Research, Society and Development, v. 11, n. 6, e23211628925, 2022
(CC BY 4.0) | ISSN 2525-3409 | DOI: http://dx.doi.org/10.33448/rsd-v11i6.28925

Navegação robô móvel diferencial com AMCL utilizando o ROS

PI controller implementation for the two wheels of a differential robot using NI-MyRio

Diseño de un controlador PI para las ruedas de un robot móvil diferencial utilizando NI-MyRio

Recebido: 06/04/2022 | Revisado: 13/04/2022 | Aceito: 22/04/2022 | Publicado: 26/04/2022

Ronaldo do Amaral Oliveira
ORCID: https://orcid.org/0000-0003-4781-088X
Instituto Federal do Espírito Santo, Brasil
E-mail: ronaldo.oliveira@ifes.edu.br
Marco Antonio de Souza Leite Cuadros
ORCID: https://orcid.org/0000-0003-4191-1794
Instituto Federal do Espírito Santo, Brasil
E-mail: marcoantonio@ifes.edu.br
Carlos Torturella Valadão
ORCID: https://orcid.org/0000-0002-0765-7094
Instituto Federal do Espírito Santo, Brasil
E-mail: carlostvaladao@gmail.com

Resumo
Introdução. Este artigo propõe uma arquitetura de navegação para robôs móveis não Holonômicos em posições conhecidas no mapa de navegação. Esta arquitetura tem a capacidade de planejar um caminho entre o ponto atual e o destino. A navegação é garantida com o controlador *move_base* do *Robot Operating System* (ROS) para percorrer uma trajetória pré-determinada. Objetivo. Este artigo mostra a navegação de um robô com AMCL utilizando o ROS para fins didáticos e de desenvolvimento. Metodologia. O controle desenvolvido é compatível com o ROS e são apresentados alguns exemplos da aplicação desse sistema em um robô diferencial desenvolvido no Instituto Federal do Espírito Santo. A leitura dos *encoders*, os controladores de velocidade das rodas e do robô encontram-se num sistema embarcado da National Instruments, o NI-MyRio, programado utilizando LabVIEW. O ROS está instalado no Linux no minicomputador ODROID embarcado no robô que está conectado via Ethernet a um sensor laser LiDAR e ao NI-MyRio. A capacidade do ROS em trabalhar em ambiente de rede permite controlar e supervisionar os equipamentos através de computadores na mesma rede. Resultados. Foi possível realizar a navegação do robô móvel, fazendo-o chegar até a localização final desejada. Dentro dos experimentos, foi possível comprovar a funcionalidade do algoritmo AMCL e da arquitetura proposta. Conclusão. Por meio dos testes realizados com o robô, foi possível concluir que o objetivo de navegação foi concluído com êxito, validando o sistema e a aplicabilidade do algoritmo AMCL.
Palavras-chave: Robótica móvel; Controlador de trajetória; ROS; LabVIEW; AMCL; Ensino de robótica.

Abstract
Introduction. This article proposes a navigation architecture for non-holonomic mobile robots for known positions on the navigation map. This architecture can plan a path from the current point to the destination. Navigation is ensured by the move_base controller package of Robot Operating System (ROS) that guides the robot in the predetermined trajectory. Objectives. This article shows the navigation of a non-holonomic robot using (Adaptive Monte Carlo Localization) AMCL algorithm and ROS for educational and development purposes. Methodology. The developed control is compatible with ROS and some examples are shown using a differential robot developed at the Federal Institute of Espirito Santo. The encoders, wheel and robot speed controllers are read in an embedded NI-MyRio system, which is programmed using LabVIEW. ROS is installed on a Linux ODROID minicomputer, which is part of the robot and is connected via Ethernet to a LiDAR laser sensor and to the NI-MyRio. ROS ability to work in a network environment allows control and supervision of devices through computer network. Results. It was possible to perform the navigation of the mobile robot, making it reach the desired final location. Within the experiments, it was possible to prove the functionality of the AMCL algorithm and the proposed architecture. Conclusion. Through the tests performed with the robot, it was possible to conclude that the navigation objective was successfully completed, validating the system and the applicability of the AMCL algorithm.
Keywords: Mobile robotics; Trajectory controller; ROS; LabVIEW; AMCL; Robotics teaching.

Resumen
Introducción. Este artículo propone una arquitectura de navegación para robots móviles no-holonómicos en posiciones conocidas en el mapa de navegación. Esta arquitectura tiene la capacidad de planificar un camino entre el punto actual y el deseado. La navegación es asegurada con el controlador *move_base* del *Robot Operating System* (ROS) que garantiza que el robot va a seguir una trayectoria predeterminada. Objectivos. Este artículo muestra la navegación de

un robot con AMCL utilizando ROS para fines didácticos y de desarrollo. Metodología. El control desarrollado es compatible con ROS y algunos ejemplos de aplicación de este sistema en un robot diferencial desarrollado en el Instituto Federal do Espírito Santo son presentados. La lectura de los encoders, los controladores de velocidad de las ruedas y del robot están en un sistema embebido de National Instruments, NI My Rio, programado usando LabVIEW. El ROS está instalado en un miniordenador ODROID Linux integrado en el robot que está conectado a través de Ethernet a un sensor láser LiDAR ya NI-MyRio. La capacidad de ROS para trabajar en un entorno de red permite controlar y supervisar equipos a través de computadoras en la misma red. Resultados. Se logró realizar la navegación del robot móvil, haciéndolo llegar a la ubicación final deseada. Además, se logró probar la funcionalidad del algoritmo AMCL y de la arquitectura propuesta. Conclusión. A través de las pruebas realizadas con el robot se concluye que el objetivo de navegación se cumplió con éxito, validando el sistema y la aplicabilidad del algoritmo AMCL.
Palabras clave: Robótica móvil; Controlador de trayectoria; ROS; LabVIEW; AMCL; Enseñanza en robótica.

1. Introdução

A navegação é um dos principais temas da robótica móvel e inclui aspectos importantes como evitar obstáculos, erros de odometria, erros cumulativos em sensores, posição absoluta e relativa, correção de erros, entre outros (Faisal et al., 2013). Para permitir que o robô se desloque na trajetória correta, ele precisa conhecer sua postura com a maior precisão possível para servir de entrada para o controlador. Este último define a velocidade de cada motor para que o robô atinja a posição desejada (Zaman et al., 2011). No entanto, detectar a posição correta do robô pode ser uma tarefa difícil, especialmente quando se utiliza apenas sensores embutidos. A odometria, que é uma das formas mais utilizadas para medir a postura de um robô, tende a adicionar erros ao longo do tempo. Portanto, após alguns ciclos, a posição medida do robô está distante da real (Bezerra, 2004).

O *Robotics Operating System* (ROS) é um projeto colaborativo desenvolvido com o objetivo de facilitar o trabalho dos desenvolvedores com relação ao gerenciamento de componentes de baixo nível, fornecendo bibliotecas e ferramentas para ajudar os desenvolvedores de software a criar aplicativos robóticos (Quigley et al., 2009). Desta maneira, é possível ter abstração do hardware, drivers de dispositivos, bibliotecas, visualizadores, passagem de mensagens, gerenciamento de pacotes e muito mais. Além de todas essas funcionalidades, o ROS é um software de licença livre (*opensource*) (ROS, 2019). Desta maneira, pelo fato de a navegação ser uma tarefa imprescindível para a robótica, os pacotes ROS são ferramentas muito úteis e facilitam estas tarefas, tendo assim, uma grande importância para a robótica (Estefo et al., 2019).

O protótipo do robô terá a instrumentação necessária para realizar uma navegação autônoma, assim serão usados *encoders* em cada roda, para o controle de velocidade individual e a odometria do robô. Entretanto, algoritmos fundamentados em *encoders* não são precisos e acumulam erros; desta forma, o robô também usará um sensor laser LiDAR (*LIght Detecting And Ranging*) para o uso de algoritmos de localização como o Monte Carlo. O acesso aos dados e comandos do robô serão realizados, via Wi-Fi, usando o framework ROS, o que permitirá criar aplicações com diferentes linguagens de programação. O protótipo será para ambientes indoor e outdoor com uma estrutura robusta, além de ser de baixo custo comparado com as opções encontradas no mercado (Silva et al., 2016).

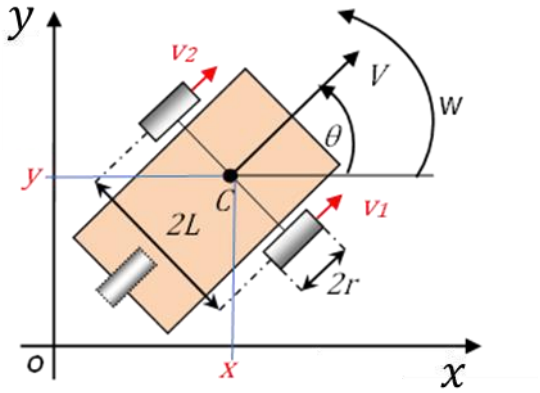
Neste trabalho serão apresentados alguns exemplos da utilização de um robô móvel por meio do ROS, como mapeamento, controle de velocidade linear, angular, aplicação do algoritmo de Monte Carlo e algoritmos de navegação. Para isto, o robô é programado no LabVIEW (Paiva et al., 2016), com controladores de velocidade linear e angular embarcado no próprio robô móvel, além de utilizar o sistema ROS.

1.1 Robôs diferenciais

A Figura 1 mostra a vista superior de um diagrama de um robô diferencial. Este tipo de modelo se comporta, em termos cinemáticos, da mesma maneira que o nosso protótipo, sendo matematicamente equivalente. Há, contudo, pequenas diferenças com relação à construção do nosso robô, as quais serão explicadas na Seção 2. A frente do robô está indicada pelo sentido do vetor velocidade V. No modelo mostrado é considerado que ambos os motores são idênticos, com raio r, e

atuadores independentes. A distância entre as rodas, também chamada de eixo virtual, é 2L.

Figura 1 - Diagrama do robô diferencial mostrando as variáveis usadas para achar o modelo do robô.



Fonte: Adaptado de (Tommasi et al., 2015).

A pose do robô é definida por sua posição no espaço e sua orientação. Matematicamente, pode ser descrita pela Equação (1):

$$q = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, \quad (1)$$

onde θ é o ângulo de orientação e (x, y) é a posição do robô dentro do plano cartesiano. O ponto C é o centro de massa do robô, e é usado como ponto de referência (Tommasi et al., 2015). Na parte posterior do robô, há uma roda castor, usada para dar um suporte complementar a toda a estrutura, sem interferir diretamente no movimento do robô.

O robô se move em uma linha reta quando ambas as rodas têm a mesma velocidade, isto é, quando $v_R = v_L$, onde v_R é a velocidade da roda direita, e v_L a velocidade da roda esquerda. Se ambas as velocidades são positivas, o robô se move para frente. Em contrapartida, se as velocidades forem negativas o robô se move para trás. Para permitir ao robô fazer curvas, as rodas direita e esquerda devem estar em velocidades diferentes. Para rotacionar a direita (no sentido horário), a velocidade da roda esquerda deve ser maior que da roda direita, isto é, $v_L > v_R$. Em contrapartida, para rotacionar a esquerda (no sentido anti-horário), é necessário que a configuração de velocidades seja oposta, isto é, a velocidade da roda direita deve ser maior que a da esquerda, ou seja, $v_R > v_L$ (Ayres et al., 2017).

Devido à natureza da sua construção, o robô diferencial possui restrições não-holonômicas, o que significa que o robô não consegue se mover lateralmente (Ben-Ari & Mondada, 2018). Para alcançar pontos na lateral, o robô deve fazer uma combinação de manobras se movendo para frente, para trás e fazendo rotações. Em resumo, isso significa que a movimentação sobre o eixo x do robô não são permitidas, já que as rodas não se movem sobre esse eixo (apenas sobre o eixo y).

O objetivo deste trabalho é apresentar uma estratégia de controle para esse robô, enfatizando os aspectos teóricos e descrevendo os passos necessários para a implementação da mesma. Além disso, são analisados os potenciais e as limitações da estratégia implementada, focando em sua aplicação para navegação autônoma e programação.

2. Metodologia

Este trabalho mostra a estrutura física e a implementação da estratégia de controle de um robô que tem como objetivo a navegação autônoma, baseado nas aplicações que foram compiladas de uma revisão sistemática da literatura (Faria & Romero, 2002; Romero et al., 2014). A metodologia utilizada baseia-se em projetar uma estrutura de um robô e utilizar um modelo matemático que se aproxime o máximo possível dessa estrutura a fim de que possam ser desenvolvidos os algoritmos de controle. Todos a pesquisa e os experimentos foram realizados no laboratório do *Grupo da Automação Industrial* (GAIn), localizado no Instituto Federal do Espírito Santo (Serra – Brasil). A pesquisa tem um caráter qualitativo e será realizada por meio do projeto e modelagem matemática de um robô, além da realização da programação para permitir que o robô seja utilizado como uma plataforma robótica educacional móvel.

Para poder implementar algoritmos de controle de trajetória será indispensável a implementação do algoritmo *dead reckoning* usando os *encoders*. Neste ponto, iniciar-se-á a implementação do ROS, o qual será instalado no ODROID, que é um minicomputador com o sistema operacional Linux. Os dados da posição serão obtidos através da odometria e dos comandos de velocidade linear e angular que serão publicados no ROS.

Com o objetivo de implementar o algoritmo de localização por Monte Carlo, é necessário configurar o LiDAR no ODROID e disponibilizar suas informações através de tópicos no ROS. Posteriormente, será selecionada uma trajetória de navegação para realizar os testes. Com o LiDAR configurado e o local selecionado, será feito um mapeamento dele e o algoritmo de Monte Carlo será parametrizado. Em cada fase do desenvolvimento serão feitos testes práticos para validar a correta configuração.

Um dos maiores desafios na robótica móvel é a navegação. Dessa forma, como todas as fases anteriormente explicadas e implementadas serão escolhidos testes de navegação para o robô móvel construído. Para a implementação destes testes será usado o pacote do ROS que controla uma navegação autônoma de um ponto inicial ao final.

2.1 Odometria

A odometria é um tipo de *dead reckoning* que permite estimar a pose do robô (coordenadas x, y e θ) através da integração das medições dos deslocamentos das rodas por meio de sensores chamados *encoders* (codificadores ópticos) (Chong & Kleeman, 1997). Os *encoders* geram pulsos no giro do eixo ao qual estão acoplados. No entanto, este método está sujeito a erros que se acumulam de acordo com o espaço percorrido pelo robô, devido aos deslizamentos das rodas, assimetrias do robô, dentre outros. Estes erros inviabilizam o uso único do *dead reckoning* para a determinação da pose de um robô (Bezerra, 2004). Neste trabalho, serão utilizados *encoders* óticos incrementais (Nemec et al., 2019).

Seguindo o modelo do robô diferencial, para calcular a posição do robô (ponto "C" – que é usado como ponto de referência do robô), conforme mostra a Figura 1, é necessário obter a distância de cada pulso do encoder, como mostrado na Equação (2).

$$DistPulsos = \frac{2\pi R}{NP}, \quad (2)$$

onde $DistPulsos$ é a distância de cada pulso, $\mathbf{R}$ é o raio da roda e $\mathbf{NP}$ é a quantidade de pulsos por volta. Desta maneira, é calculada a distância percorrida por cada roda. Nas Equações **Erro! Fonte de referência não encontrada.** e **Erro! Fonte de referência não encontrada.** encontra-se a distância percorrida pelas rodas direitas e esquerdas do robô, respectivamente,

$$\Delta S_d = DistPulsos \cdot P_d, \quad (3)$$

$$\Delta S_e = DistPulsos \cdot P_e, \quad (4)$$

onde ΔS_d e ΔS_e são as distâncias médias percorridas pelas rodas direita e esquerda, respectivamente, no intervalo Δt; enquanto P_d e P_e são as quantidades de pulsos dos *encoders* das rodas direita e esquerda. A distância percorrida pelo ponto *C*, referência do robô, é dada pela Equação (5),

$$\Delta S = \frac{\Delta S_d + \Delta S_e}{2}, \quad (5)$$

Para a variação do ângulo θ, é necessário usar a Equação (6):

$$\Delta\theta = \frac{\Delta S_d + \Delta S_e}{L_s}, \quad (6)$$

onde L_s é a distância entre as rodas. Com as variações de deslocamento angular $\Delta\theta$ e com o valor atual do ângulo θ, pode-se calcular a posição atualizada (COM'), como mostrado na Equação **Erro! Fonte de referência não encontrada.**.

$$P' = \begin{vmatrix} x' \\ y' \\ \theta' \end{vmatrix} = P + \begin{vmatrix} \Delta x = \Delta S * cos(\theta + \Delta\theta/2) \\ \Delta y = \Delta S * sin(\theta + \Delta\theta/2) \\ \Delta\theta \end{vmatrix} = \begin{vmatrix} x \\ y \\ \theta \end{vmatrix} + \begin{vmatrix} \Delta x = \Delta S * cos(\theta + \Delta\theta/2) \\ \Delta y = \Delta S * sin(\theta + \Delta\theta/2) \\ \Delta\theta \end{vmatrix} = \begin{vmatrix} x + \Delta x \\ y + \Delta y \\ \theta + \Delta\theta \end{vmatrix}. \quad (7)$$

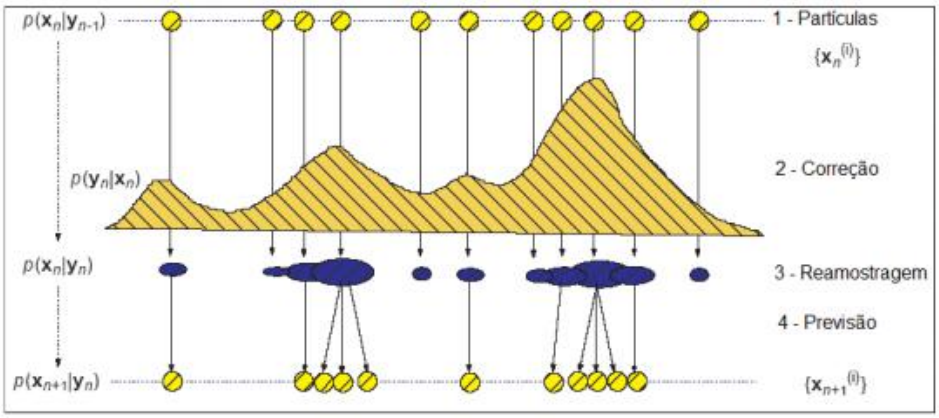
2.2 Filtro de partículas

Segundo (Thrun et al., 2005), os Filtros de Partículas (FP), no contexto da localização, são conhecidos como Monte Carlo Localization (MCL) uma vez que utilizam o método de integração sequencial de Monte Carlo para estimar a distribuição de probabilidade, criando amostras a partir de uma Função de Densidade de Probabilidade (FDP) conhecida (Fox et al., 1997, 1999; Thrun et al., 2005).

Esta distribuição de probabilidade vem da informação proveniente dos sensores fixados no robô, sendo necessário, a todo instante, um novo cálculo da FDP para cada célula do mapa (Gouveia, 2008). Para esta análise se exige um mapa fidedigno ao ambiente de navegação (Dellaert et al., 1999a).

Supondo que o robô não saiba onde ele se encontra no mapa, o filtro de partículas gera aleatoriamente K partículas de possíveis localizações reais do robô, conforme são vistas na Figura 2, onde a FDP é estimada pela ação conjunta das partículas. Cada partícula será lida por meio dos sensores e comparada com o robô real gerando uma pontuação. As partículas com os maiores valores de pontuação receberão os maiores pesos, isto é, estas partículas serão as mais relevantes, visto que o peso é diretamente proporcional ao score (Fortes, 2018; Salarolli et al., 2017).

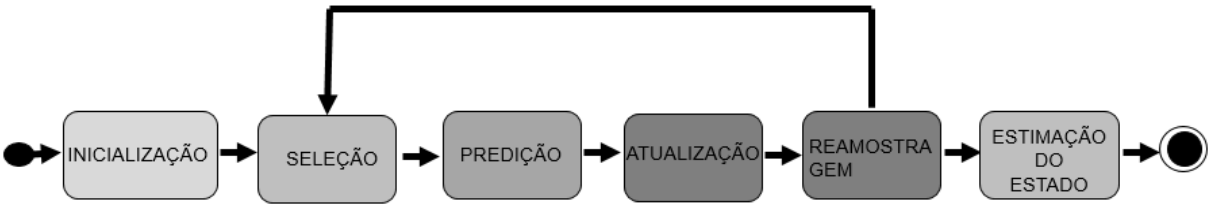
Figura 2 - Etapas do Filtro de Partículas.



Fonte: (Chen et al., 2003).

O fluxograma simplificado apresentando as etapas do FP, pode ser visto na Figura 3. Os blocos disponíveis no LabVIEW permitem um melhor planejamento relativo à programação do robô, sendo que dentro do código implementado foi feito um controlador considerando os parâmetros da Equação (7) e um PID ideal com anti-wind-up.

Figura 3 - Fluxo do Funcionamento do Filtro de Partículas.



Fonte: Fortes (2018).

As etapas do FP são:

1) Inicialização: gerar K partículas aleatoriamente no mapa. Como a posição inicial do robô não é conhecida, estima-se uma direção para todas as partículas e calcula-se o valor da FDP.

2) Seleção: as partículas com alta pontuação são mantidas e, por outro lado, as partículas com baixa pontuação são eliminadas.

3) Predição (previsão): as partículas eliminadas na etapa anterior agora são substituídas por outras, porém agora não de forma aleatória, mas sim perto das possíveis partículas candidatas ao robô real.

4) Atualização: cada partícula possui seu próprio sensor (uma réplica do sensor do robô), os sensores das partículas são independentes entre as partículas. Quando o robô se desloca para frente, por exemplo, algumas partículas ficam com pontuação baixa e são eliminadas.

5) Reamostragem: existem casos em que algumas partículas terão pesos tão elevados que irão "excluir" outras partículas, sendo, por isso, necessária a etapa de reamostragem. Nesta etapa uma porcentagem das partículas, com

peso pequeno, é substituída por outras partículas aleatórias. O restante das partículas com peso pequeno, conhecidas como partículas filhas, são geradas próximas às partículas de peso alto.

O processo recursivo faz com que o conjunto de K partículas convirjam uniformemente para a posição esperada seguindo a distribuição de probabilidade.

2.3 Robot operating system

Em 2007, Eric Berger e Keenan Wyrobek, dois estudantes de doutorado de Stanford, fizeram vários programas pequenos e os juntaram, iniciando, desta forma, o nascimento do *Robot Operating System* (ROS). A filosofia deles se baseava que um excelente desenvolvedor de software poderia não ter o conhecimento de hardware necessário, mas deveria ainda assim ser capaz de programar um robô e, com esse pensamento, surgiu o ROS (Tellez, 2019).

Em paralelo, a Willow Garage, iniciada em janeiro de 2007, era uma incubadora de tecnologia que estava trabalhando em um SUV (*Sport Utility Vehicle* - veículo utilitário esportivo) e um barco solar, ambos autônomos. O fundador da Willow Garage, Scott Hassan compartilhava da mesma ideia de Berger e Wyrobek de um "Linux para robótica" e convidou-os a trabalhar na empresa. Com isto, foi feito o primeiro comitê do código ROS no SourceForge em 7 de novembro de 2007 (Fairchild & Harman, 2017; Joseph, 2017; Tellez, 2019).

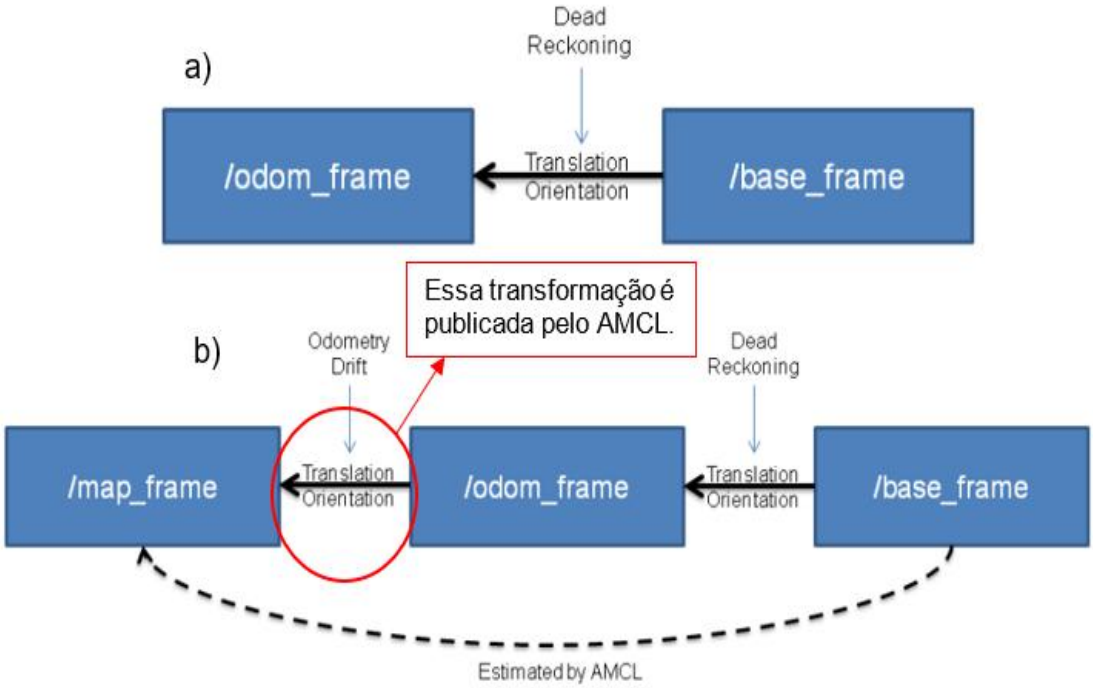
2.4 Pacote AMCL

O AMCL (*Adaptive Monte Carlo Localization)* é um sistema de localização probabilístico para um robô móvel que se movimenta em um plano. O algoritmo de Localização por Monte Carlo (*Monte Carlo Localization* - MCL) foi apresentado por (Dellaert et al., 1999), enquanto a parte adaptativa é baseada na técnica de amostragem KLD (*Kullback-Leibler Distance*), proposta por (Fox, 2003). O AMCL usa o filtro de partículas para rastrear a pose de um robô em relação a um mapa conhecido (Gamarra et al., 2019; Gerkey, 2020).

O algoritmo AMCL gera várias partículas em diferentes localizações e orientações de forma aleatória, cada uma dessas partículas é uma possível pose do robô. Para o funcionamento deste algoritmo é indispensável ter o tópico */laserscan* no ROS, visto que este tópico publica as distâncias medidas pelo sensor laser LiDAR a partir do robô até os obstáculos; assim estas medidas representam as medidas reais. As partículas do algoritmo AMCL também terão a medida dos obstáculos usando o mapa. Desta forma, o AMCL dará maior peso às partículas que tenham maior concordância com as medidas reais, e gerará outras partículas próximas às de maior peso, eliminando as de menor peso. À medida que o AMCL gera mais partículas, sua execução demandará maior processamento; assim a parte adaptativa do AMCL diminuirá a quantidade de partículas quando as partículas forem convergindo para uma pose. Pelo exposto, o AMCL também depende de um mapa, o qual será obtido do tópico /*map* (exemplo 3). Além dos tópicos /map e /laserscan, o AMCL também está subscrito aos tópicos /*initialpose* e /*tf*, necessários para seu funcionamento. O tópico /*tf* disponibiliza as transformações de rotação e translação entre os *frames* (sistemas de coordenadas) da odometria, do robô e do mapa (Meeussen, 2010; Romero et al., 2014; ROS, 2015).

A Figura 4 mostra a diferença entre a localização usando odometria e AMCL. Durante a operação, o AMCL estima a transformação do quadro base (~base_frame_id) em relação ao quadro global (~map_frame_id), mas publica apenas a transformação entre o quadro global e o quadro de odometria (~odom_frame_id). Essencialmente, essa transformação é responsável pelos erros que derivam usando o *dead reckoning*.

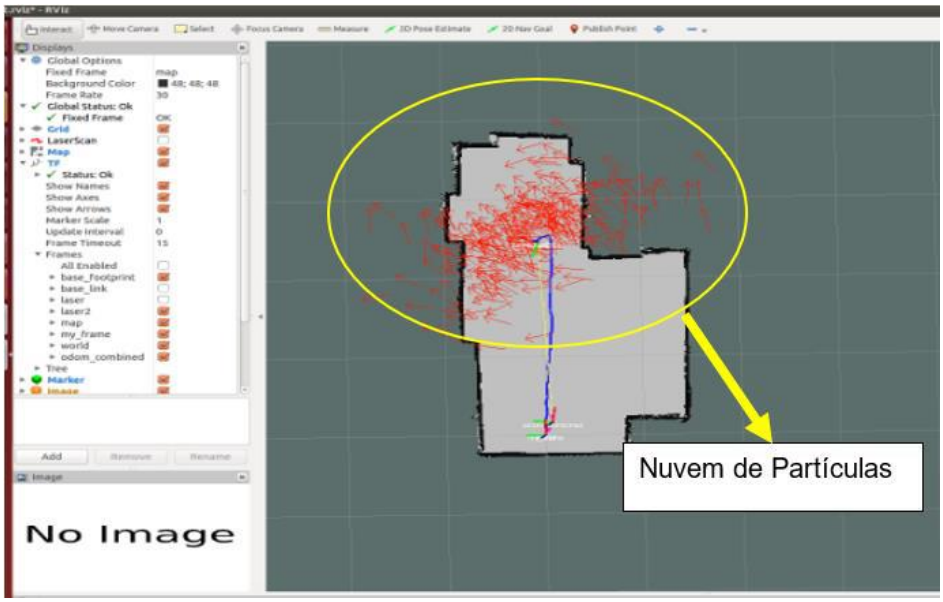
Figura 4 - Comparação entre localização usando só odometria e o AMCL. a) Localização usando só odometria, b) Localização usando AMCL.



Fonte: Gerkey (2020).

A nuvem de partículas e a utilização do AMCL é exemplificada na Figura 5, mostrando o ponto inicial e final do robô utilizando um software de visualização chamado RVIz.

Figura 5 - O AMCL no RVIz, vermelho são as partículas, azul a trajetória do robô, verde posição inicial e final do robô.

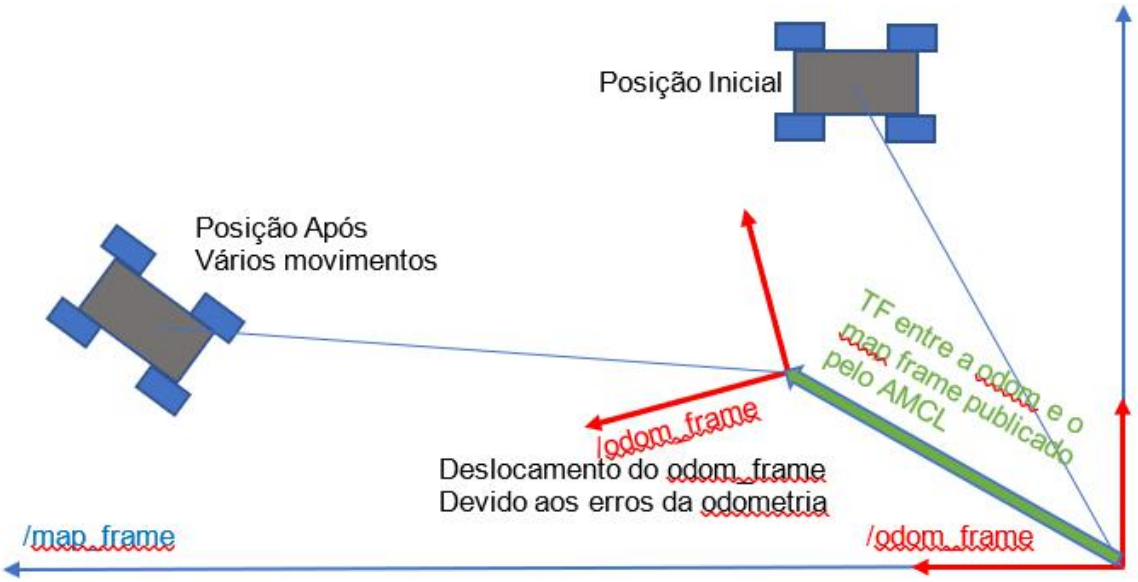


Fonte: ROS (2015).

Na Figura 6 é apresentado um exemplo do comportamento do AMCL na presença dos erros da odometria. No início do movimento os *frames* 'map' e 'odom' coincidem na mesma origem, entretanto, com o deslocamento do robô aparecem os erros

da odometria e o frame ‘odom’ é deslocado com respeito ao *frame* ‘map’. Assim, o AMCL publica a transformada de translação e orientação do *frame* ‘odom’. Desta forma, a posição do robô no *frame* global sempre será a correta, devido ao fato de que o erro será considerado.

Figura 6 - Exemplo do AMCL para o movimento de um Robô.



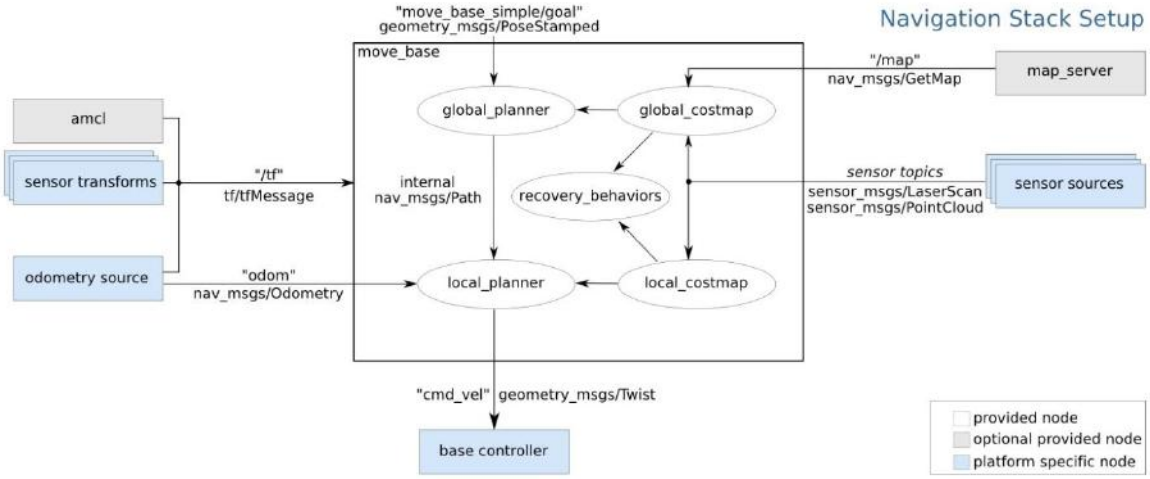
Fonte: Autores.

2.5 O pacote *move-base*

Considera-se que o robô se move autonomamente em um ambiente através de um mapa conhecido. O mapa consiste em informações, como obstruções, paredes ou tabelas (chamadas de custos em robótica), que são conhecidas por nós. Esta informação conhecida é tratada como informação global (Joseph, 2017).

O planejador global e o mapa global de custos são usados para criar planos de longo prazo em todo o ambiente, como o planejamento do caminho para o robô atingir sua meta. O planejador local e o mapa local de custos são usados principalmente para metas intermediárias e prevenção de obstáculos (Fairchild & Harman, 2017). Enquanto o robô tenta se movimentar com essas informações globais, ele pode ser introduzido a mudanças dinâmicas repentinas no ambiente, como mudanças na posição de uma cadeira ou movimento dinâmico, como uma pessoa andando. Essas mudanças são tratadas como informação local. Em termos simples, as informações globais são referentes ao ambiente e as informações locais são referentes ao robô. O *move_base* é um nó complicado (em termos de código), mas simples (em termos de entendimento) que ajuda a vincular informações globais e locais para realizar uma tarefa de navegação. O *move_base* é uma implementação de ação do ROS. Quando é determinado um objetivo, a base do robô tenta alcançar tal objetivo. Uma representação do *move_base* é mostrada na Figura 7 (Joseph, 2017):

Figura 7 - Diagrama do *move_base* com todos os tópicos.



Fonte: Joseph (2017).

Os pacotes que o *move_base* utiliza para realizar as tarefas de navegação não serão detalhados. Entretanto, mais informações podem ser vistas nas referências (Fairchild & Harman, 2017).

3. Resultados e Discussão

Nesta seção é mostrado como a metodologia aplicada permite que o sistema seja projetado com um controlador adequado, o qual demonstrou ser funcional para as tarefas propostas, sendo capaz de se mover em um caminho retilíneo.

3.1 Navegação do robô usando *move_base*

O pacote *move_base* do ROS permite que o movimento do robô seja realizado usando uma pilha de navegação. Este pacote tem o movimento nó base, que contém planejadores globais e locais, mapas de custos e um nó de comportamento de recuperação (Eppstein, 2020).

Os experimentos são executados no ROS tanto em C++ como em Python, utilizando dois métodos de localização, a odometria e o AMCL. O robô contém dois *encoders* e um laser como sensores. Além disso, o software RVIz é usado para exibir os dados do robô de forma visual e traçar o caminho, mapas de custos (do AMCL) e todos os tópicos necessários para comparar os dois métodos de localização, como o /laserscan, /map e /odom, que são, respectivamente, os dados do laser, o mapa e os dados da odometria dos *encoders*. O controlador de base de movimento é o *move_base*, que será usado em ambos os experimentos.

Foram realizados dois experimentos utilizando combinações distintas de métodos de localização. O primeiro e mais simples usa apenas a odometria; o segundo usa odometria e AMCL. A Figura 8 mostra como cada método de localização deveria se comportar no mapa, em uma forma ideal. O robô deve partir da origem no ponto A (0,0,0) e em todas as simulações ele deve passar por todos os pontos marcados, desde o ponto inicial, passando pelos pontos de B a C, antes de completar o ciclo retornando à posição de origem no ponto A (0,0,0).

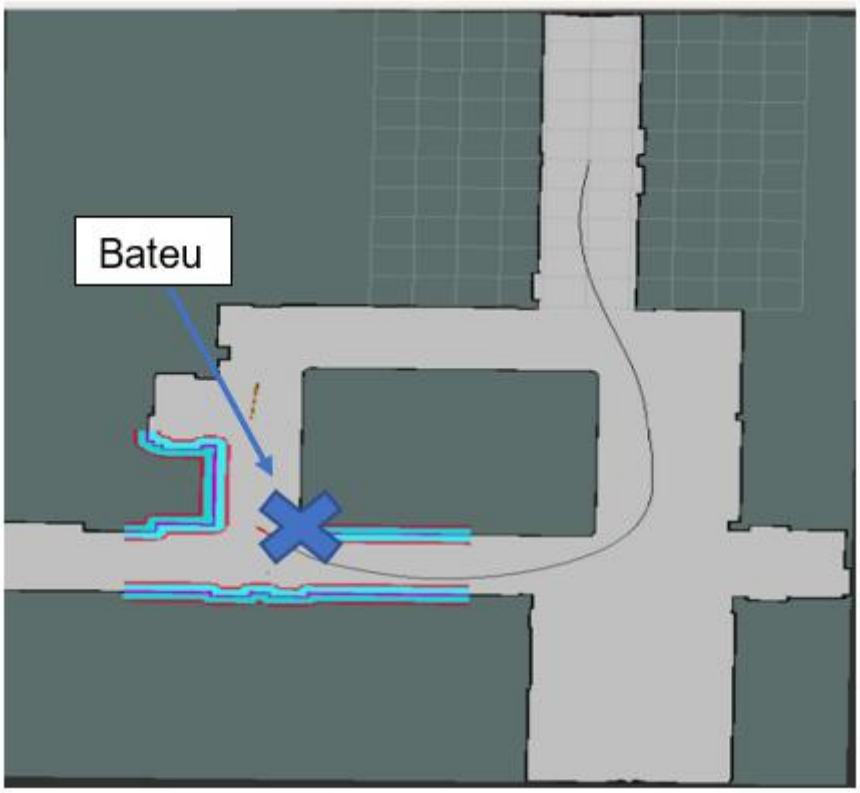
Figura 8 - Mapa para navegação do robô.

Fonte: Autores.

3.2 Localização usando apenas odometria do robô

A abordagem de localização de odometria usa apenas os *encoders* e a variação da pose do robô ao longo do tempo para calcular sua posição atual. Embora muito prática, a odometria é sensível a erros, já que desvios e deslizamentos às vezes não são detectados, e o robô se move sem que os *encoders* percebam o movimento. Portanto, durante esse tempo, os erros vão se acumulando, fazendo com que a posição real do robô seja consideravelmente diferente daquela conhecida pelo controlador. A Figura 9 mostra a abordagem usando odometria, onde o robô aparentemente fará curva, mas na verdade está colidindo com a pilastra. O uso de um método de localização não confiável pode levar a colisões. O vídeo dessa experiência é encontrado no link: https://youtu.be/T2LpuR6EFcY (Oliveira, 2021b).

Figura 9 - Trajetória do robô usando Odometria.

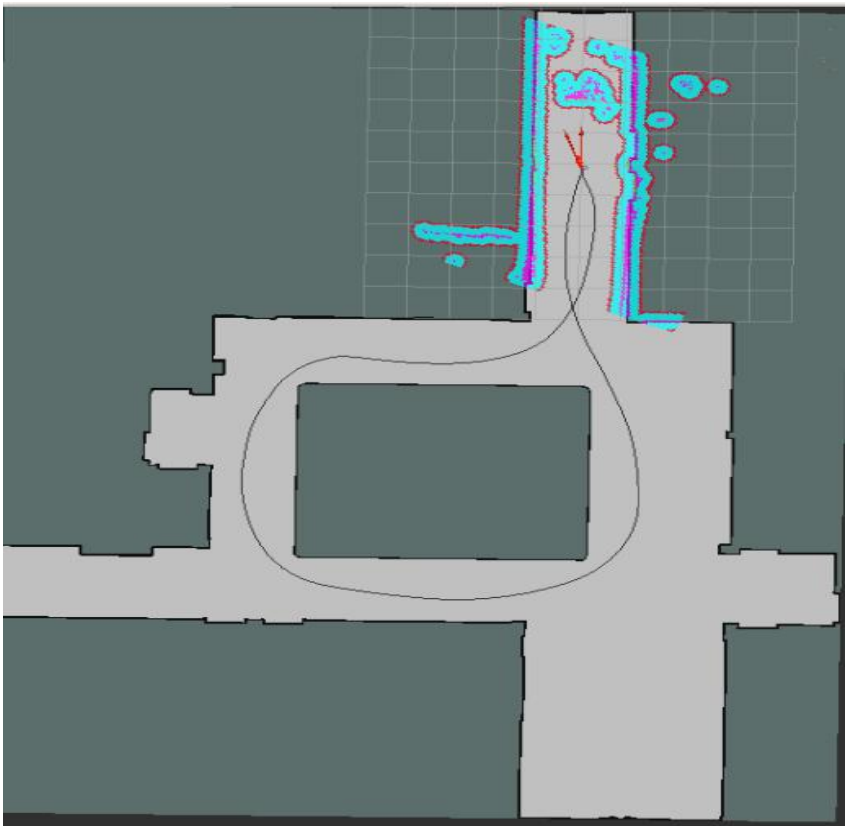


Fonte: Autores.

3.3 Localização usando odometria e AMCL

Adicionar o algoritmo AMCL ao sinal de odometria permite ao robô recalcular sua posição e compensar os erros devido às medições erradas do controlador. A navegação com o AMCL é mais precisa, pois o robô pode chegar ao destino sem muitos erros em comparação com a primeira abordagem. A Figura 10 mostra o caminho para odometria com localização AMCL chegando ao destino. O vídeo dessa experiência é encontrado no link: https://youtu.be/MRgpewObNQI (Oliveira, 2021a).

Figura 10 - Trajetória do robô usando odometria e o AMCL.



Fonte: Autores.

4. Conclusão

No trabalho desenvolvido foi realizada a navegação do robô diferencial e a localização com o AMCL. Os resultados mostraram que a trajetória apresentou um bom desempenho.

A navegação incluindo o AMCL permitiu que o robô tivesse uma posição mais precisa de sua postura real, compensando os erros gerados pelos dados da odometria e usando mais sensores para inferir a postura do robô.

As experiências mostraram que o uso apenas da odometria como método de localização fazia com que o robô acumulasse mais erros, uma vez que não havia correções. A odometria com AMCL, por outro lado, compensou esses erros, dando uma localização mais precisa. O método de navegação apenas com a odometria acabou fazendo com que o robô não conseguisse completar o percurso definido. Finalmente, o método com AMCL teve um desempenho melhor, apresentando menor erro. Outra conclusão importante é que o AMCL é necessário para que o robô encontre sua posição real no mapa. O experimento utilizando apenas a odometria não apresentou um bom desempenho, o que mostra que o AMCL, nesta configuração, desempenha um papel importante para corrigir a posição do robô e minimizar o erro entre a posição calculada e a real.

Como trabalhos futuros, se pretende realizar a fusão dos dados de vários métodos de localização obtidos com o uso de pacotes do ROS. Desta maneira, será possível estimar a posição e orientação do robô através da odometria das rodas e sensores inerciais, IMU (*Inertial Measurement Unit*), com acelerômetros e giroscópios. Após a coleta dos dados advindos desses sensores, pode-se utilizar um algoritmo de fusão de sensores para reduzir o ruído e melhorar o valor de localização estimado pela odometria.

Agradecimentos

Os autores gostariam de agradecer ao apoio técnico e financeiro do Grupo de Pesquisa da Automação Industrial (GAIn) e do Instituto Federal de Educação, Ciência e Tecnologia do Espírito Santo (IFES).

Referências

Ayres, L. M., Batista, L. G., da Silva, J. R., Motta, V. da R., Marques, V. M., & Cuadros, M. A. S. L. (2017, October 4). Desenvolvimento e implementação de uma arquitetura de navegação para um robô móvel utilizando comandos de voz, algoritmo A* e o controlador backstepping. XIII Simpósio Brasileiro de Automação Inteligente - SBAI.

Ben-Ari, M., & Mondada, F. (2018). Elements of Robotics. In Elements of Robotics (1st ed.). Springer International Publishing. https://doi.org/10.1007/978-3-319-62533-1

Bezerra, C. G. (2004). Localização de um robô móvel usando odometria e marcos naturais. Universidade Federal do Rio Grande do Norte. https://repositorio.ufrn.br/jspui/handle/123456789/15411

Chen, S.-M., HSU, Y.-S., & PEARN, W. L. (2003). Capability measures for m -dependent stationary processes. Statistics: A Journal of Theoretical and Applied Statistics, 37(1), 1–24. https://doi.org/10.1080/02331880309257

Chong, K. S., & Kleeman, L. (1997). Accurate odometry and error modelling for a mobile robot. Proceedings of International Conference on Robotics and Automation, 4, 2783–2788. https://doi.org/10.1109/ROBOT.1997.606708

da Silva, J. R., de Freitas, B. P. S., Medeiros, M. G., & Cuadros, M. A. D. S. L. (2016, September 27). desenvolvimento de um sistema de localização indoor utilizando a intensidade do sinal de rádio frequência de módulos bluetooth ®. XLIV Congresso Brasileiro de Educação Em Engenharia.

Dellaert, F., Fox, D., Burgard, W., & Thrun, S. (1999). Monte Carlo localization for mobile robots. In IEEE (Ed.), Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C) (Vol. 2, Issues 1–2, pp. 1322–1328). IEEE. https://doi.org/10.1109/ROBOT.1999.772544

Eppstein, E. M. (2020, September 3). Move_base. ROS. http://wiki.ros.org/move_base

Estefo, P., Simmonds, J., Robbes, R., & Fabry, J. (2019). The Robot Operating System: Package reuse and community dynamics. Journal of Systems and Software. https://doi.org/10.1016/j.jss.2019.02.024

Fairchild, C., & Harman, T. L. (2017). ROS robotics by example: learning to control wheeled, limbed, and flying robots using ROS Kinetic Kame (2nd ed., Vol. 1). Packt Publishing.

Faisal, M., Hedjar, R., al Sulaiman, M., & Al-Mutib, K. (2013). Fuzzy logic navigation and obstacle avoidance by a mobile robot in an unknown dynamic environment. International Journal of Advanced Robotic Systems, 10(October). https://doi.org/10.5772/54427

Faria, G., & Romero, R. A. F. (2002). Navegação de robôs móveis utilizando aprendizado por reforço e lógica fuzzi. In Sba: Controle & Automação Sociedade Brasileira de Automática (Vol. 13, Issue 3, pp. 219–230). https://doi.org/10.1590/S0103-17592002000300002

Fortes, L. L. S. (2018). Implementação de um filtro de partículas para localização de um robô. Instituto Federal do Espírito Santo.

Fox, D. (2003). Adapting the sample size in particle filters through KLD-sampling. International Journal of Robotics Research, 22(12), 985–1003. https://doi.org/10.1177/0278364903022012001

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. IEEE Robotics and Automation Magazine, 4(1), 23–33. https://doi.org/10.1109/100.580977

Fox, D., Burgard, W., & Thrun, S. (1999). Markov localization for mobile robots in dynamic environments. Journal of Artificial Intelligence Research, 11, 391–427. https://doi.org/10.1613/jair.616

Gamarra, D. F. T., Legg, A. P., de Souza Leite Cuadros, M. A., & da Silva, E. S. (2019). Sensory integration of a mobile robot using the embedded system odroid-XU4 and ROS. Proceedings - 2019 Latin American Robotics Symposium, 2019 Brazilian Symposium on Robotics and 2019 Workshop on Robotics in Education, LARS/SBR/WRE 2019, 198–203. https://doi.org/10.1109/LARS-SBR-WRE48964.2019.00042

Gerkey, B. P. (2020, August 27). AMCL. ROS. http://wiki.ros.org/amcl

Gouveia, M. C. M. (2008). Estudo e implementação de um algoritmo de localização baseado na correspondência de mapas. Universidade do Porto.

Joseph, L. (2017). ROS robotics projects (1st ed., Vol. 1). Packt. https://www.packtpub.com/product/ros-robotics-projects/9781783554713

Meeussen, W. (2010, October 27). Coordinate frames for mobile platforms. ROS. https://www.ros.org/reps/rep-0105.html

Nemec, D., Šimák, V., Janota, A., Hruboš, M., & Bubeníková, E. (2019). Precise localization of the mobile wheeled robot using sensor fusion of odometry, visual artificial landmarks and inertial sensors. Robotics and Autonomous Systems. https://doi.org/10.1016/j.robot.2018.11.019

Oliveira, R. do A. (2021a, April 16). AMCL com Robô 1 (pp. 1–1). GAIn. https://youtu.be/MRgpewObNQI

Oliveira, R. do A. (2021b, April 16). Odometria do Robô 1 (pp. 1–1). GAIn. https://www.youtube.com/watch?v=T2LpuR6EFcY

Paiva, B., de Freitas, S., Medeiros, M. G., Ruella Da Silva, J., Maia De Almeida, G., Antonio De Souza, M., & Cuadros, L. (2016). Utilização de exemplos criados no software labview ® implementados no starter kit 2.0 como ferramenta no ensino-aprendizagem da robótica. XLIV Congresso Brasileiro de Educação Em Engenharia (COBENGE), 1.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Ng, A. Y. (2009). ROS: an open-source Robot Operating System. ICRA Workshop on Open Source Software, 6.

Romero, R. A. F., Silva Junior, E. P. e, Osório, F. S., & Wolf, D. F. (2014). Robótica móvel (1st ed., Vol. 1). LTC.

ROS. (2015, September 11). AMCL parameters. ROS. https://answers.ros.org/question/217462/amcl-parameters-spreading-out-is-good-or-bad/

ROS. (2019). Documentation. ROS. http://wiki.ros.org/

Salarolli, P. F., Da, V., Motta, R., De, M. A., & Cuadros, S. L. (2017). Fusão dos dados do Dead Reckoning e do giroscópio usando o filtro de Kalman estendido aplicado à localização de uma cadeira de rodas motorizada. XIII Simpósio Brasileiro de Automação Inteligente - SBAI, 1571–1576.

Tellez, R. (2019). A history of ROS. The Construct. https://www.theconstructsim.com/history-ros/

Thrun, S., Burgard, W., & Fox, D. (2005). Probabilistic robotics (1st ed., Vol. 1). The MIT Press.

Tommasi, E., Faria, H., Cuadros, M., Almeida, G., Resende, C., & Gamarra, D. (2015). Estudo Comparativo de Controladores de Seguimento de Trajetória para Robôs de Tração Diferencial: Fuzzy, Ganhos Fixos e Backstepping. XII Simpósio Brasileiro de Automação Inteligente (SBAI), 1–6.

Zaman, S., Slany, W., & Steinbauer, G. (2011). ROS-based mapping, localization and autonomous navigation using a Pioneer 3-DX robot and their relevant issues. 2011 Saudi International Electronics, Communications and Photonics Conference (SIECPC), 1–5. https://doi.org/10.1109/SIECPC.2011.5876943