja P_{init} hipóteses situadas em células livres. O Apêndice A mostra a formação da população inicial em diferentes circunstâncias.

Os algoritmos de localização possuem um mecanismo para ajustar o tamanho da população ao longo das iterações e mantê-lo entre $P_{\min}$ e P_{init}. O AMCL possui uma maneira própria de adaptar a quantidade de indivíduos, conforme já descrito na seção 3.2. O PMGL, por sua vez, reduz a sua população à medida que seus candidatos convergem para uma região, conforme descrito na seção 4.1.2. Já os métodos PSGL, DEGL e GAGL se baseiam em uma mecanismo simples de filtragem de hipóteses ao longo das iterações, descrito como segue.

Em cada iteração, um número fixo de $f_{\max}$ indivíduos são filtrados, limitando-se às $i_{\max}$ primeiras iterações. A escolha do valor de $i_{\max}$ se baseia na velocidade de convergência dos métodos, garantindo-lhes uma quantidade suficiente de iterações para que convirja (a seção 4.3 apresenta a metodologia para determinar $i_{\max}$, enquanto o valor escolhido para este trabalho está apresentado no Capítulo 5). A ideia consiste em eliminar aos poucos $f_{\max}$ candidatos de baixo desempenho até alcançar um tamanho de população mínimo $P_{\min}$.

Entretanto, se por um lado eliminar indivíduos aleatoriamente pode resultar na exclusão de candidatos relevantes, por outro, comparar todos os indivíduos para encontrar e descartar o pior implica em um alto esforço computacional. Portanto, com o objetivo de manter uma boa relação entre eficiência de filtragem e custo computacional, um grupo pequeno de indivíduos é selecionado aleatoriamente e o pior deles é removido. Esta abordagem é detalhada como segue.

Na primeira iteração, existem P_{init} hipóteses. Em cada iteração consecutiva, o seguinte procedimento é realizado $f_{\max}$ vezes: quatro hipóteses são selecionadas ale-

atoriamente da população e suas respectivas importâncias são comparadas entre si. Então, o candidato com a menor importância é removido da população. A filtragem é interrompida quando o tamanho da população for igual a $P_{\min}$. Conforme descrito pela Eq. (4.1), $f_{\max}$ é o número de candidatos filtrados por iteração necessários para encolher a população de P_{init} para $P_{\min}$ ao longo de $i_{\max}$ iterações:

$$f_{\max} = \frac{P_{\text{init}} - P_{\min}}{i_{\max}}. \tag{4.1}$$

Função de *fitness*

Assim como a maioria das heurísticas de otimização, uma função de *fitness* é necessária para avaliar o desempenho de soluções candidatas. A função de *fitness* utilizada pelos métodos de localização descritos neste capítulo são baseados no erro de *matching* do PM. Porém, utilizar a função exibida na Fig. 3.8 pode ser prejudicial para os algoritmos baseados em população porque um candidato ruim com muitos pontos descartados pode assumir, eventualmente, um valor de *fitness* melhor que um candidato razoável com muitos pontos preservados. Esta situação é possível por causa da etapa de filtragem de amostras de pontos de laser muito distantes dos contornos, utilizando o parâmetro A. Portanto, neste trabalho, este parâmetro não foi utilizado e a importância de cada solução candidata $\mathbf{s}_k$, dados N pontos na nuvem, é uma adaptação da Eq. (3.13), dada por:

$$\text{score}(\mathbf{s}_k) = \begin{cases} \text{err}(\mathbf{s}_k)^{-1} & \text{se } \mathbf{s}_k \text{ está sobre uma célula livre,} \\ 0 & \text{caso contrário,} \end{cases} \tag{4.2}$$

onde

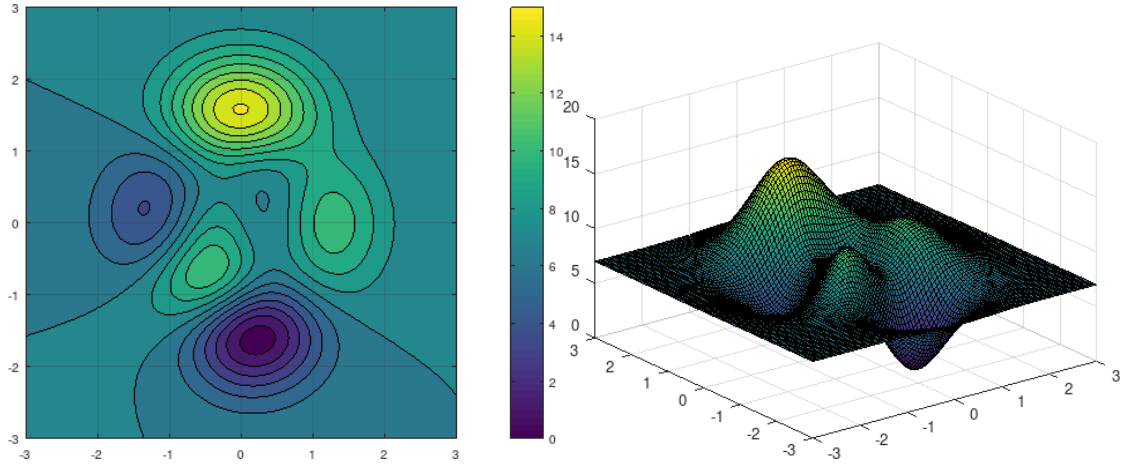
$$\text{err}(\mathbf{s}_k) = \sum_{i=0}^{N-1} \left(1 - \frac{L_c^2}{L_c^2 + [\text{dist}({}^W\mathbf{T}_R \, {}^R\mathbf{q}_i)]^2} \right). \tag{4.3}$$

Nota-se que, ao contrário da equação de erro de *matching*, a Eq. (4.2) atribui valores maiores para poses com menor erro. Além disso, hipóteses que se encontram em células ocupadas recebem importância igual a zero, como forma de penalizar candidatos improváveis, visto que tais hipóteses podem, eventualmente, assumir uma alta similaridade entre a nuvem de pontos e os contornos do mapa.

Todas as heurísticas apresentadas neste capítulo se baseiam na Eq. (4.2) – uma adaptação do erro de *matching* do PM. As hipóteses são constituídas por variáveis de três dimensões (x, y, θ), de maneira que a população se encontra em um espaço de busca 3D. Porém, para exibir graficamente uma função de *fitness* para um espaço de busca em 3D seria necessário em um gráfico de quatro dimensões. Então, um gráfico de *fitness* em função da pose não seria adequado para ilustrar a dinâmica das heurísticas.

Por isso, as seções 4.1.3 4.1.4 e 4.1.5 apresentam ilustrações do comportamento das heurísticas com partículas dispersas em um espaço de busca 2D, ao invés de 3D, apenas como exemplos hipotéticos. Estas ilustrações servem para demonstrar o princípio de funcionamento de cada método dentro de um espaço de busca limitado a $(x, y) \in ([-3, 3], [-3, 3])$. Neste caso, utilizou-se a função *Peaks* descrita pela Eq. (4.4) que possui alguns máximos e mínimos locais, como mostra a Fig. 4.2. O *Peaks* foi escolhido porque é um recurso disponível no *software* Octave[1] que exibe uma função com diversos máximos e mínimos locais, sendo útil neste trabalho para a demonstração dos métodos de otimização a seguir.

$$f(x,y) = 3(1-x)^2 e^{[-x^2-(y+1)^2]} - 10(\frac{x}{5} - x^3 - y^5)e^{(-x^2-y^2)} - \frac{1}{3}e^{[-(x+1)^2-y^2]} \quad (4.4)$$



(a) Gráfico em 2D, com barra de cores à direita

(b) Gráfico de superfície

Figura 4.2: Gráficos da função *Peaks* – Eq. (4.4), com as cores indicando os máximos e mínimos locais

4.1.2 Perfect Match Global Localization

Uma maneira intuitiva para realizar a localização global é distribuir várias hipóteses ao longo do mapa e executar algum algoritmo de rastreamento de pose para cada hipótese. Então, a solução mais provável pode ser escolhida comparando as hipóteses entre si. O PMGL reproduz esta abordagem executando o PM para cada solução candidata. Para que possa suavemente convergir para uma solução final, o algoritmo elimina iterativamente as poses com baixa importância e combina candidatos similares (isto é, muito próximos).

[1] `https://octave.sourceforge.io/octave/function/peaks.html`

O PMGL é uma adaptação do procedimento *Initial Position Computation* descrito por Pinto *et al.* [15]. O algoritmo inicia distribuindo aleatoriamente P_{init} hipóteses no espaço livre do mapa. Nos passos seguintes, as hipóteses são atualizadas periodicamente de acordo com as velocidades linear e angular fornecidas pelo sistema de odometria. Sempre que o robô realiza um deslocamento mínimo linear (d_{min}) ou angular (a_{min}), as hipóteses são atualizadas com o mesmo deslocamento e em seguida três operações são realizadas: otimização das poses, atualização das importâncias e eliminação de hipóteses. A Fig. 4.3 resume a sequência de operações dentro de uma iteração do PMGL.

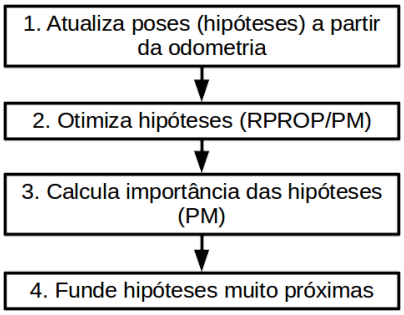


Figura 4.3: Fluxograma do PMGL dentro de uma iteração; os passos 1 e 3 utilizam as Eqs.(2.29) e (4.3), respectivamente

A otimização das poses é baseada no RPROP, como mencionado na seção 3.3. Todas as soluções candidatas são otimizadas individualmente por meio da minimização do erro de *matching* entre a mais recente nuvem de pontos do laser e os contornos do mapa.

O PMGL não aplica a dependência temporal entre estimações consecutivas, como proposto por Lauer *et al.* [11], nem a incorporação da odometria utilizando o EKF, proposta por Sobreira *et al.* [14]. A dependência temporal implica na redução da possibilidade de explorar as regiões próximas, uma vez que a movimentação das hipóteses se torna limitada. A incorporação da odometria, por sua vez, é realizada pelo PMGL conforme descrito na seção 4.1.1 e ilustrando na Fig. 4.3.

A otimização das poses utilizando o RPROP (conforme descrito na subseção 3.3.3) é realizada a cada iteração. Sendo assim, é possível que hipóteses próximas sejam otimizadas para poses bastante similares, representando um mínimo local. Há, portanto, uma etapa de fusão de hipóteses similares – aquela com menor *score* é removida da população. Dois parâmetros definem a distância máxima para combinar duas hipóteses: d_{merge} e a_{merge}.

Pinto *et al.* [15] utilizou $d_{merge} = 70$ cm e $a_{merge} = 45°$ argumentando que o RPROP é capaz de convergir para um mínimo local quando a distância entre as hipóteses é menor que estes parâmetros (neste caso, o cenário foi um campo de futebol de robôs). Entretanto, considerando que os cenários descritos no capítulo

seguinte são possivelmente mais complexos, neste trabalho foram utilizados limites mais restritos. Dado que os resultados dos testes apresentados na subseção 5.2.1 mostram que, em alguns casos, a inicialização do PM não é adequada para erros da pose maiores que 12 cm, considerou-se neste trabalho $d_{merge} = 10$ cm. Com relação ao erro angular da pose, caso seja maior que 10°, podem ocorrer erros de matching maiores que 30 cm em amostras a mais de 2 m de distância. Sendo assim, utilizou-se $a_{merge} = 10°$. Este valores, definido empiricamente mas inspirados na observação dos experimentos, caracterizam uma abordagem mais conservadora e pode favorecer a diversidade populacional do PMGL. Esta etapa de fusão é realizada sempre que houver mais do que P_{min} hipóteses. Finalmente, o indivíduo com o maior *score* é selecionado como a pose estimada pelo PMGL

Devido ao fato de que o PMGL realiza a otimização da hipótese e compara-a com as demais, este método tem um alto custo computacional. Apesar de ser uma abordagem simples e direta, o PMGL apresenta esta desvantagem em relação a outros métodos se forem consideradas limitações de recursos de *hardware*.

4.1.3 Particle Swarm Global Localization

Eberhart e Kennedy [85] introduziram o método PSO para otimização de funções não-lineares. De modo geral, a meta-heurística inspirada na natureza consiste em mover um conjunto de P soluções candidatas dentro de um espaço em D dimensões em direção às regiões com as melhores soluções encontradas. O algoritmo mantém o *score* de cada partícula obtida até então $\mathbf{p}_{best} \in \mathbb{R}^D$ (*Personal Best*) e o melhor score de todo o enxame $\mathbf{g}_{best} \in \mathbb{R}^D$ (*Global Best*). A cada iteração, o PSO atualiza a posição $\mathbf{p}_i \in \mathbb{R}^D$ e a velocidade $\mathbf{v}_i \in \mathbb{R}^D$ de cada partícula i utilizando as Eqs. (4.5) and Eq. (4.6):

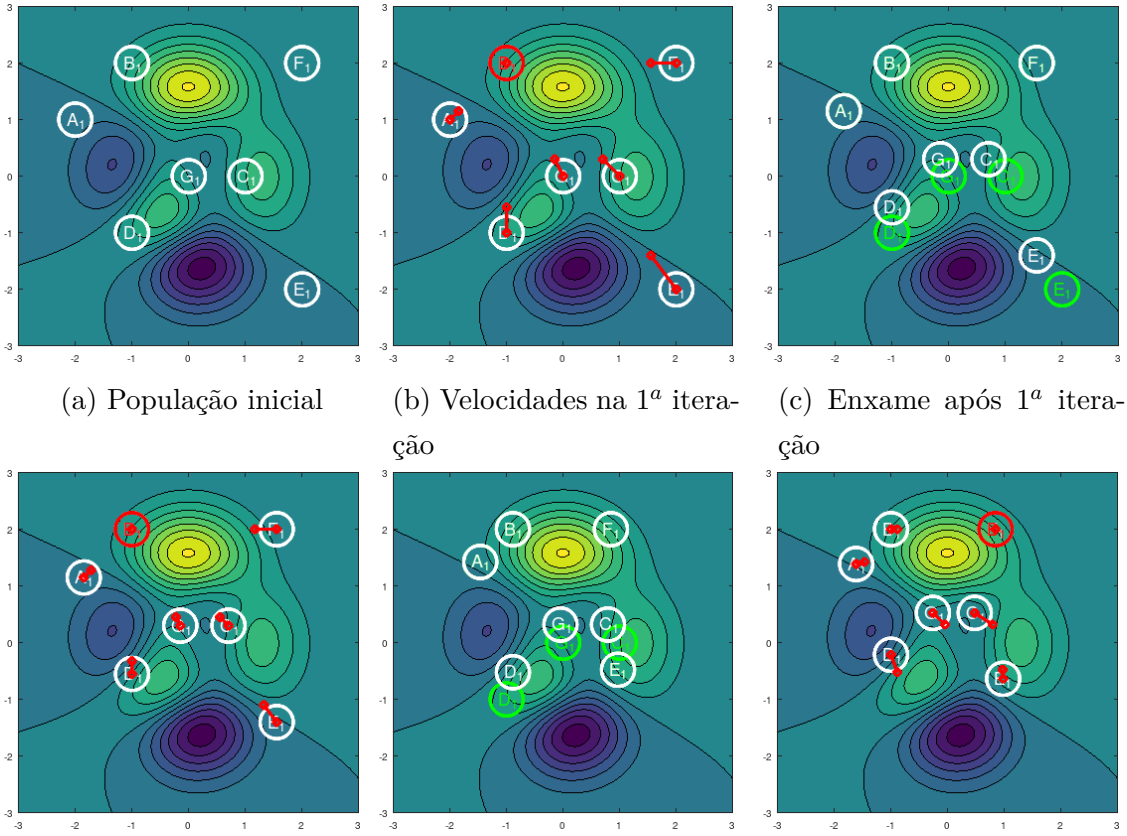
$$\mathbf{p}_{i(k)} = \mathbf{p}_{i(k-1)} + \mathbf{v}_{i(k-1)} \tag{4.5}$$

$$\mathbf{v}_{i(k)} = K[\mathbf{v}_{i(k-1)} + \phi_1 r_{1(k)}(\mathbf{p}_{best} - \mathbf{p}_{i(k)}) + \phi_2 r_{2(k)}(\mathbf{g}_{best} - \mathbf{p}_{i(k)})], \tag{4.6}$$

onde ϕ_1 e ϕ_2 são constantes de aceleração e $r_{1(k)}$ e $r_{2(k)}$ são números pseudo-aleatórios gerados a partir de uma distribuição uniforme $U(0, 1)$. O parâmetro K, conforme Eberhart e Shi [86], é um fator de constrição definido pela Eq. (4.7) para reduzir a excitação das partículas nas iterações posteriores , sendo útil para estabilizar as partículas à medida que o enxame converge para uma solução otimizada.

$$K = \frac{2}{\left|2 - \psi - \sqrt{\psi^2 - 4\psi}\right|}, \quad \text{onde} \begin{cases} \psi = \phi_1 + \phi_2, \\ \psi > 4. \end{cases} \tag{4.7}$$

A Fig. 4.4 mostra um enxame de partículas distribuídas ao longo do espaço de busca da Eq. (4.4). O objetivo do PSO é encontrar a solução com maior *score*, que seria representada por uma partícula no centro da região de tom amarelo. Uma população inicial $P = \{\mathbf{A}_1, \mathbf{B}_1, ..., \mathbf{E}_1\}$ está distribuída aleatoriamente no espaço, como mostra a Fig. 4.4a. Neste instante inicial, o indivíduo com maior *score* é $\mathbf{B}_1$, portanto, este é o $\mathbf{g}_{best}$.



(a) População inicial (b) Velocidades na 1^a iteração (c) Enxame após 1^a iteração

(d) Velocidades na 2^a iteração (e) Enxame após 3^a iteração (f) Velocidades na 4^a iteração

Figura 4.4: Ilustração das etapas do PSO em um espaço de busca 2D. Partículas estão representadas em branco, $\mathbf{p}_{bests}$ em verde e o $\mathbf{g}_{best}$ em vermelho. Os traços vermelhos ilustram a velocidade e a direção das partículas

Em seguida, as velocidades das partículas são atualizadas pela Eq. (4.6), conforme mostrado pelos vetores em vermelho na Fig. 4.4b. Observa-se que as partículas se movem em direção ao $\mathbf{g}_{best}$ (nesta ilustração, o valor de $r_{1(k)}$ é bem menor que $r_{2(k)}$ para favorecer a demonstração do método).

Após a atualização das hipóteses pela Eq. (4.5), as partículas assumem novas posições (Fig. 4.4c). Nota-se que os scores de $\mathbf{C}_1$, $\mathbf{D}_1$, $\mathbf{E}_1$ e $\mathbf{G}_1$ reduziram, e portanto os respectivos $\mathbf{p}_{bests}$ correspondem às posições anteriores e estão representados em verde, enquanto que para $\mathbf{A}_1$, $\mathbf{B}_1$ e $\mathbf{F}_1$ os respectivos $\mathbf{p}_{bests}$ são as próprias poses. Em seguida, a Fig. 4.4d mostra que as partículas continuam a se mover em direção

ao $\mathbf{g}_{best}$, $\mathbf{B}_1$.

Em iterações seguintes mostradas nas Figs. 4.4e e 4.4f, $\mathbf{F}_1$ assume um *score* maior que $\mathbf{B}_1$, de modo que o enxame substitui seu $\mathbf{g}_{best}$. Também nota-se que as partículas $\mathbf{C}_1$, $\mathbf{D}_1$ e $\mathbf{G}_1$ apresentam $\mathbf{p}_{bests}$ em posições anteriores, e portanto suas velocidades são uma combinação entre os respectivos $\mathbf{p}_{bests}$ e o $\mathbf{g}_{best}$. Portanto, observa-se que, apesar de $\mathbf{B}_1$ e $\mathbf{F}_1$ estarem mais próximos do mínimo global, $\mathbf{C}_1$, $\mathbf{D}_1$ e $\mathbf{G}_1$ exploram mínimos locais em regiões distintas.

Este trabalho apresenta o PSO executando de maneira compartilhada com o PM de forma a obter o PSGL, um algoritmo de localização global que estima a pose do robô dentro de uma grade de ocupação em 2D. Um enxame inicial é gerado e distribuído aleatoriamente ao longo do espaço livre do mapa. Nas iterações seguintes, a pose e a velocidade de cada partícula são atualizadas usando as Eqs. (4.5) e (4.6). Além disso, conforme descrito na subseção 4.1.1, a odometria do robô é utilizada para manter as hipóteses sincronizadas com o robô.

A função de *fitness* é o erro de *matching* para cada partícula, computado conforme a Eq. 4.2. Se score($\mathbf{p}_i$) é o maior valor para o indivíduo i até então, $\mathbf{p}_{best,i}$ é substituído por $\mathbf{p}_i$. Se score(p_i) é o maior valor para todo o enxame, $\mathbf{g}_{best}$ é substituído por $\mathbf{p}_i$. A pose estimada pelo PSGL em cada iteração é representada pelo $\mathbf{g}_{best}$. A Fig. 4.5 resume o fluxograma do PSGL em uma iteração.

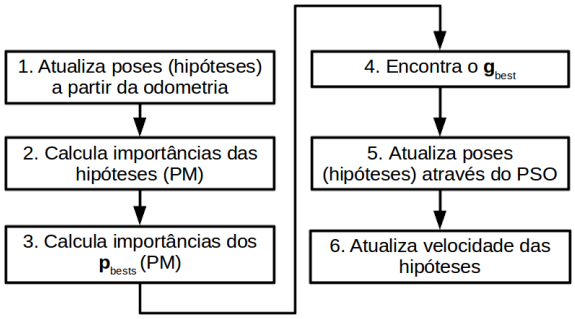


Figura 4.5: Fluxograma do PSGL dentro de uma iteração; os passos 2 e 3 utilizam a Eq.(4.3), enquanto os passos 1, 5 e 6 utilizam as Eqs. (2.29), (4.5) e (4.6), respectivamente

4.1.4 Differential Evolution Global Localization

A Evolução Diferencial (*Differential Evolution* – DE) foi proposta por Storn & Price [87] como um método de otimização baseado em um conjunto de candidatos (a população) e uma equação simples. Cada candidato é representado por um vetor de elementos que compreendem uma possível solução. A população inicial é distribuída aleatoriamente pelo espaço de busca assumindo uma distribuição de probabilidade uniforme. O método DE se baseia em criar desvios nos indivíduos da população

extraindo informações de distância e direção de outros três indivíduos selecionados aleatoriamente: para cada hipótese $\mathbf{s}_k$, o método gera uma solução $\mathbf{z}_k$ combinada entre outro indivíduo $\mathbf{a}_k$ e a diferença ponderada entre dois indivíduos distintos $\mathbf{b}_k$ e $\mathbf{c}_k$. Se o novo indivíduo gerado se mostrar uma melhor solução (possuir melhor *fitness*), este substitui $\mathbf{s}_k$.

Três parâmetros são decisivos no processo de otimização por DE. $F \in \mathbb{R}$ é uma constante que adiciona um peso para a diferença entre dois indivíduos. O parâmetro de probabilidade de cruzamento CR e o índice aleatório R são usados para substituir elementos do vetor da solução com o objetivo de introduzir diversidade na população.

A essência do DEGL é descrita a seguir, utilizando como exemplo um espaço de busca 2D. No início de uma nova iteração i, os indivíduos são atualizados de acordo com as informações da odometria utilizando a Eq. (2.29). Em seguida, para cada indivíduo $\mathbf{s}_k = (x, y, \theta)$ da população os seguintes passos do método DE são executados:

1. Três indivíduos distintos $\mathbf{a}_k$, $\mathbf{b}_k$, e $\mathbf{c}_k$ diferentes de $\mathbf{s}_k$ são selecionados aleatoriamente;

2. Encontra-se uma solução intermediária $\mathbf{z}'_k$:

$$\mathbf{z}'_k = \mathbf{a}_k + F \times (\mathbf{b}_k - \mathbf{c}_k) \tag{4.8}$$

3. Seleciona-se aleatoriamente um índice $R \in [0, 1]$;

4. Para cada j-ésimo parâmetro de $\mathbf{z}'_k$:

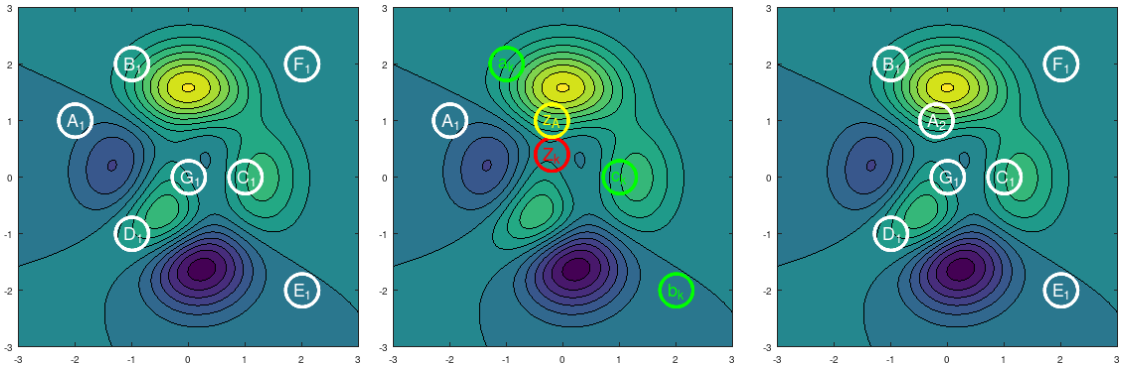
   (a) seleciona-se aleatoriamente um número aleatório r_i de uma distribuição uniforme $U(0, 1) \in \mathbb{R}$;

   (b) encontra-se $\mathbf{z}_k(j)$ de acordo com:

$$\mathbf{z}_k(j) = \begin{cases} \mathbf{z}'_k(j), & \text{se } r_i < CR \text{ ou } j = R \\ \mathbf{s}_k(j), & \text{caso contrário.} \end{cases} \tag{4.9}$$

A Fig. 4.6 mostra uma parte do método DE para o problema de otimização da Fig. 4.2. A população inicial $P = \{\mathbf{A}_1, \mathbf{B}_1, ..., \mathbf{F}_1\}$ é mostrada na Fig. 4.6a. Na iteração $k = 0$, o indivíduo $\mathbf{A}_1$ é escolhido e outros três são selecionados aleatoriamente, $a_k = \mathbf{B}_1$, $b_k = \mathbf{E}_1$ e $c_k = \mathbf{C}_1$, como mostra a Fig. 4.6b, destacando a_k, b_k e c_k em verde. Uma nova solução $\mathbf{Z}_k$ é gerada, mostrada em vermelho.

Em seguida, devido ao procedimento que envolve o índice R e o número aleatório r_i, apenas a coordenada horizontal de $\mathbf{Z}_k$ é mantida, enquanto a coordenada vertical é mantida conforme a anterior $\mathbf{A}_1$. A nova solução obtida $\mathbf{Z}_A$ possui *score* maior que



(a) População inicial

(b) Cruzamento entre três indivíduos

(c) População após substituição de A_1 pelo novo indivíduo A_2

Figura 4.6: Ilustração das etapas do DE em um espaço de busca 2D

$\mathbf{A}_1$, e portanto esta última é substituída na população, como mostra a Fig. 4.6c. Estas etapas são executadas também para os demais indivíduos.

O algoritmo DEGL se baseia no DE1-scheme descrito por Storn e Price [87] com adaptações para satisfazer o PLG. Os parâmetros F e CR são configurados em $0,8$ e $0,9$, respectivamente, seguindo uma recomendação genérica de Price *et al.* [88].

O *score* do indivíduo $\mathbf{z}_k$ recém criado a partir de $\mathbf{s}_k$ e outros três selecionados é dado pela Eq. (4.2). Finalmente, se a importância de $\mathbf{z}_k$ for maior que a importância de $\mathbf{s}_k$, então o último é substituído pelo primeiro no conjunto de hipóteses. A Fig. 4.7 ilustra os passos do DEGL em uma iteração.

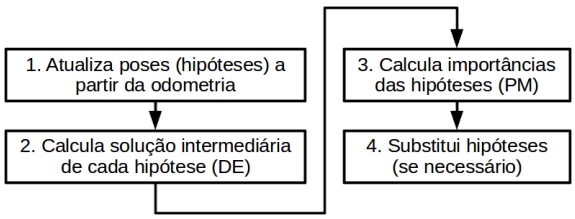


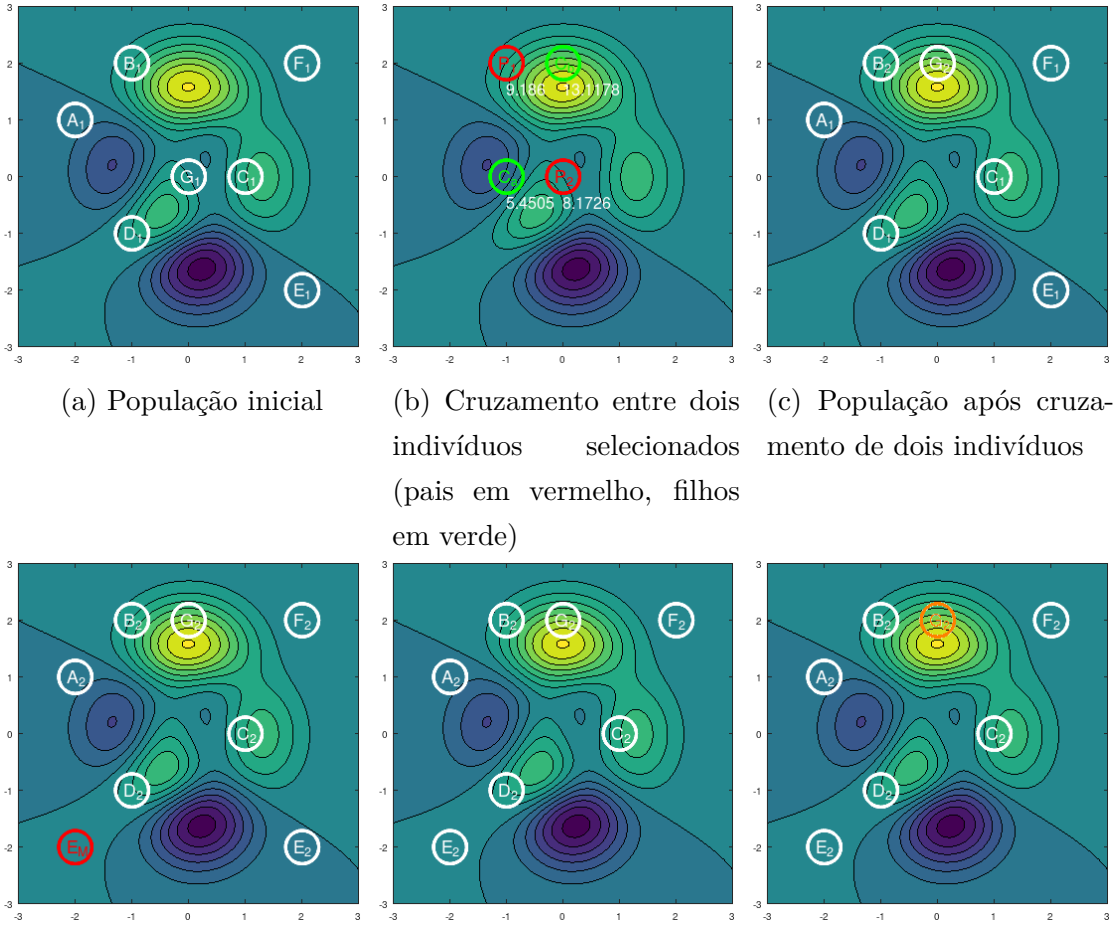
Figura 4.7: Fluxograma do DEGL em uma iteração; os passos 1, 2, 3 e 4 utilizam as Eqs. (2.29),(4.8),(4.3) e (4.9), respectivamente

4.1.5 Genetic Algorithm Global Localization

O AG é uma meta-heurística bioinspirada amplamente estudado nas áreas do conhecimento que envolvem problemas de busca e otimização. Conhecido desde a década de 70 através do trabalho de Holland [83], o algoritmo se baseia nos operadores de seleção natural típicos de biologia, como seleção, cruzamento (*crossover*) e mutação. Várias soluções candidatas distribuídas no espaço de busca do problema compreendem a população. O algoritmo consiste em evoluir a população através de gerações

consecutivas desta população utilizando os operadores citados anteriormente e uma medida de qualidade de cada solução – a função de *fitness*.

A Fig. 4.8 mostra uma iteração do AG para o problema de otimização da Fig. 4.2. A população inicial $P = \{\mathbf{A}_1, \mathbf{B}_1, ..., \mathbf{F}_1\}$ é mostrada na Fig. 4.8a. O *score* de cada indivíduo é computado, de modo que a probabilidade de ser selecionado como um dos "pais" da geração seguinte deve ser proporcional.



(a) População inicial

(b) Cruzamento entre dois indivíduos selecionados (pais em vermelho, filhos em verde)

(c) População após cruzamento de dois indivíduos

(d) Mutação de um indivíduo ($\mathbf{E}_2$ se torna $\mathbf{E}_M$, em vermelho)

(e) População após mutação de um indivíduo

(f) População após elitismo (o melhor indivíduo está em laranja)

Figura 4.8: Ilustração das etapas do Algoritmo Genético em um espaço de busca 2D

Na Fig. 4.8b dois indivíduos $p_1 = \mathbf{B}_1$ e $p_2 = \mathbf{G}_1$ são selecionados e o cruzamento entre eles geram dois indivíduos c_a, "filho" de $\mathbf{B}_1$ e c_b, "filho" de $\mathbf{G}_1$. A figura também mostra logo abaixo dos indivíduos seus respectivos *scores*. Visto que score(p_1) >score(c_a) e score(p_2) <score(c_b), $\mathbf{B}_1$ permanece (agora, $\mathbf{B}_2$) na população e $\mathbf{G}_1$ é substituído pelo novo indivíduo $\mathbf{G}_2$.

A mutação ocorre com probabilidade M_P e, na população representada pela Fig. 4.8d, $\mathbf{E}_2$ sofre mutação para $\mathbf{E}_M$, destacado em vermelho. Uma vez que score($\mathbf{E}_M$) >score($\mathbf{E}_2$), $\mathbf{E}_2$ é substituído. Por fim, a etapa de elitismo garante que

o melhor indivíduo faça parte da geração seguinte (mesmo que não tenha sido selecionado para realização do cruzamento). Na prática, se o melhor indivíduo da nova população recém formada for melhor que o melhor indivíduo da população anterior, não há alteração; caso contrário, o melhor indivíduo da população anterior é trazido para a nova população, entrando no lugar do pior indivíduo da nova população. A Fig. 4.8f mostra que o indivíduo $\mathbf{G}_2$ (em laranja) é melhor do que o melhor indivíduo da população anterior (Fig. 4.8a) e por isso o melhor indivíduo da população anterior ($\mathbf{C}_1$) não foi trazido.

O GAGL é um algoritmo de localização baseado em AG. Cada indivíduo é representado por uma hipótese de pose $\mathbf{s}_k$ no mapa 2D, e seus atributos são as coordenadas e a orientação desta pose (x, y, θ). Uma população inicial com um número par de indivíduos P_{init} distribuída uniformemente ao longo da área não ocupada do mapa é concebida. A medida de qualidade, ou importância, de cada indivíduo é dado pela Eq. (4.2). Conforme citado anteriormente, GAGL evolui a população quando o robô realiza um deslocamento linear ou angular mínimos, d_{min} ou a_{min}, respectivamente. A Fig. 4.9 resume os passos realizados dentro de uma iteração do GAGL.

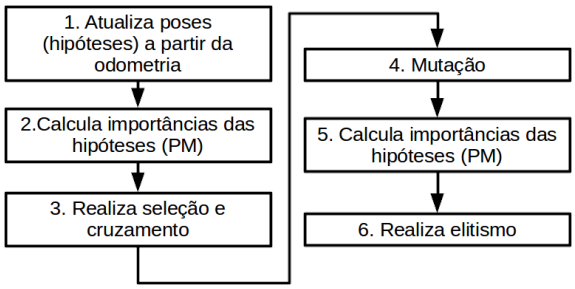


Figura 4.9: Fluxograma do GAGL em uma iteração; o passo 1 utiliza a (2.29) e os passos 2 e 5 utilizam a Eq. (4.3)

A etapa de Seleção é baseada no *Fitness Proportionate Selection*, também conhecido como seleção por roda de roleta. Nesta abordagem, as hipóteses são alocadas em uma memória do tipo *buffer* circular, de modo que o espaço ocupado neste círculo é proporcional ao seu valor de *fitness*. Em seguida, um ponteiro escolhe de modo aleatório um dos pontos do *buffer* que irá apontar para o espaço ocupado por uma das hipóteses. Na prática, isto significa que indivíduos com maior importância têm mais chances de serem selecionados para serem ancestrais da próxima geração de indivíduos.

No GAGL, a implementação do *crossover* é uma permutação simples dos atributos (x_1, y_1, θ_1) e (x_2, y_2 e θ_2) de dois ancestrais $\mathbf{s}_1$ e $\mathbf{s}_2$. A probabilidade de ocorrer um *crossover* $0 \leq C_R \leq 1 \in \mathbb{R}$ influencia a frequência da permuta dos atributos, portanto um número aleatório $0 \leq l_c \leq 1$, $l_c \in \mathbb{R}$ é gerado para cada par de ances-

trais, e se $l_c < C_R$ então um dos atributos aleatoriamente escolhidos (x, y or θ) sofre a permuta.

A etapa de mutação é utilizada para manter a diversidade da população, substituindo parcialmente os atributos de alguns indivíduos por valores aleatórios restritos. O parâmetro de mutação $0 \leq M_P \leq 1 \in \mathbb{R}$ determina a probabilidade de ocorrer tais substituições. Um número aleatório $0 \leq l_m \leq 1$, $l_c \in \mathbb{R}$ é gerado para cada atributo de cada indivíduo para determinar se a mutação deve ser realizada.

Após o *crossover* e a mutação os valores de *fitness* dos novos indivíduos são atualizados. Finalmente, uma etapa de Elitismo é executada para garantir que o melhor indivíduo da geração anterior seja parte da próxima geração. Neste caso, o indivíduo com menor importância da nova geração é substituído pelo indivíduo com maior importância da geração anterior.

4.2 Incerteza sobre a pose

O Capítulo 3 mostrou que a incerteza sobre a pose é, de alguma forma, inferida a partir de variáveis aleatórias dos modelos probabilísticos. Entretanto, as constantes que modelam os erros normalmente dependem da configuração do robô, das características do ambiente e frequentemente são estabelecidas a partir de experimentos em ambiente controlado [55]. Por outro lado, existem algoritmos de *map matching* que estimam a incerteza em tempo de execução a partir das informações sobre a percepção. O PM, por exemplo, utiliza as derivadas parciais de segunda ordem da função de erro de *matching* como grandezas que caracterizam a incerteza. Nas implementações do PM por Lauer *et al.* [11] e Sobreira *et al.* [14], este cálculo se baseia apenas nos gradientes obtidos do mapa. Portanto, nesta seção, propõe-se que o erro de *matching* também seja utilizado para estimar a incerteza sobre a pose.

A partir dos gradientes da transformada de distâncias de uma grade de ocupação em 2D, pode-se dizer quão distinto é um mínimo local obtido na otimização da pose. Por outro lado, pode-se considerar que o erro de *matching* apresentado na seção 3.3.2 carrega informação útil a respeito da confiança sobre a pose. Neste sentido, será descrito a seguir uma nova abordagem para estimar a incerteza que se baseia no erro de *matching* e nos gradientes nas direções x e y. A abordagem proposta é denominada Proposta de Método para Cálculo da Covariância (PMCov) e está ilustrada no fluxograma da Fig. 4.10.

No início, este método recebe a grade de ocupação e calcula a TDE e as imagens de gradientes utilizando o Filtro de Sobel presente na biblioteca de visão computacional Open Source Computer Vision Library (OpenCV)[2]. Para fins de ilustração,

[2]OpenCV está disponível sob licença Apache 2.0 em `https://opencv.org/`

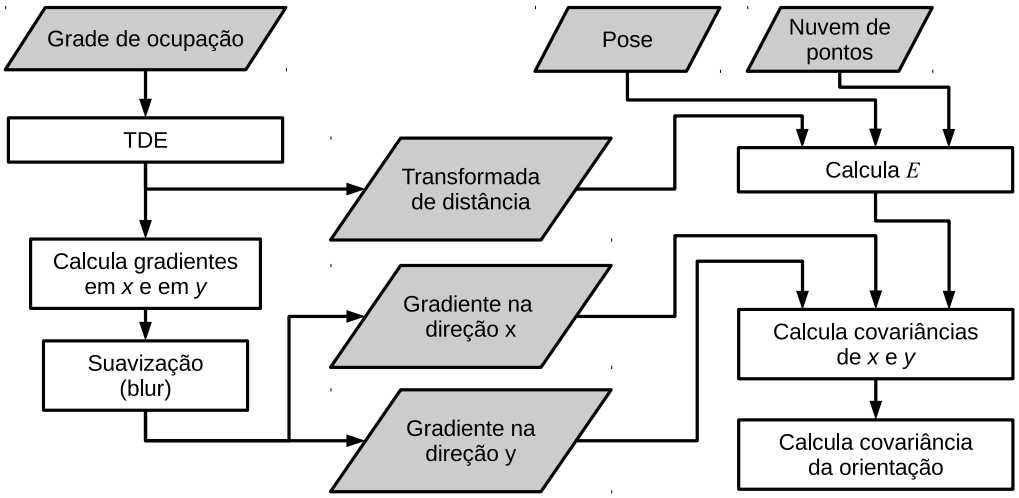


Figura 4.10: Fluxograma da abordagem proposta para o cálculo da incerteza

considera-se a grade de ocupação do Departamento de Engenharia Elétrica e de Computação da Universidade Federal da Bahia (DEEC/UFBA) a aproximadamente 45 cm do piso, exibido na Fig. 4.11. Neste ambiente, observa-se uma sala à esquerda (o LaR) e um saguão à direita, ligados por um longo corredor. O corredor possui aproximadamente 51 m de comprimento e 1 m de largura, enquanto que as regiões do mapa que compreendem o LaR e o saguão possuem aproximadamente 90 m^2 e 150 m^2 de área, respectivamente.

Para obter uma TDE adequada, a grade de ocupação é previamente convertida em uma imagem binária para que sejam representadas apenas células ocupadas e não-ocupadas (como mostra a Fig. 4.12). Conforme mencionado na subseção 3.3.2, cada pixel da transformada de distâncias contém a distância até a célula ocupada mais próxima – a imagem resultante é apresentada pela Fig. 4.13. Portanto, quando uma amostra do laser incide sobre um contorno do mapa, espera-se que a transformada de distância atribua zero a este ponto.

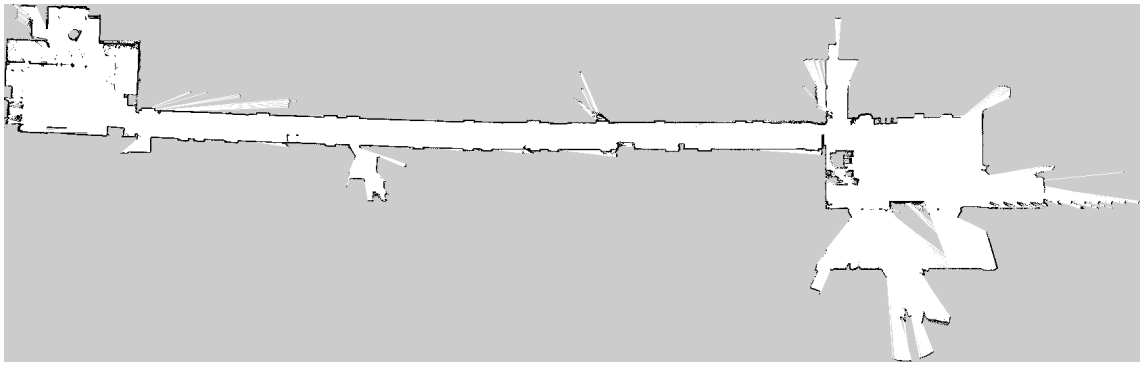


Figura 4.11: Grade de ocupação do DEEC/UFBA, com resolução de 5 cm, abrangendo uma área de aproximadamente $26,5 \times 84$ m.

Figura 4.12: Grade de ocupação do DEEC/UFBA binarizada

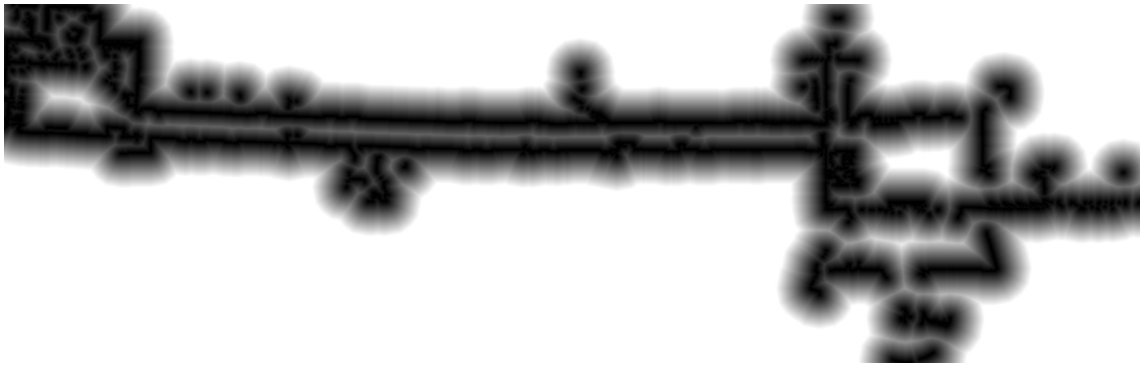
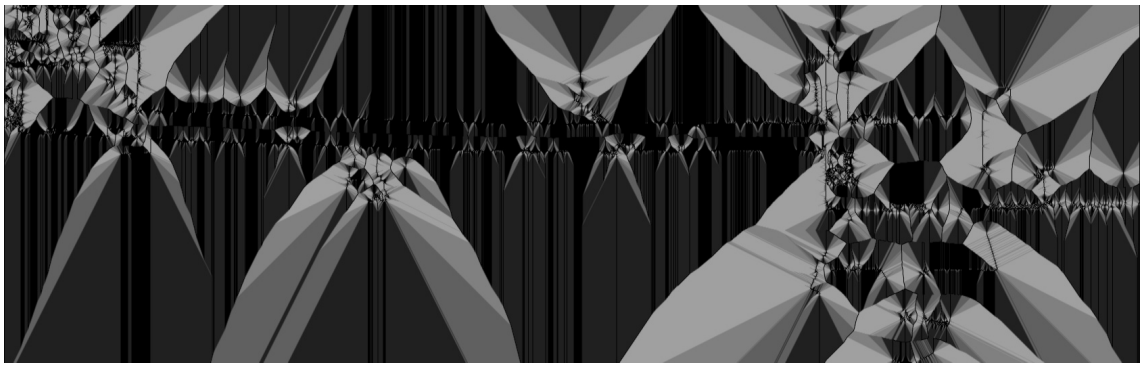


Figura 4.13: Mapa da transformada de distâncias da grade de ocupação do DEEC/UFBA: pixels mais escuros significam distâncias menores até o contorno mais próximo

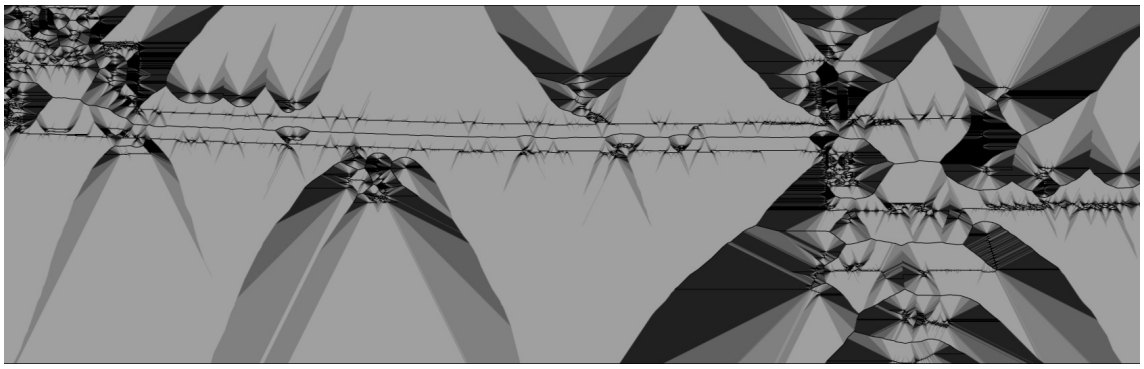
No passo seguinte, os gradientes na horizontal e na vertical são obtidos através do cálculo das primeiras derivadas em relação a x e a y utilizando o operador Scharr [89] – um método alternativo ao Filtro de Sobel. Nota-se que os gradientes podem ser positivos ou negativos, mas neste contexto apenas os valores absolutos são utilizados.

As imagens produzidas possuem altas frequências, conforme exposto nas Figs. 4.14a e 4.14b que mostram apenas o módulo das derivadas. Portanto, na sequência, um filtro passa-baixa é aplicado para suavizar as bordas das imagens, evitando a variação abrupta dos gradientes à medida que o robô se move. As imagens suavizadas dos gradientes na horizontal e na vertical são mostradas nas Figs. 4.15a e 4.15b, respectivamente.

Após o pré-processamento da grade de ocupação, o método PMCov recebe a pose estimada pelo PM e a nuvem de pontos do laser para computar o erro de *matching* utilizando a mesma Eq. (3.13) do PM. Este cálculo ocorre sempre que uma nova pose é estimada pelo PM – ou seja, durante o rastreamento da pose. Até este momento, estão disponíveis a pose, a última nuvem de pontos do laser, os gradientes e o erro de *matching*. Estas informações são utilizadas para estimar as variâncias das coordenadas x, y e θ que caracterizam a incerteza sobre a pose.



(a) Mapa de gradientes na direção x



(b) Mapa de gradientes na direção y

Figura 4.14: Mapas de gradientes do DEEC/UFBA

Utilizando os mapas de gradientes suavizados G_x e G_y e cada ponto do laser $\mathbf{s}_k$ com coordenadas (x_k, y_k), tem-se a média dos gradientes Grad_x e Grad_y de N pontos do laser:

$$\text{Grad}_x = \sum_{k=0}^{N-1} \frac{|G_{x,\mathbf{s}_k}|}{N} \ , \qquad \text{Grad}_y = \sum_{k=0}^{N-1} \frac{|G_{y,\mathbf{s}_k}|}{N}. \tag{4.10}$$

Gradientes maiores representam maior confiança sobre a pose otimizada, ao passo que maior erro de *matching* indica menor certeza. Portanto, as variâncias σ_x^2 e σ_y^2 são determinadas pelas equações a seguir que utilizam o erro de *matching* normalizado $E_N = \frac{E}{N}$:

$$\sigma_x^2 = \text{cov}(x,x) = \frac{E_N}{(\text{Grad}_x)^2} \ , \ \sigma_y^2 = \text{cov}(y,y) = \frac{E_N}{(\text{Grad}_y)^2}. \tag{4.11}$$

Finalmente, a variância angular σ_θ^2 é determinada empiricamente baseada em E_N e na diferença relativa entre as médias dos gradientes na horizontal e vertical:

$$\sigma_\theta^2 = \text{cov}(\theta,\theta) = E_N \cdot \pi \cdot \left| \frac{\text{Grad}_x - \text{Grad}_y}{\text{Grad}_x + \text{Grad}_y} \right|. \tag{4.12}$$

(a) Mapa de gradientes na direção x após suavização

(b) Mapa de gradientes na direção y após suavização

Figura 4.15: Mapas de gradientes do DEEC/UFBA após suavização

4.3 Metodologia dos experimentos

Esta seção descreve a metodologia dos experimentos realizados e os recursos utilizados. Este trabalho utilizou a plataforma robótica *Unmanned Ground Vehicle* (UGV) Husky A200 controlada por um sistema baseado no *framework Robot Operating System* (ROS). O Husky possui um modelo virtual consistente e uma unidade física disponível no LaR. O ROS e a plataforma robótica Husky estão descritos nas subseções 4.3.1 e 4.3.2, respectivamente.

Testes preliminares descritos na subseção 4.3.3 foram realizados em ambiente controlado para avaliar a viabilidade de uma das propostas de localização global, baseada em PSO. Em seguida, simulações e ensaios exaustivos foram executados para todos os métodos apresentados na subseção 4.1. Estes métodos são sistemas baseados em ROS e foram testados em cenários virtuais e reais. As simulações em ambiente virtual são pertinentes porque é possível construir ambientes com as características desejadas e obter dados sem influência de ruídos. Os ensaios em ambiente físico, por sua vez, servem para validar o sistema de localização em um contexto real. As simulações em ambientes virtuais são descritas na subseção 4.3.4, enquanto os ensaios reais são apresentados na subseção 4.3.5.

Detalhes sobre a metodologia utilizada para avaliação dos métodos de localização

global, incluindo a análise do custo computacional, são apresentados na subseção 4.3.6. Por fim, a subseção 4.3.7 descreve os ensaios para avaliação do método de estimação da incerteza sobre a pose.

4.3.1 *Robot Operating System*

Os sistemas que operam nos robôs são tipicamente constituídos por rotinas de *software* distintas que controlam e coordenam a execução de tarefas complexas de forma concorrente. Muitas vezes, estas rotinas são construídas por diferentes desenvolvedores, mas ainda assim precisam executar de maneira colaborativa. Por isso, ao lado da acelerada expansão da robótica, surge a necessidade de um sistema capaz de incorporar de maneira rápida e eficiente os diferentes componentes de *software* que são executados em um robô.

O ROS é um *framework* desenvolvido para facilitar a construção, integração e execução de rotinas ou processos, chamados de nós, em um sistema robótico. Com ele, é possível iniciar nós de forma simultânea e estabelecer uma comunicação estruturada, assíncrona e independente. Por exemplo, é possível iniciar um nó para ler as informações de odometria, em paralelo com um nó para ler dados de uma câmera e enviar as informações processadas de ambos para um terceiro nó que executa um algoritmo de localização. Em grande parte da literatura recente utilizada neste trabalho, as soluções de robótica apresentadas constituem sistemas baseados em ROS [8–10, 14, 22, 26, 28, 33, 35, 39, 53, 84].

A comunicação entre os nós pode ocorrer através de diferentes modelos, como *Publishsubscribe*, *Request-response* ou *Request-response* preemptivo. Estes três modelos estão presentes nas interfaces implementadas no ROS, *Topics*, *Services* e *Actions*, respectivamente.

Os *Topics*, ou tópicos, são pertinentes quando há um fluxo de dados contínuo que pode interessar a mais de um nó, como dados dos sensores a laser, por exemplo. *Services*, ou serviços, são normalmente utilizados em chamadas de procedimento remoto que terminam rapidamente, quando um processo consulta o estado de outro, por exemplo. Já o modelo de comunicação *Actions*, ou ações, é similar ao *Services*, porém é aplicável em uma chamada que demora para ser concluída, mas que pode fornecer um *feedback* da sua execução ou até mesmo ser interrompida. Uma tarefa que demora vários segundos para ser executada é um exemplo de uma aplicação do modelo *Actions*.

Geralmente, os tópicos são mais frequentemente utilizados porque, dentre os modelos, permite a maior independência entre os nós. Os tópicos armazenam mensagens enviadas por um nó até que aqueles interessados nas mensagens leiam. É possível que mais de um nó envie, ou publique, em um mesmo tópico, assim como

mais de um nó pode receber, ou se inscrever, em um mesmo tópico. A Fig. 4.16 ilustra um grafo com nós (elipses) que publicam ou se inscrevem em tópicos (retângulos).

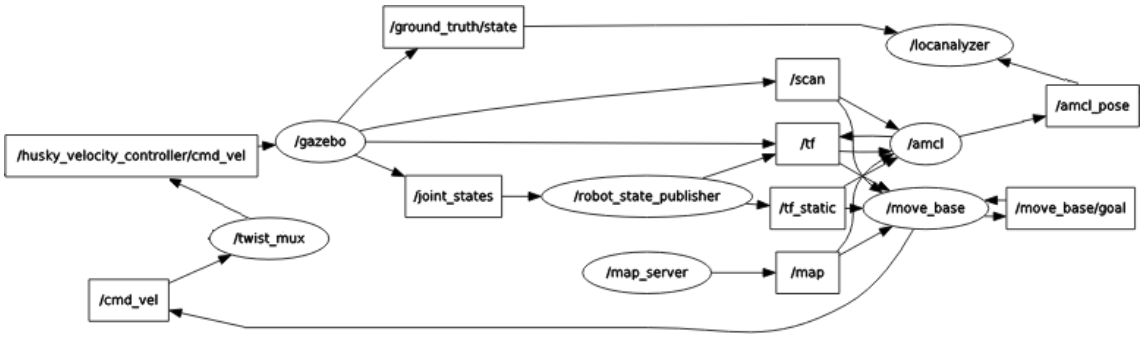


Figura 4.16: Grafo de tópicos (retângulos) e nós (elipses) de um sistema baseado em ROS

A Fig. 4.16 ilustra, oportunamente, os nós e tópicos que compõem o sistema baseado em ROS utilizado para realizar as simulações neste trabalho. O `gazebo` é responsável por realizar a simulação das interações físicas entre o robô e o ambiente. Por isso, este nó publica a pose do modelo virtual do robô (tópico `ground_truth/state`) e o estado de das partes móveis (tópico `joint_states`). O `gazebo` também publica a nuvem de pontos detectada pelo sensor LiDAR (tópico `scan`), que também possui um modelo, e a árvore de transformadas do Husky nos tópicos `tf` e `tf_static` que armazenam a estrutura apresentada na Fig. 2.6.

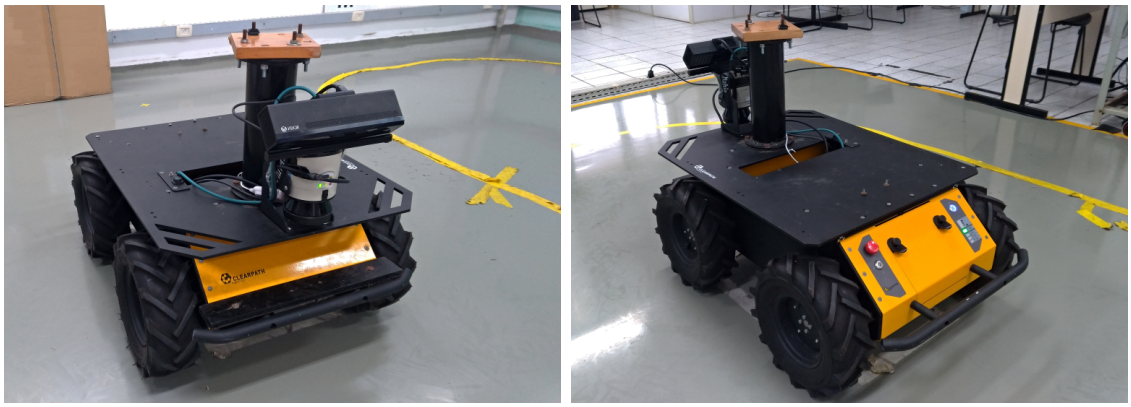
O nó `move_base` guia o robô utilizando os tópicos `move_base/goal` e `cmd_vel`, bem como o mapa armazenado no tópico `map` fornecido pelo nó `map_server`. Já os nós `twist_mux` e `robot_state_publisher`, são rotinas auxiliares utilizadas para traduzir mensagens em diferentes formatos.

O algoritmo de localização é representado pela elipse que publica a pose estimada. No caso desta figura, o algoritmo é o AMCL e o respectivo nó, `amcl`, publica a pose estimada em `amcl_pose`. Por fim, o nó `locanalyzer` faz parte do mecanismo de avaliação desenvolvido neste trabalho e compara a pose estimada pelo método de localização com o *ground truth* a fim de determinar o sucesso da localização.

Uma das principais vantagens de utilizar o ROS é a possibilidade de reproduzir ensaios realizados. O *Framework* permite que as mensagens registradas em tópicos sejam gravadas em arquivo, denominado *bag* para posterior reprodução. Este recurso permite, por exemplo, que ensaios realizados na plataforma robótica sejam reproduzidos virtualmente, em outro momento, quantas vezes forem necessárias, representando fielmente o conteúdo e o carimbo de tempo de cada mensagem.

4.3.2 Plataforma Husky

O Husky é uma plataforma robótica *all-terrain*[3] com tração diferencial e independente em quatro rodas, capacidade de carga de até 75 Kg e alimentado por uma bateria para uso contínuo de até três horas. Este UGV, mostrado na Fig. 4.17 possui um minicomputador com suporte ao ROS e uma superfície rígida para montagem de sensores e manipuladores. Esta plataforma também pode ser controlada por acesso remoto via computador utilizando rede wi-fi ou através de um *Joystick* fornecido pelo fabricante.



(a) Visão frontal (b) Visão traseira

Figura 4.17: Plataforma robótica Husky A200 equipado com um sensor 2D-LiDAR LMS10x

Conforme mostra a Fig. 4.18, o Husky possui aproximadamente 1 m de comprimento, 67 cm de largura e 39 cm de altura, desconsiderando os sensores montados. As rodas possuem diâmetro de 33 cm com distância de aproximadamente 54 cm entre os eixos dianteiro e traseiro. Neste trabalho, também foi utilizado um modelo virtual do Husky A200 UGV equipado com um sensor LiDAR SICK LMS1xx (Fig. 4.19) para simulações em ambiente virtual.

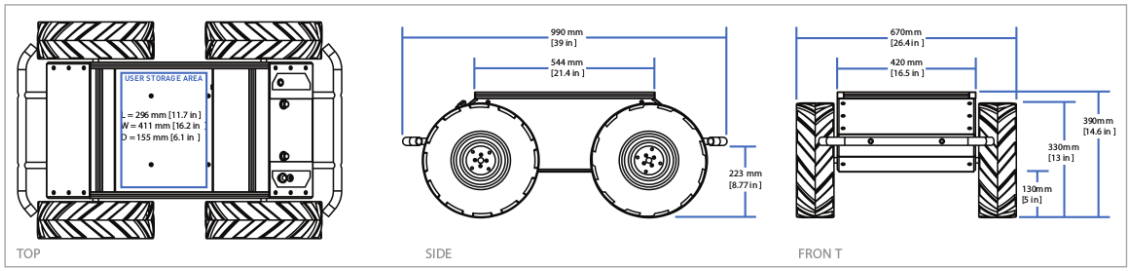
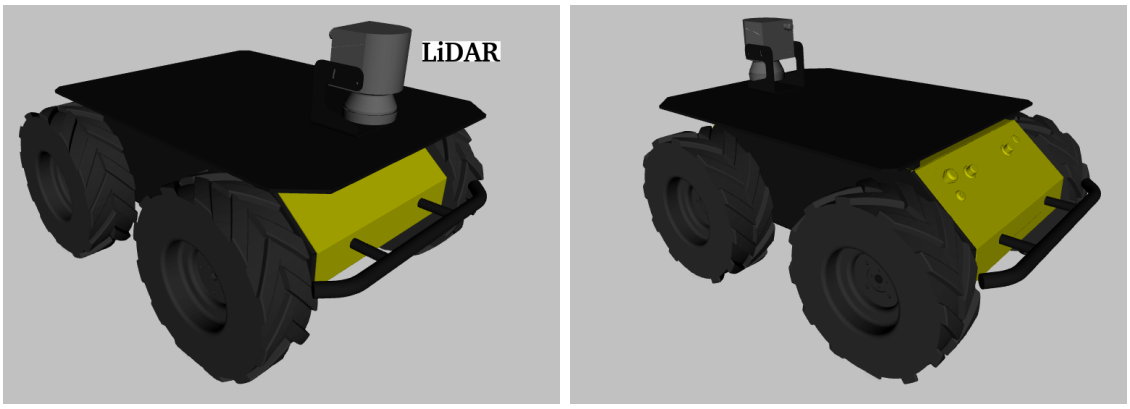


Figura 4.18: Da esquerda para a direita: vistas superior, lateral e frontal do modelo do Husky; imagem extraída do Manual do Fabricante [38]

[3] https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/



(a) Visão frontal (b) Visão traseira

Figura 4.19: Modelo virtual do Husky equipado com o sensor LiDAR SICK LMS10x.

No ambiente real, a plataforma Husky navega pelo LaR, apresentado nas subseções 4.3.3 e 4.3.5. No ambiente de simulação, o robô virtual navega em três ambientes virtuais que representam locais fechados e não estruturados, descritos na subseção 4.3.4.

4.3.3 Ensaios preliminares para a localização global

O objetivo dos ensaios preliminares foi analisar a viabilidade da utilização de uma das heurísticas de otimização, o PSO, juntamente com o PM na resolução do PLG. Para este fim, foi desenvolvido o algoritmo *Particle Swarm Localizer* (PSL), que é um protótipo do PSGL. A partir do sucesso do PSL foi possível aprimorar o método de localização global baseado em PSO, o PSGL, bem como desenvolver os demais métodos descritos neste trabalho baseados em inteligência de enxame e algoritmos evolucionários.

O PSL foi testado utilizando a plataforma Husky. O ambiente de ensaio foi o LaR, que possui aproximadamente 90 m^2 e está ocupado com mesas, cadeiras, armários e duas salas separadas. A grade de ocupação com $2,5$ cm de resolução (Fig. 4.20) foi construída anteriormente utilizando o algoritmo de mapeamento Hector SLAM [39] a partir dos dados de *scan* coletados durante a navegação do Husky.

Os experimentos com o PSL utilizaram dados *offline* capturados durante um percurso de 16 minutos dentro do laboratório e armazenados em um arquivo *bag*. Neste teste, as iterações do algoritmo de localização ocorrem sempre que novos dados de odometria são produzidos. Ou seja, o enxame de partículas evolui na mesma taxa de publicação de mensagens da odometria. Portanto, dado que o sistema de odometria do Husky publica mensagens em intervalos de 100 ms, a taxa de evolução do enxame foi de 10 Hz. O algoritmo executou em um computador com processador Intel Core

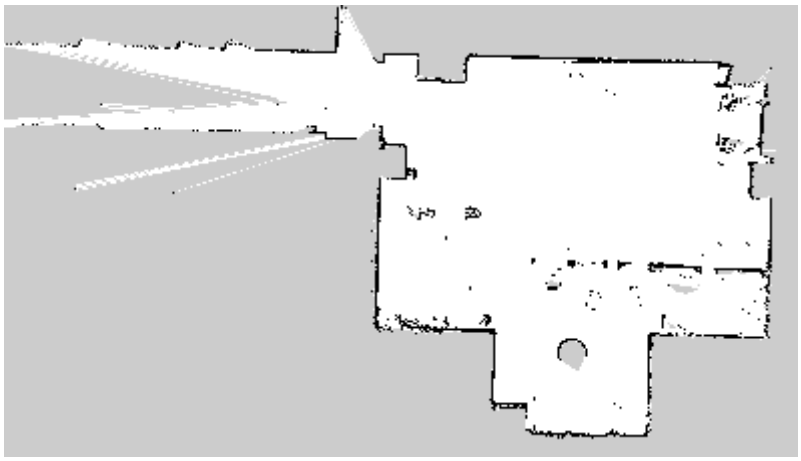


Figura 4.20: Grade de ocupação do LaR (em maio/2019), com células de $6,25$ cm^2. Mapa construído utilizando o Hector SLAM

i5 e 8 GB de memória RAM sem apresentar restrições de desempenho aparentes. No entanto, o uso do recurso computacional pode ser diminuído, se necessário, através da redução do número de partículas ou da acumulação de mensagens da odometria para cada iteração.

A Fig. 4.21 ilustra o sistema de teste proposto. Neste caso, o algoritmo PM é executado de forma concorrente, mas foi adaptado para auxiliar o PSL. O PM usa o mapa e a nuvem de pontos do *scan* para refinar a pose estimada, enquanto o PSL utiliza o mapa para gerar o enxame e as informações da odometria para atualizar a pose das partículas p_{best}. O PSL invoca o PM para obter os valores de *fitness* para cada partícula do enxame sempre que as poses destes são alteradas.

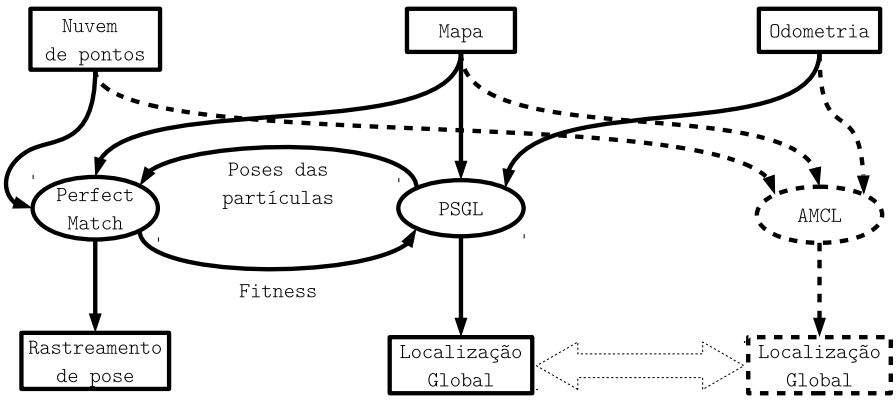


Figura 4.21: Diagrama do sistema de teste do PSL.

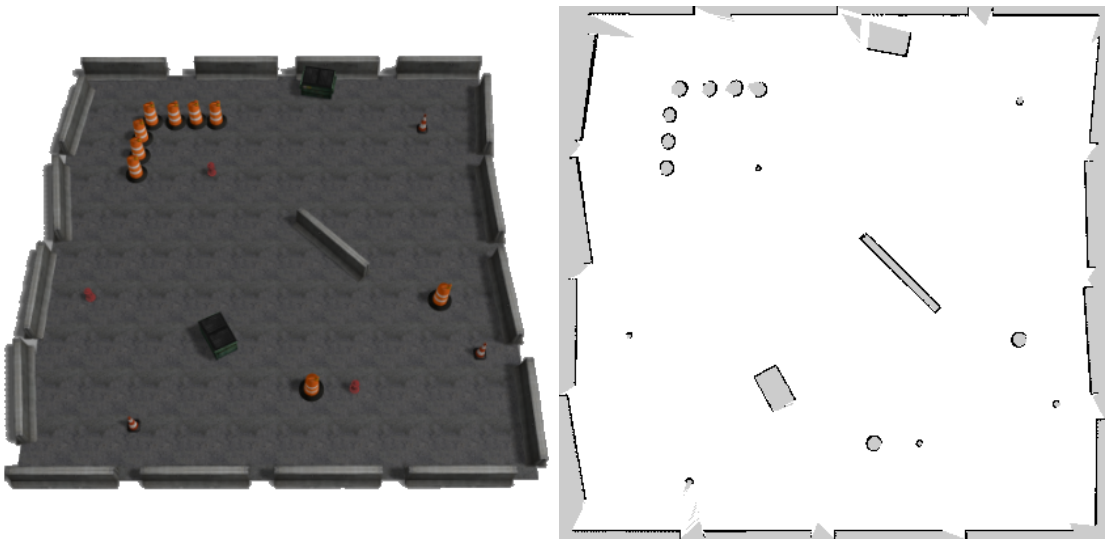
Além disso, a Fig. 4.21 mostra, utilizando linhas tracejadas, que o desempenho do PSL foi comparado com um algoritmo de localização global típico, o AMCL. Neste trabalho, foi utilizada a versão *Open Source* do AMCL [4]. Os resultados

[4]O *software* do AMCL está disponível sob licença Lesser General Public License (LGPL) em

destes ensaios preliminares são apresentados na seção 5.1.

4.3.4 Simulação em Ambiente Virtual

Para realização de simulações exaustivas com os métodos PSGL, PMGL, DEGL, GAGL e AMCL em ambientes virtuais, foram selecionados um ambiente pequeno com contornos acidentados e obstáculos dispersos (Playpen, Fig. 4.22a), um escritório típico de tamanho intermediário (CPR Office, Fig. 4.23a), e um espaço amplo e altamente simétrico, representando um galpão (Warehouse, Fig. 4.24a). Os dois primeiros ambientes são disponibilizados gratuitamente pela Clearpath Robotics[5] e o último foi desenvolvido especificamente para este trabalho. Os mapas dos respectivos ambientes possuem 5 cm de resolução e foram construídos automaticamente utilizando os algoritmos de mapeamento GMapping [54] (Figs. 4.22b e 4.23b) e Hector SLAM [39] (Fig. 4.24b) durante a navegação do Husky.



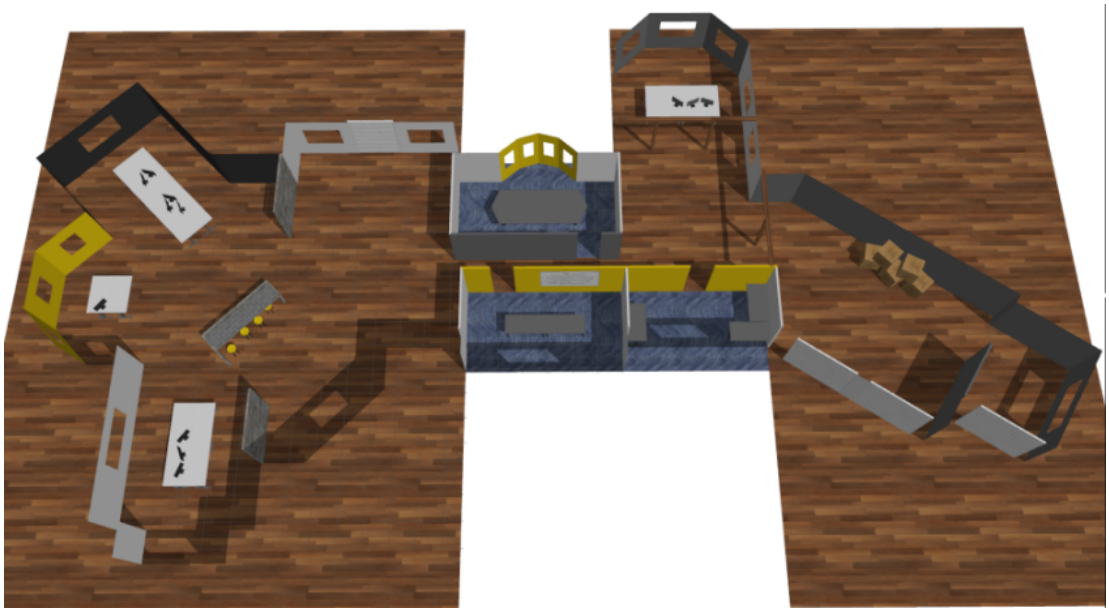
(a) Ambiente virtual (b) Grade de Ocupação

Figura 4.22: Ambiente virtual e grade de ocupação do Playpen (aproximadamente 20,3 m x 20,8 m)

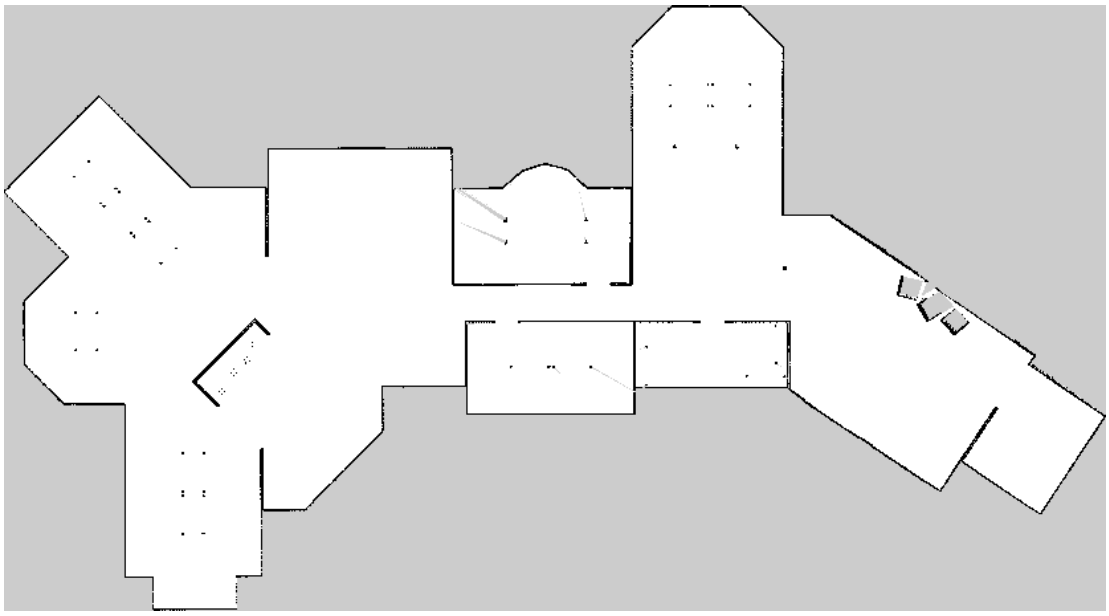
A simulação é dividida em duas etapas, conforme ilustra a Fig. 4.25: Geração de estímulos e Reprodução de estímulos. A primeira etapa consiste em um *framework* baseado em ROS e usa o *software* Gazebo como simulador do ambiente e do Husky. Durante esta etapa, o robô percorre todo o ambiente, visitando diferentes regiões do mapa em ordem aleatória, conduzido por um *software* que guia o movimento do Husky (*World explorer node*). O *ground truth* é publicado periodicamente pelo

`http://wiki.ros.org/amcl`

[5]Playpen: `https://github.com/husky/`; CPR Office: `https://github.com/clearpathrobotics/cpr_gazebo`.



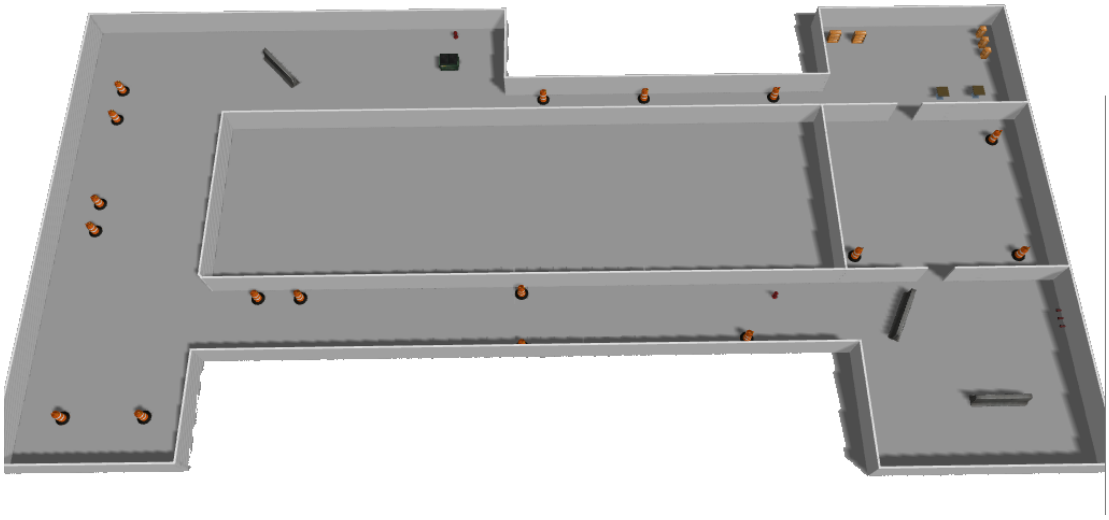
(a) Ambiente virtual



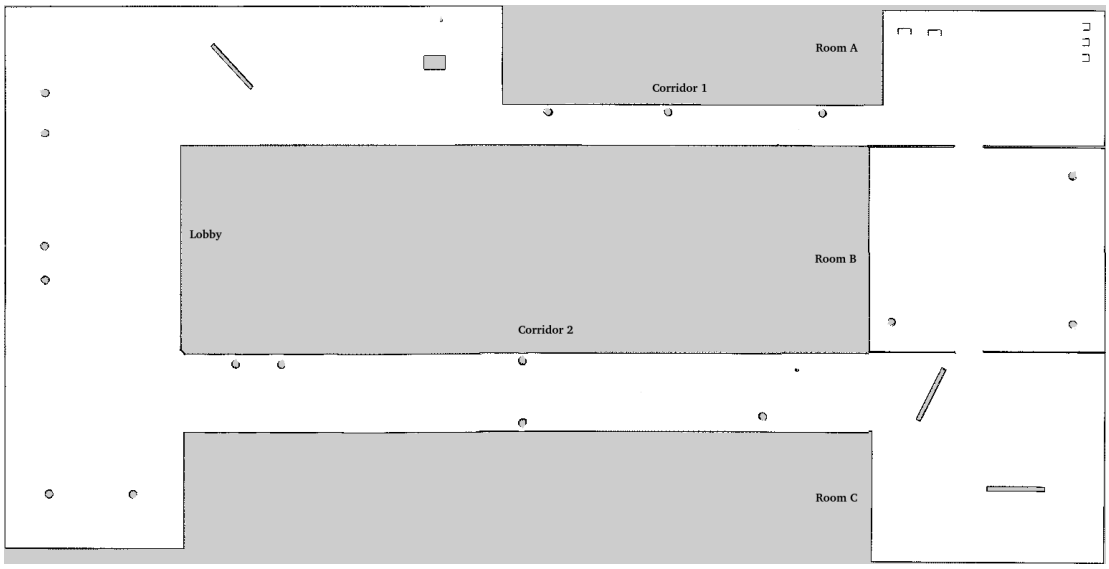
(b) Grade de Ocupação

Figura 4.23: Ambiente virtual e grade de ocupação do CPR Office (aproximadamente 22,8 m x 41,6 m)

Gazebo e esta informação é utilizada como a pose de referência. Simultaneamente, o mapa, o estado do robô e as leituras dos sensores são continuamente gravados em um arquivo *bag* para ser utilizado na segunda etapa (Fig. 4.25b). Na reprodução dos estímulos, o arquivo *bag* é reproduzido e um dos algoritmos sob teste é executado.



(a) Ambiente virtual

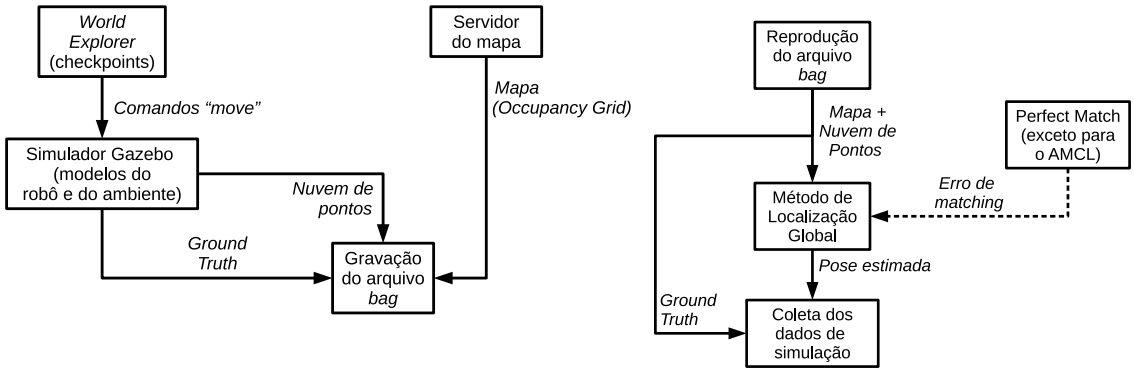


(b) Grade de Ocupação

Figura 4.24: Ambiente virtual e grade de ocupação do Warehouse (aproximadamente 38,9 m x 76,8 m)

4.3.5 Ensaios em ambiente real

Para testar os métodos de localização utilizando dados de experimentos reais, foram selecionados dois cenários: um percurso dentro do LaR e outro contido em um *dataset* público e extensamente utilizado por pesquisadores, o Intel Research Lab Dataset.



(a) Geração de estímulos (b) Reprodução de estímulos

Figura 4.25: Diagrama de blocos da simulação em duas etapas

Laboratório de Robótica da UFBA

O Laboratório de Robótica da Universidade Federal da Bahia (Fig. 4.26a) foi utilizado para realizar ensaios com a plataforma Husky. Este espaço possui aproximadamente 90 m^2, excluindo o corredor. O mapa do laboratório (Fig. 4.26b) possui 5 cm de resolução e foi construído utilizando o algoritmo Hector SLAM.

O Husky percorreu o LaR durante 12 minutos e os estímulos gerados foram salvos em um arquivo *bag*. A Fig. 4.27a mostra o sistema utilizado para capturar e armazenar os estímulos produzidos durante o experimento. Dado que não há *ground truth* neste ensaio real, a pose de referência é oferecida pelo algoritmo de rastreamento PM. Ainda assim, a consistência da pose de referência pôde ser facilmente atestada, comparando-se visualmente a trajetória observada durante a gravação do experimento, a projeção da nuvem de pontos do lidar sobre os contornos do mapa e o percurso representado pelo PM. Ressalta-se que esta validação foi realizada para todo o percurso do robô.

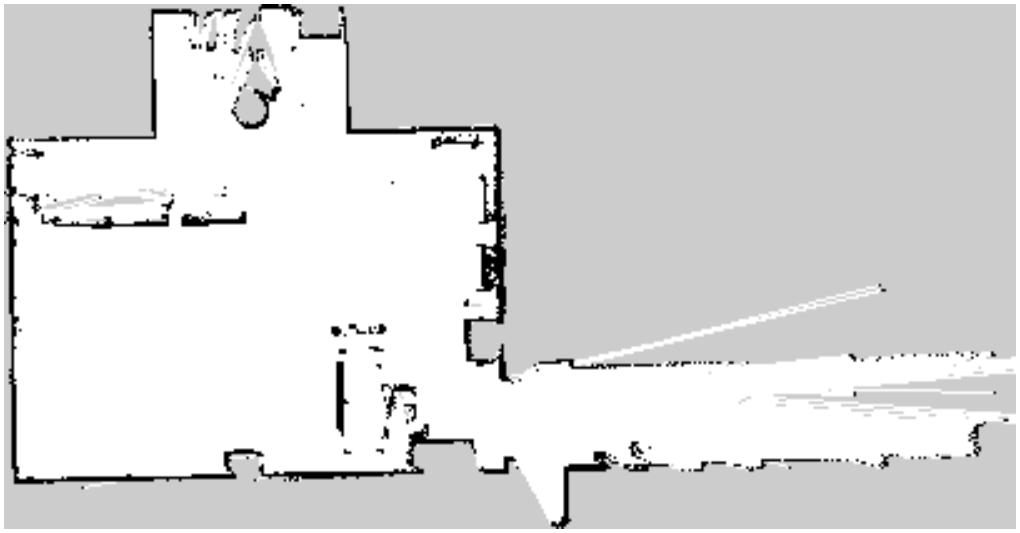
Considerando o reduzido número de *outliers*, a razoável fidelidade do mapa, do sensor LiDAR e da odometria do Husky, quaisquer desvio da pose do PM em relação à pose real estaria reduzida a poucos centímetros. Sendo assim, a pose estimada pelo PM pôde ser utilizada como um *ground truth*. Na etapa seguinte, cada algoritmo de localização global foi testado separadamente utilizando os estímulos armazenados no arquivo *bag* (Fig. 4.27b).

Intel Research Lab

O segundo cenário para ensaios com dados reais é baseado em estímulos reunidos em um banco de dados frequentemente utilizado na literatura [47, 49, 51, 52, 54, 80] chamando de Intel Research Lab Dataset. Este é um recurso de domínio público [6]

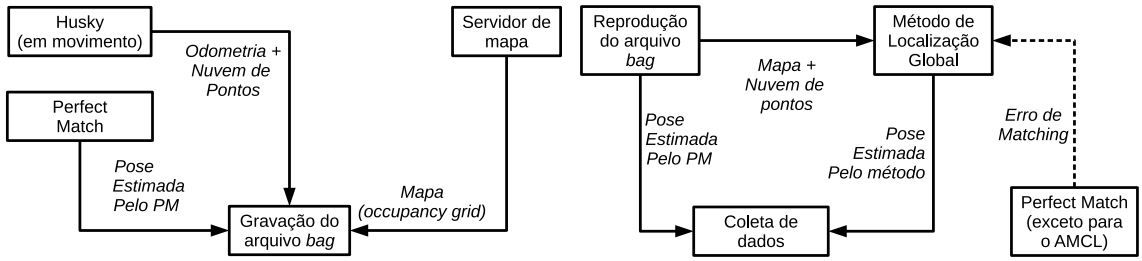
[6]Sob licença Creative Commons CC0 1.0 Universal

(a) Fotografia do Laboratório de Robótica



(b) Mapa do Laboratório de Robótica

Figura 4.26: Ambiente dos ensaios de localização global (LaR, em nov/2021). A fotografia da Fig. 4.26a, que mostra o Husky à esquerda. foi capturada do canto inferior esquerdo da Fig. 4.26b. As dimensões aproximadas da grade de ocupação são de 8,6 m x 18,4 m



(a) Geração dos estímulos (b) Reprodução dos estímulos

Figura 4.27: Diagrama de blocos do experimentos no mundo real em duas etapas

e é parte da coleção Robotics Data Set Repository (Radish) [90]. A fim de adaptar o Intel Research Lab Dataset para a estrutura de simulação desenvolvida neste trabalho, foi necessário converter o arquivo do *dataset* público para o formato *bag*.

O Intel Research Lab Dataset contém a odometria e nuvens de pontos do sensor a laser SICK LMS, que por sua vez tem abertura angular de 180°. Estes dados foram capturados de uma trajetória de 45 minutos ao longo de todo o ambiente, que possui aproximadamente 30×30 m. A grade de ocupação (Fig. 4.28), que possui cinco centímetros de resolução, também está disponível no repositório. Os experimentos com os métodos de localização global neste cenário foram realizados utilizando a mesma estrutura de simulação ilustrada pela Fig. 4.27b.

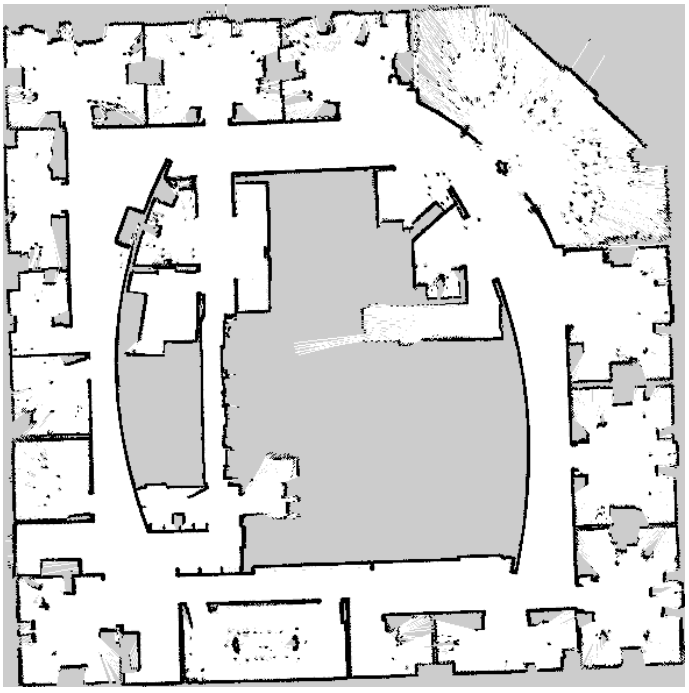


Figura 4.28: Grade de ocupação do Intel Research Lab

4.3.6 Definições para Localização Global

Um ponto sensível da análise de desempenho dos algoritmos é a definição do critério de sucesso do procedimento de localização global. Estimar a pose com erros linear e angular nulos é virtualmente impossível para o sistema de localização. Sendo assim, encontrar a pose correta consiste em estimar a localização com erros linear e angular pequenos. Portanto, os erros máximos de posição e orientação aceitáveis para a localização global, L_{dev} e A_{dev}, precisam ser estabelecidos, bem como número máximo de iterações para encontrar a pose correta, $i_{\max}$, conforme discutido a seguir.

Além disso, os parâmetros de deslocamento mínimo $d_{\min}$ e $a_{\min}$, que afetam a taxa de atualização dos métodos, e detalhes sobre a coleta dos resultados e a avaliação do custo computacional também estão descritos nesta subseção.

Parâmetros L_{dev} e A_{dev}

A escolha dos desvios máximos aceitáveis linear (L_{dev}) e angular (A_{dev}) para que a pose seja considerada correta confronta algumas divergências na literatura. Thrun [3, p. 238] afirma que células com tamanho de 15x15 cm e 5º de resolução na dimensão rotacional é uma configuração típica para localização baseada em grade. Isto implica que os algoritmos de localização podem apresentar desvios de até 15 cm e 5º mesmo quando estima corretamente a célula. Martín *et al.* [52] e Moreno *et al.* [47], por outro lado, utilizam uma tolerância ao erro linear de 50 cm mas não menciona a tolerância ao erro de rotação. Estudos [15, 20] mostraram que a robustez do PM permite que sua inicialização, ainda que com erros de alguns centímetros ou graus, possa convergir para uma melhor estimação.

Em outras palavras, se o procedimento de localização global fornecer uma pose levemente desviada, esta pode ser suficiente para iniciar o rastreamento da localização porque o RPROP é capaz de aproximar para um mínimo local. Portanto, deve-se conhecer os desvios iniciais (antes do RPROP) máximos em distância L_{dev} e rotação A_{dev} em relação ao *ground truth* no qual o PM garante uma convergência correta durante a sua inicialização. Neste trabalho, observando os valores utilizados na literatura, a definição de convergência correta é uma pose otimizada com menos de 10 cm e 5° de desvio em relação ao *ground truth*, mantida por pelo menos 5 iterações consecutivas. Desta forma, pode-se assumir que a localização global foi bem-sucedida para qualquer estimação com erros em posição e rotação menores que L_{dev} e A_{dev}, respectivamente.

Porém, a fim de encontrar os melhores valores para estes parâmetros de desvio em diferentes cenários, a inicialização do PM foi executada 10 mil vezes com erros adicionados intencionalmente variando entre 0 e 50 cm em distância e entre 0° e 90° em ângulo a partir de diversos pontos em cada um dos três ambientes virtuais.

Finalmente, considera-se que a inicialização do PM admite um desvio linear de no máximo L_{dev} e um desvio angular de até A_{dev} se o mesmo for capaz de convergir corretamente qualquer pose com erros menores que L_{dev} e A_{dev}, respectivamente. Os resultados coletados nesta simulação são apresentados no capítulo 5.

Confirmação de localização bem-sucedida

Para todos os algoritmos estudados foi estabelecido um número máximo de iterações para encontrar a pose correta, identificado como $i_{\max}$. Portanto, para este estudo, o algoritmo só obtém sucesso na localização global se encontrar a pose correta em uma iteração $i \leq i_{\max}$. Além da tolerância de desvios para a pose correta, L_{dev} e A_{dev}, a pose estimada deve ser sustentada por algumas iterações para que seja considerado o sucesso na localização global. Neste trabalho, foi definido de maneira empírica que o algoritmo de localização deve sustentar a pose correta por pelo menos 12 iterações, visto que isto compreende um intervalo onde o robô percorreu uma distância razoável, indicado que está ocorrendo o acompanhamento de sua pose correta pelo algoritmo.

As 12 iterações definidas correspondem a aproximadamente $2,4$ m de distância percorrida, caso esteja se movimentando em linha reta, ou 360° caso ele esteja girando. Portanto, se as poses estimadas entre as iterações $i - 12$ e $i < i_{\max}$ estiverem próximas da pose de referência (ou seja, dentro dos limites de L_{dev} e A_{dev}), considera-se que a localização global foi bem-sucedida na i-ésima iteração, chamada de i_{hit}.

Ressalta-se que a quantidade de iterações necessárias para a localização global pode variar de acordo com as características do ambiente e da própria abordagem do método. Portanto, o valor de $i_{\max}$ deve ser definido de acordo com a quantidade de iterações necessárias para os algoritmos estudados convergirem à pose correta em cada um dos cenários estudados. Sendo assim, simulações preliminares foram realizadas a fim de definir o valor de $i_{\max}$, conforme apresentado no capítulo 5.

Parâmetros $d_{\min}$ e $a_{\min}$

A subseção 4.1.1 mostrou que as partículas dos métodos desenvolvidos atualizam suas poses apenas quando um deslocamento mínimo é realizado. Portanto, a taxa de atualização da população depende das velocidades linear e angular do robô. A velocidade do Husky é controlada por um sistema autônomo de navegação (`move_base` na Fig. 4.16) no meio virtual e por um *joystick* no ambiente real. Em ensaios preliminares, observou-se que o Husky sempre se move com velocidades linear e angular limitadas a $|v_k^o| \leq 0,5$ m/s e $|\omega_k^o| \leq 34,4$ °/s, respectivamente. Portanto, foram estabelecidos os valores $d_{\min} = 20$ cm e $a_{\min} = 30°$, o que dá aos algoritmos de

localização global uma taxa de atualização de até $2,5$ Hz quando o Husky se move em linha reta e até $1,15$ Hz quando está girando.

Coleta dos resultados

Os dados dos ensaios são coletados pelo nó `Data Capturer` nas Figs. 4.25b e 4.27b. Este nó captura a pose de referência e a pose estimada pelo método de localização global em cada iteração do método. Além disso, também são capturados o carimbo de tempo da pose publicada e o tempo gasto pelo método em uma iteração, para fins de avaliação do custo computacional. O `Data Capturer` ainda registra qual o número da tentativa de localização global (onde cada tentativa possui 200 iterações) e calcula os erros linear e angular a partir das poses capturadas. Todas as informações produzidas são armazenadas em um arquivo de texto com o formato semelhante à tabela 4.1.

Tabela 4.1: Formato do arquivo de análise da localização global com parte dos dados coletados durante um dos ensaios

Carimbo de tempo	nº da tentativa	*Ground truth*			Pose estimada			Iteração	Erro		Tempo da iteração
		x	y	θ	x	y	θ		linear	angular	
1338,028	1	-2,122	-8,875	6,147	-4,022	-3,515	1,682	1	5,687	1,323	15,162
1342,108	1	-2,150	-8,795	5,564	2,016	8,183	1,168	2	17,481	1,888	6,760
1346,909	1	-2,212	-8,710	0,996	-8,853	-6,237	4,413	3	7,086	2,867	3,356
1349,906	1	-2,202	-7,593	2,026	-9,270	-5,159	6,145	4	7,475	2,164	2,985
1351,918	1	-2,576	-7,127	2,363	-2,148	-7,652	1,919	5	0,677	0,445	2,012
1353,227	1	-2,840	-6,848	2,282	6,911	-2,258	3,973	6	10,778	1,691	1,308
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1472,205	1	-4,578	-3,537	3,656	-4,500	-3,490	3,652	198	0,091	0,004	0,187
1472,815	1	-4,789	-3,656	3,652	-4,672	-3,585	3,647	199	0,137	0,005	0,154
1473,416	1	-4,935	-3,736	3,661	-4,861	-3,685	3,657	200	0,090	0,004	0,151
1492,450	2	-4,315	-3,396	0,593	4,192	4,605	4,686	1	11,679	2,190	18,666
1501,508	2	-0,339	-2,642	0,078	-11,430	-2,571	0,745	2	11,091	0,667	9,058
1505,028	2	1,420	-2,618	0,007	-7,654	2,559	3,215	3	10,447	3,075	3,520
1507,306	2	2,568	-2,625	6,218	-7,522	2,944	3,045	4	11,524	3,110	2,278
1509,258	2	3,458	-2,598	0,125	7,954	5,086	5,309	5	8,902	1,099	1,952
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Análise do custo computacional

Conforme mostra a Tabela 4.1, o sistema de ensaio coleta a duração de cada iteração do método de localização global. Esta informação é utilizada para calcular o custo computacional aproximado de cada método. É válido ressaltar que as primeiras iterações de uma tentativa de localização são mais longas que as últimas, visto que o número de hipóteses na população inicial é maior e reduz ao longo das iterações. Uma vez que os dados coletados contem o tempo de cada iteração e que os ensaios consistem em centenas de iterações ao longo de dezenas de tentativas de localização, os resultados apresentados no capítulo seguinte mostram a média de tempo de iteração para diferentes números de iteração e separados por tamanho de população inicial ($P_{\max}$).

4.3.7 Ensaios de avaliação da incerteza sobre a pose

O método de estimação da incerteza sobre a pose PMCov foi comparado ao PM, na versão apresentada por Sobreira *et al*, [14] e ao método de localização bem conhecido na literatura, o AMCL. O *software* do PM foi utilizado sob autorização dos autores e o AMCL é uma implementação de código aberto disponível na internet[7].

O rastreamento da pose foi analisado em diferentes locais do DEEC/UFBA, representado pelo mapa da Fig. 4.11. Os dados produzidos para o ensaio contendo odometria e nuvens de pontos do sensor LiDAR foram capturados durante um percurso do saguão até o LaR realizado pelo Husky. Este *dataset* inclui dois cenários estratégicos: o caminho dentro do longo corredor e o percurso dentro de um espaço com mobília e *outliers* (como mostra a Fig. 4.26a).

Para realizar a avaliação, foram selecionadas cinco situações diferentes durante o percurso do robô para investigar a estimação da incerteza. Duas situações representam uma condição típica, onde o método de rastreamento mantém a pose muito próxima da correta durante um longo período e outras três situações que constituem o problema do sequestro, ou seja, o método de rastreamento mantém uma pose distante da correta. O rastreamento da pose dentro do LaR foi analisado em pelo menos uma das condições descritas anteriormente. Os detalhes de cada situação estão descritas na seção 5.3.

[7] `https://wiki.ros.org/amcl`

Capítulo 5

Resultados

Neste capítulo são apresentados os resultados obtidos com os ensaios preliminares (seção 5.1), experimentos exaustivos dos métodos de localização global (seção 5.2) e ensaios para avaliação da estimação da incerteza sobre a pose (seção 5.3).

5.1 Ensaios Preliminares

As Figs. 5.1 e 5.2 mostram o algoritmo de localização global PSL executado dentro do LaR com 1000 partículas. Cada figura contém nove quadros que capturam as iterações 1,3 e 5 na linha superior, 10, 20 e 30 na linha do meio e 50, 100 e 150 na última linha. A seta em azul escuro representa a pose estimada pelo PSL e a linha vermelha é o *ground truth* obtido anteriormente utilizando o PM. Este algoritmo de localização foi escolhido para determinação do *ground truth* porque observou-se em simulação com o modelo virtual do Husky que o mesmo possuía menor erro, em comparação com o AMCL. Observou-se que em ambos os experimentos ilustrados nas figuras, o algoritmo PSL encontrou a pose correta em menos de 10 iterações. Em ambas as figuras, a seta em azul escuro coincide com a seta vermelha a partir da décima iteração.

A Fig. 5.2 mostra um experimento em especial que foi realizado para testar a robustez do PSL quando os contornos do ambiente não estão bem alinhados em relação à grade de ocupação. Neste caso, uma prancha de madeira de aproximadamente 150×150 cm foi posicionada a 40 cm da parede superior do laboratório, conforme indicado pelo rótulo "BOARD" no canto superior esquerdo da figura. A nuvem de pontos do sensor LiDAR está exibida em amarelo e os contornos da prancha são detectados por este sensor conforme apresentado na parte superior do mapa. Neste experimento, a localização global inicia quando o veículo está posicionado próximo à prancha. Conforme mostra a Fig. 5.2, o PSL também encontra a pose em menos de 10 iterações, considerando um enxame com 1000 partículas.

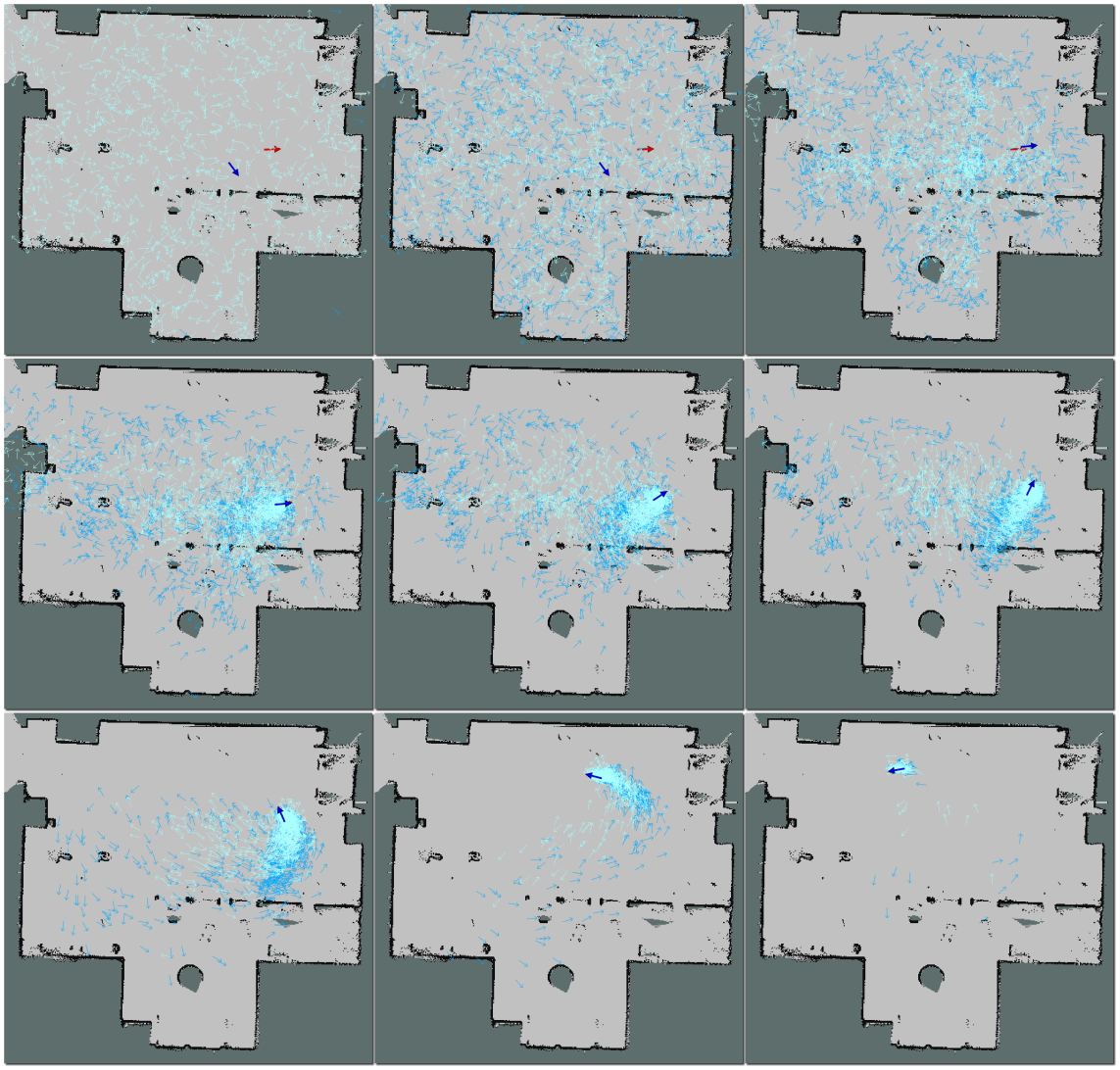


Figura 5.1: Do canto superior esquerdo ao inferior direito: iterações 1, 3, 5, 10, 20, 30, 50, 100 e 150 do PSGL. O enxame de partículas está em azul claro, e os P_{best}'s em azul médio. A seta em azul escuro representa a pose G_{best}, enquanto a seta vermelha é o *ground truth*

Outros ensaios foram realizados neste mesmo ambiente, desta vez sem a presença da prancha, com o objetivo de comparar o desempenho do PSL em relação ao AMCL. Para realizar uma comparação direta, ambos os algoritmos foram executados de em paralelo com o mesmo tamanho de população (1000 partículas). Porém, devido à detalhes de implementação já descritos na seção 4.1, as partículas do AMCL são atualizadas apenas quando ocorre um deslocamento mínimo linear (d_{min}) ou angular (a_{min}). Então, em um primeiro teste, d_{min} e a_{min} foram configurados para 1 cm e $0,01$ radianos, respectivamente. O erro de estimação de cada algoritmo é determinado pela distância euclidiana entre a pose estimada e o *ground truth* e foi capturado em quatro tentativas de localização global diferentes, conforme mostra a

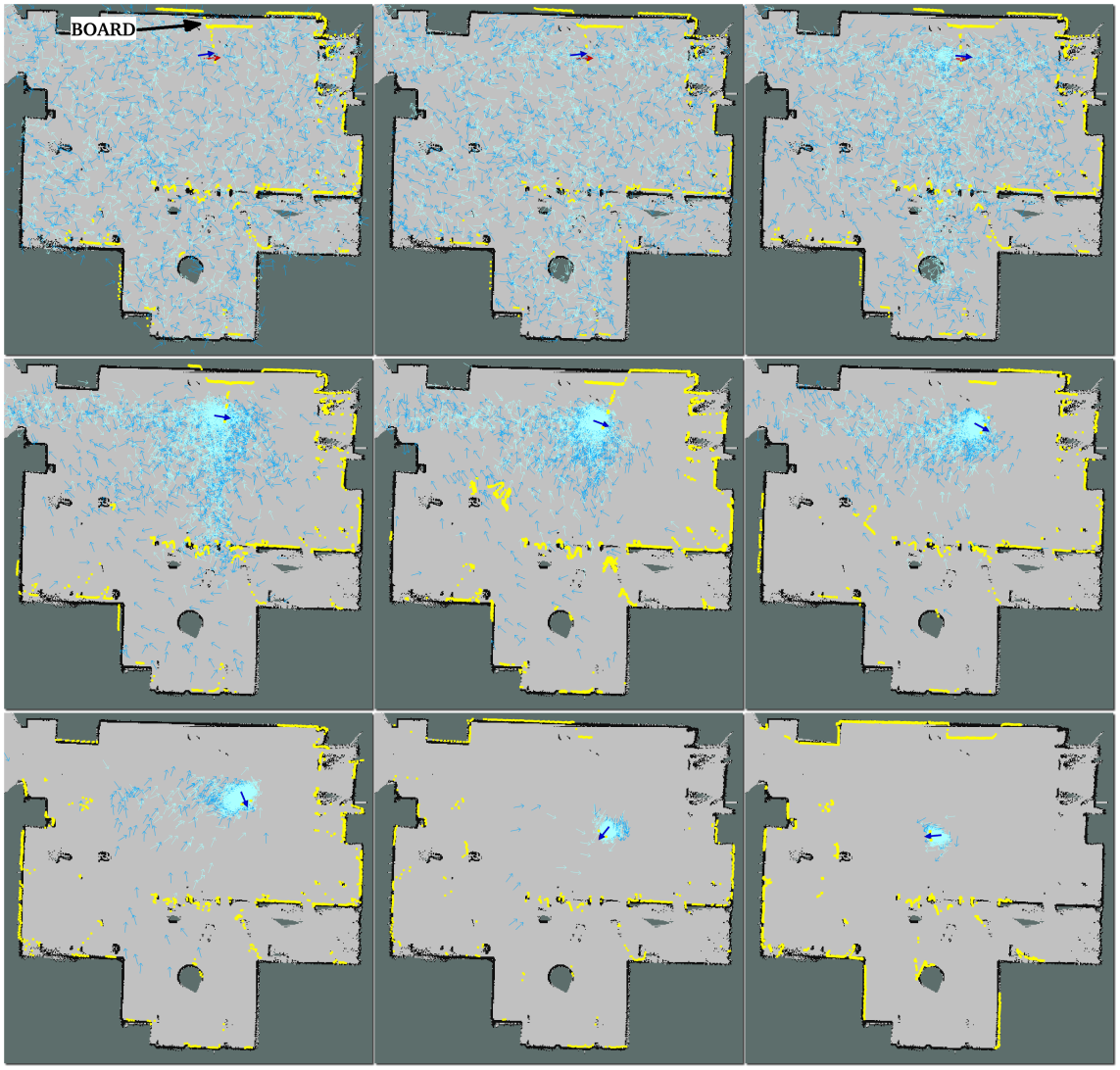
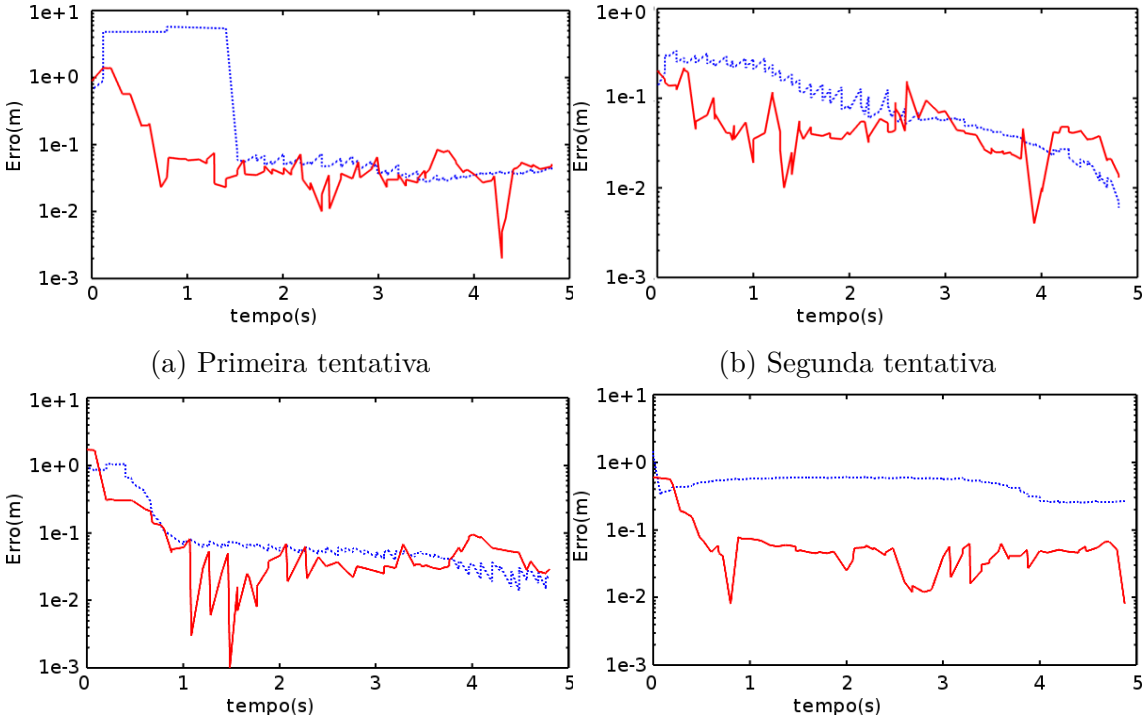


Figura 5.2: PSGL em cenário com contornos imprevistos

Fig. 5.3, cujo eixo y está em escala logarítmica para melhor visualização. Os dados coletados compreendem 5 segundos de execução a partir do início da localização global e se estendem por 50 iterações do PSL.

O gráfico da Fig. 5.3 mostra que o PSL converge para uma boa estimativa antes do AMCL, predominantemente no primeiro segundo da localização global. Porém, ambos os algoritmos mantém um nível de erro compatível a medida que as partículas se aproximam do mínimo global.

Em outro teste, o AMCL foi configurado com parâmetros típicos: $d_{min} = 20$ cm e $a_{min} = 0,2$ radianos. Os resultados estão apresentados na Fig. 5.4, que reproduz o erro da pose estimada em escala logarítmica e compreendem aproximadamente 25 segundos de localização global e 200 iterações do PSL. O gráfico sugere que o PSL apresenta um desempenho melhor quando AMCL possui configurações padrão e o mesmo tamanho de população inicial.



(a) Primeira tentativa

(b) Segunda tentativa

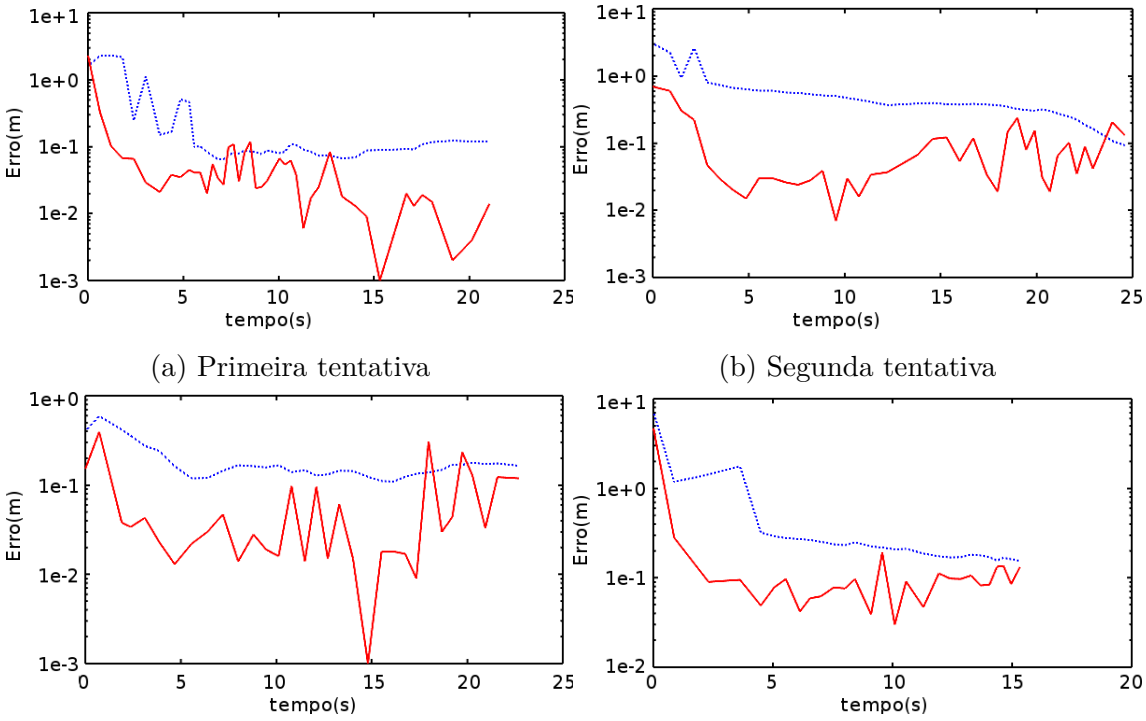
(c) Terceira tentativa

(d) Quarta tentativa

Figura 5.3: Erro de posição estimada em metros (eixo y em escala logarítmica) do PSL (linha sólida em vermelho) e AMCL (linha pontilhada em azul). AMCL está configurado para uma taxa de atualização alta. o eixo x representa o tempo decorrido desde o início da localização global, em segundos.

O PSL também foi testado para diferentes tamanhos de população (P_{size}). A Tabela 5.1 mostra diferentes valores de P_{size} e o número de tentativas de localização global bem sucedidas – o que significa que PSL encontrou uma pose estimada próxima do *ground truth*, e o número de falhas – quando o algoritmo não pôde estimar uma pose aceitável nas primeiras 200 iterações. A taxa de sucesso, que indica o desempenho do método, é mostrada na última coluna da tabela.

O tamanho do ambiente e a resolução do mapa são fatores sensíveis para o sistema de localização, portanto o número de partículas deve ser proporcional ao tamanho do espaço de busca. O mapa da Fig. 2.14 possui aproximadamente 90 m^2, onde 68 m^2 constitui espaço livre, ou seja, células que podem ser visitadas. O gráfico da Fig. 5.5 mostra a relação entre a densidade da população (eixo x) e a porcentagem de localizações bem sucedidas (eixo y). Portanto nos experimentos realizados, observa-se que o PSL obteve uma baixa taxa de sucesso quando a densidade da população foi menor que aproximadamente cinco partículas por metro quadrado.



(a) Primeira tentativa

(b) Segunda tentativa

(c) Terceira tentativa

(d) Quarta tentativa

Figura 5.4: Erro de posição estimada em metros (eixo y em escala logarítmica) do PSL (linha sólida em vermelho) e AMCL (linha pontilhada em azul). O AMCL está configurado para uma taxa de atualização padrão. o eixo x representa o tempo decorrido desde o início da localização global, em segundos.

5.2 Experimentos de localização global

Esta seção descreve os resultados dos experimentos realizados para avaliar o desempenho dos métodos de localização global propostos. Conforme mencionado na seção 4.3, o critério de sucesso da localização global depende dos parâmetros L_{dev}, A_{dev} e $i_{\max}$, que devem ser escolhidos de acordo com as características dos métodos e do PM. Portanto, as subseções 5.2.1 e 5.2.2 descrevem os experimentos realizados para determinar os valores destes parâmetros. Em seguida, os resultados dos ensaios e a análise de custo computacional são apresentados nas subseções 5.2.3 e 5.2.6, respectivamente.

5.2.1 Determinação dos parâmetros L_{dev} e A_{dev}

Os gráficos da Fig. 5.6 foram obtidos de um experimento realizado no mapa Playpen que mostra que, dentro das 100 primeiras iterações, o PM pode convergir ao *ground thruth* (reduzindo os erros linear e angular) ou divergir, a partir de uma pose inicial fornecida com erros fixos err_x e err_y nas coordenadas x e y, respectivamente, e err_θ na orientação. Enquanto o Husky navegava, um grande número de inicializações do

Tabela 5.1: Eficácia do PSL por tamanho de população.

P_{size}	N° de tentativas	Bem sucedidas	Mal sucedidas	Taxa de sucesso (%)
10	40	9	31	22,5
15	39	13	26	33,3
25	41	24	17	58,5
50	40	21	19	52,5
100	42	28	14	66,7
200	42	33	9	78,6
400	42	37	5	88,1
600	42	37	5	88,1
800	45	41	4	91,1
1000	43	40	3	93,0

PM foi realizada, atribuindo poses iniciais com erros: para o experimento das Figs. 5.6a e 5.6b, foram atribuídos $err_x = err_y = \pm 0,7$ m e $err_\theta = \pi/2$ radianos (90°), e nos demais casos (Figs. 5.6c e 5.6d), $err_x = err_y = \pm 0,5$ m e $err_\theta = 0,4$ radianos ($\sim 22°$). O erro linear inicial é dado pelo módulo do vetor:

$$|(err_x, err_y)| = \sqrt{{err_x}^2 + {err_y}^2} \ . \tag{5.1}$$

A Fig. 5.6 mostra que, com erro inicial linear de aproximadamente 1 m e angular de 90°, o PM dificilmente conseguiu convergir, enquanto que com erros próximos de 0,7 m e 22°, o algoritmo falhou apenas uma vez no ambiente Playpen. Em um segundo experimento, um número maior de inicializações com erros iniciais variáveis foi obtido, também durante a navegação do Husky, em três mapas diferentes.

A Fig. 5.7 exibe o erro linear inicial (eixo x) e o erro angular inicial (eixo y) para cada uma das 10000 inicializações, conforme descrito na subseção 4.3.6. As inicializações bem-sucedidas estão representadas por pontos azuis e marcam as condições em que o PM convergiu corretamente. As cruzes vermelhas representam inicializações mal-sucedidas, ou seja, nestas condições o PM não conseguiu aproximar a pose inicial para o *ground truth*.

Além disso, a Fig. 5.7 também exibe retângulos em preto que delimitam as regiões, ou condições, nas quais as inicializações do PM foram sempre bem-sucedidas. Estas regiões sugerem que o PM é capaz de tolerar erros de até 31 cm e 31° no mapa Playpen (Fig. 5.7a), 32 cm e 21° no mapa CPR Office (Fig. 5.7b) e 12 cm e 22° no mapa Warehouse (Fig. 5.7c). Foram escolhidos estes três ambientes, que dispõem do *ground truth*, para que seja possível validar a análise da robustez do PM em pelo menos três condições distintas. Portanto, deduz-se que é razoável definir $L_{\text{dev}} = 12$

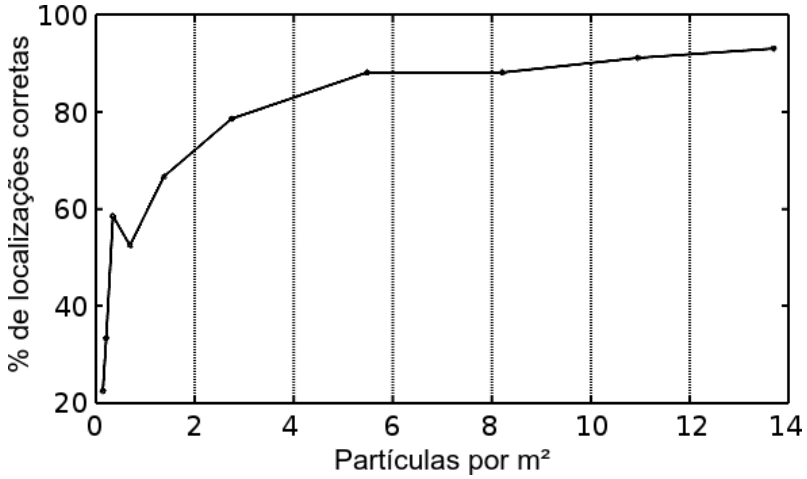


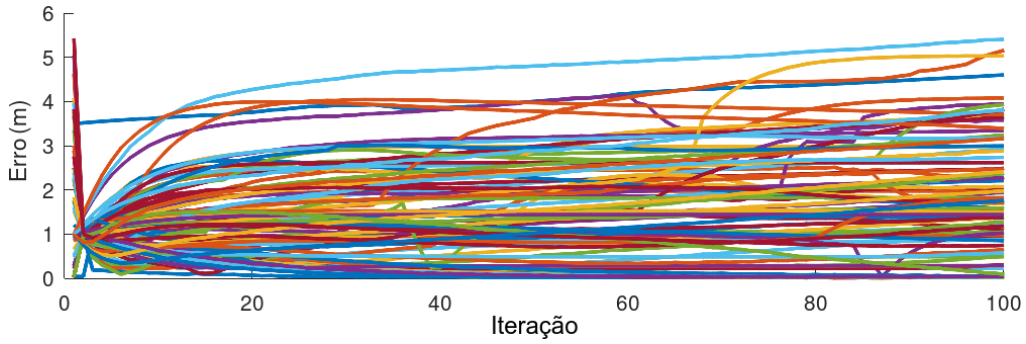
Figura 5.5: Proporção de localização global bem-sucedida *versus* número de partículas por m^2.

cm e $A_{\text{dev}} = 20°$ como valores limite para os erros de estimação da localização global. Em outras palavras, a localização global é classificada como bem-sucedida somente se a pose estimada possuir erros menores que 12 cm e 20°.

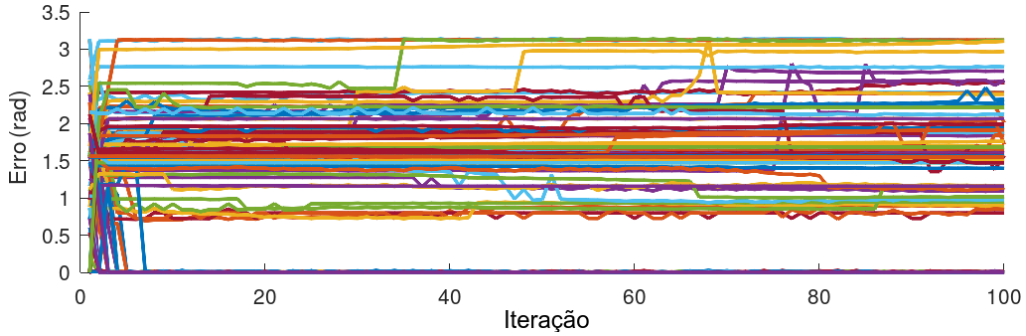
5.2.2 Determinação do parâmetro $i_{\max}$

Para definir o número máximo de iterações ($i_{\max}$) para a realização da localização global, foram realizadas simulações preliminares com os algoritmos propostos. Nestas simulações, foi observado que as localizações bem-sucedidas normalmente ocorrem antes da 200ª iteração, conforme observado na Fig. 5.8 cujos dados foram coletados em simulações nos ambientes Playpen, CPR Office e Warehouse utilizando diferentes tamanhos de população inicial ($P_{\text{init}} \in [500, 100, 1500, ..., 5000]$).

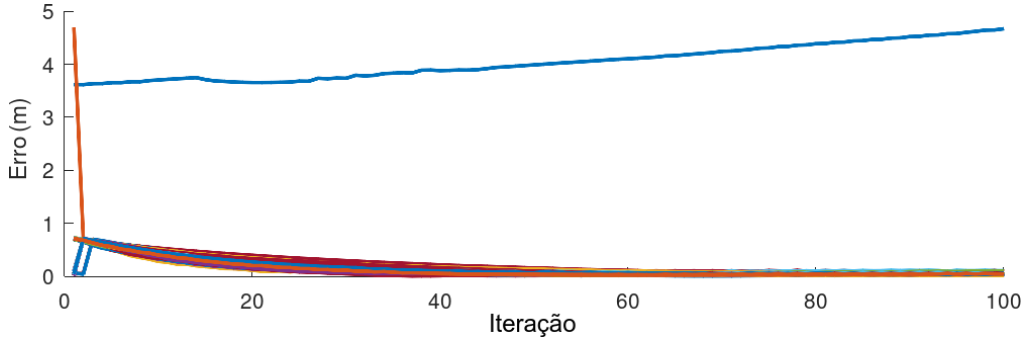
Cada *bin* do histograma da Fig. 5.8 agrega 10 iterações i_{hit} e representa a frequência destas dentro do universo de tentativas realizadas na simulação. No ambiente Playpen (Fig. 5.8a), a maioria dos métodos convergem antes da 130ª iteração, com exceção do DEGL que apresentou um tempo de convergência mais disperso. No ambiente CPR Office (Fig. 5.8b), as convergências para a pose correta se concentraram antes da centésima iteração, com exceção, mais uma vez, do DEGL que concentrou a frequência de i_{hit} entre aproximadamente 100 e 180. Finalmente, no ambiente Warehouse (Fig. 5.8c), a latência de convergência para a pose correta se concentrou em diferentes regiões para cada método. Porém, observa-se que a partir da 170ª iteração, os *bins* do histograma estão em declínio. Portanto, a partir destas observações, foi definido que todos os algoritmos têm até 200 iterações para realizar a localização, ou seja, $i_{\max} = 200$.



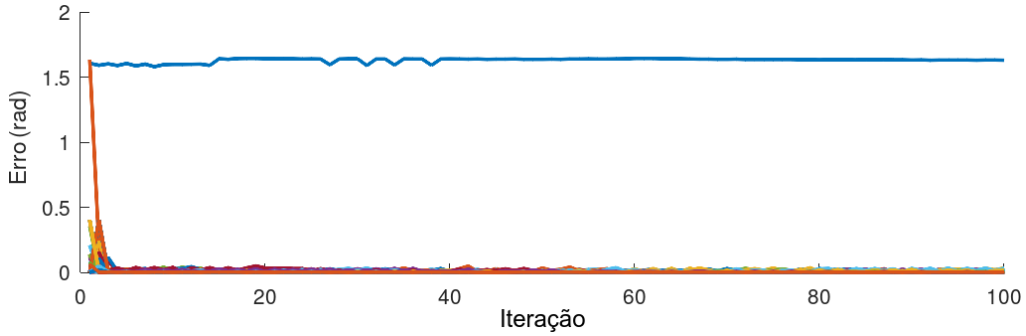
(a) Erro linear ($err_x = err_y = \pm 0,7$, $err_\theta = \frac{\pi}{2}$)



(b) Erro angular ($err_x = err_y = \pm 0,7$, $err_\theta = \frac{\pi}{2}$)

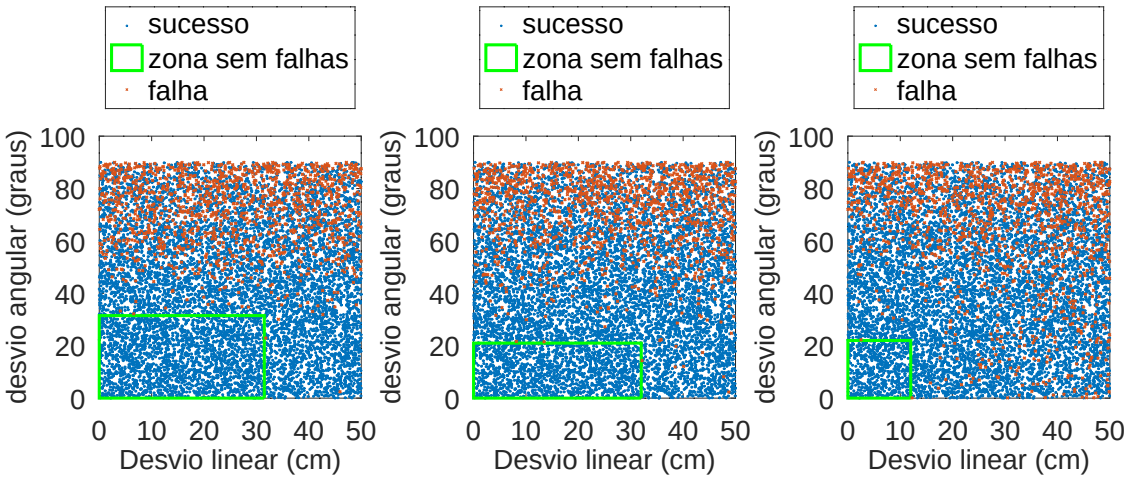


(c) Erro linear ($err_x = err_y = \pm 0,5$, $err_\theta = 0,4$)



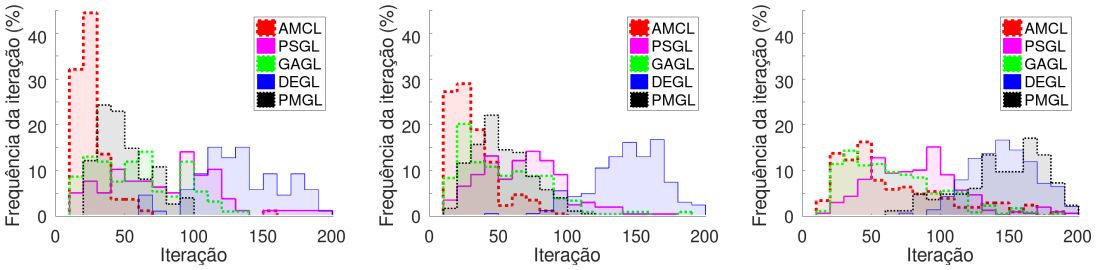
(d) Erro angular ($err_x = err_y = \pm 0,5$, $err_\theta = 0,4$)

Figura 5.6: Magnitude dos erros linear e angular (eixo y) da pose estimada pelo PM durante sua inicialização. O eixo x indica a iteração do PM. Erros linear e angular exibidos em metros e radianos, respectivamente. Cada série traçada por cores distintas representa uma inicialização do PM em uma pose diferente



(a) Playpen (b) CPR Office (c) Warehouse

Figura 5.7: Robustez da inicialização do Perfect Match para diferentes erros iniciais linear (eixo x, em metros) e angular (eixo y, em graus). Os pontos em azul representam inicializações bem sucedidas, enquanto as cruzes em vermelho indicam falha



(a) Playpen (b) CPR Office (c) Warehouse

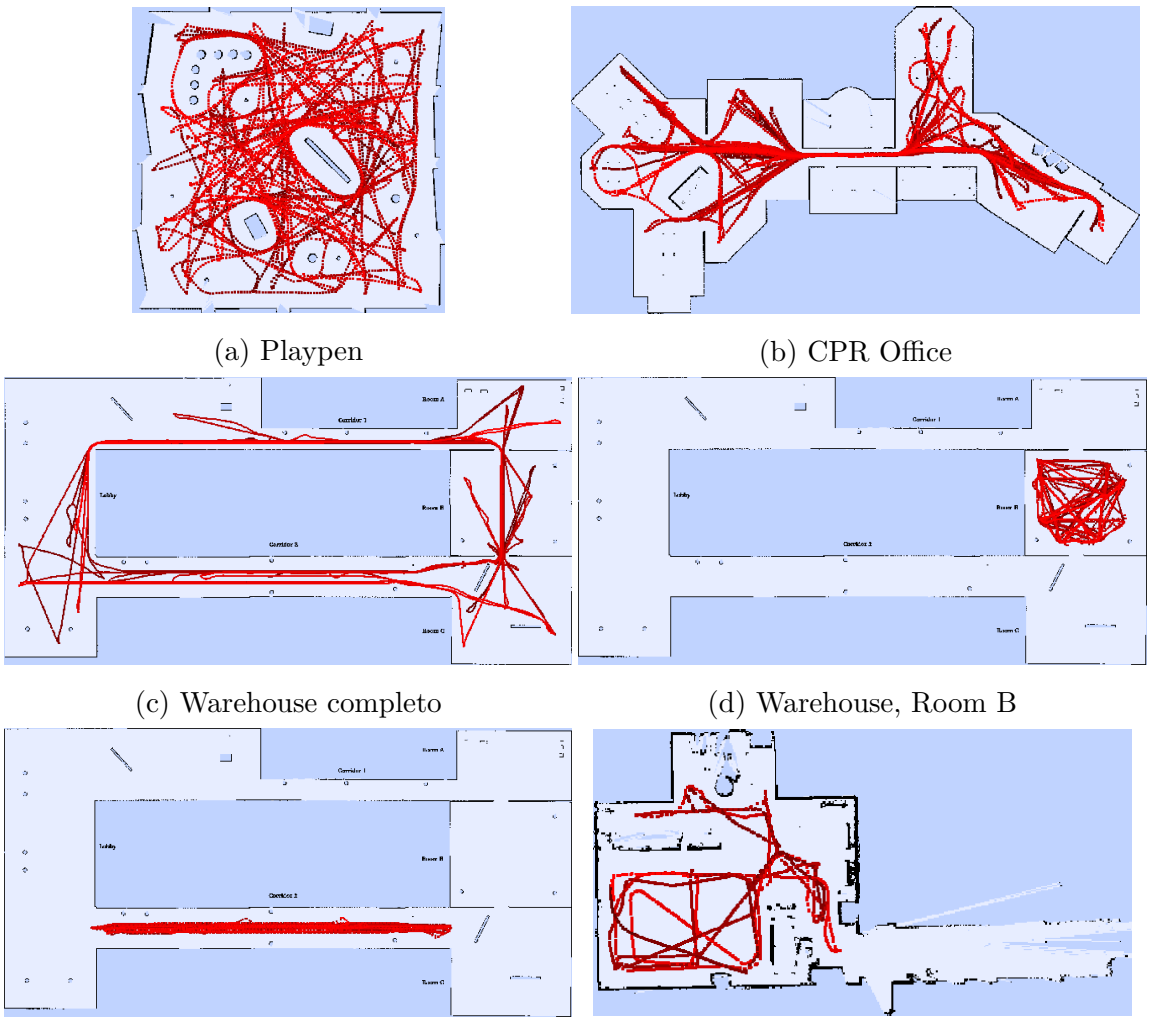
Figura 5.8: Histogramas que representam a frequência do i_{hit} – a iteração a partir da qual o método de localização global mantém a pose estimada correta

5.2.3 Análise dos ensaios

Durante a simulação, o Husky percorreu três ambientes virtuais, visitando localizações determinadas aleatoriamente, como mostra a Fig. 5.9. A trajetória do robô está desenhada pelos pontos em vermelho que representam sucessivas posições assumidas pelo robô durante seu percurso.

No LaR, a trajetória foi guiada por controle remoto. Porém, para permitir que o percurso seja longo sem depender de um ensaio demorado, foi realizada uma trajetória curta em ciclo (aproximadamente 120 m), de forma que a pose inicial e final coincidem. Desta forma, a trajetória completa na reprodução dos estímulos consistiu na repetição do ciclo realizado no ensaio, em até 15 vezes, para cumprir com todas as tentativas de localização global.

Nos ambientes Playpen e CPR Office (Figs. 5.9a e 5.9b), o robô navegou por todo o mapa. Em Warehouse, foram realizados três percursos diferentes: por todo o mapa (Fig. 5.9c), apenas em Room B (Fig. 5.9d), e apenas em Corridor 2 (Fig. 5.9e). A tabela 5.2 mostra a distância percorrida em cada um dos casos acima.



(a) Playpen

(b) CPR Office

(c) Warehouse completo

(d) Warehouse, Room B

(e) Warehouse, Corridor 2

(f) Laboratório de Robótica

Figura 5.9: Trajetória do Husky em cada mapa. Pontos em vermelho constituem posições *ground truth* registradas

Foram analisadas as taxas de sucesso de cada algoritmo em termos de proporção de localizações bem-sucedidas em relação ao total de tentativas realizadas. O tamanho da população inicial (P_{init}) variou de 500 até 5000 em intervalos de 500, e o tamanho mínimo (P_{min}) foi fixado em 100. O Apêndice A ilustra como as populações são aleatoriamente inicializadas para diferentes mapas e tamanhos. Conforme citado anteriormente, o sucesso depende da avaliação do erro da pose estimada, que deve estar dentro dos limites estabelecidos por L_{dev} e A_{dev}. Os erros de cada método ao longo das iterações para diferentes populações estão apresentados no Apêndice B. Os desempenhos de cada método estão apresentados na Fig. 5.10.

Tabela 5.2: Distância percorrida pelo Husky em cada ambiente (em metros)

Ambiente	Distância (m)	Ambiente	Distância (m)
Playpen	2118	Warehouse, Room B	1156
CPR Office	1676	Warehouse, Corridor 2	2004
Warehouse completo	1892	Laboratório de Robótica	2002

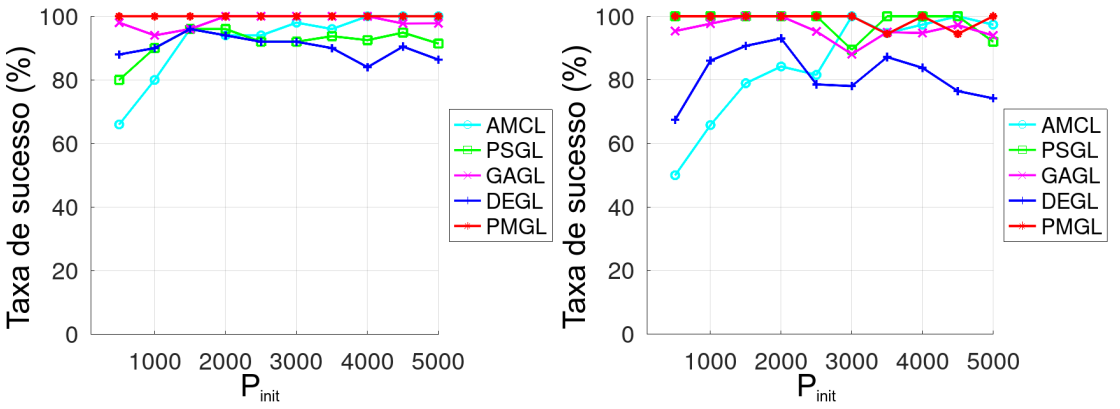
Todos os algoritmos demonstraram bom desempenho em Playpen, conforme observado na Fig. 5.10a. Não houve falhas em tentativas do PMGL com qualquer valor de P_{init}, do GAGL com $2000 \leq P_{\text{init}} \leq 4000$ e AMCL com $P_{\text{init}} \geq 4000$. PSGL e DEGL apresentaram desempenhos entre 84% e 96% de taxa de sucesso para $P_{\text{init}} \geq 2000$.

No ambiente CPR office, os algoritmos também apresentaram desempenho satisfatório, com exceção do DEGL cuja performance permaneceu abaixo da média dos demais algoritmos. Conforme exibido na Fig. 5.10b, PSGL e PMGL apresentaram uma taxa de sucesso entre 90% e 100%, bem como alcançaram 100% em 8 casos. AMCL apresentou resultados na média de 98% para $P_{\text{init}} \geq 3000$, enquanto GAGL se manteve em 96%, em média, para qualquer valor de P_{init}, mas alcançou 100% em apenas dois casos.

O ambiente Warehouse provou ser mais difícil, conforme esperado. Inicialmente, foi analisado o percurso por todo o mapa (Fig. 5.9c). Considerando $P_{\text{init}} \geq 2000$, GAGL e PSGL se destacaram com taxas de sucesso de 86% e 83%, respectivamente, como mostra a Fig. 5.10c. DEGL apresentou um desempenho entre 67% e 76% para $P_{\text{init}} \leq 4000$, mas foi inferior nos demais casos. Em contraste, AMCL apresentou uma taxa de sucesso acima de 70% para $P_{\text{init}} \geq 4500$, oscilando entre 21% e 56% com populações menores. PMGL, por sua vez, exibiu desempenho abaixo de 58% para todos os tamanhos de população.

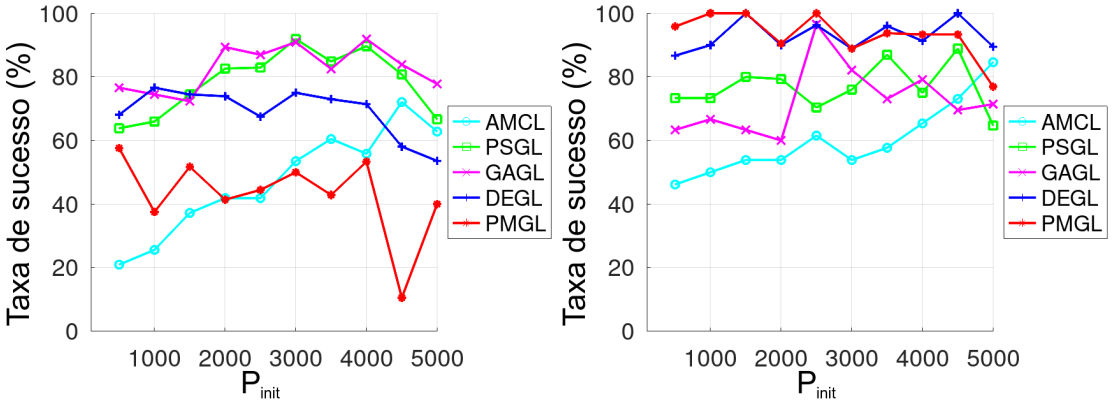
Os desempenhos relativos no percurso em Room B (Fig. 5.9d) se mostraram um pouco diferentes dos cenários anteriores, como mostra a Fig. 5.10d. O ambiente Room B é um espaço simétrico e praticamente vazio, com duas entradas e três balizas próximas de três cantos. Os métodos PMGL com $P_{\text{init}} \leq 4500$ e DEGL apresentaram melhor desempenho em comparação aos demais, oscilando em torno de 90% e alcançando 100% em em pelo menos dois casos (PMGL com $P_{\text{init}} \in [1000, 1500, 2500]$ e DEGL com $P_{\text{init}} \in [1500, 4500]$).

O PSGL oscilou em torno de 80% sem aparente vantagens para populações grandes, porém seu desempenho foi melhor que GAGL e AMCL na maioria dos casos. O GAGL exibiu seu melhor desempenho, 95%, usando $P_{\text{init}} = 2500$, mas se manteve entre 60% e 80% para outros tamanhos de população. O AMCL apresentou um



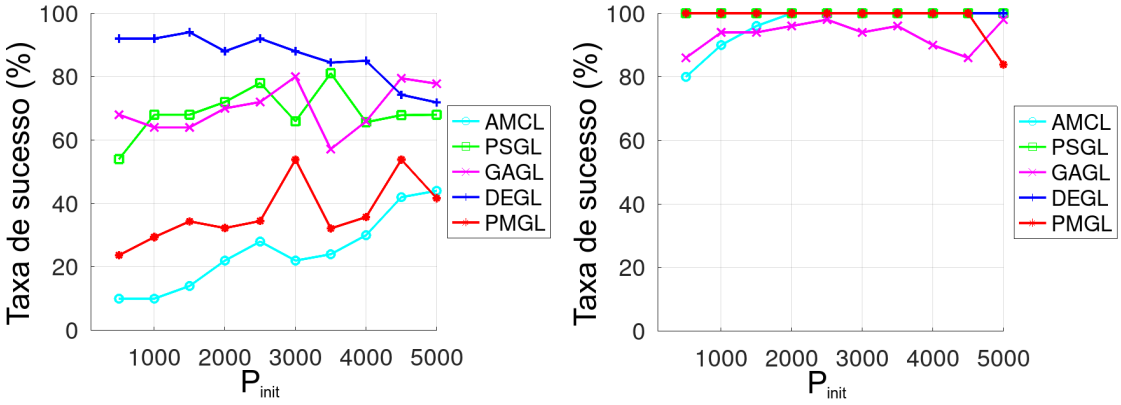
(a) Playpen

(b) CPR Office



(c) Warehouse completo

(d) Warehouse, Room B



(e) Warehouse, Corridor 2

(f) Laboratório de Robótica

Figura 5.10: Taxa de sucesso em cada ambiente, dada pela relação entre o número de localização bem sucedidas e o total de tentativas. O eixo x indica o tamanho da população inicial (P_{init})

desempenho progressivo de 46% a 85% conforme o valor de P_{init} aumenta.

No corredor 2 do ambiente Warehouse (Fig. 5.9e), os algoritmos foram submetidos às condições mais desafiadores, como mostra a Fig. 5.10e. Neste caso, o sensor produz uma nuvem de pontos que esboça duas linhas paralelas, tornando difícil

encontrar a coordenada relativa ao eixo paralelo aos contornos do corredor. Além disso, é possível confundir-se com poses no corredor 1.

Neste cenário DEGL apresentou o melhor desempenho, próximo de 90% para populações menores, porém menor que 80% para $P_{\text{init}} \geq 4500$. PSGL e GAGL demonstraram desempenhos entre 55% and 80% e PMGL progrediu de menos de 30% para mais de 40% para $P_{\text{init}} \geq 4500$. O AMCL iniciou em torno de 10% para $P_{\text{init}} = 500$ e evoluiu para próximo de 45% com $P_{\text{init}} = 5000$.

Até então, nota-se que o PSGL e GAGL demonstram um desempenho acima da média para as duas primeiras condições (Figs. 5.10a, 5.10b e 5.10c). Por outro lado, o DEGL se sobressai em dois cenários desafiadores (Figs. 5.10d e 5.10e). Embora estes cenários sejam aproximações de ambientes reais, eles são inteiramente criados em simulação. Portanto, a avaliação dos métodos propostos deve incluir também dados coletados durante ensaios reais.

A Fig. 5.10f mostra o desempenho dos métodos em um cenário real, onde a trajetória é mostrada pela Fig. 5.9f. O espaço é relativamente pequeno e com grande variedade de contornos, o que pode ser favorável para localização baseada em varredura a laser. Nestas condições PSGL, DEGL e PMGL não falharam para nenhum tamanho de população estabelecido, exceto o PMGL utilizando $P_{\text{init}} = 5000$ que alcançou aproximadamente 84%. O AMCL manteve uma taxa de sucesso de 100% para $P_{\text{init}} \geq 2000$ e o GAGL oscilou entre 86% e 98%.

Em resumo, a Fig. 5.10 mostra que os algoritmos de localização demonstram desempenho satisfatório em cenários pequenos e típicos, como Playpen, CPR Office e LaR. Considerando populações menores, estes algoritmos demonstram taxas de sucesso melhores quando comparados com o AMCL. Acredita-se que isto é devido ao comportamento colaborativo das partículas nos algoritmos evolucionários ou baseados em inteligência de enxame. A troca de informação entre partículas permite que populações menores explorem o espaço de busca de maneira estratégica, o que não é uma característica do AMCL, que por sua vez consiste em reamostrar partículas de uma distribuição de probabilidade.

Com relação ao cenário Warehouse, o desempenho dos algoritmos oscilou. Entretanto, nota-se que o DEGL se destaca como o melhor método para cenários desafiadores como Room B e Corredor 2. Por outro lado, PSGL e GAGL demonstraram ser mais robustos neste ambiente quando comparados com AMCL e PMGL, especialmente no caso que o robô circula pelo mapa inteiro (Fig. 5.9c).

5.2.4 Experimentos utilizando o Intel Lab Dataset

O Intel Lab é um *dataset* público frequentemente usado por pesquisados para analisar algoritmos relacionados a navegação e localização. Este *dataset* contém nuvens

de pontos em 2D de um sensor SICK LMS e dados de odometria, coletados durante um percurso de 45 minutos dentro do Laboratório de pesquisa da Intel.

Porém, este *dataset* não possui *ground truth*, e as informações de odometria não são suficientes devido a propagação de erro. Por outro lado, a pose inicial é conhecida e o *ground truth* pode ser obtido com boa precisão (baseado no alinhamento da nuvem de pontos com os contornos do mapa) utilizando o algoritmo PM. Portanto, o percurso realizado pode ser visualizado na Fig. 5.11, onde as posições registradas são mostradas por pontos em vermelho.

Visto que os pontos inicial e final do percurso coincidem, foi possível reproduzir o *dataset* em *loop* para obter uma trajetória mais longa (permitindo, então um maior número de tentativas de localização global). Nos experimentos realizados, a trajetória foi de 2343 metros percorridos

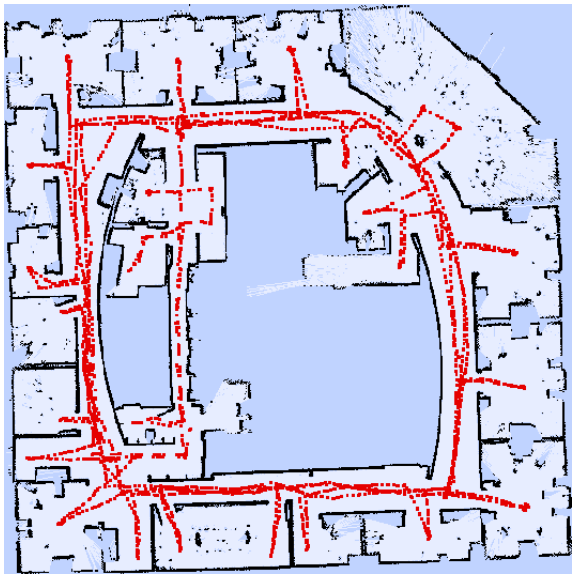


Figura 5.11: Pontos registrados (em vermelho) que foram visitados pelo robô no cenário Intel Lab. O percurso completo é de aproximadamente 2343 metros

A odometria no Intel Lab possui mais ruído do que aquela fornecida pelo Husky, visto que eventualmente a pose se move bruscamente na direção oposta ao percurso. Por causa disso, a etapa de atualização das hipóteses utilizando a Eq. (2.29) pode apresentar uma discrepância significativa em relação ao real movimento do robô. Além disso, o mapa é relativamente grande, possui muitos contornos e salas simétricas, o que pode resultar em um maior nível de ambiguidades. Estas características tornam o Intel Lab um cenário complexo para a localização global baseada em nuvem de pontos e odometria.

Os desempenhos de todos os métodos de localização global são apresentados na Fig. 5.12. Observa-se que os desempenhos de PSGL, GAGL e PMGL não se diferem muito do AMCL, onde as taxas de sucesso oscilaram entre 58% e 84% para

$P_{\text{init}} \in [2000, 4000]$. AMCL, GAGL e PMGL ostentaram a maior taxa de sucesso (84%) quando utilizaram P_{init} igual a 4500, 2500 e 1500, respectivamente. Em contraste, DEGL exibiu resultados insatisfatórios para qualquer valor de P_{init}.

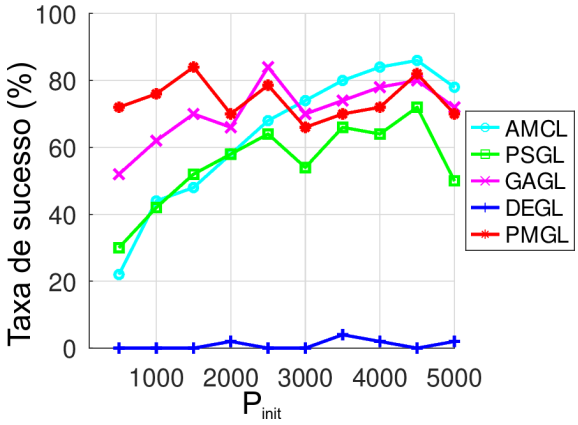


Figura 5.12: Taxas de sucesso no Intel Lab, dado pela relação entre o número de localização bem sucedidas e o total de tentativas

5.2.5 Desempenho médio da localização global

Considerando que a taxa de sucesso oscila em função do cenário, é necessário comparar o desempenho médio de cada método para todas as condições analisadas. A Tabela 5.3 mostra a taxa de sucesso média de cada algoritmo em todos os cenários, divididos pelo tamanho inicial da população. As colunas 2 a 6 mostram que as melhores taxas de sucesso por cada P_{init} são dominados pelo PSGL, GAGL e PMGL.

Por outro lado, é pertinente avaliar o desempenho médio considerando a agregação de populações com tamanhos próximos, de modo que seja possível analisar o comportamento dos métodos para populações pequenas ou grandes. Deste modo, a Tabela 5.4 mostra a taxa de sucesso média para três diferentes faixas de P_{init}. Nesta tabela, os índices de desempenho são resultados da média aritmética das partes equivalente na Tabela 5.3. Nota-se, portanto, que o GAGL exibe o melhor desempenho geral para todas as faixas analisadas.

Até então, os resultados mostram que os métodos propostos são eficazes em resolver o problema da localização global em ambientes pequenos e típicos, como Playpen, CPR Office e LaR. Por outro lado, ambientes espaçosos e simétricos ainda são desafiadores para os métodos de localização global, incluindo o AMCL.

Obviamente, a fidelidade do mapa e a qualidade dos sensores afeta diretamente o desempenho da localização. Embora a nuvem de pontos do sensor LiDAR possua alta acurácia, o sensoriamento é limitado a medição de alcance em 2D. Consequentemente, partes do ambiente podem estar obstruídos por objetos próximos ou invisíveis

Tabela 5.3: Taxa de sucesso dos métodos de localização global por tamanho de população inicial (P_{init} na primeira coluna); colunas dois a seis mostram a porcentagem de localizações bem sucedidas em relação ao total de tentativas para cada método; linhas dois a 11 mostram o desempenho médio para todos os cenários combinados – ou seja, a taxa de sucesso média por cada P_{init}

P_{init}	AMCL	PSGL	GAGL	DEGL	PMGL
500	42,1	71,6	78,4	71,7	78,4
1000	52,2	77,0	79,2	76,4	77,6
1500	60,6	81,5	80,5	79,3	81,4
2000	64,8	84,0	83,6	77,3	76,3
2500	67,8	83,9	90,4	75,2	79,6
3000	71,6	81,3	86,4	74,5	79,8
3500	73,2	87,5	82,5	76,3	76,1
4000	76,1	83,8	85,6	73,9	79,2
4500	81,9	86,3	85,9	71,3	76,6
5000	80,9	76,1	84,3	68,2	73,2

por estarem acima ou abaixo do campo de visão do sensor. Portanto, dado que não há informação adicional além das distâncias medidas e da odometria, e considerando que alguns algoritmos apresentaram desempenho melhor que o AMCL, considera-se que estes métodos são bons candidatos para compor um sistema de localização global incorporado ao PM.

Finalmente, nota-se que um elevado número de partículas pode não beneficiar os métodos apresentados (especialmente para $P_{\text{init}} \geq 4000$) da mesma forma que acontece com o AMCL. Acredita-se que estes métodos se adaptam melhor a populações entre 2000 e 4000 indivíduos, considerando as condições analisadas, o que pode ser observado nas Figs. 5.10 e 5.12.

Além disso, populações muito grandes podem deteriorar o desempenho dos métodos propostos, visto que a eficácia da localização pode ser afetada pelo atraso de processamento provocado pela sobrecarga computacional. Portanto, deve-se levar em consideração o esforço computacional de cada algoritmo para estimar uma pose, visto que os recursos de *hardware* podem ser limitados. Destaca-se que os testes descritos neste trabalho foram realizados em um computador portátil da marca Dell, modelo Inspiron 14 (3443), com processador Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz, memória RAM SODIMM DDR3 Synchronous 1600 MHz (0,6 ns) e com disco de armazenamento secundário de estado sólido Samsung SSD 850 EVO (EMT02B6Q) com 250 GB de capacidade. Neste computador estão instalados o

Tabela 5.4: Taxa de sucesso dos métodos de localização global para cada faixa de P_{init}; colunas dois a seis mostram a porcentagem de localizações bem sucedidas em relação ao total de tentativas para cada método; Linhas dois a quatro mostram o desempenho médio para todos os cenários combinados – ou seja, a taxa de sucesso média para diferentes faixas de P_{init}: $[500, 1000, 1500]$, $[2000, 2500, ..., 5000]$ e $[4000, 4500, 5000]$.

Faixa	AMCL	PSGL	GAGL	DEGL	PMGL
$P_{\text{init}} \leq 1500$	51,6	76,7	79,4	75,8	79,1
$2000 \leq P_{\text{init}} \leq 3500$	69,4	84,2	85,7	75,8	78,0
$P_{\text{init}} \geq 4000$	79,6	82,1	85,3	71,1	76,3

Sistema Operacional Linux Ubuntu 16.04.6 LTS, o sistema ROS na versão Kinetic e o simulador Gazebo multi-robot simulator, versão 7.0.0. A subseção seguinte discute o esforço computacional dos métodos estudados.

5.2.6 Análise do custo computacional

Os desempenhos descritos anteriormente são baseados na eficácia da localização. Entretanto, os métodos possuem abordagens distintas, e portanto seus custos computacionais podem variar. Conforme mencionado na subseção 4.3.6, uma pequena rotina de captura de tempo foi inserida dentro do código dos métodos para calcular o tempo decorrido durante uma iteração.

Os intervalos foram capturados no ambiente Playpen utilizando diferentes valores de P_{init}. Para este estudo, utilizou-se as primeiras 200 iterações de cada algoritmo, e o tempo médio de cada iteração foi calculado utilizando 50 tentativas de localização. Portanto, obteve-se o intervalo médio de cada iteração de todos os algoritmos para cada tamanho de população inicial (P_{init}). Os dados obtidos estão apresentados na Fig. 5.13 utilizando um gráfico de superfície, cujo eixo z (vertical) está em escala logarítmica.

A Fig. 5.13 mostra que, nos métodos desenvolvidos neste trabalho, o custo computacional por iteração é decrescente ao longo das iterações. Isso se deve pela redução gradativa da população, que implica em uma redução de demanda pela função de *fitness*. Além disso, o custo da comunicação entre os processos do método de localização global e do PM também é reduzido na proporção do número de hipóteses, visto que uma menor quantidade delas é transferida entre os processos.

Conforme esperado, o custo computacional do PMGL nas primeiras iterações é pelo menos uma ordem de magnitude superior do que os demais métodos. Observa-

se que PSGL GAGL e DEGL possuem custos bem próximos, apesar de possuírem heurísticas distintas. Isso se deve ao fato de que estes métodos compartilham características em comum, como o mecanismo de redução da população e a comunicação com o *software* do PM. O AMCL se destaca com o menor custo computacional até a centésima iteração, o que acredita-se ser devido à rápida diminuição da população neste cenário e por não estar sujeito à comunicação com o PM Entretanto, a partir da centésima iteração, o AMCL apresenta custo computacional compatível com os demais métodos PSGL GAGL e DEGL.

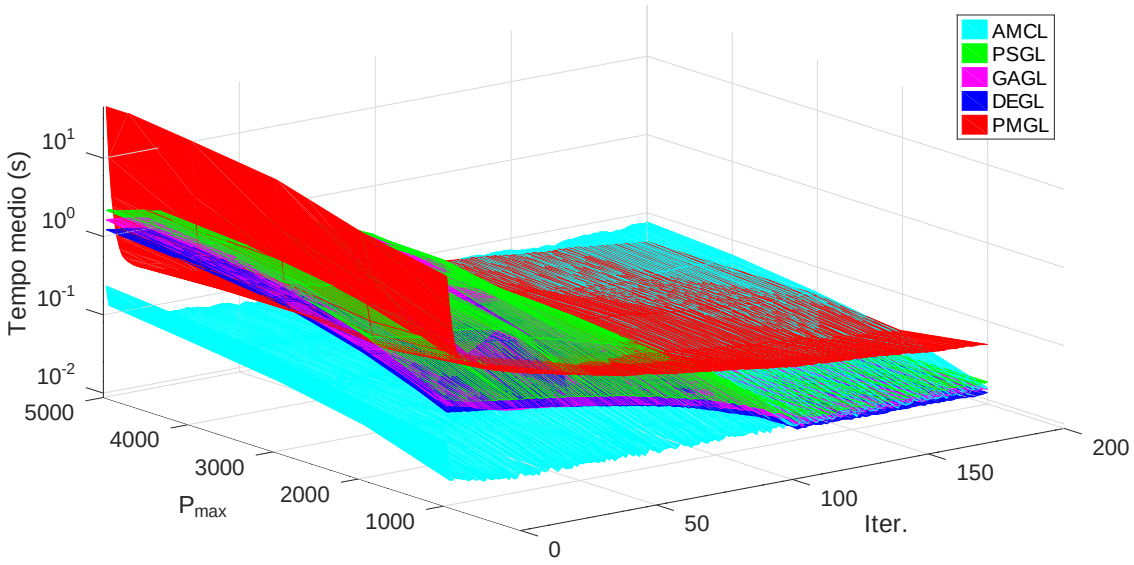


Figura 5.13: Esforço médio em segundos (escala logarítmica) por tamanho de população inicial(P_{init}) e número de iteração

A Tabela 5.5 mostra os tempos médios por iteração de cada algoritmo (colunas três a sete), classificados pelo tamanho inicial da população (P_{init} na primeira coluna) e separados por faixas de iteração (1 a 10, 11 a 100 e 101 a 200 na segunda coluna). Considerando que há um custo adicional devido a comunicação entre processos envolvendo os métodos PSGL GAGL e DEGL e o *software* do PM, necessária para obtenção dos valores de *fitness*, o tempo total de uma iteração e o tempo entre a requisição e resposta da função de *fitness* foram medidos separadamente, com o objetivo de obter os custos intrínsecos de cada heurística, que por sua vez são apresentados nas colunas oito, nove e dez da Tabela 5.5 e, proporcionalmente ao tempo total de cada iteração, nas últimas três colunas.

As colunas oito a dez da Tabela 5.5 mostram o tempo médio de cada iteração sem o atraso devido à função de *fitness*. Apenas PSGL GAGL e DEGL são apresentados porque o AMCL não contém a chamada da função de *fitness* do PM e o PMGL possui o alto custo de otimização de cada partícula, ao invés de apenas obter o valor de *fitness*.

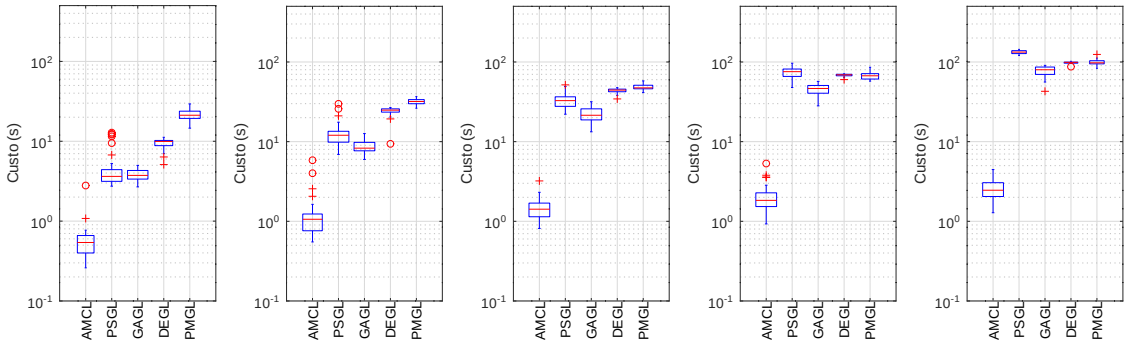
As última três colunas mostram a porcentagem do tempo total de uma iteração gasto com a chamada da função de *fitness*. Estes dados sugerem que, nos métodos baseados no PM, o atraso causado pela função de *fitness* excede 60% nas primeiras dez iterações ao utilizar $P_{\text{init}} \geq 4000$. Nas iterações intermediárias (11-100) o custo da função de *fitness* é moderadamente reduzido e nas últimas iterações (101-200) permanece em aproximadamente um terço do custo das iterações iniciais.

Conforme esperado, o custo proporcional da função de *fitness* é reduzido à medida que a população encolhe visto que menos dados (hipóteses e suas respectivas importâncias) são trafegados na comunicação entre processos. Em resumo, a Tabela 5.5 sugere que a função de *fitness* afeta consideravelmente o custo computacional dos métodos propostos.

Tabela 5.5: Esforço médio de cada algoritmo medido pelo tempo decorrido por iteração e por valor de P_{init}. A primeira e a segunda coluna exigem o tamanho inicial da população e faixa de iterações, respectivamente. Colunas 3 a 7 mostram o tempo médio por iteração de cada algoritmo, enquanto as colunas 8 a 10 mostram este mesmo tempo descontado pelo tempo de resposta da função de *fitness*. As últimas três colunas exibem a proporção de tempo gasto pela função de *fitness* em relação ao tempo total da iteração

		Tempo médio (s)					Tempo médio intrínseco (s)			Custo relativo de *fitness*		
P_{init}	Iterações	AMCL	PSGL	GAGL	DEGL	PMGL	PSGL	GAGL	DEGL	PSGL	GAGL	DEGL
1000	1-10	0,022	0,183	0,154	0,131	1,631	0,068	0,030	0,036	37%	19%	27%
	11-100	0,014	0,099	0,076	0,075	0,210	0,028	0,014	0,017	28%	18%	23%
	101-200	0,014	0,020	0,017	0,015	0,071	0,002	0,001	0,002	10%	6%	13%
2000	1-10	0,044	0,543	0,373	0,333	2,098	0,316	0,139	0,160	58%	37%	48%
	11-100	0,032	0,284	0,201	0,195	0,161	0,128	0,060	0,069	45%	30%	35%
	101-200	0,031	0,029	0,025	0,021	0,052	0,005	0,002	0,003	17%	8%	14%
3000	1-10	0,071	1,182	0,755	0,683	3,864	0,797	0,356	0,408	67%	47%	60%
	11-100	0,047	0,525	0,347	0,323	0,193	0,312	0,145	0,160	59%	42%	50%
	101-200	0,046	0,034	0,028	0,024	0,062	0,007	0,003	0,004	21%	11%	17%
4000	1-10	0,099	1,716	1,179	0,948	5,027	1,337	0,702	0,784	78%	60%	83%
	11-100	0,059	0,746	0,544	0,447	0,183	0,501	0,287	0,309	67%	53%	69%
	101-200	0,058	0,034	0,031	0,024	0,059	0,008	0,005	0,006	24%	16%	25%
5000	1-10	0,132	1,985	1,557	1,151	6,707	1,629	0,992	1,110	82%	64%	96%
	11-100	0,071	0,882	0,719	0,549	0,177	0,640	0,415	0,441	73%	58%	80%
	101-200	0,069	0,032	0,033	0,025	0,063	0,009	0,006	0,007	28%	18%	28%

Como os métodos apresentam diferentes esforços computacionais, o tempo de convergência ao melhor global pode variar entre eles. Portanto, o custo total do início da localização global até a convergência à pose correta foi medido através da soma dos intervalos de todas as iterações entre a primeira e a última (i_{hit}). Neste caso, obviamente, apenas as localizações bem sucedidas no ambiente Playpen são consideradas. Os resultados apresentados na Fig. 5.14 mostram que o AMCL possui o melhor tempo de convergência, enquanto outros algoritmos possuem desempenhos parecidos, especialmente para populações iniciais maiores que 3000.



(a) $P_{\text{init}} = 1000$ (b) $P_{\text{init}} = 2000$ (c) $P_{\text{init}} = 3000$ (d) $P_{\text{init}} = 4000$ (e) $P_{\text{init}} = 5000$

Figura 5.14: Boxplot do tempo decorrido até convergência à pose correta (no ambiente Playpen), obtido através da soma dos intervalos entre as iterações 1 e i_{hit}. O eixo y está em escala logarítmica. Círculos e cruzes em vermelho representam leituras que estão longe da maior parte do conjunto de dados

5.2.7 Discussão sobre os resultados da Localização Global

Esta subseção discorre sobre algumas conjecturas em relação ao comportamento dos métodos nos cenários analisados. A discussão se divide entre o comportamento em ambientes pequenos, em ambientes grandes e o caso particular do desempenho do DEGL no ambiente Intel Lab.

Comportamento dos métodos em ambientes pequenos

O desempenho dos métodos estudados em ambientes pequenos (área de até 400 m^2) pode ser analisado pelos gráficos das Figs. 5.10a e 5.10f. Estes gráficos sugerem que as populações devem ser maiores que 1500 para que os métodos expressem seu maior potencial de localização (quando $P_{\text{init}} \leq 1500$, a taxa de sucesso fica abaixo de 90%). Estes resultados estão compatíveis com aqueles observados com os testes preliminares utilizando o PSO (Fig. 5.5), onde a taxa de sucesso é inferior a 80% quando há menos de três partículas por metro quadrado.

Para os demais tamanhos de população analisados, $2000 \leq P_{\text{init}} \leq 5000$, esperava-se uma taxa de sucesso de aproximadamente 90% ou mais, visto que para estes tamanhos de população inicial os ambientes Playpen e LaR possuem pelo menos cinco e 22 partículas por metro quadrado, respectivamente. Nestas condições, os métodos apresentaram taxas de sucesso relativamente próximas, mas o GAGL, método com melhor desempenho médio conforme Tabela 5.4, foi o único método que teve uma taxa de sucesso inferior no LaR em relação ao Playpen. Por ser um ambiente pequeno, supõe-se que a proximidade dos contornos dentro do LaR tenha proporcionado mínimos locais menos acentuados do que no Playpen (a distância de cada ponto do mapa até o contorno mais próximo e relativamente pequena, e portanto os mínimos locais são menos distinguíveis), e que a heurística do AG seja mais sensível a esta configuração.

Apesar de possuir um mapa maior que 400 m^2, o CPR Office poderia ser considerado um mapa pequeno porque tem apenas 42% de área livre (Fig. 4.23b), enquanto que o Playpen possui 88% de área livre (Fig. 4.22b). Por isso, na prática os métodos de Localização Global no CPR Office dispõem de uma densidade de partículas similar ao Playpen. Sendo assim, nota-se que o desempenho no CPR Office está, de fato, semelhante o Playpen, com exceção do DEGL que teve o desempenho piorado. Uma das hipóteses para esta diminuição da taxa de sucesso do DEGL é apresentada nas próximas subseções.

Comportamento dos métodos no ambiente Warehouse

Conforme citado anteriormente, o desempenho dos algoritmos em um ambiente amplo como o Warehouse oscilou. Isso se deve ao fato de que a densidade de partículas é menor, pose há menos de uma partícula por metro quadrado do mapa quando $P_{\text{init}} \leq 2000$ e aproximadamente $1,7$ partículas quando $P_{\text{init}} = 5000$. Ainda assim, os métodos propostos neste trabalho apresentaram taxas de sucesso superiores à do AMCL na maioria dos casos, chegando em torno de 90% no experimento do percurso completo (Fig. 5.10c), o que reforça a aptidão de tais métodos como aprimoramento do PM. Nestes cenários, destaca-se o desempenho do DEGL, que chegou a ser maior que todos os outros métodos no Corridor 2 (Fig. 5.10e), o que pode ter sido influenciado pelo comportamento desecrito na subseção a seguir.

Desempenho inexpressivo do DEGL no Intel Lab

Confrontando os percursos apresentados nas Figs. 5.9 e 5.11 com as taxas de sucesso das Figs. 5.10 e 5.12, pode-se deduzir que o DEGL se comporta melhor quando o *ground truth* está próximo do centro do espaço de busca do algoritmo. Isso foi percebido no corredor do ambiente Warehouse, onde o DEGL demonstrou

desempenho melhor que os demais métodos provavelmente porque o robô se encontrava próximo do centro do mapa, enquanto que no Intel Lab, onde o DEGL teve muita dificuldade em encontrar a pose correta, o robô percorreu a periferia do mapa na maior parte do tempo.

Uma hipótese que pode justificar este comportamento é o fato de que cada partícula do DEGL frequentemente se move em direção à outras três selecionadas aleatoriamente, e portanto a probabilidade de que uma partícula na periferia do mapa permaneça se movendo pelos cantos do mapa é proporcional ao número de partículas nesta região. Por exemplo, supondo que a localização global inicie quando o robô está dentro da sala da extremidade inferior esquerda do mapa Intel Lab (Fig. 5.11), algumas partículas próximas a esta sala poderão convergir à pose correta. Porém, a maioria das partículas estão acima e à direita da localização do robô no mapa, justamente porque o mesmo se encontra em uma extremidade. Portanto, a probabilidade de que uma partícula próxima à sala na qual o robô se encontra selecione aleatoriamente outras três partículas fora da sala, acima e à direita da localização do robô, é maior do que a seleção de partículas dentro da sala.

Por outro lado, no caso do corredor 2 em Warehouse, o DEGL pode ter se comportado melhor que os demais métodos porque a pose correta se encontra próxima do centro do mapa na maior parte da simulação, e neste caso as partículas das extremidades têm maior probabilidade de selecionar partículas aleatórias que estão em direção ao centro do mapa. Supõe-se, portanto, que o mecanismo aleatório do método DE se comporta melhor quando o centro de massa das partículas não está afastado da localização da solução correta. O mesmo efeito não acontece com o PSGL, provavelmente porque as partículas se movem em direção à melhor solução ($\mathbf{g}_{\text{best}}$) ao invés de indivíduos aleatórios da população.

5.3 Incerteza sobre a pose

Esta seção descreve a análise da incerteza estimada pelos métodos de rastreamento de pose. Durante a movimentação do robô, todos os métodos (AMCL, PM e PMCov) foram executados concorrentemente. O AMCL foi configurado com população inicial $P_{\text{init}} = 2500$ e mínima $P_{\text{min}} = 500$, considerando que esta população é suficiente, conforme dados observados na Fig. 5.10f. O parâmetro L_c foi definido em $0,5$ em ambos PM e PMCov, em linha com os experimentos da seção anterior. As cinco situações analisadas estão ilustradas nas Figs. 5.15 a 5.19.

O *ground truth*, representado pela seta em preto, foi obtido anteriormente utilizando o método PM (por monitoramento visual, observou-se que o rastreamento da pose permaneceu correto durante todo o percurso). As setas em azul, vermelho e verde são as poses com covariâncias do AMCL, PM e PMCov, respectivamente,

estimados durante os experimentos. Cada elipse ilustrada nas figuras demarca a incerteza sobre a posição dada pela seta com a respectiva cor, enquanto os cones representam a incerteza sobre a orientação. A seta em verde está coberta pela seta em vermelho em todas as figuras porque elas compartilham da mesma posição e orientação (isso ocorre porque ambos os métodos utilizam a mesma pose fornecida pelo PM). A seta em rosa é utilizada em alguns casos (Figs. 5.16 e 5.18) para indicar uma pose erroneamente inicializada.

Os pontos em amarelo representam as amostras do sensor LiDAR projetadas a partir da pose do PM (nota-se que a novem de pontos não coincide bem com os contornos do mapa quando a pose está deslocada, como mostra as Figs. 5.16, 5.18 e 5.19). A nuvem de setas pequenas em branco representam a população do AMCL. As situações analisadas estão descritas a seguir:

- Fig. 5.15 - o robô se move dentro do LaR e todos os algoritmos de localização estão rastreando a pose correta. Os contornos do mapa apresentam ruídos, o que resulta em uma transformada de distâncias congestionada e mapas de gradientes com aparência bastante acidentada, como visto no canto superior esquerdo das Figs. 4.13 e 4.14). Além disso, há pessoas dentro da sala cujos contornos das pernas aparecem na nuvem de pontos (observa-se os contornos de quatro pernas próximo à porta de entrada do laboratório, no canto inferior direito);

- Fig. 5.16 - os métodos de localização são inicializados a partir de uma posição incorreta dentro do LaR, conforme indicado pela seta em rosa. No momento da inicialização, o *ground truth* está representado pelo ponto em vermelho na parte de baixo do ambiente. A partir daí, o robô se move em linha reta por aproximadamente dois metros (como mostra a linha pontilhada em vermelho) e termina na pose mostrada pela seta em preto. Conforme esperado, todos os métodos de localização estão rastreando uma pose incorreta.

- Fig. 5.17 - o robô se retira do saguão e se move dentro do corredor em direção ao LaR. Os contornos do mapa e a nuvem de pontos do sensor LiDAR esboçam duas linhas paralelas, o que pode afetar a confiança do robô em relação à coordenada no eixo x. Apesar disso, os métodos de localização mantêm o rastreamento da pose correta;

- Fig. 5.18 - os métodos de localização são inicializados em algum ponto do corredor (seta em rosa), nas proximidades do *ground truth*, porém na direção oposta. Então, o robô se move por aproximadamente quatro metros em linha reta e termina na pose mostrada pela seta em preto. Conforme esperado, todos os métodos de localização estão rastreando a pose incorreta;

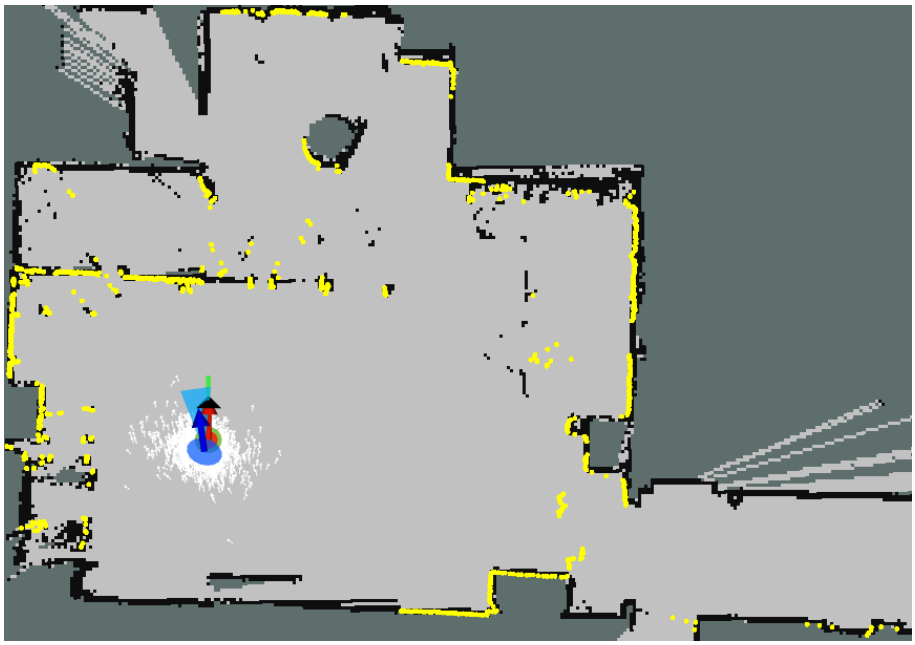


Figura 5.15: Incerteza sobre a pose no LaR: o robô se move pelo espaço livre e todos os métodos de localização rastreiam a pose correta. As setas em azul, vermelho e verde são as poses com covariâncias do AMCL, PM e PMCov, respectivamente. Pontos em amarelo representam amostras do sensor LiDAR

- Fig. 5.19 - o robô está entrando no LaR (que não está visível nesta figura) mas o mesmo assume, de forma equivocada, que sua posição está em algum lugar dentro do corredor e na direção oposta. Do ponto de vista dos métodos de localização, a nuvem de pontos do sensor LiDAR não esboçam duas linhas paralelas como seria esperado neste local.

Na primeira situação (Fig. 5.15) a nuvem de pontos do sensor LiDAR apresenta uma boa similaridade *scan-to-map*. Porém, devido à presença de *outliers* e de mobília com contornos finos, existem diversas amostras que incidem em posições distantes de células ocupadas. Considerando isso, é esperado que os métodos de localização forneçam uma uma pose com incerteza razoavelmente baixa. O PM estimou as menores covariâncias para x e y e o AMCL forneceu as maiores, como mostra a Tabela 5.6.

Ainda no LaR, a incerteza sobre a pose deveria ser maior quando o *scan-to-map matching* é precário, como ilustrado no cenário da Fig. 5.16. Neste caso, o método PMCov expressou a maior incerteza para ambos as coordenadas dos eixos x e y, enquanto o PM apresentou a menor incerteza (como visto na Tabela 5.6).

A localização no corredor é uma tarefa desafiadora devido à simetria dos contornos nestes espaços. Na Fig. 5.17 o método PMCov ostentou a maior incerteza sobre o eixo x e a menor sobre o eixo y, enquanto AMCL e PM demonstraram valores parecidos.

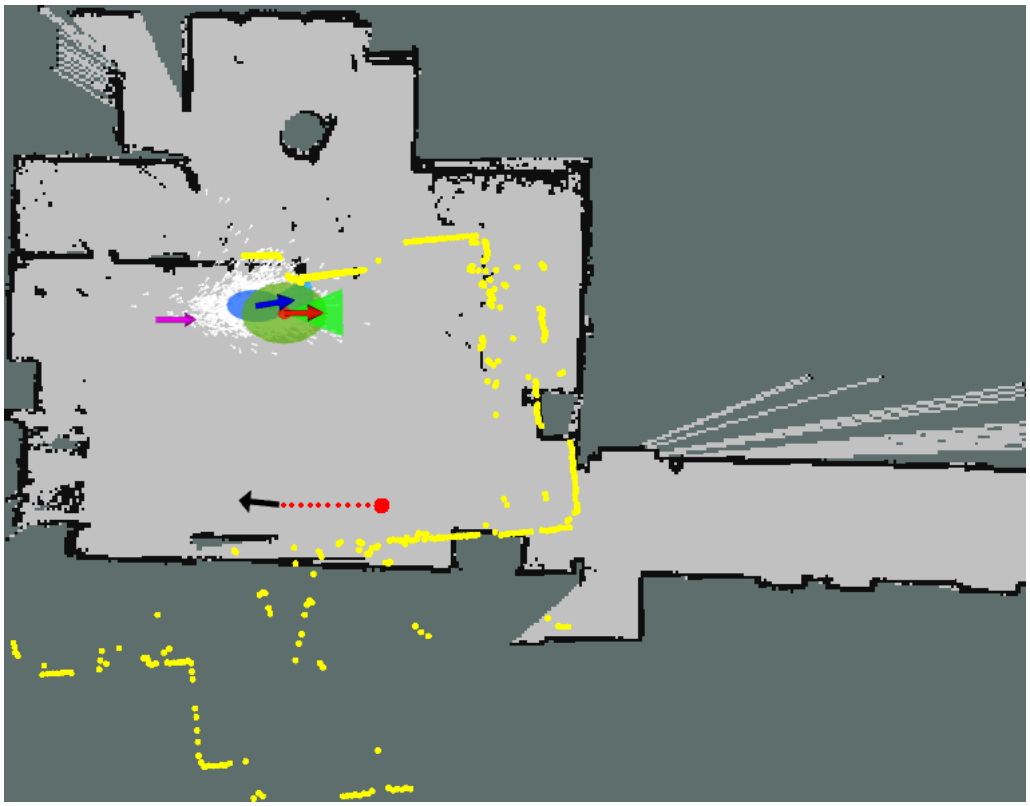


Figura 5.16: Incerteza sobre a pose no LaR: o robô foi "sequestrado" e todos os métodos de localização rastreiam uma pose incorreta. As setas em azul, vermelho e verde são as poses com covariâncias do AMCL, PM e PMCov, respectivamente. A seta em rosa indica uma pose erroneamente inicializada, enquanto o traçado em vermelho representa o percurso correto desde a inicialização. Pontos em amarelo representam amostras do sensor LiDAR

Por outro lado, quando a pose rastreada está alterada, como na situação da Fig. 5.18, PMCov exibiu um desvio padrão σ_x muito maior (que se refere à incerteza sobre o eixo que está em paralelo com corredor), enquanto o AMCL apresentou a segunda maior incerteza. Além disso, nesta situação (Fig. 5.18) os valores de σ_y fornecidos pelo AMCL e pelo PMCov foram maiores quando comparados com o rastreamento da pose correta (Fig. 5.17).

Finalmente, a Fig. 5.19 ilustra a situação em que o robô foi "sequestrado" de algum ponto do corredor para a entrada do LaR. Em outras palavras, o robô está entrando no Laboratório mas os algoritmos de localização rastreiam uma pose ao longo do corredor e na direção oposta. Nestas circunstâncias, o *scan-to-map matching* é precário e os contornos próximos às poses estimadas pelos métodos são simétricos, portanto, espera-se uma alta incerteza sobre a pose, especialmente com relação à coordenada de x.

Novamente, o método PMCov apresentou os maiores valores para σ_x e σ_y, en-

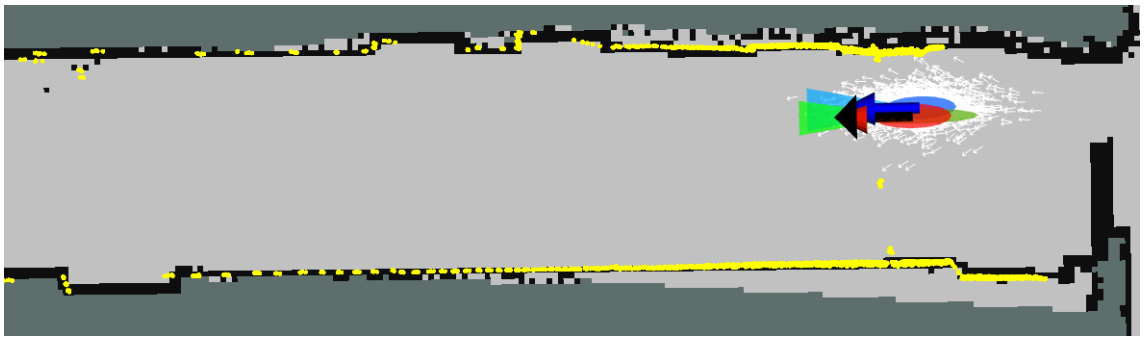


Figura 5.17: Incerteza sobre a pose no corredor: o robô está no início do corredor e se move em direção ao LaR enquanto todos os métodos de localização rastreiam a pose correta. As setas em azul, vermelho e verde são as poses com covariâncias do AMCL, PM e PMCov, respectivamente. Pontos em amarelo representam amostras do sensor LiDAR

Figura 5.18: Incerteza sobre a pose no corredor: o robô está no meio do corredor e se move em direção ao LaR enquanto todos os métodos de localização rastreiam alguma pose incorreta na direção oposta ao Laboratório. As setas em azul, vermelho e verde são as poses com covariâncias do AMCL, PM e PMCov, respectivamente. A seta em rosa indica uma pose erroneamente inicializada, enquanto o traçado em vermelho representa o percurso correto desde a inicialização. Pontos em amarelo representam amostras do sensor LiDAR

quanto o PM forneceu os menores. Não foram citadas as diferenças nas incertezas sobre θ porque os valores apresentados pelo PM foram demasiadamente pequenos se comparados com o AMCL e PMCov. Porém, a Tabela 5.6 mostra que, comparando os valores de σ_θ, os resultados se mostram compatíveis com a discussão anterior referente a σ_x e σ_y.

Em resumo, nota-se que a incerteza estimada pelo método PMCov é mais consistente com as incertezas esperadas, quando comparada com outros métodos nas situações descritas. As variâncias listadas na Tabela 5.6 e as elipses e cones de incerteza visualizadas nas Figs. 5.15 a 5.19 sugerem que o método proposto é mais apropriado, considerando os cenários estudados.

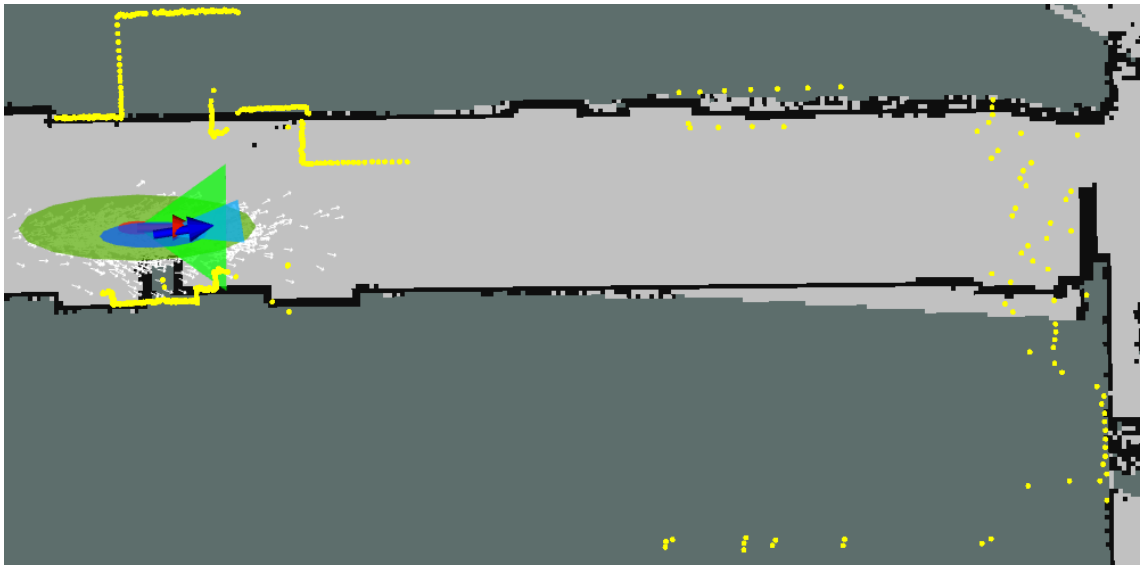


Figura 5.19: Incerteza sobre a pose no problema do sequestro, onde o robô está entrando no LaR mas todos os algoritmos rastreiam alguma pose ao longo do corredor, na direção oposta à pose correta. As setas em azul, vermelho e verde são as poses com covariâncias do AMCL, PM e PMCov, respectivamente. Pontos em amarelo representam amostras do sensor LiDAR

Tabela 5.6: Desvio padrão apresentado por cada método[a] para cada uma das variáveis da pose estimadas (x, y e θ); Cada linha representa um dis cenários descritos anteriormente dados pelas Figs. 5.15 a 5.19

	AMCL			PM			PMCov		
Situação	σ_x	σ_y	σ_θ	σ_x	σ_y	σ_θ	σ_x	σ_y	σ_θ
Fig. 5.15	0,277	0,217	13,559	0,148	0,138	0,003	0,214	0,207	1,812
Fig. 5.16	0,497	0,277	18,917	0,110	0,105	0,003	0,733	0,525	22,265
Fig. 5.17	0,324	0,100	10,719	0,356	0,105	0,003	0,592	0,063	8,690
Fig. 5.18	0,513	0,118	12,812	0,300	0,071	0,002	1,024	0,134	16,407
Fig. 5.19	0,606	0,145	13,799	0,071	0,077	0,002	1,338	0,373	35,552

[a] σ_x e σ_y dados em metros e σ_θ em graus.

Capítulo 6

Conclusões e Trabalhos Futuros

Neste trabalho, foram propostos diferentes métodos para resolver o PLG e aprimorar o cálculo da incerteza sobre a pose para um algoritmo de localização, o *Perfect Match*. Um dos objetivos foi avaliar os métodos baseados em Algoritmos Evolucionários e Inteligência de Enxame, comparando-os com um método baseado no *Initial Position Computation* proposto por Pinto *et al.* [15], o PMGL. Os métodos de otimização por multi-hipóteses desenvolvidos neste trabalho utilizam uma função de *fitness* baseada no erro de *matching* do PM como métrica de desempenho das partículas. Para avaliar o desempenho relativo destes métodos, o método de localização bem conhecido AMCL foi executado sob as mesmas condições de teste, incluindo ambientes de simulação e *datasets* reais.

A análise mostrou que, com a exceção do DEGL, todos os métodos demonstraram bom desempenho. considerando cenários pequenos e com riqueza de detalhes em seus contornos. Por outro lado, estes algoritmos reduziram a eficácia em cenários mais difíceis, como o corredor, por exemplo, no qual o DEGL apresentou um desempenho melhor. PSGL e GAGL em particular apresentaram um nível de eficácia consistente entre diferentes cenários, e observou-se que o GAGL converge para a melhor solução global em menor tempo, porém não apresentou eficácia tão alta como os demais métodos no LaR.

No geral, considerando os métodos baseados em AE e IE, PSGL e GAGL em particular apresentaram os melhores desempenhos e o último foi capaz de convergir para a melhor solução global no menor tempo. Com relação ao *dataset* público Intel Lab, que não é baseado na plataforma Husky, GAGL demonstrou robustez, registrando o melhor desempenho com até 2500 indivíduos. Por outro lado, o GAGL obteve melhor aproveitamento utilizando populações maiores.

Esperava-se que o PMGL apresentaria uma alta eficácia devido a sua abordagem intensiva do ponto de vista computacional. Porém, este método não apresentou desempenho satisfatório em cenários como o Warehouse (Fig. 5.10c) e corredor 2 da

Warehouse (Fig. 5.10e), quando comparado com o PSGL, GAGL e DEGL. Devido a sua alta demanda por recurso computacional, o PMGL não se apresentou como uma abordagem eficiente.

Os custos computacionais dos métodos baseados em AE e IE são similares, visto que eles compartilham da mesma estrutura básica para gerar a população e fazer chamadas à função de *fitness* (conforme ilustrado na Fig. 4.1). Apesar disso, é possível reduzir o custo computacional se o método for totalmente integrado ao *software* do PM, eliminando o *overhead* de comunicação. É importante notar que após a centésima iteração, os métodos propostos demonstram menor custo computacional quando comparados com o AMCL (as como mostra a Tabela 5.5).

Os experimentos de localização global foram realizados utilizando um computador portátil típico e uma plataforma robótica de médio porte, com sistema totalmente baseado em ROS. Portanto, os métodos propostos podem satisfazer aplicações onde são utilizados *Single Board Computers*, como o Raspberry Pi, caso o desenvolvedor esteja disposto a utilizar populações compatíveis com o ambiente e com a capacidade do *hardware*.

Por outro lado, no contexto de sistemas de tempo real, a viabilidade das propostas apresentadas ainda depende de mais estudos. Deve-se considerar que algoritmos apresentados são passivos, ou seja, dependem de um deslocamento mínimo do robô para realizar uma nova iteração. Uma vez que o movimento do robô em uma aplicação real muitas vezes é imprevisível, não é possível inferir o seu uso do processador de forma precisa. Por causa disso, este trabalho não envolveu estudos sobre a viabilidade dos métodos em sistemas de tempo real.

Em resumo, a escolha do método de localização global para ser incorporado ao PM pode depender da análise das condições da aplicação, dado que os desempenhos oscilam em diferentes cenários. Primeiramente, pode-se concluir que a eficácia do PMGL se estabelece ao custo de um alto esforço computacional comparado a outros métodos avaliados, especialmente em ambientes amplos. O DEGL demonstrou eficácia superior em dois cenários críticos (Figs. 5.10d e 5.10e), mas seu desempenho nos demais cenários foi preocupante.

O PSGL e o GAGL apresentaram desempenhos próximos, portanto seria possível reconhecer a superioridade de ambos. No entanto, no experimento realizado no Intel Lab, que utiliza dados de outra fonte, o GAGL apresentou um aproveitamento distintivamente superior em relação ao PSGL. Consequentemente, pode-se considerar que o GAGL apresentou o melhor custo-benefício entre eficácia, robustez e custo computacional.

Uma das principais perspectivas futuras deste trabalho é a incorporação de um dos métodos baseados em AE e IE ao PM, como já citado anteriormente, em especial o GAGL. Neste caso, a junção, que depende de um maior nível de intervenção no

software do PM, poderá reduzir consideravelmente o custo computacional da localização global e torná-lo competitivo com o AMCL ainda nas iterações iniciais. Além disso, também é viável ampliar os estudos apresentados neste trabalho utilizando outras heurísticas de otimização multi-hipóteses que não tiveram a oportunidade ser incluídas por limitação de tempo, como o Simulated Annealing e Covariance matrix adaptation evolution strategy (CMA-ES).

Em uma outra linha de investigação, observa-se a possibilidade de aplicar técnicas de *Machine Learning* na localização global, utilizando o mapa e os dados dos sensores para fazer treinamento em redes neurais capazes de identificar potenciais poses. O trabalho realizado por Wang *et al.* [21], por exemplo, utiliza uma rede neural convolucional para realizar uma localização global grosseira e uma combinação de busca binária e AMCL para fazer o refinamento desta localização. O funcionamento da rede neural se baseia em características extraídas de imagens RGB, mas uma abordagem similar seria possível utilizando a função de *fitness* do PM.

Este trabalho também avaliou o desempenho de uma nova abordagem para estimar a incerteza sobre a pose, PMCov, comparando-a com a abordagem original utilizada pelo PM e com o AMCL. Observou-se que o PMCov apresentou baixa incerteza quando a pose estimada está correta, a maior incerteza quando o robô está perdido, e a maior incerteza em relação ao eixo em paralelo com os contornos de um corredor. Portanto, os resultados sugerem que, ao menos nos cenários estudados, a abordagem proposta apresenta maior relevância para a confiança sobre a pose, quando comparado com outros métodos.

Acredita-se que a incerteza sobre a pose carrega uma informação importante no contexto do problema do sequestro. Por exemplo, o robô pode utilizar a incerteza estimada para avaliar a qualidade do rastreamento da pose e identificar se o mesmo se encontra perdido. Neste contexto, os experimentos apresentados indicam que a solução proposta neste trabalho estabelece uma boa relação entre a incerteza sobre a pose e a corretude da localização. Como trabalho futuro, observa-se a possibilidade de incorporar o modelo da odometria ao método de estimação da incerteza, bem como realizar estudos similares com diferentes *datasets*.

Referências Bibliográficas

[1] CORKE, P., *Robotics, vision and control: fundamental algorithms in MATLAB® second, completely revised*. v. 118. Springer: Cham, 2017.

[2] MORAVEC, H., ELFES, A., "High resolution maps from wide angle sonar". In: *Proceedings. 1985 IEEE international conference on robotics and automation*, v. 2, pp. 116–121, 1985.

[3] THRUN, S., BURGARD, W., FOX, D., *Probabilistic robotics*. MIT Press: Cambridge, 2005.

[4] KLANCAR, G., ZDESAR, A., BLAZIC, S., et al., *Wheeled mobile robotics: from fundamentals towards autonomous systems*. Butterworth-Heinemann, 2017.

[5] ALMEIDA, T., SANTOS, V., MOZOS, O. M., et al., "Comparative Analysis of Deep Neural Networks for the Detection and Decoding of Data Matrix Landmarks in Cluttered Indoor Environments", *Journal of Intelligent & Robotic Systems*, v. 103, n. 1, pp. 1–14, 2021.

[6] DIGIACOMO, F., BOLOGNA, F., INGLESE, F., et al., "MechaTag: A Mechanical Fiducial Marker and the Detection Algorithm", *Journal of Intelligent & Robotic Systems*, v. 103, n. 3, pp. 1–11, 2021.

[7] PEI, F., ZHU, M., WU, X., "A decorrelated distributed EKF-SLAM system for the autonomous navigation of mobile robots", *Journal of Intelligent & Robotic Systems*, v. 98, n. 3, pp. 819–829, 2020.

[8] PRAKASH, K., MOHAMED, M. N. G., CHAKRAVORTY, S., et al., "Structure Aided Odometry (SAO): A Novel Analytical Odometry Technique Based on Semi-Absolute Localization for Precision-Warehouse Robotic Assistance in Environments with Low Feature Variation", *Journal of Intelligent & Robotic Systems*, v. 102, n. 4, pp. 1–24, 2021.

[9] SOBREIRA, H., ROCHA, L., COSTA, C., et al., "2D Cloud Template Matching - A Comparison between Iterative Closest Point and Perfect Match". In:

2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC), pp. 53–59, 2016.

[10] FILOTHEOU, A., TSARDOULIAS, E., DIMITRIOU, A., et al., "Pose Selection and Feedback Methods in Tandem Combinations of Particle Filters with Scan-Matching for 2D Mobile Robot Localisation", *Journal of Intelligent & Robotic Systems*, v. 100, n. 3, pp. 925–944, 2020.

[11] LAUER, M., LANGE, S., RIEDMILLER, M., "Calculating the Perfect Match: An Efficient and Accurate Approach for Robot Self-localization". In: *RoboCup 2005: Robot Soccer World Cup IX*, pp. 142–153, Springer: Berlin, Heidelberg, 2006.

[12] GOUVEIA, M., MOREIRA, A. P., COSTA, P., et al., "Robustness and precision analysis in map-matching based mobile robot self-localization". In: *EPIA, 14th Portuguese Conference on Artificial Intelligent*, pp. 243–253, 2009.

[13] PINTO, M., MOREIRA, A. P., MATOS, A., et al., "Novel 3D matching self-localisation algorithm", *International Journal of Advances in Engineering & Technology*, v. 5, n. 1, pp. 1–12, 2012.

[14] SOBREIRA, H., PINTO, M., MOREIRA, A. P., et al., "Robust robot localization based on the perfect match algorithm". In: *CONTROLO2014–Proceedings of the 11th Portuguese Conference on Automatic Control*, pp. 607–616, 2015.

[15] PINTO, M., SOBREIRA, H., MOREIRA, A. P., et al., "Self-localisation of indoor mobile robots using multi-hypotheses and a matching algorithm", *Mechatronics*, v. 23, n. 6, pp. 727–737, 2013.

[16] FARIAS, P., SOUSA, I., SOBREIRA, H., et al., "Approach for supervising self-localization processes in mobile robots". In: *EPIA Conference on Artificial Intelligence*, pp. 461–472, 2017.

[17] SOBREIRA, H. M. P., *Fiabilidade e robustez da localização de robôs móveis*, Ph.D. Thesis, Faculdade de Engenharia da Universidade do Porto, 2017.

[18] GUEDES, P. M. L., *Localização e Navegação de robôs em ambientes dinâmicos e com troca automática de baterias*, Ph.D. Thesis, Faculdade de Engenharia da Universidade do Porto, 2018.

[19] TEIXEIRA, E. P., *Localização e Navegação de AGVs Industriais*, Ph.D. Thesis, Faculdade de Engenharia da Universidade do Porto, 2018.

[20] SOBREIRA, H., COSTA, C. M., SOUSA, I., et al., "Map-matching algorithms for robot self-localization: a comparison between Perfect Match, Iterative Closest Point and Normal Distributions Transform", *Journal of Intelligent & Robotic Systems*, v. 93, n. 3, pp. 533–546, 2019.

[21] WANG, Z., GAO, F., ZHAO, Y., et al., "A Deep Hierarchical Framework for Robot Global Localization", *Journal of Intelligent & Robotic Systems*, v. 106, n. 2, pp. 1–12, 2022.

[22] CARVALHO, J. L. C., FARIAS, P. C. M., DE SOUZA, E. E. P., et al., "Particle Swarm Localization for Mobile Robots Using a 2D Laser Sensor". In: *2019 8th Brazilian Conference on Intelligent Systems (BRACIS)*, pp. 281–286, 2019.

[23] ZHANG, L., ZAPATA, R., LÉPINAY, P., "Self-adaptive Monte Carlo localization for mobile robots using range sensors". In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1541–1546, 2009.

[24] SCHILLING, R., *Fundamentals of robotics: analysis and control*. Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

[25] CORRELL, N., *Introduction to Autonomous Robots*. Magellan Scientific, 2016.

[26] PYO, Y., CHO, H., JUNG, R., et al., "ROS Robot Programming: from the basic concept to practical programming and robot application", *ROBOTIS Co., December*, 2017.

[27] MIHELJ, M., BAJD, T., UDE, A., et al., *Robotics*. Springer, 2019.

[28] FOOTE, T., "tf: The transform library". In: *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on, Open-Source Software workshop*, pp. 1–6, April 2013.

[29] SPONG, M. W., HUTCHINSON, S., VIDYASAGAR, M., *Robot modeling and control*. v. 3. Wiley New York, 2006.

[30] SICILIANO, B., SCIAVICCO, L., VILLANI, L., et al., *Robotics: modelling, planning and control*. Springer, 2009.

[31] CHOSET, H., LYNCH, K. M., HUTCHINSON, S., et al., *Principles of robot motion: theory, algorithms, and implementations*. MIT press: Cambridge, MA, 2005.

[32] HAMILTON, W. R., "Researches Respecting Quaternions. First Series", *The Transactions of the Royal Irish Academy*, v. 21, pp. 199–296, 1846.

[33] RAIBERT, M., BLANKESPOOR, K., NELSON, G., et al., "Bigdog, the rough-terrain quadruped robot", *IFAC Proceedings Volumes*, v. 41, n. 2, pp. 10822–10825, 2008.

[34] LILJEBÄCK, P., PETTERSEN, K. Y., STAVDAHL, Ø., et al., "A review on modelling, implementation, and control of snake robots", *Robotics and Autonomous systems*, v. 60, n. 1, pp. 29–40, 2012.

[35] FERNANDES, L. C., SOUZA, J. R., PESSIN, G., et al., "CaRINA intelligent robotic car: architectural design and applications", *Journal of Systems Architecture*, v. 60, n. 4, pp. 372–392, 2014.

[36] ANGELES, J., *Fundamentals of robotic mechanical systems: theory, methods, and algorithms*. Springer: Switzerland, 2014.

[37] BOAS, L. S. V., CONCEIÇAO, A. G., "Modelagem Cinemática de Robôs Móveis da Classe Skid-Steer". In: *Congresso Brasileiro de Automática-CBA*, v. 2, 2020.

[38] Clearpath Robotics, *HUSKY UGV User Manual*, 2018, Rev. 1.1.3.

[39] KOHLBRECHER, S., VON STRYK, O., MEYER, J., et al., "A flexible and scalable SLAM system with full 3D motion estimation". In: *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pp. 155–160, 2011.

[40] CHOI, J., MAURER, M., "Hybrid map-based SLAM with Rao-Blackwellized particle filters". In: *17th international conference on information fusion (FUSION)*, pp. 1–6, Salamanca, Spain, 2014.

[41] LEE, H., CHUN, J., JEON, K., et al., "Efficient EKF-SLAM Algorithm Based on Measurement Clustering and Real Data Simulations". In: *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pp. 1–5, 2018.

[42] JOO, S.-H., LEE, U.-H., KUC, T.-Y., et al., "A robust SLAM algorithm using hybrid map approach". In: *2018 International Conference on Electronics, Information, and Communication (ICEIC)*, pp. 1–2, 2018.

[43] JOUBERT, D., BRINK, W., HERBST, B., "Pose uncertainty in occupancy grids through Monte Carlo integration", *Journal of Intelligent & Robotic Systems*, v. 77, n. 1, pp. 5–16, 2015.

[44] IVANJKO, E., PETROVIC, I., "Extended Kalman filter based mobile robot pose tracking using occupancy grid maps". In: *Proceedings of the 12th IEEE Mediterranean Electrotechnical Conference (IEEE Cat. No.04CH37521)*, v. 1, pp. 311–314 Vol.1, 2004.

[45] CHIEN, C.-H., WANG, W.-Y., JO, J., et al., "Enhanced Monte Carlo localization incorporating a mechanism for preventing premature convergence", *Robotica*, v. 35, n. 7, pp. 1504, 2017.

[46] YILMAZ, A., TEMELTAS, H., "Self-adaptive Monte Carlo method for indoor localization of smart AGVs using LIDAR data", *Robotics and Autonomous Systems*, v. 122, pp. 103285, 2019.

[47] MORENO, L., MARTÍN, F., MUÑOZ, M. L., et al., "Differential Evolution Markov Chain filter for global localization", *Journal of Intelligent & Robotic Systems*, v. 82, n. 3-4, pp. 513–536, 2016.

[48] PINTO, A. M., MOREIRA, A. P., COSTA, P. G., "A localization method based on map-matching and particle swarm optimization", *Journal of Intelligent & Robotic Systems*, v. 77, n. 2, pp. 313–326, 2015.

[49] ZHANG, Q.-B., WANG, P., CHEN, Z.-H., "An improved particle filter for mobile robot localization based on particle swarm optimization", *Expert Systems with Applications*, v. 135, pp. 181–193, 2019.

[50] ELFES, A., "Sonar-based real-world mapping and navigation", *IEEE Journal on Robotics and Automation*, v. 3, n. 3, pp. 249–265, 1987.

[51] NEDJAH, N., MACEDO MOURELLE, L., ALBUQUERQUE DE OLIVEIRA, P. J., "Simultaneous localization and mapping using swarm intelligence based methods", *Expert Systems with Applications*, v. 159, pp. 113547, 2020.

[52] MARTÍN, F., MORENO, L., GARRIDO, S., et al., "Kullback-Leibler divergence-based differential evolution Markov Chain filter for global localization of mobile robots", *Sensors*, v. 15, n. 9, pp. 23431–23458, 2015.

[53] PENG, G., ZHENG, W., LU, Z., et al., "An improved AMCL algorithm based on laser scanning match in a complex and unstructured environment", *Complexity*, v. 2018, 2018.

[54] GRISETTI, G., STACHNISS, C., BURGARD, W., "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters", *IEEE Transactions on Robotics*, v. 23, n. 1, pp. 34–46, 2007.

[55] SIEGWART, R., NOURBAKHSH, I. R., SCARAMUZZA, D., *Introduction to autonomous mobile robots*. MIT press, 2011.

[56] MALIS, E., CHAUMETTE, F., BOUDET, S., "2 1/2 D visual servoing", *IEEE Transactions on Robotics and Automation*, v. 15, n. 2, pp. 238–250, 1999.

[57] SICK, *LMS1xx Product Information*, 2016, Rev. 8012468/2016-01-14.

[58] BRESENHAM, J. E., "Algorithm for computer control of a digital plotter", *IBM Systems journal*, v. 4, n. 1, pp. 25–30, 1965.

[59] THRUN, S., "A probabilistic on-line mapping algorithm for teams of mobile robots", *The International Journal of Robotics Research*, v. 20, n. 5, pp. 335–363, 2001.

[60] PEDROSA, E., PEREIRA, A., LAU, N., "Efficient localization based on scan matching with a continuous likelihood field". In: *2017 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pp. 61–66, 2017.

[61] THRUN, S. B., "Exploration and model building in mobile robot domains". In: *IEEE international conference on neural networks*, pp. 175–180, 1993.

[62] KALMAN, R. E., "A new approach to linear filtering and prediction problems", *Journal of Basic Engineering*, 1960.

[63] METROPOLIS, N., ULAM, S., "The monte carlo method", *Journal of the American statistical association*, v. 44, n. 247, pp. 335–341, 1949.

[64] SORENSON, H. W., "Least-squares estimation: from Gauss to Kalman", *IEEE spectrum*, v. 7, n. 7, pp. 63–68, 1970.

[65] BROWN, R. G., HWANG, P. Y., *Introduction to random signals and applied Kalman filtering: with MATLAB exercises and solutions*. John Wiley & Sons, Inc.: Hoboken, NJ, 2012.

[66] WELCH, G., BISHOP, G., *An introduction to the Kalman filter*, Tech. rep., University of North Carolina at Chapel Hill, 11 1995.

[67] THRUN, S., FOX, D., BURGARD, W., et al., "Robust Monte Carlo localization for mobile robots", *Artificial intelligence*, v. 128, n. 1-2, pp. 99–141, 2001.

[68] FOX, D., "KLD-sampling: Adaptive particle filters and mobile robot localization", *Advances in neural information processing systems (NIPS)*, v. 14, n. 1, pp. 26–32, 2001.

[69] RIEDMILLER, M., BRAUN, H., "A direct adaptive method for faster backpropagation learning: The RPROP algorithm". In: *IEEE international conference on neural networks*, pp. 586–591, 1993.

[70] OPENCV TEAM, "OpenCV: Sobel Derivatives", `https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html`, 2022.

[71] CAMPBELL, S., O'MAHONY, N., CARVALHO, A., et al., "Where am I? Localization techniques for Mobile Robots A Review". In: *2020 6th International Conference on Mechatronics and Robotics Engineering (ICMRE)*, pp. 43–47, 2020.

[72] WU, Z., ZHANG, J., YUE, Y., et al., "Infrastructure-Free Global Localization in Repetitive Environments: An Overview". In: *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, pp. 626–631, 2020.

[73] MINGUEZ, J., MONTESANO, L., LAMIRAUX, F., "Metric-based iterative closest point scan matching for sensor displacement estimation", *IEEE Transactions on Robotics*, v. 22, n. 5, pp. 1047–1054, 2006.

[74] KONECNY, J., PRAUZEK, M., HLAVICA, J., "ICP algorithm in mobile robot navigation: Analysis of computational demands in embedded solutions", *IFAC-PapersOnLine*, v. 49, n. 25, pp. 396–400, 2016.

[75] SE, S., LOWE, D., LITTLE, J., "Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks", *The international Journal of robotics Research*, v. 21, n. 8, pp. 735–758, 2002.

[76] BOURAINE, S., BOUGOUFFA, A., AZOUAOUI, O., "Particle swarm optimization for solving a scan-matching problem based on the normal distributions transform", *Evolutionary Intelligence*, pp. 1–12, 2021.

[77] ABUALKEBASH, H., HASAN, O., "Improved Global Localization and Resampling Techniques for Monte Carlo Localization Algorithm", *International Journal of Applied Mathematics Electronics and Computers*, v. 8, n. 3, pp. 102–108, 2020.

[78] CHIEN, C.-H., HSU, C.-C., WANG, W.-Y., et al., "Global localization of Monte Carlo localization based on multi-objective particle swarm optimization". In: *2016 IEEE 6th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, pp. 96–97, 2016.

[79] CHIEN, C.-H., WANG, W.-Y., HSU, C.-C., "Multi-objective evolutionary approach to prevent premature convergence in Monte Carlo localization", *Applied Soft Computing*, v. 50, pp. 260–279, 2017.

[80] ZHANG, Q., WANG, P., BAO, P., et al., "Mobile Robot Global Localization Using Particle Swarm Optimization with a 2D Range Scan". In: *Proceedings of the 2017 International Conference on Robotics and Artificial Intelligence*, pp. 105–109, 2017.

[81] VAHDAT, A. R., NOURASHRAFODDIN, N., GHIDARY, S. S., "Mobile robot global localization using differential evolution and particle swarm optimization". In: *2007 IEEE Congress on Evolutionary Computation*, pp. 1527–1534, 2007.

[82] NETO, W. A., PINTO, M. F., MARCATO, A. L., et al., "Mobile robot localization based on the novel leader-based bat algorithm", *Journal of Control, Automation and Electrical Systems*, v. 30, n. 3, pp. 337–346, 2019.

[83] HOLLAND, J. H., *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT Press: Cambridge, 1992.

[84] GAO, H., ZHANG, X., YUAN, J., et al., "A Novel Global Localization Approach Based on Structural Unit Encoding and Multiple Hypothesis Tracking", *IEEE Transactions on Instrumentation and Measurement*, v. 68, n. 11, pp. 4427–4442, 2019.

[85] KENNEDY, J., EBERHART, R., "Particle swarm optimization". In: *Proceedings of ICNN'95 - International Conference on Neural Networks*, v. 4, pp. 1942–1948 vol.4, 1995.

[86] EBERHART, R., SHI, Y., "Comparing inertia weights and constriction factors in particle swarm optimization". In: *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*, v. 1, pp. 84–88 vol.1, 2000.

[87] STORN, R., PRICE, K., "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces", *Journal of global optimization*, v. 11, n. 4, pp. 341–359, 1997.

[88] PRICE, K., STORN, R. M., LAMPINEN, J. A., *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media: Berlin, 2006.

[89] SCHARR, H., *Optimal operators in digital image processing*, Ph.D. Thesis, University of Heidelberg, Germany, 2000.

[90] HOWARD, A., ROY, N., "The Robotics Data Set Repository (Radish)", `http://radish.sourceforge.net/`, 2003.

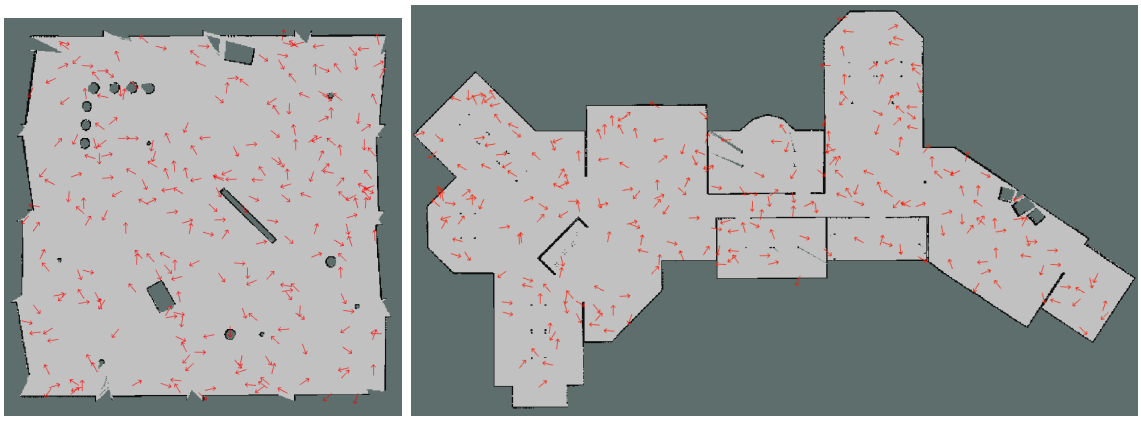
Apêndice A

Inicialização das hipóteses na localização global

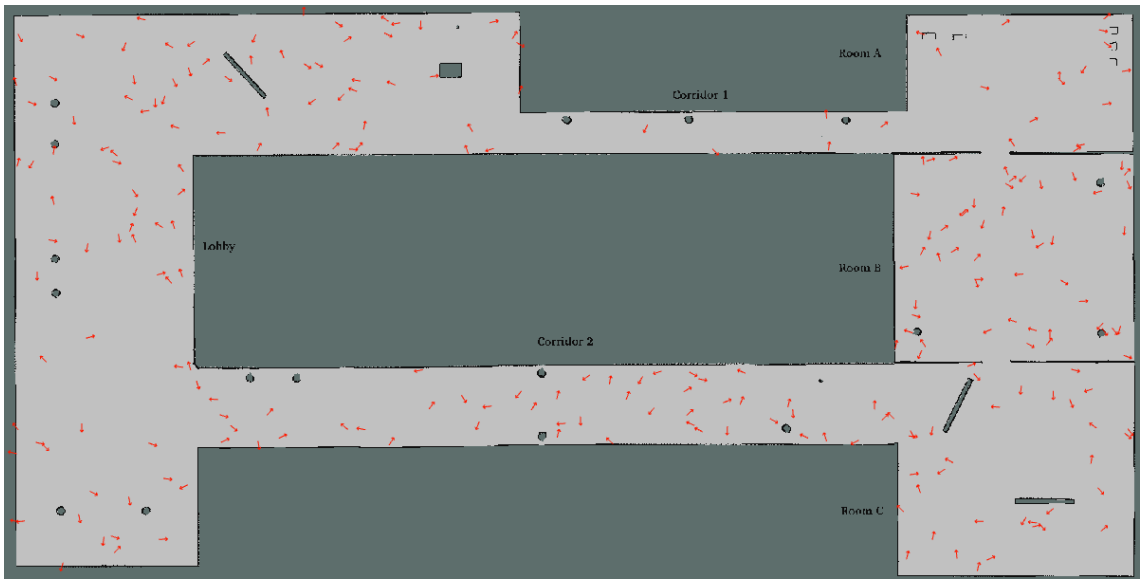
Este apêndice apresenta a forma de inicialização de hipóteses para a localização global. As Figs. A.1, A.2 e A.3 mostram as populações iniciais para populações de tamanho $P_{\max} = [250, 2500, 5000]$, nesta ordem. As ilustrações (a) a (d) mostram, nesta ordem, os seguintes mapas com as respectivas áreas representadas:

- Playpen, 422.24 m^2
- CPR Office, 948.48 m^2
- Warehouse, 2987.52 m^2
- Laboratório de Robótica, 235.64 m^2

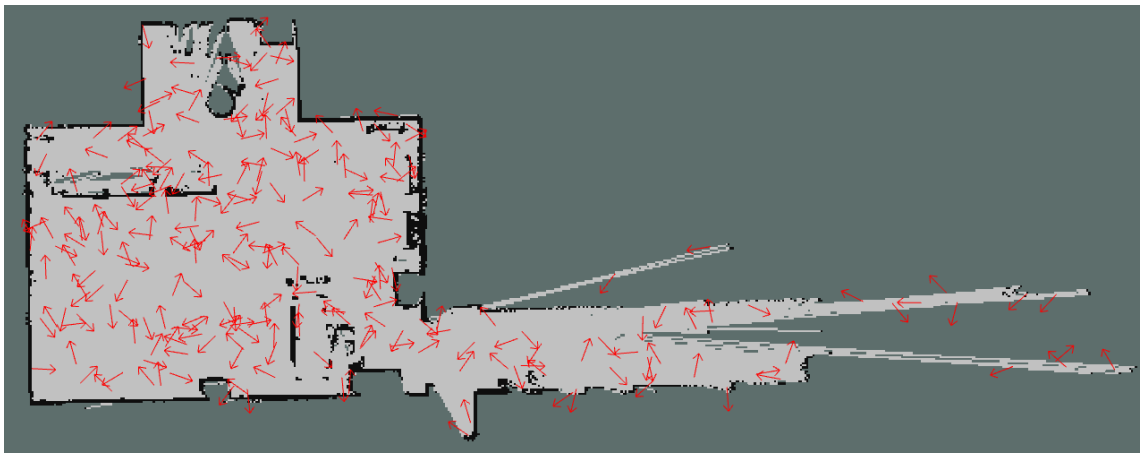
As hipóteses de poses são geradas aleatoriamente utilizando uma distribuição de probabilidade uniforme dentro dos limites do mapa. Porém, aquelas que caem sobre uma célula ocupada ou indefinida do mapa são recriadas aleatoriamente, até que caiam sobre uma célula livre. Observa-se nas Figs. A.1, A.2 e A.3 as diferentes densidades de hipóteses nos mapas.



(a) Playpen (b) CPR Office

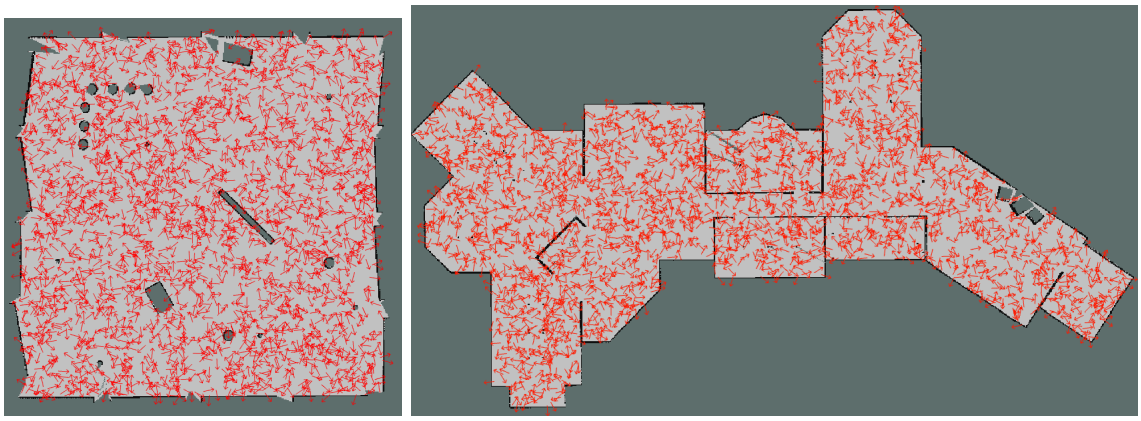


(c) Warehouse

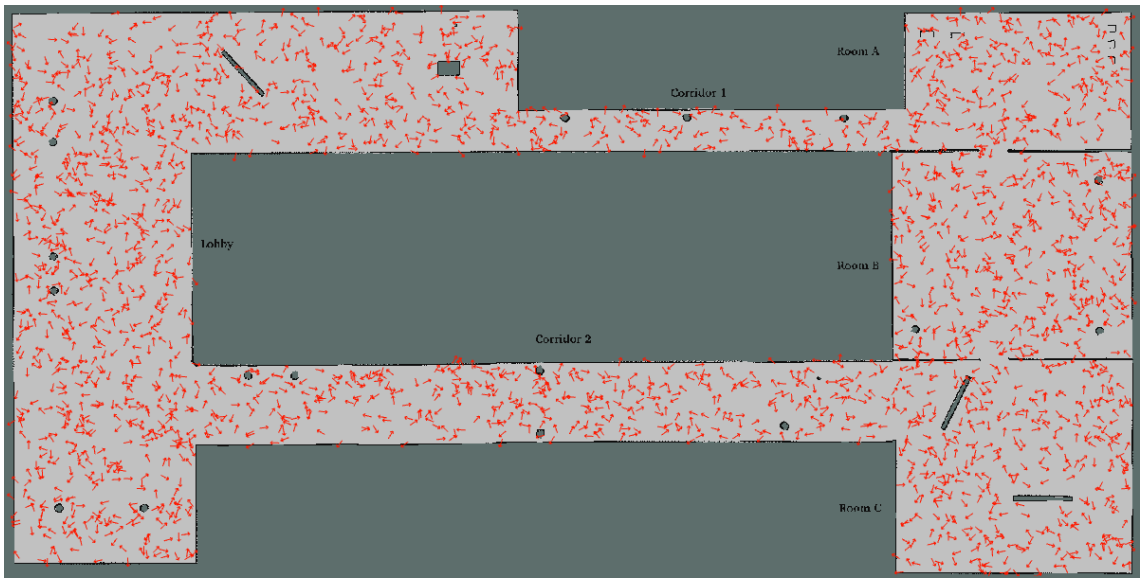


(d) Laboratório de Robótica

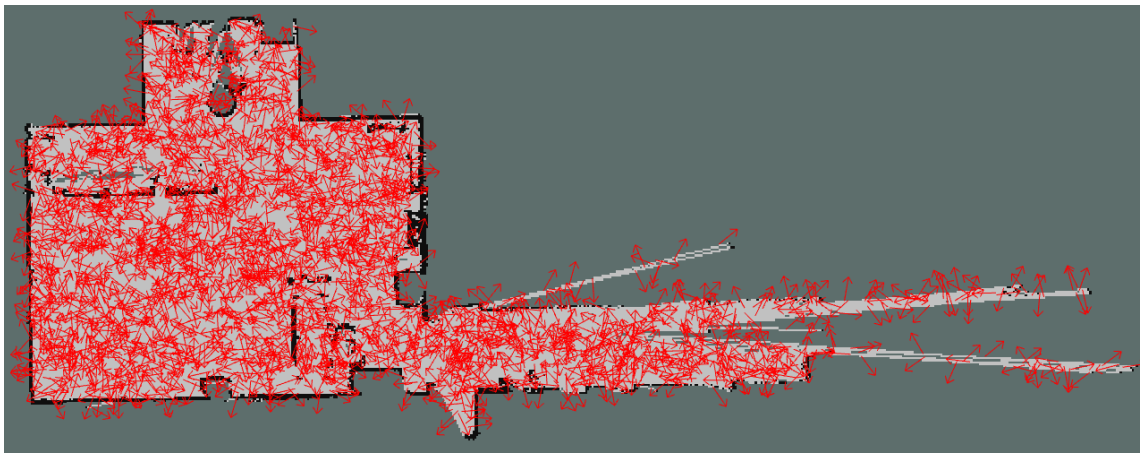
Figura A.1: População inicial em diferentes mapas ($P_{\max}$ =250)



(a) Playpen (b) CPR Office

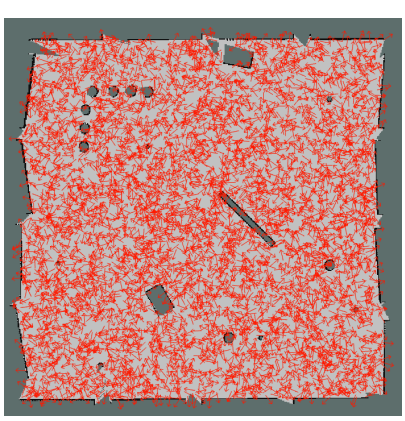


(c) Warehouse

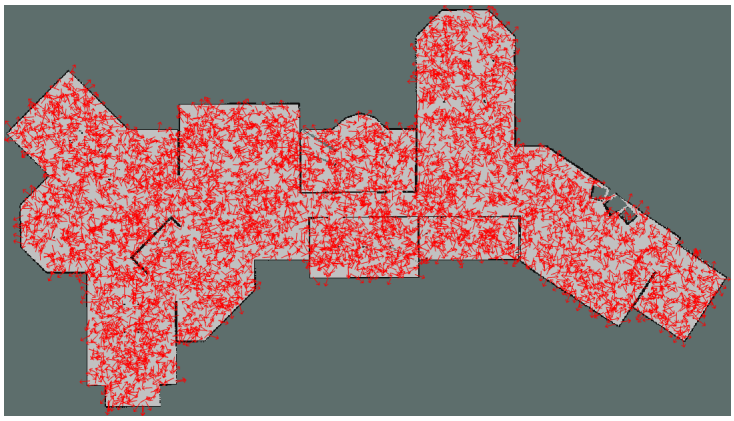


(d) Laboratório de Robótica

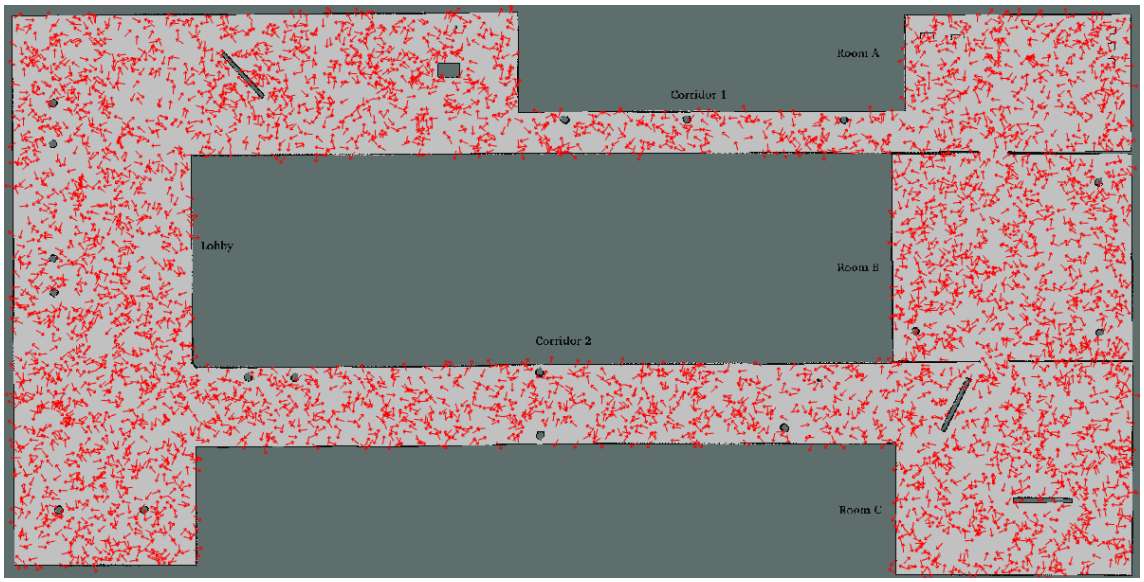
Figura A.2: População inicial em diferentes mapas ($P_{\max}$ =2500)



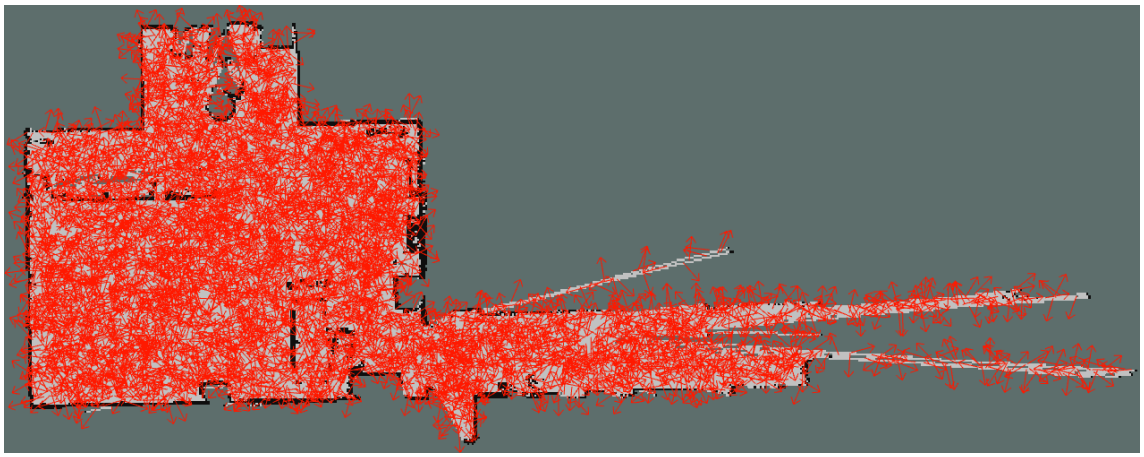
(a) Playpen



(b) CPR Office



(c) Warehouse



(d) Laboratório de Robótica

Figura A.3: População inicial em diferentes mapas ($P_{\max}$ =5000)

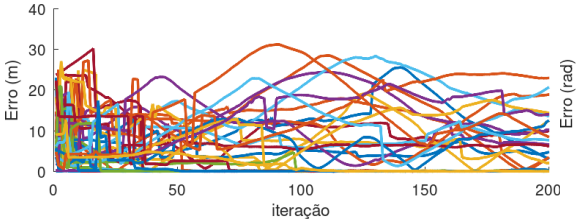
Apêndice B

Erro durante as iterações

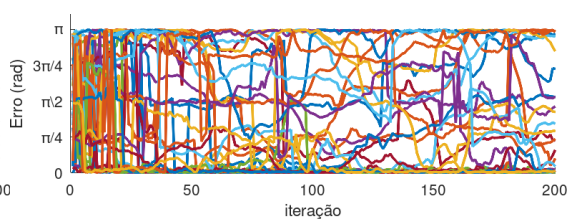
As figuras a seguir mostram os erros linear (em metros) e angular (em radianos) das poses estimadas pelos métodos AMCL, PMGL, PSGL, DEGL e GAGL, nesta ordem, ao longo das 200 primeiras iterações. Cada uma das 50 séries mostradas em diferentes cores representa uma tentativa de localização global no ambiente CPR Office. Em cada figura, os gráficos (a) a (l) mostram as tentativas de localização para diferentes tamanhos de população, em ordem crescente: $P_{\max} = [500, 1500, 2500, 3500, 4500, 5000]$.

Aqueles gráficos com menos curvas visíveis mostram métodos com maior taxa de sucesso. Deve-se lembrar que o sucesso é definido pelo valor baixo de ambos os erros, linear e angular, sustentado por algumas iterações. Alguns picos observados nos gráficos representam desvios pontuais da hipótese escolhida como pose estimada, mas que rapidamente são corrigidos pelo próprio método. Portanto, estes desvios, desde que curtos, não comprometem o sucesso da localização global.

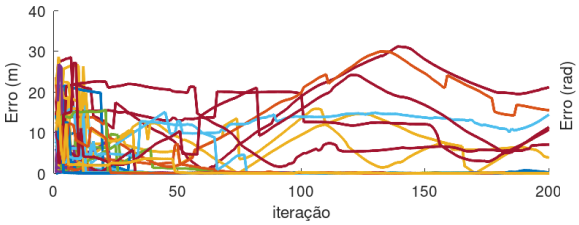
No caso do GAGL, por exemplo, há uma grande quantidade de desvios dos erros linear e angular para valores altos mesmo com tamanhos de população maiores, apesar de ser um método de elevada taxa de sucesso, conforme apresentado no Capítulo 5. O mesmo comportamento não é observado em outros métodos com alta taxa de sucesso, como o AMCL, PMGL e PSGL. Deduz-se, então, que o GAGL é mais suscetível a estes desvios momentâneos, porém ainda assim consegue manter a estimativa da pose correta no longo prazo. Nesse sentido, futuros aprimoramentos do GAGL podem incluir um mecanismo que evita tal comportamento, garantindo maior estabilidade em relação à pose estimada.



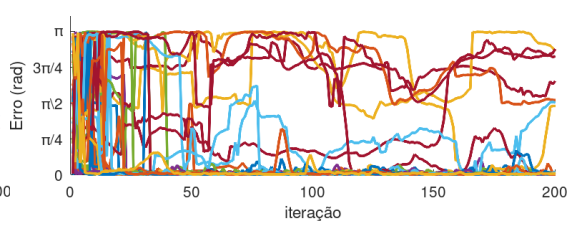
(a) Erro linear, $P_{\max}$ =500



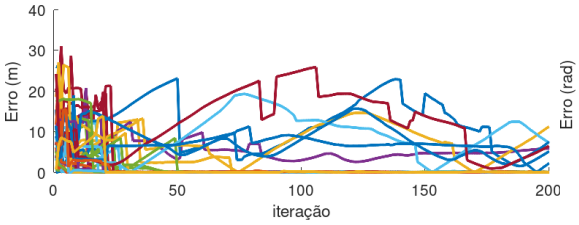
(b) Erro angular, $P_{\max}$ =500



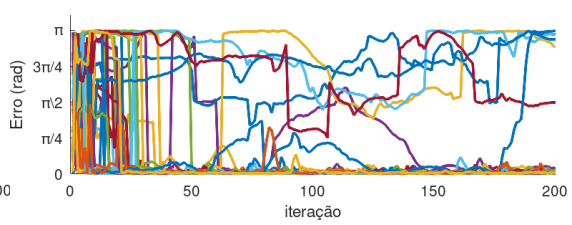
(c) Erro linear, $P_{\max}$ =1500



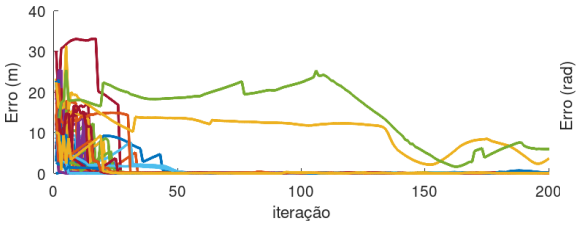
(d) Erro angular, $P_{\max}$ =1500



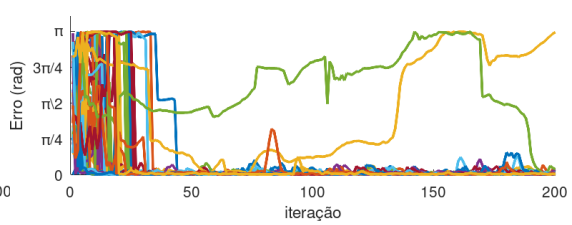
(e) Erro linear, $P_{\max}$ =2500



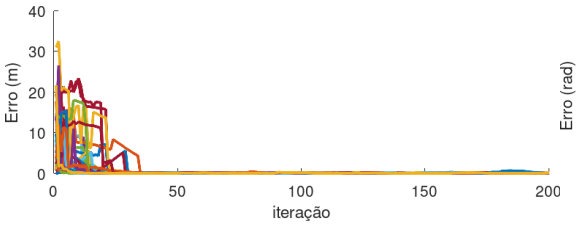
(f) Erro angular, $P_{\max}$ =2500



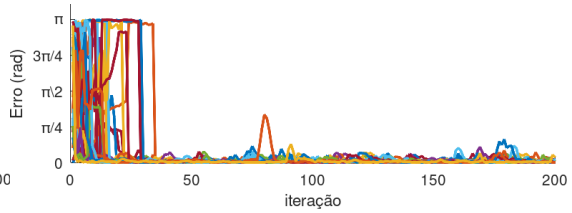
(g) Erro linear, $P_{\max}$ =3500



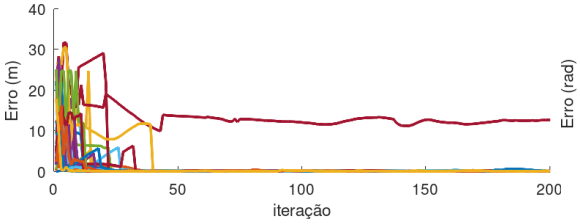
(h) Erro angular, $P_{\max}$ =3500



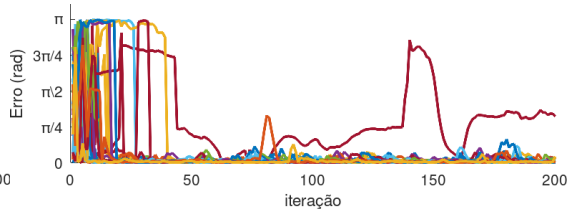
(i) Erro linear, $P_{\max}$ =4500



(j) Erro angular, $P_{\max}$ =4500

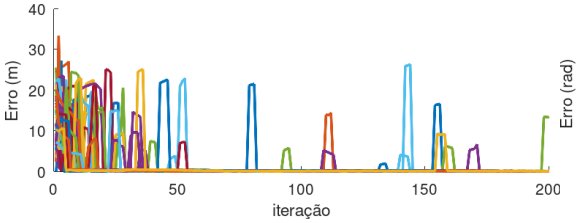


(k) Erro linear, $P_{\max}$ =5000

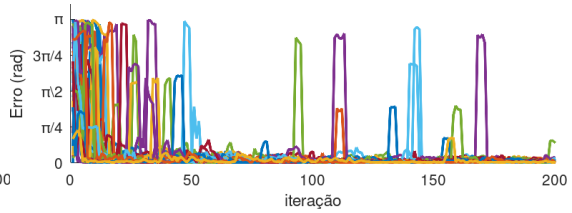


(l) Erro angular, $P_{\max}$ =5000

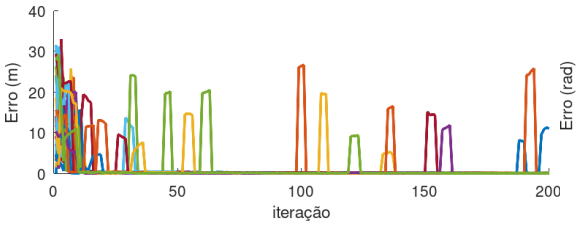
Figura B.1: AMCL, CPR Office $P_{\max}$ =[500,1500,2500,3500,4500,5000]



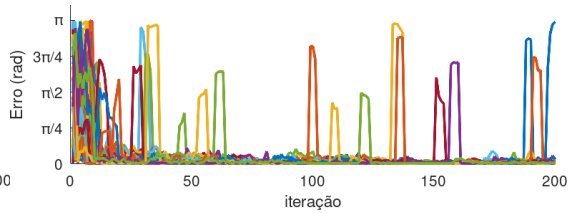
(a) Erro linear, $P_{\max}$ =500



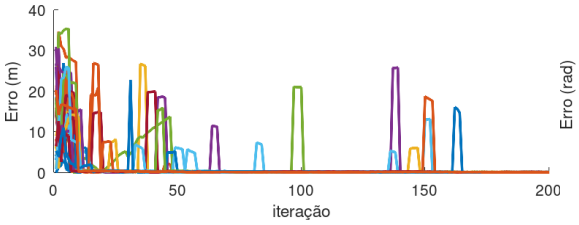
(b) Erro angular, $P_{\max}$ =500



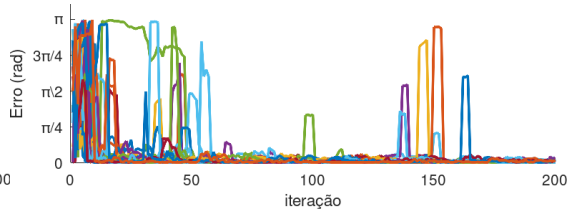
(c) Erro linear, $P_{\max}$ =1500



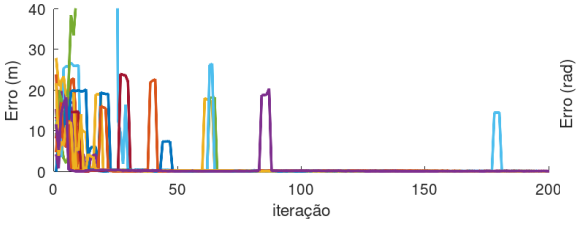
(d) Erro angular, $P_{\max}$ =1500



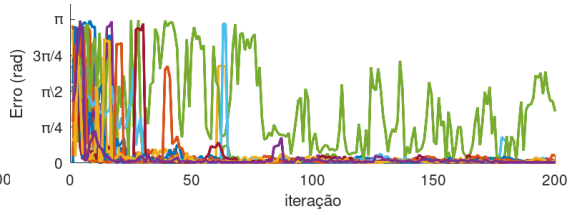
(e) Erro linear, $P_{\max}$ =2500



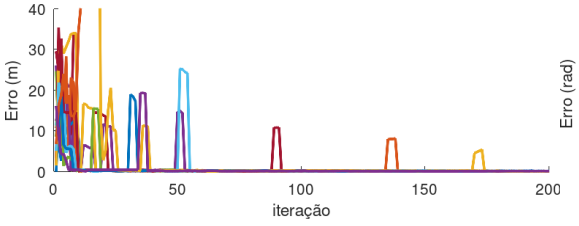
(f) Erro angular, $P_{\max}$ =2500



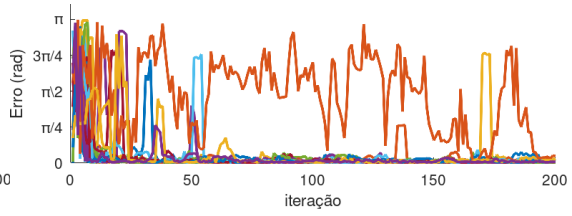
(g) Erro linear, $P_{\max}$ =3500



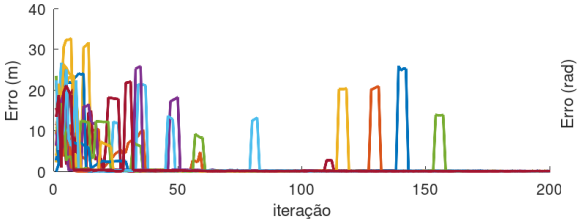
(h) Erro angular, $P_{\max}$ =3500



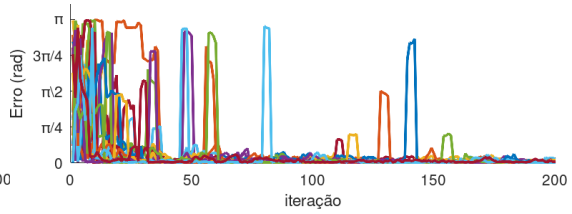
(i) Erro linear, $P_{\max}$ =4500



(j) Erro angular, $P_{\max}$ =4500

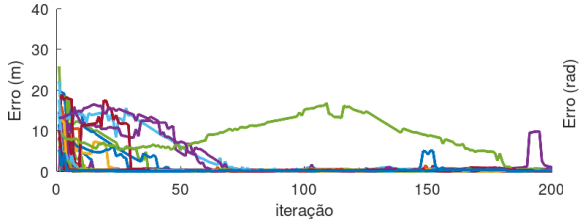


(k) Erro linear, $P_{\max}$ =5000

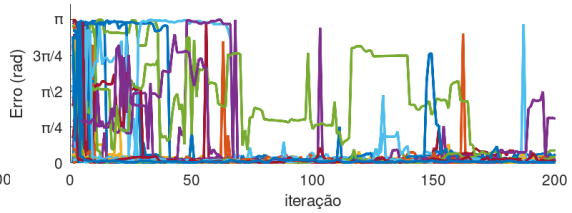


(l) Erro angular, $P_{\max}$ =5000

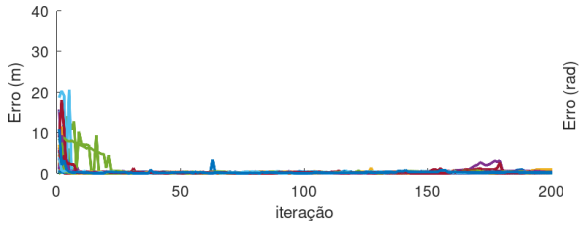
Figura B.2: PMGL, CPR Office $P_{\max}$ =[500,1500,2500,3500,4500,5000]



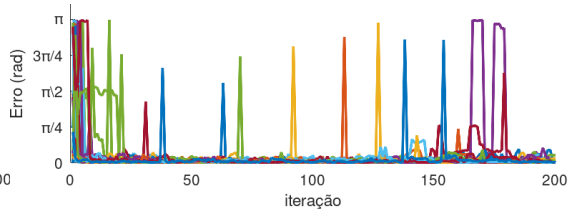
(a) Erro linear, $P_{\max}$ =500



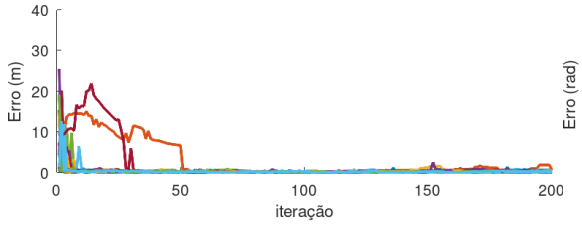
(b) Erro angular, $P_{\max}$ =500



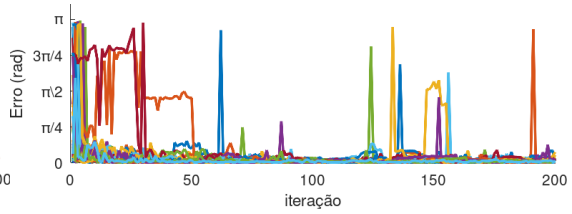
(c) Erro linear, $P_{\max}$ =1500



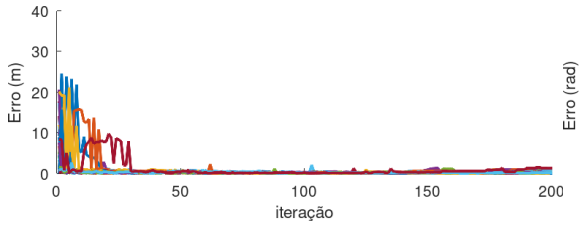
(d) Erro angular, $P_{\max}$ =1500



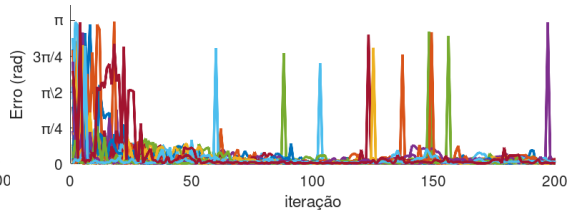
(e) Erro linear, $P_{\max}$ =2500



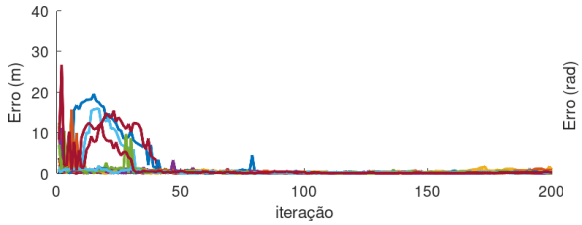
(f) Erro angular, $P_{\max}$ =2500



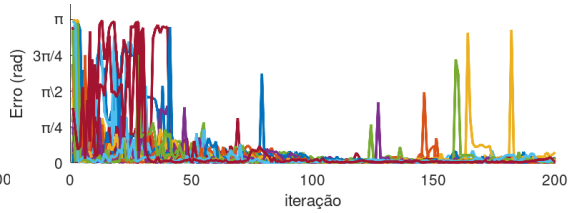
(g) Erro linear, $P_{\max}$ =3500



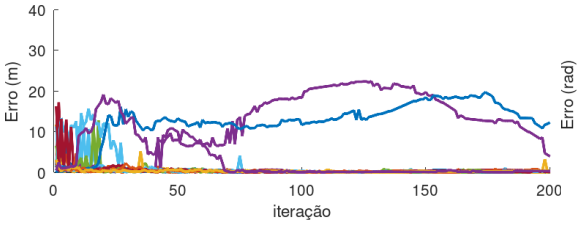
(h) Erro angular, $P_{\max}$ =3500



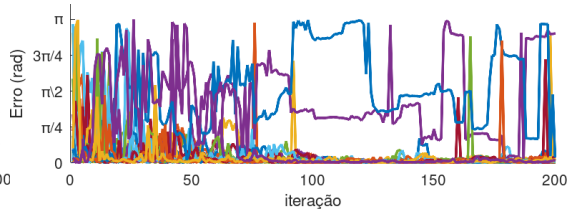
(i) Erro linear, $P_{\max}$ =4500



(j) Erro angular, $P_{\max}$ =4500

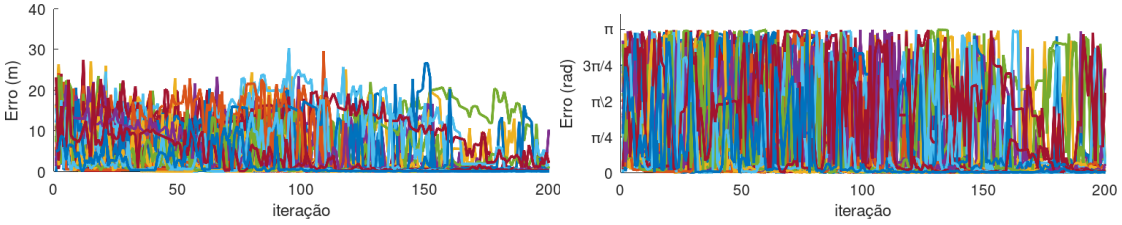


(k) Erro linear, $P_{\max}$ =5000

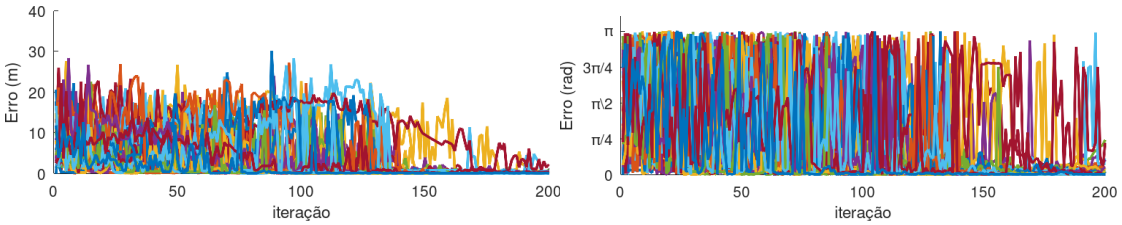


(l) Erro angular, $P_{\max}$ =5000

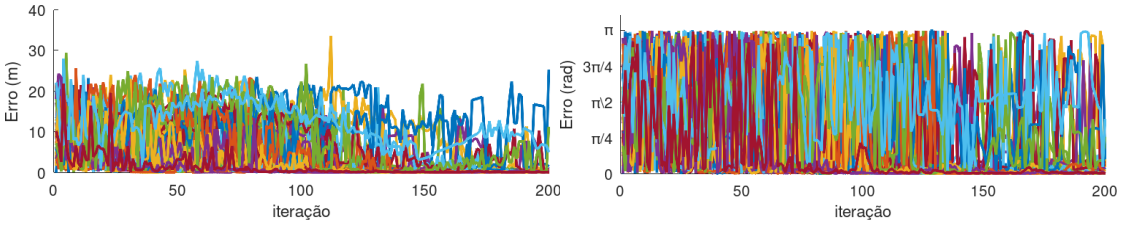
Figura B.3: PSGL, CPR Office $P_{\max}$ =[500,1500,2500,3500,4500,5000]



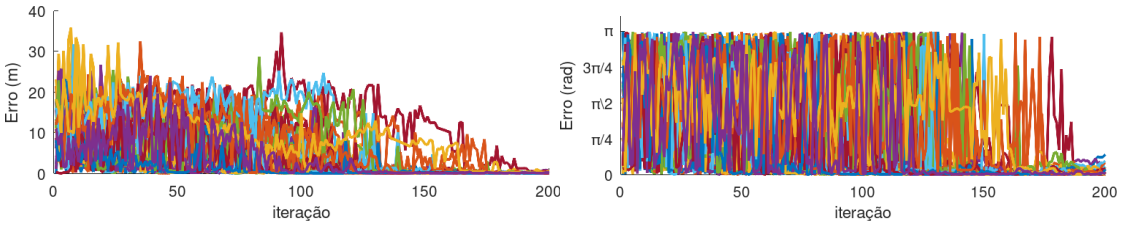
(a) Erro linear, $P_{\max}$ =500 (b) Erro angular, $P_{\max}$ =500



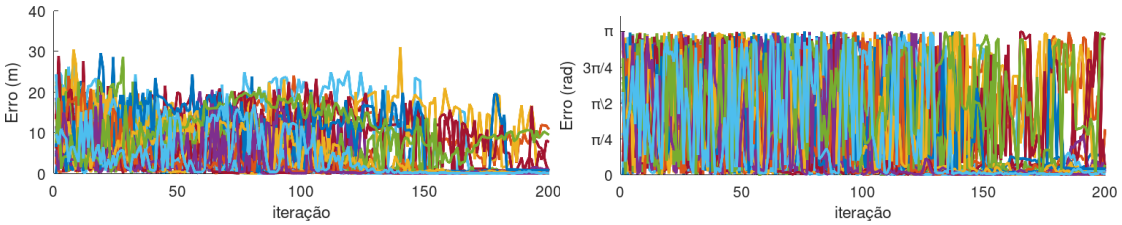
(c) Erro linear, $P_{\max}$ =1500 (d) Erro angular, $P_{\max}$ =1500



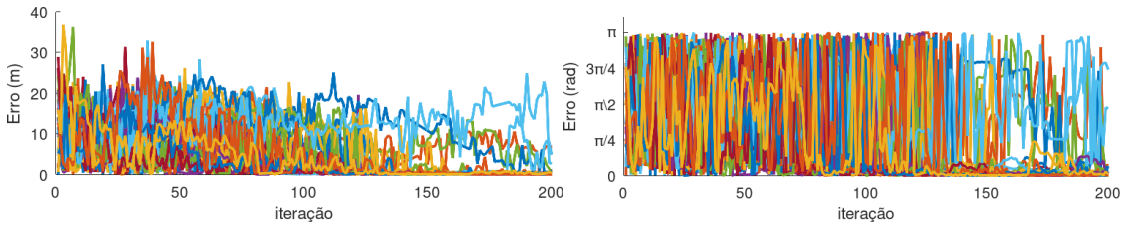
(e) Erro linear, $P_{\max}$ =2500 (f) Erro angular, $P_{\max}$ =2500



(g) Erro linear, $P_{\max}$ =3500 (h) Erro angular, $P_{\max}$ =3500

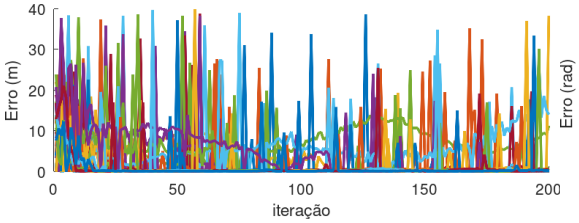


(i) Erro linear, $P_{\max}$ =4500 (j) Erro angular, $P_{\max}$ =4500

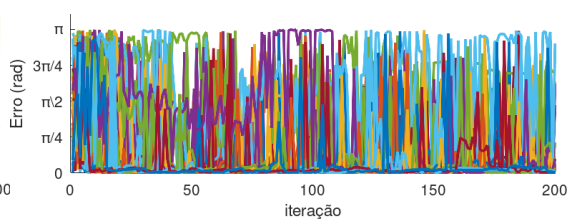


(k) Erro linear, $P_{\max}$ =5000 (l) Erro angular, $P_{\max}$ =5000

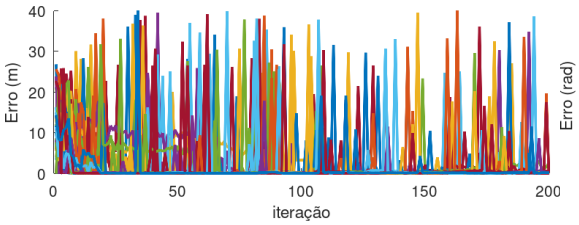
Figura B.4: DEGL, CPR Office $P_{\max}$ =[500,1500,2500,3500,4500,5000]



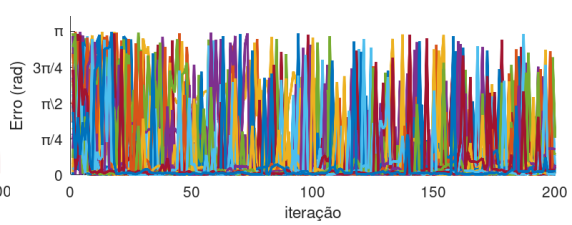
(a) Erro linear, $P_{\max}$ =500



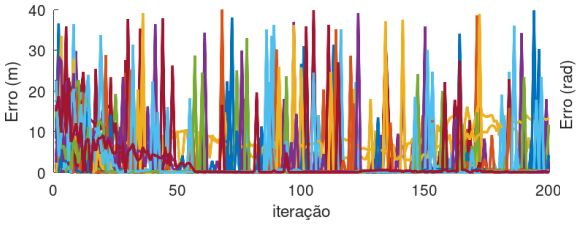
(b) Erro angular, $P_{\max}$ =500



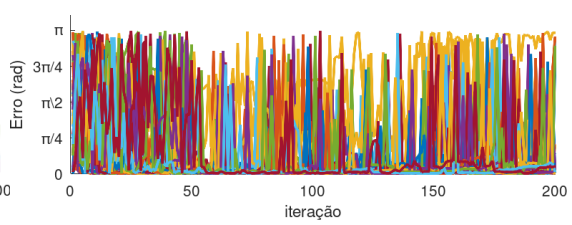
(c) Erro linear, $P_{\max}$ =1500



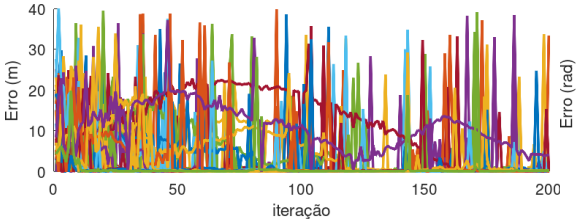
(d) Erro angular, $P_{\max}$ =1500



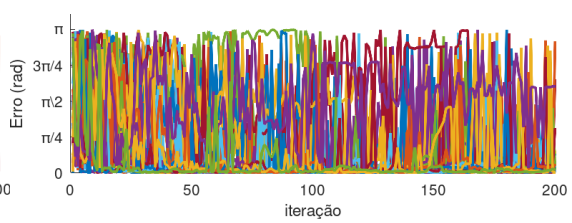
(e) Erro linear, $P_{\max}$ =2500



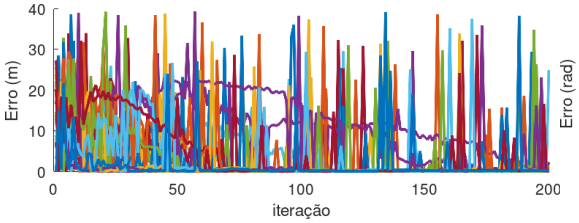
(f) Erro angular, $P_{\max}$ =2500



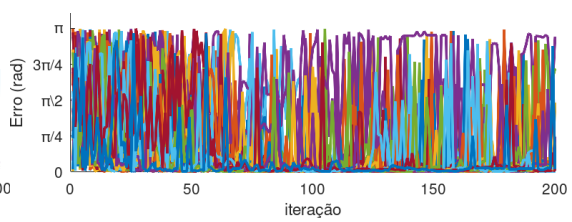
(g) Erro linear, $P_{\max}$ =3500



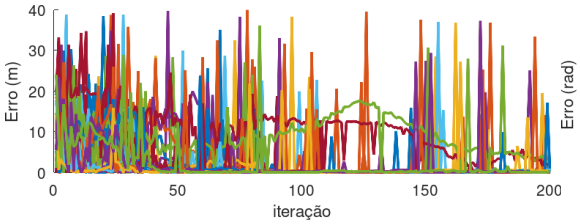
(h) Erro angular, $P_{\max}$ =3500



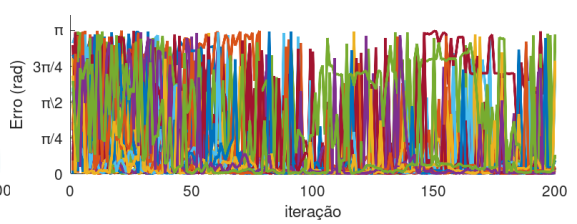
(i) Erro linear, $P_{\max}$ =4500



(j) Erro angular, $P_{\max}$ =4500



(k) Erro linear, $P_{\max}$ =5000



(l) Erro angular, $P_{\max}$ =5000

Figura B.5: GAGL, CPR Office $P_{\max}$ =[500,1500,2500,3500,4500,5000]

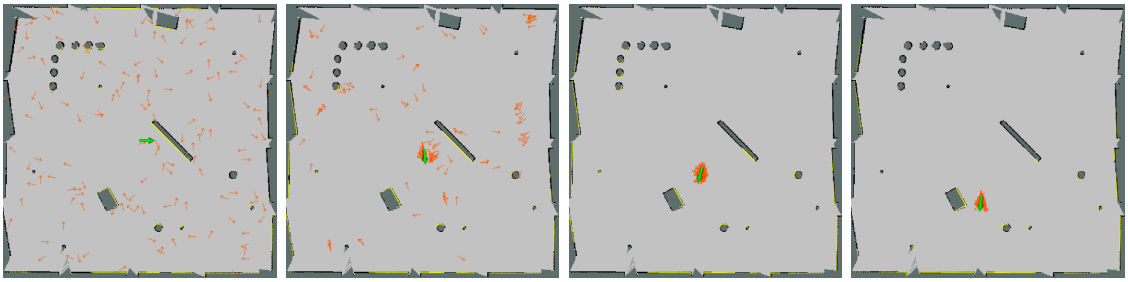
Apêndice C

Visualização da dinâmica das partículas em cada método

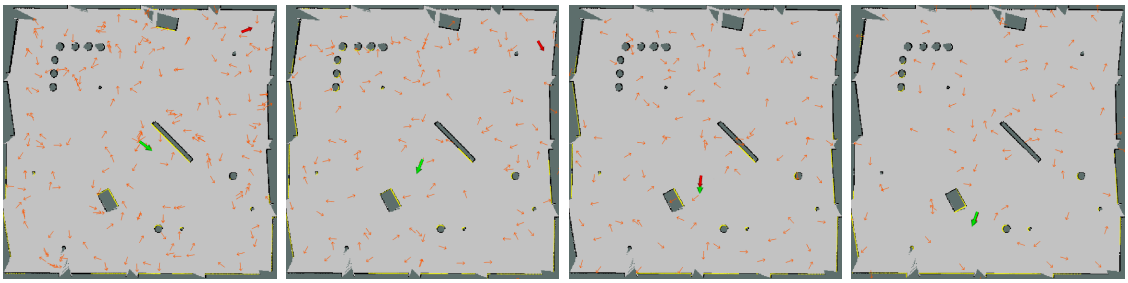
Este apêndice apresenta as figuras extraídas da simulação do procedimento de Localização Global para todos os algoritmos considerados. Esta simulação foi realizada no ambiente Playpen, utilizando o simulador Gazebo. As imagens das figuras foram obtidas a partir da captura de tela da ferramenta de visualização RViz, nos instantes indicados nas legendas. A iteração de cada método é identificada pela letra i

Todas as simulações foram realizadas utilizando o tamanho de população $P_{\max} = 250$, para melhor visualização. Cada imagem contém o mapa do Playpen e a nuvem de pontos obtida pelo sensor LiDAR (em amarelo, predominantemente sobre os contornos). As partículas de cada método, que compõem a população, são representadas por setas finas na cor laranja. No caso do PSGL, (i), (j), (k) e (l) em cada figura, o enxame é representado pelas setas em laranja e os $\mathbf{p}_{\text{best}}$'s são desenhados como setas finais azuis. As setas maiores em vermelho e verde representam o *ground thruth* e a pose estimada pelo método, respectivamente. Em algumas iterações não é possível visualizar a seta vermelha porque está sob o *ground thruth* ou encoberto pelo enxame.

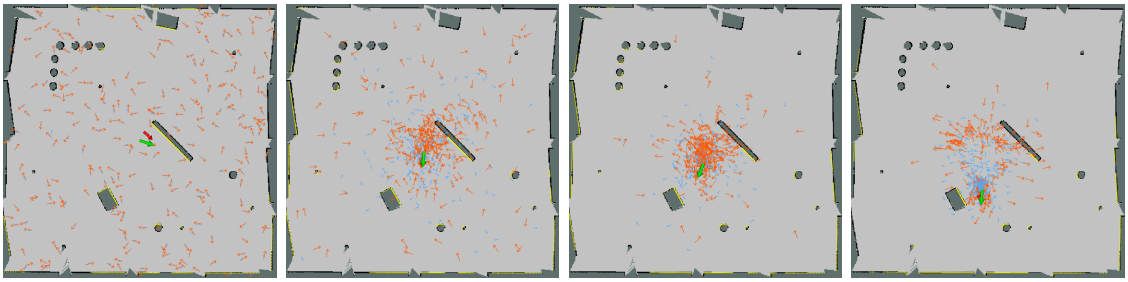
As figuras a seguir mostram o estado do método registrado a cada 10 iterações, a partir da primeira. Porém, as iterações 1, 5 e 10 são mostradas em sequência porque as primeiras iterações são críticas para a formação dos enxames de partículas em mínimos locais. Para melhor visualização, as figuras estão divididas em intervalos 4 de registros, formando uma matriz, onde as colunas exibem os diferentes métodos e as linhas reproduzem o estado em determinada iteração i.



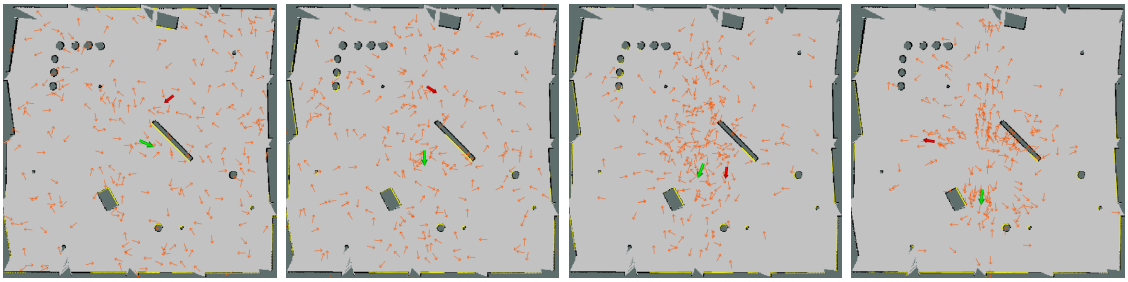
(a) AMCL, i =001 (b) AMCL, i =005 (c) AMCL, i =010 (d) AMCL, i =020



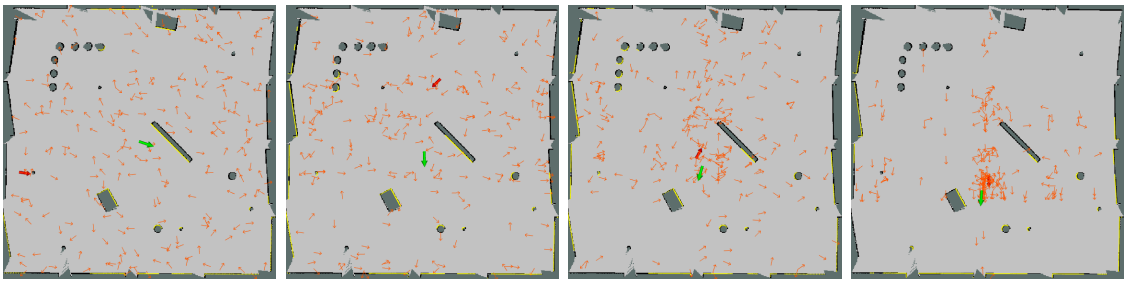
(e) PMGL, i =001 (f) PMGL, i =005 (g) PMGL, i =010 (h) PMGL, i =020



(i) PSGL, i =001 (j) PSGL, i =005 (k) PSGL, i =010 (l) PSGL, i =020

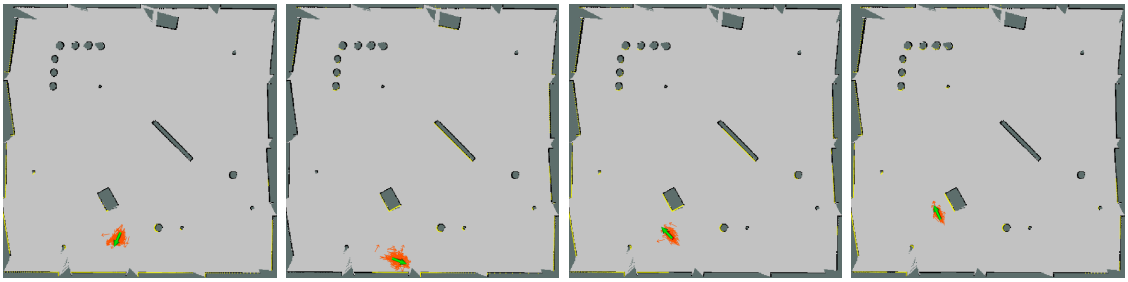


(m) DEGL, i =001 (n) DEGL, i =005 (o) DEGL, i =010 (p) DEGL, i =020

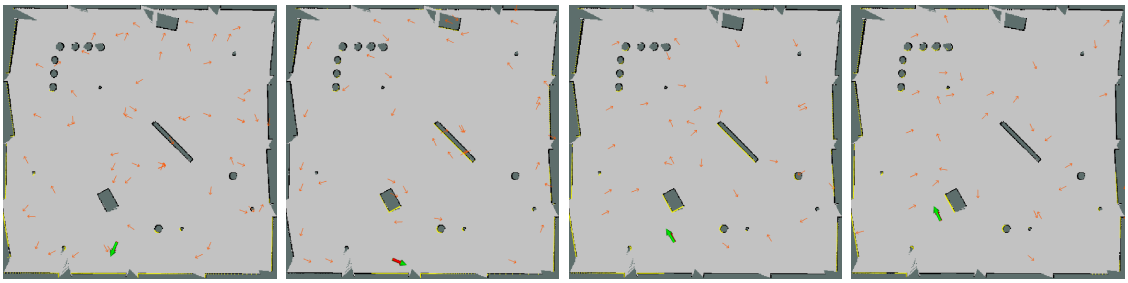


(q) GAGL, i =001 (r) GAGL, i =005 (s) GAGL, i =010 (t) GAGL, i =020

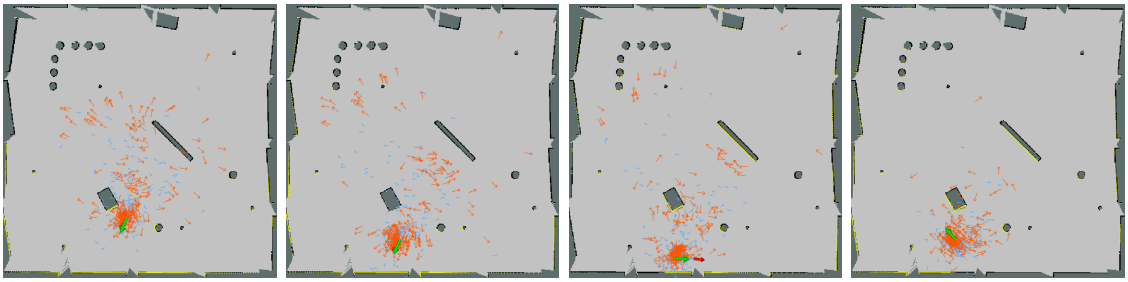
Figura C.1: Iterações [001,005,010,020] da localização global



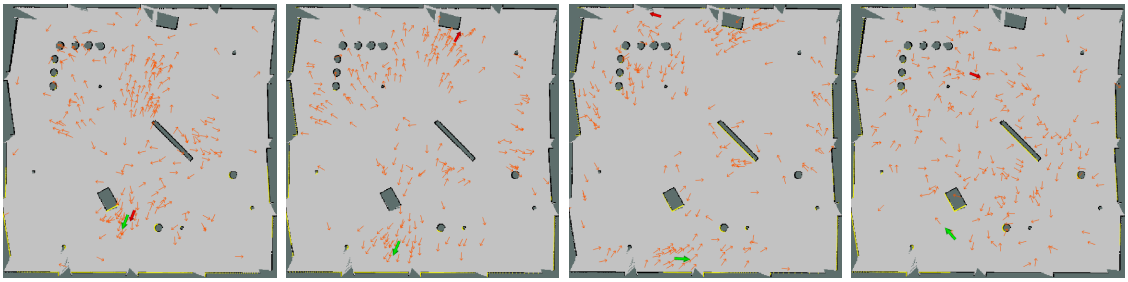
(a) AMCL, i =030 (b) AMCL, i =040 (c) AMCL, i =050 (d) AMCL, i =060



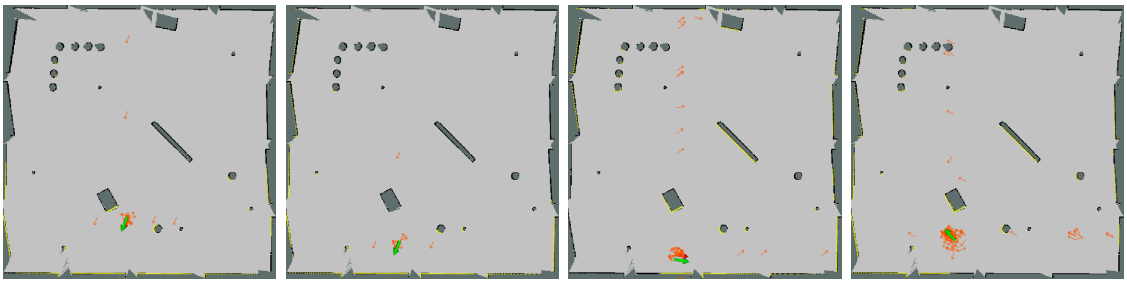
(e) PMGL, i =030 (f) PMGL, i =040 (g) PMGL, i =050 (h) PMGL, i =060



(i) PSGL, i =030 (j) PSGL, i =040 (k) PSGL, i =050 (l) PSGL, i =060

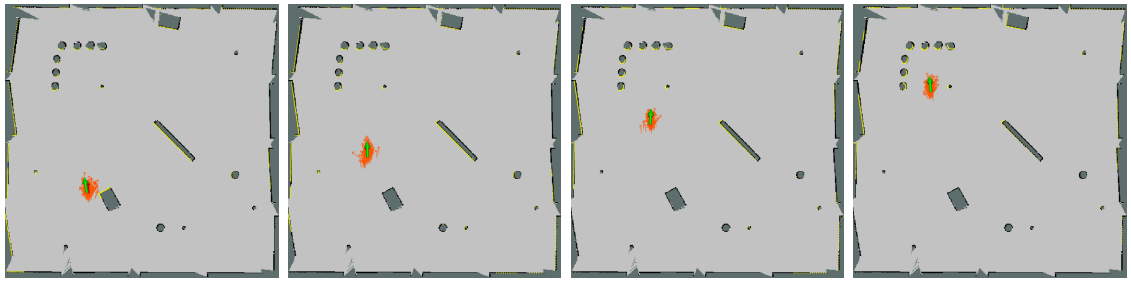


(m) DEGL, i =030 (n) DEGL, i =040 (o) DEGL, i =050 (p) DEGL, i =060

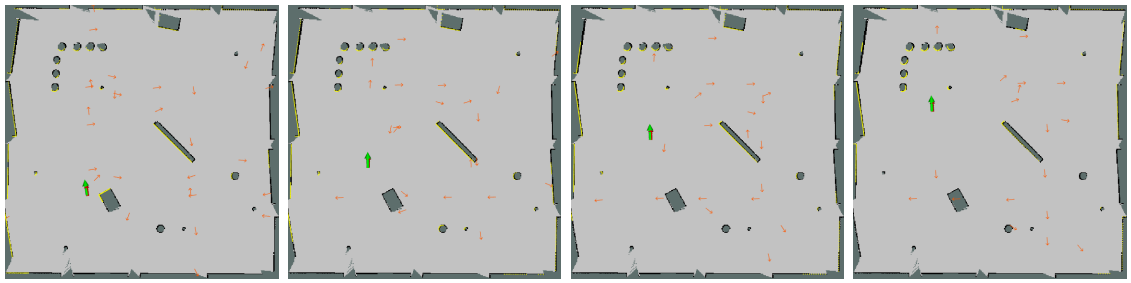


(q) GAGL, i =030 (r) GAGL, i =040 (s) GAGL, i =050 (t) GAGL, i =060

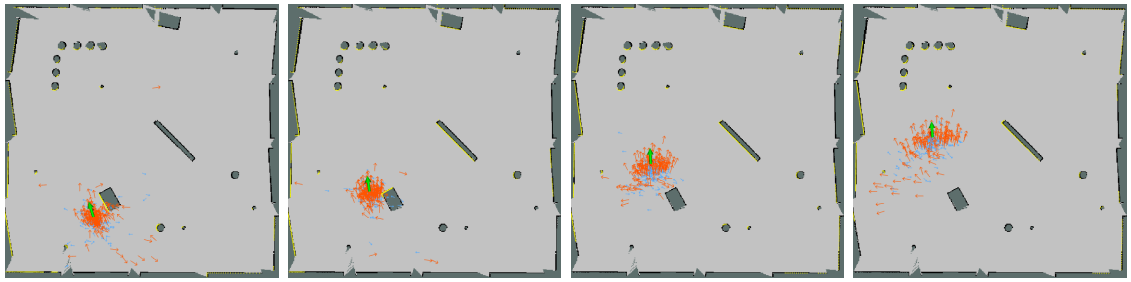
Figura C.2: Iterações [030,040,050,060] da localização global



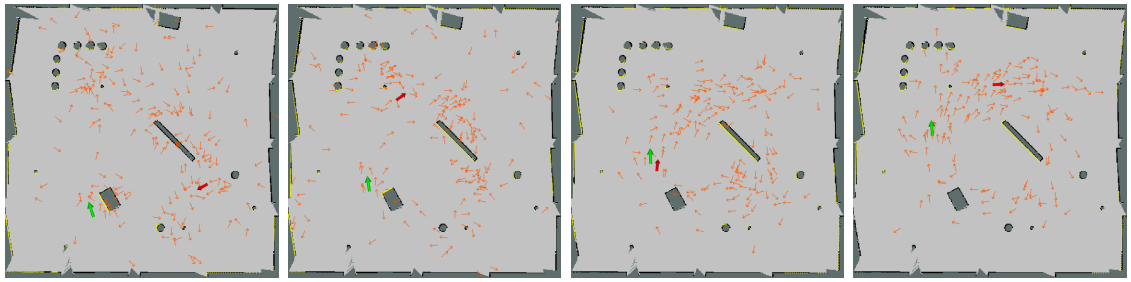
(a) AMCL, i =070 (b) AMCL, i =080 (c) AMCL, i =090 (d) AMCL, i =100



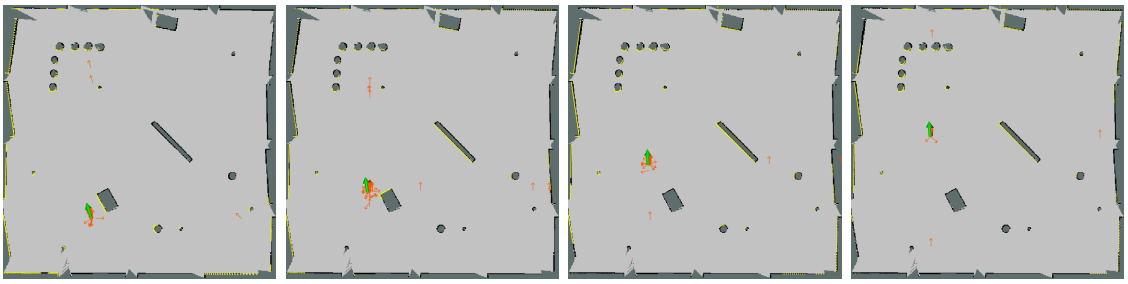
(e) PMGL, i =070 (f) PMGL, i =080 (g) PMGL, i =090 (h) PMGL, i =100



(i) PSGL, i =070 (j) PSGL, i =080 (k) PSGL, i =090 (l) PSGL, i =100

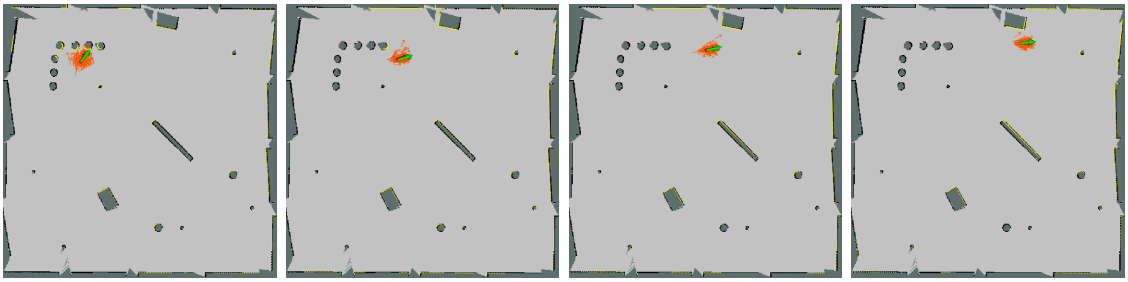


(m) DEGL, i =070 (n) DEGL, i =080 (o) DEGL, i =090 (p) DEGL, i =100

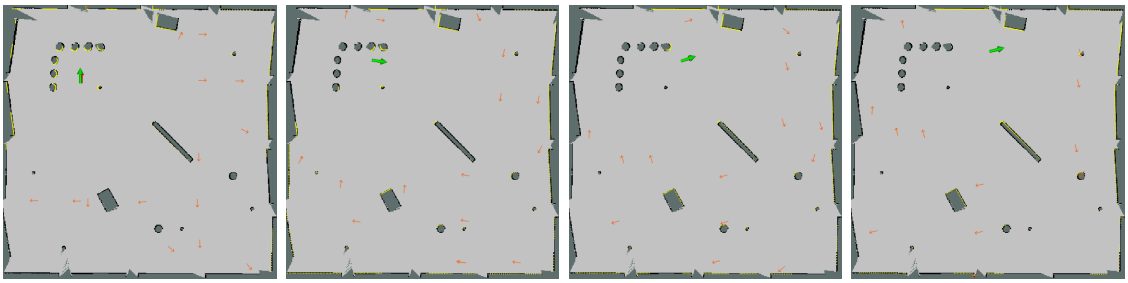


(q) GAGL, i =070 (r) GAGL, i =080 (s) GAGL, i =090 (t) GAGL, i =100

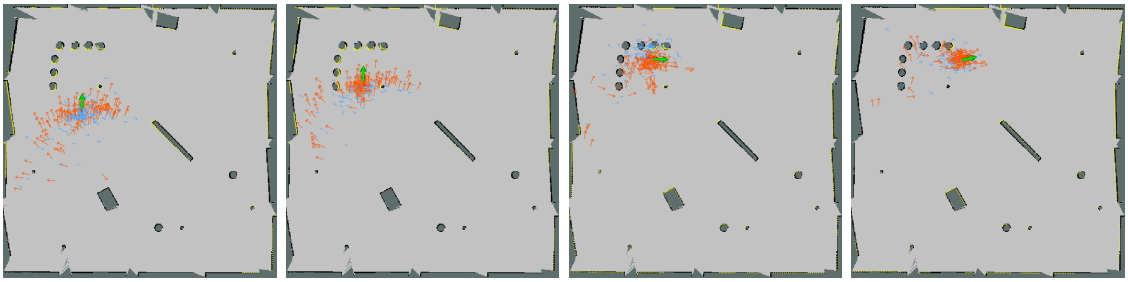
Figura C.3: Iterações [070,080,090,100] da localização global



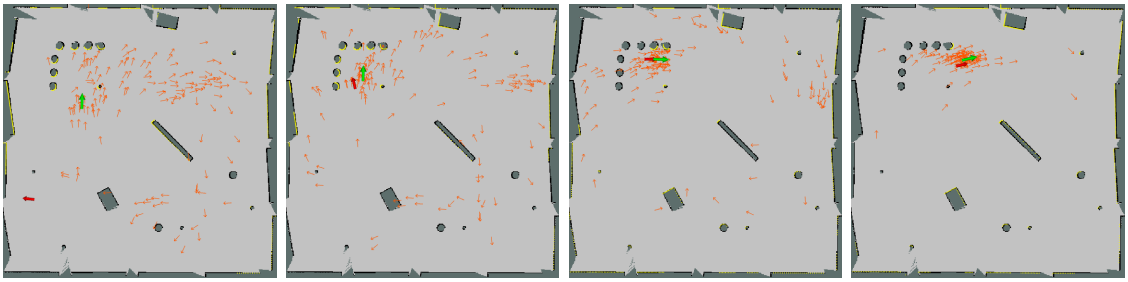
(a) AMCL, i =110 (b) AMCL, i =120 (c) AMCL, i =130 (d) AMCL, i =140



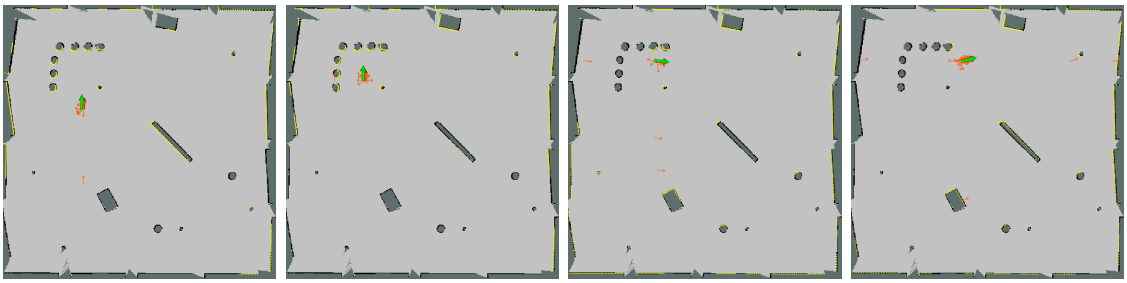
(e) PMGL, i =110 (f) PMGL, i =120 (g) PMGL, i =130 (h) PMGL, i =140



(i) PSGL, i =110 (j) PSGL, i =120 (k) PSGL, i =130 (l) PSGL, i =140

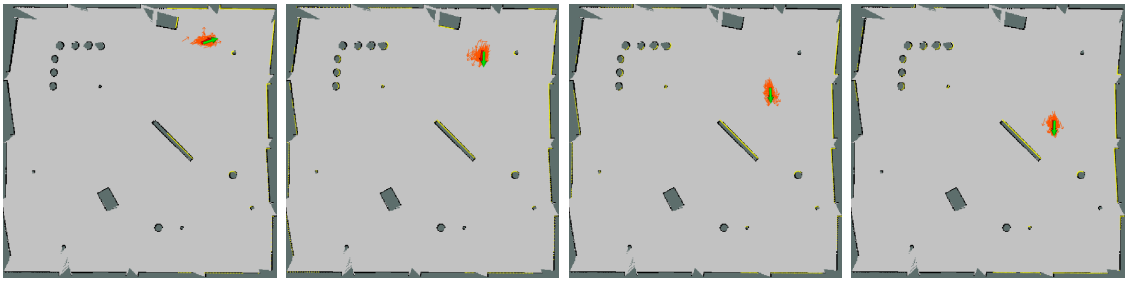


(m) DEGL, i =110 (n) DEGL, i =120 (o) DEGL, i =130 (p) DEGL, i =140

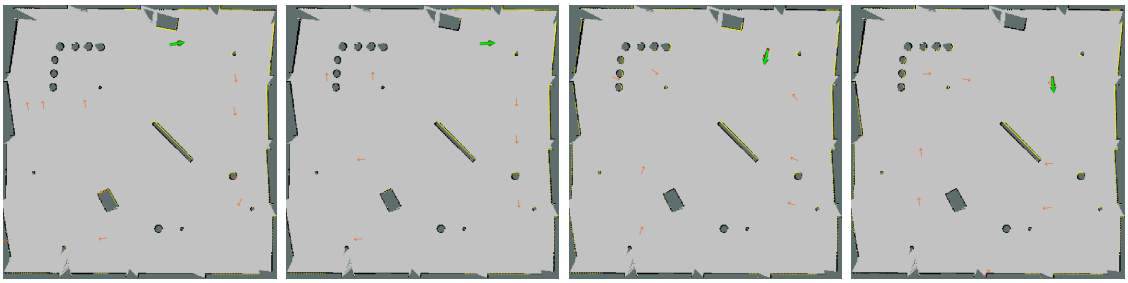


(q) GAGL, i =110 (r) GAGL, i =120 (s) GAGL, i =130 (t) GAGL, i =140

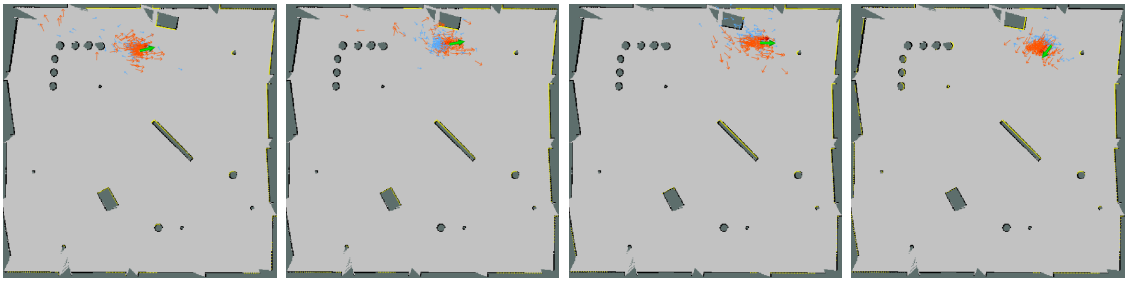
Figura C.4: Iterações [110,120,130,140] da localização global



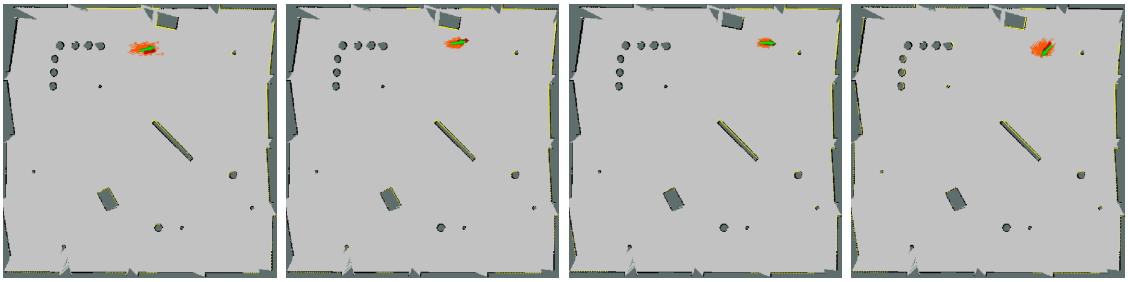
(a) AMCL, i =150 (b) AMCL, i =160 (c) AMCL, i =170 (d) AMCL, i =180



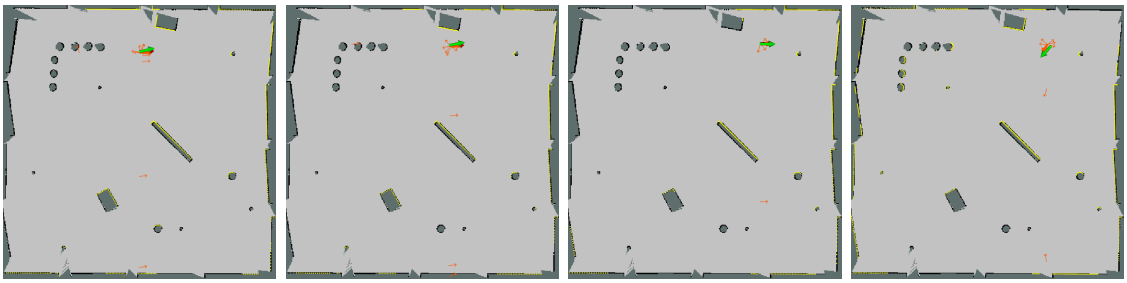
(e) PMGL, i =150 (f) PMGL, i =160 (g) PMGL, i =170 (h) PMGL, i =180



(i) PSGL, i =150 (j) PSGL, i =160 (k) PSGL, i =170 (l) PSGL, i =180



(m) DEGL, i =150 (n) DEGL, i =160 (o) DEGL, i =170 (p) DEGL, i =180



(q) GAGL, i =150 (r) GAGL, i =160 (s) GAGL, i =170 (t) GAGL, i =180

Figura C.5: Iterações [150,160,170,180] da localização global

Apêndice D

Artigos científicos produzidos a partir deste trabalho

Este apêndice apresenta a primeira página dos artigos científicos derivados deste trabalho. O primeiro artigo "*Particle Swarm Localization for Mobile Robots Using a 2D Laser Sensor*" (Figura D.1) foi aceito pelo "ROBOT'2019: *Fourth Iberian Robotics Conference*" e pelo BRACIS: *Brazilian Conference on Intelligent Systems* (2019), mas publicado apenas neste último. Esta publicação pode ser encontrada em `https://doi.org/10.1109/BRACIS.2019.00057`. O segundo artigo, "*Global Localization of Unmanned Ground Vehicles Using Swarm Intelligence and Evolutionary Algorithms*" (Figura D.2) foi submetido ao *Journal of Intelligent & Robotic Systems* e aceito para publicação em 11 de Janeiro de 2023. Esta publicação pode ser encontrada em `https://doi.org/10.1007/s10846-023-01813-6`.

2019 8th Brazilian Conference on Intelligent Systems (BRACIS)

Particle Swarm Localization for Mobile Robots Using a 2D Laser Sensor

João Luiz C. Carvalho[1,2], Paulo César M. A. Farias[1], Edmar Egidio P. de Souza[1], Eduardo F. de Simas Filho[1]
[1]*Digital Systems Laboratory, Electrical Engineering Program*
Federal University of Bahia
Salvador, Brazil
[2]*Center for Science and Technology in Energy and Sustainability*
Federal University of The Recôncavo da Bahia
Feira de Santana, Brazil
joao.luiz@ufrb.edu.br, {paulo.farias, edmar.egidio, eduardo.simas}@ufba.br

***Abstract*—Mobile robot localization is a complex task, specially in unstructured indoor environments, due to noise and wrong data association from sensors. The localization procedure is even harder when the vehicle has low confidence about its last pose estimate, situation that requires a Global Localization procedure. In this work, a Global Localization algorithm based on Particle Swarm Optimization (PSO) is integrated with a Pose Tracking algorithm, the Perfect Match, to obtain a robust localization technique. Results show that this approach can solve the problem of Global Localization with good performance.**

***Index Terms*—Mobile Robot; Global Localization; Particle Swarm Optimization; Map Matching**

I. Introduction

In virtually all mobile robotics applications, localization plays a key role. To move around and perform tasks safely and efficiently, the mobile terrestrial robot must know its own position (x, y coordinates) and orientation (θ) in relation to the environment (the world coordinate system) with low degree of uncertainty.

In some situations the robots localize themselves only using proprioceptive sensors that measure system's internal values, an approach known as dead reckoning. On the other hand, map based localization techniques relies on the position matching between features extracted from the environment (using laser-based sensors or camera, for instance) and its respective coordinates on the map. Since the uncertainty in position grows without bound using dead-reckoning, the robustness of map-based algorithms is substantially higher than the former approach [1].

Most of the mobile robots rely on the perception of the environment to obtain their location by detecting features, like landmarks or contours, present in both the environment and the map. These elements are usually represented by points in the map coordinate system. Also, the robot can detect the relative position of these elements using perceptive sensors. Since the robot knows the coordinates of all features in the map (x^f, y^f) and also is able calculate its range and bearing from itself (d_x^f, d_y^f, ϕ^f), it is only required a linear transformation between the robot coordinate system and the map coordinate system to obtain the robot pose (x, y, θ) in relation to the map.

However, the coordinates of detected features are only approximate due to the noise and uncertainty of sensor measurements. Moreover, the correspondence between the observed feature and its true position is not always guaranteed, due to features similarities or the uncertainty from the sensing. The uncertainty about its own localization can be utmost when the vehicle has to localize itself from scratch, i.e., without any previous localization estimate, a process known as Global Localization. Thus, a robust localization algorithm must be able to handle common ambiguities and multiple hypotheses when determining a robot pose.

This work introduces a method to localize a mobile robot based on the contours of the 2D map, which is a common approach for unstructured indoor environments. The method is based on a Pose Tracking algorithm, described in section II, and a Global Localization algorithm based on PSO, described in section III. The section IV details the methodology used in the tests and section V discuss the experimental results obtained from a real Automated guided vehicle (AGV) in different tests. Finally, the section VI comprises the authors conclusions about this work.

II. Perfect Match Algorithm

The Perfect Match is a light computational pose tracking algorithm that uses a map matching approach. It was firstly introduced by M. Lauer, S. Lange, and M. Riedmiller [2] to localize a soccer playing robot on the game field with white markings using an omnidirectional color camera. H. Sobreira et al. [3] extended the use of Perfect Match to localize autonomous robots in a 2D occupancy grid map using laser sensors to detect the contours of the environment and M. Pinto et al. [4] adapted the Perfect Match from 2D to the 3D localization using 3D laser sensors.

The algorithm implemented in [3] incrementally tries to find the best fit of the vehicle on a 2D map based on its previous pose estimate and the N points produced by the laser sensor. The new pose estimate is computed by minimizing the fitting error E between the laser's point cloud and the occupied cells

2643-6264/19/$31.00 ©2019 IEEE
DOI 10.1109/BRACIS.2019.00057

Figura D.1: Primeira página do artigo publicado no BRACIS: *Brazilian Conference on Intelligent Systems* (2019)

Journal of Intelligent & Robotic Systems (2023) 107:45
https://doi.org/10.1007/s10846-023-01813-6

SHORT PAPER

Global Localization of Unmanned Ground Vehicles Using Swarm Intelligence and Evolutionary Algorithms

João L. C. Carvalho[1,2] · Paulo C. M. A. Farias[1] · Eduardo Furtado Simas Filho[1]

Received: 15 July 2022 / Accepted: 11 January 2023

Abstract
Mobile robot localization is a complex task, specially in unstructured indoor environments, due to noise and wrong scan-to-map association. The localization procedure becomes critical when the vehicle has low confidence about its last pose estimate, situation that requires a global localization procedure. An intuitive approach to solve the Global Localization Problem (GLP) is to distribute several pose hypotheses all over the map and select the most likely one according to an optimization heuristic such as Monte Carlo, Swarm Intelligence or Evolutionary Algorithm. However, hardware limitations and environment characteristics may affect the localization efficacy. Furthermore, we found relatively few studies exploring the effectiveness and the computing cost of different localization methods under different scenarios e.g. offices, corridors and big warehouses. In this work, we analyze different global localization methods based on multi-hypothesis optimization metaheuristics. We use the scan-to-map matching error computed by a pose tracking algorithm, the Perfect Match (PM), as the metric to score the hypotheses. Our main contribution is to propose an enhanced localization system by integrating a multi-hypothesis global localization method with the PM. We also analyzed different optimization heuristics applied to the GLP under typical and special conditions. Using simulations and real-world experiments, we measured the success rate and computing cost using several population sizes. Results show that studied methods perform differently in distinct scenarios, but our proposals based on Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) showed an average success rate above 83%, while other methods did not achieved 80%. Furthermore, PM-based methods exhibit lower computing cost when compared to the traditional Adaptive Monte Carlo Localization (AMCL) after the 100th iteration. In summary, our study shows that the GA-based proposal, which performed slightly better than the PSO-based, represents the best candidate to integrate a robust localization system.

Keywords Global localization problem · Swarm intelligence · Evolutionary algorithm · Map matching

1 Introduction

Localization is a task performed by the robot to identify its position and orientation with respect to the world. In many mobile robot applications, the robot is defined as a rigid body that moves on the ground and need to know its 2D coordinates on the reference plane (x, y) and its orientation (θ) in relation to a fixed reference [1]. To move around and perform tasks safely and efficiently, the mobile terrestrial robot must know its pose (x, y, θ) with low degree of uncertainty. A digital representation of the environment, such as an occupancy grid map [2], is often used to support the localization process.

Typically, the robot knows a previous pose estimate and only needs to update it based on most recent measurements from sensors and odometry. However, in some cases the previous pose is unknown or unreliable due to lack of past estimations or poor measurements [3]. As consequence, the robot may need to abort the pose tracking and recover its localization. This situation is known as Global Localization Problem (GLP), and the robot usually assumes it can be located anywhere on the map. From then on, the robot uses

✉ João L. C. Carvalho
joao.luiz@ufrb.edu.br

Paulo C. M. A. Farias
paulo.farias@ufba.br

Eduardo Furtado Simas Filho
eduardo.simas@ufba.br

[1] Digital Systems Laboratory, Federal University of Bahia, Salvador, Bahia, Brazil

[2] Center for Science and Technology in Energy and Sustainability, Federal University of Recôncavo da Bahia, Feira de Santana, Bahia, Brazil

Published online: 20 March 2023

Figura D.2: Primeira página do artigo publicado no *Journal of Intelligent & Robotic Systems* (2023)