UNIVERSIDADE FEDERAL DE ITAJUBÁ

PROGRAMA DE PÓS GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Algoritmo Híbrido Auto-adaptativo para Localização em Robótica Móvel

Audeliano Wolian Li

Itajubá, julho de 2016

UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Audeliano Wolian Li

Algoritmo Híbrido Auto-adaptativo para Localização em Robótica Móvel

Dissertação submetida ao Programa de Pós-Graduação em Ciência e Tecnologia da Computação como parte dos requisitos para obtenção do Título de Mestre em Ciência e Tecnologia da Computação

Área de Concentração: Matemática da Computação

Orientador: Prof. Dr. Guilherme Sousa Bastos

Julho de 2016
Itajubá - MG

Dedico aos meu pais e à Thalita que forneceram
todo o apoio necessário para o desenvolvimento deste trabalho.

Agradecimentos

Agradeço primeiramente aos meus pais e familiares pelo apoio incondicional.

Ao meu orientador e amigo, Guilherme Sousa Bastos, por toda ajuda, mesmo antes de iniciar este trabalho, e ensinamentos, sem os quais eu não teria chegado até o presente momento.

À banca examinadora, pela leitura cuidadosa e pelas valiosas colaborações para a redação final deste texto.

Aos amigos e colegas do LRO, em especial ao Adriano, Alyson, Pedro e Lucas que foram de grandessíssima ajuda para o pontapé inicial deste trabalho.

À Capes pelo apoio financeiro durante estes 2 anos.

À Fapemig pelo financiamento do projeto de pesquisa TEC-APQ-00666-12, o qual possibilitou a compra do robô utilizado neste trabalho (Pioneer-3DX).

E um agradecimento mais que especial à Thalita, minha amada namorada, que esteve presente em todos os momentos. Só você sabe dos nossos sacrifícios e esforços feitos durante esses anos. Essa conquista é sua também!

"Às vezes são as pessoas que
ninguém espera nada
que fazem as coisas
que ninguém consegue imaginar."
(Alan Turing)

UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO
EM CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Audeliano Wolian Li

Algoritmo Híbrido Auto-adaptativo para Localização em Robótica Móvel

Dissertação aprovada por banca examinadora em 08 de Julho de 2016, conferindo ao autor o título de **Mestre em Ciência e Tecnologia da Computação**.

Banca Examinadora:
Prof. Dr. Guilherme Sousa Bastos - UNIFEI (Orientador)
Prof. Dra. Esther Luna Colombini - UNICAMP
Prof. Dr. Carlos Henrique Valério de Moraes - UNIFEI

Itajubá
2016

Resumo

A localização em robótica móvel tem como objetivo fornecer a posição e orientação do robô a cada instante e de forma precisa para que aplicações de alto nível possam ser executadas corretamente, além de resolver os problemas de localização local, global e sequestro do robô. Algoritmos mais completos são capazes de resolver estes três problemas sem exigir um alto custo computacional, sendo a maioria métodos que utilizam mais de uma técnica de localização, caracterizados como algoritmos híbridos. A proposta deste trabalho é apresentar uma nova abordagem de localização robótica baseada em filtro de partículas capaz de solucionar os três problemas da localização, que seja robusto e de baixo custo computacional. Diferentemente de outras técnicas híbridas que utilizam dois métodos distintos de localização, este algoritmo utiliza duas técnicas baseadas em filtro de partículas, o *Kullback-Leibler Distance - KLD* e o *Self-Adaptative Monte Carlo Localization - SAMCL*. Todos os testes foram executados em ambientes reais com o auxílio do *framework ROS* e os resultados experimentais demonstraram melhorias em todas as situações em relação às técnicas executadas individualmente.

Palavras-chaves: Filtro de Partículas; KLD; SAMCL.

Abstract

The localization of mobile robots aims to provide the position and orientation of the robot for each instant and accurately to high-level applications can be executed correctly, addition to solving the problems of the position tracking, global localization and kidnapping robot. More complete algorithms are able to solve these three problems without requiring a high computational cost, and most methods employing more than one localization technique, characterized as hybrid algorithms. The purpose of this work is to present a new approach to robot localization based on particle filter able to solve the three problems of localization, which has low computational cost and is robust. Unlike other hybrid techniques that use two distinct localization methods, this algorithm uses two techniques based on particle filter, the *Kullback-Leibler Distance - KLD* and the *Self-Adapatative Monte Carlo Localization - SAMCL*. All of the tests were run in real environments with the aid of the *ROS framework* and the experimental results demonstrates improvements in all situations in relation to techniques executed individually.

Keywords: Particle Filter; KLD; SAMCL.

Conteúdo

Conteúdo 9

Lista de Figuras 11

Lista de Tabelas 12

1 INTRODUÇÃO 15
1.1 Motivação 15
1.2 Objetivos 17
1.3 Contribuições 17
1.4 Estrutura do Trabalho 17

2 LOCALIZAÇÃO EM ROBÓTICA MÓVEL 18
2.1 Localização de Markov 20
2.1.1 Localização de Monte Carlo 22
2.2 Trabalhos Relacionado à Localização Robótica 26
2.2.1 Algoritmos Híbridos 27

3 ALGORITMOS PARA LOCALIZAÇÃO BASEADOS EM FILTRO DE PARTÍCULAS 28
3.1 KLD-Sampling 28
3.1.1 Distância de Kullback-Leibler 28
3.1.2 Aplicação do KLD em filtros de partículas 30
3.2 Self-Adaptative Monte Carlo Localization (SAMCL) 33
3.2.1 Etapas da localização baseada em SAMCL 33
3.2.2 Etapa de pré-armazenamento 34
3.2.3 Cálculo do SER 35
3.2.4 Localização baseada em SAMCL 37
3.3 Self-Adaptative KLD Localization (SA-KLD) 40

4 MODELAGEM DO PROBLEMA 46
4.1 Hardware 46
4.1.1 Robô Pioneer 3DX 46
4.1.2 Master 47
4.1.3 Sensor Laserscan SICK LMS 291 47
4.2 ROS 49
4.2.1 ROS Packages 51

4.3 Modelo de Cinemática . . . 52
4.4 Modelo de Observação . . . 53

5 EXPERIMENTOS E RESULTADOS . . . 55
5.1 Análise de Parâmetros . . . 58
5.2 Localização Local . . . 61
5.3 Localização Global . . . 63
5.4 Sequestro do Robô . . . 64
5.5 Testes em Diferentes Ambientes . . . 67
5.5.1 Expansão do Labirinto . . . 67
5.5.2 Ambiente Simétrico . . . 70

6 CONCLUSÃO E TRABALHOS FUTUROS . . . 72
6.1 Conclusão . . . 72
6.2 Trabalhos Futuros . . . 73

BIBLIOGRAFIA . . . 74

Lista de Figuras

Figura 2.0.1–Arquitetura do sistema de localização de um robô móvel. 19
Figura 2.1.1–Ilustração do funcionamento do filtro de partículas. 24
Figura 3.1.1–Localização com KLD. 32
Figura 3.2.1–Esquemático das etapas do SAMCL. 34
Figura 3.2.2–Diferença da quantidade de áreas SER. 36
Figura 3.3.1–Comparação do número de partículas entre KLD, SAMCL e SA-KLD. 44
Figura 3.3.2–Fluxograma do processo de localização. 45
Figura 4.1.1–Pioneer 3DX com laserscan SICK LMS 291. 46
Figura 4.1.2–Varredura bidimensional do ambiente através dos feixes do laserscan. . 48
Figura 4.1.3–Estrutura interna de um laserscan 48
Figura 4.1.4–Esquemático de um laserscan. 49
Figura 4.3.1–Translação seguida de rotação de um robô diferencial. 52
Figura 4.4.1–Modelo de observação para uma partícula com cinco feixes do laser. . . 53
Figura 5.0.1–Labirinto modelável de blocos. 55
Figura 5.0.2–Mapa do labirinto criado pelo *gmapping*. 56
Figura 5.0.3–Testes e análises dos algoritmos para diferentes situações. 57
Figura 5.0.4–Grafo do processo de localização produzido pelo *rosgraph*. 58
Figura 5.1.1–Tabela de distribuição normal z. 59
Figura 5.1.2–Gráfico do número de partículas em relação ao número de bins ocupados. 60
Figura 5.2.1–Mapa do labirinto com as marcações das cinco posições usadas para testes. 61
Figura 5.2.2–Resultados comparativos para o problema de rastreamento de posição. 62
Figura 5.3.1–Gráfico comparativo do número de partículas pelo tempo da localização global. 65
Figura 5.4.1–Posições onde ocorreram os sequestros e destinos. 66
Figura 5.4.2–Gráfico do número de partículas em função do tempo quando o sequestro é identificado. 66
Figura 5.4.3–Vídeo comparativo dos três algoritmos em ambiente real. 68
Figura 5.5.1–Diferentes Ambientes. 69
Figura 5.5.2–Distribuição das partículas através de um mapa simétrico. 71

Lista de Tabelas

Tabela 4.1.1–Especificações do laserscan *Sick LMS 291*. 49
Tabela 5.0.1–Tempo para executar o processo offline. 56
Tabela 5.1.1–Parâmetros referentes a cada filtro de partículas. 58
Tabela 5.2.1–Tempo de cada iteração em função do número de partículas. 63
Tabela 5.3.1–Número inicial de partículas, quantidade de áreas SER e bins ocupados das três situações iniciais. 63
Tabela 5.3.2–Tempo de localização para cada algoritmo em diferentes casos. 64
Tabela 5.4.1–Tabela comparativa entre SAMCL e SA-KLD para o processo de relocalização. 67
Tabela 5.5.1–Comparação entre SAMCL, SA-KLD e KLD para um mapa com dimensões maiores e 20 execuções para cada caso. 68
Tabela 5.5.2–Tabela da posição final referente ao mapa simétrico. 70

Lista de Algoritmos

1 Localização baseada em MCL 25

2 Localização baseada em KLD-Sampling 31
3 Cálculo da energia para cada célula do mapa 35
4 Cálculo do SER 37
5 Localização baseada em SAMCL 39
6 Localização baseada em SA-KLD 42

7 Rotina do modelo de observação 54

Glossário

AMCL	*Adaptative Monte Carlo Localization*
BVP	*Boundary Value Problem*
CBR	*Competição Brasileira de Robótica*
EKF	*Extended Kalman Filter*
FP	*Filtro de Partículas*
KLD	*Kullback-Leibler Distance*
LRO	*Laboratório de Robótica*
MCL	*Monte Carlo Localization*
MLE	*Maximum Likelihood Estimate*
P3DX	*Pioner 3DX*
ROS	*Robot Operating System*
SA-KLD	*Self-Adaptative KLD Localization*
SAMCL	*Self-Adaptative Monte Carlo Localization*
SER	*Similar Energy Region*
UKF	*Unscented Kalman Filter*
UPF	*Unscented Particle Filter*
VANT	*Veículo Aéreo Não Tripulado*

1 Introdução

1.1 Motivação

A área da robótica móvel autônoma vem atraindo muito interesse de pesquisadores e da sociedade, tanto que nas últimas duas décadas esta área de estudos apresentou um rápido e crescente desenvolvimento. Várias aplicações práticas surgiram nos últimos anos, onde o foco foi voltado às realizações de diferentes tarefas com interação homem-máquina, tais como: aplicações domésticas (babás e aspiradores de pó robóticos) (WISSPEINTNER et al., 2009), industriais (robôs para logística, transporte de estoque e manipuladores autônomos) (NICKERSON et al., 1998), urbanas (carros autônomos, sistema de entregas utilizando VANTs) (GEORGIEV; ALLEN, 2004), militares (sistema de reconhecimento e vigilância com VANTs, transporte de suprimentos e equipamentos) (RAIBERT et al., 2008) e de segurança e defesa civil (localização e detonação de minas subaquáticas e robôs de resgate) (LIU; WANG; FENG, 2005). Desta forma, a robótica abrange inúmeras áreas havendo um grande interesse econômico para desenvolvimento de aplicações que facilitem ou solucionem os problemas do cotidiano da vida moderna (WOLF et al., 2009).

Todas as aplicações citadas possuem um sistema de navegação, o qual é subdivido em quatro principais pilares: *percepção*, a forma que o robô interpreta os dados dos sensores; *cognição*, as tomadas de decisão para alcançar um determinado objetivo; *controle*, referente às técnicas de controle dos motores para executar uma trajetória desejada; e, por fim e objetivo do trabalho, *localização*, reconhecimento da posição do robô dentro de um mapa (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

Uma dos primeiras etapas da navegação a ser executada é a localização que estima a postura inicial do robô (formada por três dimensões (x, y, θ). A partir disto, os próximos processos definem suas respectivas ações. Portanto, um algoritmo de localização com uma boa precisão é essencial para uma navegação eficiente. Além de definir a posição inicial, também corrige erros acumulados da movimentação durante toda execução. Assim, o algoritmo implementado deverá ter baixo custo computacional, caso contrário afetará o processamento de outros algoritmos essenciais para a navegação. Por último, deve possuir um tempo de localização reduzido para corrigir com rapidez os efeitos de uma derrapagem dos pneus e possíveis erros gerados pelos sensores (FOXYZ; BURGARDY; THRUNZ, 1999).

Várias técnicas já foram propostas baseadas em localização absoluta e relativa. As técnicas do primeiro grupo determinam a postura do robô em relação ao mapa, enquanto técnicas do segundo grupo encontram apenas a posição relativa do robô com base nas medições da odometria, acelerômetros e giroscópios. Em relação à localização absoluta,

existem três métodos distintos: (a) o geométrico, método que utiliza medidas de distâncias e ângulos relacionados a três ou mais marcos para determinar a posição do robô, assim, necessitando de uma adaptação do ambiente, tornando-o em alguns casos inviável e até mesmo custoso (LEONARD; DURRANT-WHYTE, 1991), (b) de correspondência de mapas, os quais determinam a localização utilizando um mapa 2D do ambiente e medições dos sensores para fazer um reconhecimento de padrões e encontrar uma função de erro da posição estimada (GOUVEIA, 2008), e (c) o estatístico que, apesar de seu tratamento ser mais complexo, se comparado ao primeiro método, tem sido largamente utilizado devido à velocidade de resposta, possibilitando sua utilização em tempo de execução, sendo os algoritmos mais comuns o Filtro de Kalman Estendido, o Markov e o Filtro de Partículas (THRUN et al., 1998; FOXYZ; BURGARDY; THRUNZ, 1999; FOX et al., 1999).

O *ROS* (ROS, 2007) possui um sistema de localização utilizado para navegação robótica, o qual utiliza um método estatístico baseado em filtro de partículas chamado *AMCL*[1]. Este algoritmo foi utilizado na Competição Brasileira de Robótica - CBR no ano de 2015 e notou-se que haviam algumas limitações, as quais contribuíram para a motivação do trabalho.

Apesar do algoritmo ser capaz de localizar o robô dentro de um mapa conhecido, há algumas situações em que o *AMCL* tende a falhar: (a) caso o robô seja transportado de sua posição atual para uma posição qualquer do mapa, sem aviso prévio, (b) se houver um grande deslizamento das rodas ocasionado por choque em um obstáculo ou deslocamento em terreno acidentado e (c) caso a localização convirja em uma postura incorreta, o robô ficará perdido e não será capaz de identificar e corrigir o erro. Uma forma de evitar que o algoritmo erre a localização é fornecer a informação da postura aproximada do robô no mapa. Porém disponibilizar este dado não é sempre possível e é inviável para uma navegação autônoma não ter uma rotina que identificação e correção dos erros de localização.

Desta forma, após o término do evento, foi feito uma busca por outros algoritmos baseados em filtro de partículas no ROS e todos resolviam os problemas da localização parcialmente.

Dado que os métodos probabilísticos de localização apresentam melhores resultados em aplicações de tempo real, em relação aos métodos de correspondência de mapas e geométrico, e que o *ROS* carece de um algoritmo de localização baseado em filtro de partículas menos susceptível a falhas, a motivação do trabalho está em desenvolver um algoritmo híbrido para resolver tais erros, além de solucionar os problemas da localização.

[1] *Adaptative Monte Carlo Localization* - *AMCL* é um pacote *ROS* desenvolvido para tratar a localização em ambientes bidimensionais. O algoritmo é implementado utilizando um filtro de partículas juntamente com a técnica de *Kullback-Leibler Distance* - *KLD* descrita por Fox (2003).

1.2 Objetivos

O principal objetivo deste trabalho é propor um algoritmo híbrido, o qual utiliza duas técnicas distintas baseadas em filtro de partículas, de tal forma que seja capaz de resolver os três sub-problemas da localização: (a) rastreamento de posição, (b) localização global e (c) sequestro do robô. Além de encontrar a localização mais rapidamente em relação aos outros métodos de localização baseado em filtro de partículas, e que apresente baixo custo computacional. Como auxílio para desenvolvimento, testes e análises do algoritmo desenvolvido, será utilizado o *framework ROS*.

1.3 Contribuições

A principal contribuição do trabalho é o desenvolvimento de um algoritmo híbrido para localização robótica, denominado *Self-Adaptative KLD Localization - SA-KLD*, o qual é capaz de resolver os três problemas da localização: o rastreamento de posição, a localização global e o rapto do robô.

Uma segunda contribuição do SA-KLD é a distribuição de um número variável, conforme a necessidade, de partículas sobre áreas do mapa que possuem maiores probabilidades de conter a verdadeira postura do robô.

1.4 Estrutura do Trabalho

Este trabalho é estruturado da seguinte forma: o capítulo a seguir é destinado à fundamentação teórica da localização robótica, detalhando seu funcionamento e problemas existentes, além de uma revisão bibliográfica listando os diversos tipos de abordagens utilizadas, tanto os métodos clássicos quanto os híbridos. Em seguida, no capítulo 3 são apresentados os algoritmos que servem de base para o filtro híbrido do trabalho, destacando as vantagens e desvantagens de cada método, além de introduzir a nova abordagem desenvolvida neste trabalho. O capítulo 4 é destinado à apresentação das ferramentas utilizadas para os testes dos três algoritmos de localização: hardware, software e modelos de cinemática e de observação. O capítulo seguinte é formado pelos resultados experimentais, o qual apresentará os testes feitos e as análises comparativas entre o algoritmo do trabalho e os algoritmos utilizados como base para seu desenvolvimento. Conclusão e trabalhos futuros são discutidos no último capítulo.

2 Localização em Robótica Móvel

A função da localização consiste em estimar a postura do robô dentro de um ambiente. É essencial que a localização forneça a informação de forma precisa para que qualquer processo da navegação (por exemplo, planejamento de trajetórias e execução de tarefas) seja executado corretamente (FOXYZ; BURGARDY; THRUNZ, 1999).

O sistema de odometria está presente em grande parte dos modelos de robôs móveis. Assim, é possível estimar o deslocamento dos mesmos e, com auxílio de funções matemáticas, determinar a posição no ambiente. Em robôs cuja movimentação é feita por rodas, os sensores (*encoders*) são acoplados a elas e a relação quantidade de giros por tempo fornece seu deslocamento. Entretanto, este modelo apresenta erros acumulados de arredondamento dos cálculos e de fontes mecânicas, tais como: diferença no diâmetro das rodas, pressão do pneu, irregularidades do terreno, deslizamento, falta de lubrificação, entre outros. Apesar destes erros serem pequenos, eles se acumulam ao longo do tempo e se não forem corrigidos, inviabilizarão a localização e, consequentemente, a navegação. Visando contornar este problema, sensores de naturezas diferentes são adicionados como: sensores ultrassônicos, laser, câmera e infravermelho. Assim, fornecido um mapa, a localização é estimada através de informações dos sensores de odometria e de percepção. Contudo, todos os sensores fornecem dados imprecisos (alguns mais do que outros), e estes erros de medição devem ser levados em consideração sendo modelados matematicamente. A figura 2.0.1 ilustra a arquitetura do sistema de localização de um robô móvel.

Além disso, a localização enfrenta problemas que são relacionadas ao nível de conhecimento do estado atual e da dificuldade em encontrar uma solução precisa, robusta e de baixo consumo de processamento. Através destas questões, a localização pode ser dividida em três sub-problemas: (a) rastreamento de posição ou localização local, (b) localização global, e (c) rapto do robô (ZHANG, 2010).

- **Rastreamento de posição ou Localização local:** Assumindo que o robô conheça sua postura inicial em um ambiente conhecido, durante seu deslocamento os dados de odometria e de sensores de percepção são filtrados e os erros reduzidos, fornecendo um estado estimado com maior precisão.

- **Localização global:** A localização global é um problema mais complexo, pois a postura inicial do robô não é conhecida, dessa forma o algoritmo precisa determinar sua posição e orientação através dos dados de controle e de sensores. Assim que a localização global é determinada, o problema torna-se um rastreamento de posição.

- **Rapto do robô:** O problema mais complexo de localização é dividido em duas categorias: (a) rapto real, e (b) falhas de localização. O primeiro é considerado

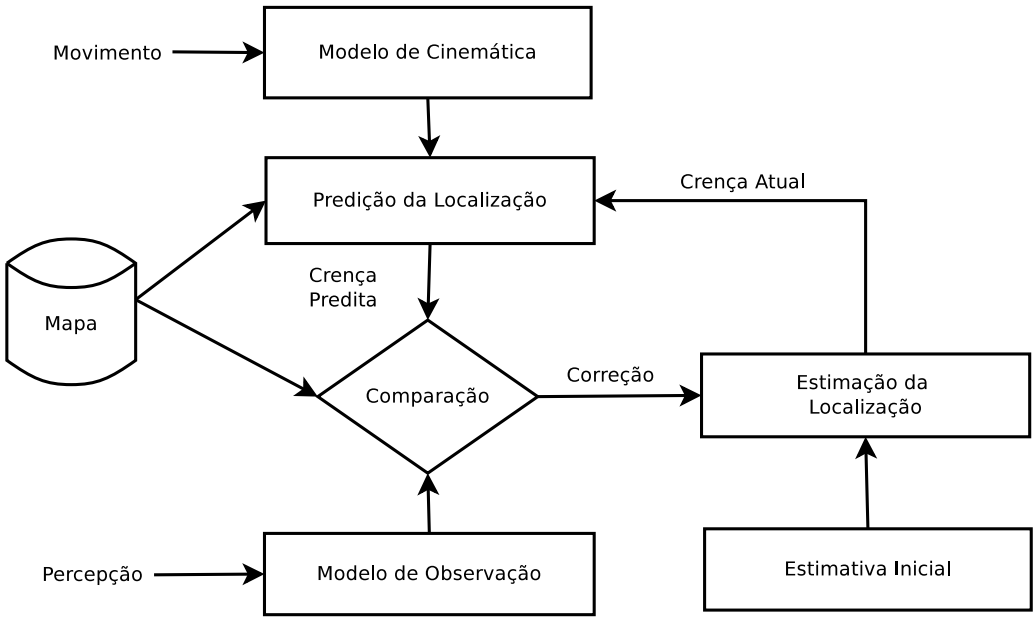


Figura 2.0.1 – Arquitetura do sistema de localização de um robô móvel. Adaptada de Costa e Selvatici (2014). O processo de localização precisa, primeiramente, que o robô esteja em movimento e durante o deslocamento os ciclos de movimento e percepção se intercalam. Assim, dado uma estimativa da localização, o sistema faz a leitura da odometria, dos dados de controle e do mapa para estimar a posição a posteriori. A próxima etapa é fazer a leitura do ambiente utilizando sensores de percepção, como sonares, lasers, câmeras ou infravermelhos, e com o auxílio do mapa é estimada uma localização observada, a qual é comparada com a estimativa encontrada no processo cinemático. A diferença entre as estimativas são utilizadas para corrigir a estimativa atual e a crença do robô a respeito de sua localização.

quando o robô é retirado de sua postura atual, podendo ser obra de alguma pessoa ou um evento que provoque uma grande alteração na localização (por exemplo, deslizamento das rodas). O segundo são falhas que levam o robô a crer que foi sequestrado. Por exemplo, caso o robô se desloque até uma área incompleta do mapa e faça a leitura de objetos não modelados, presença de objetos estocásticos, falhas mecânicas, erros nos dados dos sensores ou áreas com verossimilhanças muito próximas. A dificuldade do problema está em identificar de forma precisa os momentos em que isto ocorre. Uma vez identificado e corrigido, o processo torna-se uma localização global.

O ambiente real é repleto de incertezas e estas são originadas de ruídos do hardware, aproximações do software e situações imprevisíveis de ambientes estocásticos. Visto que estas incertezas estão sempre presentes em aplicações reais, abordagens probabilísticas são ideais para tratar este problema, além de determinar a postura do robô através de distribuições de probabilidade espalhadas por todo o mapa. A abordagem mais comum é a Localização de Markov, uma técnica por método probabilístico baseado no filtro de Bayes,

um algoritmo de estimação estatística aplicado aos problema da robótica, o qual tem demonstrado melhores resultados para a solução dos três problemas da localização (WOLF et al., 2009). Existem diversas formas de implementar essa abordagem, sendo as mais comuns o *Filtro de Kalman*, baseado em *Grid* e o *Monte Carlo* (THRUN; BURGARD; FOX, 2005).

2.1 Localização de Markov

O método Localização de Markov foi desenvolvido para resolver os problemas de rastreamento de posição, localização global e rapto do robô (THRUN; BURGARD; FOX, 2005). Além disso, é um método bastante eficiente devido à propriedade de Markov: efetuando uma ação em um determinado estado, o resultado dependerá apenas da ação e do estado atual, ou seja, não é de interesse saber a sequência de ações e estados passados (PELLEGRINI; WAINER, 2007), refletindo diretamente no tempo de processamento do algoritmo e no uso do espaço de memória.

A ideia central é representar a postura do robô com uma crença através de uma distribuição probabilística, podendo ser representada como um estado $s =< x, y, \theta >$, onde x e y são coordenadas cartesianas e θ, sua orientação. A distribuição *Bel(s)* é referente à crença do robô estar no estado s (FOX et al., 1999).

Como o algoritmo de Markov é recursivo, para calcular a crença a posteriori $Bel(s_t)$ é necessário uma crença inicial $Bel(s_0)$ no tempo $t = 0$. Assim, a informação da postura inicial irá definir a forma de representar a crença inicial (THRUN; BURGARD; FOX, 2005):

- **Postura inicial bem definida:** Teoricamente, a crença $Bel(s_0)$ é inicializada com 1 para o estado com a postura inicial e 0 para as demais posições. Na prática, esta postura é aproximada, portanto o valor da crença deverá ser definida através de uma função de distribuição gaussiana bem estreita (baixo valor de variância) e com média igual à postura inicial.

- **Postura inicial desconhecida:** Neste caso o método de localização global inicializa o processo através de uma crença uniformemente distribuída para todos os estados.

- **Postura inicial parcialmente conhecida:** Para casos em que exista uma informação aproximada da postura inicial, os estados ao redor desta localização receberão crenças uniformemente distribuídas, enquanto os outros estados receberão zero.

Para atualizar a crença, são utilizados dois modelos probabilísticos: (a) modelo de ação, e (b) modelo de percepção. O primeiro é modelado por uma probabilidade condicional $P(s|s', u)$, que representa a probabilidade de chegar no estado s, dado que uma ação

u foi executada no estado s'. Portanto, a crença $Bel(s)$ pode ser calculada baseada na fórmula utilizada nas cadeias de Markov (PELLEGRINI; WAINER, 2007):

$$Bel(s) \leftarrow \int P(s|s', u)Bel(s')ds' \tag{2.1}$$

O modelo de percepção é formado pela leitura de sensores z e a probabilidade da verossimilhança dado um estado s é representado como $P(z|s)$ (FOX et al., 1999). A atualização da crença pode ser dada pela equação:

$$Bel(s) \leftarrow \alpha P(z|s)Bel(s) \tag{2.2}$$

onde α é o fator de normalização, o qual garante que o somatório das crenças seja sempre igual a 1.

Algoritmos clássicos de localização de Markov são capazes de resolver os problemas da localização, sendo os mais comuns o (a) Filtro de Kalman Estendido (EKF), (b) Localização em Grid, e (c) Localização de Monte Carlo (MCL):

- **Localização baseada no Filtro de Kalman Estendido:** Essa abordagem é baseada no filtro de Kalman, o qual utiliza a informação da postura atual, os dados de deslocamento e dos sensores para encontrar determinados pontos de referência no ambiente, assim estimando a próxima localização do robô. Assume-se que as incertezas da postura do robô podem ser representadas como uma distribuição Gaussiana unimodal, ou seja, seu método de estimação de estados não aceita ambiguidades. Devido a esta característica, esse método aborda apenas o problema do rastreamento de posição e seus principais passos são: linearizar utilizando a expansão de Taylor as funções não lineares de transição de estado e de medições dos sensores, e aplicar a fusão de dados da odometria e dos sensores de percepção a fim de reduzir os erros causados pelos ruídos (LEONARD; DURRANT-WHYTE, 1991).

- **Localização baseada em Grid:** Este modelo consiste em discretizar o mapa em várias células e cada uma possui a informação da probabilidade do robô estar ocupando-a. Assim, com o deslocamento do robô, cada célula terá sua probabilidade atualizada. Devido a este armazenamento de informações, a técnica é capaz de resolver a localização local, a global e o sequestro do robô. Porém esta característica torna o algoritmo muito lento e custoso, inviabilizando a utilização em tempo de execução. A resolução do grid é a variável que determina a precisão da localização, assim uma resolução pequena fornece uma precisão maior, mas com custo computacional grande (ZHANG, 2010). Esta técnica, em contraste com a localização de Markov, não representa a incerteza do robô em sua crença atual e, portanto, não pode lidar adequadamente com situações ambíguas (FOX; BURGARD; THRUN, 1999).

- **Localização baseada em técnicas de Monte Carlo:** Técnica não-paramétrica e multi-modal baseada em filtro de partículas. Sendo que cada partícula é interpretada como um pseudo-robô com posição e orientação próprios, e possui também um fator de importância, denominado peso, para quantificar a crença da localização do robô em relação ao mapa do ambiente. Esta técnica cria e distribui as partículas por todo o mapa e durante o processo de execução, são comparados os dados dos sensores de percepção com as informações do mapa, assim os valores dos pesos de cada partícula são atualizados, de forma que quanto maior o peso, maior é a probabilidade da partícula representar a real localização do robô. Desta forma, as partículas com os pesos mais baixos são eliminadas e as com pesos mais altos são replicadas, de tal forma que após algumas iterações todas as partículas irão convergir ao redor da localização correta do robô. Isso torna-se uma desvantagem para casos de rapto do robô ou para casos em que o algoritmo localiza erroneamente, pois uma vez que as partículas convergem em um local, elas permanecerão ao redor deste, mesmo que seja uma localização errada (FOX et al., 1999).

EKF possui as vantagens de ser robusto e fornecer uma resposta ótima, porém sua característica unimodal impossibilita seu uso na localização global, dessa forma, para sua utilização a postura inicial sempre deverá ser fornecida. Entretanto, existem casos em que fornecer esse dado da localização é inviável ou, até mesmo impossível.

Por outro lado, a localização baseada em Grid é multi-modal e é capaz de solucionar a deficiência do EKF, além da sua solução englobar todos os problemas da localização, entrentanto,requerendo um processamento muito maior. Caso os dados das medidas não sejam tratados até o fim do ciclo, eles serão perdidos afetando a informação do estado, inviabilizando assim sua utilização em aplicações em tempo real.

Por fim, o MCL tem a capacidade de localizar o robô localmente e globalmente, apesar do problema de sequestro não ser tratado. Caso as partículas convirjam em uma postura errada, o algoritmo não possui uma rotina para identificar e corrigir, mantendo o robô constantemente perdido. Entretanto, é uma abordagem bastante veloz e pode ser utilizada em tempo de execução, possui uma facilidade de implementação por ser não-paramétrica e é bastante versátil, sendo facilmente alterável.

2.1.1 Localização de Monte Carlo

Localização de Monte Carlo (MCL) é baseada no filtro de partículas (FP), o qual representa a crença da localização a posteriori $Bel(s_t)$ através de um conjunto de partículas S_t distribuído de acordo com esta crença. Cada partícula s_t é interpretada como um pseudo-robô com posição e orientação, e também possui um fator de importância w_t, conhecido como peso que indica o quão próximo a localização desta partícula está da localização

real do robô (ZHANG, 2010).

$$S_t =< s_t^n, w_t^n >,\ n = 1, ..., N \tag{2.3}$$

onde cada partícula s_t^n representa a postura do robô no tempo t, N é o número de partículas do conjunto S_t e w_t^n os pesos com valores não negativos.

Neste algoritmo a primeira etapa é a criação de um número de partículas espalhadas randomicamente por todo o mapa, sendo que cada uma representa um robô com uma postura (x, y, θ) aleatória. A partir da movimentação do robô e a cada iteração é recalculada a probabilidade, através da regra de Bayes, de cada partícula estar na posição correta, sendo que as com menor probabilidade são realocadas em locais onde as partículas possuem maiores probabilidades. Dessa forma, algumas áreas apresentarão 'nuvens' de partículas maiores e outras menores. Com o decorrer do deslocamento do robô, as partículas irão convergir ao redor da posição real e resultar em apenas uma nuvem, a qual conterá a localização do robô (ROMERO et al.,).

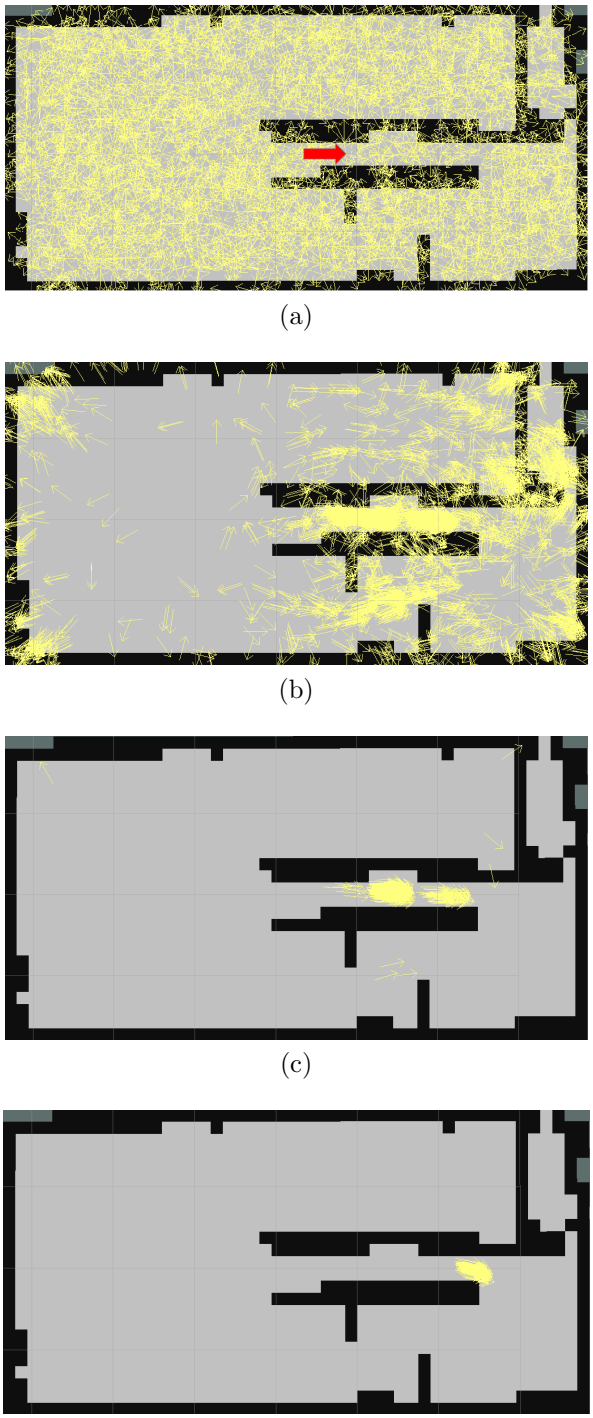
O funcionamento do FP é ilustrado na figura 2.1.1, onde é possível visualizar a evolução da crença da localização do robô. Primeiramente, as partículas são espalhadas uniformemente sobre a área livre do mapa. Através do deslocamento do robô, novas observações são feitas pelos sensores e as crenças são atualizadas, resultando em aglomerados de partículas sob localizações com maior probabilidade de se encontrar a postura do robô.

Como o MCL é um filtro bayesiano, ele possui dois modelos probabilísticos: (a) de predição, e (b) de atualização. O primeiro é o modelo de ação e sua probabilidade condicional é dada como $P(S_t|S_{t-1}, u_t)$, sendo u_t o movimento aplicado à amostra. Enquanto o segundo é o modelo de percepção, dado por $P(z_t|S_t)$. Na etapa de atualização, as observações z_t são utilizadas para calcular os pesos w_t^n utilizados no processo conhecidos como *resampling*. A normalização dos pesos é feita através da equação 2.4.

$$w_t^n = \frac{w_{t-1}^n P(z_t|S_t)}{\sum_{n=1}^{N} w_{t-1}^n P(z_t|S_t^n)} \tag{2.4}$$

O algoritmo básico do MCL é descrito no Algoritmo 1, sendo sua primeira etapa gerar as partículas s_t^n de forma aleatória sobre as áreas livres do mapa m, em seguida carregar os valores de S_{t-1}, u_t, z_t, m. Durante o movimento do robô, as etapas de predição e atualização são executadas (linhas 4 a 7). É necessário normalizar os pesos (linha 9) para executar a rotina de *resampling* (linhas 10 a 14), responsável por atualizar o conjunto de partículas resultando em uma aglomeração sobre as localizações com maiores probabilidades de se encontrar a postura do robô.

Resampling ou reamostragem é um processo que seleciona as partículas que serão eliminadas e as que serão duplicadas. Utiliza uma técnica de sorteio através dos pesos de tal forma que partículas com maiores pesos terão maiores probabilidades de serem duplicadas e as de menores pesos terão maiores probabilidades de serem eliminadas.



(a)

(b)

(c)

(d)

Figura 2.1.1 – Ilustração do funcionamento do filtro de partículas. As setas em amarelo representam as partículas e estão espalhadas no mapa conforme suas crenças da localização do robô. A seta vermelha representa a posição inicial do robô e este se movimenta em linha reta até o término do corredor. (2.1.1a) Inicialmente, as partículas são criadas com probabilidades uniformemente distribuídas sobre toda a área livre do mapa. (2.1.1b) Conforme o robô se movimenta, a probabilidade de cada partícula é atualizada e a rotina de *resampling* as realoca conforme seus pesos. (2.1.1c) A localização ainda possui algumas ambiguidades. Em (2.1.1d), após algumas iterações, o MCL define corretamente a localização atual do robô.

Um método utilizado por Udacity e Thrun (2011) para definir a postura da localização é através de uma função de validação (linha 15), a qual segue os seguintes passos: (a) através da postura de cada partícula, é feito uma média aritmética simples para x, y e θ, (b) em seguida, calculado o somatório das distâncias euclidianas (dadas a partir de cada partícula até a média calculada), (c) normaliza-se o valor encontrado e (d) caso seja menor que um valor de erro pré-definido, a postura é definida como o valor da média calculada anteriormente.

Algoritmo 1: Localização baseada em MCL

Entrada: S_{t-1}, u_t, z_t, m

1 **início**
2 **Predição e Atualização**
3 **para** $n = 1$ *até* N **faça**
4 Movimentar as partículas
5 $s_t^n \leftarrow P(s_t|s_{t-1}^n, u_t, m)$
6 Calcular os pesos
7 $w_t^n = P(z_t|s_t^n, m)$
8 **fim**
9 normalizar w_t através da equação 2.4
10 **Resamplig**
11 **para** $n = 1$ *até* N **faça**
12 Selecionar as partículas s_t^n a partir do peso w_t^n
13 Adicionar s_t^n ao S_t
14 **fim**
15 **Função de Validação**
16 **para** $n = 1$ *até* N **faça**
17 Média aritmética simples de todas as posturas
18 **fim**
19 **para** $n = 1$ *até* N **faça**
20 d = postura$_n$ − $media$
21 soma = $\sum_{n=1}^{N} \sqrt{d^2}$
22 **fim**
23 **Normalizar a soma**
24 $soma = \frac{1}{N} soma$
25 **se** $soma < erro$ **então**
26 Postura = média aritmética simples
27 **fim**
28 **fim**

Saída: S_t

MCL é um filtro não-paramétrico e portanto apresenta característica multi-modal. Assim esta abordagem é capaz de solucionar as localizações global e local, mas em situações de sequestro não é capaz de identificá-lo. Na prática, o aumento do número de partículas traz uma melhoria na precisão da localização, mas afeta negativamente na eficiência computacional.

Em relação ao uso dos ruídos da movimentação e da percepção, é interessante que não sejam muito pequenos, pois isso implicará em uma população de partículas com pouca diversidade após o processo de *resampling* e muito próximas da postura real do robô. O objetivo de se ter partículas não tão próximas é justamente de manter uma nuvem de partículas ao redor do robô para a correção de pequenos erros provenientes da cinemática. Uma solução simples para sensores muito precisos é adicionar ruídos artificiais à medição.

A seção a seguir abordará métodos utilizados para melhorar a localização baseada em Monte Carlo. Formas de otimização, identificação e tratamento para casos de sequestro do robô, além de métodos híbridos.

2.2 Trabalhos Relacionado à Localização Robótica

A localização é a base da navegação autônoma e sua eficiência influencia diretamente na execução de tarefas de alto nível, portanto é uma das áreas da robótica mais exploradas nos últimos anos. Sendo assim, várias pesquisas com abordagens distintas foram feitas para tratar deste problema, as quais são divididas em duas categorias: (a) localização relativa, que foca nas técnicas de rastreamento de posição, e (b) localização absoluta, técnicas que localizam o robô em relação ao mapa através de leituras de marcos (ZHOU; LIU; HUANG, 2007).

O Filtro de Kalman Estendido (EKF) é o algoritmo mais utilizado para resolver o rastreamento de posição, sendo a postura do robô representada por uma distribuição Gaussiana unimodal. Como o sistema é não-linear, é necessário passar por um processo de linearização da propagação das médias e covariâncias, *i.e.*, calcular a jacobiana do sistema e do modelo de observação (JETTO; LONGHI; VENTURINI, 1999). Visando eliminar este processo, Wan e Merwe (2000) propuseram o *Unscented Kalman Filter - UKF* que através de uma transformada realiza a estimação do estado de um sistema não-linear não necessitando do processo de linearização. Porém, essas abordagens tratam apenas do problema da localização local, assim Merwe et al. (2000) propuseram o *Unscented Particle Filter*, que utiliza o UKF para cada partícula a fim de estimar a postura atual com maior precisão. Esta técnica apresentou uma melhoria considerável ao filtro de partículas, porém requer uma complexidade adicional para modelar o sistema e o modelo de observação.

Infelizmente, nenhuma destas técnicas, inclusive o filtro de partículas, são capazes de solucionar o problema de sequestro do robô. Visando resolvê-lo, Thrun, Burgard e Fox (2005) propuseram o *Augmented MCL* que é um algoritmo baseado em localização por

Monte Carlo. Ao identificar um sequestro, mais partículas são criadas e espalhas pelo mapa, tornando a relocalização muito lenta e pouco eficiente. Nota-se que abordagens de uma única técnica não tratam todos os problemas da localização, apesar de apresentarem grandes vantagens em certos pontos, há desvantagens em outros. Portanto, uma abordagem híbrida torna-se bastante interessante e algumas pesquisas serão apresentadas a seguir.

2.2.1 Algoritmos Híbridos

Um algoritmo híbrido criado por Baltzakis e Trahanias (2002) tem como proposta um *framework* probabilístico para modelagem do estado do robô e informações dos sensores, os quais são baseados no modelo de *switching state-space*. O *framework* generaliza os modelos probabilísticos de Filtro de Kalman linear e modelo oculto de Markov, combinando as vantagens de ambos os modelos.

Gasparri et al. (2007) propuseram uma ideia semelhante e sua implementação é formada por duas etapas. Primeiro um filtro de partículas é utilizado para gerar hipóteses sobre uma postura possível, supondo que não há movimentos para evitar colisões. Posteriormente, trajetórias seguras são planejadas e executadas para reduzir as ambiguidades restantes, enquanto as hipóteses são monitoradas e validadas por um conjunto de filtros de Kalman estendido.

Prestes, Ritt e Fuhr (2008) apresentam uma combinação da localização de Monte Carlo com um planejador de trajetórias baseado em problemas de valor de contorno (*BVP-path planner*) para localização global em ambientes esparsos. Este tipo de ambiente representa uma situação muito difícil para a localização, uma vez que várias regiões não fornecem informações relevantes para permitir que o robô se relocalize. A proposta é utilizar uma estratégia para distribuir as partículas apenas em partes relevantes do ambiente. Em seguida o robô é deslocado ao longo destas regiões através de uma solução numérica de um BVP envolvendo equação de Laplace.

3 Algoritmos para Localização Baseados em Filtro de Partículas

Vários algoritmos foram criados para tratar os problemas da localização, como apresentado na seção 2.2 e os que demonstraram melhores resultados foram os baseados em filtros de partículas. Sendo o *Kullback-Leibler Distance - KLD* o mais explorado, porém é um sistema incompleto em relação ao tratamento dos três problemas da localização. Devido a esta deficiência, conceitos de um outro algoritmo, o *Self-Adaptative Monte Carlo Localization - SAMCL*, foram implementados para suprir estas faltas e melhorar o desempenho da localização baseada no KLD. As seções seguintes irão detalhar cada um dos algoritmos e introduzir o sistema híbrido, *Self-Adaptative KLD Localization - SA-KLD*, proposto pelo trabalho.

3.1 KLD-Sampling

A ideia principal desta abordagem é determinar um número de partículas através da distância de Kullback-Leibler (KLD), de modo que a cada iteração o erro entre o real estado a posteriori e a aproximação do estado baseado nas partículas seja menor que um *threshold* ϵ (FOX, 2003).

3.1.1 Distância de Kullback-Leibler

A distância de Kullback-Leibler é uma medida de distância entre duas distribuições de probabilidade, sendo que seu resultado é uma medida de ineficiência de uma distribuição q em relação a uma distribuição verdadeira p, ou seja, quanto menor o valor desta distância, mais próximo q está de p (PAVÃO, 2011). Assim, o número de partículas será determinado conforme a distância entre a postura determinada pela verossimilhança máxima estimada – do inglês *Maximum Likelihood Estimate* (MLE) – das amostras e a real postura a posteriori. A equação 3.1 representa a distância KL entre duas distribuições de probabilidade p e q.

$$K(p,q) = \sum p_i \log \frac{p_i}{q_i} \tag{3.1}$$

onde p_i e q_i são as probabilidades do evento i de uma amostra aleatória.

A fim de calcular o número de partículas necessárias n, o algoritmo do filtro de partículas básico precisa ser modificado. Inicialmente, o ambiente deve ser discretizado em várias células de mesmas dimensões, denominadas *bins*. Assim, supondo que estas n partículas são distribuídas de forma discreta em diferentes k bins e o número de partículas

por *bin* é representado por $X = (x_1, ..., x_k)$ que são distribuídos de acordo com uma distribuição multinomial, *i.e.*, $X \sim Multinomial_k(n, p)$, onde $p = p_1, ..., p_k$ representa a real probabilidade de cada *bin*. Portanto, a máxima verossimilhança estimada de p usando n partículas é dado por $\widehat{p} = n^{-1}X$ e a estatística da razão de verossimilhança λ_n é:

$$\log \lambda_n = \sum_{j=1}^{k} X_j \log \frac{\widehat{p}_j}{p_j} \tag{3.2}$$

Se X_j for idêntico ao $n\widehat{p}_j$ tem-se:

$$\log \lambda_n = n \sum_{j=1}^{k} \widehat{p}_j \log \frac{\widehat{p}_j}{p_j} \tag{3.3}$$

Através das equações 3.1 e 3.3 pode-se notar que a estatística da razão de verossimilhança é n vezes a distância KL entre o MLE e a real distribuição, tem-se assim:

$$\log \lambda_n = nK(\widehat{p}, p) \tag{3.4}$$

Isso mostra que a razão de verossimilhança converge para uma distribuição chi-quadrado[1] com $k - 1$ graus de liberdade.

$$2 \log \lambda_n \rightarrow_d \chi^2_{k-1}, \text{ com } n \rightarrow \infty \tag{3.5}$$

Se a probabilidade da distância KL entre a verdadeira distribuição p e o MLE for menor ou igual a um limite, denominado por ϵ, tem-se que:

$$P_p(K(\widehat{p}, p) \leq \epsilon) = P_p(2nK(\widehat{p}, p) \leq 2n\epsilon) \tag{3.6}$$

$$= P_p(2 \log \lambda_n \leq 2n\epsilon) \tag{3.7}$$

$$= P(\chi^2_{k-1} \leq 2n\epsilon) \tag{3.8}$$

Através de 3.4 tem-se 3.7, e de 3.5 tem-se 3.8. O quantil da distribuição de chi-quadrado é dado por:

$$P(\chi^2_{k-1} \leq \chi^2_{k-1,1-\delta}) = 1 - \delta \tag{3.9}$$

Se o valor de n for escolhido de forma que $2n\epsilon = \chi^2_{k-1,1-\delta}$, ao combinar as equações 3.8 e 3.9 tem-se:

$$P_p(K(\widehat{p}, p) \leq \epsilon) = 1 - \delta \tag{3.10}$$

[1] A distribuição chi-quadrado é um método não-paramétrico e utilizado para testes de hipóteses para avaliar a relação entre as frequências observadas e esperadas de um determinado evento. Assim, quanto menor a diferença entre o valor observado e o de controle, maior será o grau de semelhança entre as amostras (CONTI, 2009).

Através da equação 3.10 tem-se uma relação entre o número de partículas e a qualidade da aproximação. Resumindo, o número n de partículas escolhido deverá ser:

$$n = \frac{1}{2\epsilon}\chi^2_{k-1,1-\delta} \tag{3.11}$$

Assim, é possível garantir com uma probabilidade de $(1-\delta)$ que a distância KL entre o MLE e a distribuição verdadeira é menor que o *threshold* ϵ. Para determinar n usando 3.11 é necessário calcular os quantis da distribuição chi-quadrado a cada iteração. Portanto, para reduzir este trabalho, é usada uma aproximação dada pela transformação de Wilson-Hilferty:

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

onde $z_{(1-\delta)}$ é o quantil de ordem $(1-\delta)$, obtido da tabela de distribuição normal (TEAM-MATH, 2011) e k é a quantidade de *bins* ocupados pelas partículas. Assim, para cada iteração do filtro, o erro entre o MLE e a distribuição verdadeira é menor que ϵ com probabilidade $(1-\delta)$.

3.1.2 Aplicação do KLD em filtros de partículas

O primeiro passo para implementar o KLD-sampling em um filtro de partículas é definir o tamanho dos *bins*, ou seja, o intervalo de discretização do mapa. Grandes dimensões de *bin* geram poucos k *bins* (número de células com dimensões bin x bin), sendo necessários um número de partículas n menor. Por outro lado, dimensões pequenas de *bin* resultarão em maior processamento para identificar e contar os *bins* ocupados, assim, devido a este aumento de precisão, um maior número de partículas serão geradas, resultando em maior custo computacional e maior tempo de localização. Além do tamanho do *bin*, os valores de ϵ e $z_{(1-\delta)}$ são previamente definidos, portanto em tempo de execução, apenas a quantidade k de *bins* influenciará no tamanho de n.

A implementação segue os seguintes passos: para cada iteração, deverão ser contados quantos k *bins* as partículas estão ocupando[2], assim, conforme o conjunto de amostras converge, o número de k *bins* ocupados também reduz e consequentemente o número de partículas n também será menor. Caso a verossimilhança diminua e o conjunto de amostras fique mais espaçado, implicará em mais k *bins* ocupados; este valor de k irá influenciar no aumento da quantidade de partículas necessárias. Portando, da mesma forma que o algoritmo reduz o número de partículas, ele também aumenta essa quantidade conforme o necessário.

Nota-se que é uma abordagem que trata apenas das localizações locais e globais. Porém, sua característica adaptativa do tamanho do conjunto de amostras em tempo de execução é bastante eficiente e reduz muito o custo computacional.

[2] É considerado como *bin* ocupado caso a célula contenha pelo menos uma partícula.

O Algoritmo 2 mostra os passos da localização baseada em KLD.

Algoritmo 2: Localização baseada em KLD-Sampling

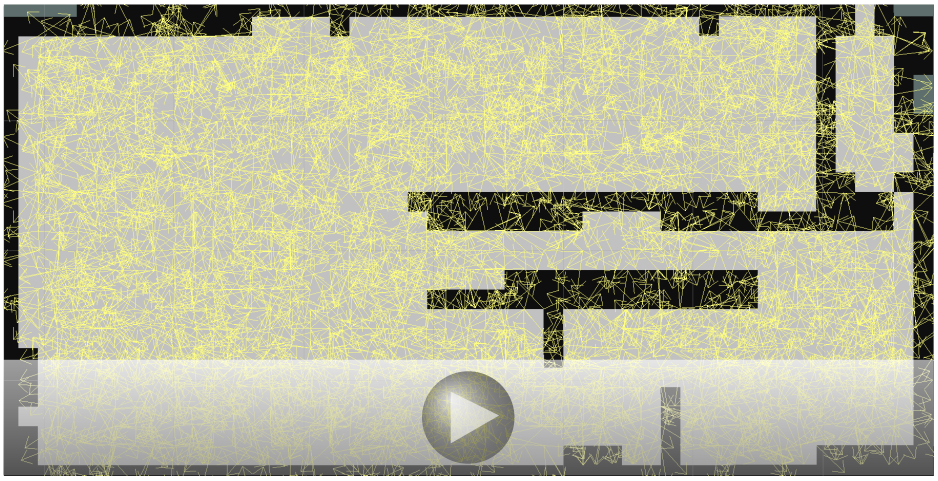
Entrada: $S_{t-1}, u_t, z_t, \epsilon, z_{(1-\delta)}$, *dimensão do bin*

início
  Criação de partículas
  para $n = 1$ *até* n_{max} **faça**
    movimentar a partícula s_t^n baseado em $P\left(s_t|s_{t-1}^n, u_t\right)$
    calcular o peso $w_t^n = P\left(z_t|s_t^n\right)$
  fim
  Resamplig
  normalizar w_t
  para $n = 1$ *até* n_χ **faça**
    calcular o peso $w_t^n = P\left(z_t|s_t^n\right)$
    atualizar s_t usando w_t^n
  fim
  Contagens de bins ocupados
  para *todos os bins* **faça**
    se *bin* == *ocupado* **então**
      k += 1
    fim
  fim
  Cálculo do número de partículas necessárias
  $\mathrm{n}_\chi = \frac{k-1}{2\epsilon}\left\{1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}}z_{(1-\delta)}\right\}^3$
fim

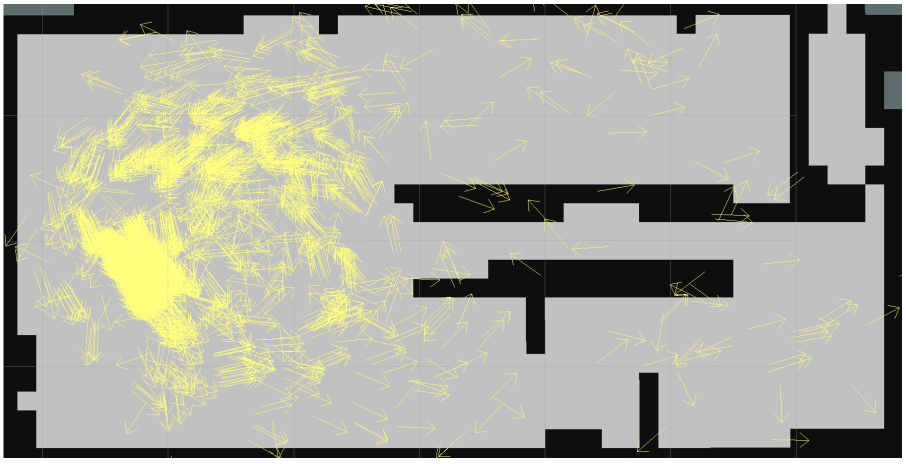
Saída: S_t

A figura 3.1.1 ilustra o algoritmo em funcionamento, onde o número de partículas é reduzido com o decorrer das iterações, sendo que cada partícula é representada por uma seta amarela.

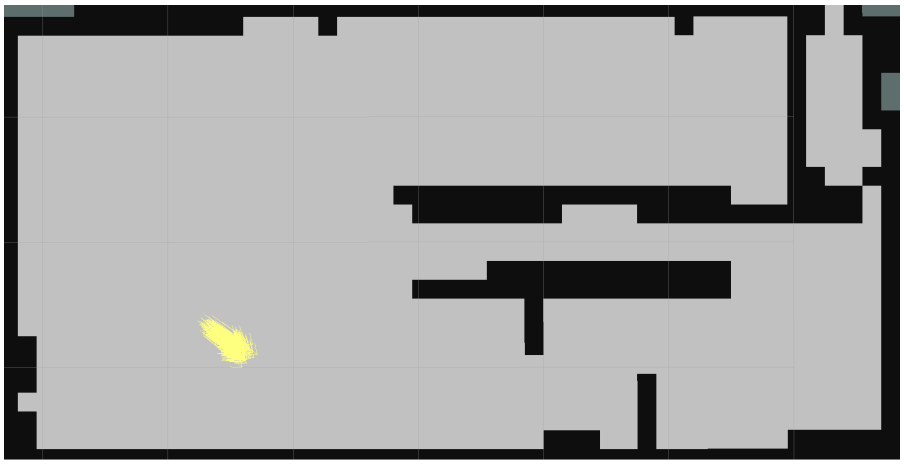
Embora o KLD reduza o número de partículas, o fato do algoritmo criá-las espalhadas por todo mapa, torna-o mais lento em suas primeiras iterações. Experimentos feitos por Lauer, Lange e Riedmiller (2005) mostraram que aproximadamente 98% das partículas inicialmente criadas não contribuíram para determinar a posição final estimada. Portanto, para um algoritmo de localização mais eficiente, é de grande interesse que as partículas sejam distribuídas em regiões com maiores probabilidades de conter a postura correta do robô. A seção 3.2 introduzirá uma outra abordagem, a qual procura tratar também deste problema.



(a) 5000 partículas



(b) 3630 partículas



(c) 230 partículas

Figura 3.1.1 – Localização com KLD: (3.1.1a) ao inicializar o algoritmo, o número de partículas é máximo (5000). (3.1.1b) Com o deslocamento do robô, as partículas começaram a convergir na posição real e, como consequência, a quantidade de amostras foram reduzidas para 3630. Em (3.1.1c) as partículas convergiram na localização real e o rastreamento de posição é feito com em média 230 partículas.

3.2 Self-Adaptative Monte Carlo Localization (SAMCL)

Apesar do MCL resolver os problemas de localização global e local, a precisão e o custo computacional são dependentes da quantidade de partículas utilizadas, além de não possuir tratamento para falhas de localização. Em contrapartida, o algoritmo baseado em KLD-sampling, introduzido na seção 3.1, tem a capacidade de ajustar o número de partículas, aliviando o processamento. Porém também não possui rotina para o sequestro do robô e suas partículas são espalhadas pelo mapa todo, tornando sua localização mais lenta e computacionalmente custosa durante as primeiras iterações.

Em busca de um algoritmo que trate os três problemas da localização, Zhang (2010) propôs um sistema de localização de Markov adaptativo, do inglês *Self-Adaptative Monte Carlo Localization* – SAMCL, o qual introduziu três conceitos muito importantes:

- A técnica de pré-armazenagem, essencial para a redução do processamento em tempo real, cuja função é criar dois tipos de matrizes (*grids*) na fase offline. A primeira contendo possíveis posturas do robô (x, y, θ), denotada por G_{3D} e a segunda formada por informações de energias e orientações, G_E.

- Tal energia é uma importante informação extraída dos sensores de percepção e atribuída a cada célula do mapa. Através dos dados do G_E são calculadas as Regiões de Energias Similares, do inglês *Similar Energy Region* – SER, usadas para selecionar as áreas com maior probabilidade que contenham a posição real do robô. Dessa forma, a criação de partículas será de forma randômica apenas nestas áreas, ao invés do mapa todo, resultando em uma localização mais eficiente e necessitando de um número menor de partículas.

- Diferentemente do KLD-sampling, cujo algoritmo trabalha com um conjunto de amostras de tamanho adaptativo, o SAMCL usa um tamanho fixo de partículas. Este conjunto de amostras é divido automaticamente sempre que a função de sequestro é ativada, sendo uma porção destinada às amostras locais e a outra em amostras globais. A primeira é usada para manter as partículas na postura atual para o rastreamento de posição, enquanto as amostras globais são espalhadas aleatoriamente pelas áreas SER a fim de encontrar a nova postura.

3.2.1 Etapas da localização baseada em SAMCL

O algoritmo SAMCL é implementado em três etapas e seu esquemático é mostrado na Figura 3.2.1.

- **Pré-armazenamento:** É a primeira etapa, na qual são criadas as matrizes G_{3D} e G_E através dos dados do mapa m e dos modelos de movimentação e de percepção.

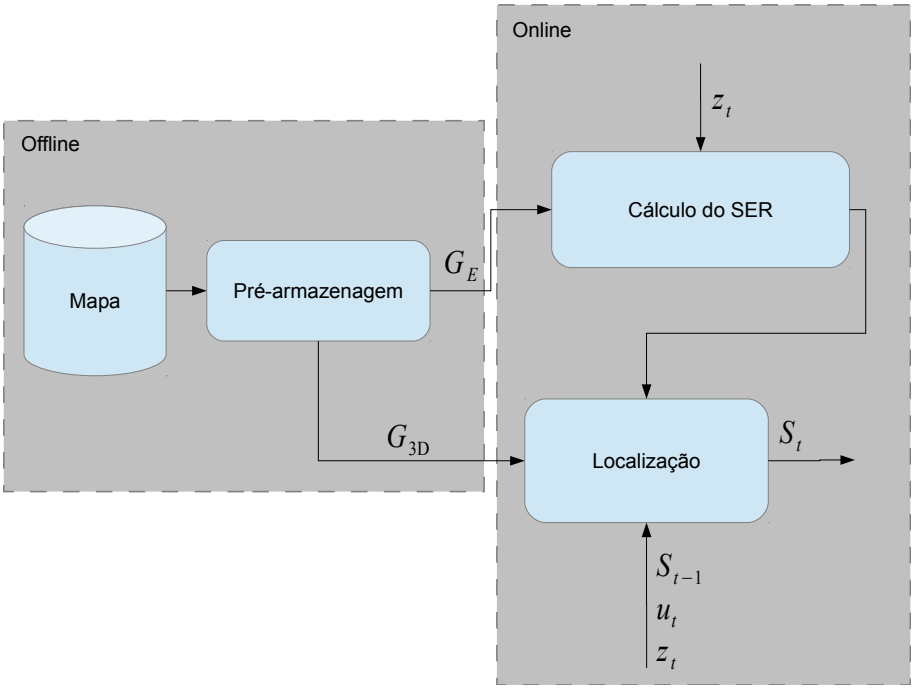


Figura 3.2.1 – Esquemático das etapas do SAMCL.

Como este processo é executado offline, o custo computacional para a criação destas matrizes não afeta o processamento em tempo-real.

- **Cálculo do SER:** Assim que a etapa offline é concluída, os dados da matriz de energia G_E e os dados dos sensores de percepção z_t são carregados, gerando o SER.
- **Localização:** A última etapa é a localização propriamente dita. Ela recebe como entradas o conjunto de partículas S_{t-1}, dados de controle u_t, dados de observação z_t, a matriz de posturas G_{3D} e o SER. Tem como saída o conjunto de partículas atualizado S_t.

3.2.2 Etapa de pré-armazenamento

Para solucionar o problema da localização robótica, o mapa do ambiente deverá ser previamente conhecido, do tipo grade de ocupação e estático. Então, o algoritmo decompõe este mapa em diversas células e, através do modelo de observação, computa os dados para cada uma, gerando as matrizes G_{3D} e G_E.

- **Matriz tridimensional (G_{3D}):** Composta por coordenadas cartesianas (x, y) e orientação (θ) de pseudo-robôs representados em cada grade do mapa. Como o processo é realizado offline, para encontrar as partículas com as posturas desejadas basta executar um algoritmo de busca, reduzindo o esforço computacional.
- **Matriz bidimensional de energia (G_E):** Formada por dados de energias e orientações dos pseudo-robôs presentes em cada célula do mapa. O cálculo da energia

é feito através das medições do modelo de observação de cada pseudo-robô, sendo que para cada dado de distância z é calculada por $1 - z_i/z_{max}$, onde o z_{max} é o seu alcance máximo. Se o robô possuir mais de um sensor, é feito uma somatória das energias para cada orientação dada. Caso a distribuição dos sensores ao redor do robô for um anel de sensores igualmente espaçados, não há necessidade de armazenar a orientação dos pseudo-robôs.

Os passos para o cálculo da energia são mostrados no Algoritmo 3. A partir dos dados do mapa m, o cálculo da energia e_i^k é feita para cada sensor i do pseudo-robô presente na célula k (linha 4). Na linha 5, o cálculo da energia total do pseudo-robô é feita através da somatória da energia de todos os seus sensores. Na linha 7, é feito a normalização da energia para que seu valor esteja no intervalo [0,1].

Algoritmo 3: CÁLCULO DA ENERGIA PARA CADA CÉLULA DO MAPA

Entrada: m

1 **início**
2 **para** *todas as células* $k \in \{1, ..., K\}$ **faça**
3 **para** *todas as leituras dos sensores* $i \in \{1, ..., I\}$ **faça**
4 $e_i^k = 1 - z_i^k / z_{max}$
5 $E(k) = \sum_{i=1}^{I} e_i^k$
6 **fim**
7 normalizar $E(k) = \frac{1}{I} E(k)$
8 **fim**
9 **fim**

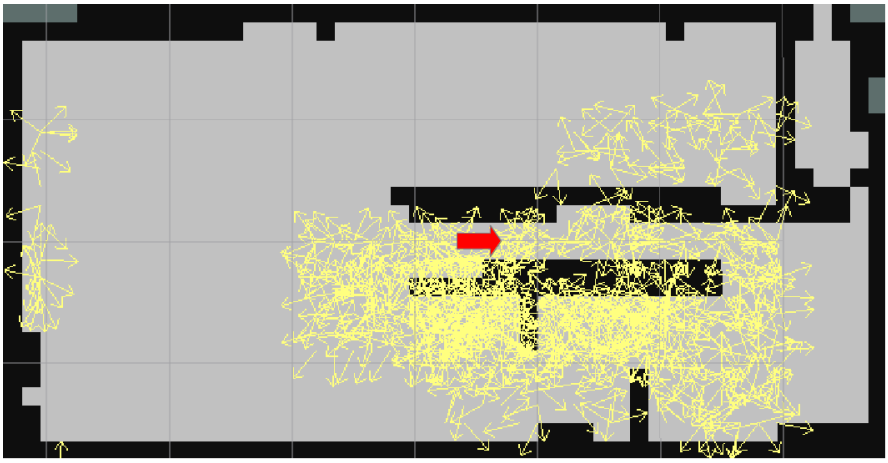
Saída: G_E

3.2.3 Cálculo do SER

Regiões de energia similar (SER) são áreas onde as energias dos pseudo-robôs são aproximadamente iguais à energia do robô real ou, em outras palavras, são áreas com uma probabilidade alta de conterem a localização correta. Dessa forma, as partículas são espalhadas apenas em algumas áreas e consequentemente um número menor de partículas serão necessárias, reduzindo assim o tempo de localização e custo computacional.

Caso o robô inicie a localização no corredor ou em uma área aberta, a quantidade de áreas SER será diferente devido à energia calculada para cada situação. A figura 3.2.2 ilustra bem esta situação, sendo que foram utilizadas 1000 partículas; cada uma é representada por uma seta amarela e a postura inicial pela seta em vermelho.

O Algoritmo 4 apresenta os passos para o cálculo do SER. Como entrada são usados os dados de G_E e dos sensores do robô real z_t no tempo t. As linhas 2 até 6 são as sequências para o cálculo da energia do robô. A linha 8 mostra matematicamente o modo como as



(a) Corredor

(b) Área aberta

Figura 3.2.2 – Diferença da quantidade de áreas SER para os casos em que o algoritmo inicializa o robô (3.2.2a) no corredor e (3.2.2b) em uma área aberta do mapa. A seta em vermelho representa a postura inicial do robô e em amarelo as partículas.

áreas SER são determinadas: somente se o módulo da diferença da energia do robô com

a energia de cada pseudo-robô for menor que um *threshold*, δ.

Algoritmo 4: CÁLCULO DO SER

Entrada: G_E, z_t

início

  para *todas as leituras dos sensores do robô real* $i \in \{1, ..., I\}$ **faça**

    $e_i = 1 - z_i / z_{max}$

    $E_{real} = \sum_{i=1}^{I} e_i$

  fim

  normalizar $E_{real} = \frac{1}{I} E_{real}$

  para *todas as células* $k \in \{1, ...K\}$ **faça**

    armazenar as células k cujo SER torne a expressão $|E - E(k)| < \delta$ verdadeira

  fim

fim

Saída: SER

3.2.4 Localização baseada em SAMCL

As partículas auto-adaptativas do SAMCL são responsáveis pela solução do problema do rapto do robô. Quando a localização é bem definida, apenas as amostras locais são usadas, mas nos casos de sequestro do robô o conjunto de partículas se dividem em amostras locais e globais. As locais continuam próximas da localização a priori e as globais se espalham através das áreas SER. Assim que a nova localização é encontrada, todas as amostras tornam-se locais.

O processo de localização do SAMCL é representado no Algoritmo 5 e tem como entradas: (a) o conjunto de partículas a priori, S_{t-1}, (b) o controle de movimento u_t, (c) leitura de todos os sensores de percepção z_t, (d) a matriz G_{3D}, e (e) SER. Tem como saída um conjunto de partículas atualizado no tempo t, S_t. As variáveis N_T, N_G e N_L são o número de partículas total, global e local, respectivamente. O algoritmo pode ser dividido em cinco partes (não necessariamente nesta ordem):

- **Parte 1 – Criação/movimentação de partículas:** Dadas as partículas no tempo $t - 1$, estas são deslocadas através do controle de movimento u_t do robô e suas posturas são atualizadas no tempo t (linha 4). Na linha 5 é calculado o peso de cada partícula.

- **Parte 2 – Determinando o tamanho das amostras globais e locais:** Analisa-se o maior peso dentre as partículas e se for menor que ξ, uma porção torna-se amostras globais e a outra mantém-se como amostras locais, sendo α o fator que determina esta proporção. Sempre que as amostras globais são criadas, significa

que o algoritmo identificou que houve um rapto. Porém, há casos em que ocorrem os falsos sequestros (erros nos dados dos sensores ou mapa incompleto), por isso é importante manter uma porção maior de amostras locais para que estes erros interfiram o mínimo possível na localização.

- **Parte 3 – *Resamplig* das amostras locais:** Esta etapa é idêntica ao *resampling* do algoritmo MCL; partículas com pesos pequenos são realocadas em posturas de partículas com pesos maiores.

- **Parte 4 – Criando as amostras globais:** Ocorre apenas quando é ativada a função de sequestro. Como as partículas são espalhadas através das áreas SER, isso torna o algoritmo mais eficiente do que as abordagens que as distribuem através de toda a área livre do mapa.

- **Parte 5 – Unindo os dois conjuntos de partículas:** Por fim, os dois conjuntos de amostras locais e globais são unidos, gerando um novo conjunto a posteriori que será usado na próxima iteração.

Algoritmo 5: Localização baseada em SAMCL

Entrada: S_{t-1}, u_t, z_t, G_{3D} e SER

início
  Criação/movimentação de partículas
  para $n = 1$ *até* N_T **faça**
    movimentar a partícula s_t^n baseado em $P\left(s_t|s_{t-1}^n, u_t\right)$
    calcular o peso $w_t^n = P\left(z_t|s_t^n, G_{3D}\right)$
  fim
  Determinando o tamanho das amostras globais e locais
  se $w_t^{max} < \xi$ **então**
    $N_L = \alpha * N_T$
    senão
      $N_L = N_T$
    fim
  fim
  $N_G = N_T - N_L$
  Resamplig das amostras locais
  normalizar w_t
  para $n = 1$ *até* N_L **faça**
    atualizar $s_t^{n,L}$ usando w_t^n
    adicionar $s_t^{n,L}$ em S_t^L
  fim
  Criando as amostras globais
  para $n = 1$ *até* N_G **faça**
    criar $s_t^{n,G}$ usando distribuição uniforme através do SER
    adicionar $s_t^{n,G}$ em S_t^G
  fim
  Unindo os dois conjuntos de partículas
  $S_t = S_t^L \cup S_t^G$
fim

Saída: S_t

Apesar do algoritmo tratar os três problemas da localização e espalhar as partículas em áreas SER, seu conjunto de partículas é fixo. Assim, com o aumento do mapa, um número maior de partículas são necessárias para garantir uma boa precisão na localização. Desta forma, na seção 3.3 é introduzida uma proposta de uma nova abordagem, a qual utiliza as características e propriedades do KLD e do SAMCL para realizar uma localização mais eficiente.

3.3 Self-Adaptative KLD Localization (SA-KLD)

Visando um método de localização eficiente, robusto e que resolva os problemas de localização local, global e rapto do robô, o novo algoritmo proposto tem como objetivo uma abordagem híbrida para obter tais especificações, o *Self-Adaptative KLD Localization* – SA-KLD.

Como apresentado na seção 3.1 o KLD é uma abordagem muito eficiente para adaptar o tamanho do conjunto de partículas do filtro de partículas, tornando-o muito mais eficiente se comparado às abordagens de conjunto fixo. Entretanto, nos casos em que o KLD converge numa localização errada, a postura do robô continuará sempre errada. Por outro lado, o SAMCL apresentado na seção 3.2 corrige este problema, além de aumentar o desempenho da localização global. Porém, sua maior desvantagem está em trabalhar com um conjunto fixo de partículas. Em outras palavras, o KLD é um FP que resolve muito bem a localização local e também trata da localização global, mas de uma forma não eficiente, enquanto o SAMCL traz uma solução melhor para a localização global e sequestro do robô, mas possui restrições quanto à localização local, por exemplo um mapa de grandes dimensões.

Com base nestes algoritmos, foi desenvolvido um novo método de filtro de partículas, o qual possui um tamanho adaptativo do conjunto de partículas e faz o espalhamento apenas nas áreas com maior probabilidade, além de solucionar os três problemas da localização. As desvantagens presentes em cada abordagem são solucionadas graças à junção dos métodos, desta forma, um algoritmo torna-se complementar do outro gerando uma abordagem de localização robótica mais versátil e completa.

O maior diferencial deste novo método está na rotina de criação de partículas. Diferentemente do KLD e do SAMCL que inicialmente criam as partículas sempre com um tamanho do conjunto de amostras fixo, o SA-KLD analisa as áreas SER e calcula o número de partículas necessárias, mantendo assim a precisão da localização, reduzindo o custo computacional e o tempo para convergir.

Através do algoritmo do SAMCL foram realizadas algumas modificações: (a) tamanho do conjunto de partículas variável a cada processo de *resampling* e a cada etapa de criação de partículas (na fase inicial do algoritmo ou na ativação da função de sequestro), e (b) como as matrizes G_E e G_{3D} possuem a variável de orientação em comum, foram unificadas e criada uma matriz de quatro variáveis, G_{4D}, contendo (x, y, θ e $energia$), a fim de facilitar no desenvolvimento do algoritmo, tanto para armazenamento quanto para a busca dos dados. Desta forma, apenas as duas primeiras partes tiveram alterações sendo adicionada uma etapa responsável pela variação do tamanho do conjunto de amostras, enquanto as outras etapas permaneceram inalteradas. Pode-se acompanhar abaixo os processos que sofreram alterações:

- **Parte 1 - Criação/movimentação de partículas:** O primeiro passo é calcular

o SER através dos sensores de observação e da matriz G_{4D} como mostrado no Algoritmo 4. Em seguida, é feita a contagem de *bins* ocupados pelas áreas SER calculando o número de partículas necessárias através da equação 3.12. Para cada partícula do tempo $t-1$, é feito um deslocamento conforme dados do controle de movimento u_t (linha 11). Na linha 12 é calculado o peso de cada partícula.

- **Parte 2 - Determinando o tamanho das amostras globais e locais:** Analisa-se o maior peso dentre as partículas e se for menor que ξ, o SER é calculado. Os *bins* ocupados pelas áreas SER são contados determinndo-se o número total de partículas N_T. Assim, uma parte de N_T torna-se amostras globais e a outra mantém-se como amostras locais. O fator α determina a proporção das amostras. Sempre que as amostras globais são criadas, significa que o algoritmo identificou que houve o sequestro. Porém, há casos em que ocorrem os falsos sequestros (erros nos dados dos sensores, mapa incompleto), por isso é importante manter uma porção de amostras locais para que estes erros interfiram o mínimo possível na localização.

- **Parte Adicional - Cálculo do número de partículas necessárias:** Após o *resampling*, são contados os *bins* ocupados e através da equação 3.12 é determinado a quantidade de partículas necessárias.

A seguir é apresentado o Algoritmo 6 desenvolvido neste trabalho:

Algoritmo 6: LOCALIZAÇÃO BASEADA EM SA-KLD

Entrada: $S_{t-1}, u_t, z_t, G_{4D}$, *SER*, ϵ, δ, *dimensão do bin*

1 **início**
2 **Criação/movimentação de partículas**
3 Calcular as áreas SER
4 Contagem de k *bins* ocupados pelas áreas SER
5 Calcular o número de partículas N_T através da equação 3.12
6 $N_T = n_\chi$
7 **para** $n = 1$ *até* N_T **faça**
8 movimentar a partícula s_t^n baseado em $P\left(s_t | s_{t-1}^n, u_t\right)$
9 calcular o peso $w_t^n = P\left(z_t | s_t^n, G_{4D}\right)$
10 **fim**
11 **Determinando o tamanho das amostras globais e locais**
12 **se** $w_t^{max} < \xi$ **então**
13 Calcular as áreas SER
14 Contagem de k *bins* ocupados pelas áreas SER
15 Calcular o número de partículas N_T através da equação 3.12
16 $N_T = n_\chi$
17 $N_L = \alpha * N_T$
18 **senão**
19 $N_L = N_T$
20 **fim**
21 **fim**
22 $N_G = N_T - N_L$
23 **Resamplig das amostras locais**
24 Veja linhas 16 a 20 do Algoritmo 5
25 **Cálculo do número de partículas necessárias**
26 Contagem de k *bins* ocupados
27 Calcular o número de partículas N_T através da equação 3.12
28 **Criando as amostras globais**
29 Veja linhas 22 a 25 do Algoritmo 5
30 **Unindo os dois conjuntos de partículas**
31 Veja linha 27 do Algoritmo 5
32 **fim**

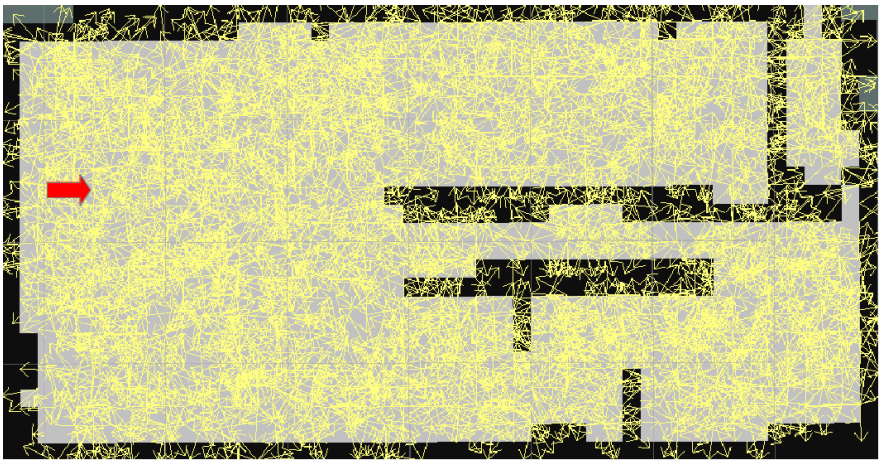
Saída: S_t

Como as partículas são criadas em função do espaço que as áreas SER ocupam, há situações em que existem poucas regiões com altas verossimilhanças. Por exemplo, na figura 3.3.1 em que o robô encontra-se em uma área aberta, assim, menos partículas serão

necessárias para uma boa localização.

A vantagem do SA-KLD não está apenas na fase de inicialização, mas também na rotina de sequestro. Uma vez que o sequestro é identificado, uma parcela do conjunto de partículas é espalhada pelas áreas SER e, se levado em consideração a situação da figura 3.3.1, o SA-KLD terá uma relocalização mais rápida, pois precisará gerar uma quantidade menor do que o número máximo de partículas. No Capítulo 5 são feitos estudos comparativos para mostrar a eficiência e robustez do algoritmo proposto no trabalho.

A figura 3.3.2 ilustra o fluxograma do processo geral da localização baseada no SA-KLD.



(a) KLD

(b) SAMCL

(c) SA-KLD

Figura 3.3.1 – Comparação do número de partículas, representadas pelas setas amarelas, e suas distribuições em torno do mapa para uma postura inicial, representada pela seta vermelha. Enquanto o (3.3.1a) KLD e o (3.3.1b) SAMCL sempre iniciam seus algoritmos criando o número máximo de partículas, na figura são utilizadas 5000, o (3.3.1c) SA-KLD precisa de apenas 975 para obter a mesma precisão de localização. Isso influencia diretamente no tempo para o algoritmo convergir em uma postura verdadeira do robô.

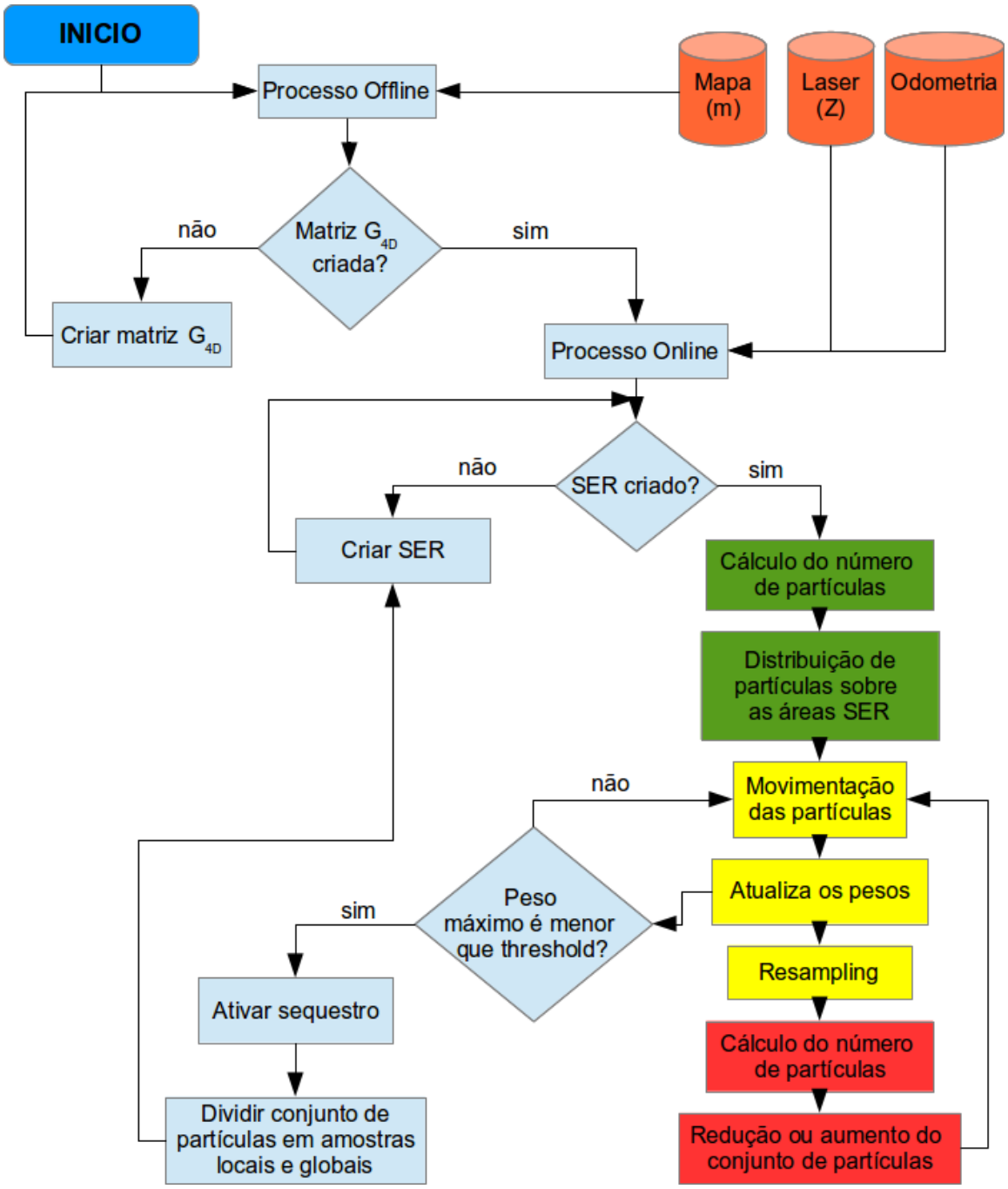


Figura 3.3.2 – Fluxograma do processo de localização baseado no SA-KLD. O algoritmo utiliza três técnicas para seu desenvolvimento: (a) os blocos em amarelo representam a localização de Monte Carlo simples citada na seção 2.1.1, o qual é utilizado por todas as técnicas, (b) em vermelho, a técnica de KLD-sampling, o qual ajusta o tamanho do conjunto de partículas em tempo de execução, (c) em azul são os blocos utilizados na localização baseada no SAMCL. Os dados do mapa, laser e odometria estão ilustrados em laranja e, por fim, os blocos em verde são implementações para que o SA-KLD distribua um conjunto variável de partículas sobre as áreas SER.

4 Modelagem do Problema

Este capítulo é destinado à apresentação das ferramentas utilizadas para os testes dos três algoritmos de localização, tratados no capítulo 3. Primeiramente serão detalhados os hardwares utilizados, em seguida o *framework* ROS (ROS, 2007) e seus pacotes, os modelos de cinemática e de observação.

4.1 Hardware

4.1.1 Robô Pioneer 3DX

Pioneer 3DX é um modelo de robô móvel ideal para uso *indoor*, da empresa *Adept MobileRobots*. Ele possui duas rodas em um eixo comum controladas independentemente e uma roda adicional do tipo castor, cuja função é apenas de apoio. Ao redor de sua estrutura há 16 sensores do tipo ultrassônicos, um giroscópio para medir o giro do robô e converter em orientação, um *encoder* em cada roda para a leitura da odometria, um microcontrolador com *firmware* ARCOS, um computador industrial modelo *Mamba EBX-37* que possui um processador *Dual Core* 2,26 GHz e 8 Gb de memória RAM, e uma conexão com protocolo TCP/IP (MOBILEROBOTS, 2011). A figura 4.1.1 ilustra o hardware citado.

Figura 4.1.1 – Pioneer 3DX com laserscan SICK LMS 291 instalado sobre a plataforma do robô móvel.

A empresa disponibiliza uma ferramenta para desenvolvimento de aplicações, *Pioneer Software Development Kit - Pioneer SDK*, a qual possui uma grande variedade de bibliotecas e aplicações robóticas para acelerar e ajudar no desenvolvimento de novos projetos. Um *framework* chamado *ARIA* também é fornecido para controle e leitura de dados de

diversos tipos de sensores, além de possuir suporte para comunicação entre robôs e também entre robô e computadores externos do tipo cliente-servidor. Também fornece um simulador de código aberto (*MobileSim*), uma interface gráfica para operações remotas e monitoramento dos robôs (*MobileEyes*), uma ferramenta de criação e edição de mapas para navegação robótica (*Mapper 3-Basic*) e vários outros *softwares* para desenvolvimento de robôs da plataforma *MobileRobots*.

Sua versatilidade é um dos seus pontos fortes, pois uma grande variedade de sensores podem ser adicionados em sua estrutura tais como manipuladores para executar funções de manipulação de objetos, câmeras do tipo usb, stereo ou PTZ para aplicações de visão robótica, sensores do tipo *laserscan* para mapeamento, localização e navegação, e vários outros tipos disponíveis no mercado. Dessa forma, o P3DX, como também é conhecido, tem a flexibilidade de ser usado em várias áreas de pesquisa: mapeamento, teleoperação, localização, monitoramento, reconhecimento, visão, manipulação, navegação autônoma e cooperação multi-robô.

Portanto, o conjunto *hardware* e *software* tornou os robôs da família Pioneer um dos robôs móveis mais utilizados no meio científico e acadêmico, sendo citado em vários outros trabalhos (KIM et al., 2007; MEI et al., 2005; OLIVEIRA et al., 2012; SASSI; WOLF,).

4.1.2 Master

Para fazer a leitura dos dados, executar os algoritmos e efetuar a localização do robô foi utilizado um computador não embarcado para processar os dados. Sendo suas especificações um processador i3 de 2,4 GHz e 4 Gb de memória RAM.

4.1.3 Sensor Laserscan SICK LMS 291

Utilizando o Pioneer como uma base móvel, sua função é fornecer dados de odometria e de controle, enquanto as medições de observação são dados pelo sensor do tipo *laserscan* da empresa *Sick* e modelo *LMS 291*, o qual é fixado no topo da plataforma do robô, figura 4.1.1.

O *lasercan* é um dispositivo que utiliza luz infravermelha com uma largura de onda λ conhecida para determinar a distância aproximada até um obstáculo (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011). Seu principal deferencial é criar vários feixes de luz lado-a-lado formando uma varredura bidimensional do ambiente, representado na figura 4.1.2. Esta varredura se deve à existência de um espelho interno do sensor que gira a altas rotações, assim espalhando os feixes pelo ambiente, figura 4.1.3. A figura 4.1.4 ilustra a emissão e recepção de um feixe do laser para determinar a distância do sensor até o objeto, dessa forma, conhecendo a frequência de modulação f e a velocidade da luz c, a largura de onda λ é definida pela equação $c = f\lambda$. Portanto, a distância total D_n do

laser até o ponto P é dado pela equação 4.1:

$$D_n = L + 2D = L + \frac{\theta}{2\pi}\lambda \tag{4.1}$$

onde D e L são distâncias definidas na figura 4.1.3, D_n é a distância de um feixe de luz do sensor ao objeto, λ o comprimento de onda, e θ o deslocamento de fase entre o sinal transmitido e o refletido.

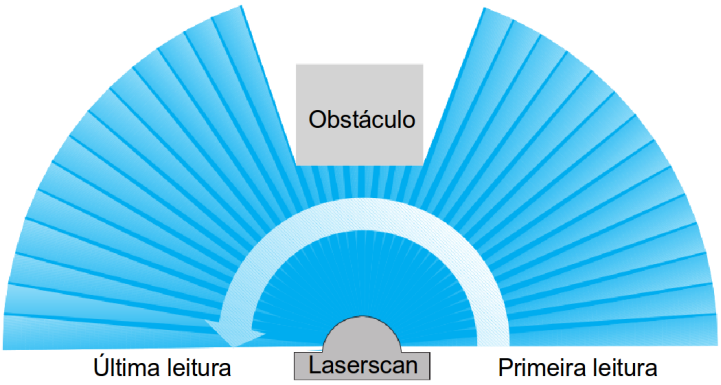


Figura 4.1.2 – Varredura bidimensional do ambiente através dos feixes do laserscan. Imagem adaptada de (SICK, 2006)

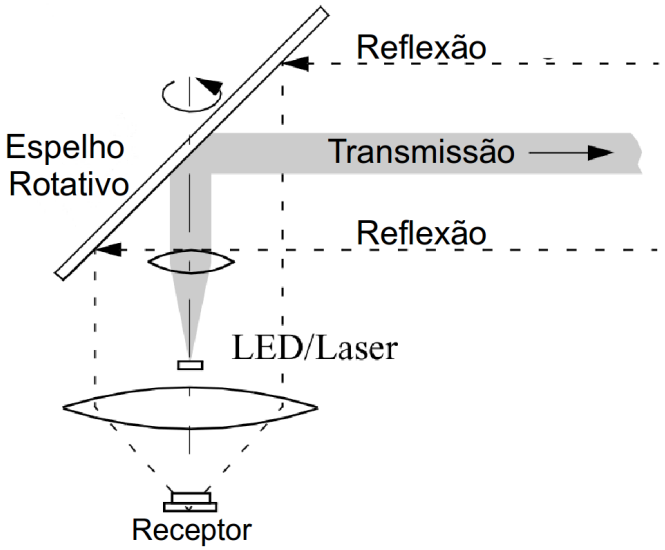


Figura 4.1.3 – Estrutura interna de um laserscan. Imagem adaptada de (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011)

O modelo utilizado para as experiências foi o *LMS 291* e possui as seguintes especificações listadas na tabela 4.1.1 (SICK, 2006).

Suas aplicações são inúmeras e não se restringem apenas na área da robótica, tais como: (a) determinação de volumes de objetos (medição de pacotes, malas, contêineres) (LEE, 2002), (b) determinação da posição de objetos (FOD; HOWARD; MATARIC, 2002), (c) prevenção de colisão de veículos (YOON; CRANE, 2008), (d) controle de processos de

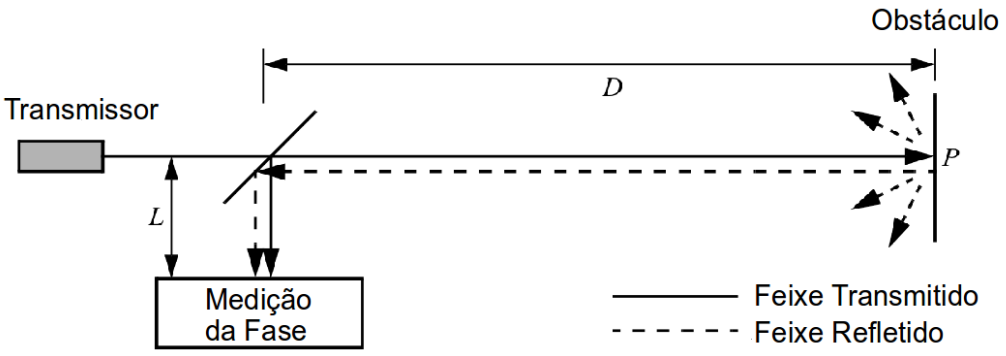


Figura 4.1.4 – Esquemático de um laserscan. Imagem adaptada de (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011)

Tabela 4.1.1 – Especificações do laserscan *Sick LMS 291*.

Ângulo de varredura	180°
Resolução Angular	0,25°; 0,5°; 1°
Resolução	10mm
Precisão	±35mm
Alcance Máximo	80m
Velocidade de Rotação	75Hz
Largura de Onda	$\lambda = 905$nm
Transferência de Dados	RS 232 ou RS 422
Alimentação	24V

ancoragem (STAHN; HEISERICH; STOPP, 2007), (e) classificação de objetos (MENDES; BENTO; NUNES, 2004), e (f) automação de processos (AUFRÈRE et al., 2003).

4.2 ROS

O *framework* ROS – *Robot Operating System* disponibiliza diversas bibliotecas e ferramentas para auxiliar no desenvolvimento de aplicações robóticas. Também fornece abstração de hardware, *device drivers*, visualizadores, transmissão de mensagens, gerenciamento de pacotes, utiliza protocolo TCP/IP, é licenciado sob uma licença livre (*open source*) e suporta várias linguagens de programação (C++, Phyton, Java, Octave e LISP) (JULIO, 2015). Esta licença livre torna o ROS um *framework* flexível possibilitando um desenvolvimento colaborativo de software. Dessa forma, um grupo de pesquisadores com foco em desenvolvimento de algoritmos de navegação autônoma pode criar o sistema e disponibilizar para que outros pesquisadores ao redor do mundo possam utilizar em suas pesquisas. Isso contribui muito para o desenvolvimento de algoritmos de mais alto nível, por exemplo, de tomadas de decisão, uma vez que o sistema de navegação esteja operando, os esforços podem ser voltados ao tema da pesquisa, reduzindo o tempo e aumentando a produtividade (QUIGLEY et al., 2009). É um sistema criado em 2007 pelo *Stanford Artificial Intelligency Laboratory* (SAIL) e com o decorrer dos anos tornou-se muito popular

no meio científico e acadêmico, estando presente em inúmeras aplicações e pesquisas ao redor do mundo. Sua estrutura de comunicação é do tipo ponto-a-ponto e possui alguns conceitos básicos que fornecem dados de diferentes modos:

- ***Nós:*** São os executáveis do projeto. Como uma das características do ROS é ser modular e possuir vários módulos pequenos, um projeto desenvolvido nesta plataforma poderá ter diversos nós. Por exemplo, um projeto de navegação autônoma terá um nó para leitura da odometria, um outro para cálculo da trajetória, um terceiro para cálculo do controle das rodas, etc.

- ***Master:*** Todos os nós se registram no ROS Master, assim um nó pode se comunicar com outro, trocando mensagens ou invocando serviços.

- ***Parameter Server:*** Valores dos parâmetros são armazenados em um local central para que todos os nós tenham acesso.

- ***Mensagens:*** Uma estrutura de dados simples que contém campos tipados usada para comunicação entre nós. São aceitos tipos primitivos de dados como *integer*, *double*, *string*, e *boolean*.

- ***Tópicos:*** As mensagens são transportadas através dos tópicos e, fazendo uma analogia, são como um barramento de dados, sendo que seu sistema utiliza um padrão de *publish - subscribe*. Seu funcionamento é o seguinte: um nó publica uma mensagem em um determinado tópico a uma certa taxa (frequência) de publicação, assim, estes dados ficam disponíveis para que qualquer nó subscreva no mesmo tópico e faça a leitura das mensagens. É importante salientar que caso haja mais de um nó publicando no mesmo tópico, cada um deverá fazê-lo em momentos distintos, para evitar uma sobreposição de dados. Por outro lado, um mesmo tópico poderá ter quantos *subscribers* forem necessários. Como o objetivo dos tópicos é desacoplar a produção de informação de seu consumo (JULIO, 2015), um nó publicador simplesmente publica suas mensagens e não precisa ter conhecimento dos nós inscritos no tópico, assim, o nó pode publicar mesmo que não exista um nó subscrito.

- ***Serviços:*** Os tópicos atualizam os dados a uma dada frequência, enquanto os serviços são do tipo requisição/resposta. Assim, um nó que oferece um serviço fica aguardando uma mensagem de requisição proveniente de um outro nó cliente. Este nó cliente envia a requisição e fica aguardando uma resposta do nó servidor.

- ***Bags:*** Os *bags* são arquivos que armazenam todas as mensagens dos tópicos de interesse e podem ser reproduzidos posteriormente. Por exemplo, ao executar a navegação de um robô em uma certa área, tem-se tópicos de odometria, laser, transformadas, trajetórias, imagem da câmera, etc, que são difíceis de coletar em simulações devido à modelação dos ruídos. Assim, o *bag* salva todos os dados destes tópicos e,

ao reproduzir, pode-se testar algoritmos com dados em ambientes reais e quantas vezes forem necessárias.

Tendo o conhecimento dos conceitos básicos do ROS, vários pacotes foram utilizados para testar os algoritmos de localização e serão listados na subseção seguinte.

4.2.1 ROS Packages

Um pacote é a principal unidade de organização de software em ROS, podendo conter executáveis (nós), bibliotecas, conjunto de dados, arquivos de configuração, parte do software de terceiros ou qualquer outra estrutura útil para a aplicação. Seu objetivo principal é fornecer um software funcional e útil de forma que seja facilmente reutilizado por terceiros (ROS, 2007). Utilizando destes benefícios disponibilizados pelo ROS, para implementar e testar os algoritmos de localização foram utilizados vários pacotes presentes no ROS wiki:

- ***p2os:*** Pacote para movimentar o robô Pioneer. O p2os é o driver antecessor do Aria e foi desenvolvido para ser utilizado em robôs da família Pioneer 2 e PeopleBot. Desenvolvedores fizeram algumas modificações e adaptaram o p2os para ser utilizado na plataforma ROS e em qualquer robô que use o p2os ou o firmware ARCOS. Seguindo o mesmo raciocínio, a adaptação do Aria é chamado de ROSAria e sua grande vantagem é a utilização das bibliotecas tanto do Aria quanto do ROS. Porém, várias transformadas e arquivos de configurações necessários para a navegação não estavam criados, enquanto o p2os já possuía tudo implementado, sendo este o escolhido para o trabalho.

- ***rviz:*** Ferramenta de visualização 3D do ROS que possibilita acompanhar o processo em tempo de execução através da leitura (*subscribe*) dos tópicos de interesse.

- ***sicktoolbox_wrapper:*** *Device driver* para lasers SICK LMS2xx. Possibilita ajustes do sensor através de vários parâmetros (resolução, ângulo de varredura, taxa de transmissão e vários outros). A leitura dos dados é feita pelo tópico *scan*.

- ***teleop_twist_keyboard:*** Nó utilizado para movimentar o robô através do ambiente com auxílio de um teclado (teleoperação). Possui um publicador no tópico *cmd_vel*, cuja função é variar a velocidade do robô para assim deslocá-lo.

- ***rosbag:*** *Rosbag* é o nó que salva os tópicos em um arquivo de extensão *.bag* e reproduz estes dados para futuras visualizações ou testes de algoritmos.

- ***gmapping:*** Ferramenta para criar mapas 2D de grade de ocupação através de dados do *laserscan* e da postura (odometria).

- ***map_server:*** Usado para carregar um mapa a fim de utilizar em alguma aplicação ou salvar o mapa gerado pelo *gmapping* em dois arquivos de extensões *.pgm* e *.yaml*, sendo este último um arquivo com dados do tipo resolução do mapa, origem, *threshold* para grids ocupados, *threshold* para grids livres e outros. Ao carregar um mapa, o pacote utiliza uma biblioteca chamada *SDL_Image* e dados de valores de *threshold* para quantificar os tons de cinza, assim o mapa de ocupação é representado por uma matriz com valores inteiros, sendo 0 para pixel livre, 100 para ocupado e -1 para não determinado.

4.3 Modelo de Cinemática

Como apresentado na seção 2.1.1, cada partícula representa um pseudo-robô com sua posição e orientação. Além disso, a cada deslocamento feito pelo robô, cada uma destas partículas também executarão a mesma movimentação. Portanto, é preciso ter um modelo cinemático bem definido, o qual será mostrado nesta seção.

O robô Pioneer possui acionamento diferencial, um mecanismo de direção simples constituído por duas rodas em um eixo comum controladas de forma independente e uma roda do tipo castor que serve como apoio. Seu modelo cinemático pode ser simplificado, conforme a equação 4.2, graças às transformadas utilizadas pelo ROS, fornecendo os valores de odometria em cada eixo (x e y) em metros e desvio angular (θ) em radianos. A figura 4.3.1 mostra uma translação seguida de rotação.

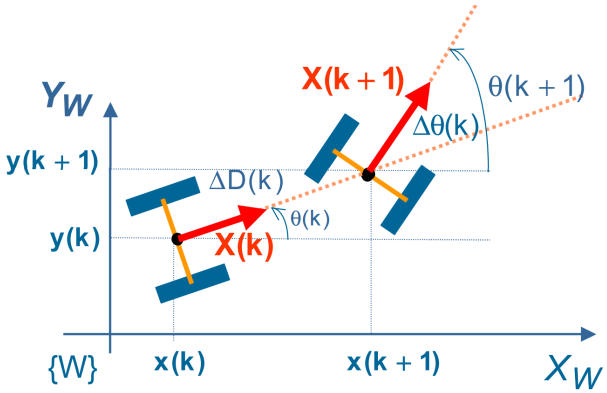


Figura 4.3.1 – Translação seguida de rotação de um robô diferencial. Imagem extraída de Ribeiro (1999).

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \begin{bmatrix} x_t + \Delta D_t.cos\theta_t \\ y_t + \Delta D_t.sen\theta_t \\ \theta_t + \Delta\theta_t \end{bmatrix} + \begin{bmatrix} \sigma_T \\ \sigma_T \\ \sigma_R \end{bmatrix} \tag{4.2}$$

onde θ é obtido pelos dados do giroscópio, o deslocamento, ΔD_t, no instante t é calculado pela equação 4.3. σ_T e σ_R são os desvios-padrão de translação e rotação, respectivamente.

$$\Delta D_t = \sqrt{(x_{t-1} - x_t)^2 + (y_{t-1} - y_t)^2} \tag{4.3}$$

Portanto, para as n partículas criadas, cada uma delas terá o mesmo deslocamento conforme a equação 4.2. É perceptível a economia de tempo e esforço na modelagem da cinemática, uma vez que foi utilizado a plataforma ROS e suas características de código livre e modularidade, possibilitando o reuso dos códigos. Reforçando, assim, os benefícios e vantagens deste *framework*.

4.4 Modelo de Observação

O modelo de observação foi criado com base no modelo de cinemática, de tal forma que a figura 4.4.1 ilustra um exemplo simplificado para apenas uma partícula.

Considerando que dos 180 feixes de luz que o laserscan fornece, sejam usados somente 5 feixes espalhados igualmente entre 0°e 180°, assim cada pseudo-robô também terá um laser com 5 feixes. Para representar a varredura de cada feixe a equação 4.2 foi adaptada para encontrar o obstáculo mais próximo, resultando na equação 4.4.

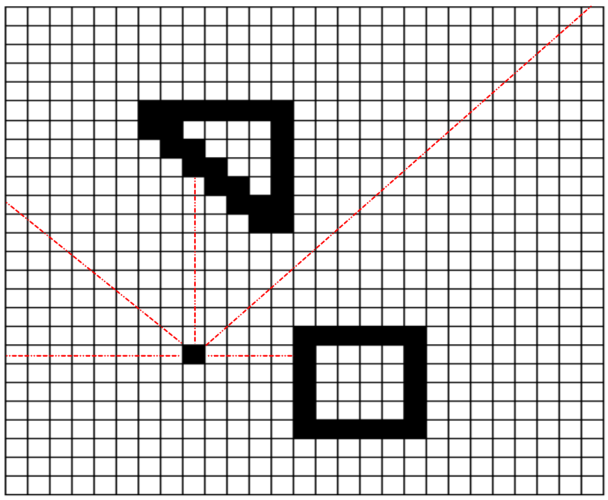


Figura 4.4.1 – Modelo de observação para uma partícula com cinco feixes do laser.

$$\begin{bmatrix} x_{obstaculo} \\ y_{obstaculo} \end{bmatrix} = \begin{bmatrix} x_t + D_{passo}.cos\theta_f \\ y_t + D_{passo}.sen\theta_f \end{bmatrix} + \sigma_{Obs} \tag{4.4}$$

sendo x_t e y_t a posição da partícula, D_{passo} é um valor inicializado com zero e a cada busca é incrementado uma constante (passo), o θ_f é a soma da orientação do robô em relação ao mapa θ_t com o ângulo do feixe em relação ao robô e σ_{Obs} é o desvio-padrão da observação. O D_{passo} é incrementado até que $x_{obstaculo}$ e $y_{obstaculo}$ coincidam com uma

célula ocupada, indicando que foi encontrado um obstáculo, ou até o alcance máximo do sensor.

O algoritmo 7 mostra como é realizada a rotina de busca de obstáculos para uma partícula.

Algoritmo 7: Rotina do modelo de observação

Entrada: *número de feixes* f*, posição da partícula* x_t *e* y_t*, resolução do mapa* res *e* σ_{obs}

```
1  início
2      para l = 0 até f faça
3          para p = passo até alcance máximo faça
4              x_obstaculo = x[i] + p.cos(l)
5              y_obstaculo = y[i] + p.sen(l)
6              x_obstaculo = x_obstaculo/res //Converte valores métricos em pixels
7              y_obstaculo = y_obstaculo/res
8              se (x_obstaculo, y_obstaculo) é um obstáculo então
9                  Distância_obstaculo = p + σ_obs
10                 p = alcance máximo
11             fim
12         fim
13     fim
14 fim
```

Saída: Distância do obstáculo para cada feixe

Este algoritmo retorna a distância do obstáculo para cada feixe do pseudo-robô, assim, conforme apresentado na seção 2.1.1, o peso de cada partícula é calculado. Nota-se que é necessário uma análise da quantidade de partículas e feixes a serem utilizados devido ao alto custo computacional deste modelo de observação (complexidade $O(n^3)$). No capítulo 5 será abordado essa análise e várias outras a fim de comprovar as melhorias oferecidas pelo algoritmo de localização proposto.

5 Experimentos e Resultados

Os experimentos para analisar os algoritmos foram realizados no Laboratório de Robótica da Unifei – LRO e com os materiais presentes neste, tais como: robô Pioneer 3DX, *laserscan* SICK LMS 291, figura 4.1.1, e um labirinto modelável, figura 5.0.1, foi possível testar de forma prática utilizando dados reais ao invés de um sistema simulado, cujos dados e ruídos são aproximados.

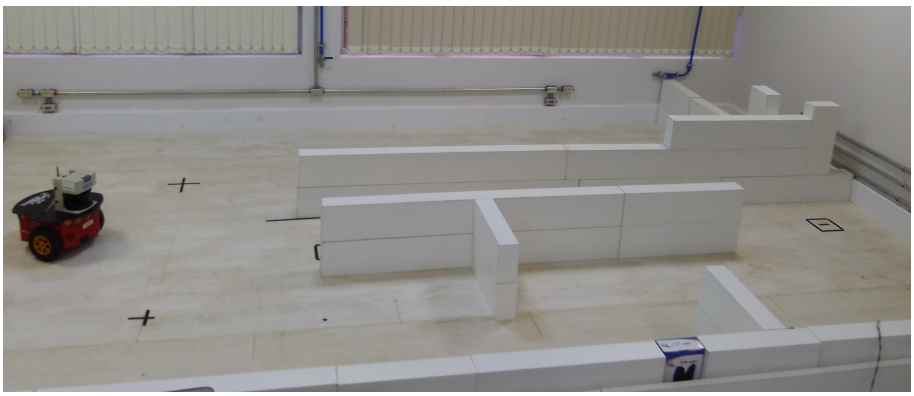


Figura 5.0.1 – Labirinto modelável de blocos de madeira em MDF localizado no Laboratório de Robótica da UNIFEI.

Foram implementados, utilizando o *framework* ROS, os três algoritmos do capítulo 3 baseados no filtro de partículas simples apresentado na seção 2.1.1, com o intuito de fazer um estudo comparativo mais preciso, uma vez que foram usadas as mesmas técnicas de estruturas de dados em todos os algoritmos.

Um algoritmo de localização para ser considerado robusto e eficiente deverá se localizar com uma certa precisão, ter baixo tempo de convergência e custo computacional, e solucionar os três principais problemas da localização: (a) localização local, (b) global e (c) sequestro do robô. Desta forma, serão feitas análises comparativas entre as três abordagens para cada um dos problemas separadamente e, consequentemente, os requisitos de precisão e custo computacional serão tratados dentro de cada análise.

Os testes iniciais foram feitos através de um ambiente montado dentro do laboratório de dimensões 7m x 3,7m, figura 5.0.1, e um mapa de resolução 0,15m, o qual foi criado através dos pacotes *gmapping* e *map_server*, resultando na figura 5.0.2. Os ruídos utilizados nos modelos de observação e de cinemática são distribuições Gaussianas de média zero e desvios-padrão $\sigma_{obs} = 0,031$, $\sigma_T = 0,004$ e $\sigma_R = 0,003$, sendo σ_{obs} obtido através de várias amostragens do *laserscan*, sendo os outros dois calculados a partir da equação 5.1.

$$ruido = \frac{\sigma}{alcance_{maximo}} * 100\% \tag{5.1}$$

sendo que o ruído foi usado um valor de 10% e o alcance máximo foi dado pela equação 5.2.

$$alcance_{maximo} = velocidade_{maxima}/frequencia \tag{5.2}$$

onde a velocidade máxima para translação é de 400mm/s e para rotação 300mm/s. A frequência é dada pela frequência de publicação do tópico de odometria, 10Hz.

Figura 5.0.2 – Mapa do labirinto criado pelo *gmapping*.

Entretanto, na seção 2.1.1 foi mencionado que é importante adicionar ruídos artificiais aos ruídos da movimentação e da percepção, assim, valores de $\sigma_T = 0,020$ e $\sigma_R = 0,015$ apresentaram experimentalmente uma melhor localização local.

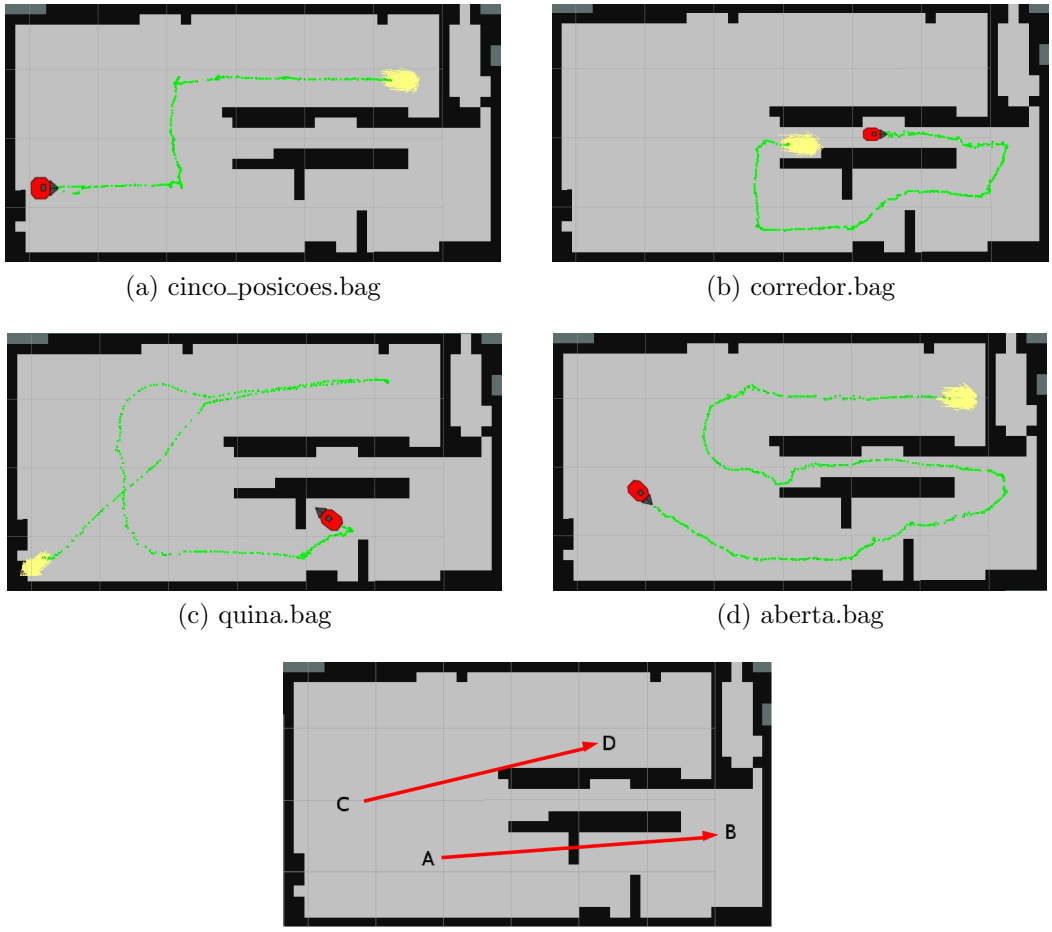
O tempo para gerar as matrizes no processo offline (pré-armazenagem) está diretamente relacionado às dimensões do mapa e à sua resolução. A tabela 5.0.1 lista a média do tempo para este processo para diferentes dimensões e resoluções.

Tabela 5.0.1 – Tempo para executar o processo offline.

Dimensões	Resolução [m]	Tempo do Processo Offline [s]
7m x 3,7m	0,05	27,926
	0,10	3,864
	0,15	1,182
	0,20	0,410
	0,25	0,176
7m x 9m	0,15	2,881

Após o mapeamento da área de testes, o robô foi colocado em 5 posições iniciais distintas e suas trajetórias foram criadas de forma teleoperada, figura 5.0.3, sendo que para cada situação diferente, foi criado um arquivo *bag* para salvar os dados dos tópicos de odometria, laser, transformadas e tempo. A função do *bag* é salvar os dados de interesse e reproduzí-los a cada teste, desta forma, todos os experimentos terão os mesmo conjuntos de dados e é possível fazer análises comparativas mais precisas. Os *bags* criados foram:

- *cinco_posicoes.bag*, figura 5.0.3a, criado para comparar a precisão do rastreamento de posição entre os algoritmos. Dado um ponto inicial bem definido, o robô é movimentado através de cinco pontos conhecidos.



(a) cinco_posicoes.bag (b) corredor.bag

(c) quina.bag (d) aberta.bag

(e) sequestro.bag

Figura 5.0.3 – Arquivos *.bag* criados para testes e análises dos algoritmos. A figura em vermelho é o robô representando a postura inicial, em verde a trajetória e amarelo são as partículas convergidas na postura em que o robô se localiza (postura final).

- *corredor.bag*, figura 5.0.3b, criado com o robô no corredor do labirinto como sua posição inicial, mas neste caso sua postura inicial não é informada, portanto utilizado para análises de localização global.
- *quina.bag*, figura 5.0.3c, inicializa com o robô apontando para uma quina qualquer e a localização global estima sua posição.
- *aberta.bag*, figura 5.0.3d, inicializa o robô em uma área mais aberta e com os obstáculos mais distantes. Este *bag* também é utilizado para a análise da localização global.
- *sequestro.bag*, figura 5.0.3e, um *bag* no qual ocorre dois sequestros durante o processo. O primeiro do ponto A para o ponto B e o segundo de C para D.

Uma vez que o máximo ângulo de varredura do *laserscan* é de 180°, existem pontos cegos no robô, portanto a matriz G_E também precisa armazenar as orientações do robô.

Nos testes foram utilizados 18 orientações diferentes, com passo de 20°, implicando em 18 valores de energia e orientação para cada posição de área SER.

O grafo do processo de localização é apresentado na figura 5.0.4. O nó *play* é o *rosbag*, o qual fornece dados de transformada, odometria e laser, através dos tópicos *tf, odom e scan*, respectivamente, salvos a partir do robô Pioneer. *Map_server* fornece metadados do mapa, coordenadas (x,y) de células ocupadas e livres. Enquanto o nó do filtro de partículas faz a leitura de todos estes dados para o processo de localização e produz como resposta um conjunto de partículas. Todos os nós foram executados no *master*, um computador não embarcado, citado em 4.1.2.

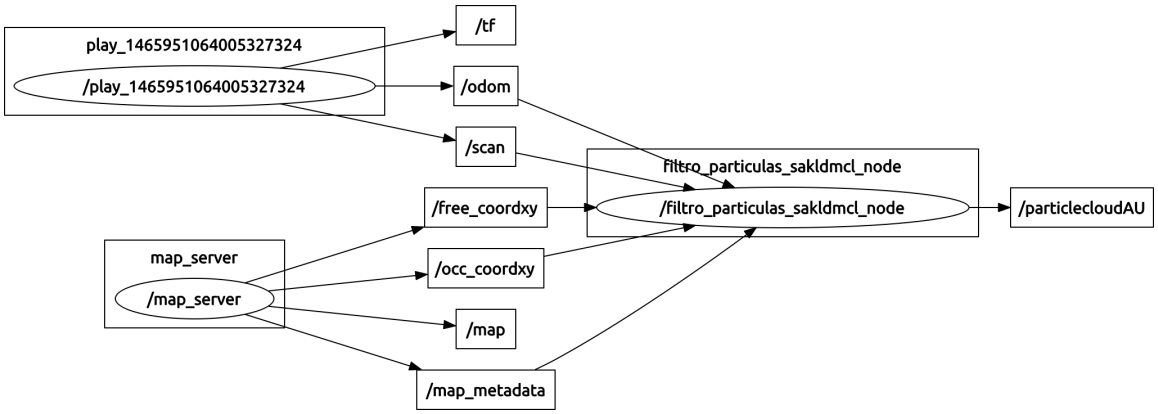


Figura 5.0.4 – Grafo do processo de localização produzido pelo *rosgraph*.

Primeiramente, é importante fazer um estudo dos efeitos causados pela variação dos valores de parâmetros e após defini-los é possível fazer uma melhor análise comparativa para cada tipo de problema de localização.

5.1 Análise de Parâmetros

Filtros de partículas baseados em KLD e SAMCL possuem características e parâmetros distintos. Como o algoritmo desenvolvido utiliza o conceito de ambos, os parâmetros são herdados e agem de forma complementar, ou seja, para setar um parâmetro originário do KLD, um outro parâmetro do SAMCL deverá ser levado em consideração. A seguir a tabela 5.1.1 que lista os parâmetros de cada algoritmo:

Tabela 5.1.1 – Parâmetros referentes a cada filtro de partículas.

KLD	SAMCL
Erro ϵ	SER_threshold
Quantil de uma distribuição normal $z_{(1-\delta)}$	weight_threshold
bins	alpha sample set α_s

Através da análise feita em Fox (2003), o autor chegou à conclusão de que variando a probabilidade $(1-\delta)$ de 90% à 99,9%, o número de partículas necessárias em relação ao

número de bins ocupados apresentou alterações menos significativas em relação à variação do erro ϵ (de 0,015 até 0,2). Portanto, $(1 - \delta)$ foi fixado em 99% e utilizando a tabela de distribuição normal, ilustrado na figura 5.1.1, o valor de $z_{(1-\delta)}$ encontrado foi de 2,33, enquanto o ϵ foi ajustado com o auxílio do gráfico da figura 5.1.2 para obter uma melhor performance.

z	.00	.01	.02	.03	.04	.05	.06	.07	.08	.09
.0	.5000	.5040	.5080	.5120	.5160	.5199	.5239	.5279	.5319	.5359
.1	.5398	.5438	.5478	.5517	.5557	.5596	.5636	.5675	.5714	.5753
.2	.5793	.5832	.5871	.5910	.5948	.5987	.6026	.6064	.6103	.6141
.3	.6179	.6217	.6255	.6293	.6331	.6368	.6406	.6443	.6480	.6517
.4	.6554	.6591	.6628	.6664	.6700	.6736	.6772	.6808	.6844	.6879
.5	.6915	.6950	.6985	.7019	.7054	.7088	.7123	.7157	.7190	.7224
.6	.7257	.7291	.7324	.7357	.7389	.7422	.7454	.7486	.7517	.7549
.7	.7580	.7611	.7642	.7673	.7704	.7734	.7764	.7794	.7823	.7852
.8	.7881	.7910	.7939	.7967	.7995	.8023	.8051	.8078	.8106	.8133
.9	.8159	.8186	.8212	.8238	.8264	.8289	.8315	.8340	.8365	.8389
1.0	.8413	.8438	.8461	.8485	.8508	.8531	.8554	.8577	.8599	.8621
1.1	.8643	.8665	.8686	.8708	.8729	.8749	.8770	.8790	.8810	.8830
1.2	.8849	.8869	.8888	.8907	.8925	.8944	.8962	.8980	.8997	.9015
1.3	.9032	.9049	.9066	.9082	.9099	.9115	.9131	.9147	.9162	.9177
1.4	.9192	.9207	.9222	.9236	.9251	.9265	.9279	.9292	.9306	.9319
1.5	.9332	.9345	.9357	.9370	.9382	.9394	.9406	.9418	.9429	.9441
1.6	.9452	.9463	.9474	.9484	.9495	.9505	.9515	.9525	.9535	.9545
1.7	.9554	.9564	.9573	.9582	.9591	.9599	.9608	.9616	.9625	.9633
1.8	.9641	.9649	.9656	.9664	.9671	.9678	.9686	.9693	.9699	.9706
1.9	.9713	.9719	.9726	.9732	.9738	.9744	.9750	.9756	.9761	.9767
2.0	.9772	.9778	.9783	.9788	.9793	.9798	.9803	.9808	.9812	.9817
2.1	.9821	.9826	.9830	.9834	.9838	.9842	.9846	.9850	.9854	.9857
2.2	.9861	.9864	.9868	.9871	.9875	.9878	.9881	.9884	.9887	.9890
2.3	.9893	.9896	.9898	.9901	.9904	.9906	.9909	.9911	.9913	.9916

Figura 5.1.1 – Tabela de distribuição normal z. Para um valor de 99% de probabilidade acumulada, o valor de $z_{(99\%)}$ é de 2,33. Imagem extraída de (TEAM-MATH, 2011).

Fox (2003) mostrou experimentalmente que para uma bom resultado de rastreamento de posição é necessário em média 186 partículas, e para uma boa localização global a partir de 3000. Executando o SA-KLD para o mapa citado e o parâmetro *bins* setado em 2, o valor de 0,020 para o ϵ mostrou-se ser mais vantajoso, pois a taxa de acertos da localização global foi maior para todas as trajetórias salvas nos *bags* e utiliza em média 3000 partículas toda vez que são criadas (início do algoritmo ou sequestro). Para o rastreamento de posição são usadas aproximadamente 250.

Enquanto ϵ e $z_{(1-\delta)}$ são parâmetros relacionados à quantidade de partículas, o *bin* é usado para discretizar o mapa. Um mapa muito discretizado, ou seja, com um valor de *bin* pequeno, terá células com dimensões pequenas e consequentemente a contagem dos *bins*

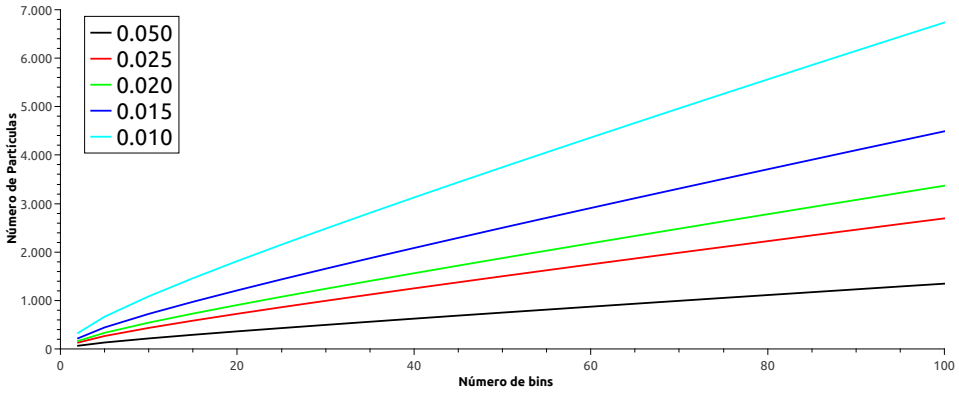


Figura 5.1.2 – Gráfico do número de partículas em relação ao número de bins ocupados.

ocupados será maior e um número maior de partículas serão necessárias. A fim de não comprometer no custo computacional devido ao grande número de partículas, conforme o aumento das dimensões do mapa, é interessante também aumentar o valor do parâmetro.

Em relação aos parâmetros do algoritmo SAMCL, o *SER_threshold* ajusta os limites inferior e superior da energia, assim, um alto valor irá gerar um maior número de áreas SER, resultando em um maior número de *bins* ocupados e partículas. Por outro lado, usar um valor muito pequeno poderá resultar em poucas áreas e, possivelmente, gerar um erro na localização. Como o cálculo da energia depende da quantidade de feixes do laser utilizados, para um valor diferente de feixes o *threshold* deverá ser configurado novamente. Assim, dos 180 feixes que o laser disponibiliza foram utilizados apenas 30 e para o SER_threshold foi fixado, de forma experimental, um valor de 0,02.

Além do SER_threshold, o *weight_threshold* também é dependente do número de feixes do laser e seu valor tem a função de identificar um possível sequestro. Valores grandes deixam o sistema muito sensível e a função de sequestro é ativada mesmo o robô estando na posição correta. Entretanto, para um *threshold* muito pequeno, o sistema irá demorar muito para identificar o erro de localização e o robô poderá neste tempo causar uma catástrofe. Após vários testes, para 30 feixes do laser um bom weight_threshold encontrado experimentalmente foi de 0,045.

Como a função de sequestro depende apenas do peso máximo, há momentos em que ocorrem os falsos sequestros devido a erros de leituras do laser ou devido aos erros da odometria gerados pelo deslize das rodas. Portanto, para que a postura média do robô não sofra grandes alterações é interessante utilizar um α_s grande. Assim, a maioria das partículas irá continuar ao redor da postura atual, enquanto uma pequena parcela é espalhada pelas áreas SER. Dessa forma, a posição média irá apresentar uma mínima alteração. Para o experimento foi utilizado um α_s de 0,7, ou seja, 70% das partículas serão do tipo amostras locais e permanecerão próximas da postura atual do robô.

Nota-se que para cada ambiente com mapa de dimensões ou resoluções diferentes, os parâmetros devem ser ajustados e como não se tem um modelo matemático para definir

os melhores valores, eles são determinados experimentalmente.

5.2 Localização Local

Considerando que o robô conheça sua postura inicial, foi usado um *bag* contendo seu deslocamento através de cinco pontos conhecidos e bem definidos do mapa. O conjunto de partículas foi variado de 1000, 2500, 5000 e 7500, foram coletados dados de 50 amostragens e calculados suas médias e desvios-padrão em metros. A figura 5.2.1 mostra o mapa com as cinco posições, sendo a ilustração do robô representando a postura inicial, em verde a trajetória percorrida por ele e em amarelo o conjunto de partículas na posição final.

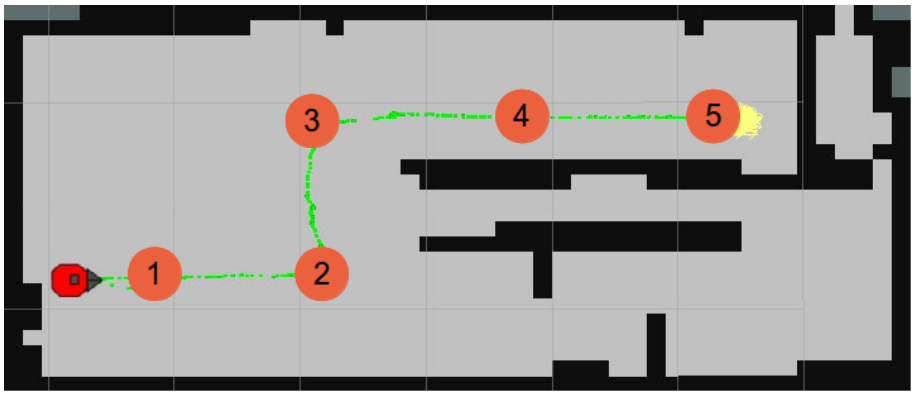
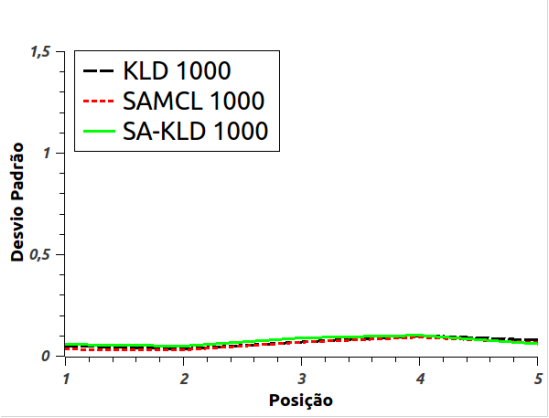


Figura 5.2.1 – Mapa do labirinto com as marcações das cinco posições usadas para testes.

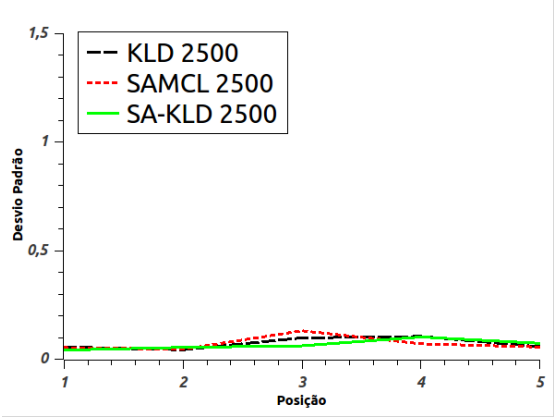
Na figura 5.2.2, o desvio padrão referente a cada posição é calculado a partir das distâncias Euclidianas das 50 amostras para cada respectivo ponto conhecido.

O SAMCL apresenta ótimos resultados para pequenas quantidades de partículas 5.2.2a e 5.2.2b, tanto em termos de desempenho, quanto de precisão. Com o aumento do número de partículas, seu custo computacional se eleva e o rastreamento de posição apresenta um atraso, devido às perdas de pacotes de dados ocasionadas pelo alto processamento para os vários processos de *resampling* de um conjunto grande de partículas. Em 5.2.2c, o SAMCL já apresenta um erro significativo a partir da posição 3 devido ao elevado custo computacional e este erro se acumula com o decorrer do movimento. Para 7500 partículas 5.2.2d o erro em algumas amostras foi grande o suficiente para ativar a função de sequestro (posição 3) e ajustar a localização do robô. Por outro lado, o KLD mostrou uma precisão muito boa em todas as situações, pois independentemente do número de partículas inicial, após algumas iterações esse número foi reduzido para, em média, 250 amostras. Isso deixa o processamento mais leve e evita a perda de dados dos sensores. Essa característica do KLD é herdada pelo SA-KLD, o qual também apresentou uma excelente precisão em todos os casos, como pode-se observar em 5.2.2e.

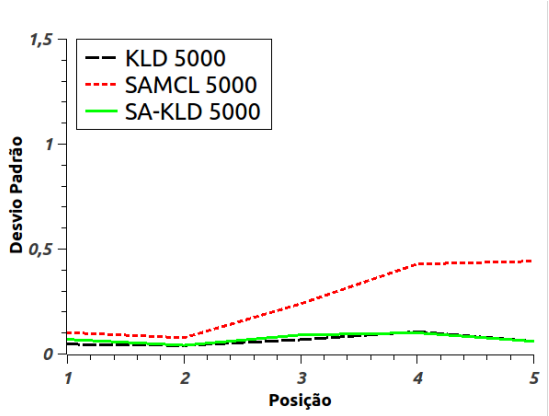
Na tabela 5.2.1 é listado o tempo de cada iteração em função do número de partículas, reforçando o fato de que utilizando uma grande quantidade de partículas resulta em um custo computacional mais elevado.



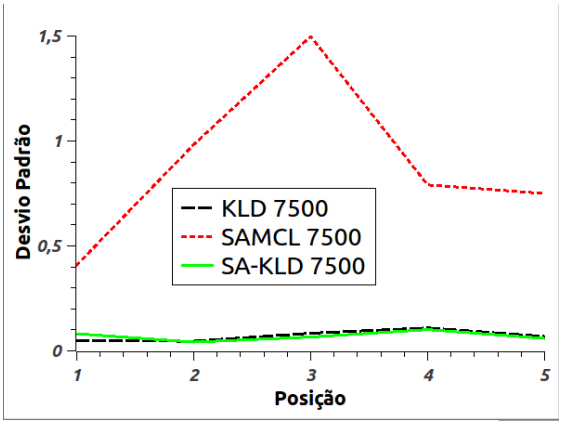
(a) 1000 Partículas



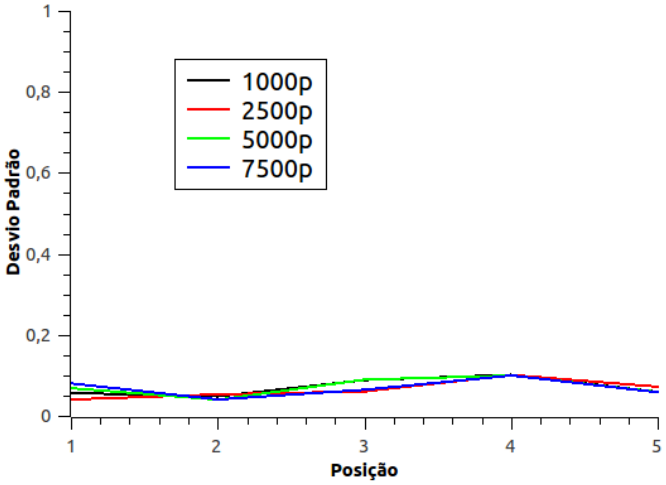
(b) 2500 Partículas



(c) 5000 Partículas



(d) 7500 Partículas



(e) SA-KLD para 1000, 2500, 5000 e 7500 partículas

Figura 5.2.2 – Resultados comparativos para o problema de rastreamento de posição.

Tabela 5.2.1 – Tempo de cada iteração em função do número de partículas.

Número de Partículas	Tempo de Cada Iteração [s]
7500	1,359
6740	1,177
5567	0,931
3543	0,629
2212	0,377
1043	0,206
694	0,102
504	0,066
422	0,050
379	0,039
285	0,028

5.3 Localização Global

Diferentemente do rastreamento de posição, para analisar a localização global, a postura inicial é totalmente desconhecida, portanto, o primeiro passo para localizar o robô é distribuir as partículas sobre as áreas livres do mapa. A ideia principal do trabalho está em criar essas partículas dentro de áreas com maiores probabilidades (áreas SER) ao invés de espalhar pelo mapa todo. Para os testes foram utilizados três situações diferentes: o corredor, a quina, e a área aberta, conforme ilustrado na figura 5.0.3. Para cada caso foram computados diferentes quantidades de áreas SER e consequentemente, número de partículas necessárias. Foram analisados o tempo para o filtro de partículas definir uma postura, os erros de localização e a quantidade de ativações da função de sequestro.

A partir de cada caso, foram coletados os dados de número inicial de partículas, quantidade de áreas SER e *bins* ocupados, conforme tabela 5.3.1.

Tabela 5.3.1 – Número inicial de partículas, quantidade de áreas SER e bins ocupados das três situações iniciais.

	Número de Partículas	Áreas SER	bins ocupados
Corredor	3281	1690	97
Quina	1274	206	31
Área Aberta	2333	911	65

Nota-se que, variando a quantidade de áreas SER, os *bins* ocupados acompanham de forma diretamente proporcional, assim como o número de partículas. Houve uma diferença significativa na quantidade de áreas SER do corredor em relação à quina devido à maior quantidade de pontos com alta verossimilhança e como a quina possui dados de laser muito pequenos, o cálculo da energia restringiu bastante essas áreas de SER. Visualmente, o teste feito no corredor apresentou uma grande densidade de pontos dentro deste e seus arredores, enquanto o outro em apenas algumas poucas quinas. Essas diferenças entre os

dois casos extremos refletiu no tempo de localização. Enquanto que, para o SA-KLD, a quina levou 5,89s, o corredor precisou de 8,06s.

Uma comparação entre os três algoritmos pode ser feita através da relação número de partículas pelo tempo, mostrado na figura 5.3.1. Nota-se que em todos os casos o SA-KLD obteve um menor tempo de localização se comparado ao KLD e nos casos 5.3.1a e 5.3.1c foi necessário um menor número de partículas para o processo. Uma vez que em 5.3.1b o SA-KLD precisou do número máximo de partículas, isso aumentou o tempo para convergir, resultando em 8,06s, enquanto o SAMCL levou apenas 5,76s. Isso ocorreu devido ao tempo necessário para aumentar o número de partículas no algoritmo híbrido. A tabela 5.3.2 lista o tempo de localização para cada algoritmo nas situações citadas.

Tabela 5.3.2 – Tempo de localização para cada algoritmo em diferentes casos.

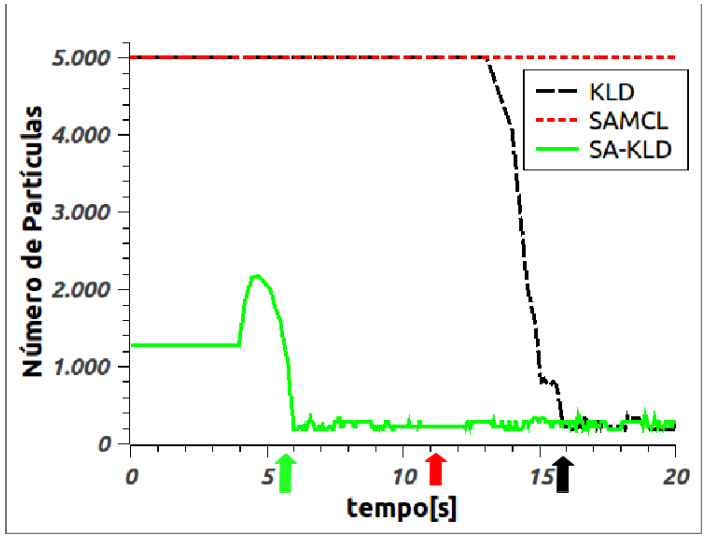
	KLD	SAMCL	SA-KLD
Quina	15,71s	11,34s	5,89s
Corredor	10,03s	5,76s	8,06s
Área Aberta	13,46s	7,88s	6,02s

Apesar do tempo do SAMCL ser menor no caso 5.3.1b, o algoritmo apresenta muito erro de rastreamento de posição e dependendo da trajetória e velocidade do robô esse erro se acumula e torna-se muito grande. Já o KLD teve um alto índice de erros de localização global, em média 8 (mais de 10%); isso se dá devido à forma que o algoritmo inicializa o processo de localização (espalhando as partículas pelo mapa todo). Assim, o algoritmo pode não convergir corretamente devido à baixa densidade de partículas em uma região de alta probabilidade. Em contrapartida, o SA-KLD teve bons resultados para o problema de localização global, apresentando nenhum erro, além de ser melhor no rastreamento de posição e levar menos tempo para convergir.

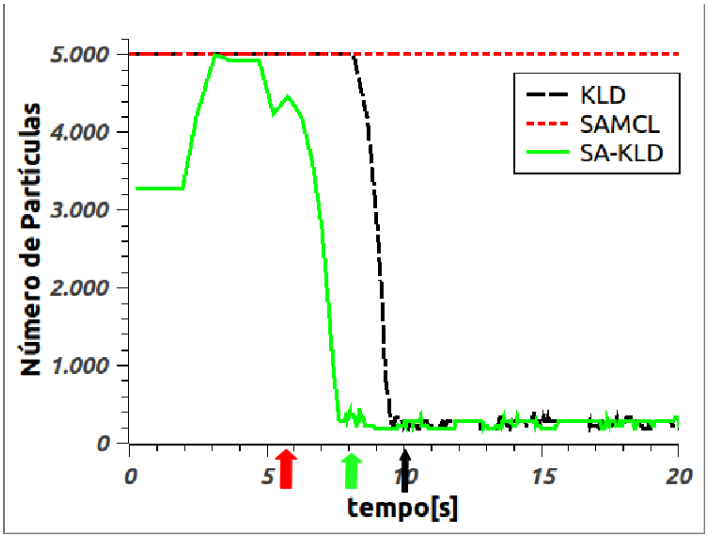
5.4 Sequestro do Robô

Uma situação bastante difícil que o robô precisa reconhecer é o sequestro. Como existem os casos real e falso, torna-se uma tarefa complicada afirmar que isso ocorreu. Tanto o SAMCL quanto o SA-KLD analisam o peso máximo das partículas em cada iteração e, se este valor for menor do que um *threshold*, a função de sequestro é ativada.

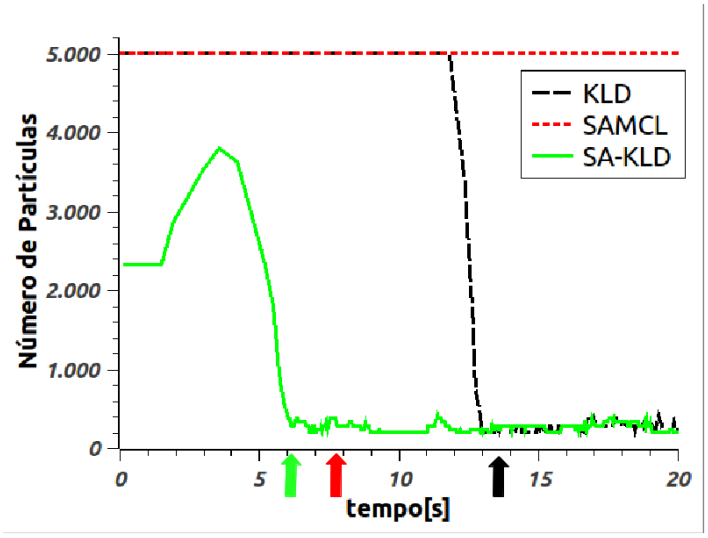
A seguir a figura 5.4.2 mostra o gráfico número de amostras pelo tempo de um teste em que houveram dois sequestros sendo um no tempo 32s (do ponto A ao ponto B, figura 5.4.1) e outro no 74s (do ponto C ao ponto D). No primeiro rapto, devido ao giroscópio, ambos os algoritmos identificam uma variação na orientação do robô e espalham as partículas, porém utilizando informações erradas do laser, assim aos 35s é identificado o erro de localização, a função é ativada novamente e por fim, a postura correta é definida. No segundo caso



(a) Quina



(b) Corredor



(c) Área aberta

Figura 5.3.1 – Gráfico comparativo do número de partículas pelo tempo da localização global, sendo que as setas representam os tempos de localização de cada algoritmo.

não houve esse erro porque, coincidentemente durante o sequestro, as partículas foram criadas próximas da postura correta.

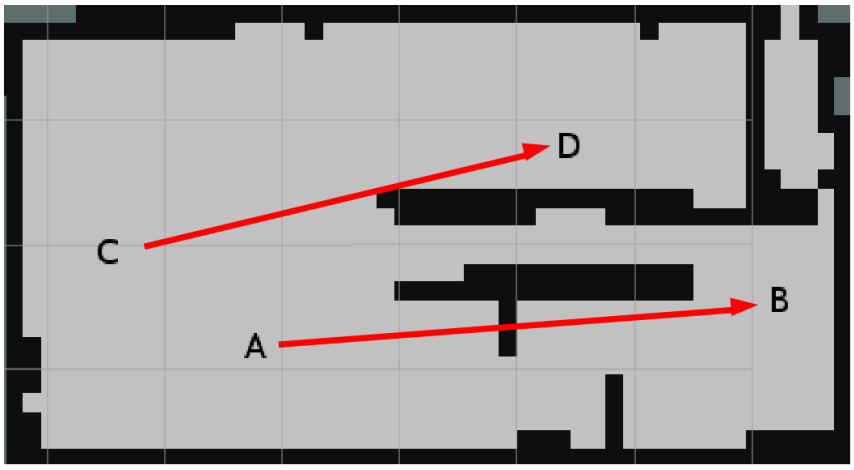


Figura 5.4.1 – Posições onde ocorreram os sequestros e destinos.

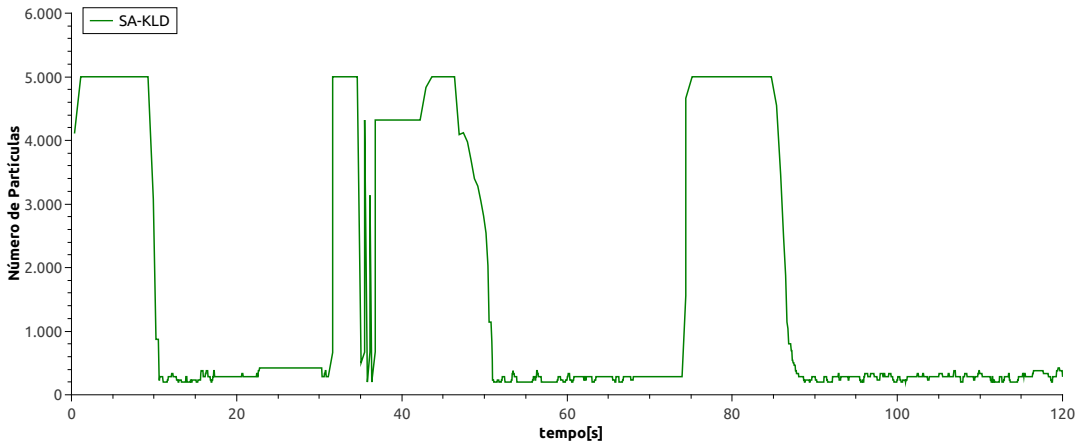


Figura 5.4.2 – Gráfico do número de partículas em função do tempo quando o sequestro é identificado.

Durante o teste, o tempo de relocalização após o rapto do robô, o número de partículas utilizadas e a postura ao final do deslocamento foram armazenados e comparados, veja a tabela 5.4.1 a seguir.

Assim, a partir dos dados armazenados nota-se que:

- O tempo de relocalização do SAMCL foi maior, o qual está diretamente relacionado ao número de partículas utilizadas, uma vez que é necessário um maior número de reamostragens (*resampling*). Isto implicou em dois erros de localização, pois ao final do deslocamento, o algoritmo não foi capaz de relocalizar o robô após o segundo sequestro.

- O parâmetro *bin*, herdado da abordagem KLD e aplicado no SA-KLD, ajustou o número de partículas utilizadas durante a execução do algoritmo e como resultado,

Tabela 5.4.1 – Tabela comparativa entre SAMCL e SA-KLD para o processo de relocalização.

	SAMCL	SA-KLD	
Parâmetro *bin*	–	2	3
Tempo para Relocalização [s]	10,86	8,40	4,87
Número Máximo de Partículas	5000	5000	2899
Número Mínimo de Partículas	5000	1751	1043
Número Médio de Partículas	5000	3565	2010
Quantidade de Erros de Localização	2	0	0

houve uma variação no tempo de relocalização. Ambos os testes tiveram valores nulos de erro de localização na postura final e menores tempos em relação ao SAMCL.

- Apesar de que, com o aumento do valor de *bin*, houve uma redução considerável no tempo para convergir na localização correta, a função de sequestro foi ativada muito mais vezes, devido à pequena quantidade de partículas. Assim, em alguns casos, o rastreamento de posição foi afetado devido à alta taxa de redistribuição de partículas. Então, pode-se concluir que o processo de relocalização para *bin* igual a 3 é menos preciso em relação à um valor menor, no caso, igual a 2.

A desvantagem da função de sequestro está no fato do algoritmo levar em conta apenas o peso máximo, assim em dados momentos em que o robô está na postura correta, por motivos de erros de leituras do laser, a função é ativada. Uma solução para minimizar estes efeitos, foi trabalhar com um α_s de 0,7 para que a postura não varie bruscamente durante a navegação do robô.

O código fonte do SA-KLD pode ser encontrado em GitHub - SA-KLD e o vídeo comparativo dos três algoritmos é apresentado na figura 5.4.3 ou através do link *A Hybrid Self-Adaptive Particle Filter Through KLD-Sampling and SAMCL.*

5.5 Testes em Diferentes Ambientes

Outros testes foram feitos em diferentes ambientes e situações a fim de visualizar as diferenças e semelhanças entre os três algoritmos.

Inicialmente, foram criados dois mapas distintos: (a) uma expansão do mapa anterior, figura 5.5.1a, e (b) um ambiente simétrico com poucos marcos para corrigir a localização, figura 5.5.1b.

5.5.1 Expansão do Labirinto

A partir da postura inicial representada pela seta em vermelho na figura 5.5.1a, o robô executa a trajetória marcada em verde e o tempo de localização é armazenado. Durante

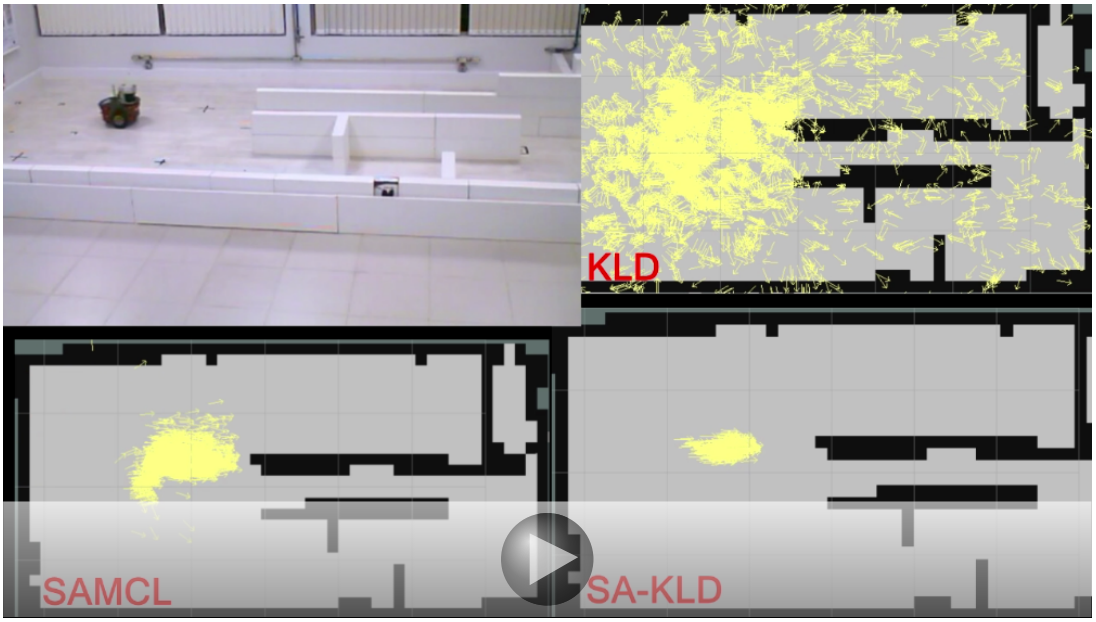


Figura 5.4.3 – Vídeo comparativo dos três algoritmos em ambiente real. Durante o experimento, o robô desloca-se pelo ambiente a fim de encontrar sua localização em relação ao mapa. Com o decorrer do teste, o robô é transportado para outra posição com o objetivo de ativar a função de sequestro e analisar a capacidade dos algoritmos de relocalizar o robô.

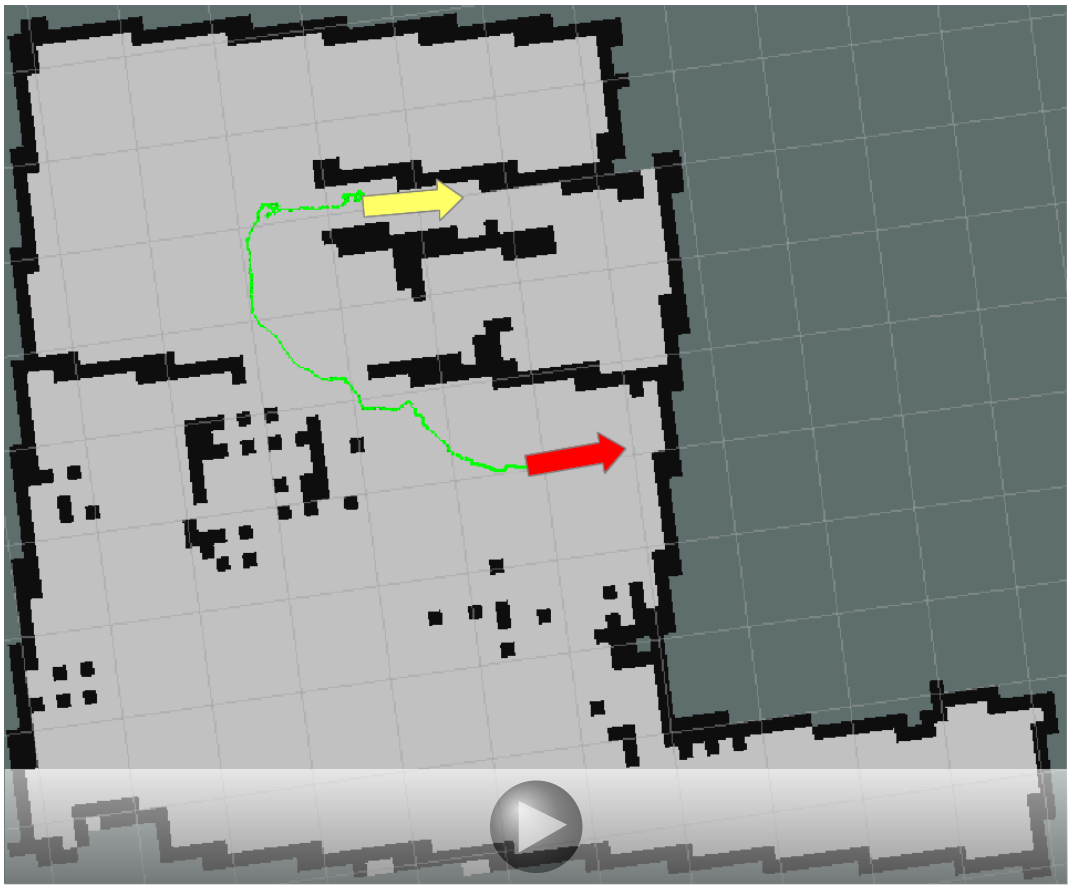
toda a movimentação são contados o número de ativações da função de sequestro e ao fim do deslocamento (seta amarela) é armazenado a postura atual do robô. Assim, é possível analisar os efeitos causados pela quantidade de partículas sobre a localização global e local. O número de partículas do SAMCL foi variado ao passo de 1000 a partir de 500 partículas. A seguir a tabela 5.5.1:

Tabela 5.5.1 – Comparação entre SAMCL, SA-KLD e KLD para um mapa com dimensões maiores e 20 execuções para cada caso.

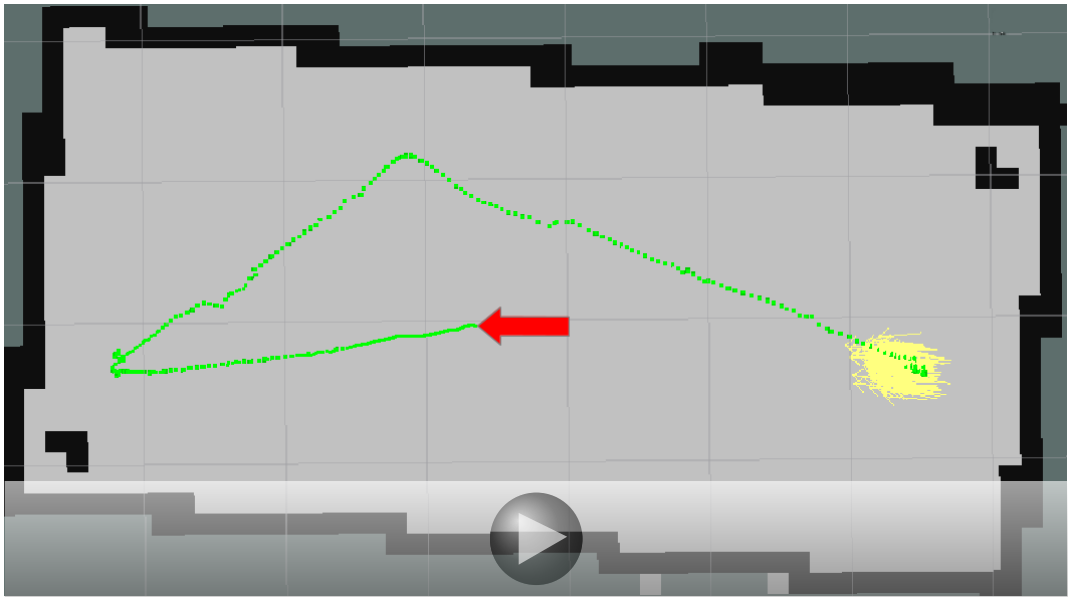
	Número Máximo de Partículas	Tempo de Localização [s]	Média de Sequestros	Número de Erros de Localização	Valor do Parâmetro bin
SAMCL	500	6,02	14,8	1	—
	1500	7,49	3,1	1	—
	2500	9,57	2,1	0	—
	3500	13,35	1,0	0	—
	4500	14,38	0,9	1	—
	5500	18,29	0,6	3	—
SA-KLD	2650	7,98	2,9	0	5
	5500	15,11	1,3	1	3
KLD	5500	16,56	—	3	5

A partir da tabela pode-se fazer algumas considerações:

- O tempo de localização representa o tempo da primeira vez que o algoritmo conver-



(a) Mapa da expansão do labirinto.



(b) Mapa simétrico.

Figura 5.5.1 – Em 5.5.1a: mapa da expansão do labirinto, sendo as setas em vermelho e amarelo representando as posturas inicial e final, respectivamente, e a trajetória em verde. Em 5.5.1b: mapa simétrico, sendo setas em vermelho e amarelo representando as posturas inicial e final, respectivamente, e a trajetória em verde.

giu as partículas, assim, não necessariamente é uma postura correta e isso pode ser deduzido através da quantidade de ativações da função de sequestro. Assim, para os testes do SAMCL que apresentam uma média menor de sequestros representa uma localização global mais precisa, confirmando o fato de que o aumento de partículas está diretamente relacionada ao aumento da precisão da localização global. Porém como apresentado na seção 5.2, o aumento do conjunto de amostras afeta a localização local e no custo computacional. Assim, acima de 3500 partículas, o SAMCL começa a apresentar divergências na postura final.

- Em relação ao SA-KLD, as análises feitas não diferem das realizadas na seção 5.4. Sendo que o erro de localização encontrado é dado pelo espalhamento das partículas próximo do tempo final, assim o algoritmo não convergiu a tempo e sua postura final foi divergente dos outros testes.

- Por fim, o KLD apresentou um tempo superior ao algoritmo híbrido e erros de localização global, algo já previsto, dado as características deste algoritmo, como já apresentado na seção 3.1.

5.5.2 Ambiente Simétrico

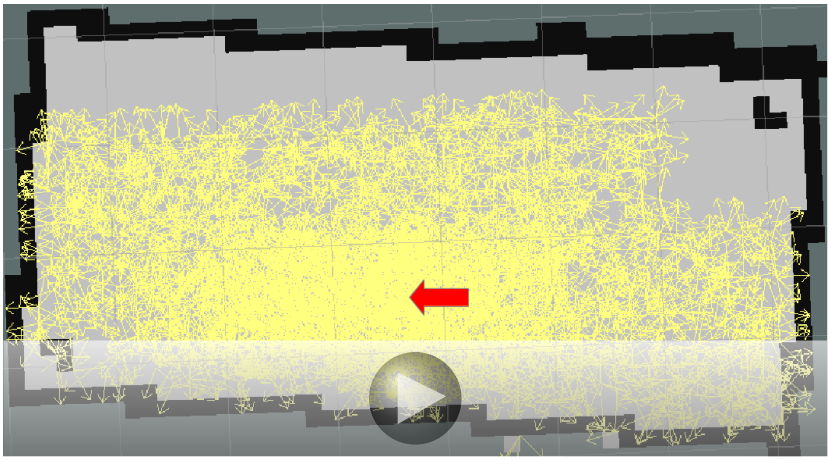
Definir uma localização precisa em ambientes simétricos ou com poucos pontos de referência é um desafio grande para um algoritmo de localização. A figura 5.5.2 ilustra bem esta situação.

Apesar do robô se deslocar pelo mapa todo, as partículas levam mais tempo para convergirem e há uma taxa maior de erros de localização. Em ambientes reais e de dimensões maiores há uma probabilidade menor desta simetria ocorrer. A seguir, a tabela 5.5.2 apresenta as quantidades de localizações à esquerda e à direita do mapa, sendo a primeira a correta. Foram feitos 10 testes para cada algoritmo apenas para ilustrar esta situação.

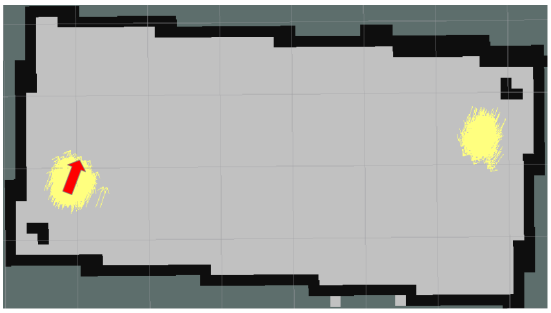
Tabela 5.5.2 – Tabela da posição final referente ao mapa simétrico.

	Posição Final	
	À Esquerda do Mapa	À Direita do Mapa
SA-KLD	7	3
SAMCL	5	5
KLD	4	6

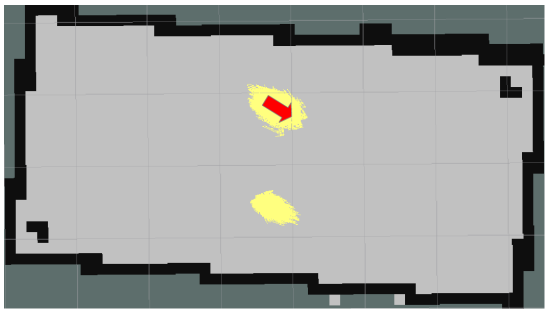
Em alguns testes, o algoritmo localizou o robô à direita do mapa e em outros, ao lado oposto. Isto ocorre devido ao método randômico de criação das partículas, as quais são distribuídas de forma aleatória dentro das áreas SER. Assim, para testes em que criou-se um número levemente maior de partículas tendendo a um lado do mapa, ao deslocar o robô, estas partículas convergem para este mesmo lado. Portanto, se forem realizados



(a) Áreas SER



(b) Mapa simétrico com duas nuvens de partículas no instante t.



(c) Mapa simétrico com duas nuvens de partículas instantes à frente.

Figura 5.5.2 – Distribuição das partículas através de um mapa simétrico. A figura 5.5.2a ilustra as áreas SER para um robô na postura representada pela seta vermelha. A característica do mapa faz com que a distribuição das partículas seja simétrica e mesmo com a movimentação do robô, o algoritmo apresenta dois conjuntos de partículas com crenças semelhantes, figuras 5.5.2b e 5.5.2c. Este efeito ocorre apenas para o SAMCL, pois como o SA-KLD e o KLD reduzem o número de partículas, a nuvem com menor densidade será eliminada pelo processo de *resampling*, mas isto não significa, necessariamente, que a nuvem resultante representará a postura correta.

infinitos testes neste ambiente, a localização tenderá a 50% dos resultados de um lado e 50% de outro.

Nota-se que utilizando apenas um sensor que fornece dados de distância não é suficiente para uma localização em ambientes simétricos, tornando necessário o uso de sensores de naturezas diferentes, tais como: GPS e bússola digital.

6 Conclusão e Trabalhos Futuros

6.1 Conclusão

O algoritmo de localização de Monte Carlo é um método para estimar a postura do robô e sua implementação é relativamente simples. Possui várias limitações, tais como: número fixo de partículas, não trata do problema de sequestro do robô e em ambientes de grandes dimensões é necessário uma quantidade maior de partículas, assim, afetando o tempo de localização, o custo computacional e o rastreamento de posição. Apesar destes problemas, o MCL é um algoritmo eficiente e, devido a estes fatores, ele tem sido bastante estudado nos últimos anos.

A fim de variar a quantidade de partículas ao invés de utilizar um número fixo, o algoritmo KLD utiliza uma aproximação para o cálculo da quantidade necessária de partículas em função do número de *bins* ocupados e apresentou uma grande melhoria para a localização local. Porém o fato do algoritmo distribuir as partículas sobre o mapa todo, torna-o mais lento em suas primeiras iterações e é necessário um maior número de partículas para manter a precisão da localização caso haja aumento da dimensão do mapa.

Em contrapartida, o SAMCL possui uma excelente localização global, devido às áreas SER e à função de sequestro, mas apresenta limitações para o rastreamento de posição, uma vez que a abordagem utiliza um número fixo de partículas. Este algoritmo trouxe uma contribuição muito útil para reduzir o custo computacional, o processo de pré-armazenagem para a criação das matrizes de postura e energia.

Visando aproveitar as qualidades de cada método e propor soluções para suprir as deficiências do MCL, foi proposto um algoritmo híbrido, o qual é capaz de variar o número de partículas tanto no processo de criação, quanto no de *resampling*. E, aplicado junto às áreas SER, resultou em uma abordagem com tempo de localização reduzido, melhor precisão, tanto local quanto global, e um menor custo computacional. Portanto, fazendo uma comparação entre SA-KLD, SAMCL e KLD, o método híbrido mostrou melhores resultados em todos os sub-problemas da localização: local, global e sequestro. É importante definir corretamente os valores de parâmetros, pois o SA-KLD herdou todos os parâmetros dos algoritmos de base e criou-se uma relação entre eles, de tal forma que ao modificar um parâmetro originário do KLD é provável que seja necessário modificar um outro do SAMCL.

A maior contribuição do SA-KLD está no conjunto de amostras variável, principalmente no processo de criação de partículas, cuja vantagem está no cálculo da quantidade de partículas necessárias em função das áreas SER. Dessa forma, uma vez definidos os parâmetros do algoritmo, basta ajustar o valor de *bin* conforme a dimensão do mapa.

6.2 Trabalhos Futuros

O próximo passo deste trabalho é criar um pacote ROS do SA-KLD, de forma que seja compatível com o pacote de navegação autônoma, o *Navigation*[1], já existente e funcional nesta plataforma.

Vários trabalhos futuros poderão ser desenvolvidos, tais como:

- Implementar uma técnica de aprendizagem por reforço para ajustar os parâmetros do algoritmo e encontrar as melhores combinações visando uma otimização do sistema.
- Como para os testes foi usado um número fixo de feixes de luz do laserscan, é de interesse tornar este valor variável em função do número de partículas para uma melhoria na precisão, no tempo de localização ou redução do custo computacional.
- Com o objetivo de reduzir o tempo de localização, outros sensores podem ser adicionados para aumentar as informações do ambiente e reduzir as áreas SER, resultando em menor número de partículas.
- Adicionar dados de sensores de naturezas diferentes para corrigir o problema em ambientes simétricos.
- Implementar técnicas de reconhecimento de padrão para validar a ativação da função de sequestro, por exemplo a transformada de Hough, evitando gastos de processamento em casos de falso sequestro.
- Adaptar o SA-KLD para plataformas multi-robô.
- Utilizar técnicas para localizar o robô em ambientes esparsos, assim possibilitando que outros robôs com sensores de alcance menor utilizem o algoritmo proposto.
- Implementar a técnica de *Filtro de Kalman* para aumentar a precisão da postura dada pelo rastreamento de posição ou qualquer outra técnica para localização robótica que traga alguma melhoria para o algoritmo.

[1] *Navigation é um pacote ROS utilizado para navegação autônoma e possui algoritmos de controle de movimento, planejador de trajetórias, mapeamento, localização e outros. Para que o SA-KLD possa ser oferecido como uma outra opção de um algoritmo de localização, é necessário implementar as transformadas para referenciar os sensores à base do robô e configurar os frames para criar uma ligação laser-odometria-mapa.*

Bibliografia

AUFRÈRE, R. et al. Perception for collision avoidance and autonomous driving. *Mechatronics*, Elsevier, v. 13, n. 10, p. 1149–1161, 2003. Citado na página 49.

BALTZAKIS, H.; TRAHANIAS, P. Hybrid mobile robot localization using switching state-space models. In: IEEE. *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*. [S.l.], 2002. v. 1, p. 366–373. Citado na página 27.

CONTI, F. *Qui Quadrado*. 2009. Disponível em: <http://www.cultura.ufpa.br/dicas/biome/bioqui.htm>. Acesso em: 13 maio 2016. Citado na página 29.

COSTA, A. H. R.; SELVATICI, A. H. P. Robótica móvel. In: _____. [S.l.: s.n.], 2014. cap. Localização usando representação do ambiente, p. 139–160. Citado na página 19.

FOD, A.; HOWARD, A.; MATARIC, M. A laser-based people tracker. In: IEEE. *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*. [S.l.], 2002. v. 3, p. 3024–3029. Citado na página 48.

FOX, D. Adapting the sample size in particle filters through kld-sampling. *The international Journal of robotics research*, SAGE Publications, v. 22, n. 12, p. 985–1003, 2003. Citado 4 vezes nas páginas 16, 28, 58 e 59.

FOX, D. et al. Monte carlo localization: Efficient position estimation for mobile robots. *AAAI/IAAI*, v. 1999, p. 343–349, 1999. Citado 4 vezes nas páginas 16, 20, 21 e 22.

FOX, D.; BURGARD, W.; THRUN, S. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, p. 391–427, 1999. Citado na página 21.

FOXYZ, D.; BURGARDY, W.; THRUNZ, S. Markov localization for reliable robot navigation and people detection. Citeseer, 1999. Citado 3 vezes nas páginas 15, 16 e 18.

GASPARRI, A. et al. A hybrid active global localisation algorithm for mobile robots. In: IEEE. *Robotics and Automation, 2007 IEEE International Conference on*. [S.l.], 2007. p. 3148–3153. Citado na página 27.

GEORGIEV, A.; ALLEN, P. K. Localization methods for a mobile robot in urban environments. *Robotics, IEEE Transactions on*, IEEE, v. 20, n. 5, p. 851–864, 2004. Citado na página 15.

GOUVEIA, M. C. M. *Estudo e Implementação de um Algoritmo de Localização baseado na correspondência de mapas*. Tese (Doutorado) — Universidade do Porto, 2008. Citado na página 16.

JETTO, L.; LONGHI, S.; VENTURINI, G. Development and experimental validation of an adaptive extended kalman filter for the localization of mobile robots. *Robotics and Automation, IEEE Transactions on*, IEEE, v. 15, n. 2, p. 219–229, 1999. Citado na página 26.

JULIO, R. E. Dbml: Uma biblioteca de gerenciamento dinâmico de banda para sistemas multirrobôs baseado em ros. 2015. Citado 2 vezes nas páginas 49 e 50.

KIM, M. et al. Rfid-enabled target tracking and following with a mobile robot using direction finding antennas. In: IEEE. *Automation Science and Engineering, 2007. CASE 2007. IEEE International Conference on*. [S.l.], 2007. p. 1014–1019. Citado na página 47.

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*. [S.l.]: Springer, 2005. p. 142–153. Citado na página 31.

LEE, D.-J. *Three-dimensional optical volume measurement for objects to be categorized*. [S.l.]: Google Patents, 2002. US Patent 6,369,401. Citado na página 48.

LEONARD, J. J.; DURRANT-WHYTE, H. F. Mobile robot localization by tracking geometric beacons. *Robotics and Automation, IEEE Transactions on*, IEEE, v. 7, n. 3, p. 376–382, 1991. Citado 2 vezes nas páginas 16 e 21.

LIU, J. N.; WANG, M.; FENG, B. ibotguard: an internet-based intelligent robot security system using invariant face recognition against intruder. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, IEEE, v. 35, n. 1, p. 97–105, 2005. Citado na página 15.

MEI, Y. et al. A case study of mobile robot's energy consumption and conservation techniques. In: IEEE. *Advanced Robotics, 2005. ICAR'05. Proceedings., 12th International Conference on*. [S.l.], 2005. p. 492–497. Citado na página 47.

MENDES, A.; BENTO, L. C.; NUNES, U. Multi-target detection and tracking with a laser scanner. In: IEEE. *Intelligent Vehicles Symposium, 2004 IEEE*. [S.l.], 2004. p. 796–801. Citado na página 49.

MERWE, R. V. D. et al. The unscented particle filter. In: *NIPS*. [S.l.: s.n.], 2000. v. 2000, p. 584–590. Citado na página 26.

MOBILEROBOTS, A. *Pioneer 3DX*. 2011. Disponível em: <http://www.mobilerobots.com/ResearchRobots/PioneerP3DX.aspx>. Acesso em: 13 outubro 2015. Citado na página 46.

NICKERSON, S. et al. The ark project: Autonomous mobile robots for known industrial environments. *Robotics and Autonomous Systems*, Elsevier, v. 25, n. 1, p. 83–104, 1998. Citado na página 15.

OLIVEIRA, T. E. A. de et al. Sistema autônomo para a estimaçao da pose de um objeto com faces planas em ambiente nao estruturado. 2012. Citado na página 47.

PAVÃO, R. *Entropia informacional e aprendizagem de sequências*. Tese (Doutorado) — Universidade de São Paulo, 2011. Citado na página 28.

PELLEGRINI, J.; WAINER, J. Processos de decisão de markov: um tutorial. 2007. Citado 2 vezes nas páginas 20 e 21.

PRESTES, E.; RITT, M.; FUHR, G. Improving monte carlo localization in sparse environments using structural environment information. In: IEEE. *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. [S.l.], 2008. p. 3465–3470. Citado na página 27.

QUIGLEY, M. et al. Ros: an open-source robot operating system. In: *ICRA workshop on open source software*. [S.l.: s.n.], 2009. v. 3, n. 3.2, p. 5. Citado na página 49.

RAIBERT, M. et al. Bigdog, the rough-terrain quadruped robot. In: *Proceedings of the 17th World Congress*. [S.l.: s.n.], 2008. v. 17, n. 1, p. 10822–10825. Citado na página 15.

RIBEIRO, M. Localização em robótica móvel-odometria. *Instituto Superior Técnico, Instituto de Sistemas e Robótica, Portugal*, 1999. Citado na página 52.

ROMERO, V. et al. Tutorial sobre o filtro de partículas aplicado em localização de robôs móveis. *Escola Politécnica da Universidade de São Paulo, sem data de publicação*. Citado na página 23.

ROS. *Wiki ROS*. 2007. Disponível em: <http://wiki.ros.org/>. Acesso em: 1 agosto 2014. Citado 3 vezes nas páginas 16, 46 e 51.

SASSI, A. B.; WOLF, D. F. Detecção de landmarks para localização de robôs móveis utilizando-se uma câmera monocular. Citado na página 47.

SICK. *SICK LMS 291*. 2006. Disponível em: <http://sicktoolbox.sourceforge.net/docs/sick-lms-technical-description.pdf>. Acesso em: 15 dezembro 2015. Citado na página 48.

SIEGWART, R.; NOURBAKHSH, I. R.; SCARAMUZZA, D. *Introduction to autonomous mobile robots*. [S.l.]: MIT press, 2011. Citado 4 vezes nas páginas 15, 47, 48 e 49.

STAHN, R.; HEISERICH, G.; STOPP, A. Laser scanner-based navigation for commercial vehicles. In: IEEE. *2007 IEEE Intelligent Vehicles Symposium*. [S.l.], 2007. p. 969–974. Citado na página 49.

TEAM-MATH. *Tabela normal z*. 2011. Disponível em: <http://www.normaltable.com/>. Acesso em: 7 abril 2016. Citado 2 vezes nas páginas 30 e 59.

THRUN, S.; BURGARD, W.; FOX, D. *Probabilistic robotics*. [S.l.]: MIT press, 2005. Citado 2 vezes nas páginas 20 e 26.

THRUN, S. et al. Integrating topological and metric maps for mobile robot navigation: A statistical approach. In: *AAAI/IAAI*. [S.l.: s.n.], 1998. p. 989–995. Citado na página 16.

UDACITY; THRUN, S. *Artificial Intelligence for Robotics*. 2011. Disponível em: <https://www.udacity.com/course/viewer#!/c-cs373/l-48704330/e-48748083/m-48665991>. Acesso em: 20 outubro 2015. Citado na página 25.

WAN, E. A.; MERWE, R. V. D. The unscented kalman filter for nonlinear estimation. In: IEEE. *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*. [S.l.], 2000. p. 153–158. Citado na página 26.

WISSPEINTNER, T. et al. Robocup@ home: Scientific competition and benchmarking for domestic service robots. *Interaction Studies*, John Benjamins Publishing Company, v. 10, n. 3, p. 392–426, 2009. Citado na página 15.

WOLF, D. F. et al. Robótica móvel inteligente: Da simulação às aplicações no mundo real. In: *Mini-Curso: Jornada de Atualização em Informática (JAI), Congresso da SBC*. [S.l.: s.n.], 2009. p. 13. Citado 2 vezes nas páginas 15 e 20.

YOON, J.; CRANE, C. D. Ladar based obstacle detection in an urban environment and its application in the darpa urban challenge. In: IEEE. *Control, Automation and Systems, 2008. ICCAS 2008. International Conference on*. [S.l.], 2008. p. 581–585. Citado na página 48.

ZHANG, L. *Self-Adaptive Markov Localization for Single-Robot and Multi-Robot Systems*. Tese (Doutorado) — Université Montpellier II-Sciences et Techniques du Languedoc, 2010. Citado 4 vezes nas páginas 18, 21, 23 e 33.

ZHOU, Y.; LIU, W.; HUANG, P. Laser-activated rfid-based indoor localization system for mobile robots. In: IEEE. *Robotics and Automation, 2007 IEEE International Conference on*. [S.l.], 2007. p. 4600–4605. Citado na página 26.