Universidade Federal de Juiz de Fora
Programa de Pós-Graduação em
Engenharia Elétrica

Lara Furtado Bastos

OTIMIZAÇÃO BIOINSPIRADA APLICADA NA LOCALIZAÇÃO DE ROBÔS MÓVEIS

Dissertação de Mestrado

Juiz de Fora
2016

Lara Furtado Bastos

Otimização Bioinspirada Aplicada na Localização de Robôs Móveis

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica, Área de Concentração em Sistemas de Energia, da Faculdade de Engenharia da Universidade Federal de Juiz de Fora como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Prof. D.Sc. André Luís Marques Marcato
Co-orientador: Prof. D.Sc. Ivo Chaves da Silva Junior

Juiz de Fora
2016

Lara Furtado Bastos

Otimização Bioinspirada Aplicada na Localização de Robôs Móveis

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica, Área de Concentração em Sistemas de Energia, da Faculdade de Engenharia da Universidade Federal de Juiz de Fora como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica.

Aprovada em 08 de Setembro de 2016.

BANCA EXAMINADORA:

Prof. D.Sc. André Luís Marques Marcato - Orientador
Universidade Federal de Juiz de Fora, UFJF

Prof. D.Sc. Ivo Chaves da Silva Junior - Co-orientador
Universidade Federal de Juiz de Fora, UFJF

Prof. D.Sc. Tiago Pereira Nascimento
Universidade Federal da Paraíba, UFPB

Prof. D.Sc. Leonardo Willer de Oliveira
Universidade Federal de Juiz de Fora, UFJF

Prof. D.Sc. Leonardo Rocha Olivi
Universidade Federal de Juiz de Fora, UFJF

Dedico esta dissertação aos meus pais e minha irmã que, com o mais puro amor, fizeram com que eu chegasse até aqui.

AGRADECIMENTOS

Agradeço, primeiramente, a Deus, por permitir que eu fosse capaz, me dando saúde, força de vontade e esperança. Aos meus pais e minha irmã pelo amor incondicional, pela nossa amizade, por serem minha inspiração e fonte de carinho e compreensão. Aos meus amigos e amigas por sempre acreditarem em mim e estarem por perto em todos os momentos. Ao meu amor, que esteve ao meu lado, dia-a-dia, me estimulando, me dando a mão, quando necessário, e não me deixando desistir nunca. Agradeço ao CNPq pelo auxílio financeiro que possibilitou a realização desse trabalho e ao GOHB (Grupo de Otimização Heurística e Bioinspirada) por toda contribuição teórica. Aos meus queridos professores André, Ivo e Leonardo Olivi, meus alicerces e suportes, sem os seus conhecimentos, todas as lições e ajudas, eu não seria capaz. Por fim, meu muito obrigado aos amigos do GRIn, que além de toda ajuda nas nossas rotinas de laboratório, foram como uma família pra mim.

"Pouco conhecimento faz com que as pessoas se sintam orgulhosas. Muito conhecimento, que se sintam humildes..."

Leonardo Da Vinci

RESUMO

O presente trabalho apresenta a adaptação e utilização de um algoritmo da área de inteligência artificial evolucionária, bioinspirado no sistema de ecolocalização de morcegos, para resolver o problema da localização global de robôs móveis em ambientes bidimensionais com mapas conhecidos. Sabe-se, por meio da literatura, que a localização de robôs baseada apenas em dedução via hodometria, do inglês *deduced reckoning* ou *dead-reckoning*, acumula diversos erros de origem estocástica, os quais não podem ser eliminados de maneira determinística, fazendo-se necessários métodos de filtragem estatística para a correta obtenção da localização. Dentre as diversas alternativas conhecidas para solucionar o problema de localização, escolheu-se o Método Recursivo de Monte Carlo, também denominado por Filtro de Partículas, para comparação com os resultados obtidos pelo algoritmo de morcego, por suas características multimodais e não-paramétricas, sendo este um algoritmo clássico na área de localização robótica. O algoritmo de morcegos, do inglês *Bat Algorithm*, é um método recursivo de otimização de estados de um sistema que se encontra num ambiente multimodal. É bioinspirado nos sistemas de ecolocalização encontradas em morcegos e outros animais na natureza. Nos resultados de comparação entre ambos os métodos, a técnica proposta demonstrou melhores resultados tanto para o erro entre a localização real e a estimada pelos métodos quanto para o número de iterações necessárias para alcançar a solução e, consequentemente, o tempo de convergência do algoritmo. Para o desenvolvimento deste trabalho, utilizou-se o programa Matlab® integrado com a plataforma ROS, juntamente com o robô móvel terrestre *Pioneer P3-DX* para os resultados simulados e reais.

Palavras chave: Localização, Robô Móvel, Algoritmo de Morcego, Filtro de Partículas, Algoritmo Bioinspirado.

ABSTRACT

This work presents the adaptation and use an algorithm from evolutionary artificial intelligence area, bioinspired in the echolocation system of bats to solve the problem of global location for mobile robots in two-dimensional environments with known maps. It is widely known in literature that the localization of robots based only on deduced reckoning accumulates many stochastic errors, which cannot be eliminated deterministically, requesting statistical filtering methods to obtain the correct location. Among the various alternatives known to solve the problem of localization, we chose the Recursive Method of Monte Carlo, also kown as Particle Filter, for comparison purposes with the results obtained by the Bat Algorithm, because of its multimodal and non-parametric features, and alse because it is a classic algorithm in robotics localization area. The Bat Algorithm is a recursive optimization method of system states immerse in multimodal environments. It is bioinspired in the echolocation systems found in bats and other animals in nature. In comparison results between the two methods, the proposed technique showed the best results for both localization error and the number of iterations required to reach the solution, and consequently the algorithm convergence time. To develop this work, the Matlab software was used with the ROS framework along with the terrestrial mobile robot Pioneer P3-DX for simulated and real results.

Keywords: Localization, Mobile Robot, Bat Algorithm, Particle Filter, Bioinspired Algorithm.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 APRESENTAÇÃO

A Revolução Industrial teve grande contribuição no desenvolvimento e criação dos robôs. A necessidade de aumentar a produtividade e a qualidade dos produtos levou ao esforço de automatizar as operações industriais. Com o contínuo progresso, as fábricas procuraram equipar-se com máquinas cada vez mais eficientes garantindo sua permanência no mercado competitivo (MURPHY, 2000).

O termo **robô** tem origem na palavra tcheca *robota*, que significa trabalho forçado. É um dispositivo, em sua maioria, eletromecânico capaz de realizar atividades com controle humano, controle parcial com supervisão, de forma autônoma ou pré-programada. São classificados em duas grandes classes:

1. Manipuladores Robóticos: têm forte atuação nas indústrias e se destacam por oferecer aumento da produtividade, alta flexibilidade, qualidade e melhoria da segurança;

2. Robôs Móveis: não possuem base fixa, por isso têm a capacidade de locomoção no espaço tridimensional ou no espaço planar (CRAIG, 2005).

Robótica é a ciência ou o estudo da tecnologia associado com o projeto, fabricação, teoria e aplicação dos robôs. Integra diversos tipos de engenharia, pois requer conhecimentos sobre cinemática, pneumática, hidráulica, eletrônica, programação, controle, entre outros. O processo padrão de criação de robôs começa pela exploração dos sensores, algoritmos e atuadores que serão requeridos para o projeto.

Dotar robôs da capacidade de autonomia, inteligência e interação com o meio é uma área de pesquisa muito atrativa e sua evolução tem desenvolvido robôs cada vez mais eficientes. O termo "inteligente" refere-se à capacidade de atuar e se adaptar, de maneira coerente, ao ambiente onde está inserido; enquanto "autonomia" trata-se do mínimo necessário de intervenção e controle por parte dos seres humanos.

A diferença entre robótica móvel e as demais áreas de pesquisa em robótica é a ênfase em problemas de movimentação em ambientes compostos de obstáculos estáticos e móveis. Para isso, o robô deve ter a capacidade de adquirir e utilizar conhecimento sobre o ambiente, estimar uma posição, ter habilidade de reconhecer obstáculos e responder em tempo real a diferentes situações.

Um dos problemas fundamentais da robótica é a navegação: processo de localizar-se e mover-se em um ambiente de trabalho, povoado de obstáculos, de um ponto de partida até um ponto destino, cumprindo uma trajetória minimamente planejada através do controle do motor. A localização é uma atividade extremamente importante para o bom desempenho do robô em suas missões. Localizar um robô móvel consiste em determinar sua posição (x,y) e orientação (θ) no espaço em um determinado instante de tempo, representadas por um vetor denominado "pose" $[x, y, \theta]$. É uma funcionalidade básica dos robôs móveis para que qualquer tarefa de navegação seja executada. Assim, o robô pode planejar uma trajetória até o seu destino e cumprir as tarefas que lhe forem alocadas.(WOLF et al., 2009)

Devido ao seu baixo custo, a hodometria é um método de localização bastante utilizado em robôs móveis com rodas. Esse método utiliza *encoders* (sensores proprioceptivos de rotação) para medir a rotação das rodas, permitindo calcular a localização do robô, integrando seus movimentos. Porém, podem ocorrer erros devido ao escorregamento das rodas, arredondamento dos valores lidos pelos *encoders* e medidas erradas das dimensões físicas do robô. Ou seja, a hodometria pode gerar erros que se propagam cumulativamente com a distância percorrida. Com isso, o uso da hodometria pode se tornar proibitivo quando o robô percorre grandes distâncias.(BEZERRA, 2004)

A solução para esse problema consiste em lidar com a imprecisão dos sensores e manter uma estimativa consistente da posição do robô no ambiente. A presente dissertação traz, como estratégia para minimização dos erros de hodometria, um algoritmo de otimização bioinspirada, denominado *Bat Algorithm* (BA), ou Algoritmo de Morcego, desenvolvido na Universidade de Cambridge pelo chinês Xin-She Yang, em 2010 (YANG, 2010b). Destaca-se que na literatura especializada não foram encontradas aplicações desta metodologia para o problema em questão, sendo esta uma motivação do trabalho.

A natureza oferece uma grande diversidade de fontes de inspiração para o desenvolvimento de algoritmos de otimização. Tais algoritmos são chamados de bioinspirados ou inspirados na natureza e estão inseridos na grande área de pesquisa chamada Com-

putação Natural.

As metaheurísticas biologicamente inspiradas são reconhecidas como abordagens eficientes para resolução de diversos problemas de otimização. Os algoritmos que compõem esta classe podem utilizar diferentes rotinas de exploração e intensificação na busca pela solução ótima. A exploração refere-se à exploração global, enquanto a intensificação refere-se à exploração local no espaço de busca das soluções do problema. Dentre os diferentes algoritmos bioinspirados pode-se destacar o grupo de Inteligência de Enxame.

A Inteligência de Enxame se caracteriza por algoritmos que possuem inspiração no comportamento coletivo de insetos, como formigas, abelhas, cupins e também de animais como peixes, morcegos e pássaros. Neste grupo encontram-se, por exemplo, o algoritmo de colônia de formigas (*Ant Colony Optimization*)(COLORNI et al., 1992) inspirado no comportamento de busca por alimento das formigas, otimização de enxame de partículas (*Particle Swarm Optimization*)(LI; SHAO; QIAN, 2002) inspirado no movimento de cardumes de peixes e bandos de pássaros, algoritmo da abelha (*Bee Algorithm*)(NAKRANI; TOVEY, 2004) inspirado na busca de alimentos das abelhas, dentre outros (ANDRÉ; PARPINELLI, 2014).

O BA é uma teoria inspirada na sofisticada capacidade biológica utilizada pelos morcegos, o processo de ecolocalização (Figura 1). Baseia-se na emissão de ondas ultrassônicas e a correspondente medição do tempo gasto para estas ondas voltarem à fonte após serem refletidas pelo alvo (presa ou obstáculo). A taxa de emissão de pulsos e a amplitude dos pulsos sonoros emitidos pelos morcegos variam com a estratégia de caça.

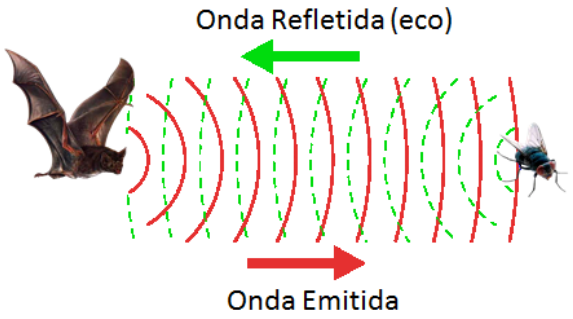

Figura 1: Sistema de Ecolocalização

De acordo com a Tabela 1, quando o morcego está no processo de busca global, ou seja, ainda está explorando o ambiente, a taxa de pulso sonoro (r) é baixa enquanto a

amplitude desses pulsos (A) é alta. Por outro lado, na busca local, quando identificada a presa, a amplitude diminui (proporcionalmente à distância percorrida) e a taxa de pulso aumenta.

Tabela 1: Processo de Ecolocalização

Tipo de Busca	**Status**	**Amplitude**	**Taxa de Pulso**
Global	Procurando	ALTA	BAIXA
Local	Encontrando	BAIXA	ALTA

No modelo computacional, cada morcego representa uma possível solução para o problema, ou seja, uma possível localização do robô móvel. Uma população de morcegos é espalhada pelo ambiente simulado e se move no espaço de busca do problema, atualizando continuamente a frequência, amplitude, velocidade e posição de cada elemento buscando encontrar a solução ótima. A cada nova iteração, cada morcego é atualizado seguindo a melhor solução encontrada pela população (KRAUSE; CORDEIRO; LOPES, 2013).

Para avaliar e comparar o desempenho do BA, foi necessária a implementação de outro método de localização. Dentre os algoritmos de localização para robôs móveis, destaca-se o método de Markov, que é baseado no filtro de Bayes, uma poderosa técnica de estimação estatística. Como se trata de uma formulação geral para a solução do problema, existem diversas formas de implementar esse algoritmo e de representar os dados do ambiente e da posição dos robôs. Dentre as quais destacam-se o filtro de Kalman, o método GRID e o método de Monte Carlo (também denominado *Particle Filter*) (THRUN; BURGARD; FOX, 2005)(KÜNSCH, 2005).

A escolha do método *Particle Filter* (PF) ou Filtro de Partículas Clássico (THRUN; BURGARD; FOX, 2005) deu-se, principlamente, por ter sido implementado pelo grupo de robótica, do laboratório de desenvolvimento desta dissertação, para uma aplicação semelhante. Além disso, sua teoria segue a mesma linha estatística do BA, o que torna justa uma comparação entre os métodos. Isso pode ser visto, por exemplo, pela definição das partículas e dos morcegos presentes no ambiente de simulação.

O PF consiste em manter várias hipóteses de localização num mapa, onde cada hipótese é representada por uma unidade chamada "partícula". Ao ser iniciado, esse algoritmo distribui partículas por todo o mapa, comparando os dados obtidos dos sensores com as informações disponíveis no mapa. A partir desta comparação, é associado um peso para cada partícula. Este peso representa a chance desta partícula ser a real

localização do robô e é calculado durante a execução do algoritmo. Partículas com peso baixo são eliminadas enquanto as partículas com peso alto são replicadas. Com o tempo, todas as partículas tendem a se concentrar próximas à localização correta do robô (Figura 2) (WOLF et al., 2009).

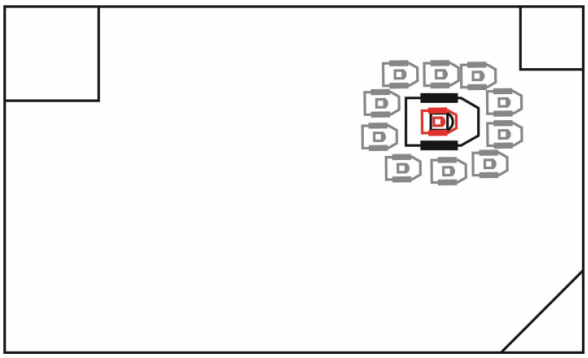

Figura 2: Cenário Final do Filtro de Partículas

1.2 OBJETIVO DA DISSERTAÇÃO

A dissertação tem como objetivo explorar a utilização do Algoritmo de Morcegos para ter a capacidade de estimar a real localização de um robô móvel terrestre dentro de um ambiente bidimensional, mapeado de maneira multimodal, minimizando os erros de hodometria ou influências externas relativas à localização, que, por consequência, podem interferir no alcance de seu objetivo final ou no percurso de sua trajetória.

Nesta dissertação, o algoritmo de busca bioinspirado é confrontado com o PF de forma a evidenciar sua competitividade para a aplicação proposta, apresentando bons resultados quanto ao desempenho computacional, tempo de convergência, número de iterações e valores baixos para o erro entre a pose apresentada pelo robô e a esperada pelo algoritmo.

1.3 MOTIVAÇÃO

A principal motivação desta dissertação é explorar uma técnica de otimização relativamente recente, onde não foi encontrada sua aplicabilidade na literatura especializada para a localização de um robô móvel terrestre em um mapa bidimensional conhecido.

1.4 PUBLICAÇÃO DECORRENTE DESSE TRABALHO

Durante a execução desta dissertação, o seguinte artigo, abordando o tema aqui em questão, foi publicado no Simpósio Brasileiro de Automação Inteligente, em 2015:

- FURTADO, L. et al. Bat Search Algorithm Aplicado na Localização de Robôs Móveis. *Simpósio Brasileiro de Automação Inteligente*, Natal, 2015.

1.5 ASPECTOS DE IMPLEMENTAÇÃO

Nesta dissertação, os programas *Matrix Laboratory* ($MATLAB^{®}$) e a plataforma de desenvolvimento *Robot System* (ROS) foram integrados para a criação dos ambientes simulado e prático, o desenvolvimento dos algoritmos e a comunicação do robô móvel terrestre com esses algoritmos propostos. Toda implementação encontra-se detalhada nos capítulos 3 e A.

O $MATLAB^{®}$ é um *software* de computação numérica de análise e visualização de dados. Embora seu nome signifique Laboratório de Matrizes, seus propósitos atualmente são bem mais amplos. Surgiu como um programa para operações matemátiOperatingcas sobre matrizes, mas ao longo dos anos transformou-se em um sistema computacional bastante útil e flexível. Seu ambiente de trabalho é fácil de ser utilizado, pois os problemas e soluções são escritos em linguagem matemática e não na linguagem de programação tradicional, como muitos outros *softwares* utilizam. Assim o $MATLAB^{®}$ é uma ferramenta e uma linguagem de programação de alto nível.

O ROS é uma estrutura flexível para a escrita de *software* para robôs. É uma coleção de ferramentas, bibliotecas e convenções que visam simplificar a tarefa de criar uma atividade ou um comportamento para o robô, que muitas vezes é complexo e robusto, através de uma ampla variedade de plataformas robóticas. Da perspectiva do robô, problemas que parecem trivial para os seres humanos, muitas vezes variam entre as instâncias de tarefas e ambientes. Para isso, o ROS foi criado com o intuito de incentivar a robótica colaborativa no desenvolvimento de *softwares* (*Open Source Robotics Foundation*, 2007).

1.6 CONTEÚDO DOS CAPÍTULOS

No primeiro capítulo tem-se uma introdução teórica a respeito dos assuntos relevantes para realizar a localização de um robô móvel terrestre pela adaptação de dois

algoritmos de busca, que é o objetivo principal do trabalho. Aqui são descritos os objetivos e a motivação desta dissertação, qual a publicação relevante, os aspectos de implementação e o conteúdo de cada capítulo.

No segundo capítulo, tem-se uma revisão bibliográfica sobre localização de robôs móveis, quais as principais técnicas utilizadas e alguns exemplos de trabalhos publicados nessa área de aplicação. Foi o capítulo de embasamento com todas as pesquisas, trabalhos científicos, citações de artigos, publicações e estudos que deram suporte teórico e técnico para a realização desta dissertação. No final do capítulo é abordado o tema de Localização por Visão que tem atraído a atenção de muitos pesquisadores da área.

No terceiro capítulo, é descrita, detalhadamente, toda a modelagem do método proposto, como foram criados os ambientes simulado e prático, o funcionamento dos métodos de localização, o modelo matemático que rege cada algoritmo de busca e o funcionamento do algoritmo principal. Neste capítulo, encontram-se todas as fórmulas matemáticas e teorias abordadas para que os algoritmos de busca realizassem a localização do robô móvel.

No quarto capítulo, são apresentados os resultados obtidos pelos algoritmos abordados nesta dissertação tanto para o ambiente simulado, quanto para o ambiente prático. São descritos os valores atribuídos a cada uma das variáveis dos algoritmos de busca e através de figuras e histogramas é feita uma análise estatística para comparação dos métodos, além da interpretação e justificativa dos testes realizados.

Por fim, o quinto capítulo traz conclusões acerca da dissertação, com argumentos explicativos mostrando pontos positivos e negativos das técnicas utilizadas. Traz sugestões para trabalhos futuros que podem dar continuidade ao tema e à linha de pesquisa, além de mostrar características do algoritmo principal que poderiam ser manipuladas de diferentes formas.

2 REVISÃO BIBLIOGRÁFICA SOBRE LOCALIZAÇÃO DE ROBÔS MÓVEIS

2.1 LOCALIZAÇÃO NA ROBÓTICA MÓVEL

A robótica móvel abrange diversas questões relacionadas ao controle de veículos e problemas de operação em ambientes complexos. O trabalho conjunto de pesquisadores e especialistas de diversas áreas de engenharia têm provido dispositivos de *hardware* e *software* cada vez mais capazes de controlar os robôs móveis de maneira eficiente.

A interação de um robô com o ambiente é realizada por meio de ciclos de percepção e ação, que consistem: em obter informação através de sensores, processar as informações para seleção da ação e executar a ação através do acionamento dos atuadores. Essas operações podem parecer simples, mas o controle de sistemas robóticos tem complicações físicas (cinemática), mecânicas (dinâmica), eletrônicas (falta de precisão dos sensores) e computacionais que fazem do controle uma tarefa difícil e sujeita a erros (BEKEY, 2005)(SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011)(PESSIN, 2013).

A capacidade de se auto-localizar é um dos requisitos fundamentais para a autonomia de um robô móvel. O objetivo da localização de robôs em ambientes bidimensionais é determinar a posição $[x, y]$ e a orientação θ do veículo em relação a um referencial externo. Nesta dissertação, a pose do robô é ciclicamente estimada a partir da informação fornecida pela leitura dos *lasers* e corrigida sempre que o método de localização, algoritmo de otimização bioinspirada ou *Particle Filter*, for solicitado.

A localização desses robôs móveis é motivo de intensa investigação científico tecnológica. Existe uma grande diversidade de técnicas baseadas em diferentes princípios físicos e algoritmos. Esses métodos apresentam algumas características menos desejáveis, podendo ser computacionalmente exigentes, requererem um tempo de processamento considerável ou serem inflexíveis, principalmente quando se pretende aplicá-los em robôs relativamente pequenos e de baixo custo.

A hodometria é um dos métodos mais simples e mais amplamente utilizados para

estimação momentânea da pose de um robô. Sendo um método de localização relativa, a sua utilização em aplicações práticas tem, quase sempre, o fornecimento de informação atualizada, em tempo real, da posição e orientação do veículo durante os intervalos de tempo. O deslocamento do robô a partir de uma posição inicial é obtida pela integração dos movimentos lineares de cada uma das rodas.

As principais vantagens da hodometria estão associadas ao baixo custo de implementação, à elevada taxa de amostragem permitida e à boa precisão alcançada a curto prazo. A principal restrição está associada com a acumulação ilimitada de erros ao longo da navegação, fruto da integração dos erros com que as medidas do movimento das rodas são afetadas. Este fato faz com que a estimativa da pose do robô se afaste dos seus valores reais, impossibilitando que a sua utilização seja bem sucedida durante um período longo de tempo, sem recorrer a um método de localização absoluta (LIMA, 2010).

Para minimizar esses erros ou até mesmo eliminá-los, um algoritmo de busca pode ser associado às atividades do robô, garantindo corrigir sua real localização de maneira otimizada. Um dos algoritmos mais populares na literatura, que cumpre esse papel, é o *Particle Filter* (PF).

Na área de estatística, foi criado, pela equipe do Projeto Manhattan, o teorema de Monte Carlo. Nomeado em homenagem ao cassino de Monte Carlo (localizado em Mônaco) do qual o pai de John vonNeumann (um dos mais importantes matemáticos do século XX) era um assíduo frequentador. É um método sofisticado de estimação estocástica que se baseia na simulação dos dados que se deseja saber/obter por meio de uma grande quantidade de partículas em diferentes estados do espaço/universo de discurso (PERALTA, 2010).

O PF pertence à classe dos métodos estocásticos e também é conhecido como Método Recursivo de Monte Carlo. É um método numérico de integração adequado para lidar com problemas não lineares e não Gaussianos, usando números aleatórios. Para resolver um problema através desse método é usada uma série de tentativas aleatórias e a precisão do resultado final depende, em geral, do número dessas tentativas. Esse equilíbrio entre a precisão do resultado e o tempo de computação é uma característica extremamente útil. Desde a década de sessenta, grande atenção tem sido devotada a estes problemas, entretanto, somente com o aumento do poder computacional foi possível tornar o seu uso mais corrente, já que esse algoritmo necessita de um elevado esforço computacional para ter uma boa resolução (MARTINS et al., 2012)

Existem diversos tipos de implementação e adaptação desse algoritmo, como, por exemplo, em Wardhana et al. (2013) que propõe um algoritmo do PF Modificado para localizar robôs móveis enquanto navega em um determinado ambiente. Essa modificação acontece para reduzir o esforço computacional do método clássico ao manipular cada uma de suas partículas. Outra adaptação pode ser vista no artigo Romero et al. (2014), que descreve a aplicação do PF para uma busca global feita de maneira cooperativa por dois robôs móveis, além da derivação das equações cinemáticas que descrevem o sistema.

Além do *Particle Filter*, muitos outros métodos são explorados para essa aplicação. A localização de robôs móveis é uma das áreas mais exploradas da robótica devido a sua importância para a resolução de problemas, como: navegação, mapeamento e SLAM (*Simultaneous Localization and Mapping*), no português: mapeamento e localização simulatânea. Muitos trabalhos apresentaram soluções envolvendo cooperação, comunicação e exploração do ambiente, onde, em geral, a localização é obtida através de ações randômicas ou puramente orientadas pelo estado de crença. Abaixo serão vistas propostas de algoritmos de busca em trajetórias.

Um método muito eficiente para localização de robôs móveis é o ICP (*Interactive Closest Point*). O algoritmo calcula o movimento relativo do robô entre duas configurações consecutivas do ambiente, visando o alinhamento dessas medidas e a estimativa da posição do robô. Para isso, faz um registro automático de dois modelos de dados geométricos (independente do seu tipo), não necessitando de condições prévias e convergindo para a solução ótima, na maioria dos casos (DJEHAICH et al., 2013).

Um outro método de localização é a utilização de Mapas e/ou Bússula Digital. Lopes, Lau e Reis (2000) apresentaram uma solução para a navegação num dado mapa conhecido, que é a introdução prévia do respectivo mapa no robô. O trajeto a ser seguido pode ser passado para o robô ou pode ser o robô a planejar o caminho a seguir. Um mapa é conhecido *a priori*, e utiliza este processo como um misto de trajetória pré-programada e capacidade de planejamento para contornar obstáculos dinâmicos não previstos. A informação de posição, em cada momento, é obtida conjugando dados do sensor com a informação de uma bússola digital.

Um outro exemplo em que o mapa é construído pelo próprio robô está em Vale et al. (2001), onde são descritos métodos de aprendizagem do meio. Sua aplicação requer algum tempo para a construção do mapa, o que pode ser um obstáculo para a sua aplicabilidade quando o tempo é limitado.

Na tese de Pinheiro (2013) é apresentado um modelo de planejamento para localização de robôs móveis utilizando POMDP (*Partially Observable Markov Decision Process*) e Localização de Markov. O método indica a melhor ação que o robô deve efetuar em cada momento, com o objetivo de diminuir a quantidade de passos.

Santana (2007) utilizou o filtro de Kalman estendido em conjunto com a hodometria para encontrar a localização de um humanóide, que não possui sensores, guiado por um robô escravo com rodas. Cruz (2013) também apresenta uma abordagem ao filtro de Kalman estendido, onde implementa a abordagem em arquiteturas reconfiguráveis para o problema de localização de robôs móveis.

Uma técnica de localização, usada por animais e humanos, é o uso de pontos de referência, quer sejam absolutos ou locais. Os pontos de referências locais podem ser obstáculos, marcas pré-definidas, faróis estáticos simples ou múltiplos. As referências absolutas podem ser o sol, as estrelas, o norte magnético ou uma referência local cuja posição absoluta é conhecida. A utilização de múltiplos pontos de referência em simultâneo pode permitir a determinação da posição do robô por triangulação (ROY et al., 1999).

Muitos trabalhos disponíveis na literatura, que abordam o tema localização, fazem uso de pontos de referência juntamente com câmeras de vídeo para localizar veículos móveis através da análise de imagens, essa localização é denominada Localização por Visão. O detalhamento e, consequentemente, a quantidade de dados que podem ser retirados de uma informação visual faz com que sua aplicação na robótica móvel seja de grande potencial.

2.2 LOCALIZAÇÃO POR VISÃO

A robótica móvel busca, com frequência, inspiração nos seres vivos, tornando natural a simulação da capacidade humana de guardar, reconhecer e relembrar uma informação visual. Essas características permitem que o robô possa navegar de forma autônoma por um ambiente com um sistema de percepção confiável.

Devido à utilização maciça de câmeras digitais, o preço do sensor de imagem diminui significativamente, tornando-as muito atraente. As câmeras podem ser usadas para resolver uma série de problemas-chave na robótica e em outras operações automatizadas, pois fornecem uma variedade de informações do ambiente, consomem pouca energia e são facilmente integradas ao *hardware* do robô.

Estas novas técnicas constroem mapas visuais que consistem, normalmente, em um conjunto único de marcos identificados pelo robô no seu caminho. Em ambientes desconhecidos, o robô automaticamente identifica marcos, e quando revisita áreas mapeadas, utiliza os conhecidos marcos para estimar e melhorar sua pose e a localização dos marcos.

Existem diversos métodos de extração de pontos de referência nas imagens, alguns mais simples, como os detectores de características distintas (bordas e cantos), até os mais robustos, que escolhem pontos de referências que possam ser identificados mesmo com alterações da cena em questão. Outro ponto relevante desse tipo de informação é o fato desses sensores (câmeras de vídeo) serem considerados baratos em relação aos demais utilizados, como os *lasers*.

Um exemplo de aplicação de Localização por Visão pode ser observado em Garcia et al. (2007), onde foi desenvolvido um arcabouço que permite a localização e identificação de grupos de robôs. O arcabouço é composto por um conjunto de câmeras fixadas sobre o ambiente de discurso e um sistema de *software* capaz de localizar e identificar os robôs, que nesse caso carregam algum tipo de marco visual (Figura 3).

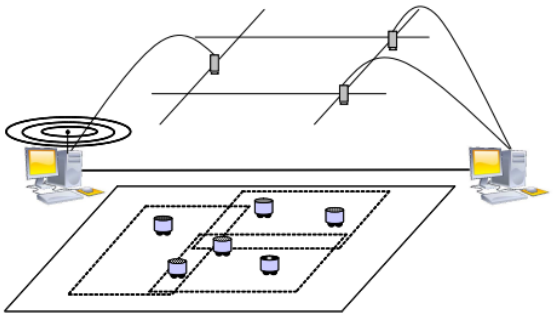

Figura 3: Localização por Visão
(GARCIA et al., 2007)

A implementação do arcabouço foi dividida em dois programas diferentes: o primeiro é um programa cliente que controla as câmeras e cuida da aquisição de informações sobre a localização dos alvos que se encontram em seu campo de visão. O segundo é um programa servidor, cuja função é centralizar as informações obtidas em todas as instâncias dos clientes, e fornecer como saída a localização de todos os alvos encontrados.

Outra aplicação pode ser vista na tese de doutorado Couto (2012). O método de localização é baseado na criação de uma memória visual (através da detecção e descrição

de pontos de referência de imagens capturadas), com o método SURF (*Speeded Up Robust Features*), associada aos dados de hodometria. Permite que a localização seja obtida, posteriormente, pelo pareamento entre quadros memorizados e a cena atual observada pelo robô.

A técnica de visão global também é apresentada por Nascimento (2014) e visa o desenvolvimento de um sistema de localização de robôs móveis, em ambientes fechados. Seu objetivo é melhorar o processo de localização de plataformas robóticas móveis, considerando apenas informações do momento atual.

2.3 OTIMIZAÇÃO BIOINSPIRADA

Nesta dissertação, o método utilizado para a localização do robô móvel terrestre foi o *Bat Algorithm*, um algoritmo de otimização bioinspirado que está inserido na grande área de pesquisa chamada Computação Natural (Figura 4). Pertence à classe de meta-heurísticas e são baseados no comportamento de certos animais, nos processos e modelos de sistemas e fenômenos biológicos.

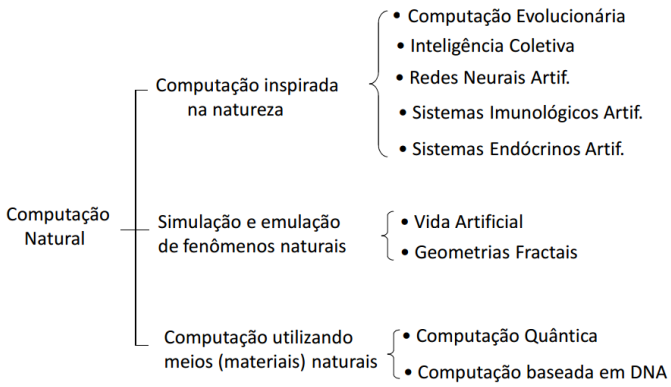

Figura 4: Áreas da Computação Natural

O algoritmo de ecolocalização é um exemplo de Inteligência de Enxame, também denominado Inteligência Coletiva, esses algoritmos são baseados no comportamento social e coletivo de insetos (formigas, cupins, abelhas) ou agrupamentos de animais (morcegos, peixes, aves). A inteligência destes sistemas dá-se pelo comportamento coletivo dos indivíduos, que é gerado através de pequenas interações individuais e chamado de comportamento emergente (ANDRÉ; PARPINELLI, 2014).

Neste grupo, encontram-se alguns exemplos como:

- Colônia de Formigas (*Ant Colony Optimization*) (COLORNI et al., 1992);

- Abelhas Artificiais (*Artificial Bee Colony*) (NAKRANI; TOVEY, 2004);
- Algoritmo de Cardume Artificial (*Artificial Fish School Algorithm*) (LI; SHAO; QIAN, 2002);
- Algoritmo do Morcego (*Bat Algorithm*) (YANG, 2010a);
- Algoritmo de Vagalumes (*Firefly Algorithm*) (YANG, 2008);
- Otimização por Enxame de Partículas (*Particle Swarm Optimization*) (KENNEDY, 1995).

Na literatura atual, não foram encontrados trabalhos que utilizassem algoritmos de ecolocalização para a localização de robôs móveis, nem em busca de trajetórias. Isso acontece porque o BA é um algoritmo recente, mas que tem expandido sua área de aplicabilidade, como, por exemplo, no Problema da Mochila Multidimensional (KRAUSE; CORDEIRO; LOPES, 2013) ou no planejamento estático da expansão de sistemas de transmissão de energia elétrica via ecolocalização (ARÊDES et al., 2014). Porém, se mostra promissor ao ser comparado às demais técnicas de localização devido aos poucos parâmetros de manipulação, além de uma metodologia capaz de explorar um ambiente de forma eficaz.

3 METODOLOGIA EMPREGADA NA LOCALIZAÇÃO DE ROBÔS MÓVEIS

O objetivo desse trabalho é a localização de um robô móvel terrestre em um mapa bidimensional conhecido. Essa localização tem de ser confiável e, por isso, é preciso a correção das poses do robô durante o percurso de uma trajetória qualquer dentro desse mapa. Essa correção é necessária devido a fatores anteriormente citados, como os erros de hodometria, os quais podem afetar o controle do robô, impossibilitando-o de cumprir seu objetivo.

Um exemplo de fator externo que pode influenciar significativamente na trajetória do robô é a característica do piso do ambiente em que está inserido. Supõe-se que o robô tenha que andar 1 metro em linha reta, se o piso for muito escorregadio, as rodas podem derrapar e a localização final ser superior ou inferior à esperada. Como a hodometria é calculada a partir da rotação das rodas do robô, o algoritmo de controle considera que o robô alcançou seu objetivo mesmo quando não o fez. Para solucionar esse problema é necessário um algoritmo capaz de determinar a pose real do robô através das leituras do *laser*.

Para começar essa implementação, é necessário criar um ambiente simulado de testes com elementos que representem uma situação real do robô móvel para essa aplicação. Assim, o cenário inicial consiste em um robô (com as dimensões iguais a de um P3DX) inserido em um mapa previamente conhecido, realizando uma trajetória pré-determinada pelo usuário.

3.1 AMBIENTE SIMULADO

Os algoritmos presentes nesta dissertação foram desenvolvidos em um *software* matemático chamado MATLAB®, o qual permite a criação e alteração de um ambiente simulado com as características desejadas. Sua programação é feita através de uma linguagem matemática e o programa opera, essencialmente, com um tipo de objeto que é a matriz numérica retangular. Todas as variáveis no MATLAB® representam

matrizes que podem ser interpretadas como escalares (matrizes de uma linha e uma coluna, 1x1) ou vetores (matrizes de uma linha ou uma coluna).

Para o desenvolvimento dos algoritmos, o primeiro passo foi criar um ambiente simulado, que consiste em um mapa bidimensional conhecido onde o robô irá percorrer uma trajetória qualquer. Para isso, foi preciso gerar uma figura em branco e inserir segmentos de reta que representassem as paredes e os obstáculos do mapa em questão. Esses segmentos de reta são originados por dois pontos, conforme mostra a Figura 5.

Figura 5: Segmento de Reta

Para passar esses parâmetros para o programa, a variável responsável por desenhar o mapa recebe uma matriz, apresentada pela Figura 6, onde cada linha representa um segmento de reta, ou seja, dois pontos (e 4 parâmetros: x_1, y_1, x_2, y_2).

```
                Ponto 1: P1x,y      Ponto 2: P2x,y
                P1x   P1y           P2x   P2y
L = [0          0                   4680  0       <-- reta 1
     0          0                   0     3200    <-- reta 2
     0          3200                4680  3200    <-- reta 3
     4680       0                   4680  3200    <-- reta 4
     0          2280                920   2280    <-- reta 5
     920        2280                920   3200    <-- reta 6
     4190       2850                4680  2850    <-- reta 7
     4190       2850                4190  3200    <-- reta 8
     4030       0                   4680  650];   <-- reta 9
```

Figura 6: Matriz para Construção do Mapa no $MATLAB^{®}$

A matriz da Figura 6 dá origem ao mapa apresentado pela Figura 7.

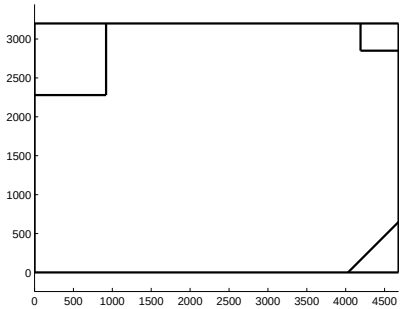

Figura 7: Mapa Simples Construído no $MATLAB^{®}$

É importante ressaltar que todos os mapas dos ambientes simulados usados nesta dissertação têm característica circular, portanto, se o robô virtual ultrapassa os limites do mapa, é transportado para o lado oposto (RUSSELL et al., 2003). Ou seja, ao atingir essas extremidades, sua pose é alterada para que esse transporte possa acontecer. Esse conceito foi inserido com o intuito de movimentar as partículas, os morcegos e o robô virtual pelo ambiente, sem que eles ficassem "presos" nas extremidades.

O próximo passo foi inserir à Figura 7, que agora tem o aspecto de um mapa, um robô virtual, Figura 8. Como esse robô deve apresentar as mesmas características e dimensões do robô real, foram coletados dados do robô móvel terrestre P3-DX para que não houvesse distorções nas dimensões de suas rodas e largura, por exemplo.

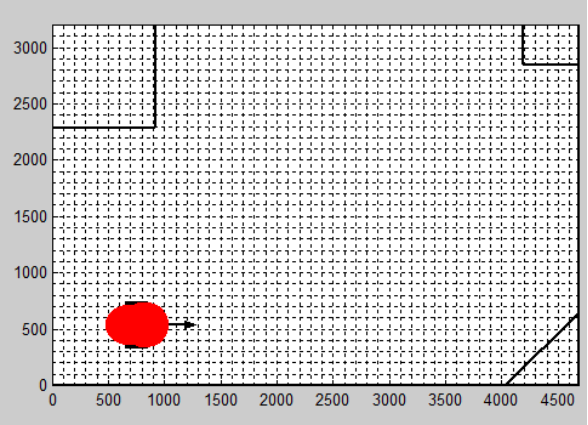

Figura 8: Robô Virtual no Ambiente Simulado

A única diferença entre o robô virtual e o robô real, nesse ponto, é o sensor que está acoplado ao robô terrestre Pioneer P3-DX, representado pela Figura 9. Esse sensor é um *laser scan* SICK LMS-200, do fabricante SICK^{TM}, possui 181 medidas de laser, onde cada uma delas traz a informação da distância entre o robô e o obstáculo que esse feixe está alcançando. É acoplado na frente do robô terrestre e, por isso, alcança uma angulação de -90^{o} a 90^{o}, com passo de 1^{o} em 1^{o}.

Figura 9: Sensor SICK LMS-200 Acoplado ao Pioneer P3-DX

Para simular o *laser* SICK LMS-200 no robô virtual, cada feixe de luz será representado por uma reta traçada entre o robô e o obstáculo mais próximo, em uma determinada direção. A equação geral da reta é dada pela Equação 3.1.

$$ax + by + c = 0 \tag{3.1}$$

Onde:

- a, b e c são números reais constantes;
- x e y são as coordenadas de um ponto pertencente à reta em questão.

Os valores dos coeficientes da equação geral da reta são obtidos pelas equações 3.2, 3.3 e 3.4.

$$det \begin{bmatrix} x & y & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{bmatrix} = 0 \tag{3.2}$$

$$\begin{aligned} a &= y_1 - y_2 \\ b &= x_2 - x_1 \\ c &= x_1 y_2 - x_2 y_1 \end{aligned} \tag{3.3}$$

$$(y_1 - y_2)x + (x_2 - x_1)y + (x_1 y_2 - x_2 y_1) = 0 \tag{3.4}$$

Onde (x_1, y_1) e (x_2, y_2) são as coordenadas de dois pontos (P_1 e P_2) não coincidentes da reta, como mostra a Figura 10.

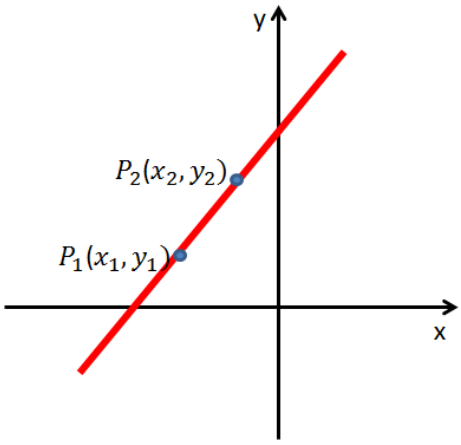

Figura 10: Pontos Pertencentes a Reta

Considerando que o sensor tenha sido instalado no centro do robô, o primeiro ponto de todas as retas desse sensor já estará definido pelas coordenadas do robô ($x_1 = x_r, y_1 = y_r$). Para gerar o outro ponto, primeiro escolhe-se a angulação do raio de *laser* que desejamos (θ_s) e, em seguida, utiliza-se as equações 3.5 e 3.6 para dar origem a uma reta que representará um feixe de *laser* (na direção do ângulo escolhido) como mostra a Figura 11.

$$x_2 = x_1 + cos(\theta_r(i) + \theta_s(i)) \tag{3.5}$$

$$y_2 = y_1 + sen(\theta_r(i) + \theta_s(i)) \tag{3.6}$$

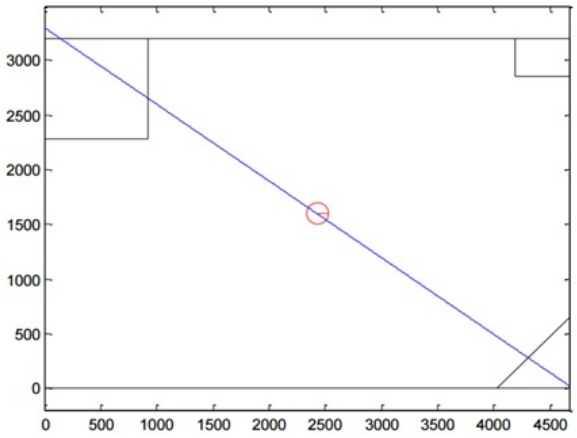

Figura 11: Simulação do Feixe de *Laser* para um ângulo de 35º

Agora, compara-se cada reta do sensor com todas as retas do mapa para saber se existe a possibilidade delas serem paralelas ou se intersectarem. Se elas se intersectarem, busca-se todos os pontos de interseção (representados pela seta verde da Figura 12) para uma avaliação:

1. A primeira avaliação serve para descartar os pontos de interseção localizados na parte traseira do robô. Isso acontece porque o *laser* utilizado no robô real (SICK LMS 200) possui um alcance de $-90°$ a $90°$ referente a parte frontal do robô. Esse pontos estão representados pelo "x" vermelho na Figura 12;

2. A segunda avaliação consiste em descartar os pontos de interseção que não estão inseridos em um segmento de reta, representados pela cor roxa na Figura 12.

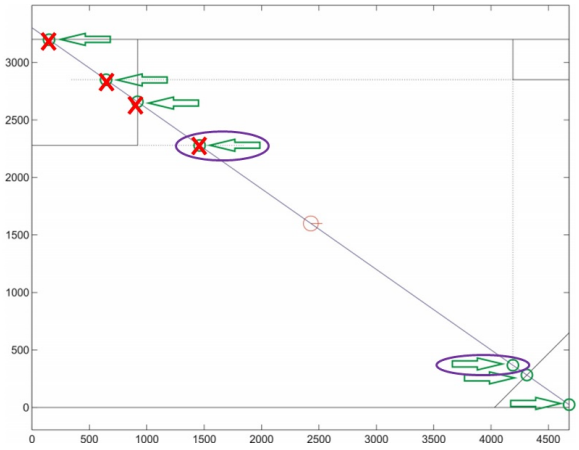

Figura 12: Pontos de Interseção e Pontos Descartados

Se essas condições são satisfeitas, o ponto analisado é um ponto de interseção válido e o próximo passo é calcular as distâncias entre o robô e esses pontos válidos. O menor valor obtido é o obstáculo mais próximo do robô e, portanto, o mesmo que está sendo lido pelo laser do robô real. Depois de testar cada laser, o robô virtual terá todas as leituras calculadas da mesma forma que o robô real, como mostra a Figura 13.

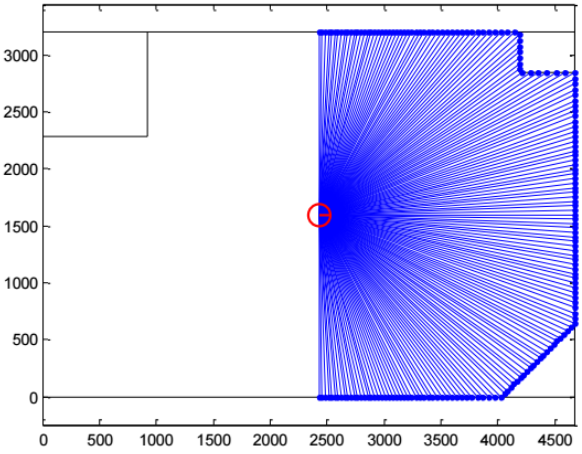

Figura 13: Leitura de Laser do Modelo Cinemático

O número de feixes de *laser* pode ser configurado para atender às necessidades do usuário, pois quanto maior esse número, maior o tempo de processamento do algoritmo. Isso pode ser feito aumentando o valor do incremento entre os ângulos do sensor como mostra a Tabela 2.

Tabela 2: Laser Virtual - Número de Feixes

Vetor de Angulação do Laser	Incremento	Número de Feixes
[-90:1:90]	1	181
[-90:10:90]	10	19
[-90:30:90]	30	7

Esse robô virtual precisa realizar uma rotina de deslocamento pelo mapa bidimensional conhecido. A programação dessa rotina permite que o usuário determine a trajetória que o robô virtual deve seguir, escolhendo a localização inicial do robô (ponto de partida - Figura 14 (a), (b) e (c)), pontos de parada (*waypoints* - Figura 14 (d)) e o ponto final (alvo/objetivo).

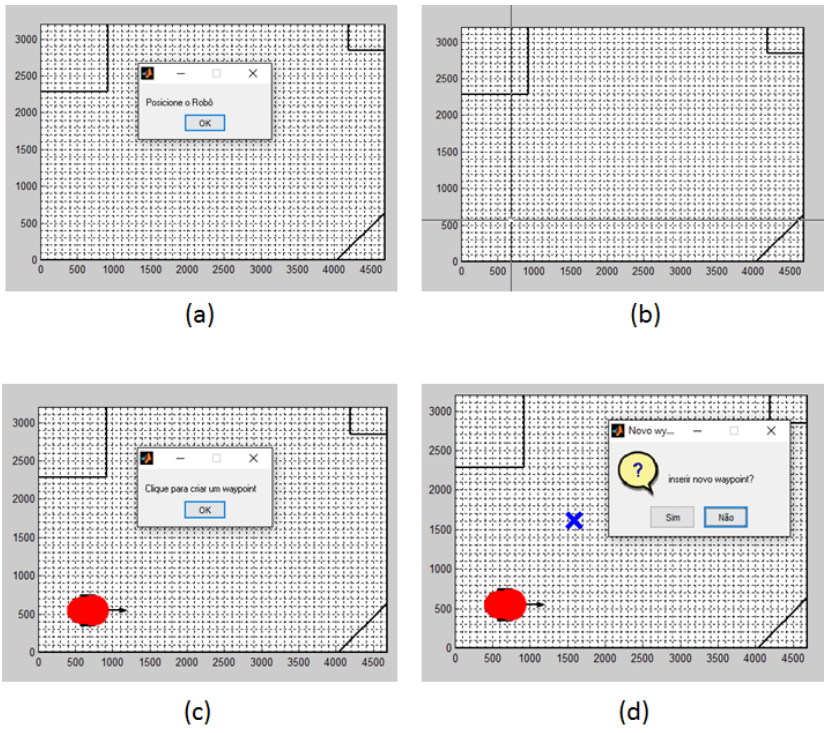

Figura 14: Trajetória do Robô Virtual

Depois de traçada a trajetória, o robô virtual precisa de um controle de velocidade que permita seu deslocamento respeitando os pontos de parada pré-determinados pelo usuário, além de parar ao atingir o ponto final. Para isso, foi escolhido um tipo de controle (PID), conhecido como Twiddle (THRUN; BURGARD; FOX, 2005), que ajusta a velocidade do robô virtual de acordo com sua distância ao objetivo, com uma relação diretamente proporcional.

O controle Proporcional Integral Derivativo (PID) é uma das técnicas mais em-

pregadas quando se deseja realizar o controle de variáveis contínuas. Consiste em um algoritmo matemático, que tem por função o controle preciso de uma variável em um sistema, permitindo-o operar de forma estável no ponto de ajuste desejado, mesmo que ocorram variações ou distúrbios que afetariam sua estabilidade.

No controle PID cada ação desenvolve uma função determinada:

- A ação proporcional elimina as oscilações da variável, tornando o sistema estável, mas não garante que a mesma esteja no valor desejado (*setpoint*), esse desvio é denominado *off-set*. A ação proporcional trabalha corrigindo o erro do sistema, multiplicando o ganho proporcional pelo erro, dessa forma agindo com uma maior amplitude de correção a fim de manter a estabilidade da variável.

- A ação integral elimina o desvio de *off-set*, fazendo com que a variável permaneça próxima ao valor desejado para o sistema mesmo após um distúrbio, ou seja, a variável permanece próximo ao *set-point* mesmo que ocorra uma variação brusca nas condições de operação. Essa ação realiza a integração do erro no tempo, portanto quanto maior for o tempo de permanência do erro no sistema, maior será a amplitude da ação integral.

- A ação derivativa tem sua resposta proporcional à taxa de variação da variável do processo, aumentando a velocidade de resposta do sistema caso a presença do erro seja detectada. Logo, em sistemas de resposta lenta, a ação derivativa permite antecipar o aumento do erro e aumentar a velocidade de resposta do sistema. Quando o sistema a ser controlado possui maior velocidade de resposta, a ação derivativa pode ser desativada, pois não há necessidade de antecipar a resposta ao erro porque o sistema pode corrigir rapidamente seu valor.

O ambiente simulado foi descrito e o robô realiza a trajetória definida pelo usuário, o próximo passo é entender como e onde os algoritmos de localização (BA e PF) serão aplicados.

3.2 FUNCIONAMENTO DOS MÉTODOS DE LOCALIZAÇÃO

Com todos os requisitos do cenário inicial atendidos (robô móvel terrestre realizando uma trajetória pelo mapa 2D conhecido), descreve-se, agora, o funcionamento do método proposto. A ideia principal é explorar todo o mapa para que cada lugar em seu interior seja considerado um ponto provável da real localização do robô móvel.

Essa exploração do ambiente de busca é feita pelas partículas. As partículas são simulações do robô real e possíveis soluções para o problema. Por isso, quanto maior sua população (η), maior a probabilidade de sucesso, porém maior o tempo de processamento do algoritmo.

Inicialmente, essas partículas são criadas no ambiente simulado, com poses ($[x,y,\theta]$) aleatórias, afim de cobrir todo o espaço de busca. Em seguida, é feita a leitura dos sensores, representada pela Figura 15, tanto para o robô real (em vermelho e com escala amplificada), quanto para os robôs simulados (em verde).

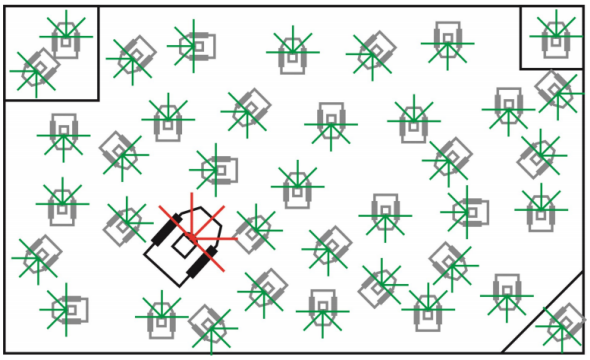

Figura 15: Robôs Realizando as Leituras do Laser

Os valores obtidos por cada sensor dos robôs simulados são comparados às medidas do robô real (reta por reta). Se essas medidas forem semelhantes, as partículas recebem um peso alto, de acordo com a Função Objetivo (FOB) proposta, o que garante seu permanecimento no cenário. Caso contrário, as partículas recebem pesos menores e são descartadas.

A FOB escolhida é uma Função de Distribuição Normal, também denominada Função de Distribuição Gaussiana (MOIVRE, 1967), é representada pela curva da Figura 16. É um importante modelo de distribuição estatística que garante a convergência da média dos dados para uma distribuição normal conforme esse número de dados aumenta, mesmo que esses dados sejam distribuídos aleatoriamente.

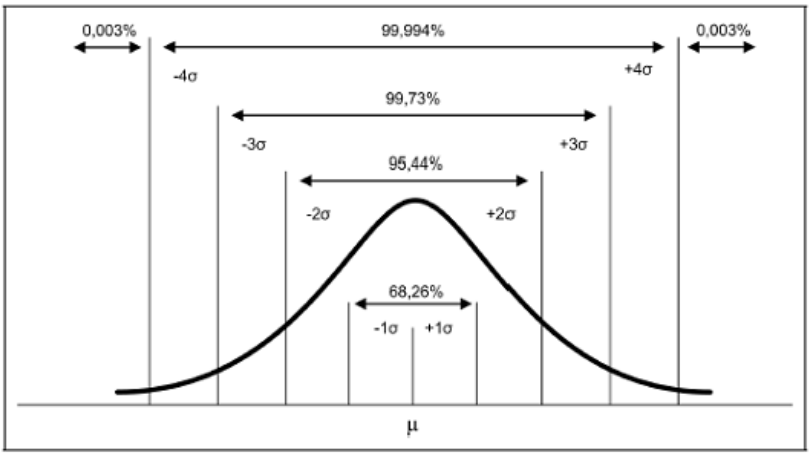

Figura 16: Curva de Distribuição Normal/Gaussiana

Onde:

- As probabilidades da variável aleatória normal são dadas pela área sob a curva;
- O desvio padrão (σ) determina o comportamento dessa curva;
- O ponto máximo da curva normal encontra-se na média, que é também a mediana e a moda da distribuição.
- É uma distribuição totalmente descrita pela média (μ) e pelo desvio padrão através da Função de Densidade de Probabilidade (FDP) (Equação 3.7);
- É simétrica em relação à média, podendo ser considerada justa em sua aplicabilidade.

$$FOB(X) = \frac{1}{\sqrt{2\pi(\sigma)^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.7}$$

O desvio padrão e a média são medidas estatísticas obtidas pela coleta de vários eventos aleatórios e servem para caracterizar uma variável aleatória e seu espaço amostral. Com uma FDP Gaussiana, em relação a uma média, 68.27% das ocorrências estão em 1σ, 95.45% estão entre 1σ e 2σ, 99.73% estão entre 2σ e 3σ e quase 100% entre 3σ e 4σ, como pode ser visto na Figura 16.

Nesta dissertação, a relação da equação 3.7 (FOB) com o peso das partículas acontece da seguinte forma:

1. O valor obtido pelo feixe de laser "i" do robô real é comparado ao valor obtido pelo mesmo feixe "i" dos sensores das partículas;

2. Para isso, o valor de distância obtido pelo feixe "i" do sensor do robô real é centrado na Gaussiana, assumindo um valor de comparação, conforme apresentado pela Figura 17.

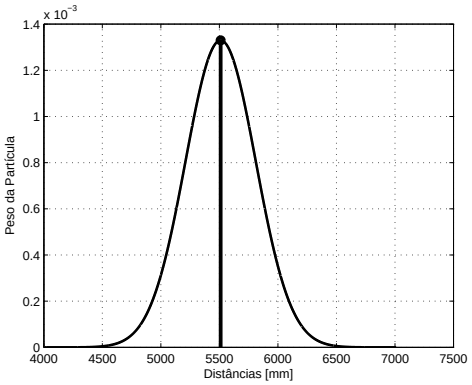

Figura 17: Leitura do Robô Real Centrada na Gaussiana

3. Os valores de distâncias, obtidos pelos feixes de *laser* "i" de cada partícula, alimentam a FOB, assim como o valor obtido pelo *laser* do robô real e um desvio padrão (pré-determinado) do sensor, $\sigma_\pi = 300$ mm, Equação 3.8. Onde "$part_{dist}(:,i)$" representa um vetor coluna com as leituras do feixe "i" de todas as partículas e "$robo_{dist}$" representa a leitura do laser do robô para o mesmo feixe "i".

$$FOB(i) = \frac{1}{\sqrt{2\pi(\sigma_\pi)^2}} e^{-\frac{(part_{dist}(:,i) - robo_{dist}(i))^2}{2\sigma_\pi^2}} \tag{3.8}$$

4. As distâncias das partículas são comparados à do robô real e recebem um valor equivalente à Gaussiana, distribuindo-se pela curva normal. Na Figura 18, os pontos vermelhos representam os valores obtidos pelas μ partículas (através da FOB) e o ponto azul indica a leitura da melhor partícula da iteração.

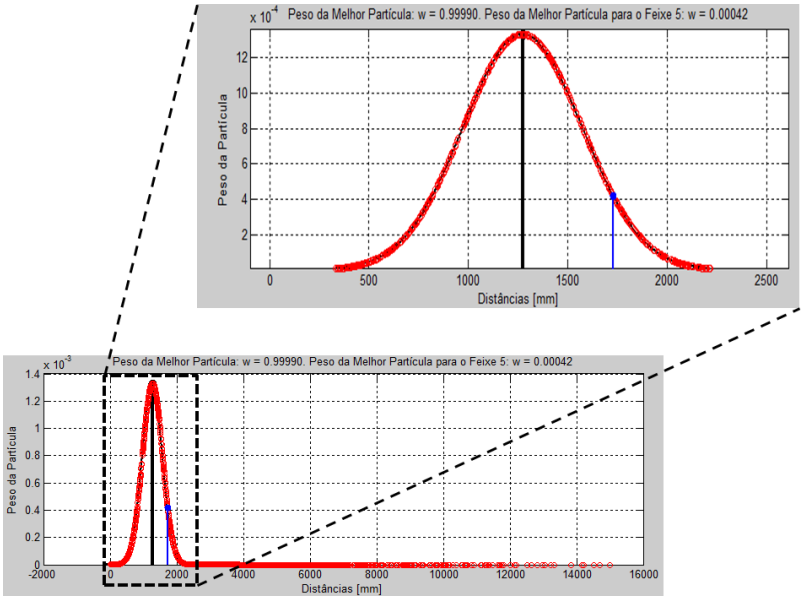

Figura 18: Leitura do Feixe de Laser 5

5. Esse processo repete-se para todos os feixes do sensor (no caso desta dissertação, 7) e o peso da partícula ($Peso_{part}(j)$) é calculado pela multiplicação dos valores, obtidos pela FOB, para cada uma das leituras do sensor ($FOB_{feixe(i)}$) de acordo com a Equação 3.9.

$$Peso_{part}(j) = FOB_{feixe1}(j) * FOB_{feixe2}(j) * ... * FOB_{feixe7}(j) \tag{3.9}$$

6. Por fim, esse valor é normalizado no intervalo [0,1], através da Equação 3.10, para facilitar a manipulação dos dados.

$$Peso_{part}(j) = \frac{Peso_{part}(j)}{\sum_1^N Peso_{part}} \tag{3.10}$$

Portanto, o maior número de partículas reamostradas se concentra próximo à localização desejada e, como visto anteriormente, recebem valores altos para seus pesos, o que significa que suas poses estão próximas à pose do robô real.

Para compensar as partículas que foram descartadas (devido aos seus pesos baixos), outras novas são criadas com localização e orientação iguais à da melhor partícula encontrada (que mais se aproximou do robô real, portanto com maior peso), somadas à um pequeno desvio padrão, em x, y e θ, com o intuito de gerar partículas próximas às melhores soluções encontradas anteriormente, como pode ser visto na Figura 19.

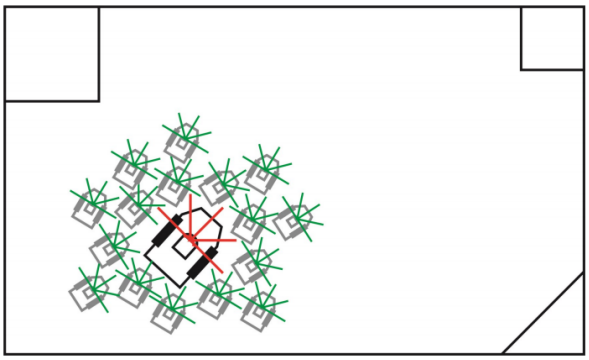

Figura 19: Robôs com Leituras Próximas às do Robô Real

Essas novas partículas passam pelo mesmo processo das anteriores, recebendo um peso e, de acordo com seu valor, podem ser descartadas ou não. Com isso, na medida que ocorrem novas iterações, as partículas concentram-se cada vez mais próximas ao robô real, Figura 20.

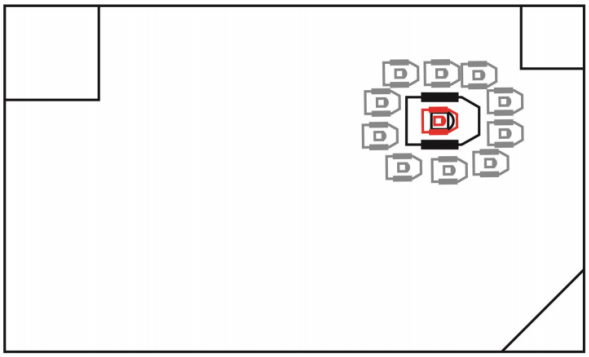

Figura 20: Cenário Final do Ambiente Simulado

O ciclo de obtenção, comparação e reprodução das melhores partículas é repetido até que:

1. O peso da melhor partícula seja maior que uma tolerância imposta pelo usuário, que pode variar de acordo com a aplicação, principalmente, pelo tamanho do mapa;

2. O número de iterações seja menor que o "número de iterações máxima" definida pelo usuário. Vale lembrar que quanto maior esse número, o tempo de convergência do algoritmo também pode se elevar.

É importante ressaltar que o robô real cumpre sua trajetória navegando pelo espaço de busca, mas cada localização é realizada com o robô parado. Ou seja, em um intervalo

de tempo (definido pelo usuário) e a cada *waypoint*, o robô para e os algoritmos (BA e PF) realizam a localização, de acordo com suas respectivas metodologias.

3.2.1 REGION OF INTEREST - ROI)

Uma região de interesse (ROI) é um subconjunto de uma imagem ou um conjunto de dados identificado para uma finalidade específica em um determinado instante de tempo. Na robótica, quando o ambiente de discurso é muito grande e/ou o algoritmo que rege o sistema exige certa complexidade computacional, é indicado o uso de uma região de interesse.

Esse conjunto de dados pode ser:

- Uma forma de onda ou conjunto de dados 1D, onde a ROI é um intervalo de tempo ou frequência;
- Imagem ou conjunto de dados 2D, onde a ROI é definida por limites determinados sobre a imagem;
- Imagem com profundidade ou conjunto de dados 3D, onde a ROI pode ser o contorno de um objeto ou uma superfície.

No caso dessa dissertação, o ambiente simulado (mapa) é particionado e uma região de interesse (de característica Gaussiana) é selecionada para que o algoritmo opere apenas dentro desses limites estabelecidos. Isso acontece porque os algoritmos, principalmente de busca, precisam processar todas as informações do mapa (global), para uma solução local.

A característica Gaussiana foi escolhida pelos mesmos motivos citados anteriormente na escolha da FOB:

- É um importante modelo de distribuição que garante a convergência da média dos dados para uma distribuição normal, mesmo que sejam distribuídos aleatoriamente;
- Fácil de manipular;
- Pode ser considerada justa em sua aplicabilidade;
- Concentra 68.27% das ocorrências próximas à solução ótima.

Portanto, conhecendo a posição anterior do robô móvel terrestre P3-DX e com as informações de movimentação do mesmo, é possível identificar uma região de interesse, dentro do mapa em que está operando, onde o robô real está localizado. Com isso, seria possível reduzir o número de partículas para explorar apenas essa área de interesse, reduzindo os cálculos e todo processamento do algoritmo apenas com as informações dessa região.

A Figura 21 representa a primeira iteração do algoritmo. Isso quer dizer que as partículas são espalhadas de maneira uniforme e aleatória por todo o mapa, pois precisa-se da localização inicial do robô real para criar a região de interesse.

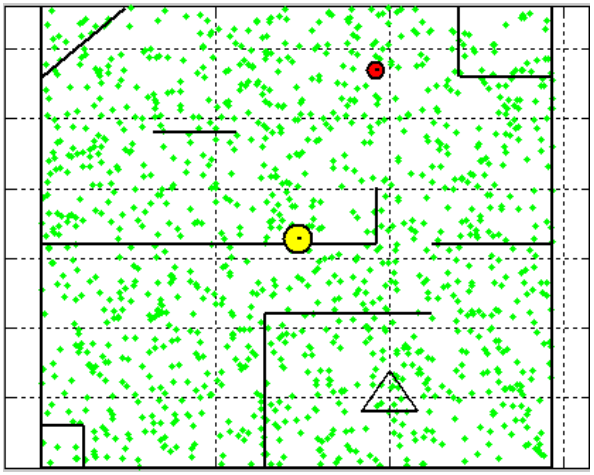

Figura 21: Primeira Iteração do Algoritmo

Na Figura 22, pode-se observar que a segunda iteração cria a região de interesse, mas parece deslocada em relação ao robô real. Isso acontece porque é a primeira localização depois da movimentação do robô, ou seja, as partículas ainda não foram avaliadas e a ROI usa informações como a distância percorrida pelo robô para ser criada. Sua área não foi delimitada para não atrapalhar o ambiente simulado. As partículas (em verde) são reamostradas em uma região próxima ao robô real (em vermelho), é possível notar que o sistema opera garantindo a característica circular do mapa, já que as partículas também aparecem na parte inferior do gráfico, respeitando a ROI.

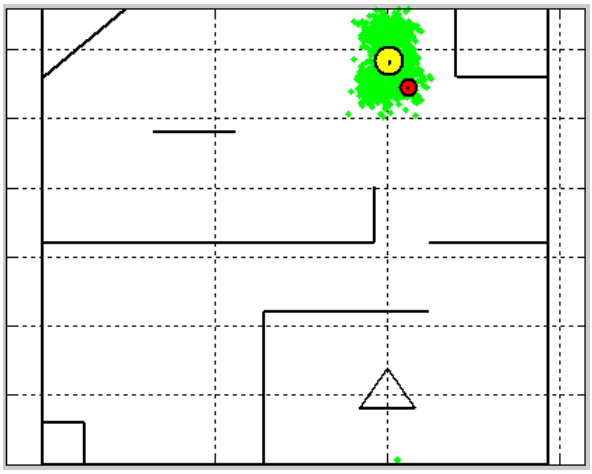

Figura 22: Região de Interesse (ROI)

A cada iteração essas partículas tendem a ficar mais aglomeradas em torno do robô real, como mostra a Figura 23. Por ter característica Gaussiana, a ROI se apresenta em forma de elipse.

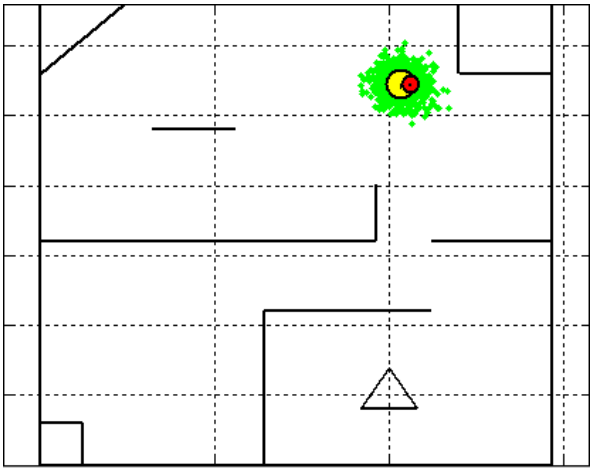

Figura 23: Região de Interesse nas Últimas Iterações

Isso traz grandes benefícios para o algoritmo, já que a complexidade computacional e, consequentemente, o tempo de processamento podem ser reduzidos, além de diminuir o problema de ambiguidade do mapa. Essa ambiguidade acontece, principalmente, em mapas mais simples. Ao fazer as leituras dos sensores do robô real e das partículas simuladas os algoritmos de busca se confundem a respeito de sua real localização porque as leituras são as mesmas para qualquer um dos pontos mostrados pelo mapa da Figura 24.

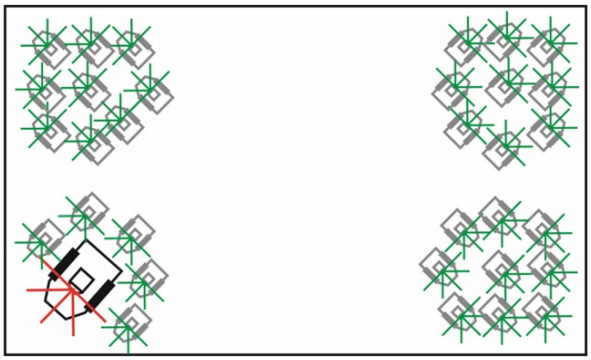

Figura 24: Cenário de Ambiguidade

Com todos os requisitos do ambiente simulado atendidos, nas próximas seções, serão descritos os funcionamentos dos algoritmos de busca, PF e BA, respectivamente.

3.3 PARTICLE FILTER (PF)

O PF é uma implementação não paramétrica do filtro de Bayes, onde seu objetivo principal é seguir uma variável de interesse enquanto evolui no tempo, tipicamente com uma Função de Densidade de Probabilidade (FDP) não-Gaussiana e potencialmente multimodal (apresenta diversos pontos de máximo ou mínimo). A base do método é construir uma representação baseada em amostras de toda a FDP. Uma série de ações são feitas, cada uma modificando o estado da variável de interesse de acordo com um modelo. Além disso, em certos momentos, uma observação que chega, restringe o estado da variável de interesse, nesse momento.

O filtro de partículas é recursivo por natureza, ou seja, é um processo de repetição que baseia-se em casos simples anteriormente manipulados para solucionar casos mais complexos. A variável de interesse (nesta dissertação, a pose do robô $[x,y,\theta]$) é representada como um conjunto de "N" amostras (as partículas), cada partícula está constituída de uma cópia da variável de interesse e um peso que define a contribuição desta partícula para a estimação global da variável. Uma estimação da variável de interesse é obtida pela soma ponderada de todas as partículas. (ROMERO et al., 2014)

O algoritmo do PF tem muitas versões dependendo da área de aplicação, mas a estrutura básica segue o seguinte roteiro:

1. Defina uma boa função objetivo (FOB) para a avaliação das partículas. Esta função objetivo será determinante nos resultados obtidos;

2. Distribua inicialmente de maneira aleatória uniforme N partículas por todo o domínio de análise do sistema que se quer otimizar;

3. Obtenha o peso de importância de cada uma das partículas por meio da função objetivo e calcule o erro entre o *goal* e o valor desejado;

4. Promova um sorteio tipo "roleta de cassino" para a reprodução das partículas baseando-se no peso de cada partícula. Dessa forma, naturalmente as partículas que possuírem bons pesos (maiores) serão reproduzidas mais vezes, e as partículas ruins serão eliminadas;

5. Repetir os passos 3 e 4 até que o peso da melhor partícula esteja dentro de uma tolerâcia previamente imposta pelo usuário (nesta dissertação representa uma distância aceitável entre a pose do robô real e a solução do problema).

A versão utilizada nesta dissertação é apresentada pelo pseudocódigo descrito na Figura 25.

1) Inicializa o espaço de discurso: carrega o mapa
2) Inicializa a FOB: $f(x)$
3) Inicializa os Parâmetros para o funcionamento do algoritmo: η, $iter_{máx}$, σ_x, σ_y, σ_θ, σ_π, TOL;
4) Inicialização das Partículas: x_i
5) Avaliação das Partículas: $f(x_i)$
6) Atualiza a Melhor Partícula: $Particula_{Melhor}$
7) Enquanto ($Número\ de\ Iterações < iter_{máx}$) {
8) Para $i = 1$ *até* η {
9) Avaliação das Partículas: $f(x_i)$
10) Atualiza a Melhor Partícula: $Particula_{Melhor}$
11) Reamostra as Partículas com Melhores Pesos através da "Roleta de Cassino"
12) Atualiza a Média das Melhores Partículas } }

Figura 25: Algoritmo Adaptado do *Particle Filter*

Esse algoritmo segue a rotina:

- Linha 1: carrega o mapa desejado que será o universo de discurso (criado através de uma matriz, como, por exemplo, a da Figura 6);

- Linha 2: cria uma função objetivo, que no caso dessa dissertação foi a Equação 3.8. Essa função tem como parâmetro de entrada as leituras (distâncias) dos lasers dos robôs (simulados e real) e retorna o peso de cada partícula;

- Linha 3: são passados os parâmetros de:

    1. Tamanho da população (η): número escalar que pode ser definido pelo usuário considerando que quanto maior esse número, maior parte do espaço de busca será preenchido, maior a probabilidade de encontrar a solução ótima com menos iterações e maior o tempo de processamento de dados;
    2. Desvio Padrão de reamostragem das partículas em x (σ_x), y (σ_y) e θ (σ_θ) e desvio padrão do sensor (σ_θ): números escalares definidos pelo usuário de acordo com o domínio de interesse;
    3. Tolerância mínima de parada ou tolerância máxima de solução ótima (TOL) e número de iterações máxima ($iter_{max}$): números escalares definidos pelo usuário de acordo com o domínio de interesse.

- Linha 4: inicializa a população de partículas aleatoriamente $x_i = [x, y, \theta]$ (com i variando de 1 até η);

- Linha 5: faz a avaliação de cada partícula através da função objetivo (Equação 3.11) utilizando os parâmetros de distâncias do robô real ($robo_{dist}$) e das partículas ($part_{dist}$), conseguidos pelas leituras dos *lasers*, e o desvio padrão do sensor (σ_π);

$$FOB(X) = \frac{1}{\sqrt{2\pi(\sigma_\pi)^2}} e^{-\frac{(part_{dist} - robo_{dist})^2}{2\sigma_\pi^2}} \tag{3.11}$$

- Linha 6: identifica as melhores partículas através de um intervalo de valores de pesos (w) definido pelo usuário. O índice das partículas que estão nesse intervalo são guardados em um vetor ($[i, j, ...]$) que atualiza a variável "melhor partícula" ($Particula_{Melhor}$) com a pose das partículas desses índices ("i", "j", ...), apresentada na forma de uma matriz (Equação 3.12);

$$\begin{bmatrix} p_{(i)}.x & p_{(i)}.y & p_{(i)}.th \\ p_{(j)}.x & p_{(j)}.y & p_{(j)}.th \\ p_{(...)}.x & p_{(...)}.y & p_{(...)}.th \end{bmatrix} \tag{3.12}$$

- Linha 7: O ciclo de avaliação e reprodução se repete até que o número de iterações máxima ($iter_{max}$) seja atingido;

- Linha 8: para cada uma das partículas, faça os passos seguintes;

- Linha 9: repete o passo da Linha 5;

- Linha 10: repete o passo da Linha 6;

- Linha 11: a reprodução é feita de maneira justa, dando oportunidade a todas as partículas, até as ruins. É modelada através do algoritmo de "roleta de cassino" e quanto maior o peso da partícula, mais chances ela terá de ser reamostrada. Essa reprodução considera um desvio padrão (σ) em torno da pose da melhor partícula, aumentando as chances de sucesso.

  O nascimento dessas novas partículas precisa obedecer o limite do mapa, sendo assim, quando a pose das melhores partículas são somadas ao desvio padrão, essas novas poses podem ultrapassar esses limites. Para que isso não aconteça, a função "mod" foi utilizada na geração das novas posições (Equações 3.13, 3.14 e 3.15). Ela retorna o resto de uma divisão, garantindo que as partículas permaneçam dentro do mapa bidimensional e sua orientação seja um ângulo menor que 360º.

$$Part_{nova}.x = mod(\frac{Particulas_{Melhores}.x + (\sigma_x * randn)}{LimiteMapa_x}) \tag{3.13}$$

$$Part_{nova}.y = mod(\frac{Particulas_{Melhores}.y + (\sigma_y * randn)}{LimiteMapa_y}) \tag{3.14}$$

$$Part_{nova}.th = mod(\frac{Particulas_{Melhores}.th + (\sigma_\theta * randn)}{360}) \tag{3.15}$$

  Assim, a nova pose das partículas é apresentada pela matriz da Equação 3.16:

$$[Part_{nova}.x, Part_{nova}.y, Part_{nova}.th] \tag{3.16}$$

- Linha 12: atualiza a variável "melhor partícula" com a média das partículas com maiores pesos através das Equações 3.17, 3.18 e 3.19.

$$Particula_{Melhor}.x = \frac{\sum Particulas_{Melhores}.x}{num.particulas.melhores} \tag{3.17}$$

$$Particula_{Melhor}.y = \frac{\sum Particulas_{Melhores}.y}{num.particulas.melhores} \tag{3.18}$$

$$Particula_{Melhor}.th = \frac{\sum Particulas_{Melhores}.th}{num.particulas.melhores} \tag{3.19}$$

Assim, a melhor partícula será apresentada por:

$$[Particula_{Melhor}.x, Particula_{Melhor}.y, Particula_{Melhor}.th] \tag{3.20}$$

Se o peso dessas partículas atingir a tolerância mínima de parada (TOL), o programa é finalizado, caso contrário, o ciclo de avaliação e reprodução continuam até que a condição de parada da Linha 7 seja satisfeita.

É possível analisar como a inicialização das partículas interfere de forma significativa no desempenho do algoritmo, por isso, em ambos os métodos, elas são inicializadas com a mesma pose. Quanto maior o número de partículas, maior a probabilidade de uma partícula nascer próxima ao valor máximo da FOB, Figura 26. Assim, essa partícula tende a levar as outras a se reamostrarem próximas a ela, onde receberão bons pesos e aumentarão as chances de encontrar a melhor posição possível.

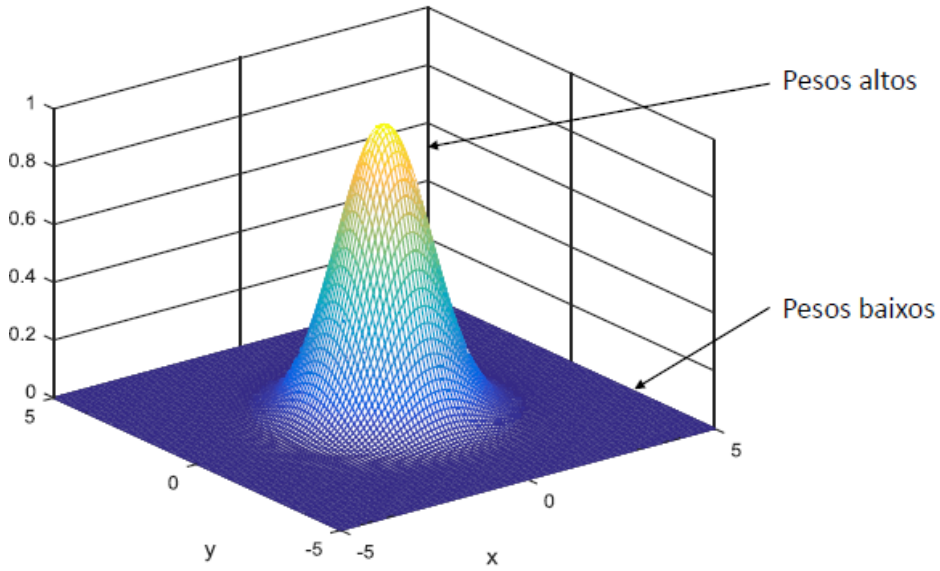

Figura 26: Comportamento da FOB

Um fator desforável em aumentar a população de partículas é que cada uma delas utiliza o modelo simulado do *laser*, exigindo todos os cálculos apresentados pela seção "Ambiente Simulado" deste capítulo. Portanto, quanto maior esse número, maior o esforço computacional e, consequentemente, mais lenta a resolução do problema.

3.4 BAT ALGORITHM (BA)

O BA é inspirado no processo de ecolocalização dos morcegos, pela emissão de ondas sonoras, utilizadas durante o seu voo para detectar presas e evitar obstáculos.

São os únicos mamíferos com asas e essa capacidade de ecolocalização é extremamente avançada, podendo ser realizada até mesmo no escuro.

O morcego emite um pulso de frequência muito alto (de 25 a 150kHz), que dura cerca de 8 a 10 ms, pela narina ou boca e detecta o tempo gasto para estas ondas voltarem ao ponto de origem após serem refletidas pela presa ou obstáculo. A amplitude e a taxa de emissão dos pulsos sonoros emitidos pelos morcegos variam de acordo com a estratégia de caça. Por exemplo, quando voam perto de sua presa, essa taxa de emissão de pulsos sonoros pode ser acelerada até cerca de 200 pulsos por segundo. (YANG, 2010a)

O pulso emitido poderia ser tão alto quanto 110 dB e o volume varia de mais alto, na busca pelas presas (exploração), a mais calmo quando voa em direção à presa (intensificação).

O controle da exploração e intensificação do algoritmo é realizado com a variação na amplitude, frequência e taxa do pulso sonoro de cada morcego. Os parâmetros responsáveis por isso são: o tamanho da população (η), o fator de decaimento da amplitude (α) e o fator de incremento da emissão de pulso (λ). (ANDRÉ; PARPINELLI, 2014)

Os morcegos podem detectar a distância, posição, tamanho, orientação e, até mesmo, a velocidade de deslocamento das suas presas. Assim, podem planejar sua nova posição antes de se deslocarem. A caça é feita em bando e utilizam a inteligência de enxames para precingir suas vítimas.

São inúmeras as formas de modelar os morcegos virtuais, mas todas elas seguem as regras básicas:

- Os morcegos virtuais utilizam a ecolocalização para detectar distâncias e são capazes de diferenciar alimento/presas de obstáculos do ambiente;
- Os morcegos virtuais voam aleatoriamente com velocidade v_i a partir de uma posição x_i, emitindo pulsos sonoros, com frequência (f) e amplitude (A) variadas, na busca pelo alimento;
- A amplitude da onda sonora pode variar de diversas formas, por isso são limitadas em um intervalo de $[A_{min}, A_{max}]$. O mesmo acontece para as frequências e taxas de emissão de pulsos ($[f_{min}, f_{max}]$ e $[r_{min}, r_{max}]$).

No pseudocódigo do BA, disponível em seu artigo, Yang omite alguns detalhes

como, por exemplo, o uso das equações responsáveis pela variação da amplitude e da taxa de emissão de pulsos. Portanto, a implementação descrita na seção seguinte trata-se de um modelo adaptado.

3.4.1 IMPLEMENTAÇÃO DO ALGORITMO DE MORCEGOS

No modelo utilizado nesta dissertação, cada morcego virtual representa uma partícula e, assim, uma possível solução para o problema, manipulado sob a forma de um vetor de posição [x, y, θ].

Uma população de morcegos virtuais (η) se move continuamente no espaço de busca (mapa conhecido), atualizando a frequência (f), velocidade (v) e posição ([x, y, θ]) de cada elemento de modo a buscar uma solução ótima. A cada nova iteração, cada morcego virtual é atualizado seguindo sempre a melhor solução (x_i^{t*}) encontrada pela população. Além da atualização da posição, o algoritmo faz o controle de exploração (busca global) e intensificação (busca local).

Conforme visto no capítulo 1, quando o morcego virtual está no processo de busca global, a taxa de emissão de pulsos sonoros (r) é baixa e vai aumentando exponencialmente de acordo com sua proximidade com a presa, Figura 27 a. O oposto acontece com a amplitude desses pulsos (A) que, na busca global, é alta e diminui conforme a proximidade com a presa, Figura 27 b.

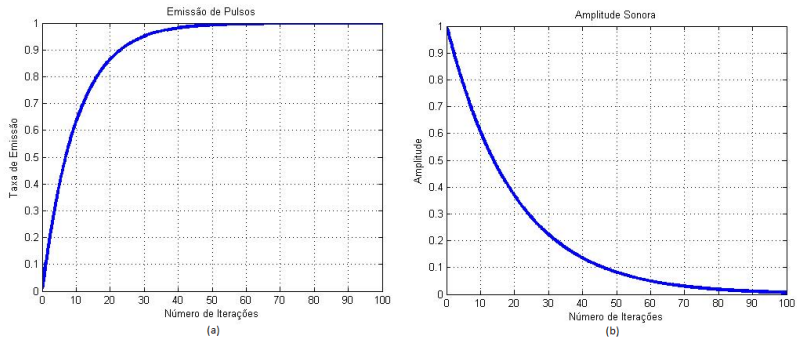

Figura 27: Comportamento da Amplitude (A) e Taxa de Emissão de Pulsos (r)

O algoritmo adaptado apresentado nesta dissertação segue a estrutura da Figura 28.

1) Inicializa o espaço de discurso: carrega o mapa
2) Inicializa os parâmetros para funcionamento do algoritmo: η, α, λ, β, v_i, f_i, A_i, r_i, TOL, $iter_{máx}$, σ_π, σ_x, σ_y, σ_θ;
3) Cálculo da FOB: $f(x)$
4) Inicialização dos Morcegos: x_i
5) Avaliação dos Morcegos: $f(x_i)$
6) Atualiza o Melhor Morcego: x_o^t
7) Enquanto ($Peso\ do\ Melhor\ Morcego > TOL\ e\ Número\ de\ Iterações < iter_{máx}$) {
8) Para $i = 1\ até\ \eta$ {
9) $f_i = f_{min} + (f_{máx} - f_{min})\beta, \beta \in [0,1]$
10) $v_i^{t+1} = v_i^t + (x_o^t - x_i^t)f_i$
11) $x_i^{temp} = x_i^t + v_i^{t+1}$
12) Se $\xi < r_i, \xi \in [0,1]$, então {
13) $x_i^{temp} = x_i^t + \varepsilon\ média(A), \varepsilon \in [-1,1]$
14) Se $\xi < A_i$ ou $f(x_i^{temp})$, então {
15) $x_i^{t^*} = x_i^{temp}$
16) $r_i^{t+1} = 1 - e^{-\lambda t}$
17) $A_i^{t+1} = \alpha A_i^t$ }
18) Atualiza o Melhor Morcego: $x*$}}

Figura 28: Algoritmo de Morcego Adaptado

Onde:

- Linha 1: carrega o mapa desejado que será o universo de discurso (criado através de uma matriz, como, por exemplo, a da Figura 6);
- Linha 2: são passados os parâmetros de:
    1. Tamanho da população (η): número escalar que pode ser definido pelo usuário considerando que quanto maior esse número, maior parte do espaço de busca será preenchido, maior a probabilidade de encontrar a solução ótima com menos iterações e maior o tempo de processamento de dados;
    2. Fator de incremeto da frequência ($\beta \in [0,1]$): vetor aleatório originado de uma distribuição uniforme;
    3. Fator de decaimento da amplitude (α) e fator de incremento da emissão de pulso (λ): números escalares, constantes;
    4. Velocidade inicial (v_i), frequência incial (f_i), amplitude inicial (A_i) e taxa inicial de emissão de pulsos (r_i): números escalares que podem ser definidos pelo usuário de acordo com a aplicação do algoritmo e o tamanho do domínio de interesse.

5. Tolerância mínima de parada ou tolerância máxima de solução ótima (TOL): número escalar definido pelo usuário também de acordo com o domínio de interesse;
6. Número de iterações máxima ($iter_{max}$): número escalar definido pelo usuário de acordo com a aplicação do algoritmo.
7. Desvio Padrão do Sensor: σ_π
8. Desvio Padrão de reamostragem das partículas em x (σ_x), y (σ_y) e θ (σ_θ) e desvio padrão do sensor (σ_θ): números escalares definidos pelo usuário de acordo com o domínio de interesse;

- Linha 3: cria uma função objetivo que, no caso desta dissertação, é apresentada pela Equação 3.8. Essa função tem como parâmetro de entrada as leituras (distâncias) dos lasers do robô e dos morcegos virtuais, e retorna o peso de cada partícula;
- Linha 4: inicializa a população de morcegos virtuais aleatoriamente $x_i = [x, y, \theta]$ (com i variando de 1 até η);
- Linha 5: faz a avaliação de cada morcego virtual através da função objetivo (FOB), Equação 3.8;
- Linha 6: atualiza a variável "melhor morcego" ($Morcego_{Melhor}$) com os dados do morcego virtual que apresentou o maior valor para a FOB (Equação 3.8), na etapa anterior;
- Linha 7: enquanto o peso do melhor morcego for maior que uma tolerância mínima de parada (TOL) e o número de iterações do algoritmo for inferior ao número de iterações máxima ($iter_{max}$), faça os passos seguintes;
- Linha 8: para cada um dos morcegos virtuais, faça os passos seguintes;
- Linha 9: ajusta a frequência (f) dos pulsos aleatoriamente através da Equação 3.21;

$$f_i = f_{min} + (f_{max} - f_{min})\beta \tag{3.21}$$

- Linha 10: calcula a velocidade (v) para a predição da posição do morcego virtual através da Equação 3.22 para um instante de tempo "t". Onde x_0 é a melhor localização global atual (melhor morcego atual);

$$v_i^t = v_i^{t-1} + (x_0^t - x_i^t) f_i \tag{3.22}$$

- Linha 11: o morcego virtual faz uma predição de sua nova posição, sem se movimentar, através da Equação 3.23 para um instante de tempo "t".

$$x_i^t = x_i^{t-1} + (v_i^t) \tag{3.23}$$

- Linha 12: se a taxa de pulsos emitidos pelo morcego virtual $r \in [0,1]$ for adequada, então a presa foi encontrada;

- Linha13: uma solução é selecionada entre as melhores atuais, a nova posição de cada morcego virtual é localmente gerada, da mesma forma que acontece com o PF (descrito pela Linha 11 do seu pseudocódigo), considera-se um desvio padrão em torno da pose da partícula original, em x (σ_x), y (σ_y) e θ (σ_θ), aumentando as chances de sucesso (Equações 3.13, 3.14 e 3.15).

  Essas posições sofrem impactos da amplitude, o que pode ser visto pela Equação 3.24. Onde $\epsilon \in [-1,1]$ é um número aleatório e A_{media} é a média das amplitudes dos sons de todos os morcegos nesse instante de tempo (Equação 3.25);

$$x_{novo} = x_{ant} + \epsilon A_{media} \tag{3.24}$$

$$A_{media} = \frac{\sum A}{\eta} \tag{3.25}$$

- Linha 14: se a amplitude (A) dos pulsos emitidos for adequada;

- Linha 15: o morcego virtual efetivamente se move para a posição predita, através da Equação 3.26;

$$x_{ant} = x_{novo} \tag{3.26}$$

- Linha 16: aumenta a taxa de emissão de pulsos dos morcegos virtuais através da Equação 3.27;

$$r_i^{t+1} = r_i^0[1 - \exp^{-\lambda t}] \tag{3.27}$$

- Linha 17: diminui a amplitude dos pulsos sonoros através da Equação 3.28;

$$A_i^{t+1} = \alpha A_i^t \tag{3.28}$$

- Linha 18: atualiza a variável "melhor morcego" com a pose do melhor encontrado através da Equação 3.29;

$$Morcego_{Melhor} = x* \tag{3.29}$$

Essa rotina descreve o comportamento do algoritmo e é a partir da linha 5 que começa, efetivamente, o processo de busca bioinspirado. Para isso, atualiza-se a frequência de cada morcego virtual e essa mesma é usada para calcular a nova velocidade e, consequentemente, a nova posição temporária.

Essa posição temporária também passa pela etapa de avaliação dos morcegos virtuais e é atribuído, a cada um deles, um novo peso. Se esse peso for maior que o do melhor morcego, a variável "melhor morcego" recebe a posição equivalente. Ou seja, a posição temporária é função da melhor posição encontrada até o instante de tempo "t". Após determinada a posição temporária, parte-se para a etapa de busca local.

Aqui, uma componente aleatória é inserida podendo ser usada tanto para exploração quanto para intensificação, dependendo do tamanho do passo. Mas outra maneira é sugerida por Yang (YANG, 2010b), que é a utilização de um operador de mutação não uniforme, sendo esta última a utilizada nesta dissertação. Assim, o valor da posição temporária é atualizada pela busca local sem considerar o valor da velocidade e da posição anterior.

Conhecida a solução temporária, obtida pela atualização da posição e velocidade ou pelo processo de busca local, surge a dúvida se esta solução deve ser aceita ou não. Se a condição da linha 14 for satisfeita, a solução temporária é aceita. Como a posição do melhor morcego virtual foi alterada, a estratégia de caça precisa mudar e, consequentemente, os parâmetros de amplitude e taxa de pulso.

Portanto, acontece um aumento da taxa de pulso, sendo que para "t" tendendo ao infinito, a taxa de emissão de pulsos tende ao valor unitário. Ou seja, com o passar do tempo, a busca local se intensifica. Outro parâmetro atualizado é a amplitude, onde a mesma decresce através de uma taxa de diminuição. A variação da amplitude acontece da seguinte forma: para valores altos de amplitude, tem-se uma probabilidade maior de aceitar novas soluções. Para valores de amplitudes baixos, uma solução de qualidade ruim é raramente aceita.

O funcionamento do Algoritmo de Morcegos e sua convergência para o melhor global pode ser visto pela sequência de capturas de tela do ambiente simulado, Figura 29. Nessa simulação, o robô é representado pela seta branca e seus feixes de *laser* pelas retas azuis, enquanto os morcegos são apresentados em azul e os *lasers* do melhor morcego são representados pelos pontos vermelhos. Pode-se notar que quando a convergência acontece, o ótimo global apresenta as leituras muito semelhates às do robô real, cumprindo assim o objetivo do algoritmo de busca.

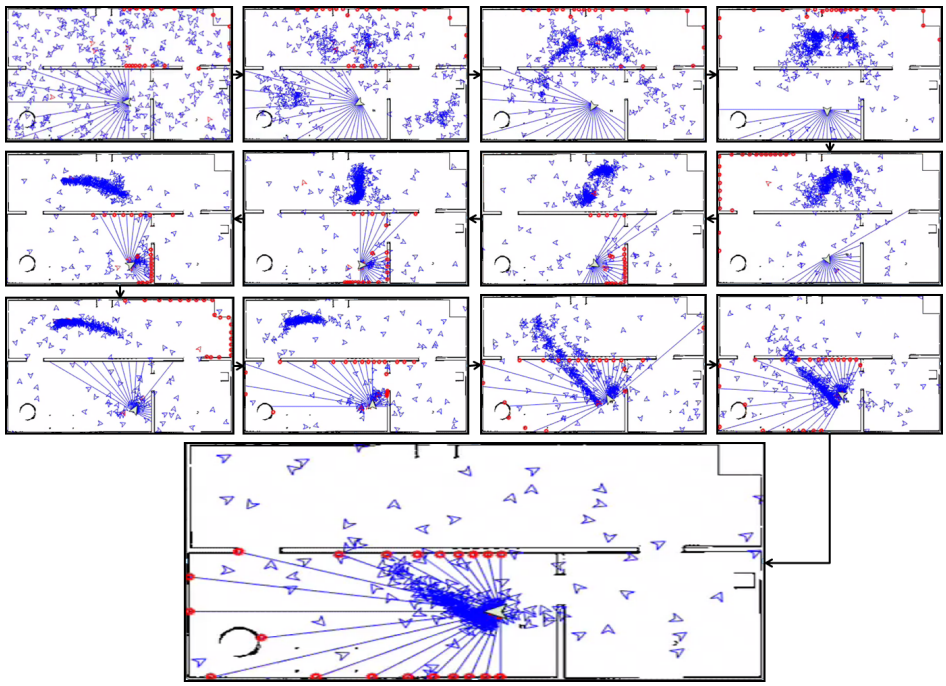

Figura 29: Capturas de Tela do Funcionamento do BA

3.5 AMBIENTE PRÁTICO

Para que os testes práticos fossem realizados, dois tipos de mapas foram criados, um em sala de aula e o outro em um corredor da universidade. Eles foram montados com placas de MDF (*Medium Density Fiberboard*), como mostra a Figura 30 e encaixes metálicos em forma de "I" e "L", Figura 31.

Figura 30: Peça de MDF para Teste Prático

Figura 31: Peças Metálicas de Encaixe para Teste Prático

Esses ambientes também precisam ser criados no programa MATLAB para que os algoritmos consigam processar suas informações, dessa forma foram criados:

1. Mapa de teste prático 1, Figura 33: primeiro mapa físico construído, a partir da Figura 32, para dar início aos testes práticos, mapa pequeno e provisório;

Figura 32: Mapa Real de Teste - Corredor

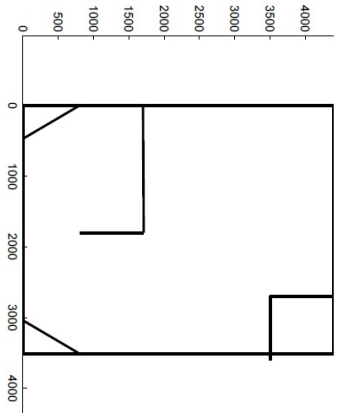

Figura 33: Mapa Simulado de Teste - Corredor

2. Mapa de teste prático 2, Figura 35: construído a partir do mapa apresentado pela Figura 34. Após a construção desse mapa, todos os testes foram realizados no mesmo, pois tinha espaço e obstáculos suficientes para que o robô realizasse trajetórias diversas e apresentasse resultados satisfatórios para análises.

Figura 34: Mapa Real de Teste - Sala

Figura 35: Mapa Simulado de Teste - Sala

Para o ambiente prático, além dos mapas construídos, Figuras 32 e 35, é necessária uma configuração do robô em um computador, através da plataforma de desenvolvimento ROS, que estará operando-o (ligando e comunicando-se com seus sensores, habilitando rede, entre outros), além da sua configuração no ambiente de desenvolvimento dos algoritmos, que no caso desta dissertação é o programa $MATLAB^{®}$, tornando possível a comunicação entre eles. Essas configurações serão apresentadas pelo capítulo A.

4 RESULTADOS

O objetivo principal deste trabalho é fazer a localização do robô móvel terrestre, dentro de um ambiente bidimensional conhecido, através de uma técnica recente de Otimização Bioinspirada (BA), não encontrada na literatura, que está sendo confrontada com o *Particle Filter* (PF).

Os testes foram realizados em dois ambientes:

1. Ambiente Simulado, Figura 36: criado no programa $MATLAB^{®}$ através de algoritmos desenvolvidos para a aplicação proposta;

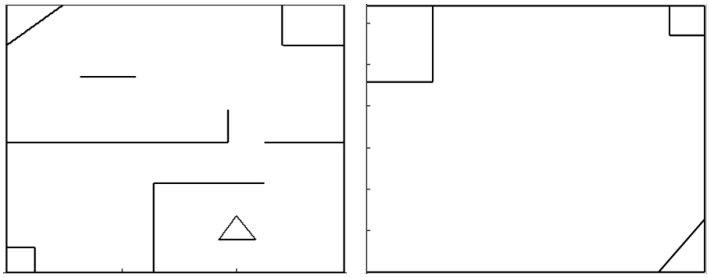

Figura 36: Ambientes Simulados

2. Ambiente Prático, Figura 37: criado em um corredor (mapa 1) e em uma sala de aula (mapa 2), com placas de MDF, Figura 30, e encaixes metálicos, Figura 31.

(a) Corredor (b) Sala de Aula

Figura 37: Ambientes Práticos

Para que os algoritmos fossem competitivos, os parâmetros comuns foram manipulados de forma igual:

- Os testes foram realizados com uma população (η) de 1500 partículas espalhadas aleatoriamente e de maneira uniforme por todo ambiente simulado. As partículas se apresentam na forma de um vetor de posição e orientação $[x, y, \theta]$. É importante ressaltar que a primeira pose das partículas é a mesma para os dois algoritmos, pois essas localizações podem influenciar no desempenho dos métodos apresentados;

- A condição de parada do algoritmo é que o peso da partícula seja inferior à tolerância mínima ($TOL = 0{,}7$) ou que o número máximo de iterações ($iter_{max} = 15$) seja atingido.

  A tolerância mínima de parada representa o limite aceitável de erro entre a pose da melhor partícula e a pose do robô real. Em (YANG, 2010a), Yang propõe uma condição de parada de 10^{-5}, equivalente a um valor de distância entre as poses; porém, esse parâmetro pode ser manipulado de diversas formas. Nesta dissertação, a diferença entre a pose das partículas e a pose do robô real é avaliada através da FOB.

- A FOB responsável pela avaliação das partículas e atribuição dos pesos das mesmas tem característica Gaussiana e é apresentada pela Equação 4.1, como citado no capítulo 3;

$$FOB(X) = \frac{1}{\sqrt{2\pi(\sigma_\pi)^2}} e^{-\frac{(part_{dist} - robo_{dist})^2}{2\sigma_\pi^2}} \tag{4.1}$$

- O caminho percorrido pelo robô real autônomo é controlado por um controlador PID sintonizado com o método Twiddle (YANG, 2010b). Os parâmetros usados por esse controlador são:

  - $K_P = 2.092$;
  - $K_I = 0.097$;
  - $K_D = 0.3081$.

- Desvio Padrão do Sensor, $\sigma_\pi = 300mm$.

- Desvio Padrão de Reamostragem em x, $\sigma_x = 300$ mm;

- Desvio Padrão de Reamostragem em y, $\sigma_y = 300$ mm;

- Desvio Padrão de Reamostragem em θ, $\sigma_\theta = 3°$.

O método de busca do *Particle Filter* é considerado não-paramétrico, dependendo, portanto, apenas da população de partículas (η) para realizar a localização. Enquanto o método de busca bioinspirado, detalhado no capítulo 3, utiliza os seguintes parâmetros para realizar, efetivamente, a busca global e local:

- Fator de Incremeto da Frequência, $\beta \in [0,1]$ (vetor aleatório);
- Fator de Incremento da Emissão de Pulsos, $\lambda = 0.05$;
- Fator de Decaimento da Amplitude, $\alpha = 0.95$;
- Amplitude Inicial, $A_i = 0.9$;
- Frequência Incial, $f_i = 0$;
- Taxa de Emissão de Pulsos Sonoros, $r_i = 0.1$;
- Velocidade Inicial $v_i = 0$;

Os valores dessas variávies são sugeridos por Yang (2010a), mas devem ser ajustados ao problema em questão, dependendo, principalmente, do ambiente de discurso. Como os testes foram realizados em abientes diferentes (prático e simulado), os resultados também precisam ser analisados separadamente.

4.1 RESULTADOS DO AMBIENTE SIMULADO

Em um ambiente simulado, os erros de hodometria são inexistentes, uma vez que esses erros são originados de fatores externos ou físicos do próprio robô móvel terrestre. O que torna possível a geração dos resultados seguintes que sobrepoem trajetórias feitas pelo robô, conseguidas pelos parâmetros de hodometria, e pelos algoritmos de localização. A intenção destes testes, é comprovar a eficácia dos algoritmos de localização para, então, realizar os testes práticos.

Em uma trajetória simples, pode-se observar que os dois algoritmos apresentam resultados satisfatórios, Figura 38, pois mostram-se confiáveis em suas localizações, uma vez que não têm acesso à hodometria do robô real (representada pelo traço azul), realizando o processo de busca de forma independente através dos dados das leituras dos sensores (representadas pelo pontilhado vermelho).

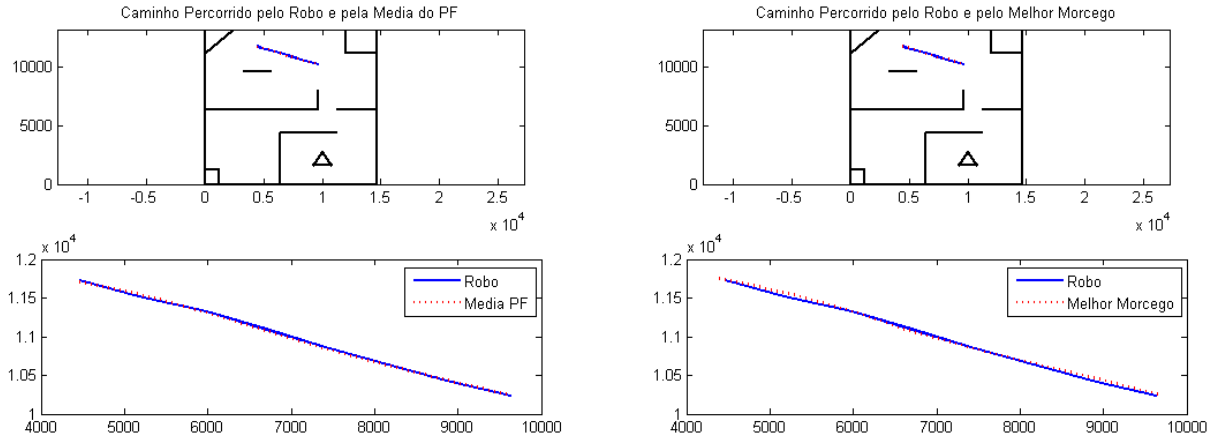

Figura 38: Resultado para Trajetória Simples

A primeira localização dos algoritmos não é delimitada pela ROI, o que torna mais difícil essa localização. Porém, os algoritmos se mostraram eficazes e convergiram para a solução ótima com as características apresentadas pela Figura 39.

```
-------------------------------------------------
------------ BAT SEARCH ALGORITHM ---------------
-------------------------------------------------

Iteracao: 0. O melhor peso: 0.0073.

A melhor particula encontrada:
            x: 4390.10
            y: 11747.47
           th: -16.37
    distancias: [2219.46 2191.17 7400.26 10724.71 6164.32 2105.15 1513.91]
            w: 0.01

Numero total de particulas: 1200
Numero total de iteracoes: 0
Melhor peso: 0.0073.
Erro linear: 76.14 milimetros.
Erro angular: 0.02 graus.
Tempo do algoritmo: 0.0000 segundos
```

```
-------------------------------------------------
---------- PARTICLES FILTER ALGORITHM ----------
-------------------------------------------------

Iteracao: 0. Erro linear: 12.0219 mm. Erro angular: -15.9253 graus.
Iteracao: 1. Erro linear: 22.2203 mm. Erro angular: -0.6427 graus.

A melhor particula encontrada:
            x: 4462.92
            y: 11705.62
           th: -15.70
    distances: [2168.56 2154.34 7384.47 10613.23 6051.68 2139.81 1552.32]
            w: 0.01

Numero total de particulas: 1200
Numero total de iteracoes: 1
Erro linear: 22.22 milimetros.
Erro angular: -0.64 graus.
Tempo do algoritmo: 0.4503 segundos
```

Figura 39: Dados da Primeira Localização

Em uma trajetória pouco mais elaborada, Figura 40, os algoritmos apresentaram resultados diferentes, possibilitando uma análise sobre o desempenho de cada método. Para isso, foi feito um teste com tamanho de população igual a 100 ($\eta = 100$).

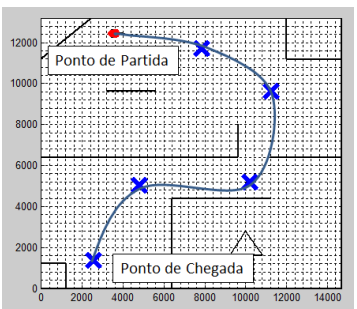

Figura 40: Trajetória a Ser Percorrida Pelo Robô

Com apenas 100 partículas e as demais configurações iniciais apresentadas na introdução deste capítulo, os resultados obtidos pelos métodos de busca, foram:

- O BA é capaz de convergir de forma ótima, Figura 41, com uma média de 2s e 2 iterações por localização. Apresenta, em média, um erro linear de 30mm e um erro angular de 4º.

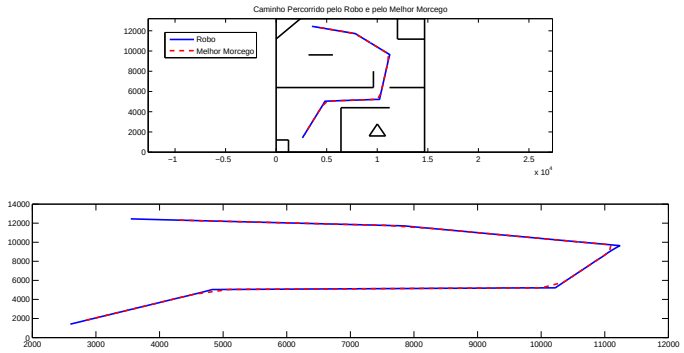

Figura 41: Localizações Realizadas pelo BA - $\eta = 100$

- O PF não consegue convergir, em algumas localizações, com um limite de 15 iterações, Figura 42. Quando converge, apresenta uma média de 8s e 12 iterações por localização, além de uma média de erro linear de 600mm e erro angular de 30º.

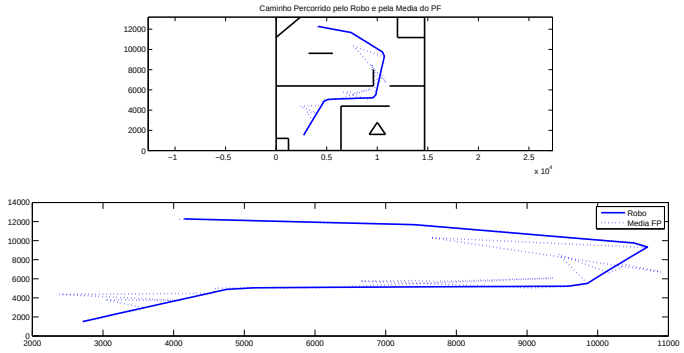

Figura 42: Localizações Realizadas pelo PF - $\eta = 100$

Para que o algoritmo do PF apresentasse um resultado semelhante ao do BA com apenas 100 partículas, foi necessário uma população de 2000 partículas, Figura 43. Com esse aumento na população, o PF é capaz de convergir de forma ótima com uma média de 6s e 5 iterações por localização. Apresenta, em média, um erro linear de 90mm e um erro angular de 7º.

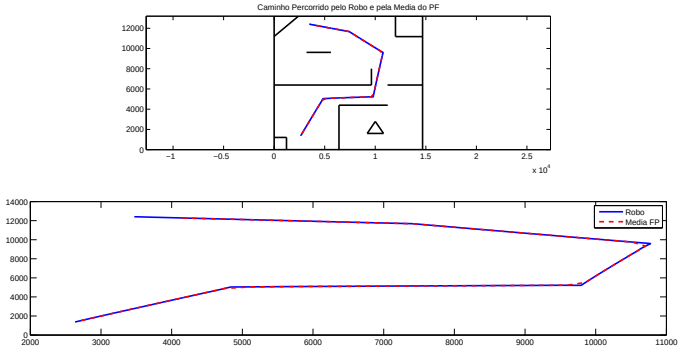

Figura 43: Localizações Realizadas pelo PF - $\eta = 2000$

Com os resultados acima, é possível confirmar que o BA se torna competitivo na área de localização de robôs móveis. O algoritmo faz a movimentação dos morcegos virtuais de forma a explorar o ambiente de discurso. Sua estratégia é capaz de encontrar a solução ótima muito mais rápido, fazendo com que o número de iterações e o tempo de convergência sejam muito menores. Além disso, apresentam erros lineares e angulares (entre a pose do robô real e a localizada pelo algoritmo) com valores baixos.

O desempenho da metodologia do PF depende, diretamente, do valor de sua população para explorar o ambiente de discurso, uma vez que as partículas só se movimentam quando o robô real também movimenta. Por isso, é necessário um número muito maior de partículas para que o algoritmo apresente resultados ótimos.

Vários fatores influenciam no desempenho dos algoritmos como, por exemplo, a pose de nascimento das partículas, o que pode favorecer algum dos métodos. Por isso, os resultados não devem ser analisados para um caso individualmente. Para tornar essa comparação justa, o algoritmo principal foi executado 1000 vezes, com as configurações iniciais apresentadas no início deste capítulo, gerando os histogramas apresentados pelas Figuras 44, 45, 46 e 47.

Esses histogramas foram gerados com o intuito de comparar o número de iterações, Figuras 44 e 45, e o tempo de convergência, Figuras 46 e 47, necessários para encontrar a solução ótima em cada localização, tanto para o algoritmo de otimização bioinspirada, BA, quanto para o PF. Assim, os histogramas permitem uma análise numérica do desempenho de cada um desses algoritmos.

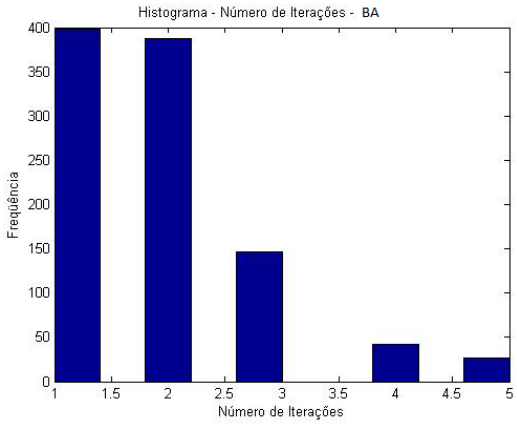

Figura 44: Histograma do Número de Iterações da Convergência do BA

Observa-se que o BA convergiu por 400 vezes na primeira iteração e não precisou de mais que 5 iterações durante os 1000 experimentos para encontrar uma solução ótima.

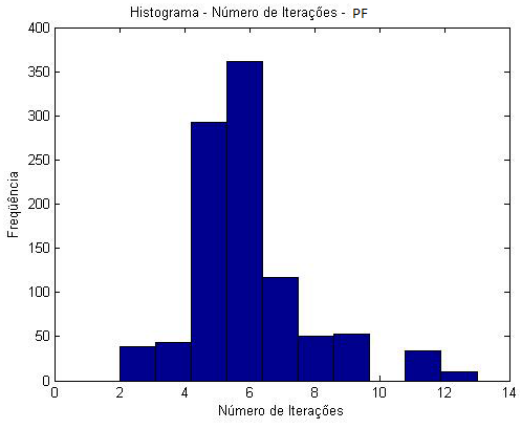

Figura 45: Histograma do Número de Iterações da Convergência do PF

O *Particle Filter* não convergiu nenhuma vez na primeira iteração, requerendo 13 iterações máximas durante os experimentos para encontrar a solução ótima.

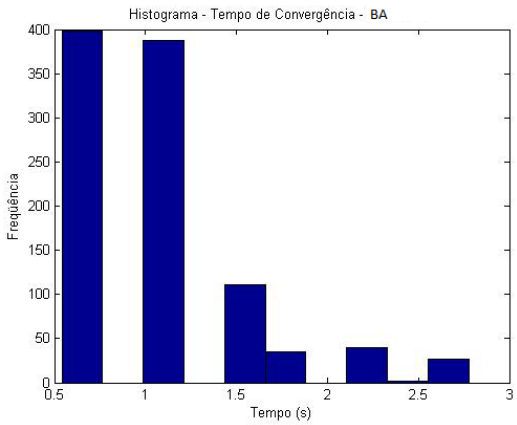

Figura 46: Histograma do Tempo de Convergência do BA

Em relação ao tempo de convergência, o BA convergiu por 400 vezes entre 0,5 s e 1 s e não precisou de mais que 3 s para encontrar uma solução ótima durante os 1000 experimentos.

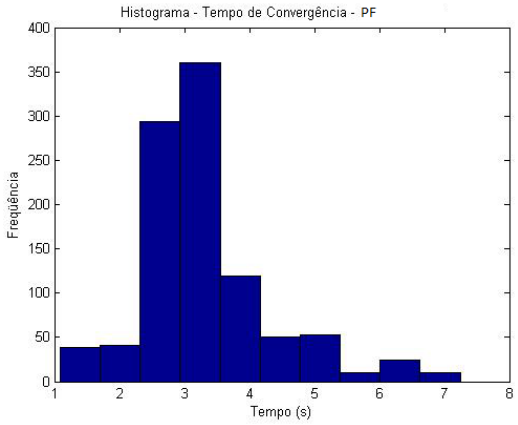

Figura 47: Histograma do Tempo de Convergência do PF

O PF não convergiu nenhuma vez antes de 1 s e precisou de até 7 s para que encontrasse a solução ótima durante os 1000 experimentos. Esses resultados confirmam as análises anteriores de que o BA se mostrou confiável e eficaz para essa aplicação, além de apresentar soluções ótimas com qualidade e baixo tempo de processamento.

4.2 RESULTADOS DO AMBIENTE PRÁTICO

Como citado anteriormente, os testes práticos foram realizados em dois ambientes: mapa 1 e mapa 2, Figura 37. Os limites e obstáculos desses mapas são as paredes do próprio lugar ou placas de MDF de 90 cm de largura e 60 cm de altura, acopladas com encaixes metálicos.

Nestes testes, os erros de hodometria influenciam nos valores de posição apresentados pelo robô, por isso, a trajetória definida pelo usuário através dos pontos inicial, final e *waypoints* será considerada como a solução ideal do problema (traçada em vermelho nas figuras de resultados).

Admitindo a mesma configuração dos parâmetros (população, desvios padrões, entre outros) citada na introdução deste capítulo, os resultados para os algoritmos de localização, *Bat Algorithm* e *Particle Filter*, realizados no mapa 1, são apresentados nas Figuras 48 e 49. Onde a linha vermelha representa a trajetória ideal e a linha em azul, a trajetória traçada pelos algoritmos de busca.

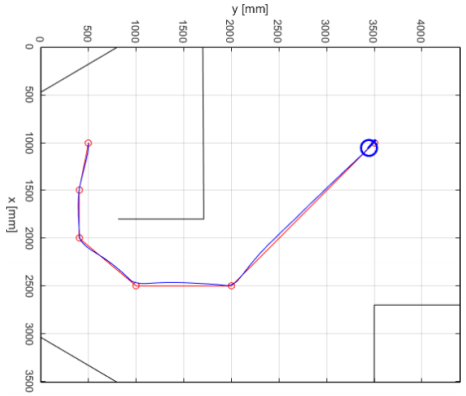

Figura 48: Resultado do BA - Mapa 1

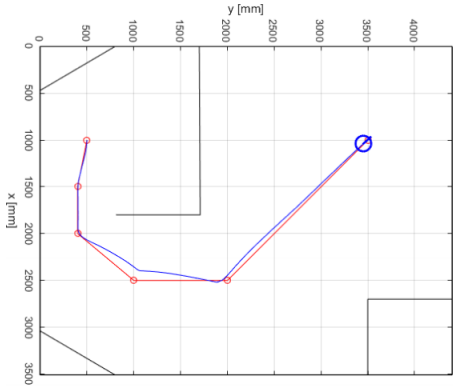

Figura 49: Resultado do PF - Mapa 1

Neste teste, realizado no mapa 1, é possível notar que os dois algoritmos apresentam bons resultados quanto às localizações efetuadas. Esse mapa não dispõe de espaço suficiente para que o robô real possa explorar e, assim como no primeiro teste do ambiente simulado, realiza uma trajetória muito simples. Por isso, foi necessário um segundo teste em um ambiente maior.

Os resultados para o mapa 2, construído em sala de aula (também com a configuração inicial dos parâmetros apresentada na introdução deste capítulo), estão disponíveis em um vídeo que pode ser acessado através do site:

https://www.youtube.com/watch?v=DVvPJmMGdVM

O resultado apresentado pelo método bioinspirado (BA) no vídeo, foi traçado pela Figura 50. Onde a linha em azul representa a trajetória traçada pelo algoritmo e a pontilhada, em vermelho, a trajetória ideal definida pelo usuário.

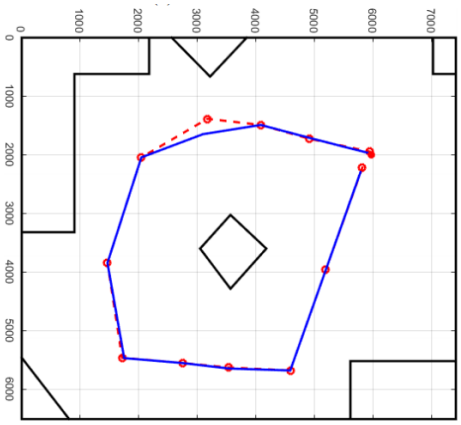

Figura 50: Resultado do BA - Mapa 2

5 CONCLUSÃO E TRABALHOS FUTUROS

5.1 CONCLUSÃO

Os resultados permitem concluir que o Algoritmo de Morcegos tem aplicabilidade na área de localização de robôs móveis e se mostra competitivo ao apresentar convergências rápidas para a solução do problema, executando-as de forma eficiente, apresentando baixos valores de erros lineares e angulares, além de possuir uma teoria de fácil compreensão e implementação.

O algoritmo bioinspirado tem uma vantagem em sua metodologia, quando comparado ao PF, acerca da movimentação das partículas. Essa movimentação acontece durante todo o processo de busca, sendo capaz de explorar o ambiente com maior eficácia. Sua estratégia de busca global e local é eficiente, o que interfere diretamente no número de iterações e tempo de convergência.

O desempenho do método *Particle Filter* depende, diretamente, do valor de sua população, a movimentação das partículas está atrelada à movimentação do robô e, assim, explora o ambiente de forma mais lenta. Por isso, necessita de um número muito maior de partículas para apresentar resultados ótimos, o que justifica valores mais altos para o número de iterações e tempo de covergência.

O espaço e material disponíveis para a construção dos ambientes práticos não permitiram um mapa com maior complexidade, limitando os testes à trajetórias mais simples. No entanto, o BA mostrou-se promissor para a aplicação. O próximo passo seria uma comparação com métodos de localização mais sofisticados e atuais como, por exemplo, o ICP (*Interative Closest Point*).

5.2 TRABALHOS FUTUROS

A utilização da ROI pode melhorar o desempenho dos algoritmos de busca limitando o espalhamento das partículas em uma área de interesse. Considerando que

essess algoritmos não podem ter informações acerca da posição do robô real, poderia ser estudada uma outra maneira de configurar essa região de interesse para melhorar, ainda mais, a eficiência dos algoritmos, sem que os mesmos perdessem sua credibilidade e autonomia.

Outra opção de trabalhos futuros seria programar o robô para desviar dos obstáculos do mapa bidimensional de forma automática. Nos testes aqui presentes, é necessário que o usuário defina uma trajetória, através da marcação de *waypoints*, para que o robô não colida com os obstáculos do ambiente. Uma solução para esse problema seria aliar o algoritmo principal ao algoritmo de campos potenciais ou explorar funções de planejamento de trajetória, que faz o cálculo de menor caminho, desviando de obstáculos e considerando uma distâcia mínima de proximidade entre o robô e esses obstáculos.

Para melhorar o desempenho do algoritmo bioinspirado, poderia ter sido feito um estudo acerca de melhores configurações para as variáveis responsáveis pelos processos de busca global e local. Para garantir o sucesso e a credibilidade do método, poderiam ter sido feitos testes práticos em mapas maiores e mais complexos (o que não foi possível devido ao limite de espaço e material), além de uma comparação com métodos de localização mais sofisticados e atuais como, por exemplo, o *Interactive Closest Point* (ICP).

Como sugestão para continuação e aprimoramento da metodologia abordada nesta dissertação, a localização do robô móvel poderia ser conseguida através da localização por visão ou até mesmo utilizar a câmera para inserir configurações de um mapa (fixo ou dinâmico) no próprio robô móvel. Para isso, seria necessário uma câmera externa capaz de detectar formas geométricas e marcos naturais, por exemplo. O detalhamento e, consequentemente, a quantidade de dados que podem ser retirados de uma informação visual faz com que sua aplicação na robótica seja de grande potencial. Além do fato dessas câmeras de vídeo serem consideradas baratas quando comparadas aos *lasers* infravermelhos.

Para que a técnica de localização por visão, sugerida acima, tenha uma ótima performance, o robô precisa ser capaz de identificar diferentes características do mapa, que podem ser classificadas em três tipos:

- *Long Term Features*: são as características de longo prazo, como paredes, pilastras e qualquer outo elemento fixo da construção daquele ambiente;
- *Short Term Features*: são as características de curto prazo, ou seja, que podem

ser alterados, como, por exemplo, móveis e itens de decoração;

- *Dynamic Term Features*: são as características dinâmicas, como, por exemplo, pessoas que podem estar passado pelo ambiente de discurso.

Se o robô móvel for capaz de detectar e selecionar as características citadas acima, ele é capaz de se localizar com eficácia e traçar a melhor rota para atingir o objetivo.

Uma última sugestão seria criar um algoritmo capaz de selecionar a melhor técnica de localização, através de uma análises de parâmetros, para a aplicação e o ambiente de discurso em questão. Assim, aumentariam as chances de sucesso para qualquer que sejam as características do meio em que o robô está inserido.

REFERÊNCIAS

ANDRÉ, L.; PARPINELLI, R. S. Tutorial sobre o uso de técnicas para controle de parâmetros em algoritmos de inteligência de enxame e computação evolutiva. *Revista de Informática Teórica e Aplicada*, v. 21, n. 2, p. 90–135, 2014.

ARÊDES, C. et al. Planejamento estático da expansão de sistemas de transmissão de energia elétrica via ecolocalização. *CBA*, 2014.

BEKEY, G. A. *Autonomous robots: from biological inspiration to implementation and control*. [S.l.]: MIT press, 2005.

BEZERRA, C. G. *Localização de um robô móvel usando odometria e marcos naturais*. Tese (Doutorado) — Universidade Federal do Rio Grande do Norte, 2004.

COLORNI, A. et al. An investigation of some properties of an "ant algorithm". In: *PPSN*. [S.l.: s.n.], 1992. v. 92, p. 509–520.

COUTO, L. N. *Sistema para localização robótica de veículos autônomos baseado em visão computacional por pontos de referência*. Tese (Doutorado) — Universidade de São Paulo, 2012.

CRAIG, J. J. *Introduction to robotics: mechanics and control*. [S.l.]: Pearson Prentice Hall Upper Saddle River, 2005.

CRUZ, S. M. *Implementação de um filtro de Kalman estendido em arquiteturas reconfiguráveis aplicado ao problema de localização de robôs móveis*. Dissertação (Mestrado), 2013.

DJEHAICH, M. et al. Slam-icp with a boolean method applied on a car-like robot. In: IEEE. *Programming and Systems (ISPS), 2013 11th International Symposium on*. [S.l.], 2013. p. 116–121.

FERNÁNDEZ, E. et al. *Learning ROS for Robotics Programming*. [S.l.]: Packt Publishing Ltd, 2015.

FURTADO, L. et al. Bat search algorithm aplicado na localizacao de robôs móveis. *Simpósio Brasileiro de Automação Inteligente*, 2015.

GARCIA, R. F. et al. Um arcabouço para a localização de enxames de robôs. *Proc. of VIII SBAI*, 2007.

KENNEDY, J. R, eberhart. *Particle swarm optimization*, IEEE Int, v. 1, 1995.

KOUBAA, A. *Robot Operating System (ROS): The Complete Reference*. [S.l.]: Springer, 2016.

KRAUSE, J.; CORDEIRO, J. A.; LOPES, H. S. Comparação de métodos de computação evolucionária para o problema da mochila multidimensional. *Meta-Heurísticas em Pesquisa Operacional*, p. 87–98, 2013.

KÜNSCH, H. R. Recursive monte carlo filters: algorithms and theoretical analysis. *Annals of Statistics*, JSTOR, p. 1983–2021, 2005.

LI, X.-l.; SHAO, Z.-j.; QIAN, J.-x. An optimizing method based on autonomous animats: fish-swarm algorithm. *System Engineering Theory and Practice*, v. 22, n. 11, p. 32–38, 2002.

LIMA, D. D. S. *Localização absoluta de robôs móveis em ambientes industriais*. Tese (Doutorado) — UNIVERSIDADE DO PORTO, 2010.

LOPES, L. S.; LAU, N.; REIS, L. P. Intelligent control and decision-making demonstrated on a simple compass-guided robot. In: IEEE. *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*. [S.l.], 2000. v. 4, p. 2419–2424.

MARTINS, J. M. F. et al. *Melhoramento do desempenho do Robot de Serviço de Limpeza: comparação de desempenho de filtros de Kalman e de filtros de partículas para auto-localização*. Dissertação (Mestrado), 2012.

Mathworks®. *Matlab*. 2014. Access date: 04/12/2015. Disponível em: <http://http://www.mathworks.com/hardware-support/robot-operating-system.html/>.

MOIVRE, A. D. *The doctrine of chances*. [S.l.]: Chelsea Publishing Company, 1967.

MURPHY, R. *An Introduction to AI Robotics (Intelligent Robotics and Autonomous Agents)*. [S.l.]: Mit Press, 2000.

NAKRANI, S.; TOVEY, C. On honey bees and dynamic server allocation in internet hosting centers. *Adaptive Behavior*, SAGE Publications, v. 12, n. 3-4, p. 223–240, 2004.

NASCIMENTO, R. C. A. do. *Localização de Robôs Móveis em Ambientes Fechados em Tempo Real Utilizando Câmeras Montadas no Teto*. Tese (Doutorado) — UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE, 2014.

NGUYEN, Q.-H. et al. A visual slam system on mobile robot supporting localization services to visually impaired people. In: SPRINGER. *European Conference on Computer Vision*. [S.l.], 2014. p. 716–729.

PERALTA, L. Introdução aos métodos de simulação monte carlo no transporte de radiação. *Faculdade de Ciências da Universidade de Lisboa. Notes available for Dosimetry and Radiological Protection classes*, 2010.

PESSIN, G. *Estratégias inteligentes aplicadas em robôs móveis autônomos e em coordenação de grupos de robôs*. Tese (Doutorado) — Universidade de São Paulo, 2013.

PINHEIRO, P. G. *Planejamento para localização de robôs móveis utilizando padrões arquitetônicos em um modelo hierárquico de POMDP*. Dissertação (Mestrado), 2013.

QUIGLEY, M.; GERKEY, B.; SMART, W. D. *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*. [S.l.]: "O'Reilly Media, Inc.", 2015.

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*, 2014.

ROY, N. et al. Coastal navigation-mobile robot navigation with uncertainty in dynamic environments. In: IEEE. *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. [S.l.], 1999. v. 1, p. 35–40.

RUSSELL, S. J. et al. *Artificial intelligence: a modern approach*. [S.l.]: Prentice hall Upper Saddle River, 2003.

SANTANA, A. M. *Localização e planejamento de caminhos para um robô humanóide e um robô escravo com rodas*. Dissertação (Mestrado), 2007.

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

THRUN, S.; BURGARD, W.; FOX, D. *Probabilistic robotics*. [S.l.]: MIT press, 2005.

VALE, A. et al. Multi-sensor navigation without an a priori map. *Acta do Encontro Cientifico do Festival Nacional de Robótica, Guimarães*, 2001.

WARDHANA, A. A. et al. Mobile robot localization using modified particle filter. In: IEEE. *Instrumentation Control and Automation (ICA), 2013 3rd International Conference on*. [S.l.], 2013. p. 161–164.

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.

YANG, X.-S. Firefly algorithm. *Nature-inspired metaheuristic algorithms*, v. 20, p. 79–90, 2008.

YANG, X.-S. *Nature-inspired metaheuristic algorithms*. [S.l.]: Luniver press, 2010.

YANG, X.-S. A new metaheuristic bat-inspired algorithm. In: *Nature inspired cooperative strategies for optimization (NICSO 2010)*. [S.l.]: Springer, 2010. p. 65–74.

APÊNDICE A – PLATAFORMA ROS - ROBOT OPERATING SYSTEM

A.1 INTRODUÇÃO

Nessa dissertação, todos os algoritmos foram desenvolvidos no programa $MATLAB^{®}$ e a comunicação com o robô móvel terrestre P3-DX foi feita por meio de uma interface de comunicação com a plataforma ROS. ROS é um *framework* de desenvolvimento livre aplicado na área da robótica, traz grande contribuição para a área e tem um crescimento considerável, pois muitos colaboradores e pesquisadores aderiram a sua utilização, bem como fabricantes de robôs, sensores e atuadores que fornecem *toolboxes* compatíveis com seus produtos.

O ROS é um sistema de código aberto, com flexibilidade de programação, permite abstração de *hardware* (não excluindo a possibilidade de acesso de baixo nível), oferece ferramentas, bibliotecas e pacotes que ajudam a simplificar a tarefa de criar rotinas, facilitando o desenvolvimento de *software* para a robótica de forma rápida e simples. Consiste em um conjunto de convenções que visam a padronização no desenvolvimento dessas rotinas, uma multiplataforma que possibilita a integração com diversos *hardwares* e *softwares*. Foi criado na *Stanford Artificial Intelligence Lab.*, por uma comunidade de colaboradores e, atualmente, é mantido pela *Open Source Robotics Foundation*. É uma plataforma consolidada em Linux, mas está em desenvolvimento em *Windows* e outros Sistemas Operacionais.

Suas principais características são a reutilização de códigos, o grande volume de algoritmos já desenvolvidos, totalmente *open source*, a liberdade para uso comercial e em pesquisa, a possibilidade de fazer o download de códigos em repositórios, melhorá-lo e compartilhar novamente.

A.2 IMPLEMENTAÇÃO

Existem várias versões do ROS. A que está sendo usada para os testes desta dissertação é a *Indigo Igloo*, lançada em julho de 2014, em um sistema operacional *Linux*

Ubuntu. A plataforma ROS é constituída por um grande número de programas independentes que funcionam comunicando-se entre si.

O primeiro passo para familiarizar-se com o ROS é realizando a sua instalação e seguindo as instruções dos tutorias disponíveis, respectivamente, em:

- http://wiki.ros.org/ROS/Installation
- http://wiki.ros.org/ROS/Tutorials

A instalação e configuração do ROS requer certa atenção, por isso, encontra-se na literatura alguns materiais de apoio como os livros Robot Operating System (ROS): *The Complete Reference* (KOUBAA, 2016), *Learning ROS for Robotics Programming* (FERNÁNDEZ et al., 2015) e *Programming Robots with ROS* (QUIGLEY; GERKEY; SMART, 2015).

O sistema de compilação ROS utilizado nessa configuração foi o *catkin*. Um sistema de compilação é um conjunto de ferramentas utilizadas pelo ROS para gerar programas executáveis, bibliotecas, *scripts* e interfaces que outros códigos poderão manipular.

Antes de escrever qualquer código ROS é necessário configurar um *workspace* (espaço de trabalho), que é uma pasta na qual pode-se modificar, construir e instalar pacotes. É possível ter vários *workspaces* ROS, mas devem ser manipulados de forma independente.

Para manipular os pacotes dessa plataforma, é necessário entender alguns conceitos básicos:

- ROS Master: é um processo único que gerencia a comunicação entre os nós, Figura 51, é definido por uma variável ambiente ROS_MASTER_URI e seu principal objetivo é permitir que um "nó" localize outro;

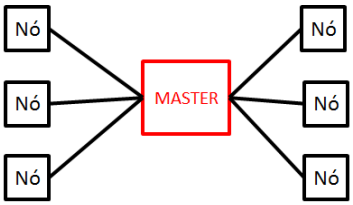

Figura 51: ROS Master

- Nó: são processos que, geralmente, controlam uma única funcionalidade do sistema, como a leitura do *laser* ou o controle da velocidade das rodas. Os nós podem se comunicar através de tópicos e serviços;

- Mensagens: é a forma pela qual os "nós" se comunicam, definidas por arquivos texto. Existem várias mensagens já definidas, mas elas também podem ser criadas pelo próprio usuário;

- Tópicos: meio pelo qual a mensagem irá trafegar;

- Serviços: é um mecanismo pelo qual um "nó" envia uma requisição para outro e recebe uma resposta à chamada;

- Parâmetros: servem para a configuração dos "nós" e para a sua inicialização. Através deles é possível mudar o comportamento de um "nó" sem precisar recompilar o código. Os parâmetros ficam armazenados em um servidor de parâmetros;

- Bags: são criados para salvar e reproduzir informações de mensagens, tópicos e serviços. São úteis para testar algoritmos, visualizar o que aconteceu durante a execução de alguma operação e divulgação de resultados e exemplos;

Os *softwares* ROS estão organizados em pacotes (*packages*):

- *Packages*: é o menor nível na organização do *software* do ROS. Pode conter um "nó", bibliotecas, conjunto de dados, etc. e, por isso, possui arquivos fontes, mensagem, compilação, executáveis, etc.;

- *Manifest*: é uma descrição do pacote, onde define suas dependências;

- *Stacks*: é uma coleção de pacotes formando uma camada de nível superior.

- *Stacks Manifest*: é uma descrição do "stacks", onde define suas dependências;

Para conectar o robô P3-DX à plataforma ROS, é preciso seguir alguns passos:

1. Possibilitar a comunicação dos "nós" através do **.bashrc**, que é um *script* executado na abertura de cada nova seção do terminal. É nesse *script* que define-se o "master" e os "nós" que vão se comunicar através de uma porta (11311). Para isso, basta abrir um terminal, pressionado as teclas **CTRL+ALT+T** na tela principal do Linux, em seguida, abrir o .bashrc, digitando **.bashrc** no terminal e acrescentar as linhas de comando:

    - Quando um único computador executa todos os nós:
      export ROS_MASTER_URI=http://(ip_PC1):11311
      export ROS_IP=(ip_PC1)

- Quando um computador executa o nó principal (*master*) e outro computador executa os demais nós:

  export ROS_MASTER_URI=http://(ip_PC1):11311

  export ROS_IP=(ip_PC2)

Onde (ip_PC1) representa o IP (*Internet Protocol*) do computador 1, assim como ip_PC2, o IP do computador 2.

2. Executar o **roscore**. *Roscore* é uma coleção de "nós" e programas que são pré-requisitos de um sistema ROS. É necessário ter um roscore em execução para que os "nós" possam se comunicar. Para isso, basta digitar **roscore** em um novo terminal linux.

3. Dar permissão para a porta USB (*Universal Serial Bus*), onde está conectado o robô P3-DX, através da linha de comando: **sudo chmod 777 /dev/ttyUSB***, onde:

   (a) **sudo** é o comando de super usuário que pode requerer uma senha;

   (b) **chmod** é o comando de mudança de permissões;

   (c) **777** valor referente a um tipo de permissão que dá autonomia de escrita, leitura, entre outros, para a porta em questão.

   (d) **/dev/ttyUSB0** endereço da porta onde está conectado o dispositivo (que pode variar dependendo da porta).

4. Dar permissão para a porta USB, onde está conectado o laser SICK LMS acoplado ao robô, através da linha de comando: **sudo chmod 777 /dev/ttyUSB1**, que segue a mesma configuração do item anterior;

Nesse ponto, o robô está ligado, fisicamente, ao ROS, mas ainda é necessário executar os nós responsáveis pela manipulação dos dados:

1. RosAria é um nó disponível na plataforma ROS, da biblioteca de código aberto "ARIA", fornece uma interface ROS para a maioria dos *Adept MobileRobots*, incluindo o Pioneer P3-DX. Permite visualizar as informações básicas, velocidade e controle de aceleração do robô. O nó publica os dados recebidos do controlador do robô em tópicos e também permite alterá-los publicando nesses tópicos;

Para rodar esse nó, é necessário conhecer o comado **rosrun** do pacote "rosbash" (contém comandos úteis e básicos da ROS). Esse comando executa arquivos de um pacote ROS, através da linha de comando: **rosrun rosaria RosAria**, onde:

(a)O primeiro argumento especifica o nome do pacote, "rosaria";

(b)O segundo argumento especifica o nome do nó, "Rosaria".

2.sicklms é um nó disponível na plataforma ROS, da biblioteca de código aberto "sicktoolbox", fornece uma interface ROS para os *lasers* SICK LMS2xx. Permite ao sensor publicar dados de distâncias entre o robô e os obstáculos através de um tópico e é executado pela linha de comando: **rosrun sicktoolbox_wrapper sicklms**.

Para confirmar essa comunicação com o robô móvel, um comando que poderia ser usado é o "rostopic list" que exibe uma lista dos tópicos atuais. "Rostopic" é um pacote de comandos que exibe informações sobre tópicos ROS, incluindo *publishers* (publicadores), *subscribers* (leitores/assinantes), taxa de publicação e mensagens ROS.

Pode-se perceber que existem vários tópicos relacionados ao nó RosAria para controle de velocidade, informações sobre a bateria, posição, sonar, entre outros. O tópico responsável por trasmitir as mensagens do *laser* é o "/scan".

Para visualizar as mensagens que estão sendo exibidas pelo tópico, será utilizado outro comando do pacote "rostopic", denominado **rostopic echo**. Para isso, a estrutura da linha de comando é: **rostopic echo /nome_topico**.

Para facilitar a comunicação de uma rede ROS com o $MATLAB^{®}$, foi criada, pela $Mathworks^{®}$, a *Robotics System Toolbox*™, a qual permite o acesso às funcionalidades do ROS e está disponível nas versões do MATLAB 2015a ou superior. Essa comunicação acontece de forma interativa e permite explorar as capacidades do robô e visualizar dados de sensores.

A *Robotics System Toolbox*™fornece:

1.Algoritmos e conectividade de *hardware*, para desenvolver aplicações para robôs móveis autônomos;

2.Interface entre MATLAB/$Simulink^{®}$ e a ROS, que permite testar e verificar aplicações em robôs ROS habilitados e simuladores de robôs, como o Gazebo.

3.Suporte para geração de código C++, permitindo-lhe gerar um nó do tipo ROS e implantá-lo em uma rede ROS.

4.Integração dos algoritmos para projetar e prototipar em MATLAB/*Simulink*®;

5.Exemplos de rotinas e atividades com robôs virtuais e reais.

Configurar uma rede ROS permite a comunicação entre diferentes dispositivos. Todos os dispositivos devem ser ligados à mesma rede ROS, real ou virtual, para poder trabalhar. É possível criar um mestre ROS em *MATLAB*®, ou se conectar a um mestre ROS existente, que está sendo executado em um dispositivo diferente. Para se conectar a um mestre externo, precisa saber o endereço IP ou nome do host do dispositivo. (Mathworks®, 2014)

A comunicação de dados é alcançada através do envio de mensagens usando entidades chamadas *publishers*, *subscribers* e serviços. Os *publishers* enviam dados via "tópicos", que os *subscribers* recebem, em seguida, através da rede. Enquanto os serviços utilizam clientes para solicitar informações de um servidor.

A programação é feita semelhante a qualquer outro programa em *MATLAB*®, porém as funções relacionadas ao ROS possuem algumas modificações, como por exemplo, para criar um nó *subscriber*, o comando **subscriber** equivale a **rossubscriber(nome do tópico, tipo do tópico, fila de mensagens)** no programa *MATLAB*®.

Assim, para que seja possível o desenvolvimento dos algoritmos e a manipulação de dados pelo *MATLAB*®, é preciso estabelecer uma rede ROS entre o *MATLAB*® e o sistema em que o robô está conectado. Para isso, o primeiro passo é configurar os IP's e definir o "master" e os "nós":

- Define o IP onde será executado o ROS Master:

  global rosMasterIp; rosMasterIp = 'http://ip_PC1:porta';

- Define o IP onde será executado o nó onde encontra-se o MATLAB (nó denominado Controle 3):

  global localhostIp; localhostIp = 'ip_PC2';

No ambiente do MATLAB, para dar início ao ROS, utiliza-se a linha de comando:

rosinit(rosMasterIp,'NodeHost',localhostIp,'NodeName','/Controle3');

Para facilitar a manipulação de dados, foi criado um objeto **robot** da classe **MobileRobot2**, onde várias funções são definidas para carregarem dados do robô real através da rede ROS.

global robot;

robot = MobileRobot2();

Para o funcionamento dos algoritmos dessa dissertação, é necessário receber os dados publicados pelo *laser* acoplado ao robô móvel, para isso, cria-se um nó *subscriber* requisitando essas informações, através da linha de comando:

laser_subs = rossubscriber('/scan',@robot.Callback_Laser);

Onde:

- **/scan** representa o tópico que trafega as mensagens;
- **@robot.Callback_Laser** representa uma função do objeto **robot** que armazena os dados do *laser*.

Para confirmar a comunicação do sistema, basta executar a ferramenta **rqt_graph** (tanto no terminal do Linux quanto no MATLAB), é uma ferramenta gráfica que consulta e visualiza nós e tópicos, como mostra a Figura 52.

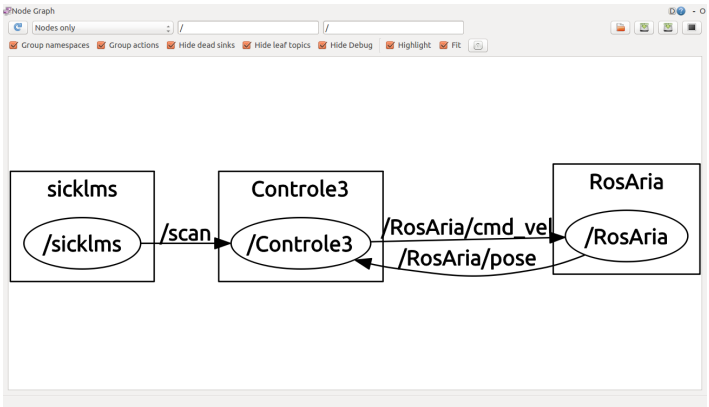

Figura 52: Nós e Tópicos do Sistema

É fácil visualizar os nós ativos (representados pelas elipses) e os tópicos criados para trafegar mensagens (representados pelas setas), confirmando a correta comunicação entre os nós do sistema ROS-MATLAB.